

Scilab manual

Scilab manual

Table of Contents

I. Scilab	1
abort	2
add_demo	3
ans	4
argn	5
backslash (\)	6
banner	7
boolean	8
brackets	9
break	10
case	11
clear	12
clearfun	13
clearglobal	14
colon	15
comma	16
comments	17
comp	18
comparison	19
continue	21
debug	22
delbpt	23
dispbpt	24
do	25
dot	26
edit	28
else	29
elseif	30
empty	31
end	32
equal	33
errcatch	34
errclear	35
error	36
error_table	37
evstr	45
exists	46
exit	47
external	48
extraction	49
for	52
format	53
funcprot	55
funptr	56
getdebuginfo	57
getmd5	58
getmemory	59
getmodules	60
getos	61
getscilabmode	62
getshell	63
getvariablesystack	64
getversion	65
global	66
gstacksize	67

hat	68
ieee	69
if then else	70
insertion	71
intppty	75
inv_coeff	76
iserror	77
isglobal	78
lasterror	79
left	80
less	81
macr2lst	82
macr2tree	84
matrices	85
matrix	86
mode	87
mtlb_mode	89
names	90
newfun	91
null	92
parents	93
pause	95
percent	96
perl	97
plus	98
poly	99
power	100
predef	101
quit	102
quote	103
rational	104
readgateway	105
resume	106
return	107
sciargs	108
scilab	109
select	113
semicolon (;)	114
setbpt	115
sethomedirectory	116
slash	117
stacksize	118
star	119
startup	120
symbols	121
testmatrix	122
then	123
tilda	124
try	125
type	127
typename	128
user	129
varn	130
ver	131
warning	132
what	133
where	134
whereami	135

while	136
who	137
who_user	138
whos	139
with_atlas	140
with_gtk	141
with_javasci	142
with_macros_source	143
with_module	144
with_pvm	145
with_texmacs	146
with_tk	147
II. Differential Equations, Integration	148
bode	149
dae	163
daeoptions	168
dasrt	170
dassl	174
feval	179
impl	180
int2d	182
int3d	184
intc	187
integrate	188
intg	190
intl	192
ode	193
ode_discrete	198
ode_optional_output	199
ode_root	201
odedc	203
odeoptions	206
III. Elementary Functions	207
abs	208
acos	209
acosd	210
acosh	211
acoshm	212
acosm	213
acot	214
acotd	215
acoth	216
acsc	217
acscd	218
acsch	219
adj2sp	220
amell	221
and	222
&	223
asec	224
asecd	225
asech	226
asin	227
asind	228
asinh	229
asinhm	230
asinm	231
atan	232

atand	234
atanh	235
atanhm	236
atanm	237
base2dec	238
bin2dec	239
binomial	240
bitand	242
bitor	243
bloc2exp	244
bloc2ss	247
cat	250
ceil	251
cell2mat	252
cellstr	253
char	254
conj	256
cos	257
cosd	258
cosh	259
coshm	260
cosm	261
cotd	262
cotg	263
coth	264
cothm	265
csc	266
cscd	267
csch	268
csgn	269
cumprod	270
cumsum	271
dec2bin	272
dec2hex	273
dec2oct	274
delip	275
diag	276
diff	277
double	278
dsearch	279
eval	281
exp	282
eye	283
factor	284
fix	285
flipdim	286
floor	287
frexp	288
gsort	289
hex2dec	291
imag	292
imult	293
ind2sub	294
int	295
intersect	296
intrtrap	298
isdef	299
isempty	300

isequal	301
isequalbitwise	302
isinf	303
isnan	304
isreal	305
isvector	306
kron	307
lex_sort	308
linspace	309
log	310
log10	311
log1p	312
log2	313
logm	314
logspace	315
lstsize	316
max	317
maxi	318
meshgrid	319
min	320
mini	321
minus	322
modulo	323
ndgrid	324
ndims	326
nearfloat	327
nextpow2	328
norm	329
not	330
number_properties	331
oct2dec	333
ones	334
or	335
.....	336
pen2ea	337
perms	338
permute	339
pertrans	340
primes	341
prod	342
rand	343
rat	345
real	346
resize_matrix	347
round	348
sec	349
secd	350
sech	351
setdiff	352
sign	353
signm	354
sin	355
sinc	356
sind	357
sinh	358
sinhm	359
sinm	360
size	361

solve	363
sort	364
sp2adj	366
speye	367
spones	368
sprand	369
spzeros	370
sqrt	371
sqrtn	372
squarewave	373
ssrand	374
sub2ind	376
sum	377
sysconv	378
sysdiag	380
syslin	381
tan	383
tand	384
tanh	385
tanhm	386
tanm	387
toeplitz	388
trfmod	389
trianfml	390
tril	391
trisolve	392
triu	393
typeof	394
union	396
unique	397
vectorfind	399
zeros	400
IV. Functions	401
add_profiling	402
bytecode	403
bytecodewalk	404
deff	405
exec	406
execstr	408
fun2string	410
function	411
functions	413
genlib	415
get_function_path	416
getd	417
getf	418
head_comments	420
lib	421
librarieslist	423
library	424
libraryinfo	425
listfunctions	426
macro	427
macrovar	428
plotprofile	429
profile	430
recompilefunction	431
remove_profiling	432

reset_profiling	433
showprofile	434
varargin	435
varargout	436
whereis	437
V. Files : Input/Output functions	438
basename	439
chdir	440
copyfile	441
createdir	442
deletefile	443
dir	444
dirname	445
dispfiles	446
fileext	447
fileinfo	448
fileparts	450
filesep	451
findfiles	452
fprintf	453
fprintfMat	454
fscanf	455
fscanfMat	456
fullfile	457
fullpath	458
get_absolute_file_path	459
getdrives	460
getlongpathname	461
getrelativefilename	462
getshortpathname	463
%io	464
isdir	465
isfile	466
listfiles	467
listvarinfile	468
ls	469
maxfiles	470
mclearerr	471
mclose	472
mdelete	473
meof	474
merror	475
mfprintf	476
mscanf	477
mget	480
mgetl	482
mgetstr	483
mkdir	484
mopen	485
movefile	488
mput	489
mputl	491
mputstr	492
mseek	493
mtell	494
newest	495
pathconvert	496
pathsep	497

pwd	498
removedir	499
rmdir	500
save_format	501
scanf	504
scanf_conversion	505
sscanf	507
VI. Graphics Library	508
GlobalProperty	509
Graphics	514
LineSpec	558
Matplot	560
Matplot1	563
Matplot_properties	565
Sfgrayplot	566
Sgrayplot	568
addcolor	571
alufunctions	572
arc_properties	573
autumncolormap	575
axes_properties	576
axis_properties	585
bar	588
barh	590
barhomogenize	592
bonecolormap	594
captions	595
champ	597
champ1	599
champ_properties	600
clear_pixmap	602
clf	603
color	605
color_list	606
colorbar	626
colordef	628
colormap	629
Compound_properties	630
contour	631
contour2d	634
contour2di	636
contourf	638
coolcolormap	640
coppercolormap	641
copy	642
delete	643
dragrect	644
draw	645
drawaxis	646
drawlater	648
drawnow	650
edit_curv	651
errbar	652
eval3d	653
eval3dp	654
event handler functions	655
fac3d	657
fchamp	658

fcontour	660
fcontour2d	661
fec	662
fec_properties	665
fgrayplot	667
figure_properties	668
fplot2d	671
fplot3d	672
fplot3d1	673
gca	674
gce	675
gcf	676
gda	677
gdf	678
ged	679
genfac3d	680
geom3d	681
get	682
get_figure_handle	684
getcolor	685
getfont	686
getlinestyle	687
getmark	688
getsymbol	689
glue	690
graduate	691
graphics_entities	692
graphics_fonts	695
graycolormap	697
grayplot	698
grayplot_properties	699
graypolarplot	701
havewindow	703
hist3d	704
histplot	706
hotcolormap	708
hsv2rgb	709
hsvcolormap	711
is_handle_valid	712
isoview	713
jetcolormap	714
label_properties	715
legend	717
legend_properties	719
legends	722
locate	724
Math rendering in Scilab graphics	725
mesh	727
milk_drop	729
move	730
name2rgb	731
newaxes	732
nf3d	733
object_editor	734
oceancolormap	739
oldplot	740
param3d	741
param3d1	743

paramfplot2d	746
pie	747
pinkcolormap	748
plot	749
plot2d	754
plot2d1	759
plot2d2	761
plot2d3	762
plot2d4	763
plot2d_old_version	764
plot3d	768
plot3d1	777
plot3d2	780
plot3d3	783
plotframe	786
plzr	788
polarplot	789
polyline_properties	791
rainbowcolormap	795
rectangle_properties	796
relocate_handle	799
replot	800
rgb2name	801
rotate	802
rotate_axes	803
rubberbox	804
sca	805
scaling	806
scf	807
sd2sci	808
sda	809
sdf	810
secto3d	811
segs_properties	812
set	815
seteventhandler	817
show_pixmap	818
show_window	819
springcolormap	820
square	821
stringbox	822
subplot	824
summercolormap	825
surf	826
surface_properties	830
swap_handles	834
text_properties	836
title	839
titlepage	841
twinkle	842
unglue	843
unzoom	844
whitecolormap	845
winsid	846
wintercolormap	847
xarc	848
xarcs	849
xarrows	850

xbasc	851
xbasr	852
xchange	853
xclear	854
xclick	855
xdel	857
xfarc	858
xfarcs	859
xfpoly	860
xfpolys	861
xfrect	863
xget	864
xgettech	866
xgetmouse	868
xgraduate	870
xgrid	871
xinfo	872
xlfont	873
xload	875
xname	876
xnumb	877
xpause	878
xpoly	879
xpolys	880
xrect	881
xrects	882
xrpoly	883
xsave	884
xsegs	885
xselect	887
xset	888
xsetech	891
xsetm	893
xstring	894
xstringb	896
xstringl	897
xtitle	899
zoom_rect	900
VII. Graphics : exporting and printing	902
driver	903
xend	905
xinit	906
xs2bmp	907
xs2emf	908
xs2eps	909
xs2fig	910
xs2gif	911
xs2jpg	912
xs2pdf	913
xs2png	914
xs2ppm	915
xs2ps	916
xs2svg	917
VIII. Boolean	918
bool2s	919
find	920
IX. CACSD	922
abcd	923

abinv	924
arhmk	928
arl2	929
arma	931
arma2p	933
armac	934
armax	935
armax1	937
arsimul	939
augment	940
balreal	942
bilin	943
black	944
bode	946
bstap	948
cainv	949
calfrq	951
canon	952
ccontrg	954
chart	955
cls2dls	957
colinout	958
colregul	959
cont_frm	960
cont_mat	961
contr	962
contrss	964
copfac	965
csim	966
ctr_gram	968
dbphi	969
dcf	970
ddp	971
des2ss	973
des2tf	974
dhinf	975
dhnorm	978
dscr	979
dsimul	980
dt_ility	981
dtsi	982
equil	983
equil1	984
evans	985
feedback	986
findABCD	987
findAC	990
findBD	992
findBDK	996
findR	999
findx0BD	1002
flts	1005
fourplan	1008
frep2tf	1009
freq	1011
freson	1012
fspecg	1013
fstabst	1014

g_margin	1016
gainplot	1017
gamitg	1018
gcare	1019
gfare	1020
gfrancis	1021
gtild	1023
h2norm	1025
h_cl	1026
h_inf	1027
h_inf_st	1028
h_norm	1029
hankelsv	1030
hinf	1031
imrep2ss	1034
inistate	1035
invsyslin	1037
kpure	1038
krac2	1040
lcf	1041
leqr	1042
lft	1044
lin	1046
linf	1048
linfn	1049
linmeq	1050
lqe	1054
lqg	1056
lqg2stan	1057
lqg_ltr	1059
lqr	1060
ltitr	1062
m_circle	1063
macglov	1064
markp2ss	1065
minreal	1066
minss	1067
mucomp	1068
narsimul	1069
nehari	1070
noisegen	1071
nyquist	1072
obs_gram	1074
obscont	1075
observer	1077
obsv_mat	1079
obsvss	1080
p_margin	1081
parrot	1082
pfss	1083
phasemag	1084
ppol	1085
prbs_a	1086
projsl	1087
reglin	1088
repfreq	1089
ric_desc	1091
ricc	1093

riccati	1095
routh_t	1096
rowinout	1098
rowregul	1099
rtitr	1100
sensi	1103
sgrid	1104
show_margins	1105
sident	1106
sm2des	1110
sm2ss	1111
sorder	1112
specfact	1116
ss2des	1117
ss2ss	1118
ss2tf	1120
st_ility	1121
stabil	1123
svplot	1124
sysfact	1126
syssize	1127
tf2des	1128
tf2ss	1129
time_id	1130
trzeros	1132
ui_observer	1134
unobs	1136
zeropen	1137
zgrid	1138
X. Data Structures	1139
cell	1140
definedfields	1142
fieldnames	1143
getfield	1144
hypermat	1145
hypermatrices	1146
iscell	1147
iscellstr	1148
isfield	1149
isstruct	1150
list	1151
lsslist	1153
lstcat	1154
mlist	1155
rlist	1157
setfield	1158
struct	1159
tlist	1160
XI. Shell	1162
clc	1163
lines	1164
prompt	1165
tohome	1166
XII. Console	1167
console	1168
XIII. Completion	1169
completion	1170
XIV. History manager	1172

addhistory	1173
displayhistory	1174
gethistory	1175
gethistoryfile	1176
historymanager	1177
historysize	1178
loadhistory	1179
removelinehistory	1180
resethistory	1181
saveafterncommands	1182
saveconsecutivecommands	1183
savehistory	1184
sethistoryfile	1185
XV. GUI	1186
1. Tree	1188
uiConcatTree	1189
uiCreateNode	1190
uiCreateTree	1191
uiDeleteNode	1192
uiDisplayTree	1194
uiDumpTree	1195
uiEqualsTree	1196
uiFindNode	1197
uiGetChildrenNode	1199
uiGetNodePosition	1200
uiGetParentNode	1201
uiInsertNode	1202
about	1204
addmenu	1205
clipboard	1207
close	1208
delmenu	1209
exportUI	1210
figure	1211
findobj	1213
gcbo	1214
getcallbackobject	1215
getinstalledlookandfeels	1216
getlookandfeel	1217
getvalue	1218
messagebox	1220
printfigure	1222
printsetupbox	1223
progressionbar	1224
root_properties	1225
setlookandfeel	1226
setmenu	1228
toolbar	1229
toprint	1230
uicontrol	1232
uigetcolor	1237
uigetdir	1239
uigetfile	1240
uigetfont	1242
uimenu	1243
uiputfile	1245
unsetmenu	1246
usecanvas	1247

waitbar	1250
x_choices	1251
x_choose	1252
x_choose_modeless	1253
x_dialog	1254
x_matrix	1255
x_mdialog	1256
x_message_modeless	1257
XVI. Dynamic/incremental Link	1258
G_make	1259
VCtoLCCLib	1260
addinter	1261
c_link	1264
call	1265
chooselcccompiler	1268
configure_lcc	1269
configure_ifort	1270
configure_msvc	1271
dllinfo	1272
findlcccompiler	1273
findmsifortcompiler	1274
findmsvccompiler	1275
fort	1276
getdynlibext	1278
haveacompiler	1279
ilib_build	1280
ilib_compile	1288
ilib_for_link	1289
ilib_gen_Make	1291
ilib_gen_cleaner	1293
ilib_gen_gateway	1294
ilib_gen_loader	1296
ilib_mex_build	1297
ilib_verbose	1299
link	1301
ulink	1303
with_lcc	1304
XVII. Integers	1305
iconvert	1306
int8	1307
inttype	1308
XVIII. Interpolation	1309
bsplin3val	1310
cshep2d	1312
eval_cshep2d	1314
interp	1316
interp1	1319
interp2d	1321
interp3d	1323
interp1n	1325
intsplin	1326
linear_interpn	1327
lsq_splin	1330
smooth	1332
splin	1333
splin2d	1336
splin3d	1339
XIX. Input/Output functions	1341

file	1342
getenv	1345
getio	1346
getpid	1347
getscilabkeywords	1348
halt	1349
host	1350
input	1351
load	1352
oldload	1353
oldsave	1354
read	1355
read4b	1357
readb	1358
readc_	1359
save	1360
setenv	1362
unix	1363
unix_g	1364
unix_s	1365
unix_w	1366
unix_x	1367
writb	1368
write	1369
write4b	1370
XX. Output functions	1371
diary	1372
disp	1375
mprintf	1376
msprintf	1377
prettyprint	1378
print	1380
printf	1381
printf_conversion	1382
sprintf	1385
ssprint	1386
XXI. Intersci	1387
intersci	1388
XXII. JVM	1389
javaclasspath	1390
javalibrarypath	1391
jre_path	1392
system_getproperty	1393
system_setproperty	1396
with_embedded_jre	1397
XXIII. Linear Algebra	1398
aff2ab	1399
balanc	1401
bdiag	1402
chfact	1403
chol	1404
chsolve	1405
classmarkov	1406
cmb_lin	1407
coff	1408
colcomp	1409
companion	1410
cond	1411

det	1412
eigenmarkov	1413
ereduc	1414
expm	1415
fstair	1416
fullrf	1418
fullrfk	1419
genmarkov	1420
givens	1421
glever	1422
gschur	1423
gspec	1424
hess	1425
householder	1426
im_inv	1427
inv	1428
kernel	1429
kroneck	1430
linsolve	1432
lsq	1433
lu	1435
lyap	1437
nlev	1438
orth	1439
pbig	1440
pencan	1441
penlaur	1442
pinv	1443
polar	1444
proj	1445
projspec	1446
psmall	1447
qr	1448
quaskro	1450
randpencil	1452
range	1453
rank	1454
rankqr	1455
rcond	1457
rowcomp	1458
rowshuff	1460
rref	1461
schur	1462
spaninter	1466
spanplus	1467
spantwo	1468
spec	1470
sqroot	1473
squeeze	1474
sva	1475
svd	1476
sylv	1477
trace	1478
XXIV. Localization	1479
dgettext	1480
getdefaultlanguage	1481
getlanguage	1482
gettext	1483

LANGUAGE	1484
setdefaultlanguage	1485
setlanguage	1486
XXV. Optimization and Simulation	1487
2. Neldermead	1489
fminsearch	1490
neldermead	1499
overview	1520
nmplot	1524
optimget	1535
optimplotfunccount	1537
optimplotfval	1538
optimplotx	1539
optimset	1540
3. Optimization base	1544
optimbase	1545
4. Optimization simplex	1559
optimsimplex	1560
NDcost	1576
aplat	1579
datafit	1580
derivative	1583
fit_dat	1587
fsolve	1589
karmarkar	1592
leastsq	1593
list2vec	1599
lmisolver	1600
lmitool	1602
lsqrsolve	1603
numdiff	1606
optim	1608
qld	1622
qp_solve	1625
qpsolve	1628
quapro	1631
readmps	1632
recons	1636
semidef	1637
vec2list	1640
XXVI. Overloading	1641
overloading	1642
XXVII. Polynomials	1645
bezout	1646
clean	1647
cmndred	1648
coeff	1649
coffg	1650
colcompr	1651
degree	1652
denom	1653
derivat	1654
determ	1655
detr	1656
diophant	1657
factors	1658
gcd	1660
hermit	1661

horner	1662
hrmt	1663
htrianr	1664
invr	1665
lcm	1666
lcmdiag	1667
ldiv	1668
numer	1669
pdiv	1670
pol2des	1671
pol2str	1672
polfact	1673
residu	1674
roots	1675
rowcompr	1676
sfact	1677
simp	1678
simp_mode	1679
sylm	1680
systmat	1681
XXVIII. Signal Processing	1682
5. How to	1684
How to design an elliptic filter	1685
Signal	1693
analpf	1697
bilt	1699
buttmag	1701
casc	1702
cepstrum	1703
cheb1mag	1704
cheb2mag	1705
chepol	1706
convol	1707
corr	1709
cspect	1712
czt	1715
detrend	1717
dft	1719
ell1mag	1720
eqfir	1721
eqiir	1722
faurre	1723
ffilt	1724
fft	1725
fft2	1727
fftshift	1729
filt_sinc	1731
filter	1732
find_freq	1733
findm	1734
frfit	1735
frmag	1736
fsfirlin	1737
group	1739
hank	1740
hilb	1741
hilbert	1742
iir	1743

iirgroup	1744
iirlp	1745
intdec	1746
jmat	1747
kalm	1748
lattn	1749
lattp	1750
lev	1751
levin	1752
lindquist	1755
mese	1756
mfft	1757
mrfit	1758
%asn	1759
%k	1760
%sn	1761
phc	1762
pspect	1764
remez	1767
remezb	1769
rpem	1771
sincd	1773
srfaur	1774
srkf	1776
sskf	1777
syredi	1778
system	1780
trans	1781
wfir	1783
wiener	1784
wigner	1785
window	1786
yulewalk	1788
zpbutt	1789
zpch1	1790
zpch2	1791
zpell	1792
XXIX. Sparses Matrix	1793
full	1794
gmres	1795
ludel	1797
lufact	1798
luget	1800
lusolve	1801
mtlb_sparse	1802
nnz	1803
pcg	1804
qmr	1808
sparse	1810
spchol	1811
spcompact	1812
spget	1814
XXX. Special Functions	1815
besseli	1816
beta	1819
calerf	1821
dlgamma	1822
erf	1823

erfc	1824
erfcx	1825
erfinv	1826
gamma	1827
gammaln	1828
legendre	1829
oldbesseli	1832
XXXI. Strings	1835
ascii	1836
blanks	1837
code2str	1838
convstr	1839
emptystr	1840
grep	1841
isalphanum	1843
isascii	1844
isdigit	1845
isletter	1846
isnum	1847
justify	1848
length	1849
part	1850
regex	1851
sci2exp	1852
str2code	1853
strcat	1854
strchr	1855
strcmp	1856
strcmpi	1857
strcspn	1858
strindex	1859
string	1861
strings	1862
stripblanks	1863
strncpy	1864
strchr	1865
strev	1866
strsplit	1867
strspn	1869
strstr	1870
strsubst	1871
strtod	1872
strtok	1873
tokenpos	1874
tokens	1875
tree2code	1876
XXXII. Symbolic	1877
addf	1878
ldivf	1879
mulf	1880
rdivf	1881
subf	1882
XXXIII. Time and Date	1883
calendar	1884
clock	1885
date	1886
datetime	1887
datevec	1889

eomday	1890
etime	1891
getdate	1892
now	1894
realtimeinit	1895
sleep	1896
tic	1897
timer	1898
toc	1899
weekday	1900
XXXIV. Statistics	1901
cdfbet	1902
cdfbin	1903
cdfchi	1904
cdfchn	1905
cdff	1906
cdffnc	1907
cdfgam	1908
cdfnbn	1909
cdfnor	1910
cdfpoi	1911
cdfn	1912
center	1913
wcenter	1914
cmoment	1915
correl	1916
covar	1917
ftest	1918
ftuneq	1919
geomean	1920
harmean	1921
iqr	1922
labostat	1923
mad	1924
mean	1925
meanf	1926
median	1927
moment	1928
msd	1929
mvvacov	1930
nancumsum	1931
nand2mean	1932
nanmax	1933
nanmean	1934
nanmeanf	1935
nanmedian	1936
nanmin	1937
nanstdev	1938
nansum	1939
nfreq	1940
pca	1941
perctl	1942
princomp	1943
quart	1945
regress	1946
sample	1947
samplef	1948
samwr	1950

show_pca	1951
st_deviation	1952
stdevf	1953
strange	1954
tabul	1955
thrownan	1957
trimmean	1958
variance	1960
variancef	1961
XXXV. ARnoldi PACKage	1962
dnaupd	1963
dneupd	1971
dsaupd	1978
dsaupd	1986
znaupd	1991
zneupd	1999
XXXVI. Compatibility Functions	2005
asciimat	2006
firstnonsingleton	2007
makecell	2008
mstr2sci	2009
mtlb_0	2010
mtlb_a	2011
mtlb_all	2012
mtlb_any	2013
mtlb_axis	2014
mtlb_beta	2015
mtlb_box	2016
mtlb_close	2017
mtlb_colordef	2018
mtlb_conv	2019
mtlb_cumprod	2020
mtlb_cumsum	2021
mtlb_dec2hex	2022
mtlb_delete	2023
mtlb_diag	2024
mtlb_diff	2025
mtlb_dir	2026
mtlb_double	2027
mtlb_e	2028
mtlb_echo	2029
mtlb_eval	2030
mtlb_exist	2031
mtlb_eye	2032
mtlb_false	2033
mtlb_fft	2034
mtlb_fftshift	2035
mtlb_find	2036
mtlb_findstr	2037
mtlb_fliplr	2038
mtlb_fopen	2039
mtlb_format	2040
mtlb_fprintf	2041
mtlb_fread	2042
mtlb_fscanf	2043
mtlb_full	2044
mtlb_fwrite	2045
mtlb_grid	2046

mtlb_hold	2047
mtlb_i	2048
mtlb_ifft	2049
mtlb_imp	2050
mtlb_int16	2051
mtlb_int32	2052
mtlb_int8	2053
mtlb_is	2054
mtlb_isa	2055
mtlb_isfield	2056
mtlb_isletter	2057
mtlb_isspace	2058
mtlb_l	2059
mtlb_legendre	2060
mtlb_linspace	2061
mtlb_logic	2062
mtlb_logical	2063
mtlb_lower	2064
mtlb_max	2065
mtlb_min	2066
mtlb_more	2067
mtlb_num2str	2068
mtlb_ones	2069
mtlb_plot	2070
mtlb_prod	2071
mtlb_rand	2072
mtlb_randn	2073
mtlb_rcond	2074
mtlb_realmax	2075
mtlb_realmin	2076
mtlb_repmat	2077
mtlb_s	2078
mtlb_setstr	2079
mtlb_size	2080
mtlb_sort	2081
mtlb_strcmp	2082
mtlb_strcmpi	2083
mtlb_strfind	2084
mtlb_strep	2085
mtlb_sum	2086
mtlb_t	2087
mtlb_toeplitz	2088
mtlb_tril	2089
mtlb_triu	2090
mtlb_true	2091
mtlb_uint16	2092
mtlb_uint32	2093
mtlb_uint8	2094
mtlb_upper	2095
mtlb_var	2096
mtlb_zeros	2098
XXXVII. Java Interface	2099
javasci.SciBoolean	2100
javasci.SciBooleanArray	2102
javasci.SciComplex	2104
javasci.SciComplexArray	2106
javasci.SciDouble	2109
javasci.SciDoubleArray	2111

javasci.SciInteger	2113
javasci.SciIntegerArray	2114
javasci.SciString	2115
javasci.SciStringArray	2117
javasci.Scilab	2119
Compile and run with javasci	2121
javasci	2123
javasci FAQ	2124
XXXVIII. Maple Interface	2125
sci2map	2126
XXXIX. Matlab to Scilab Conversion Tips	2127
About M2SCI tools	2128
Contents	2130
Cste	2131
Equal	2132
Funcall	2133
Infer	2134
Matlab-Scilab_character_strings	2135
Operation	2136
Type	2137
Variable	2138
get_contents_infer	2139
m2scideclare	2140
matfile2sci	2142
mfile2sci	2143
sci_files	2146
translatepaths	2148
XL. Tcl/Tk Interface	2149
ScilabEval	2150
TCL_CreateSlave	2152
TCL_DeleteInterp	2153
TCL_EvalFile	2154
TCL_EvalStr	2156
TCL_ExistArray	2158
TCL_ExistInterp	2159
TCL_ExistVar	2160
TCL_GetVar	2161
TCL_GetVersion	2163
TCL_SetVar	2164
TCL_UnsetVar	2166
TCL_UpVar	2167
browsevar	2168
config	2169
editvar	2170
tk_getdir	2171
tk_savefile	2172
winclose	2173
winlist	2174
XLI. Texmacs	2175
pol2tex	2176
texprint	2177
XLII. Sound file handling	2178
analyze	2179
auread	2180
auwrite	2181
beep	2182
lin2mu	2183
loadwave	2184

mapsound	2185
mu2lin	2186
playsnd	2187
savewave	2188
sound	2189
soundsec	2190
wavread	2191
wavwrite	2192
XLIII. Randlib	2193
grand	2194
XLIV. Development tools	2200
bench_run	2201
tbx_build_cleaner	2203
tbx_build_gateway	2204
tbx_build_gateway_clean	2205
tbx_build_gateway_loader	2206
tbx_build_help	2207
tbx_build_help_loader	2208
tbx_build_loader	2209
tbx_build_macros	2210
tbx_build_src	2211
tbx_builder_gateway	2213
tbx_builder_gateway_lang	2214
tbx_builder_help	2215
tbx_builder_help_lang	2216
tbx_builder_macros	2217
tbx_builder_src	2218
tbx_builder_src_lang	2219
test_run	2220
XLV. Demo Tools	2224
demo_begin	2225
demo_choose	2226
demo_compiler	2227
demo_end	2228
demo_file_choice	2229
demo_function_choice	2230
demo_mdialog	2231
demo_message	2232
demo_run	2233
XLVI. Spreadsheet	2234
excel2sci	2235
read_csv	2236
readxls	2237
write_csv	2238
xls_open	2239
xls_read	2241
XLVII. call_scilab API	2243
Boolean management	2244
Complex management	2246
DisableInteractiveMode	2248
Double management	2249
GetLastJob	2251
ScilabHaveAGraph	2252
SendScilabJob	2253
SendScilabJobs	2254
StartScilab	2255
String management	2256
TerminateScilab	2258

call_scilab	2259
Compile and run with Call Scilab	2261
creadbmat (obsolete)	2264
creadchain (obsolete)	2266
creadcmat (obsolete)	2268
creadmat (obsolete)	2270
cwritebmat (obsolete)	2272
cwritechain (obsolete)	2274
cwritecmat (obsolete)	2276
cwritemat (obsolete)	2278
fromc	2280
fromjava	2281
XLVIII. FFTW	2282
fftw	2283
fftw_flags	2285
fftw_forget_wisdom	2287
get_fftw_wisdom	2288
set_fftw_wisdom	2289
XLIX. UMFPACK Interface	2290
PlotSparse	2291
ReadHBSparse	2292
cond2sp	2294
condestsp	2296
rafiter	2298
res_with_prec	2300
taucs_chdel	2301
taucs_chfact	2302
taucs_chget	2304
taucs_chinfo	2306
taucs_chsolve	2307
taucs_license	2308
umf_license	2309
umf_ludel	2310
umf_lufact	2311
umf_luget	2313
umf_luinfo	2315
umf_lusolve	2317
umfpack	2318
L. Genetic Algorithms	2320
coding_ga_binary	2321
coding_ga_identity	2322
crossover_ga_binary	2323
crossover_ga_default	2324
init_ga_default	2326
mutation_ga_binary	2327
mutation_ga_default	2328
optim_ga	2329
optim_moga	2331
optim_nsga	2333
optim_nsga2	2335
pareto_filter	2337
selection_ga_elitist	2338
selection_ga_random	2340
LI. Simulated Annealing	2342
compute_initial_temp	2343
neigh_func_csa	2344
neigh_func_default	2345
neigh_func_fsa	2346

neigh_func_vfsa	2347
optim_sa	2348
temp_law_csa	2350
temp_law_default	2352
temp_law_fsa	2353
temp_law_huang	2355
temp_law_vfsa	2357
LII. Parameters	2359
add_param	2360
get_param	2361
init_param	2362
is_param	2363
list_param	2364
remove_param	2365
set_param	2366
LIII. Atoms	2367
Getting started	2368
Functions Summary	2370
atomsAutoloadAdd	2371
atomsAutoloadDel	2373
atomsAutoloadList	2375
atomsDepTreeShow	2376
atomsGetInstalled	2377
atomsGetLoaded	2379
atomsInstall	2380
atomsIsInstalled	2383
atomsIsLoaded	2385
atomsList	2387
atomsLoad	2388
atomsRemove	2389
atomsRepositoryAdd	2391
atomsRepositoryDel	2392
atomsRepositoryList	2393
atomsSearch	2394
atomsSetConfig	2395
atomsShow	2397
atomsSystemUpdate	2398
atomsUpdate	2399
LIV. Matlab binary files I/O	2401
loadmatfile	2402
matfile_close	2403
matfile_listvar	2404
matfile_open	2405
matfile_varreadnext	2406
matfile_varwrite	2407
savematfile	2408
LV. xcos	2410
6. Batch functions	2412
lincos	2413
scicos	2414
scicos_simulate	2415
scicosim	2417
steadycos	2419
7. palettes	2421
1. Annotations palette	2421
2. Commonly used blocks palette	2424
3. Continuous time systems palette	2432
4. Demonstrations blocks palette	2457

5. Discontinuities palette	2474
6. Discrete time systems palette	2487
7. Electrical palette	2504
8. Event handling palette	2562
9. Implicit palette	2608
10. Integer palette	2613
11. Lookup tables palette	2644
12. Math operations palette	2651
13. Matrix operation palette	2698
14. Port & Subsystem palette	2761
15. Signal processing palette	2768
16. Signal routing palette	2773
17. Sinks palette	2809
18. Sources palette	2864
19. Thermohydraulics palette	2917
20. User defined functions palette	2931
21. Zero crossing detection palette	2957
8. Programming xcos Blocks	2967
1. C Computational Functions	2967
2. Scilab Computational Functions	2981
3. Utilities Functions	2985
9. Scilab Data Structures	2996
1. Blocks	2996
2. Compilation/Simulation	3007
3. Diagram	3015
4. Links	3020
10. Scilab Utilities Functions	3023
buildouttb	3024
create_palette	3025
get_scicos_version	3026
scicos_debug	3027
var2vec	3028
vec2var	3029
xcos	3031
Menu_entries	3032
LVI. Text editor	3042
edit_error	3043
Editor	3044
LVII. API Scilab	3048
11. Scilab Gateway API	3050
1. How to	3050
CheckColumn	3072
CheckDimProp	3073
CheckDims	3074
CheckLength	3075
CheckLhs	3076
CheckRhs	3077
CheckRow	3078
CheckSameDims	3079
CheckScalar	3081
CheckSquare	3082
CheckVector	3083
CreateListVarFrom	3084
CreateListVarFromPtr	3087
CreateVar	3090
FindOpt	3092
FirstOpt	3094
GetListRhsVar	3096

GetRhsVar	3098
GetType	3100
IsOpt	3101
Lhs	3103
LhsVar	3104
NumOpt	3106
OverLoad	3108
Rhs	3110
Scierror	3111
Scilab C Types	3112
get_optionals	3114
istk	3116
sci_types	3117
sciprint	3120
stk	3121
12. list_management	3122
Boolean reading (Scilab gateway)	3123
Boolean writing (Scilab gateway)	3135
Boolean sparse reading (Scilab gateway)	3139
Boolean sparse writing (Scilab gateway)	3151
Create List (Scilab gateway)	3155
Double reading (Scilab gateway)	3159
Double writing (Scilab gateway)	3171
Get child item (Scilab gateway)	3175
Item Number (Scilab gateway)	3178
Integer reading (Scilab gateway)	3181
Integer writing (Scilab gateway)	3193
Pointer reading (Scilab gateway)	3198
Pointer writing (Scilab gateway)	3210
Polynomial reading (Scilab gateway)	3214
Polynomial writing (Scilab gateway)	3226
Sparse reading (Scilab gateway)	3230
Sparse writing (Scilab gateway)	3242
String reading (Scilab gateway)	3246
String writing (Scilab gateway)	3258
api_scilab	3262
Boolean reading (Scilab gateway)	3263
Boolean writing (Scilab gateway)	3265
Boolean sparse reading (Scilab gateway)	3268
Boolean sparse writing (Scilab gateway)	3271
Variable Reference (Scilab gateway)	3274
Variable dimension (Scilab gateway)	3278
Variable Type (Scilab gateway)	3282
Variable Complexity (Scilab gateway)	3286
Matrix Type (Scilab gateway)	3290
Double reading (Scilab gateway)	3294
Double writing (Scilab gateway)	3297
Integer Precision (Scilab gateway)	3301
Integer reading (Scilab gateway)	3305
Integer writing (Scilab gateway)	3312
Pointer reading (Scilab gateway)	3319
Pointer writing (Scilab gateway)	3321
Polynomial Symbolic Variable (Scilab gateway)	3324
Polynomial reading (Scilab gateway)	3328
Polynomial writing (Scilab gateway)	3333
Sparse matrix reading (Scilab gateway)	3335
Sparse writing (Scilab gateway)	3338
String reading (Scilab gateway)	3341

String writing (Scilab gateway)	3344
LVIII. Online help management	3346
add_help_chapter	3347
apropos	3348
del_help_chapter	3349
help	3350
help_from_sci	3351
help_skeleton	3353
man	3354
manedit	3358
%helps	3359
xmltochm	3360
xmltohtml	3362
xmltojar	3364
xmltopdf	3366
xmltops	3368

Part I. Scilab

Name

`abort` — interrupt evaluation.

Description

`abort` interrupts current evaluation and gives the prompt. Within a `pause` level `abort` return to level 0 prompt.

See Also

`quit` , `pause` , `break`

Name

`add_demo` — Add an entry in the demos list

```
add_demo(title,path)
```

Parameters

`title`

a character string, the demo title

`path`

a character string, the path of the scilab script associated with the demo

Description

This function adds a new entry in the demos list. The demo should be executed by a Scilab script file. If the given `title` already exists in the demo list associated with the same file nothing is done. The function checks if the file exist.

Examples

```
//create a simple demo script
path=TMPDIR+'/foo.sce';
mputl('disp Hello',path)
add_demo('My first demo',path)
//the demo can now be run using the "Demos" menu.
```

See Also

`add_help_chapter`

Authors

Serge Steer , INRIA

Name

`ans` — answer

Description

`ans` means "answer". Variable `ans` is created automatically when expressions are not assigned. `ans` contains the last unassigned evaluated expression.

Variable `ans` is not protected by `predef`.

See Also

`predef`

Name

`argn` — Returns the number of input/output arguments in a function call

```
[lhs [,rhs] ]=argn()  
lhs=argn(1)  
rhs=argn(2)
```

Description

This function is used inside a function definition. It gives the number of actual inputs arguments (`rhs`) and output arguments (`lhs`) passed to the function when the function is called. It is usually used in function definitions to deal with optional arguments.

Examples

```
function concat=myOwnFunction(name,optional)  
    [lhs,rhs]=argn(0)  
    if rhs <= 1 then  
        optional="my Optional value"  
    end  
    if rhs == 0 then  
        error("Expect at least one argument")  
    end  
    concat=name+" "+optional  
endfunction
```

See Also

`function` , `varargin`

Name

backslash (\) — left matrix division.

```
x=A\b
```

Description

Backslash denotes left matrix division. $x=A\b$ is a solution to $A^*x=b$.

If A is square and nonsingular $x=A\b$ (uniquely defined) is equivalent to $x=\text{inv}(A)*b$ (but the computations are much cheaper).

If A is not square, x is a least square solution. i.e. $\text{norm}(A*x-b)$ is minimal (euclidian norm). If A is full column rank, the least square solution, $x=A\b$, is uniquely defined (there is a unique x which minimizes $\text{norm}(A*x-b)$). If A is not full column rank, then the least square solution is not unique, and $x=A\b$, in general, is not the solution with minimum norm (the minimum norm solution is $x=\text{pinv}(A)*b$).

$A.\B$ is the matrix with (i,j) entry $A(i,j)\B(i,j)$. If A (or B) is a scalar $A.\B$ is equivalent to $A*\text{ones}(B).\B$ (or $A.\(B*\text{ones}(A))$

$A\B$ is an operator with no predefined meaning. It may be used to define a new operator (see overloading) with the same precedence as $*$ or $/$.

Examples

```
A=rand(3,2);b=[1;1;1]; x=A\b; y=pinv(A)*b; x-y
A=rand(2,3);b=[1;1]; x=A\b; y=pinv(A)*b; x-y, A*x-b, A*y-b
A=rand(3,1)*rand(1,2); b=[1;1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b
A=rand(2,1)*rand(1,3); b=[1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b

// A benchmark of several linear solvers

[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa"

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\ ',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

See Also

slash , inv , pinv , percent , iieee , linsolve , umfpack

Name

banner — show scilab banner (Windows)

```
banner()
```

Description

show scilab banner.

Examples

```
clc();banner()
```

Authors

Allan CORNET

Name

boolean — Scilab Objects, boolean variables and operators & | ~

Description

A boolean variable is %T (for "true") or %F (for "false"). These variables can be used to define matrices of booleans, with the usual syntax. Boolean matrices can be manipulated as ordinary matrices for elements extraction/insertion and concatenation. Note that other usual operations (+, *, -, ^, etc) are undefined for booleans matrices, three special operators are defined for boolean matrices:

`~b`

is the element wise negation of boolean `b` (matrix).

`b1&b2`

is the element wise logical and of `b1` and `b2` (matrices).

`b1|b2`

is the element wise logical or of `b1` and `b2` (matrices).

Boolean variables can be used for indexing matrices or vectors.

For instance `a([%T,%F,%T],:)` returns the submatrix made of rows 1 and 3 of `a`. Boolean sparse matrices are supported.

Examples

```
[1,2]==[1,3]
[1,2]==1
a=1:5; a(a>2)
```

See Also

matrices , or , and , not

Name

brackets — ([,]) left and right brackets

```
[a11, a12, ...; a21, a22, ...; ...]  
[s1, s2, ...]=func(...)
```

Parameters

a11,a12,...

any matrix (real, polynomial, rational, `syslin` list ...) with appropriate dimensions

s1,s2,...

any possible variable name

Description

Left and right brackets are used to note vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call

Inside concatenation brackets, blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

Note : to avoid confusions it is safer to use commas instead of blank to separate columns.

Within multiple lhs brackets variable names must be separated by comma.

Examples

```
[6.9, 9.64; sqrt(-1) 0]  
[1 +%i 2 -%i 3]  
[]  
['this is'; 'a string'; 'vector']  
s=poly(0, 's'); [1/s, 2/s]  
[tf2ss(1/s), tf2ss(2/s)]  
  
[u, s]=schur(rand(3,3))
```

See Also

comma , semicolon

Name

`break` — keyword to interrupt loops

Description

Inside a `for` or `while` loop, the command `break` forces the end of the loop.

Examples

```
k=0;
while 1==1,
    k=k+1;
    if k > 100 then
        break
    end;
end
```

See Also

`while` , `if` , `for` , `abort` , `return`

Name

case — keyword used in select

Description

Keyword used in `select ... case`

Use it in the following way:

```
select expr0,  
  case expr1 then instructions1,  
  case expr2 then instructions2,  
  ...  
  case exprn then instructionsn,  
  [else instructions],  
end
```

See Also

`select` , `while` , `end` , `for`

Name

`clear` — kills variables

```
clear a
```

Description

This command kills variables which are not protected. It removes the named variables from the environment. By itself `clear` kills all the variables except the variables protected by `predef`. Thus the two commands `predef(0)` and `clear` remove all the variables.

Normally, protected variables are standard libraries and variables with the percent prefix.

Note the particular syntax `clear a` and not `clear(a)`. Note also that `a=[]` does not kill `a` but sets `a` to an empty matrix.

See Also

`predef` , `who`

Name

clearfun — remove primitive.

```
ret=clearfun('name')
```

Description

`clearfun('name')` removes the primitive 'name' from the set of primitives (built-in functions).
clearfun returns %t or %f. This function allows to rename a primitive : a Scilab primitive can be replaced by a user-defined function. For experts...

See Also

newfun , funptr

Name

`clearglobal` — kills global variables

```
clearglobal()  
clearglobal nam1 .. namn  
clearglobal('nam1', .., 'namn')
```

Parameters

`nam1,..., namn`
valid variable names

Description

`clearglobal()` kills all the global variables.

`clearglobal nam1 .. namn` kills the global variables given by their names

Note that `clearglobal()` only clears the global variables, the local copies of these global variables are not destroyed.

Examples

```
global a b c  
a=1;b=2;c=3;  
who('global')  
clearglobal b  
who('global')
```

See Also

`global` , `who`

Name

colon — (:) colon operator

Description

Colon symbol `:` can be used to form implicit vectors. (see also `linspace`, `logspace`)

`j:k`
is the vector `[j, j+1, ..., k]` (empty if `j>k`).

`j:d:k`
is the vector `[j, j+d, ..., j+m*d]`

The colon notation can also be used to pick out selected rows, columns and elements of vectors and matrices (see also `extraction`, `insertion`)

`A(:)`
is the vector of all the elements of `A` regarded as a single column.

`A(:,j)`
ys the `j`-th column of `A`

`A(j:k)`
is `[A(j), A(j+1), ..., A(k)]`

`A(:,j:k)`
is `[A(:,j), A(:,j+1), ..., A(:,k)]`

`A(:)=w`
fills the matrix `A` with entries of `w` (taken column by column if `w` is a matrix).

See Also

`matrix`, `for`, `linspace`, `logspace`

Name

comma — (,) column, instruction, argument separator

Description

Commas are used to separate parameters in functions or to separate entries of row vectors.

Blanks can also be used to separate entries in a row vector but use preferably commas.

Also used to separate Scilab instructions. (Use ; to have the result not displayed on the screen).

Examples

```
a=[1, 2, 3; 4, 5, 6];  
a=1, b=1; c=2
```

See Also

semicolon , brackets

Name

comments — comments

Description

A sequence of two consecutive slashes `//` out of a string definition marks the beginning of a comment. The slashes as well as all the following characters up to the end of the lines are not interpreted.

Inside a function, the first comment lines, up to the first instruction or an empty line may be used to provide the default contents for the function help.

Examples

```
g=9.81// the gravity

text='a//b'

function y=myfunction(x)
    // myfunction computes y=x^2+1
    // x should be a vector or matrix
    y=x^2+1
endfunction

help myfunction
```

Name

comp — scilab function compilation

```
comp(function [,opt])
```

Parameters

function

a scilab function, not compiled (type 11)

opt

flag with value 0 (default), 1 or 2.

Description

`comp(function)` compiles the function `function`. Compiled and interpreted functions are equivalent but usually compiled functions are much faster. The functions provided in the standard libraries are compiled.

The online definition as well as the short syntax of the commands `exec` and `deff` generate compiled functions. So `comp` has to be used in very particular cases. To produce uncompiled functions one must use `>exec` or `deff` with the option `"n"`.

The value `opt==2` causes the function to be compiled "for profiling". Note that now it is possible to add profiling instruction after compilation using the `add_profiling` function.

The obsolete `opt==1` option was specific to code analysis purposes and is now ignored, i.e treated as `opt==0`.

Note: the compilation takes part "in place", i.e the original function is modified and no new object is created.

See Also

`type`, `deff`, `exec`, `function`, `add_profiling`, `profile`

Name

comparison — comparison, relational operators

```
a==b
a~=b or a<>b
a<b
a<=b
a>b
a>=b
```

Parameters

a

any type of variable for `a==b`, `a~=b` `a<>b` equality comparisons and restricted to real floating point and integer array for order related comparisons `a<b`, `a<=b`, `a>b`, `a>=b`.

b

any type of variable for `a==b`, `a~=b` `a< > b` equality comparisons and restricted to real floating point and integer arrays for order related comparisons `a<b`, `a<=b`, `a>b`, `a>=b`.

Description

Two classes of operators have to be distinguished:

The equality and inequality comparisons:

`a==b`, `a~=b` (or equivalently `a<>b`). These operators apply to any type of operands.

The order related comparisons:

`a<b`, `a<=b`, `a>b`, `a>=b`. These operators apply only to real floating point and integer arrays.

The semantics of the comparison operators also depend on the operands types:

With array variables

like floating point and integer arrays, logical arrays, string arrays, polynomial and rational arrays, handle arrays, lists... the following rules apply:

- If `a` and `b` evaluates as arrays with same types and identical dimensions, the comparison is performed element by element and the result is an array of booleans of the same.
- If `a` and `b` evaluates as arrays with same types, but `a` or `b` is a 1 by 1 array the scalar is compared with each element of the other array. The result is an array of booleans of the size of the non scalar operand.
- In the others cases the result is the boolean `%f`
- If the operand data types are different but "compatible" like floating points and integers a type conversion is performed before the comparison.

With other type of operands

like `function`, `libraries`, the result is `%t` if the objects are identical and `%f` in the other cases.

Equality comparison between operands of incompatible data types returns `%f`.

Examples

```
//element wise comparisons
(1:5)==3
(1:5)<=4
(1:5)<=[1 4 2 3 0]
1<[]
list(1,2,3)~=list(1,3,3)

//object wise comparisons
(1:10)==[4,3]
'foo'==3
1==[]
list(1,2,3)==1

isequal(list(1,2,3),1)
isequal(1:10,1)

//comparison with type conversion
int32(1)==1
int32(1)<1.5
int32(1:5)<int8(3)
p=poly(0,'s','c')
p==0
p/poly(1,'s','c')==0
```

See Also

[less](#) , [boolean](#) , [isequal](#)

Name

`continue` — keyword to pass control to the next iteration of a loop

Description

Inside a `for` or `while` loop, the command `continue` passes control to the next iteration of the loop in which it appears, skipping any remaining statements between this instruction and the loop's `end` instruction.

Examples

```
for k=1:10,K=k;if k>2&k<=8 then continue,disp('hello'),end,k,end

for j=1:2
    x=[];
    for k=1:10,if k>j+1&k<=8 then continue,end,x=[x,k];end
    x
end
```

See Also

`while` , `for` , `break` , `return`

Authors

Serge Steer, INRIA

Name

debug — debugging level

```
debug( level-int )  
level-int=debug( )
```

Parameters

level-int
integer (-1 to 4)

Description

For the values 0,1,2,3,4 of `level-int`, `debug` defines various levels of debugging. This is targeted to the parser, not to Scilab scripts, and is for Scilab experts only.

Name

delbpt — delete breakpoints

```
delbpt([macroname [, linenumb]])
```

Parameters

macroname

string

linenumb

scalar integer or vector of integers

Description

Deletes the breakpoint at line `linenumb` in the function `macroname`.

`linenumb` can be a line or column vector of line numbers, or a single scalar line number.

Line numbers in `linenumb` are *physical* line numbers in function `macroname`. Note that Scilab versions before 5.0 used *logical* line numbers. The difference between physical and logical line numbers is the number of *continued lines* (see dot).

If `linenumb` is omitted, all the breakpoints in function `macroname` are deleted.

If both `macroname` and `linenumb` are omitted, all breakpoints in all functions are deleted.

Examples

```
setbpt('foo',1), setbpt('foo',10), delbpt('foo',10), dispbpt()  
delbpt('foo',1), dispbpt()  
  
setbpt('foo1',4), setbpt('foo1',9), setbpt('foo2',6), setbpt('foo3',8), dispbpt()  
delbpt('foo2'), dispbpt()  
delbpt(), dispbpt()  
  
delbpt('foo',[1,2,5,6]), dispbpt()
```

See Also

setbpt, dispbpt, pause, resume

Name

dispbpt — display breakpoints

```
dispbpt()
```

Description

`dispbpt()` displays all active breakpoints currently inserted in functions.

The line numbers displayed by `dispbpt` are *physical* line numbers in the mentioned functions. Note that Scilab versions before 5.0 showed *logical* line numbers. The difference between physical and logical line numbers is the number of *continued lines* (see dot).

See Also

setbpt, delbpt, pause, resume

Name

do — language keyword for loops

Description

May be used inside `for` or `while` instructions to separate the loop variable definition and the set of instructions.

See Also

`for` , `while`

Name

dot — (.) symbol

```
123.33
a.*b
[123, ..
456]
```

Description

.

Dot is used to mark decimal point for numbers : 3.25 and 0.001

.<op>

used in conjunction with other operator symbols (* / \ ^ ') to form other operators. Element-by-element multiplicative operations are obtained using .* , .^ , ./ , .\ or .' . For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j). Kronecker product is noted .*.' . Note that when dot follows a number it is always part of the number so 2.*x is evaluated as 2.0*x and 2 .*x is evaluated as (2).*x

..

Continuation mark. Two or more decimal points at the end of a line (or followed by a comment) causes the following line to be a continuation.

Continuation lines are handled by a preprocessor which builds a long logical line from a sequence of continuation lines. So the continuation marks can be used to cut a line at any point.

The following function foo:

```
function foo
    a=1
    disp(a), ..
    disp('ok')
endfunction
```

is equivalent to:

```
function foo
    a=1

    disp(a),disp('ok')
endfunction
```

The logical line formed by physical line 3 and physical line 4 is built as if it was entirely written in the physical line 4 while physical line 3 would be empty. This is done this way because continuation marks can be put anywhere even inside an expression.

Examples

```
//decimal point
1.345
```



```
//used as part of an operator
x=[1 2 3];x.^2 .*x // a space is required between 2 and dot

// used to enter continuation lines
T=[123,..//first element
   456] //second one

a="here I start a very long string... //but I'm not in the mood of continuing
   - and here I go on"
y=12..
45
```

See Also

star, hat, slash, backslash

Name

edit — function editing

```
edit(functionname)
```

Parameters

functionname
character string

Description

If functionname is the name of a defined scilab function `edit(functionname)` try to open the associated file `functionname.sci`.

If functionname is the name of a undefined scilab function `edit` create a `functionname.sci` file in the TMPDIR directory.

Examples

```
edit('edit') //opens editor with text of this function
edit('myfunction') //opens editor for a new function
```

See Also

manedit

Name

else — keyword in if-then-else

Description

Used with `if`.

See Also

`if`

Name

elseif — keyword in if-then-else

Description

See `if`, `then`, `else` .

Name

empty — ([]) empty matrix

Description

[] denotes the empty matrix. It is uniquely defined and has 0 row and 0 column, i.e. `size([])` = [0, 0] . The following convenient conventions are made:

`[] * A = A * [] = []`

`[] + A = A + [] = A`

`[] , A = [A, []] = A    inv([]) = []`

`det([])`=`cond([])`=`rcond([])`=1, `rank([])`=0

Matrix functions return [] or an error message when there is no obvious answer. Empty linear systems (`syslin` lists) may have several rows or columns.

Examples

```
s=poly(0,'s'); A = [s, s+1];
A+[], A*[]
A=rand(2,2); AA=A([],1), size(AA)
svd([])
w=ssrand(2,2,2); wr=[]*w; size(wr), w1=ss2tf(wr), size(w1)
```

See Also

`matrices` , `poly` , `string` , `boolean` , `rational` , `syslin`

Name

end — end keyword

Description

Used at end of loops or conditionals. `for`, `while`, `if`, `select` must be terminated by `end`.

See Also

`for`, `while`, `if`, `select`

Name

equal — (=) assignment , comparison, equal sign

Description

Assignment:

The equal sign = is used to denote the assignment of value(s) to variable(s). The syntax can be :

- `a=expr` where `a` is a variable name and `expr` a scilab expression which evaluates to a single result.
- `[a, b, . . . ]=expr` where `a,b,. . .` are variable names and `expr` a scilab expression which results in as many results as given variable names.

Comparison:

The equal sign = is also used in the comparison operators:

- `a==b`, denotes equality comparison between the values of the expressions `a` and `b`.
- `a~=b`, denotes inequality comparison between the values of the expressions `a` and `b`:
- `a<=b` and `a>=b` denotes ordering comparison between the values of the expressions `a` and `b`:

See comparison for semantic details.

Examples

```
a = sin(3.2)
M = [2.1,3.3,8.5;7.6,6.7,6.9;0,6.3,8.8];
[u,s] = schur(M)
[1:10] == 4
1~=2
```

See Also

less , great , boolean , isequal , comparison

Name

errcatch — error trapping

```
errcatch(n [, 'action'] [, 'option'])  
errcatch()
```

Parameters

n
integer

action, option
strings

Description

errcatch gives an "action" (error-handler) to be performed when an error of type **n** occurs. **n** has the following meaning:

if $n > 0$, **n** is the error number to trap

if $n < 0$ all errors are to be trapped

action is one of the following character strings:

"pause"

a pause is executed when trapping the error. This option is useful for debugging purposes. Use **whereami()** to get information on the current context.

"continue"

next instruction in the function or exec files is executed, current instruction is ignored. It is possible to check if an error has occurred using the **iserror** function. Do not forget to clear the error using the **errclear** function as soon as possible. This option is useful for error recovery. In many cases, usage of **errcatch(n, "continue", ...)** can be replaced by the use of **execstr** function or try control structure.

"kill"

default mode, all intermediate functions are killed, scilab goes back to the level 0 prompt.

"stop"

interrupts the current Scilab session (useful when Scilab is called from an external program).

option is the character string 'nomessage' for killing error message.

To set back default mode, enter **errcatch(-1, "kill")** or similarly **errcatch(-1).errcatch()** is an obsolete equivalent of **errcatch(-1)**.

The **errcatch** actions apply to the current evaluation context (function, exec, pause) and all the sub-levels. A second **errcatch** call in a sub-level hides the initial one for this sub-level. If a second **errcatch** call is made at the same level, the effect of the first one is removed.

When called in the context of a Scilab function or exec the **errcatch** is automatically reset when the function returns.

See Also

try, **errclear**, **iserror**, **whereami**, **execstr**

Name

errclear — error clearing

```
errclear([n])
```

Description

clears the action (error-handler) connected to error of type `n`.

If `n` is positive, it is the number of the cleared error ; otherwise all errors are cleared (default case)

See Also

errcatch , iserror , lasterror

Name

error — error messages

```
error(message [,n])
error(n)
error(n,pos)
```

Parameters

message

a character string. The error message to be displayed.

n

an integer. The number associated with the error message

pos

an integer, a parameter for the error message

Description

`error` function allow to issue an error message and to handle the error. By default `error` stops the current execution and resume to the prompt level. This default can be changed using the `errcatch` or `execstr(..., 'errcatch')` functions.

`error(message)` prints the character string contained in `message`. The number associated with the error message is 10000

`error(message,n)` prints the character string contained in `message`. The number associated with the error message is given by `n`. This number should be greater than 10000.

`error(n)` prints the predefined error message associated with the error number `n`.

Some predefined error messages require a parameter (see `error_table`). In this case the `pos` argument must be used `error(n,pos)` to give the parameter value. In the other cases the `pos` argument is ignored.

see `error_table` for a list of error messages and the associated error numbers.

Examples

```
error('my error message')
error(43)
error(52,3)
```

See Also

`warning` , `errcatch` , `execstr` , `lasterror`

Name

error_table — table of error messages

Description

This page gives the table of the predefined error messages, and their associated error number. Some of these error messages are used by Scilab itself for parser errors or specific builtin errors. Some others are of a more general use and can be used in Scilab functions. The starred one are those for which the syntax `error(n, pos)` is handled.

- 1 "Incorrect assignment."
- 2 "Invalid factor."
- 3 "Waiting for right parenthesis."
- 4 "Undefined variable: %s"
- 5 "Inconsistent column/row dimensions."
- 6 "Inconsistent row/column dimensions."
- 7 "Dot cannot be used as modifier for this operator."
- 8 "Inconsistent addition."
- 9 "Inconsistent subtraction."
- 10 "Inconsistent multiplication."
- 11 "Inconsistent right division."
- 12 "Inconsistent left division."
- 13 "Redefining permanent variable."
- 14 "Eye variable undefined in this context."
- 15 "Submatrix incorrectly defined."
- 16 "Incorrect command!"
- 17 "stack size exceeded! Use stacksize function to increase it."
- 18 "Too many variables!"
- 19 "Problem is singular."
- * 20 "Wrong type for argument %d: Square matrix expected."
- 21 "Invalid index."
- 22 "Recursion problems. Sorry..."
- 23 "Matrix norms available are 1, 2, inf, and fro."
- 24 "Convergence problem..."
- 25 "Bad call to primitive: %s"
- 26 "Too complex recursion! (recursion tables are full)"

27 "Division by zero..."

28 "Empty function..."

29 "Matrix is not positive definite."

30 "Invalid exponent."

31 "Incorrect string."

32 "singularity of log or tan function"

33 "too many '":'"""

34 "Incorrect control instruction syntax."

34 "Syntax error in a '%s' instruction." (if,while,select/case)

* 36 "Wrong input argument %d."

* 37 "Incorrect function at line %d."

38 "Wrong file name."

39 "Incorrect number of input arguments."

40 "Waiting for end of command."

41 "Incompatible output argument."

42 "Incompatible input argument."

43 "Not implemented in scilab..."

* 44 "Wrong argument %d."

* 45 "null matrix (argument # %d)."

46 "Incorrect syntax."

47 " end or else is missing..."

* 48 " input line longer than buffer size: %d"

49 "Incorrect file or format."

50 "subroutine not found: %s"

* 52 "Wrong type for argument %d: Real matrix expected."

* 53 "Wrong type for input argument %d: Real or complex matrix expected."

* 54 "Wrong type for input argument %d: Polynomial expected."

* 55 "Wrong type for argument %d: String expected."

* 56 "Wrong type for argument %d: List expected."

57 "Problem with comparison symbol..."

58 "Function has no input argument..."

59 "Function has no output."

60 "Wrong size for argument: Incompatible dimensions."
61 "Direct access : give format."
* 62 "End of file at line %d."
* 63 "%d graphic terminal?"
64 "Integration fails."
* 65 "%d: logical unit already used."
66 "No more logical units available!"
67 "Unknown file format."
68 "Fatal error!!! Your variables have been saved in the file : %s"
69 "Floating point exception."
70 "Too many arguments in fort (max 30)."
71 "This variable is not valid in fort."
72 "%s is not valid in this context."
73 "Error while linking."
74 "Leading coefficient is zero."
75 "Too high degree (max 100)."
* 76 "for x=val with type(val)=%d is not implemented in Scilab."
77 "%s: Wrong number of input arguments."
78 "%s: Wrong number of output arguments."
79 "Indexing not allowed for output arguments of resume."
80 "Incorrect function (argument n: %d)."
81 "%s: Wrong type for argument %d: Real or complex matrix expected."
82 "%s: Wrong type for argument %d: Real matrix expected."
83 "%s: Wrong type for argument %d: Real vector expected."
84 "%s: Wrong type for argument %d: Scalar expected."
85 "Host does not answer..."
86 "Uncontrollable system."
87 "Unobservable system."
88 "sfact: singular or asymmetric problem."
* 89 "Wrong size for argument %d."
* 90 "Wrong type for argument %d: Transfer matrix expected."
* 91 "Wrong type for argument %d: In state space form expected."

- * 92 "Wrong type for argument %d: Rational matrix expected."
- * 93 "Wrong type for argument %d: In continuous time expected."
- * 94 "Wrong type for argument %d: In discrete time expected."
- * 95 "Wrong type for argument %d: SISO expected."
- * 96 "time domain of argument %d is not defined."
- * 97 "Wrong type for argument %d: A system in state space or transfer matrix form expected."
- 98 "Variable returned by scilab argument function is incorrect."
- * 99 "Elements of %dth argument must be in increasing order."
- * 100 "Elements of %dth argument are not in (strictly) decreasing order."
- * 101 "Last element of %dth argument <> first."
- 102 "Variable or function %s are not in file."
- 103 "Variable %s is not a valid rational function."
- 104 "Variable %s is not a valid state space representation."
- 105 "Undefined fonction."
- 106 "Function name already used."
- * 107 "Too many functions are defined (maximum #:%d)."
- 108 "Too complex for scilab, may be a too long control instruction."
- 109 "Too large, can't be displayed."
- 110 "%s was a function when compiled but is now a primitive!"
- 111 "Trying to re-define function %s."
- 112 "No more memory."
- 113 "Too large string."
- 114 "Too many linked routines."
- 115 "Stack problem detected within a loop."
- * 116 "Wrong value for argument %d."
- * 117 "List element number %d is Undefined."
- * 118 "Wrong type for argument %d: Named variable not an expression expected."
- 120 "Indices for non-zero elements must be given by a 2 column matrix."
- 121 "Incompatible indices for non-zero elements."
- * 122 "Logical unit number should be larger than %d."
- 123 "Function not bounded from below."
- 125 "Problem may be unbounded: too high distance between two consecutive iterations."

126 "Inconsistent constraints."
127 "No feasible solution."
128 "Degenerate starting point."
129 "No feasible point has been found."
130 "Optimization fails: back to initial point."
131 "optim: Stop requested by simulator (ind=0)"
132 "optim: Wrong input parameters."
133 "Too small memory."
134 "optim: Problem with initial constants in simul."
135 "optim : Bounds and initial guess are incompatible."
136 "optim : This method is NOT implemented."
137 "NO hot restart available in this method."
138 "optim: Incorrect stopping parameters."
139 "optim: Incorrect bounds."
140 "Variable : %s must be a list"
* 141 "Incorrect function (argument n: %d)."
* 142 "Hot restart: dimension of working table (argument n:%d)."
143 "optim:: df0 must be positive !"
144 "Undefined operation for the given operands."
201 "%s: Wrong type for argument %d: Real or complex matrix expected."
202 "%s: Wrong type for argument %d: Real matrix expected."
203 "%s: Wrong type for argument %d: Real vector expected."
* 204 "%s: Wrong type for argument %d: Scalar expected."
205 "%s: Wrong size for argument %d: (%d,%d) expected."
206 "%s: Wrong size for argument %d: %d expected."
207 "%s: Wrong type for argument %d: Matrix of strings expected."
208 "%s: Wrong type for argument %d: Boolean matrix expected."
209 "%s: Wrong type for argument %d: Matrix expected."
210 "%s: Wrong type for argument %d: List expected."
211 "%s: Wrong type for argument %d: Function or string (external function) expected."
212 "%s: Wrong type for argument %d: Polynomial expected."
213 "%s: Wrong type for argument %d: Working integer matrix expected."

214 "Argument %d of %s: wrong type argument, expecting a vector"

* 215 "%dth argument type must be boolean."

* 216 "Wrong type for argument %d: Boolean or scalar expected."

* 217 "Wrong type for argument %d: Sparse matrix of scalars expected."

* 218 "Wrong type for argument %d: Handle to sparse lu factors expected."

* 219 "Wrong type argument %d: Sparse or full scalar matrix expected."

220 "Null variable cannot be used here."

221 "A sparse matrix entry is defined with two different values."

222 "%s not yet implemented for full input parameter."

223 "It is not possible to redefine the %s primitive this way (see clearfun)."

224 "Type data base is full."

225 "This data type is already defined."

226 "Inequality comparison with empty matrix."

227 "Missing index."

228 "reference to the cleared global variable %s."

229 "Operands of / and \ operations must not contain NaN or Inf."

230 "semi def fails."

231 "Wrong type for first input argument: Single string expected."

232 "Entry name not found."

233 "Maximum number of dynamic interfaces reached."

234 "link: expecting more than one argument."

235 "link: problem with one of the entry point."

236 "link: the shared archive was not loaded."

237 "link: Only one entry point allowed on this operating system."

238 "link: First argument cannot be a number."

239 "You cannot link more functions, maxentry reached."

240 "File '%s' already exists or directory write access denied."

241 "File '%s' does not exist or read access denied."

242 "Binary direct access files must be opened by 'file'."

243 "C file logical unit not allowed here."

244 "Fortran file logical unit not allowed here."

* 245 "No input file associated to logical unit %d."

246 "function not defined for given argument type(s)"

247 "Wrong value for argument %d: the lu handle is no more valid."

* 248 "Wrong value for argument %d: Valid variable name expected."

* 249 "Wrong value for argument %d: Empty string expected."

250 "Recursive extraction is not valid in this context."

251 "bvode: ipar dimensioned at least 11."

252 "bvode: ltol must be of size ipar(4)."

253 "bvode: fixpnt must be of size ipar(11)."

254 "bvode: ncomp < 20 requested."

255 "bvode: m must be of size ncomp."

256 "bvode: sum(m) must be less than 40."

257 "bvode: sum(m) must be less than 40."

258 "bvode: input data error."

259 "bvode: no. of subintervals exceeds storage."

260 "bvode: Th colocation matrix is singular."

261 "Interface property table is full."

* 262 "Too many global variables! , max number is %d."

263 "Error while writing in file,(disk full or deleted file."

* 264 "Wrong value for argument %d: Must not contain NaN or Inf."

265 "A and B must have equal number of rows."

266 "A and B must have equal number of columns."

267 "A and B must have equal dimensions."

* 268 "Invalid return value for function passed in arg %d."

* 269 "Wrong value for argument %d: eigenvalues must have negative real parts."

* 270 "Wrong value for argument %d: eigenvalues modulus must be less than one."

* 271 "Size varying argument a*eye(), (arg %d) not allowed here."

272 "endfunction is missing."

273 "Instruction left hand side: waiting for a dot or a left parenthesis."

274 "Instruction left hand side: waiting for a name."

275 "varargout keyword cannot be used here."

276 "Missing operator, comma, or semicolon."

277 "Too many commands defined."

278 "%s: Input arguments should have the same formal variable name."

See Also

warning, errcatch, execstr, lasterror

Name

evstr — evaluation of expressions

```
H=evstr(Z)
[H,ierr]=evstr(Z)
```

Parameters

Z
matrix of character strings **M** or `list(M, Subexp)`

M
matrix of character strings

Subexp
vector of character strings

H
matrix

ierr
integer, error indicator

Description

Returns the result of the evaluation of the matrix of character strings **M**. Each element of the matrix must define a valid Scilab expression.

If the evaluation of **M** expression leads to an error, the single return value version, `H=evstr(M)`, raises the error as usual. The two return values version, `[H,ierr]=evstr(M)`, on the other hand, produces no error, but returns the error number in **ierr**.

If **Z** is a list, **Subexp** is a vector of character strings, that defines sub_expressions which are evaluated before evaluating **M**. These sub_expressions must be referred to as `%(k)` in **M**, where **k** is the sub-expression's index in **Subexp**.

`evstr('a=1')` is not valid (use `execstr` instead).

Examples

```
a=1; b=2; Z=['a','b'] ; evstr(Z)

a=1; b=2; Z=list(['%(1)', '%(1)-%(2)'], ['a+1', 'b+1']);
evstr(Z)
```

See Also

`execstr`

Name

exists — checks variable existence

```
exists(name [,where])
```

Parameters

name

a character string

where

an optional character with possible values: 'l' (local), 'n' (nolocal) and 'a' (all). The default value is 'all'.

Description

`exists(name)` returns 1 if the variable named `name` exists and 0 otherwise.

Caveats: a function which uses `exists` may return a result which depends on the environment!

`exists(name, 'local')` returns 1 if the variable named `name` exists in the environment of the current function and 0 otherwise.

`exists(name, 'nolocal')` returns 1 if the variable named `name` exists in any level of the calling environment (including the Scilab shell main level) of the current function and 0 otherwise.

Warning: the `exists` function does not check if a variable exists in the global namespace.

Examples

```
deff('foo(x)',...
['disp([exists('a12'),exists('a12','local']))'
 'disp([exists('x'),exists('x','local']))'])
foo(1)
a12=[];foo(1)

function level1()
    function level2()
        disp(exists("a","all"));
        disp(exists("a","local"));
        disp(exists("a","nolocal"));
    endfunction
    level2()
endfunction
function go()
    a=1;
    level1()
endfunction
go()
```

See Also

`isdef`, `isglobal`, `whereis`, `type`, `typeof`, `macrovar`

Name

exit — Ends the current Scilab session

```
exit  
exit(n)
```

Description

Ends the current Scilab session

n

The exit status shall be n, if specified. Otherwise, the value shall be zero.

Examples

```
exit  
i = 4;  
exit(i);
```

See Also

quit, abort, break, resume

Name

external — Scilab Object, external function or routine

Description

External function or routine for use with specific commands.

An "external" is a function or routine which is used as an argument of some high-level primitives (such as `ode`, `optim`, `schur`...).

The calling sequence of the external (function or routine) is imposed by the high-level primitive which sets the arguments of the external.

For example the external function `costfunc` is an argument of the `optim` primitive. Its calling sequence must be: `[f,g,ind]=costfunc(x,ind)` and `optim` (the high-level optimization primitive) is invoked as follows:

```
optim(costfunc,...)
```

Here `costfunc` (the cost function to be minimized by the primitive `optim`) evaluates $f=f(x)$ and g = gradient of f at x (`ind` is an integer. Its use is precised in the `optim` help).

If other values are needed by the external function these variables can be defined in its environment. Also, they can be put in a list. For example, the external function

```
[f,g,ind]=costfunc(x,ind,a,b,c)
```

is valid for `optim` if the external is `list(costfunc,a,b,c)` and the call to `optim` is then:

```
optim(list(costfunc,a1,b1,c1),....
```

An external can also be a Fortran or C routine : this is convenient to speed up the computations.

The name of the routine is given to the high-level primitive as a character string. The calling sequence of the routine is also imposed.

External Fortran or C routines can also be dynamically linked (see `link`)

See Also

`ode` , `optim` , `impl` , `dassl` , `intg` , `schur` , `gschur`

Name

extraction — matrix and list entry extraction

```
x(i)
x(i,j)
x(i,j,k,...)
[...] = l(i)
[...] = l(k1)...(kn)(i) or [...] = l(list(k1,...,kn,i))
l(k1)...(kn)(i,j) or l(list(k1,...,kn,list(i,j)))
```

Parameters

x
matrix of any possible types

l
list variable

i,j,k
indices

k1,...kn
indices

Description

MATRIX CASE

i, j, k,... can be:

a real scalar or a vector or a matrix with positive elements.

- $r = x(i, j)$ builds the matrix r such as $r(l, k) = x(\text{int}(i(l)), \text{int}(j(k)))$ for l from 1 to $\text{size}(i, '*')$ and k from 1 to $\text{size}(j, '*')$.

i(j) Maximum value must be less or equal to $\text{size}(x, 1)$ ($\text{size}(x, 2)$).

- $r = x(i)$ with x a 1x1 matrix builds the matrix r such as $r(l, k) = x(\text{int}(i(l)), \text{int}(i(k)))$ for l from 1 to $\text{size}(i, 1)$ and k from 1 to $\text{size}(i, 2)$.

Note that in this case index *i* is valid only if all its entries are equal to one.

- $r = x(i)$ with x a row vector builds the row vector r such as $r(l) = x(\text{int}(i(l)))$ for l from 1 to $\text{size}(i, '*')$ *i* Maximum value must be less or equal to $\text{size}(x, '*')$.
- $r = x(i)$ with x a matrix with one or more columns builds the column vector r such as $r(l)$ (l from 1 to $\text{size}(i, '*')$) contains the $\text{int}(i(l))$ entry of the column vector formed by the concatenation of the x 's columns.

i Maximum value must be less or equal to $\text{size}(x, '*')$.

the symbol `*` :
which stands for "all elements".

- $r = x(i, :)$ builds the matrix r such as $r(l, k) = x(\text{int}(i(l)), k)$ for l from 1 to $\text{size}(i, '*')$ and k from 1 to $\text{size}(x, 2)$

- $r = x(:, j)$ builds the matrix r such as $r(l, k) = x(l, \text{int}(j(k)))$ for l from 1 to $\text{size}(r, 1)$ and k from 1 to $\text{size}(j, '*')$.
- $r = x(:)$ builds the column vector r formed by the column concatenations of x columns. It is equivalent to $\text{matrix}(x, \text{size}(x, '*'), 1)$.

a vector of boolean

If an index (i or j) is a vector of booleans it is interpreted as `find(i)` or respectively `find(j)`

a polynomial

If an index (i or j) is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i, m)` or respectively `horner(j, n)` where m and n are associated x dimensions. Even if this feature works for all polynomials, it is recommended to use polynomials in $\$$ for readability.

For matrices with more than 2 dimensions (see: hypermatrices) the dimensionality is automatically reduced when right-most dimensions are equal to 1.

LIST OR TLIST CASE

If they are present the k_i give the path to a sub-list entry of l data structure. They allow a recursive extraction without intermediate copies. The instructions

`[...] = l(k1) ... (kn)(i)`

and

`[...] = l(list(k1, ..., kn, i))`

are interpreted as:

$lk1 = l(k1) \dots lk_n = lk_{n-1}(kn) [\dots] = lk_n(i)$ And the $l(k1) \dots (kn)(i, j)$ and

`l(list(k1, ..., kn, list(i, j)))` instructions are interpreted as: $lk1 = l(k1) \dots lk_n = lk_{n-1}(kn) lk_n(i, j)$ i and j , can be: When path points on more than one list component the instruction must have as many left hand side arguments as selected components. But if the extraction syntax is used within a function input calling sequence each returned list component is added to the function calling sequence.

Note that, `l(list())` is the same as `l`.

i and j may be :

real scalar or vector or matrix with positive elements.

`[r1, ... rn] = l(i)` extracts the $i(k)$ elements from the list l and store them in r_k variables for k from 1 to $\text{size}(i, '*')$

the symbol :

which stands for "all elements".

a vector of booleans.

If i is a vector of booleans it is interpreted as `find(i)`.

a polynomial.

If i is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i, m)` where $m = \text{size}(l)$. Even if this feature works for all polynomials, it is recommended to use polynomials in $\$$ for readability.

$k1, \dots, kn$ may be :

real positive scalar,

a polynomial,
interpreted as `horner(ki,m)` where `m` is the corresponding sub-list size.

a character string
associated with a sub-list entry name.

Remarks

For soft coded matrix types such as rational functions and state space linear systems, `x(i)` syntax may not be used for vector element extraction due to confusion with list element extraction. `x(1,j)` or `x(i,1)` syntax must be used.

Examples

```
// MATRIX CASE
a=[1 2 3;4 5 6]
a(1,2)
a([1 1],2)
a(:,1)
a(:,3:-1:1)
a(1)
a(6)
a(:)
a([%t %f %f %t])
a([%t %f],[2 3])
a(1:2,$-1)
a($:-1:1,2)
a($)
//
x='test'
x([1 1;1 1;1 1])
//
b=[1/%s, (%s+1)/(%s-1)]
b(1,1)
b(1,$)
b(2) // the numerator
// LIST OR TLIST CASE
l=list(1, 'qwerw', %s)
l(1)
[a,b]=l([3 2])
l($)
x=tlist(l(2:3)) //form a tlist with the last 2 components of l
//
dts=list(1, tlist(['x'; 'a'; 'b'], 10, [2 3]));
dts(2)('a')
dts(2)('b')(1,2)
[a,b]=dts(2)(['a', 'b'])
```

See Also

`find`, `horner`, `parents`

Name

for — language keyword for loops

Description

Used to define loops. Its syntax is: `for variable=expression ,instruction, .. ,instruction,end`

`for variable=expression do instruction, ,instruction,end`

If `expression` is a matrix or a row vector, `variable` takes as values the values of each column of the matrix.

A particular case uses the colon operator to create regularly spaced row vectors, and resemble to traditional for loop forms: `for variable=n1:step:n2, ...,end`

If `expression` is a list `variable` takes as values the successive entries of the list.

Warning: the number of characters used to define the body of any conditional instruction (if while for or select/case) must be limited to 16k.

Examples

```
// "traditional" for loops
n=5;
for i = 1:n, for j = 1:n, a(i,j) = 1/(i+j-1);end;end
for j = 2:n-1, a(j,j) = j; end; a
for j= 4:-1:1, disp(j),end // decreasing loop

//loop on matrix columns
for e=eye(3,3),e,end
for v=a, write(6,v),end
for j=1:n,v=a(:,j), write(6,v),end

//loop on list entries
for l=list(1,2,'example'); l,end
```

See Also

while , end , do

Name

format — number printing and display format

```
format([type],[long])  
v = format()  
format(m)
```

Parameters

type

character string

long

integer (max number of digits (default 10))

v

a vector for the current format v(1) type format : 0 for 'e' and 1 for 'v' v(2) number of digits

m

a vector to set new format

m(1) number of digits

m(2) type format : 0 for 'e' and 1 for 'v'

Description

Sets the current printing format with the parameter `type` ; it is one of the following :

"v"

for a variable format (default)

"e"

for the e-format.

`long` defines the max number of digits (default 10). `format( )` returns a vector for the current format: first component is the type of format (1 if v ; 0 if e) ; second component is the number of digits.

In the old Scilab versions, in "variable format" mode, vectors entries which are less than %eps times the maximum absolute value of the entries were displayed as "0". It is no more the case, the `clean` function can be used to set negligible entries to zeros.

Examples

```
x=rand(1,5);  
format('v',10);x  
format(20);x  
format('e',10);x  
format(20);x  
  
x=[100 %eps];  
format('e',10);x  
format('v',10);x  
  
format("v")
```

See Also

write, disp, print

Name

funcprot — switch scilab functions protection mode

```
prot=funcprot()  
funcprot(prot)
```

Parameters

prot
integer with possible values 0,1,2

Description

Scilab functions are variable, funcprot allows the user to specify what scilab do when such variables are redefined.

- If prot==0 nothing special is done
- If prot==1 scilab issues a warning message when a function is redefined (default mode)
- If prot==2 scilab issues an error when a function is redefined

Examples

```
funcprot(1)  
deff(' [x]=foo(a)', 'x=a')  
deff(' [x]=foo(a)', 'x=a+1')  
foo=33  
funcprot(0)  
deff(' [x]=foo(a)', 'x=a')  
deff(' [x]=foo(a)', 'x=a+1')  
foo=33
```

Name

funptr — coding of primitives (wizard stuff)

```
[numptr] = funptr(name)
```

Parameters

name

a string, the name of a primitive

numptr

the internal routine number of the primitive

Description

Utility function (for experts only) to get the internal routine number `numptr` of the primitive '`name`'. `numptr` is formed from the interface number `fun` and the routine number `fin` of the primitive in its interface by `numptr = 1000*fun + fin` (`fin < 1000`). From `numptr` you can get the interface number `fun = floor(numptr/1000)` which may be useful to link a dynamical interface with arguments passed by reference (see example section).

Examples

```
// Suppose you want to load some codes via the dynamic
// loading facilities offers by addinter. By default
// arguments are passed by values but if you want to
// pass them by reference you can do the following
// (name being the scilab name of one of the interfaced
// routines) :
//
// addinter(files,spnames,fcts) // args passed by values
// num_interface = floor(funptr(name)/1000)
// intppty(num_interface) // args now passed by reference
//
// Note that if you enter the following
//
// intppty()
//
// you will see all the interfaces working by reference
```

See Also

clearfun , newfun , intppty , addinter

Name

getdebuginfo — get information about Scilab to debug

```
getdebuginfo()  
dynamic_info = getdebuginfo();  
[dynamic_info,static_info] = getdebuginfo();
```

Description

getdebuginfo get information about Scilab to debug.

dynamic_info = getdebuginfo(); returns information about your system.

[dynamic_info,static_info] = getdebuginfo(); returns information about your system and about Scilab.

See Also

getversion

Authors

A.C

Name

getmd5 — get md5 checksum

```
res=getmd5(filename)
res=getmd5(ParamString, 'string')
```

Parameters

res
md5 result (string)

filename
filename (string or matrix of strings)

ParamString
string or matrix of strings

Description

getmd5(. . .) get md5 checksum of a file or a string.

Examples

```
getmd5('hello world', 'string')
getmd5(['hello' 'world'], 'string')
getmd5(['hello' ; 'world'], 'string')

getmd5( SCI+'/modules/core/etc/core.start' )
getmd5( SCI+'/modules/core/etc/'+['core.start' 'core.quit'])
```

Authors

A.C.

Name

getmemory — returns free and total system memory

```
[free, total]=getmemory()
```

Description

getmemory() returns free system memory (kilo-octets).

[free, total]=getmemory() returns free and total system memory (kilo-octets).

Authors

A.C

Name

getmodules — returns list of modules installed in Scilab

```
res=getmodules()
```

Parameters

res
a string matrix

Description

Returns list of modules installed in Scilab.

See Also

with_module

Authors

A.c

Name

getos — return Operating System name and version

```
OS=getos()  
[OS,Version]=getos()
```

Description

getos return Operating System name and version

Examples

```
OS=getos()  
[OS,version]=getos()
```

Authors

A.C

Name

getscilabmode — returns scilab mode

```
mode = getscilabmode()
```

Description

returns scilab mode. 4 modes are possible : STD , API , NW , NWN .

APIScilab is launch as an API.

STDThe standard Scilab (gui, plot ...)

NWScilab in command line with the plots.

NWNIScilab in command line without any graphics.

Examples

```
getscilabmode()
```

See Also

scilab

Name

`getshell` — returns current command interpreter.

```
getshell()
```

Description

`getshell` returns current command interpreter.

Examples

```
getshell()
```

Authors

Allan CORNET

Name

getvariablesystack — get variable names on stack of scilab

```
s=getvariablesystack()  
s=getvariablesystack('local')  
s=getvariablesystack('global')
```

Parameters

s
a string matrix

Description

return in S variable names on scilab stack.

getvariablesystack('local') returns local variables on scilab stack.

getvariablesystack('global') returns global variables on scilab stack.

Variables are sorted by alphabetical order.

Examples

```
getvariablesystack()  
getvariablesystack('local')  
getvariablesystack('global')
```

See Also

who

Name

getversion — get scilab and modules version information

```
version=getversion()  
[version,opts]=getversion()  
ver=getversion('scilab')  
versioninfo=getversion('scilab','string_info')  
ver=getversion('<module>')  
versioninfo=getversion('<module>','string_info')
```

Parameters

version

a string

versioninfo

a string about version

ver

a integer vector

ver(1) Major version

ver(2) Minor version

ver(3) Maintenance version

ver(4) GIT timestamp

opts

a vector of string with seven entries :[compiler, pvm, tk, modelicac,
release_mode, release_date, release_time]

Description

return in **version** the Scilab version name and in **opts** build options which can be used to determine if scilab has been build with pvm, tk or modelicac and give release date and time.

Examples

```
getversion()  
[version,opts]=getversion()  
ver=getversion('scilab')  
verstr=getversion('scilab','string_info')  
ver=getversion('overloading')  
verstr=getversion('overloading','string_info')
```

See Also

getmodules

Name

global — Define global variable

```
global('nam1',..., 'namn')  
global nam1 ... namn
```

Parameters

nam1,..., namn
valid variable names

Description

Ordinarily, each Scilab function, has its own local variables and can "read" all variables created in the base workspace or by the calling functions. The `global` keyword allow to make variables read/write across functions. Any assignment to that variable, in any function, is available to all the other functions declaring it `global`.

If the global variable doesn't exist the first time you issue the `global` statement, it will be initialized to the empty matrix.

Examples

```
//first: calling environment and a function share a variable  
global a  
a=1  
deff('y=f1(x)', 'global a, a=x^2, y=a^2')  
f1(2)  
a  
//second: three functions share variables  
deff('initdata()', 'global A C ;A=10,C=30')  
deff('letsgo()', 'global A C ;disp(A) ;C=70')  
deff('letsgo1()', 'global C ;disp(C)')  
initdata()  
letsgo()  
letsgo1()
```

See Also

`who` , `isglobal` , `clearglobal` , `gstacksize` , `resume`

Name

gstacksize — set/get scilab global stack size

```
gstacksize(n)
gstacksize('max')
gstacksize('min')
sz=gstacksize()
```

Parameters

n
integer, the required global stack size given in number of double precision words

sz
2-vector [total used]

Description

Scilab stores global variables in a stack

`gstacksize(n)` allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time. Note that Scilab can increase automatically the global stacksize when needed

`sz=gstacksize()` returns a 2-vector which contains the current total and used global stack size.

`gstacksize('max')` allows the user to increase the size of this global stack to the maximum.

`gstacksize('min')` allows the user to decrease the size of this global stack to the minimum.

See Also

`who` , `stacksize`

Name

hat — (^) exponentiation

A^b

Description

Exponentiation of matrices or vectors by a constant vector.

If A is a vector or a rectangular matrix the exponentiation is done element-wise, with the usual meaning.

For square A matrices the exponentiation is done in the matrix sense.

For boolean, polynomial and rational matrices, the exponent must be an integer

Remarks

$123.^b$ is interpreted as $(123).^b$. In such cases dot is part of the operator, not of the number.

For two real or complex numbers $x1$ et $x2$ the value of $x1^x2$ is the "principal value" determined by $x1^x2 = \exp(x2 \cdot \log(x1))$.

Examples

```
2^4
(-0.5)^(1/3)
[1 2; 2 4]^(1+%i)
s=poly(0, "s");
[1 2 s]^4
[s 1; 1 s]^(-1)
```

See Also

`exp`, `inv`

Name

ieee — set floating point exception mode

```
mod=ieee()  
ieee(mod)
```

Parameters

mod
integer scalar whose possible values are 0,1,or 2

Description

ieee() returns the current floating point exception mode.

0
floating point exception produces an error

1
floating point exception produces a warning

2
floating point exception produces Inf or Nan

ieee(mod) sets the current floating point exception mode.

The initial mode value is 0.

Remarks

Floating point exception arising inside some library algorithms are not yet handled by ieee modes.

Examples

```
ieee(1);1/0  
ieee(2);1/0, log(0)
```

See Also

errcatch

Name

if then else — conditional execution

```
if expr1 then statements
elseif expr1 then statements
....
else statements
end
```

Description

The `if` statement evaluates a logical expression and executes a group of statements when the expression is true.

The `expr i` are expressions with numeric or boolean values. If `expr i` are matrix valued the condition is true only if all matrix entries are true or different from zero.

The optional `elseif` and `else` provide for the execution of alternate groups of statements. An `end` keyword, which matches the `if`, terminates the last group of statements. The line structure given above is not significant, the only constraint is that each `then` keyword must be on the same line as its corresponding `if` or `elseif` keyword.

The keyword `then` can be replaced by a carriage return or a comma.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

Examples

```
i=2
for j = 1:3,
  if i == j then
    a(i,j) = 2;
  elseif abs(i-j) == 1 then
    a(i,j) = -1;
  else a(i,j) = 0;
  end,
end
```

See Also

`try` , `while` , `select` , `boolean` , `end` , `then` , `else`

Name

insertion — partial variable assignation or modification
assignation — partial variable assignation

```
x(i,j)=a  
x(i)=a  
l(i)=a  
l(k1)...(kn)(i)=a or l(list(k1,...,kn,i))=a  
l(k1)...(kn)(i,j)=a or l(list(k1,...,kn,list(i,j)))=a
```

Parameters

x
matrix of any kind (constant, sparse, polynomial,...)

l
list

i,j
indices

k1,...kn
indices with integer value

a
new entry value

Description

MATRIX CASE

if **x** is a matrix the indices **i** and **j**, may be:

Real scalars or vectors or matrices

In this case the values given as indices should be positive and it is only their integer part which taken into account.

- if **a** is a matrix with dimensions (`size(i, '*')`, `size(j, '*')`), `x(i,j)=a` returns a new **x** matrix such as `x(int(i(l)),int(j(k)))=a(l,k)` for **l** from 1 to `size(i, '*')` and **k** from 1 to `size(j, '*')`, other initial entries of **x** are unchanged.
- if **a** is a scalar `x(i,j)=a` returns a new **x** matrix such as `x(int(i(l)),int(j(k)))=a` for **l** from 1 to `size(i, '*')` and **k** from 1 to `size(j, '*')`, other initial entries of **x** are unchanged.
- If **i** or **j** maximum value exceed corresponding **x** matrix dimension, array **x** is previously extended to the required dimensions with zeros entries for standard matrices, 0 length character string for string matrices and false values for boolean matrices.
- `x(i,j)=[ ]` kills rows specified by **i** if **j** matches all columns of **x** or kills columns specified by **j** if **i** matches all rows of **x**. In other cases `x(i,j)=[ ]` produce an error.
- `x(i)=a` with **a** a vector returns a new **x** matrix such as `x(int(i(l)))=a(l)` for **l** from 1 to `size(i, '*')`, other initial entries of **x** are unchanged.
- `x(i)=a` with **a** a scalar returns a new **x** matrix such as `x(int(i(l)))=a` for **l** from 1 to `size(i, '*')`, other initial entries of **x** are unchanged.

If **i** maximum value exceed `size(x,1)`, **x** is previously extended to the required dimension with zeros entries for standard matrices, 0 length character string for string matrices and false values for boolean matrices.

if

x is a 1x1

matrix a may be a row (respectively a column) vector with dimension $\text{size}(i, '*')$. Resulting x matrix is a row (respectively a column) vector

if

x is a row

vector a must be a row vector with dimension $\text{size}(i, '*')$

if

x is a column

vector a must be a column vector with dimension $\text{size}(i, '*')$

if

x is a general

matrix a must be a row or column vector with dimension $\text{size}(i, '*')$ and i maximum value cannot exceed $\text{size}(x, '*')$,

- $x(i)=[]$ kills entries specified by i .

The `:` symbol

the `:` symbol stands for "all elements".

- $x(i, :)=a$ is interpreted as $x(i, 1:\text{size}(x, 2))=a$
- $x(:, j)=a$ is interpreted as $x(1:\text{size}(x, 1), j)=a$
- $x(:)=a$ returns in x the a matrix reshaped according to x dimensions. $\text{size}(x, '*')$ must be equal to $\text{size}(a, '*')$

Vectors of boolean

If an index (i or j) is a vector of booleans it is interpreted as `find(i)` or respectively `find(j)`

Polynomials

If an index (i or j) is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i, m)` or respectively `horner(j, n)` where m and n are associated x dimensions. Even if this feature works for all polynomials, it is recommended to use polynomials in `$` for readability.

LIST OR TLIST CASE

- If they are present the k_i give the path to a sub-list entry of l data structure. They allow a recursive insertion without intermediate copies. The $l(k_1) \dots (k_n)(i)=a$ and $l(\text{list}(k_1, \dots, k_n, i)=a)$ instructions are interpreted as:

$lk_1 = l(k_1) \dots = \dots$

$lkn = lkn-1(kn) \quad lkn(i) = a$

$lkn-1(kn) = lkn \dots = \dots \quad l(k_1) = lk_1$

And the $l(k_1) \dots (k_n)(i, j)=a$ and $l(\text{list}(k_1, \dots, k_n, \text{list}(i, j))=a)$ instructions are interpreted as:

$lk_1 = l(k_1) \dots = \dots$

$$l_{kn} = l_{kn-1}(kn) \quad l_{kn}(i, j) = a$$

$$l_{kn-1}(kn) = l_{kn} \quad \dots = \dots \quad l(k_1) = l_{k_1}$$

- i may be :
 - a real non negative scalar. $l(0)=a$ adds an entry on the "left" of the list $l(i)=a$ sets the i entry of the list l to a . if $i > \text{size}(l)$, l is previously extended with zero length entries (undefined). $l(i)=\text{null}()$ suppress the i th list entry.
 - a polynomial. If i is a polynomial it is interpreted as $\text{horner}(i, m)$ where $m = \text{size}(l)$. Even if this feature works for all polynomials, it is recommended to use polynomials in \$ for readability.
- $k_1, \dots, k_n$ may be :
 - real positive scalar.
 - a polynomial, interpreted as $\text{horner}(k_i, m)$ where m is the corresponding sub-list size.
 - a character string associated with a sub-list entry name.

Remarks

For soft coded matrix types such as rational functions and state space linear systems, $x(i)$ syntax may not be used for vector entry insertion due to confusion with list entry insertion. $x(1, j)$ or $x(i, 1)$ syntax must be used.

Examples

```
// MATRIX CASE
a=[1 2 3;4 5 6]
a(1,2)=10
a([1 1],2)=[-1;-2]
a(:,1)=[8;5]
a(1,3:-1:1)=[77 44 99]
a(1)=%s
a(6)=%s+1
a(:)=1:6
a([%t %f],1)=33
a(1:2,$-1)=[2;4]
a($:-1:1,1)=[8;7]
a($)=123
//
x='test'
x([4 5])=['4','5']
//
b=[1/%s, (%s+1)/(%s-1)]
b(1,1)=0
b(1,$)=b(1,$)+1
b(2)=[1 2] // the numerator
// LIST OR TLIST CASE
l=list(1,'qwerw',%s)
l(1)='Changed'
l(0)='Added'
l(6)=['one more';'added']
//
```

```
//  
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));  
dts(2).a=33  
dts(2)('b')(1,2)=-100
```

See Also

find , horner , parents , extraction

Name

intppty — set interface argument passing properties

```
funcs=intppty()  
intppty(fun)
```

Parameters

fun

integer an interface number (see funptr)

funcs

integer vector, vector of interface number (see funptr)

Description

The interface programs may be written in 2 different ways for the mode of function argument passing.

In the first and default way, the arguments are passed by value. With the following syntax:

```
foo(A, 1+2)
```

the argument associated with `A` will be passed by value (a copy of `A` is made before `foo` is called, and the argument associated with `1+2` will be passed by value.

In the second way arguments may be passed by reference if there are "named arguments" (no copy of the variable value is done). `intppty(fun)` with `fun>0` tells Scilab that the interface with number `fun` can handle arguments passed by reference. With the following syntax:

```
foo(A, 1+2)
```

the argument associated with `A` will be passed by reference, and the argument associated with `1+2` will be passed by value.

Warning, declaring that the interface with number `fun` can handle arguments passed by reference if it is not the case should produce unpredictable results.

`intppty(fun)` with `fun<0` suppress this property for the interface `-fun`.

`intppty()` returns the vector of interfaces which handle arguments passed by reference.

This function may be useful for dynamically loaded interface (see `addinter`).

See Also

funptr , addinter

Name

inv_coeff — build a polynomial matrix from its coefficients

```
[P]=inv_coeff(C,[,d,[name]])
```

Parameters

C
big matrix of the coefficients

d
Polynomial matrix degree. optional parameter with default value $d=-1+\text{size}(C, 'c')/\text{size}(C, 'r')$

name
string giving the polynomial variable name (default value 'x').

Description

$P=\text{inv\_coeff}(M_p, k)$, when k is compatible with M_p size, returns a polynomial matrix of degree k . $C=[C_0, C_1, \dots, C_k]$ and $P= C_0 + C_1*x + \dots + C_k*x^k$.

Examples

```
A=int(10*rand(2,6))
// Building a degree 1 polynomial matrix
P=inv_coeff(A,1)
norm(coeff(P)-A)
// Using default value for degree
P1=inv_coeff(A)
norm(coeff(P1)-A)
```

See Also

poly , degree , coeff

Name

iserror — error occurrence test

```
iserror([n])
```

Description

tests if error number n has occurred (after a call to `errcatch`). `iserror` returns 1 if the error occurred and 0 otherwise

$n > 0$ is the error number ; all errors are tested with $n < 0$.

See Also

`error` , `errcatch`

Name

`isglobal` — check if a variable is global

```
t=isglobal(x)
```

Parameters

`x`
any variable

`t`
a boolean

Description

`isglobal(x)` returns true if `x` has been declared to be a global variable and false otherwise.

Examples

```
isglobal(1)
global a
isglobal(a)
```

See Also

`global` , `clearglobal` , `who`

Name

lasterror — get last recorded error message

```
str=lasterror( [opt] )  
[str,n]=lasterror([opt])  
[str,n,line,func]=lasterror([opt])
```

Parameters

str
vector of character strings or an empty matrix. The last recorded error message.

n
integer, 0 or the last recorded error number.

line
integer, 0 or the last recorded function line number.

func
string, the last recorded function name

opt
boolean, if %t recorded message is cleared. Default is %t.

Description

Each time an error occurs, the Scilab error handler records it in an internal table (only the last one is retained). The `lasterror` function allows to get the message, the error number, the current function (if any) and the current line number in the current function out of this table.

The reported line number is the *physical* line number where the last error occurred. Note that Scilab versions before 5.0 used to report the *logical* line number of the last error. There is a difference only if the function in error includes *continued lines* (see `dot`) before the point where the error happened.

This function is especially useful while using `errcatch` or `execstr`.

The recorded error message may be retained for a further call to `lasterror` using `lasterror(%f)`.

Examples

```
ierr=execstr('a=zzzzzzz','errcatch')  
if ierr>0 then disp(lasterror()),end
```

See Also

`errcatch`, `execstr`, `error`, `errclear`, `edit_error`

Name

left — (l) left bracket

```
[a11,a12,...;a21,a22,...;...]  
[s1,s2,...]=func(...)
```

Parameters

a11,a12,...

matrix of any compatibles types with compatibles dimensions s1,s2,... : any possible variable name

Description

Left and right brackets are used for vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call

Inside concatenation brackets blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

Note : to avoid confusions it is safer to use comma instead of blank to separate columns.

Within multiple lhs brackets variable names must be separated by comma.

Examples

```
[6.9,9.64; sqrt(-1) 0]  
[1 +%i 2 -%i 3]  
[]  
['this is'; 'a string'; 'vector']  
s=poly(0, 's'); [1/s, 2/s]  
[tf2ss(1/s), tf2ss(2/s)]  
  
[u, s]=schur(rand(3,3))
```

See Also

comma , semicolon

Name

less — ($<$) lower than comparison
great — ($>$) greater than comparison

Description

logical comparison symbol

$<>$
means "different" (same as $\neq$)

$<$
means "lower than"

$>$
means "larger than"

$<=$
means lower than or equal to.

$>=$
means larger than or equal to

See Also

if , comparison , equal

Name

macr2lst — function to list conversion

```
[txt]=macr2lst(function-name)
```

Description

This primitive converts a compiled Scilab function `function-name` into a list which codes the internal representation of the function (reverse polish notation).

The first entry of the list is the function name, the second and third are respectively the vectors of left hand side variables and right hand side variables names. The following entries are either basic operation records either lists with contains the hierarchical control structures like if, for, ...

Basic operation records are described by a character string vector whose first element represents the opcode.

op codes	meaning	parameters
"0"	ignored opcode	none
"1"	No more used	
"2"	variable or function reference	variable name, #rhs, #lhs
"3"	put a string in the stack	the string
"4"	put an empty matrix in the stack	none
"5"	apply an operation	operation code, #rhs, #lhs
"6"	put a number in the stack	the number
"12"	pause command	none
"13"	break command	none
"14"	abort command	none
"15"	end of line mark	none
"17"	quit command	none
"18"	named variable	variable name
"19"	create recursive index structure	path length, number of final indices
"20"	function call	function name, #rhs, #lhs
"23"	create variable from name	variable name
"24"	put a variable with type 0 in the stack	none
"25"	profile record	number of call, time spend
"26"	put a vector of strings in the stack	#rows, #columns, element sequence
"27"	put a builtin reference in the stack	interface number, position in interface, function name
"28"	continue command	none
"29"	assignment	#lhs, display mode, (variable name, #rhs)*
"30"	logical expression short circuit	type, jump size
"31"	comment	the comment

"99"	return command	none
> "100"	builtin call (obsolete)	100*fun, #rhs, #lhs, fin

The fun2string function can be used to generate the initial code.

Examples

```
//DISPLAY
function y=foo(x,flag)
    if flag then
        y=sin(x)
    else
        y=cos(x)
    end
endfunction
L=macr2lst(foo)
fun2string(L)
```

See Also

macrovar , fun2string , macr2tree , tree2code

Name

macr2tree — function to tree conversion

```
t=macr2tree(function-name)
```

Parameters

function-name
a Scilab macro

t
a Scilab "tree"

Description

This primitive converts a compiled Scilab function `function-name` into a tree (imbricated tlists) which codes the internal representation of the function. For use with `tree2code`.

Examples

```
tree=macr2tree(cosh);  
txt=tree2code(tree,%T);  
write(%io(2),txt,'(a)');
```

See Also

`tree2code`

Authors

V.C.

Name

matrices — Scilab object, matrices in Scilab

Description

Matrices are basic objects defined in Scilab. They can be defined as follows:

```
E=[e11,e12,...,e1n;  
   e21,e22,...,e2n;  
   .....  
   em1,em2,...,emn];
```

Entries e_{ij} can be real or complex numbers, polynomials, rationals, strings, booleans.

Vectors are seen as matrices with one row or one column.

Examples

```
E=[1,2;3,4]  
E=[%T,%F;1==1,1~=1]  
s=poly(0,'s');E=[s,s^2;1,1+s]  
E=[1/s,0;s,1/(s+1)]  
E=['A11','A12';'A21','A22']
```

See Also

poly , string , boolean , rational , empty , hypermatrices

Name

`matrix` — reshape a vector or a matrix to a different size matrix

```
y=matrix(v,n,m)
y=matrix(v,[sizes])
```

Parameters

`v`
a vector, a matrix or an hypermatrix

`n,m`
integers

`sizes`
vector of integers

`y`
a vector, a matrix or hypermatrix

Description

For a vector or a matrix with $n \times m$ entries, the command `y=matrix(v,n,m)` or similarly `y=matrix(v,[n,m])` transforms the `v` vector (or matrix) into an $n \times m$ matrix by stacking columnwise the entries of `v`.

if one of the dimension `m` or `n` is equal to -1 it is automatically assigned to the quotient of `size(v, '*')` by the other dimension,

For an hypermatrix such as `prod(size(v))==prod(sizes)`, the command `y=matrix(v,sizes)` (or equivalently `y=matrix(v,n1,n2,...nm)`) transforms `v` into an matrix or hypermatrix by stacking "columnwise" (first dimension is varying first) the entries of `v`. `y=matrix(v,sizes)` results in a regular matrix if `sizes` is a scalar or a 2-vector.

Examples

```
a=[1 2 3;4 5 6]
matrix(a,1,6)
matrix(a,1,-1)
matrix(a,3,2)
```

See Also

`matrices` , `hypermatrices` , `ones` , `zeros` , `rand` , `poly` , `empty`

Name

mode — select a mode in exec file

```
mode(k)
k = mode()
```

Description

Used **exclusively** inside an exec-file or a scilab function `mode(k)` allows to change the information displayed during the execution, depending on the value of `k`:

`k = 0`

The new variable values are displayed if required (see help on semicolon or comma).

`k = -1`

the exec file or scilab function executes silently. (this is the default value for scilab functions)

`k = 2`

default value on scilab prompt.

Please notices that mode does not used at prompt, only in an exec-file or a scilab function.

`k = 1` or `k = 3`

each line of instructions is echoed preceded of the prompt(if possible). The new variable values are displayed if required. This is the default for exec files.

`k = 4` or `k = 7`

The new variable values are displayed if required, each line of instructions is echoed (if possible) and a prompt (`>>`) is issued after each line waiting for a carriage return .

If carriage return follows character "p" the execution is paused (see pause).

`k > 7`

user mode. if you define your own mode, it is recommended to use a value `> 100`.

Line display is disabled for compiled scilab function (see `comp`). By default, Scilab functions are executed using the silent ("`-1`") mode.

functions in a function library are executed using "`-1`" mode.

Examples

```
// copy - paste in scilab

function example_mode( level_mode )
  mode(level_mode)
  a = 3
  disp(mode());
endfunction

mode(2)

example_mode(0)
example_mode(-1)
example_mode(2)
example_mode(1)
example_mode(3)
```

```
example_mode(7)
```

```
mode(2)
```

See Also

[exec](#), [execstr](#), [semicolon](#), [comma](#)

Name

mtlb_mode — switch Matlab like operations

```
mmode=mtlb_mode()  
mtlb_mode(mmode)
```

Parameters

mmode
boolean

Description

Scilab and Matlab additions and subtractions work differently when used with empty matrices:

Scilab

```
a+[] -->a  
a-[] -->a  
[]+a -->a  
[]-a -->-a
```

Matlab

```
a+[] -->[]  
a-[] -->[]  
[]+a -->[]  
[]-a -->[]
```

mtlb_mode(%t) switches to Matlab evaluation mode for additions and subtractions.
mtlb_mode(%f) switches back to Scilab mode.

mtlb_mode() return the current mmode' value

See Also

empty

Name

names — scilab names syntax

Description

Names of variables and functions must begin with a letter or one of the following special characters '%', '_', '#', '!', '\$', '?'.

Next characters may be letters or digits or any special character in '_', '#', '!', '\$', '?'

Names may be as long as you want but only the first 24 characters are taken into account. Upper and lower case letters are different.

Examples

```
//Valid names
%eps
A1=123
#Color=8
My_Special_Color_Table=rand(10,3)
//Non valid names
//1A , b%, .C
```

Name

`newfun` — add a name in the table of functions

```
newfun("function-name", nameptr)
```

Description

Utility function (for experts only). Adds the name "function-name" in the table of functions known to the interpreter. "nameptr" is an integer $1000 * \text{fun} + \text{fin}$ where `fun` and `fin` is the internal coding of the primitive "function-name". This function is useful to associate a primitive to a routine interfaced in "matusr.f" (`fun=14`). Used with `funptr` and `clearfun` one can redefine a primitive by a function with same name.

See Also

`clearfun`

Name

null — delete an element in a list

```
l(i)=null()
```

Description

Deletion of objects inside a list

Examples

```
l=list(1,2,3);  
l(2)=null() // get list(1,3)
```

See Also

list , clear

Name

parents — () left and right parenthesis

```
(expression)
[...] = func(e1, e2, ...)
[x1, x2, ...] = (e1, e2, ...)
x(i, j)
v(i)
[...] = l(i)
```

Parameters

x
matrix of any possible type

v
row or column vector of any possible type

l
list variable

func
any function name

e1, e2, ...
any possible type expression

Description

Left and right parenthesis are used to

* Specify evaluation order within expressions,

* Form right-hand-side functions argument list. Within multiple rhs arguments must be separated by comma.

* Select elements within vectors, matrices and lists. see help on extraction and insertion for more precisions

* `[x1, x2, ...] = (e1, e2, ...)` is equivalent to first performing `%t_1 = e1`, `%t_2 = e2`, ..., and then `x1 = %t_1`, `x2 = %t_2`, ..., where the variables `%t_i`, `i = 1, 2, ...` are invisible to the user.

Examples

```
3^(-1)
x=polynomial(0, "x");
//
(x+10)/2
i3=eye(3,3)
//
a=[1 2 3;4 5 6;7 8 9], a(1,3), a([1 3], :), a(:, 3)
a(:, 3)=[ ]
a(1, $)=33
a(2, [$ $-1])
a(:, $+1)=[10;11;12]
//
```

```
w=ssrand(2,2,2);ssprint(w)
ssprint(w(:,1))
ss2tf(w(:,1))
//
l=list(1,2,3,4)
[a,b,c,d]=l(:)
l($+1)='new'
//
v=%t([1 1 1 1 1])
//
[x,y,z]=(1,2,3)
```

See Also

colon , comma , brackets , list , extraction , insertion

Name

pause — pause mode, invoke keyboard

Description

Switch to the **pause** mode; inserted in the code of a function, **pause** interrupts the execution of the function: one receives a prompt symbol which indicates the level of the **pause** (e.g. -1->). The user is then in a new workspace in which all the lower-level variables (and in particular all the variable of the function) are available. To return to the calling workspace enter "**return**"

In this mode, `[...]=return(...)` returns the variables of the argument `(...)` to the calling workspace with names in the output `[...]`. Otherwise, the lower-level variables are protected and cannot be modified.

The **pause** is extremely useful for debugging purposes.

This mode is killed by the command "**abort**".

See Also

halt , return , abort , quit , whereami , where , sleep

Name

percent — (%) special character

Description

Some predefined variables names begin with %, such as %i (for `sqrt(-1)`), %inf (for Infinity), %pi (for 3.14...), %T (for the boolean constant "true"),...

In addition, functions whose names begin with % are special : they are used for primitives and operators overloading (see [over loading](#)).

For example the function %rmr performs the multiplication (m) operation $x*y$ for x and y rational matrices (r). The coding conventions are given by the readme file in directory SCIDIR/macros/percent.

Examples

```
x1=tlist('x',1,2);
x2=tlist('x',2,3);
deff('x=%mx(x1,x2)','x=list('x',x1(2)*x2(2),x2(3)*x2(3))');
x1*x2
```

See Also

[overloading](#)

Name

perl — Call Perl script using appropriate operating system executable

```
perl('perlfile')  
perl('perlfile',arg1,arg2,...)  
result = perl(...)
```

Description

perl('perlfile') calls the Perl script perlfile, using the appropriate operating system Perl executable.

perl('perlfile',arg1,arg2,...) calls the Perl script perlfile, using the appropriate operating system Perl executable, and passes the arguments arg1, arg2, and so on, to perlfile.

result = perl(...) returns the results of attempted Perl call to result.

See Also

unix

Authors

A.C

Name

plus — (+) addition operator

```
X+Y  
str1+str2
```

Parameters

X,Y

scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

str1,str2

a character string, a vector or a matrix of character strings

Description

Addition.

For numeric operands addition as its usual meaning. If one of the operands is a matrix and the other one a scalar the scalar is added to each matrix entries. if one of the operands is an empty matrix the other operand is returned (this default behavior can be modified by the function `mtlb_mode`).

For character strings + means concatenation.

Addition may also be defined for other data types through "soft-coded" operations (see `overloading`).

Examples

```
[1,2]+1  
[]+2  
s=poly(0,"s");  
s+2  
1/s+2  
"cat"+"enate"
```

See Also

`addf` , `mtlb_mode` , `overloading`

Name

poly — polynomial definition

```
p=poly(a,vname, ["flag"])
```

Parameters

a

matrix or real number

vname

String, the symbolic variable name. If the string have more than 4 characters only the first 4 are taken into account.

"flag"

string ("roots", "coeff"), default value is "roots".

Description

If a is a matrix,

p is the characteristic polynomial i.e. $\det(x \cdot \text{eye}() - a)$, x being the symbolic variable.

If v is a vector,

- `poly(v, "x", ["roots"])` is the polynomial with `roots` the entries of `v` and `"x"` as formal variable. (In this case, `roots` and `poly` are inverse functions). Note that Infinite roots gives zero highest degree coefficients.
- `poly(v, "x", "coeff")` creates the polynomial with symbol `"x"` and with coefficients the entries of `v` (`v(1)` is the constant term of the polynomial). (Here `poly` and `coeff` are inverse functions).

`s=poly(0, "s")` is the seed for defining polynomials with symbol `"s"`.

Examples

```
s=poly(0, "s"); p=1+s+2*s^2;  
A=rand(2,2); poly(A, "x")  
//rational fractions  
h=(1+2*s)/poly(1:4, 's', 'c')
```

See Also

`coeff` , `roots` , `varn` , `horner` , `derivat` , `matrices` , `rational`

Name

power — power operation (^,.)

```
t=A^b
t=A**b
t=A.^b
```

Parameters

A,t
scalar, polynomial or rational matrix.

b
a scalar, a vector or a scalar matrix.

Description

- "(A:square)^(b:scalar)" If A is a square matrix and b is a scalar then A^b is the matrix A to the power b.
- "(A:matrix).^(b:scalar)" If b is a scalar and A a matrix then $A.^b$ is the matrix formed by the element of A to the power b (elementwise power). If A is a vector and b is a scalar then A^b and $A.^b$ performs the same operation (i.e elementwise power).
- "(A:scalar).^(b:matrix)" If A is a scalar and b is a matrix (or vector) A^b and $A.^b$ are the matrices (or vectors) formed by $a^{(b(i,j))}$.
- "(A:matrix).^(b:matrix)" If A and b are vectors (matrices) of the same size $A.^b$ is the $A(i)^{b(i)}$ vector ($A(i,j)^{b(i,j)}$ matrix).

Notes:

- For square matrices A^p is computed through successive matrices multiplications if p is a positive integer, and by diagonalization if not.

- ** and ^ operators are synonyms.

Examples

```
A=[1 2;3 4];
A^2.5,
A.^2.5
(1:10)^2
(1:10).^2

s=poly(0,'s')
s^(1:10)
```

See Also

exp

Name

predef — variable protection

```
n=predef()  
oldnew=predef(n)  
oldnew=predef('all')  
oldnew=predef('clear')
```

Description

Utility function used for defining "oldest" variables as "protected". Protected variables cannot be killed. They are not saved by the 'save' command. The "oldest" are those appearing last in the who('get').

predef() gets the number of protected variables

predef('a[ll]') sets all the variables protected, it also return the old and new value of protected variables number.

predef('c[lear]') unprotect all but the last 7 variables, it also return the old and new value of protected variables number.

predef(n) sets the max(n, 7) last defined variables as protected, it also return the old and new value of protected variables number.

Remarks:

Variable ans created automatically when expressions are not assigned is never protected by predef('all').

A number of protected variables are set in the start-up file SCI/etc/scilab.start. User may in particular set its own predefined variables in user's startup files SCIHOME/.scilab and SCI-HOME/scilab.ini

See Also

clear , save , ans , startup

Name

quit — Terminates Scilab or decreases the pause level

```
quit
```

Description

The `quit` command has two different meanings depending on the calling context:

If there is no `pause` active,
then the `quit` command makes Scilab terminate, even if the command is called inside a function.

If there is a `pause` active,
then the `quit` command makes aborts the instructions started at this pause level ando terminates the current pause level.

Examples

```
// quit Scilab
function foo(x),if x then quit,end,endfunction
foo(%t) //quits scilab

//terminate instruction started in a pause context
function foo(x),if x then quit,end,endfunction
pause
foo(%t) //returns at the main prompt level

function foo1(x),
    mprintf('P1\n')
    if x then pause, mprintf('P2\n'),end,
    mprintf('P3\n')
endfunction

foo1(%t) //enter quit at the following prompt
```

See Also

`pause` , `break` , `abort` , `exit`

Name

quote — (') transpose operator, string delimiter

Description

quote (') is used for (Conjugate) Transpose of matrix.

quote (.') is used for (non Conjugate) Transpose of matrix.

Simple (') or double (") quotes are also used to define character strings. (Character strings are defined between two quotes). A Quote within a character string is denoted by two quotes.

Examples

```
[1+%i, 2]'  
[1+%i, 2].'  
x='This is a character string'  
'He said:''Good'''
```

Name

rational — Scilab objects, rational in Scilab

Description

A rational r is a quotient of two polynomials $r=\text{num}/\text{den}$. The internal representation of a rational is a list. $r=\text{tlist}(['r', 'num', 'den', 'dt'], \text{num}, \text{den}, [])$ is the same as $r=\text{num}/\text{den}$. A rational matrix can be defined with the usual syntax e.g. $[r_{11}, r_{12}; r_{21}, r_{22}]$ is a 2x2 matrix where r_{ij} are 1x1 rationals. A rational matrix can also be defined as above as a list $\text{tlist}(['r', 'num', 'den', 'dt'], \text{num}, \text{den}, [])$ with num and den polynomial matrices.

Examples

```
s=poly(0, 's');
w=[1/s, 1/(s+1)]
w'*w
Num=[s, s+2; 1, s]; Den=[s*s, s; s, s*s];
tlist(['r', 'num', 'den', 'dt'], Num, Den, [])
H=Num./Den
syslin('c', Num, Den)
syslin('c', H)
[Num1, Den1]=simp(Num, Den)
```

See Also

poly , syslin , simp

Name

readgateway — get primitives list of a module

```
readgateway(module_name)
primitives = readgateway(module_name);
[primitives,primitivesID] = readgateway(module_name);
[primitives,primitivesID,gatewayID] = readgateway(module_name);
```

Description

get primitives list of a module.

primitives : list of primitives of a module.

primitivesID : list of ID for primitives.

gatewayID : list of ID of gateway associated to a module.

Examples

```
[primitives,primitivesID,gatewayID] = readgateway('core');
primitives(1) // 'debug' primitive
primitivesID(1) // 1 is ID of 'debug' in 'core' gateway
gatewayID(1) // 13 is ID of 'core' gateway in scilab
```

See Also

getmodules

Name

`resume` — return or resume execution and copy some local variables

```
resume
[x1, ..., xn]=resume(a1, ..., an)
```

Parameters

`x`
...

Description

In a function `resume` stops the execution of the function, `[..]=resume(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

In `pause` mode, it allows to return to lower level `[..]=resume(..)` returns to lower level and put the local variables `ai` in calling environnement under names `xi`.

In an `execstr` called by a function `[..]=resume(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

`resume` is equivalent to `return`.

See Also

`abort` , `break`

Name

`return` — return or resume execution and copy some local variables

```
return  
[x1, ..., xn]=return(a1, ..., an)
```

Parameters

`x`
...

Description

In a function `return` stops the execution of the function, `[..]=return(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

In `pause` mode, it allows to return to upper level `[..]=return(..)` returns to upper level and put the local variables `ai` in calling environnement under names `xi`.

In an `execstr` called by a function `[..]=return(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

`resume` is equivalent to `return`.

See Also

`abort` , `break`

Name

sciargs — scilab command line arguments

```
args = sciargs()
```

Description

This function returns a row vector of character strings containing the arguments of the Scilab command line. First `args` entry contains the path of the launched executable file.

This function corresponds to the `getarg` function in C language

See Also

`getenv`

Name

scilab — Major unix script to execute Scilab and miscellaneous tools

```
scilab <Options>
scilab-adv-cli <Options>
scilab-cli <Options>
```

Description

-args Arguments

If this option is present, arguments are passed to Scilab. They can then be got by `sciargs` function. For multi arguments passing use a quoted, blank separated sequence of words like: `Scilab -args 'foo1 foo2'`. Without this option, unknown arguments will not be accepted.

-display Display

For use under Xwindow systems only to set a specific X server display. Default display is `unix:0.0`

`-display` can be abbreviated by `-d`

-debug

Start Scilab under the debugger `gdb` (Unix/linux only).

Define the variable `SCILAB_GDB_OPT` to add custom options to `gdb`.

Advise: use this option on a Scilab source tree.

-debug-kdbg

Start Scilab under `kdbg` (Unix/linux only).

Advise: use this option on a Scilab source tree.

-profiling

Start Scilab under `valgrind` (Unix/linux only).

Define the variable `SCILAB_VALGRIND_OPT` to add custom options to `valgrind` (and override the existing `valgrind` options).

Advise: use this option on a Scilab source tree.

-profiling-visu

Start Scilab under `callgrind` (Unix/linux only).

Define the variable `SCILAB_VALGRIND_OPT` to add custom options to `callgrind` (and override the existing `callgrind` options).

Advise: use this option on a Scilab source tree.

-electric-fence

Start Scilab with the Electric Fence (Unix/linux only).

Advise: use this option on a Scilab source tree.

-e Instruction

If this option is present then Scilab instruction `Instruction` is executed first (just after startup file execution) into Scilab. `-e` and `-f` options are mutually exclusive.

Note that several instructions can be used in with `-e`.

```
scilab-cli -e "a=1+%i; aPlusPi=a+%pi; disp(aPlusPi);exit;" -nb
```

-f file

If this option is present then Scilab script `file` is executed first (just after startup file execution) into Scilab. `-e` and `-f` options are mutually exclusive.

-l lang

If this option is present it fixes the user language. *lang* can be: `ca_ES de_DE en_US es_ES fr_FR ja_JP pt_BR ru_RU zh_CN zh_TW` (from Scilab 5.2).

Other possible `lang` values are `'fr'` for french and `'en'` for english for compatibility reasons. The default language is english. This default value is fixed the `scilab.start` file.

On some systems, locales must be compiled to render correctly UTF-8 fonts.

Scilab can be also called the following way:

```
LANG=ja_JP scilab
# equivalent to
scilab -l ja_JP
```

-mem N

Set the initial stacksize, for use with `-ns` option. Without `-ns` option the initial stacksize is set by `scilab.start` script.

-nb

If this option is present then the scilab welcome banner is not displayed.

-ns

If this option is present the startup file `SCI/etc/scilab.start` and the user startup files `SCIHOME/.scilab`, `SCIHOME/scilab.ini` are not executed.

This option will disable many features in Scilab (Only use if you know what you are doing).

-nouserstartup

If this option is present the user startup files `SCIHOME/.scilab`, `SCIHOME/scilab.ini` are not executed.

-nw

If this option is present, Scilab is started as command line with advanced features (graphics...).

This option may be used with `-f` or `-e` options.

From Scilab 5.2:

- Scilab distribution also provides a dedicated binary which is doing the same as `-nw`: `scilab-adv-cli` (Scilab Advanced Command Line Interpreter).
- pipes are enabled for all operating systems (see the examples for further details).

-nwni / -nogui

If this option is present, Scilab is started as command line without advanced features.

This option may be used with `-f` or `-e` options.

From Scilab 5.2:

- Scilab distribution also provides a dedicated binary which is doing the same as `-nwni`: `scilab-cli` (Scilab Command Line Interpreter).
- pipes are enabled for all operating systems (see the examples for further details).

This mode does not load the Java Virtual Machine (faster to start and uses less memory)

--texmacs

This option is reserved for TeXMacs.

-version

This option prints product version and exits.

Java Virtual Machine options

Starting from Scilab 5.0, the graphical user interface (GUI) and the build documentation are based on Java features. In some cases, it can be important to edit the JVM options (Java Virtual Machine).

These options are available in the *jvm_options.xml* file.

In version 5.0.X and 5.1.X, this file is stored as *SCI/modules/jvm/etc/jvm_options.xml*

In version $\geq 5.2.0$, the file is available in *etc/jvm_options.xml*

By default, the three following options are easily accessible in the configuration file:

-XmxXXXm

This option set the amount of memory available by the Java Virtual Machine. By default, 256M are allocated. If you change this value, check that the value does not exceed the memory available on the system.

-Djava.compiler=JIT

This option with the argument *JIT* enables the Java Just In Time compiler. It is activated by default. *NONE* disables the JIT and decreases dramatically performances.

-verbose:jni / -Xcheck:jni

These options enable more checks and output from the JNI calls. These options are usefull in case of debugging and are disabled by default since they decreases performances.

Many more options are available. They can improve the performances, change look and feel, change memory managements... See: <http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp> or <http://blogs.sun.com/watt/resource/jvm-options-list.html>

Examples

```
# Let's start Scilab in profiling mode without attaching a gdb one a SIGSEGV is
# We are under Bash shell
export SCILAB_VALGRIND_OPT="--deb-attach=no --log-file=myfile.txt"
scilab -profiling
```

```
# Let's start Scilab in debug mode without stopping after each SIGSEGV
# First, we write a small command file
echo "handle SIGSEGV nostop" &gt; debug.txt
# Now set the custom option
# We are under Bash shell
export SCILAB_GDB_OPT="--command=debug.txt"
# Start Scilab in debug mode
scilab -debug
```

```
# Under GNU/Linux, Mac OS X or Unix:
$ echo "disp(%pi)"|scilab-cli
or
$ echo "disp(%pi)"|scilab -nwni
```

```
# Only open the Scilab help window:
$ scilab-adv-cli -e "help()"
or
$ scilab -nw -e "help()"

# Scilab can be used for scripting aspects:
echo "if 1<>2 then exit(99) end"|scilab-cli
echo $?
```

See Also

[exit](#)

Name

select — select keyword

Description

```
select expr,  
    case expr1 then instructions1,  
    case expr2 then instructions2,  
    ...  
    case exprn then instructionsn,  
    [else instructions],  
end
```

Notes:

- The only constraint is that each "then" keyword must be on the same line as corresponding "case" keyword.
- The keyword "then" can be replaced by a carriage return or a comma. `instructions1` are executed if `expr1=expr`, etc.

Warning: the number of characters used to define the body of any conditional instruction (if while for or select/case) must be limited to 16k.

Examples

```
while %t do  
    n=round(10*rand(1,1))  
    select n  
    case 0 then  
        disp(0)  
    case 1 then  
        disp(1)  
    else  
        break  
    end  
end
```

See Also

if , while , for

Name

semicolon (;) — ending expression and row separator

Description

semicolons are used to separate rows in a matrix definition (within brackets).

semicolons may also be used at the end of an instruction (in a file or in Scilab console). In this case it means that the result(s) is(are) not displayed. Conversely use comma (,) to get the display.

Examples

```
a=[1, 2, 3;4, 5, 6];  
a=1;b=1, c=2
```

See Also

comma , brackets

Name

setbpt — set breakpoints

```
setbpt(macroname [, linenumb])
```

Parameters

macroname

string

linenumb

scalar integer or vector of integers

Description

setbpt interactively inserts a breakpoint in the line number `linenumb` (default value is 1) of the function `macroname`.

`linenumb` can be a line or column vector of line numbers, or a single scalar line number.

Line numbers in `linenumb` are *physical* line numbers in function `macroname`. Note that Scilab versions before 5.0 used *logical* line numbers. The difference between physical and logical line numbers is the number of *continued lines* (see dot).

When reaching a breakpoint, Scilab evaluates the specified physical line and stops the execution flow. If the function is not compiled (see `comp`) the line is printed on the screen. Then Scilab goes into a pause mode in which the user can check current values. The pause is exited with `resume` or `abort`.

Redefining the function does not clear the breakpoints, the user must explicitly delete breakpoints using `delbpt`. The maximum number of functions with breakpoints enabled must be less than 100 and the overall maximum number of breakpoints is set to 1000.

Examples

```
setbpt('foo'), setbpt('foo',10), dispbpt()  
delbpt()  
  
setbpt('foo',[1,2,5,6]), dispbpt()
```

See Also

delbpt, dispbpt, pause, resume

Name

sethomedirectory — Set Scilab home directory

```
[home,scilabhome] = sethomedirectory()
```

Description

DEPRECATED : You need to use "SCIHOME" and "home" scilab variables.

Set Scilab home path : "SCIHOME" variable.

On Windows 2k and XP , C:\Documents and Settings\<User>\Scilab\<Scilab-Version>

On Windows Vista , C:\Users\<User>\Scilab\<Scilab-Version>

On Unix, /home/<User>/.Scilab/<Scilab-Version>

Authors

Allan CORNET

Name

slash — (/) right division and feed back

Description

Right division. $x=A \ / \ b$ is the solution of $x*b=A$.

$b/a = (a' \setminus b')'$.

$a ./ b$ is the matrix with entries $a(i,j)/b(i,j)$. If b is scalar (1x1 matrix) this operation is the same as $a./b*\text{ones}(a)$. (Same convention if a is a scalar).

Remark that $123./b$ is interpreted as $(123.)/b$. In this cases dot is part of the number not of the operator.

Backslash stands for left division.

System feed back. $S=G/.K$ evaluates $S=G*(\text{eye}()+K*G)^{-1}$ this operator avoid simplification problem.

Remark that $G/.5$ is interpreted as $G/(.5)$. In such cases dot is part of the number, not of the operator.

Comment `//` comments a line i.e lines which begin by `//` are ignored by the interpreter.

See Also

`inv` , `percent` , `backslash` , `ieee`

Name

stacksize — set scilab stack size

```
stacksize(n)
stacksize('max')
stacksize('min')
sz=stacksize()
```

Parameters

n
integer, the required stack size given in number of double precision words

SZ
2-vector [total used]

Description

Scilab stores "usual" variables in a stack `stk` (for global variables see `gstacksize`).

`stacksize(n)` allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time.

`stacksize('max')` allows the user to increase the size of this stack to the maximum.

`stacksize('min')` allows the user to decrease the size of this stack to the minimum.

This function with the `n` argument can now be used everywhere.

`sz=stacksize()` returns a 2-vector which contains the current total and used stack size.

See Also

`who` , `gstacksize`

Name

star — (*) multiplication operator

Description

Multiplication. Usual meaning. Valid for constant, boolean, polynomial, rational matrices and for `syslin` lists (the meaning is series connection)

Element-wise multiplication is denoted $x \cdot * y$. If x or y is scalar (1x1 matrix) $\cdot *$ is the same as $*$.

Kronecker product is $x \cdot * \cdot y$

$A * \cdot B$ is an operator with no predefined meaning. It may be used to define a new operator (see overloading) with the same precedence as $*$ or $/$.

See Also

slash , backslash , syslin

Name

startup — startup file

Description

The startup file `SCIHOME/.scilab` and `SCIHOME/scilab.ini` in are automatically executed (if present) when Scilab is invoked, in addition with the file `scilab.star` in the Scilab directory (SCI).

Remarks

Last line of startup file must be terminated by a newline to be taken into account.

SCIHOME definition : On Windows : `C:/Documents and Settings/<User>/Scilab/<Scilab-Version>`

or on Vista : `C:/<User>/AppData/Roaming/Scilab/<Scilab-Version>`

On Linux/Unix : `/home/<User>/Scilab/<Scilab-Version>`

See Also

`scilab`

Name

symbols — scilab operator names

Description

Use the following names to get help on a specific symbol.

operator	name in Scilab help
' , " , ' ,	quote
+	plus
-	minus
,,*	star
/, ./, /.	slash
\, \, \.	backslash
.	dot
=, ==	equal
<, >, >=, <=, <>	less
~	tilda
[	left
]	right
()	parents
%	percent
:	colon
,	comma
;	semicolon
^	hat
.^	power
	or_op
&	and_op
.*, ./, \.	kron

Remark

For historical reasons, different symbols may represent the same operator:

{ as the same meaning as [

} as the same meaning as]

@ as the same meaning as ~

` as the same meaning as <

It is highly recommended not to use these features because they will be removed in the future

See Also

overloading

Name

testmatrix — generate special matrices, such as Hilbert, Franck

```
[y]=testmatrix(name,n)
```

Parameters

name
a character string

n
integers, matrix size

y
n x m matrix

Description

Create some particular matrices

testmatrix('magi',n)
returns a magic square of size n .

testmatrix('frk',n)
returns the Franck matrix :

testmatrix('hilb',n)
is the inverse of the nxn Hilbert matrix ($H_{ij} = 1/(i+j-1)$).

Name

then — keyword in if-then-else

Description

Used with `if`.

See Also

`if`

Name

tilda — ($\sim$) logical not

$\sim m$

Parameters

m
boolean matrix

Description

$\sim m$ is the negation of m .

Name

try — beginning of try block in try-catch control instruction
catch — beginning of catch block in try-catch control instruction

```
try
statements
catch
statements
end
```

Description

The try-catch control instruction can be used to manage codes that could possibly generate errors.

When a try-catch control instruction is executed, normally only the statements between the **try** and **catch** keywords are executed. However, if an error occurs during execution of any of these statements, the error is recorded, the remaining statements up to the **catch** keyword are skipped and the statements between the **catch** and **end** keywords are executed using the default error handling mode (see `errcatch`).

The recorded error can be retrieved using the `lasterror` function.

The **catch** statements as well as the **catch** keyword can be omitted if no alternative statements are given.

Note that one can also use the `execstr` function with `'errcatch'` argument for error handling. This can be particularly useful for handling syntactical errors.

Note also that **try-catch** is more or less similar to:

```
if execstr("<try instructions>", "errcatch") <> 0 then
    <catch instructions>
end
```

It uses the same internal mechanism as `errcatch`. It is the reason why `errcatch` or `execstr(..., 'errcatch')` cannot be included in try-catch control structures. This context is detected and produces a specific error message (this error is caught and stored like any other error if it is triggered in the **try** block).

However, try-catch control structures can be nested (see example 2 below).

Examples

```
// example 1
file_path=TMPDIR+'/wrong'
try
    u=mopen(file_path, 'r')
    x=mget(10, 'c', u)
catch
    disp(['file '+file_path+ 'cannot be read',
        'using default values for x'])
    x=1:10
end
[error_message, error_number]=lasterror(%t)
```

```
// example 2 (nested try/catch structures)
function nestedtry(a,b)
disp("START")
mprintf("\ta is %s\t\tb is %s\n",string(a),string(b))
try
    disp("try 1")
    try
        disp("try 2")
        z=a+1; // err when string
    catch
        disp("catch 2")
        t=b+1; // err when string
    end
    disp("after try 2")
catch
    disp("catch 1")
end
disp("after try 1 - THE END")
endfunction
nestedtry(1,1)
nestedtry("a string",1)
nestedtry(1,"a string")
nestedtry("a string","a string")
```

See Also

[error](#), [execstr](#), [if](#), [lasterror](#), [errcatch](#)

Authors

Serge Steer, INRIA

Name

type — Returns the type of a variable

```
[i]=type(x)
```

Parameters

x
Scilab object

i
integer

Description

type(x) returns an integer which is the type of x as following :

- 1 : real or complex constant matrix.
- 2 : polynomial matrix.
- 4 : boolean matrix.
- 5 : sparse matrix.
- 6 : sparse boolean matrix.
- 7 : Matlab sparse matrix.
- 8 : matrix of integers stored on 1 2 or 4 bytes.
- 9 : matrix of graphic handles.
- 10 : matrix of character strings.
- 11 : un-compiled function (Scilab code).
- 13 : compiled function (Scilab code).
- 14 : function library.
- 15 : list.
- 16 : typed list (tlist).
- 17 : matrix oriented typed list (mlist).
- 128 : pointer (See lufact).
- 129 : size implicit polynomial used for indexing.
- 130 : Scilab intrinsic (C or Fortran code).

See Also

typeof

Name

`typename` — associates a name to variable type

```
[types [ [, names]]=typename()  
typename(name, type)
```

Parameters

`types`

integer column vector: the types codes of each defined data types.

`names`

column vector of strings: the names associated to type codes.

`type`

integer: the type code of new data type.

`name`

string: the name associated to the type code

Description

The function and operator overloading make use of a formal name associated to data types to form the name of the overloading function (see [overloading](#)). The `typename` can be used to handle this formal names for hard coded data types (the `tlist` or `mlist` coded data types formal names are defined in an other way, see [overloading](#)).

Called without right hand side argument, `typename` returns information on defined data types.

Called with right hand side argument, `typename` associates a name to a data type code.

`typename(' ', type)` suppress the data type given by its code `type` out of the table of known data types.

Number max. of defined types is 50.

See Also

`type` , `typeof` , [overloading](#) , `tlist` , `mlist`

Name

`user` — interfacing a Fortran or C routine

```
[s_1,s_2,...,s_lhs]=user(e_1,e_2,...,e_rhs)
```

Description

With this command it is possible to use an external program as a Scilab command where $(s_1, s_2, \dots, s_{lhs})$ are the output variables and $(e_1, e_2, \dots, e_{rhs})$ are the input variables. To insert this command in Scilab one has to write a few lines in the `user` fortran subroutine of Scilab. See `intersci` or the Scilab documentation for more information.

See Also

`fort`, `link`

Name

`varn` — symbolic variable of a polynomial

```
[ symb ]=varn(p)
[ pm ]=varn(x, var)
```

Parameters

`p`
polynomial (or matrix polynomial)

`symb`
character string

`x`
polynomial or polynomial matrix

`var`
symbolic variable (character string)

`pm`
matrix or polynomial matrix

Description

`symb=varn(p)` returns in `symb` the symbolic variable of the polynomial `p` (i.e. `varn(poly(0, 'x'))` is `'x'`).

`varn(x, 's')` returns a polynomial matrix with same coefficients as `x` but with `'s'` as symbolic variable (change of variable name).

Examples

```
s=poly(0, 's'); p=[s^2+1, s];
varn(p)
varn(p, 'x')
```

See Also

`horner` , `poly`

Name

ver — Version information for Scilab

```
r = ver()
```

Parameters

r
a matrix of strings

Description

Version information for Scilab.

returns a matrix of string with information about Scilab.

Examples

```
ver
```

Authors

A.C

Name

warning — warning messages

```
warning('string')
warning('off')
warning('on')
mode = warning('query')
```

Description

prints the character string 'string' in a warning message

'on' enable warning messages.

'off' disable warning messages.

'query' get state 'on' or 'off'.

Examples

```
warning('on')
warning('this is a warning')
warning('off')
warning('this is a warning')
warning('query')
warning('on')
```

See Also

[error](#)

Name

what — list the Scilab primitives

```
what()  
[primitives,commands]=what();
```

Description

List of low level primitives and commands.

Authors

A.C

Name

where — get current instruction calling tree

```
[linenum,mac]=where()
```

Parameters

linenum

column vector of integer

mac

column vector of strings

Description

returns `linenum` and `mac` such as current instruction has been called by the `linenum(1)` line of function `mac(1)`, `mac(1)` has been called by the `linenum(2)` line of function `mac(2)` and so on

`mac(i)` is in general the name of a function but it may also be "exec" or "execstr" if instruction lies in an `exec` file or an `execstr` instruction

See Also

`whereami` , `pause`

Name

whereami — display current instruction calling tree

```
whereami()
```

Description

Displays calling tree to instruction which contain whereami(). May be used within pause levels.

Examples

```
deff('y=test(a)', ['y=sin(a)+1';  
                  'y=t1(y)';  
                  'y=y+1'])  
deff('y=t1(y)', ['y=y^2'; 'whereami()'])  
test(1)
```

See Also

where , pause , errcatch

Name

while — while keyword

Description

while clause. Must be terminated by "end"

```
while expr ,instructions,...[,else instructions], end
```

```
while expr do instructions,...[,else instructions], end
```

```
while expr then instructions,...[,else instructions], end
```

Notes:

- The only constraint is that each `then` or `do` keyword must be on the same line as `while` keyword.
- Keywords `then` or `do` can be replaced by a carriage return or a comma. For compatibility with Matlab it is also possible, but not recommended, to put a space between the end of the expression and the beginning of the first instruction.
- The optional `,else instructions` construction allows to give instructions which are executed when `expr` expression becomes false.

Warning: the number of characters used to define the body of any conditional instruction (if while for or select/case) must be limited to 16k.

Examples

```
e=1; a=1; k=1;
while norm(a-(a+e),1) > %eps, e=e/2; k=k+1; end
e, k
```

See Also

for , select , break , return , pause

Name

who — listing of variables

```
who
who()
names=who('local')
[names,mem]=who('local')
names=who('global')
[names,mem]=who('global')
who('sorted')
names=who('local','sorted')
[names,mem]=who('local','sorted')
names=who('global','sorted')
[names,mem]=who('global','sorted')
```

Description

who displays current variable names.

who('local') or who('get') Returns current variable names and memory used in double precision words.

who('global') returns global variable names and memory used in double precision words.

who('sorted') displays in alphabetical order all variables.

See Also

whos , who_user

Name

who_user — listing of user's variables

```
who_user()
```

Description

who_user displays user's variable names.

See Also

whos , who

Name

whos — listing of variables in long form

```
whos()  
whos -type typ  
whos -name nam
```

Parameters

typ
name of selected variable type (see typeof)

nam
first characters of selected names

Description

whos () displays all current variable names, types and memory used.

whos -type typ displays all current variables with specified type.

whos -name nam displays all current variables whose names begin with nam.

Note : If a variable is global, a * appears ahead of his type.

Examples

```
lines(0)  
whos()  
whos -type boolean  
whos -name %
```

See Also

who , who_user , typeof

Name

with_atlas — Checks if Scilab has been built with Atlas Library

```
r=with_atlas()
```

Parameters

r
a boolean

Description

Returns %t if Scilab as been built with Atlas Library or %f if not.

Name

with_gtk — Checks if Scilab has been built with the "GIMP Toolkit" library

```
r=with_gtk()
```

Parameters

r
a boolean

Description

Returns always %f gtk library is not supported by Scilab 5 and more.

Name

with_javasci — Checks if Scilab has been built with the java interface

```
r=with_javasci()
```

Parameters

r
a boolean

Description

Returns %t if Scilab as been built with the java interface or %f if not.

Name

`with_macros_source` — Checks if macros source are installed

```
r=with_macros_source()
```

Parameters

`r`
a boolean

Description

Returns %t if macros source are installed or %f if not.

Name

`with_module` — Checks if a Scilab module is installed

```
r=with_module(module_name)
```

Parameters

`r`

a boolean

`module_name`

a string. example : 'core'

Description

Returns %t Checks if a Scilab module is installed.

See Also

`getmodules`

Authors

A.C

Name

with_pvm — Checks if Scilab has been built with the "Parallel Virtual Machine" interface

```
r=with_pvm()
```

Parameters

r
a boolean

Description

Returns %t if Scilab as been built with the "Parallel Virtual Machine" interface or %f if not.

Name

`with_texmacs` — Checks if Scilab has been called by texmacs

```
r=with_texmacs()
```

Parameters

`r`
a boolean

Description

Returns %t if Scilab as been been called by TeXmacs

Name

with_tk — Checks if Scilab has been built with TCL/TK

```
r=with_tk()
```

Parameters

r
a boolean

Description

Returns %t if Scilab as been built with TCL/TK interface or %f if not.

Part II. Differential Equations, Integration

Name

bvode — boundary value problems for ODE using collocation method
bvodeS — Simplified call to bvode

```
zu=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,l_tol,tol,fixpnt,fsub,dfsub,gsub,dgsub)
zu=bvodeS(xpoints,m,N,x_low,x_up,fsub,gsub,zeta, <optional_args>)
```

Parameters

zu
a column vector of size M. The solution of the ode evaluated on the mesh given by points. It contains $z(u(x))$ for each requested points.

xpoints
an array which gives the points for which we want to observe the solution.

N
a scalar with integer value, number of differential equations ($N \leq 20$).

m
a vector of size N with integer elements. It is the vector of order of each differential equation: $m(i)$ gives the order of the i-th differential equation. In the following, M will represent the sum of the elements of m.

x_low
a scalar: left end of interval

x_up
a scalar: right end of interval

zeta
a vector of size M, $zeta(j)$ gives j-th side condition point (boundary point). One must have $x_{low} \leq zeta(j) \leq zeta(j+1) \leq x_{up}$

All side condition points must be mesh points in all meshes used, see description of **ipar(11)** and **fixpnt** below.

ipar
an array with 11 integer elements:

[nonlin, collpnt, subint, ntol, ndimf, ndimi, iprint, iread, iguess, rstart, nfxpnt]

nonlin: ipar(1)
0 if the problem is linear, 1 if the problem is nonlinear

collpnt: ipar(2)
Gives the number of collocation points per subinterval where $\max(m(j)) \leq \text{collpnt} \leq 7$

if $\text{ipar}(2)=0$ then collpnt is set to $\max(\max(m(j))+1, 5-\max(m(j)))$

subint: ipar(3)
Gives the number of subintervals in the initial mesh. if $\text{ipar}(3) = 0$ then bvode arbitrarily sets subint = 5.

ntol: ipar(4)
Gives the number of solution and derivative tolerances. We require $0 < \text{ntol} \leq M$. **ipar(4)** must be set to the dimension of the **tol** argument or to 0. In the latter case the actual value will automatically be set to `size(tol, '*')`.

ndimf: ipar(5)

Gives the dimension of `fspace` (a real work array). its value provides a constraint on `nmax` the maximum number of subintervals.

The ipar(5) value must respect the constraint $\text{ipar}(5) \geq n_{\max} * \text{nsizf}$ where

$\text{nsizf} = 4 + 3 * M + (5 + \text{collpnt} * N) * (\text{collpnt} * N + M) + (2 * M - n_{\text{rec}}) * 2 * M$ (`nrec` is the number of right end boundary conditions).

ndimi: ipar(6)

Gives the dimension of `ispace` (an integer work array). its value provides a constraint on `nmax`, the maximum number of subintervals.

The ipar(6) value must respect the constraint $\text{ipar}(6) \geq n_{\max} * \text{nsizei}$ where

$\text{nsizei} = 3 + \text{collpnt} * N + M$.

iprint: ipar(7)

output control, make take the following values:

- 1
for full diagnostic printout
- 0
for selected printout
- 1
for no printout

iread: ipar(8)

- = 0
causes bode to generate a uniform initial mesh.
- = xx
Other values are not implemented yet in Scilab
- = 1
if the initial mesh is provided by the user. it is defined in `fspace` as follows: the mesh will occupy `fspace(1), ..., fspace(n+1)`. the user needs to supply only the interior mesh points `fspace(j) = x(j)`, $j = 2, \dots, n$.
- = 2 if the initial mesh is supplied by the user
as with `ipar(8)=1`, and in addition no adaptive mesh selection is to be done.

iguess: ipar(9)

- = 0
if no initial guess for the solution is provided.
- = 1
if an initial guess is provided by the user through the argument `guess`.
- = 2
if an initial mesh and approximate solution coefficients are provided by the user in `fspace`. (the former and new mesh are the same).
- = 3
if a former mesh and approximate solution coefficients are provided by the user in `fspace`, and the new mesh is to be taken twice as coarse; i.e., every second point from the former mesh.

= 4

if in addition to a former initial mesh and approximate solution coefficients, a new mesh is provided in fspace as well. (see description of output for further details on iguess = 2, 3, and 4.)

ireg: ipar(10)

= 0

if the problem is regular

= 1

if the first relax factor is equal to ireg, and the nonlinear iteration does not rely on past coverage (use for an extra sensitive nonlinear problem only).

= 2

if we are to return immediately upon (a) two successive nonconvergences, or (b) after obtaining error estimate for the first time.

nfxpnt: ipar(11)

Gives the number of fixed points in the mesh other than x_{low} and x_{up} (the dimension of fixpnt). ipar(11) must be set to the dimension of the fixpnt argument or to 0. In the latter case the actual value will automatically be set to size(fixpnt, '*').

ltol

an array of dimension ntol=ipar(4). ltol(j) = l specifies that the j-th tolerance in the tol array controls the error in the l-th component of Z . It is also required that:

$$1 \leq ltol(1) < ltol(2) < \dots < ltol(ntol) \leq M$$

tol

an array of dimension ntol=ipar(4).

tol(j) is the error tolerance on the ltol(j) - th component of Z . Thus, the code attempts to satisfy

$$((z(v) - z(u))_{ltol(j)}) \leq tol(j) \cdot ((z(u))_{ltol(j)}) + tol(j), \text{ for } j = 1:ntol$$

on each subinterval

if V is the approximate solution vector and u the exact solution (unknown).

fixpnt

an array of dimension nfxpnt=ipar(11). it contains the points, other than x_{low} and x_{up} , which are to be included in every mesh. The code requires that all side condition points other than x_{low} and x_{up} (see description of zeta) be included as fixed points in fixpnt.

fsub

an external used to evaluate the column vector $f =$

$$[f_1(x, z(u(x))), f_2(x, z(u(x))), \dots, f_N(x, z(u(x)))] \text{ for any } x \text{ such as } x_{low} \leq x \leq x_{up} \text{ and for any } z = z(u(x)) \text{ (see description below)}$$

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine fsub(x, zu, f)
double precision zu(*), f(*), x
```

- In C the function prototype must be

```
void fsub(double *x, double *zu, double *f)
```

- And in Scilab

```
function f=fsub(x,zu,parameters)
```

dfsub

an external used to evaluate the Jacobian of $f(x, z(u))$ at a point x . Where $z(u(x))$ is defined as for `fsub` and the (N) by (M) array df should be filled by the partial derivatives of f :

$$df(i,j) = \frac{\partial f_i}{\partial z_j} \text{ for } \begin{cases} i = 1:N \\ j = 1:M \end{cases}$$

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine dfsub(x,zu,df)
double precision zu(*), df(*),x
```

- In C the function prototype must be

```
void dfsub(double *x, double *zu, double *df)
```

- And in Scilab

```
function df=dfsub(x,zu,parameters)
```

gsub

an external used to evaluate $g_i(\zeta_i, z(u(\zeta_i)))$ given $z = z(u(\zeta_i))$ $z = \text{zetaeta}(i)$ for $1 \leq i \leq M$.

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine gsub(i,zu,g)
double precision zu(*), g(*)
integer i
```

- In C the function prototype must be

```
void gsub(int *i, double *zu, double *g)
```

- And in Scilab

```
function g=gsub(i,zu,parameters)
```

Note that in contrast to f in `fsub`, here only one value per call is returned in `g`.

`dgsub`

an external used to evaluate the i -th row of the Jacobian of $g(x, u(x))$. Where $z(u)$ is as for `fsub`, i as for `gsub` and the M -vector `dg` should be filled with the partial derivatives of g , viz, for a particular call one calculates

$$dg(i,j) = \frac{\partial g_i}{\partial z_j} \text{ for } \begin{cases} i = 1:M \\ j = 1:M \end{cases}$$

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine dgsb(i,zu,dg)
double precision zu(*), dg(*)
```

- In C the function prototype must be

```
void dgsb(int *i, double *zu, double *dg)
```

- And in Scilab

```
function dg=dgsb(i,zu,parameters)
```

`guess`

An external used to evaluate the initial approximation for $z(u(x))$ and `dmval(u(x))` the vector of the m_j -th derivatives of $u(x)$. Note that this subroutine is used only if `ipar(9) = 1`, and then all M components of `zu` and N components of `dmval` should be computed for any x such as $x_{\text{low}} \leq x \leq x_{\text{up}}$.

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine guess(x,zu,dmval)
double precision x,z(*), dmval(*)
```

- In C the function prototype must be

```
void fsub(double *x, double *zu, double *dmval)
```

- And in Scilab

```
function [dmval,zu]=fsub(x,parameters)
```

<optional_args>

It should be either:

- any left part of the ordered sequence of values: guess, dfsub, dgsub, fixpnt, ndimf, ndimi, ltol, tol, ntol, nonlin, collpnt, subint, iprint, ireg, ifail
- or any sequence of arg_name=argvalue with arg_name in: guess, dfsub, dgsub, fixpnt, ndimf, ndimi, ltol, tol, ntol, nonlin, collpnt, subint, iprint, ireg, ifail

Where all these arguments excepted ifail are described above. ifail can be used to display the bode call corresponding to the selected optional arguments. If guess is given iguess is set to 1

Description

These functions solves a multi-point boundary value problem for a mixed order system of ode-s given by

$$\begin{cases} u_i^{(m_i)} = f_i(x, z(u(x))) & i = 1:N \\ g_j(\zeta_j, z(u(\zeta_j))) = 0, j = 1:M \end{cases}$$

Where

$$\begin{cases} M = \sum_{i=1}^N m_i \\ z(u(x)) = [u_1(x); u_1^1(x); \dots; u_1^{(m_1-1)}(x); \dots; u_N(x); u_N^1(x); \dots; u_N^{(m_N-1)}(x)] \\ x_l \leq \zeta_1 \leq \dots \leq \zeta_M \leq x_u \end{cases}$$

The argument ZU used by the external functions and returned by bode is the column vector formed by the components of z(u(x)) for a given x.

The method used to approximate the solution u is collocation at gaussian points, requiring m(i)-1 continuous derivatives in the i-th component, i = 1:N. here, k is the number of collocation points (stages) per subinterval and is chosen such that k .ge. max m(i). a runge-kutta-monomial solution representation is utilized.

Examples

The first two problems below are taken from the paper [1] of the Bibliography.

- **The problem 1** describes a uniformly loaded beam of variable stiffness, simply supported at both ends.

It may be defined as follows:

Solve the fourth order differential equation:

$$\frac{d^4}{dx^4} u(x) = \frac{1 - 6x^2 \cdot \frac{d^3}{dx^3} u(x) - 6x \cdot \frac{d^2}{dx^2} u(x)}{x^3}$$

Subjected to the boundary conditions:

$$\begin{cases} u(1) = 0 \\ \frac{d^2}{dx^2} u(x)(1) = 0 \\ u(2) = 0 \\ \frac{d^2}{dx^2} u(x)(2) = 0 \end{cases}$$

The exact solution of this problem is known to be:

$$u(x) = \frac{1}{4}(10\log(2) - 3)(1 - x) + \frac{1}{2}\left[\frac{1}{x} + (3 + x)\log(x) - x\right]$$

```
N=1; // just one differential equation
m=4; // a fourth order differential equation
M=sum(m);

x_low=1; x_up=2; // the x limits
zeta=[x_low, x_low, x_up, x_up]; // two constraints (on the value of u and its second derivative)

// The external functions
// These functions are called by the solver with zu=[u(x); u'(x); u''(x); u'''(x)]

// - The function which computes the right hand side of the differential equation
function f=fsub(x, zu), f=(1-6*x^2*z(4)-6*x*z(3))/x^3, endfunction

// - The function which computes the derivative of fsub with respect to zu
function df=dfsub(x, zu), df=[0, 0, -6/x^2, -6/x], endfunction

// - The function which computes the ith constraint for a given i
function g=gsub(i, zu),
    select i
    case 1 then // x=zeta(1)=1
        g=zu(1) // u(1)=0
    case 2 then // x=zeta(2)=1
        g=zu(3) // u''(1)=0
    case 3 then // x=zeta(3)=2
        g=zu(1) // u(2)=0
    case 4 then // x=zeta(4)=2
        g=zu(3) // u''(2)=0
    end
```

```

endfunction

// - The function which computes the derivative of gsub with respect to z
function dg=dgsub(i,z)
    select i
    case 1 then //x=zeta(1)=1
        dg=[1,0,0,0]
    case 2 then //x=zeta(2)=1
        dg=[0,0,1,0]
    case 3 then //x=zeta(3)=2
        dg=[1,0,0,0]
    case 4 then //x=zeta(4)=2
        dg=[0,0,1,0]
    end
endfunction

// - The function which computes the initial guess, unused here
function [zu,mpar]=guess(x),zu=0;mpar=0,endfunction

//define the function which computes the exact value of u for a given x ( for
function zu=trusol(x)
    zu=0*ones(4,1)
    zu(1) = 0.25*(10*log(2)-3)*(1-x) + 0.5 *( 1/x + (3+x)*log(x) - x)
    zu(2) = -0.25*(10*log(2)-3) + 0.5 *(-1/x^2 + (3+x)/x + log(x) - 1)
    zu(3) = 0.5*( 2/x^3 + 1/x - 3/x^2)
    zu(4) = 0.5*(-6/x^4 - 1/x/x + 6/x^3)
endfunction

fixpnt=[ ];//All boundary conditions are located at x_low and x_up

//      nonlin collpnt n ntol ndimf ndimi iprint iread iguess rstart nfxpnt
ipar=[0      0      1 2      2000      200      1      0      0      0      0      ]

ltol=[1,3];//set tolerance control on zu(1) and zu(3)
tol=[1.e-11,1.e-11];//set tolreance values for these two controls
xpoints=x_low:0.01:x_up;

zu=bvide(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub,dfsub,gsub,dgsub,guess)
//check the constraints
zu([1,3],[1 $]) //should be zero
plot(xpoints,zu(1,:)) // the evolution of the solution u
zu1=[];for x=xpoints,zu1=[zu1,trusol(x)]; end;
norm(zu-zu1)

```

- Same problem using `bvideS` and an initial guess

```

function [z,lhS]=zstart(x)
    z=zeros(5,1);z(5)=1;
    lhS=[0;1];
endfunction
zu=bvide(xpoints,N,m,x_low,x_up,zeta,ltol=ltol,tol=tol,guess=zstart)

```

- **The problem 2** describes the small finite deformation of a thin shallow spherical cap of constant thickness subject to a quadratically varying axisymmetric external pressure distribution. Here phi

is the meridian angle change of the deformed shell and ψ is a stress function. For $\epsilon\mu=1e-3$ two different solutions may found depending on the starting point

$$\begin{cases} \epsilon^4 \left(\varphi'' + \frac{\varphi'}{x} - \frac{\varphi}{x^2} \right) + \psi \left(1 - \frac{\varphi}{x} \right) - \varphi + \gamma x \left(1 - \frac{x^2}{2} \right) = 0 \\ \mu \left(\psi'' + \frac{\psi'}{x} - \frac{\psi}{x^2} \right) - \varphi \left(1 - \frac{\varphi}{2x} \right) = 0 \\ 0 < x < 1 \end{cases}$$

Subject to the boundary conditions

$$\begin{cases} \varphi = 0 \\ x\psi' - 0.3\psi + 0.7x = 0 \end{cases}$$

for $x=0$ and $x=1$

```
N=2; // two differential equations
m=[2 2]; // each differential equation is of second order
M=sum(m);

x_low=0; x_up=1; // the x limits
zeta=[x_low, x_low, x_up, x_up]; // two constraints on each bound.

// The external functions
// These functions are called by the solver with zu=[u1(x); u1'(x); u2(x); u2'(x)]

// - The function which computes the right hand side of the differential equations
function f=fsub2(x, zu, eps, dm, eps4mu, gam, xt),
    f=[zu(1)/x^2 - zu(2)/x + (zu(1) - zu(3)*(1 - zu(1)/x) - gam*x*(1 - x^2/2))/eps4mu // phi''
        zu(3)/x^2 - zu(4)/x + zu(1)*(1 - zu(1)/(2*x))/dm]; // psi''
endfunction

// - The function which computes the derivative of fsub with respect to zu
function df=dfsub2(x, zu, eps, dm, eps4mu, gam, xt),
    df=[1/x^2 + (1 + zu(3)/x)/eps4mu, -1/x, -(1 - zu(1)/x)/eps4mu, 0
        (1 - zu(1)/x)/dm, 0, 1/x^2, -1/x];
endfunction

// - The function which computes the ith constraint for a given i
function g=gsub2(i, zu),
    select i
    case 1 then // x=zeta(1)=0
        g=zu(1) // u(0)=0
    case 2 then // x=zeta(2)=0
        g=-0.3*zu(3) // x*psi' - 0.3*psi + 0.7x=0
    case 3 then // x=zeta(3)=1
        g=zu(1) // u(1)=0
    case 4 then // x=zeta(4)=1
        g=1*zu(4) - 0.3*zu(3) + 0.7*1 // x*psi' - 0.3*psi + 0.7x=0
    end
endfunction

// - The function which computes the derivative of gsub with respect to z
function dg=dgsub2(i, z)
    select i
```

```

case 1 then //x=zeta(1)=1
    dg=[1,0,0,0]
case 2 then //x=zeta(2)=1
    dg=[0,0,-0.3,0]
case 3 then //x=zeta(3)=2
    dg=[1,0,0,0]
case 4 then //x=zeta(4)=2
    dg=[0,0,-0.3,1]
end
endfunction

gam=1.1
eps=1d-3
dmu=eps
eps4mu=eps^4/dmu
xt=sqrt(2*(gam-1)/gam)

fixpnt=[ ];//All boundary conditions are located at x_low and x_up
collpnt=4;
nsizef=4+3*M+(5+collpnt*N)*(collpnt*N+M)+(2*M-2)*2*M ;
nsizei=3 + collpnt*N+M;;
nmax=200;
//      nonlin  collpnt n  ntol ndimf          ndimi          iprint iread iguess rs
ipar=[1          k          10 4      nmax*nsizef  nmax*nsizei  -1      0      0

ltol=1:4;//set tolerance control on zu(1) zu(2 ) zu(3) and zu(4)
tol=[1.e-5,1.e-5,1.e-5,1.e-5];//set tolreance values for these four controls
xpoints=x_low:0.01:x_up;

zu=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub2,dfsub2,gsub2,dgsub2,guess2);
scf(1);clf();plot(xpoints,zu([1 3],:)) // the evolution of the solution phi and

//using an initial guess
// - The function which computes the initial guess, unused here
function [zu,dmval]=guess2(x,gam),
    cons=gam*x*(1-x^2/2)
    dcons=gam*(1-3*x^2/2)
    d2cons=-3*gam*x
    dmval=zeros(2,1)
    if x>xt then
        zu=[0 0 -cons -dcons]
        dmval(2)=-d2cons
    else
        zu=[2*x;2;-2*x+cons;-2*dcons]
        dmval(2)=d2cons
    end
endfunction
ipar(9)=1;//iguess

zu2=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub2,dfsub2,gsub2,dgsub2,guess2);
scf(2);clf();plot(xpoints,zu2([1 3],:)) // the evolution of the solution phi and

```

- An eigenvalue problem:

```

// y'(x)=-la*y(x)
// BV: y(0)=y'(0); y(1)=0
// Eigenfunctions and eigenvalues are y(x,n)=sin(s(n)*(1-x)), la(n)=s(n)^2,
// where s(n) are the zeros of f(s,n)=s+atan(s)-(n+1)*pi, n=0,1,2,...
// To get a third boundary condition, we choose y(0)=1
// (With y(x) also c*y(x) is a solution for each constant c.)
// We solve the following ode system:
// y'=-la*y
// la'=0
// BV: y(0)=y'(0), y(0)=1; y(1)=0
// z=[y(x) ; y'(x) ; la]

function rhs=fsub(x,z)
    rhs=[-z(3)*z(1);0]
endfunction

function g=gsub(i,z)
    g=[z(1)-z(2) z(1)-1 z(1)]
    g=g(i)
endfunction

// The following start function is good for the first 8 eigenfunctions.
function [z, lhs]=ystart(x,z,la0)
    z=[1;0;la0]
    lhs=[0;0]
endfunction

a=0;b=1;
m=[2;1];
n=2;
zeta=[a a b];
N=101;
x=linspace(a,b,N)';

// We have s(n)-(n+1/2)*pi -> 0 for n to infinity.
la0=input('n-th eigenvalue: n= ?');la0=(%pi/2+la0*%pi)^2;

z=bodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,la0));

clf()
plot(x,[z(1,:)' z(2,:)]')
xtitle(['Startvalue = '+string(la0);'Eigenvalue = '+string(z(3,1))'],'x',' ')
legend(['y(x)';'y'(x)'])

```

- A boundary value problem with more than one solution.

```

// DE: y'(x)=-exp(y(x))
// BV: y(0)=0; y(1)=0
// This boundary value problem has more than one solution.
// It is demonstrated how to find two of them with the help of
// some preinformation of the solutions y(x) to build the function ystart.
// z=[y(x);y'(x)]

a=0;b=1;m=2;n=1;
zeta=[a b];
N=101;

```

```

tol=1e-8*[1 1];
x=linspace(a,b,N);

function rhs=fsub(x,z),rhs=-exp(z(1));endfunction

function g=gsub(i,z)
    g=[z(1) z(1)]
    g=g(i)
endfunction

function [z, lhs]=ystart(x,z,M)
    //z=[4*x*(1-x)*M ; 4*(1-2*x)*M]
    z=[M;0]
    //lhs=[-exp(4*x*(1-x)*M)]
    lhs=0
endfunction

for M=[1 4]
    if M==1
        z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
    else
        z1=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
    end
end

// Integrating the ode yield e.g. the two solutions yex and yex1.

function y=f(c),y=c.*(1-tanh(sqrt(c)/4).^2)-2;endfunction
c=fsolve(2,f);

function y=yex(x,c)
    y=log(c/2*(1-tanh(sqrt(c)*(1/4-x/2)).^2))
endfunction

function y=f1(c1), y=2*c1^2+tanh(1/4/c1)^2-1;endfunction
c1=fsolve(0.1,f1);

function y=yex1(x,c1)
    y=log((1-tanh((2*x-1)/4/c1).^2)/2/c1/c1)
endfunction

disp(norm(z(1,:)-yex(x)), 'norm(yex(x)-z(1,:))= ')
disp(norm(z1(1,:)-yex1(x)), 'norm(yex1(x)-z1(1,:))= ')
clf();
subplot(2,1,1)
plot2d(x,z(1,:),style=[5])
xtitle('Two different solutions','x',' ')
subplot(2,1,2)
plot2d(x,z1(1,:),style=[5])
xtitle(' ','x',' ')

```

- A multi-point boundary value problem.

```

// DE y'''(x)=1
// z=[y(x);y'(x);y''(x)]
// BV: y(-1)=2 y(1)=2

```

```
// Side condition: y(0)=1

a=-1;b=1;c=0;
// The side condition point c must be included in the array fixpnt.
n=1;
m=[3];

function rhs=fsub(x,z)
    rhs=1
endfunction

function g=gsub(i,z)
    g=[z(1)-2 z(1)-1 z(1)-2]
    g=g(i)
endfunction

N=10;
zeta=[a c b];
x=linspace(a,b,N);

z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,fixpnt=c);

function y=yex(x)
y=x.^3/6+x.^2-x./6+1
endfunction

disp(norm(yex(x)-z(1,:)), 'norm(yex(x)-z(1,:))= ')
```

See Also

[link](#), [external](#), [ode](#), [dassl](#)

Used Functions

This function is based on the Fortran routine `colnew` developed by

U. Ascher, Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1W5, Canada

G. Bader, institut f. Angewandte mathematik university of Heidelberg; im Neuenheimer feld 294d-6900 Heidelberg 1

Authors

This help is based on the original `colnew.f` comments, adapted to Scilab by J.P Chancelier ENPC, on the `bvodeS` help page due to Rainer von Seggern, both merged and revised by S. Steer INRIA

Bibliography

1. U. Ascher, J. Christiansen and R.D. Russell, collocation software for boundary-value ODEs, *acm trans. math software* 7 (1981), 209-222. this paper contains EXAMPLES where use of the code is demonstrated.
2. G. Bader and U. Ascher, a new basis implementation for a mixed order boundary value ode solver, *siam j. scient. stat. comput.* (1987).
3. U. Ascher, J. Christiansen and R.D. Russell, a collocation solver for mixed order systems of boundary value problems, *math. comp.* 33 (1979), 659-679.

4. U. Ascher, J. Christiansen and R.D. russell, colsys - a collocation code for boundary value problems, lecture notes comp.sc. 76, springer verlag, b. childs et. al. (eds.) (1979), 164-185.
5. C. Deboor and R. Weiss, solveblok: a package for solving almost block diagonal linear systems, acm trans. math. software 6 (1980), 80-87.

Name

dae — Differential algebraic equations solver

```
y=dae(initial,t0,t,res)
[y [,hd]]=dae(initial,t0,t [,rtol, [atol]],res [,jac] [,hd])
[y,rd]=dae("root",initial,t0,t,res,ng,surface)
[y ,rd [,hd]]=dae("root",initial,t0,t [,rtol, [atol]],res [,jac], ng, surface [
```

Parameters

initial

a column vector. It may be equal to x_0 or $[x_0; \dot{x}_0]$. Where x_0 is the state value at initial time t_0 and $\dot{x}_0$ is the initial state derivative value or an estimation of it (see below).

t0

a real number, the initial time.

t

real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dae's step point by setting `%DAEOPTIONS(2)=1`.

rtol

a real scalar or a column vector of same size as x_0 . The relative error tolerance of solution. If **rtol** is a vector the tolerances are specified for each component of the state.

atol

a real scalar or a column vector of same size as x_0 . The absolute error tolerance of solution. If **atol** is a vector the tolerances are specified for each component of the state.

res

an external. Computes the value of $g(t, y, \dot{y})$. It may be

a Scilab function

In this case, Its calling sequence must be `[r,ires]=res(t,x,xdot)` and **res** must return the residue $r=g(t, x, \dot{x})$ and error flag **ires**. **ires** = 0 if **res** succeeds to compute r , =-1 if residue is locally not defined for $(t, x, \dot{x})$, =-2 if parameters are out of admissible range.

a list

This form of external is used to pass parameters to the function. It must be as follows:

```
list(res,p1,p2,...)
```

where the calling sequence of the function **res** is now

```
r=res(t,y,ydot,p1,p2,...)
```

res still returns the residual value as a function of $(t, x, \dot{x}, x_1, x_2, \dots)$, and $p_1, p_2, \dots$ are function parameters.

a character string

it must refer to the name of a C or fortran routine. Assuming that `<r_name>` is the given name.

- The Fortran calling sequence must be

`<r_name>(t,x,xdot,res,ires,rpar,ipar)`

```
double precision t,x(*),xdot(*),res(*),rpar(*)
integer ires,ipar(*)
```

- The C calling sequence must be

```
C2F(<r_name>)(double *t, double *x, double *xdot, double *res, integer *ires, double
*rpar, integer *ipar)
```

where

- **t** is the current time value
- **x** the state array
- **xdot** the array of state derivatives
- **res** the array of residuals
- **ires** the execution indicator
- **rpar** is the array of floating point parameter values, needed but cannot be set by the **dae** function
- **ipar** is the array of floating integer parameter values, needed but cannot be set by the **dae** function

jac

an external. Computes the value of $dg/dx+cj*dg/dxdot$ for a given value of parameter **cj**. It may be

a Scilab function

Its calling sequence must be **r=jac(t,x,xdot,cj)** and the **jac** function must return $r=dg(t,x,xdot)/dy+cj*dg(t,x,xdot)/dxdot$ where **cj** is a real scalar

a list

This form of external is used to pass parameters to the function. It must be as follows:

```
list(jac,p1,p2,...)
```

where the calling sequence of the function **jac** is now

```
r=jac(t,x,xdot,p1,p2,...)
```

jac still returns $dg/dx+cj*dg/dxdot$ as a function of **(t,x,xdot,cj,p1,p2,...)**.

a character string

it must refer to the name of a C or fortran routine. Assuming that **<j_name>** is the given name,

- The Fortran calling sequence must be

```
<j_name>(t, x, xdot, r, cj, ires, rpar, ipar)
```

```
double precision t, x(*), xdot(*), r(*), ci, rpar(*)
```

```
integer ires, ipar(*)
```

- The C calling sequence must be

```
C2F(<j_name>)(double *t, double *x, double *xdot, double *r, double *cj,
integer *ires, double *rpar, integer *ipar)
```

where t , x , $xdot$, $ires$, $rpar$, $ipar$ have similar definition as above, r is the results array

surface

an external. Computes the value of the column vector `surface(t,x)` with ng components. Each component defines a surface.

a Scilab function

Its calling sequence must be `r=surface(t,x)`, this function must return a vector with ng elements.

a list

This form of external is used to pass parameters to the function. It must be as follows:

```
list(surface, p1, p2, ...)
```

where the calling sequence of the function `surface` is now

```
r=surface(t,x,p1,p2,...)
```

character string

it must refer to the name of a C or fortran routine. Assuming that `<s_name>` is the given name,

- The Fortran calling sequence must be

```
<r_name>(nx, t, x, ng, r, rpar, ipar)
double precision t, x(*), r(*), rpar(*)
integer nx, ng, ipar(*)
```

- The C calling sequence must be

```
C2F(<r_name>)(double *t, double *x, double *xdot, double *r, double *cj,
integer *ires, double *rpar, integer *ipar)
```

where t , x , $rpar$, $ipar$ have similar definition as above, ng is the number of surfaces, nx the dimension of the state and r is the results array.

rd

a vector with two entries `[times num]` `times` is the value of the time at which the surface is crossed, `num` is the number of the crossed surface

hd

a real vector, as an output it stores the `dae` context. It can be used as an input argument to resume integration (hot restart).

y

real matrix. If `%DAEOPTIONS(2)=1`, each column is the vector `[t;x(t);xdot(t)]` where t is time index for which the solution had been computed. Else y is the vector `[x(t);xdot(t)]`.

Description

The `dae` function is a gateway built above the `dassl` and `dasrt` function designed for implicit differential equations integration.

```
g(t,x,xdot)=0
x(t0)=x0 and xdot(t0)=xdot0
```

If `xdot0` is not given in the `initial` argument, the `dae` function tries to compute it solving $g(t,x_0,\dot{x}_0)=0$,

if `xdot0` is given in the `initial` arguments it may be either a compatible derivative satisfying $g(t,x_0,\dot{x}_0)=0$ or an approximate value. In the latter case `%DAEOPTIONS(7)` must be set to 1.

Detailed examples using Scilab and C coded externals are given in modules/differential_equations/tests/unit_tests/dassldasrt.tst

Examples

```
//Example with Scilab code
function [r,ires]=chemres(t,y,yd)
    r(1) = -0.04*y(1) + 1d4*y(2)*y(3) - yd(1);
    r(2) = 0.04*y(1) - 1d4*y(2)*y(3) - 3d7*y(2)*y(2) - yd(2);
    r(3) = y(1) + y(2) + y(3)-1;
    ires = 0;
endfunction
function pd=chemjac(x,y,yd,cj)
    pd=[-0.04-cj , 1d4*y(3) , 1d4*y(2);
        0.04 , -1d4*y(3)-2*3d7*y(2)-cj , -1d4*y(2);
        1 , 1 , 1 ]
endfunction

x0=[1; 0; 0];
xd0=[-0.04; 0.04; 0];
t=[1.d-5:0.02:.4, 0.41:.1:4, 40, 400, 4000, 40000, 4d5, 4d6, 4d7, 4d8, 4d9, 4d10];

y=dae([x0,xd0],0,t,chemres); // returns requested observation time points

%DAEOPTIONS=list([],1,[],[],[],0,0); // ask dae mesh points to be returned
y=dae([x0,xd0],0,4d10,chemres); // without jacobian
y=dae([x0,xd0],0,4d10,chemres,chemjac); // with jacobian

//example with C code (c compiler needed) -----
// -1- create the C codes in TMPDIR - Vanderpol equation, implicit form
code=['#include <math.h>'
    'void res22(double *t,double *y,double *yd,double *res,int *ires,double *
    '{res[0] = yd[0] - y[1];'
    ' res[1] = yd[1] - (100.0*(1.0 - y[0]*y[0])*y[1] - y[0]);}'
    ', '
    'void jac22(double *t,double *y,double *yd,double *pd,double *cj,double *
    '{pd[0]=*cj - 0.0;'
    ' pd[1]= - (-200.0*y[0]*y[1] - 1.0);'
    ' pd[2]= - 1.0;'
    ' pd[3]=*cj - (100.0*(1.0 - y[0]*y[0]));}'
```

```

    'void gr22(int *neq, double *t, double *y, int *ng, double *groot, double
    '{ groot[0] = y[0];}'
mputl(code,TMPDIR+'/t22.c')
//-2- compile and load them
ilib_for_link(['res22' 'jac22' 'gr22'],'t22.c',[],'c',TMPDIR+'/Makefile',TMPDIR)
exec(TMPDIR+'/t22loader.sce')
//-3- run
rtol=[1.d-6;1.d-6];atol=[1.d-6;1.d-4];
t0=0;y0=[2;0];y0d=[0;-2];t=[20:20:200];ng=1;
//simple simulation
t=0:0.003:300;
yy=dae([y0,y0d],t0,t,atol,rtol,'res22','jac22');
clf();plot(yy(1,:),yy(2,:))
//find first point where yy(1)=0
[yy,nn,hotd]=dae("root",[y0,y0d],t0,300,atol,rtol,'res22','jac22',ng,'gr22');
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))

//hot restart for next point
t01=nn(1);[pp,qq]=size(yy);y01=yy(2:3,qq);y0d1=yy(3:4,qq);
[yy,nn,hotd]=dae("root",[y01,y0d1],t01,300,atol,rtol,'res22','jac22',ng,'gr22',
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))

```

See Also

ode, daeoptions, dassl, impl, fort, link, external

Name

daeoptions — set options for dae solver

```
daeoptions()
```

Description

If it exists in the dae function calling context the variable %DAEOPTIONS the dae function use it to sets options.

This daeoptions function interactively displays a command which should be executed to set various options of the dae solver.

CAUTION: the dae function checks if this variable exists and in this case it uses it. For using default values you should clear this variable. Note that daeoptions does not create this variable. To create it you must execute the command line displayed by daeoptions.

The variable %DAEOPTIONS is a list with the following elements:

```
list(tstop, imode, band, maxstep, stepin, nonneg, isest)
```

The default value is:

```
list([], 0, [], [], [], 0, 0)
```

The meaning of the elements is described below.

tstop

a real scalar or an empty matrix, gives the maximum time for which g is allowed to be evaluated. An empty matrix means "no limits" imposed for time.

imode

if it is 0 dae returns only the user specified time point values if it is 1 dae returns its intermediate computed values.

band

a two components vector which give the definition [ml, mu] of band matrix computed by jac ;

$r(i - j + ml + mu + 1, j) = dg(i)/dy(j) + c_j * dg(i)/dydot(j)$. If jac returns a full matrix set band=[]

maxstep

A scalar or an empty matrix, the maximum step size, empty matrix means "no limitation".

stepin

A scalar or an empty matrix, the minimum step size, empty matrix means "not specified".

nonneg

A scalar, must be set to 0 if the solution is known to be non negative. In the other case it must be set to 1.

isest

A scalar, must be set to 0 is the given initial condition is compatible: $g(t_0, x_0, xdot_0)=0$. 1 an set to 1 if xdot0 is just an estimation.

See Also

dae

Name

dasrt — DAE solver with zero crossing

```
[r,nn,[,hd]]=dasrt(x0,t0,t [,atol,[rtol]],res [,jac],ng, surf [,info] [,hd])
```

Parameters

x0

is either **y0** (**ymdot0** is estimated by **dassl** with zero as first estimate) or the matrix **[y0 ymdot0]**. **g(t,y0,ymdot0)** must be equal to zero. If you only know an estimate of **ymdot0** set **info(7)=1**

y0

real column vector of initial conditions.

ymdot0

real column vector of the time derivative of **y** at **t0** (may be an estimate).

t0

real number is the initial instant.

t

real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each **dassl**'s step point by setting **info(2)=1**.

nn

a vector with two entries **[times num]** **times** is the value of the time at which the surface is crossed, **num** is the number of the crossed surface

atol,rtol

real scalars or column vectors of same size as **y**. **atol,rtol** give respectively absolute and relative error tolerances of solution. If vectors the tolerances are specified for each component of **y**.

res

external (function or list or string). Computes the value of **g(t,y,ymdot)**. It may be :

- A Scilab function.

Its calling sequence must be **[r,ires]=res(t,y,ymdot)** and **res** must return the residue **r=g(t,y,ymdot)** and error flag **ires**. **ires = 0** if **res** succeeds to compute **r**, **=-1** if residue is locally not defined for **(t,y,ymdot)**, **=-2** if parameters are out of admissible range.

- A list.

This form allows to pass parameters other than **t,y,ymdot** to the function. It must be as follows:

```
list(res,x1,x2,...)
```

where the calling sequence of the function **res** is now

```
r=res(t,y,ymdot,x1,x2,...)
```

res still returns **r=g(t,y,ymdot)** as a function of **(t,y,ymdot,x1,x2,...)**.

Warning: this form must not be used if there is no extra argument to pass to the function.

- A string.

it must refer to the name of a C or fortran subroutine linked with Scilab.

In C The calling sequence must be:

```
void res(double *t, double y[], double yd[], double r[],
        int *ires, double rpar[], int ipar[])
```

In Fortran it must be:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

The rpar and ipar arrays must be present but cannot be used.

jac

external (function or list or string). Computes the value of $dg/dy+cj*dg/dydot$ for a given value of parameter cj

- A Scilab function.

Its calling sequence must be $r=jac(t,y,ydot,cj)$ and the **jac** function must return $r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot$ where cj is a real scalar

- A list.

it must be as follows

```
list(jac,x1,x2,...)
```

where the calling sequence of the function **jac** is now

```
r=jac(t,y,ydot,cj,x1,x2,...)
```

jac still returns $dg/dy+cj*dg/dydot$ as a function of $(t,y,ydot,cj,x1,x2,...)$.

- A character string.

it must refer to the name of a fortran subroutine linked with scilab

In C The calling sequence must be:

```
void jac(double *t, double y[], double yd[], double pd[],
        double *cj, double rpar[], int ipar[])
```

In Fortran it must be:

```
subroutine jac(t,y,yd,pd,cj,rpar,ipar)
double precision t, y(*),yd(*),pd(*),cj,rpar(*)
```

```
integer ipar(*)
```

surf

external (function or list or string). Computes the value of the column vector `surf(t,y)` with `ng` components. Each component defines a surface. It may be defined by:

- A Scilab function.

Its calling sequence must be `surf(t,y)`

- A list.

it must be as follows

```
list(surf,x1,x2,...)
```

where the calling sequence of the function `surf` is now

```
r=surf(t,y,x1,x2,...)
```

- A character string.

it must refer to the name of a fortran subroutine linked with scilab

In C The calling sequence must be:

```
void surf(int *ny, double *t, double y[], int *ng, double gout[])
```

In Fortran it must be:

```
subroutine surf(ny,t,y,ng,gout)
double precision t, y(*),gout(*)
integer ny,ng
```

info

list which contains 7 elements, default value is `list([],0,[],[],[],0,0)`

info(1)

real scalar which gives the maximum time for which `g` is allowed to be evaluated or an empty matrix `[]` if no limits imposed for time.

info(2)

flag which indicates if `dassl` returns its intermediate computed values (`flag=1`) or only the user specified time point values (`flag=0`).

info(3)

2 components vector which give the definition `[ml,mu]` of band matrix computed by `jac`; `r(i - j + ml + mu + 1,j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)".` If `jac` returns a full matrix set `info(3)=[ ]`.

info(4)

real scalar which gives the maximum step size. Set `info(4)=[ ]` if no limitation.

info(5)

real scalar which gives the initial step size. Set `info(4)=[ ]` if not specified.

info(6)

set `info(6)=1` if the solution is known to be non negative, else set `info(6)=0`.

info(7)

set `info(7)=1` if `ydot0` is just an estimation, `info(7)=0` if `g(t0,y0,ydot0)=0`.

hd

real vector which allows to store the `dassl` context and to resume integration

r

real matrix . Each column is the vector `[t;x(t);xdot(t)]` where `t` is time index for which the solution had been computed

Description

Solution of the implicit differential equation

```
g(t,y,ydot)=0
y(t0)=y0 and ydot(t0)=ydot0
```

Returns the surface crossing instants and the number of the surface reached in `nn`.

Detailed examples can be found in `SCIDIR/tests/dassldasrt.tst`

Examples

```
//dy/dt = ((2*log(y)+8)/t - 5)*y, y(1) = 1, 1<=t<=6
//g1 = ((2*log(y)+8)/t - 5)*y
//g2 = log(y) - 2.2491
y0=1;t=2:6;t0=1;y0d=3;
atol=1.d-6;rtol=0;ng=2;

deff(' [delta,ires]=res1(t,y,ydot)', 'ires=0;delta=ydot-((2*log(y)+8)/t-5)*y')
deff(' [rts]=gr1(t,y)', 'rts=[((2*log(y)+8)/t-5)*y;log(y)-2.2491]')

[yy,nn]=dasrt([y0,y0d],t0,t,atol,rtol,res1,ng,gr1);
//(Should return nn=[2.4698972 2])
```

See Also

[ode](#), [dassl](#), [impl](#), [fort](#), [link](#), [external](#)

Name

dassl — differential algebraic equation

```
[r [,hd]]=dassl(x0,t0,t [,atol,[rtol]],res [,jac] [,info] [,hd])
```

Parameters

x0

is either **y0** (**ymdot0** is estimated by **dassl** with zero as first estimate) or the matrix **[y0 ymdot0]**. **g(t,y,ymdot)** must be equal to zero. If you only know an estimate of **ymdot0** set **info(7)=1**

y0

real column vector of initial conditions.

ymdot0

real column vector of the time derivative of **y** at **t0** (may be an estimate).

t0

real number is the initial instant.

t

real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each **dassl**'s step point by setting **info(2)=1**.

atol,rtol

real scalars or column vectors of same size as **y**. **atol,rtol** give respectively absolute and relative error tolerances of solution. If vectors the tolerances are specified for each component of **y**.

res

external (function or list or string). Computes the value of **g(t,y,ymdot)**. It may be :

- A Scilab function.

Its calling sequence must be **[r,ires]=res(t,y,ymdot)** and **res** must return the residue **r=g(t,y,ymdot)** and error flag **ires**. **ires = 0** if **res** succeeds to compute **r**, **=-1** if residue is locally not defined for **(t,y,ymdot)**, **=-2** if parameters are out of admissible range.

- A list.

This form allows to pass parameters other than **t,y,ymdot** to the function. It must be as follows:

```
list(res,x1,x2,...)
```

where the calling sequence of the function **res** is now

```
r=res(t,y,ymdot,x1,x2,...)
```

res still returns **r=g(t,y,ymdot)** as a function of **(t,y,ymdot,x1,x2,...)**.

- A string.

it must refer to the name of a C or fortran subroutine linked with Scilab.

In C The calling sequence must be:

```
void res(double *t, double y[], double yd[], double r[],
        int *ires, double rpar[], int ipar[])
```

In Fortran it must be:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

The rpar and ipar arrays must be present but cannot be used.

jac

external (function or list or string). Computes the value of $dg/dy+cj*dg/dydot$ for a given value of parameter Cj

- A Scilab function.

Its calling sequence must be $r=jac(t,y,ydot,cj)$ and the **jac** function must return $r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot$ where Cj is a real scalar

- A list.

it must be as follows

```
list(jac,x1,x2,...)
```

where the calling sequence of the function **jac** is now

```
r=jac(t,y,ydot,cj,x1,x2,...)
```

jac still returns $dg/dy+cj*dg/dydot$ as a function of $(t,y,ydot,cj,x1,x2,...)$.

- A character string.

it must refer to the name of a fortran subroutine linked with scilab

In C The calling sequence must be:

```
void jac(double *t, double y[], double yd[], double pd[],
        double *cj, double rpar[], int ipar[])
```

In Fortran it must be:

```
subroutine jac(t,y,yd,pd,cj,rpar,ipar)
double precision t, y(*),yd(*),pd(*),cj,rpar(*)
integer ipar(*)
```

info

optional list which contains 7 elements. Default value is `list([],0,[],[],[],0,0);`

info(1)
real scalar which gives the maximum time for which **g** is allowed to be evaluated or an empty matrix `[]` if no limits imposed for time.

info(2)
flag which indicates if **dassl** returns its intermediate computed values (**flag**=1) or only the user specified time point values (**flag**=0).

info(3)
2 components vector which give the definition `[ml,mu]` of band matrix computed by **jac**; $r(i - j + ml + mu + 1, j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)"$. If **jac** returns a full matrix set **info(3)**=`[]`.

info(4)
real scalar which gives the maximum step size. Set **info(4)**=`[]` if no limitation.

info(5)
real scalar which gives the initial step size. Set **info(4)**=`[]` if not specified.

info(6)
set **info(6)**=1 if the solution is known to be non negative, else set .

info(7)
set **info(7)**=1 if **ydot0** is just an estimation, **info(7)**=0 if $g(t_0, y_0, ydot_0)=0$.

hd
real vector which allows to store the **dassl** context and to resume integration

r
real matrix . Each column is the vector `[t;x(t);xdot(t)]` where **t** is time index for which the solution had been computed

Description

The **dassl** function integrate the algebro-differencial equation and returns the evolution of **y** a given time points

```
g(t,y,ydot)=0
y(t0)=y0 and ydot(t0)=ydot0
```

Examples

```
function [r,ires]=chemres(t,y,yd)
    r=[-0.04*y(1)+1d4*y(2)*y(3)-yd(1)
        0.04*y(1)-1d4*y(2)*y(3)-3d7*y(2)*y(2)-yd(2)
        y(1)+y(2)+y(3)-1];
    ires=0
endfunction

function pd=chemjac(x,y,yd,cj)
    pd=[-0.04-cj , 1d4*y(3) , 1d4*y(2);
        0.04 , -1d4*y(3)-2*3d7*y(2)-cj , -1d4*y(2);
        1 , 1 , 1];
endfunction

y0=[1;0;0];
yd0=[-0.04;0.04;0];
t=[1.d-5:0.02:.4,0.41:.1:4,40,400,4000,40000,4d5,4d6,4d7,4d8,4d9,4d10];
```

```

y=dassl([y0,yd0],0,t,chemres);

info=list([],0,[],[],[],0,0);
info(2)=1;
y=dassl([y0,yd0],0,4d10,chemres,info);
y=dassl([y0,yd0],0,4d10,chemres,chemjac,info);

//Using extra argument for parameters
//-----
function [r,ires]=chemres(t,y,yd ,a,b,c)
    r=[-a*y(1)+b*y(2)*y(3)-yd(1)
        a*y(1)-b*y(2)*y(3)-c*y(2)*y(2)-yd(2)
        y(1)+y(2)+y(3)-1];
    ires=0
endfunction

function pd=chemjac(x,y,yd,cj, a,b,c)
    pd=[-a-cj , b*y(3) , b*y(2);
        a , -b*y(3)-2*c*y(2)-cj , -b*y(2);
        1 , 1 , 1 ]
endfunction

y=dassl([y0,yd0],0,t,list(chemres,0.04,1d4,3d7),list(chemjac,0.04,1d4,3d7));

//using C code
//-----
// - create the C code
rescode=['void chemres(double *t, double y[], double yd[], double r[], int *ires)
{
    r[0] = -0.04*y[0]+1.0e4*y[1]*y[2] -yd[0];
    r[1] = 0.04*y[0]-1.0e4*y[1]*y[2]-3.0e7*y[1]*y[1]-yd[1];
    r[2] = y[0]+y[1]+y[2]-1;
    *ires = 0;
}'];

jaccode=['void chemjac(double *t, double y[], double yd[], double pd[], double
{
    /* first column*/
    pd[0] = -0.04-*cj;
    pd[1] = 0.04;
    pd[2] = 1.0;
    /* second column*/
    pd[3] = 1.0e4*y[2];
    pd[4] = -1.0e4*y[2]-2*3.0e7*y[1]-*cj;
    pd[5] = 1.0;
    /* third column*/
    pd[6] = 1.0e4*y[1];
    pd[7] = -1.0e4*y[1];
    pd[8] = 1.0;
}'];

mputl([rescode;jaccode],TMPDIR+'/mycode.c') //create the C file

// - compile it
ilib_for_link(['chemres','chemjac'],'mycode.c',[],'c',TMPDIR+'/Makefile',TMPDIR

// - link it with Scilab
exec(TMPDIR+'/loader.sce') //incremental linking

```

```
// - call dassl  
y=dassl([y0,yd0],0,t,'chemres','chemjac');
```

See Also

ode, dasrt, impl, fort, link, external

Name

feval — multiple evaluation

```
[z]=feval(x,y,f)
[z]=feval(x,f)
```

Parameters

x,y

two vectors

f

function or character string (for Fortran or C call)

Description

Multiple evaluation of a function for one or two arguments of vector type :

`z=feval(x,f)`

returns the vector `z` defined by `z(i)=f(x(i))`

`z=feval(x,y,f)`

returns the matrix `z` such as `z(i,j)=f(x(i),y(j))`

`f` is an external (function or routine) accepting on one or two arguments which are supposed to be real. The result returned by `f` can be real or complex. In case of a Fortran call, the function '`f`' must be defined in the subroutine `fevaltable.c` (in directory `SCI/modules/differential_equations/src/c`)

Examples

```
deff('[z]=f(x,y)', 'z=x^2+y^2');
feval(1:10,1:5,f)
deff('[z]=f(x,y)', 'z=x+%i*y');
feval(1:10,1:5,f)
feval(1:10,1:5,'parab') //See ffeval.f file
feval(1:10,'parab')
// For dynamic link (see example ftest in ffeval.f)
// you can use the link command (the parameters depend on the machine):
// unix('make ftest.o');link('ftest.o','ftest'); feval(1:10,1:5,'ftest')
```

See Also

evstr, horner, execstr, external, link

Name

impl — differential algebraic equation

```
y=impl([type],y0,ydot0,t0,t [,atol, [rtol]],res,adda [,jac])
```

Parameters

y0,ydot0

real vectors or matrix (initial conditions).

t0

real scalar (initial time).

t

real vector (times at which the solution is computed).

res,adda

externals (function or character string or list).

type

string 'adams' or 'stiff'

atol,rtol

real scalar or real vector of the same size as as y.

jac

external (function or character string or list).

Description

Solution of the linear implicit differential equation

$A(t,y) \, dy/dt = g(t,y)$, $y(t_0) = y_0$

t0 is the initial instant, **y0** is the vector of initial conditions Vector **ydot0** of the time derivative of **y** at **t0** must also be given. **r** The input **res** is an external i.e. a function with specified syntax, or the name a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If **res** is a function, its syntax must be as follows:

```
r = res(t,y,ydot)
```

where **t** is a real scalar (time) and **y** and **ydot** are real vector (state and derivative of the state). This function must return $r = g(t,y) - A(t,y) * ydot$.

If **res** is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCI/modules/differential_equations/sci_gateway/fortran/Ex-impl.f` for an example to do that.

res can also be a list: see the help of `ode`.

The input **adda** is also an external.

If **adda** is a function, its syntax must be as follows:

```
r = adda(t,y,p)
```

and it must return $r=A(t,y)+p$ where p is a matrix to be added to $A(t,y)$.

If `adda` is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCI/modules/differential_equations/sci_gateway/fortran/Ex-impl.f` for an example to do that.

`adda` can also be a list: see the help of `ode`.

The input `jac` is also an external.

If `jac` is a function, its syntax must be as follows:

```
j = jac(t,y,ydot)
```

and it must return the Jacobian of $r=g(t,y)-A(t,y)*ydot$ with respect to y .

If `jac` is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCI/modules/differential_equations/sci_gateway/fortran/Ex-impl.f` for an example to do that.

`jac` can also be a list: see the help of `ode`.

Examples

```
y=impl([1;0;0],[-0.04;0.04;0],0,0.4,'resid','aplusp');  
// Using hot restart  
//[x1,w,iw]=impl([1;0;0],[-0.04;0.04;0],0,0.2,'resid','aplusp');  
// hot start from previous call  
//[x1]=impl([1;0;0],[-0.04;0.04;0],0.2,0.4,'resid','aplusp',w,iw);  
//maxi(abs(x1-x))
```

See Also

`dassl`, `ode`, `external`

Name

int2d — definite 2D integral by quadrature and cubature method

```
[I,err]=int2d(X,Y,f [,params])
```

Parameters

X

a 3 by N array containing the abscissae of the vertices of the N triangles.

Y

a 3 by N array containing the ordinates of the vertices of the N triangles.

f

external (function or list or string) defining the integrand $f(u,v)$;

params

real vector [tol, iclose, maxtri, mevals, iflag]. default value is [1.d-10, 1, 50, 4000, 1].

tol

the desired bound on the error. If iflag=0, tol is interpreted as a bound on the relative error; if iflag=1, the bound is on the absolute error.

iclose

an integer parameter that determines the selection of LQM0 or LQM1 methods. If iclose=1 then LQM1 is used. Any other value of iclose causes LQM0 to be used. LQM0 uses function values only at interior points of the triangle. LQM1 is usually more accurate than LQM0 but involves evaluating the integrand at more points including some on the boundary of the triangle. It will usually be better to use LQM1 unless the integrand has singularities on the boundary of the triangle.

maxtri

the maximum number of triangles in the final triangulation of the region

mevals

the maximum number of function evaluations to be allowed. This number will be effective in limiting the computation only if it is less than $94*\text{maxtri}$ when LQM1 is specified or $56*\text{maxtri}$ when LQM0 is specified.

iflag

I

the integral value

err

the estimated error

Description

int2d computes the two-dimensional integral of a function f over a region consisting of n triangles. A total error estimate is obtained and compared with a tolerance - `tol` - that is provided as input to the subroutine. The error tolerance is treated as either relative or absolute depending on the input value of `iflag`. A 'Local Quadrature Module' is applied to each input triangle and estimates of the total integral and the total error are computed. The local quadrature module is either subroutine LQM0 or subroutine LQM1 and the choice between them is determined by the value of the input variable `iclose`.

If the total error estimate exceeds the tolerance, the triangle with the largest absolute error is divided into two triangles by a median to its longest side. The local quadrature module is then applied to each of the subtriangles to obtain new estimates of the integral and the error. This process is repeated until either (1) the error tolerance is satisfied, (2) the number of triangles generated exceeds the input parameter `maxtri`, (3) the number of integrand evaluations exceeds the input parameter `mevals`, or (4) the function senses that roundoff error is beginning to contaminate the result.

Examples

```
X=[0,0;1,1;1,0];
Y=[0,0;0,1;1,1];
deff('z=f(x,y)', 'z=cos(x+y)')
[I,e]=int2d(X,Y,f)
// computes the integrand over the square [0 1]x[0 1]
```

See Also

intc, intl, int3d, intg, mesh

Authors

Fortran routine twodq Authors: Kahaner,D.K.,N.B.S., Rechar,O.W.,N.B.S.; Barnhill,Robert,Univ. of UTAH

Name

int3d — definite 3D integral by quadrature and cubature method

```
[result,err]=int3d(X,Y,Z,f [,nf[,params]])
```

Parameters

- X**
a 4 by NUMTET array containing the abscissae of the vertices of the NUMTET tetrahedrons.
- Y**
a 4 by NUMTET array containing the ordinates of the vertices of the NUMTET tetrahedrons.
- Z**
a 4 by NUMTET array containing the third coordinates of the vertices of the NUMTET tetrahedrons.
- f**
external (function or list or string) defining the integrand $f(xyz, nf)$, where xyz is the vector of a point coordinates and nf the number functions
- nf**
the number of function to integrate (default is 1)
- params**
real vector [minpts, maxpts, epsabs, epsrel]. default value is [0, 1000, 0.0, 1.d-5].
- epsabs**
Desired bound on the absolute error.
- epsrel**
Desired bound on the relative error.
- minpts**
Minimum number of function evaluations.
- maxpts**
Maximum number of function evaluations. The number of function evaluations over each subregion is 43
- result**
the integral value,or vector of the integral values.
- err**
Estimates of absolute errors.

Description

The function calculates an approximation to a given vector of definite integrals

```
I I I (F ,F ,...,F )      dx(3)dx(2)dx(1),  
    1  2      numfun
```

where the region of integration is a collection of NUMTET tetrahedrons and where

```

F = F (X(1),X(2),X(3)), J = 1,2,...,NUMFUN.
J      J

```

A globally adaptive strategy is applied in order to compute approximations `result(k)` hopefully satisfying, for each component of `I`, the following claim for accuracy: $\text{ABS}(I(K) - \text{RESULT}(K)) \leq \text{MAX}(\text{EPSABS}, \text{EPSREL} * \text{ABS}(I(K)))$

`int3d` repeatedly subdivides the tetrahedrons with greatest estimated errors and estimates the integrals and the errors over the new subtetrahedrons until the error request is met or `MAXPTS` function evaluations have been used.

A 43 point integration rule with all evaluation points inside the tetrahedron is applied. The rule has polynomial degree 8.

If the values of the input parameters `EPSABS` or `EPSREL` are selected great enough, an integration rule is applied over each tetrahedron and the results are added up to give the approximations `RESULT(K)`. No further subdivision of the tetrahedrons will then be applied.

When `int3d` computes estimates to a vector of integrals, all components of the vector are given the same treatment. That is, `I(Fj)` and `I(Fk)` for

`j` not equal to `k`, are estimated with the same subdivision of the region of integration. For integrals with enough similarity, we may save time by applying `int3d` to all integrands in one call. For integrals that varies continuously as functions of some parameter, the estimates produced by `int3d` will also vary continuously when the same subdivision is applied to all components. This will generally not be the case when the different components are given separate treatment.

On the other hand this feature should be used with caution when the different components of the integrals require clearly different subdivisions.

References

Fortran routine `dcutet.f`

Examples

```

X=[0;1;0;0];
Y=[0;0;1;0];
Z=[0;0;0;1];
[RESULT,ERROR]=int3d(X,Y,Z,'int3dex')
// computes the integrand exp(x*x+y*y+z*z) over the
//tetrahedron (0.,0.,0.),(1.,0.,0.),(0.,1.,0.),(0.,0.,1.)

//integration over a cube -1<=x<=1;-1<=y<=1;-1<=z<=1

// bottom -top-      right    -left-    front    -rear-
X=[ 0, 0,      0, 0,      0, 0,      0, 0,      0, 0,      0, 0;
   -1, -1,     -1, -1,      1, 1,     -1, -1,     -1, -1,     -1, -1;
    1, -1,      1, -1,      1, 1,     -1, -1,      1, -1,      1, -1;
    1, 1,       1, 1,       1, 1,     -1, -1,      1, 1,       1, 1];
Y=[ 0, 0,      0, 0,      0, 0,      0, 0,      0, 0,      0, 0;
   -1, -1,     -1, -1,     -1, 1,     -1, 1,     -1, -1,      1, 1;
   -1, 1,      -1, 1,      1, 1,      1, 1,     -1, -1,      1, 1;
    1, 1,       1, 1,     -1, -1,     -1, -1,     -1, -1,      1, 1];
Z=[ 0, 0,      0, 0,      0, 0,      0, 0,      0, 0,      0, 0;
   -1, -1,      1, 1,     -1, 1,     -1, 1,     -1, -1,     -1, -1;
   -1, -1,      1, 1,     -1, -1,     -1, -1,     -1, 1,     -1, 1;

```

```
-1, -1,      1, 1,      1, 1,      1, 1,      1, 1,      1, 1];  
  
function v=f(xyz,numfun),v=exp(xyz'*xyz),endfunction  
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])  
  
function v=f(xyz,numfun),v=1,endfunction  
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])
```

See Also

intc, intl, int2d

Authors

Jarle Berntsen

The Computing Centre, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Norway
Phone.. 47-5-544055 Email.. jarle@eik.ii.uib.no,

Ronald Cools

Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3030 Heverlee, Belgium Phone.. 32-16-201015 (3562) Email.. ronald@cs.kuleuven.ac.be,

Terje O. Espelid

Department of Informatics, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Norway
Phone.. 47-5-544180 Email.. terje@eik.ii.uib.no

Name

intc — Cauchy integral

```
[y]=intc(a,b,f)
```

Parameters

a,b
two complex numbers

f
"external" function

Description

If f is a complex-valued function, `intc(a,b,f)` computes the integral from a to b of $f(z)dz$ along the straight line $a \rightarrow b$ of the complex plane.

See Also

intg , intl

Authors

F. D.

Name

integrate — integration of an expression by quadrature

```
x=integrate(expr,v,x0,x1 [,atol [,rtol]])
```

Parameters

expr

Character string defining a Scilab expression.

v

Character string, the integration variable name)

x0

real number, the lower bound of integration

x1

vector of real numbers, upper bounds of integration

atol

real number (absolute error bound) Default value: 1e-8

rtol

real number, (relative error bound) Default value: 1e-14

x

vector of real numbers, the integral value for each $x1(i)$.

Description

computes :
$$x(i) = \int_{x_0}^{x_1(i)} f(v) dv$$
 for $i=1:\text{size}(x1, '*')$

Where $f(v)$ is given by the expression expr.

The evaluation hopefully satisfies following claim for accuracy: $\text{abs}(I-x) \leq \max(\text{atol}, \text{rtol} * \text{abs}(I))$ where I stands for the exact value of the integral.

Restriction

the given expression should not use variable names with a leading %.

Examples

```
x0=0;x1=0:0.1:2*pi;
X=integrate('sin(x)', 'x', x0, x1);
norm(cos(x1)-(1-X))

x1=-10:0.1:10;
X=integrate(['if x==0 then 1, '
            'else sin(x)/x, end'], 'x', 0, x1)
```

See Also

intg, inttrap, intsplin, ode

Name

intg — definite integral

```
[v,err]=intg(a,b,f [,ea [,er]])
```

Parameters

a,b

real numbers

f

external (function or list or string)

ea, er

real numbers

ea

absolute error required on the result. Default value: 1.d-14

er

relative error required on the result. Default value: 1.d-8

err

estimated absolute error on the result.

Description

`intg(a,b,f)` evaluates the definite integral from `a` to `b` of $f(t)dt$. The function $f(t)$ should be continuous.

The evaluation hopefully satisfies following claim for accuracy: $\text{abs}(I-v) \leq \max(ea, er * \text{abs}(I))$ where I stands for the exact value of the integral.

`f` is an external :

If `f` is function its definition must be as follows `y = f(t)`

If `f` is a list the list must be as follows: `list(f,x1,x2,...)` where `f` is a function with calling sequence `f(t,x1,x2,...)`.

If `f` is a string it refers to a the name of a Fortran function or a C prodedure with a given calling sequence:

In the fortran case the calling sequence should be `double precision function f(x)` where `x` is also a double precision number.

In the C case the calling sequence should be `double f(double *x)`.

Examples

```
//External écrit en Scilab
function y=f(x),y=x*sin(30*x)/sqrt(1-((x/(2*pi))^2)),endfunction
exact=-2.5432596188;
I=intg(0,2*pi,f)
abs(exact-I)

//External écrit en Scilab avec un argument
```

```
function y=f1(x,w),y=x*sin(w*x)/sqrt(1-((x/(2*pi))^2)),endfunction
I=intg(0,2*pi, list(f1,30))
abs(exact-I)

// External écrit en Fortran (un compilateur Fortran est nécessaire)
// Ecriture du code fortran
cd TMPDIR;
F=['      double precision function ffun(x)'
  '      double precision x,pi'
  '      pi=3.14159265358979312d+0'
  '      ffun=x*sin(30.0d+0*x)/sqrt(1.0d+0-(x/(2.0d+0*pi))**2)'
  '      return'
  '      end'];
mputl(F,TMPDIR+'/ffun.f')

// compilation du code Fortran
l=ilib_for_link('ffun','ffun.f',[],'f');

// link incrémental
link(l,'ffun','f')

// integration de la fonction
I=intg(0,2*pi,'ffun')
abs(exact-I)

// External écrit en C (un compilateur C est nécessaire)
// Ecriture du code C
C=['#include <math.h>'
  'double cfun(double *x)'
  '{'
  '  double y,pi=3.14159265358979312;'
  '  y=*x/(2.0e0*pi);'
  '  return *x*sin(30.0e0**x)/sqrt(1.0e0-y*y);'
  '}'];
mputl(C,TMPDIR+'/cfun.c')

// compilation du code C
l=ilib_for_link('cfun','cfun.f',[],'c');

// incremental linking
link(l,'cfun','c')

// integration de la fonction
I=intg(0,2*pi,'cfun')
abs(exact-I)
```

See Also

intc, intl, intrap, intsplin, ode

Used Functions

The associated routines can be found in SCI/modules/differential_equations/src/fortran directory :

dqag0.f and dqags.f from quadpack

Name

intl — Cauchy integral

```
[y]=intl(a,b,z0,r,f)
```

Parameters

`z0`
complex number

`a,b`
two real numbers

`r`
positive real number

`f`
"external" function

Description

If f is a complex-valued function, `intl(a,b,z0,r,f)` computes the integral of $f(z)dz$ along the curve in the complex plane defined by $z0 + r \cdot \exp(i \cdot t)$ for $a \leq t \leq b$ (part of the circle with center $z0$ and radius r with phase between a and b).

See Also

`intc`

Authors

F. D.

Name

ode — ordinary differential equation solver

```
y=ode(y0,t0,t,f)
[y,w,iw]=ode([type],y0,t0,t [,rtol [,atol]],f [,jac] [,w,iw])
[y,rd,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
y=ode("discrete",y0,k0,kvect,f)
```

Parameters

y0
real vector or matrix (initial conditions).

t0
real scalar (initial time).

t
real vector (times at which the solution is computed).

f
external (function or character string or list).

type
one of the following character string: "adams" "stiff" "rk" "rkf" "fix" "discrete" "roots"

rtol,atol
real constants or real vectors of the same size as y.

jac
external (function or character string or list).

w,iw
real vectors.

ng
integer.

g
external (function or character string or list).

k0
integer (initial time).

kvect
integer vector.

Description

ode is the standard function for solving explicit ODE systems defined by: $dy/dt=f(t,y)$, $y(t_0)=y_0$. It is an interface to various solvers, in particular to ODEPACK. The type of problem solved and the method used depend on the value of the first optional argument **type** which can be one of the following strings:

<not given>:

lsoda solver of package ODEPACK is called by default. It automatically selects between nonstiff predictor-corrector Adams method and stiff Backward Differentiation Formula (BDF) method. It uses nonstiff method initially and dynamically monitors data in order to decide which method to use.

"adams":

This is for nonstiff problems. `lsode` solver of package ODEPACK is called and it uses the Adams method.

"stiff":

This is for stiff problems. `lsode` solver of package ODEPACK is called and it uses the BDF method.

"rk":

Adaptive Runge-Kutta of order 4 (RK4) method.

"rkf":

The Shampine and Watts program based on Fehlberg's Runge-Kutta pair of order 4 and 5 (RKF45) method is used. This is for non-stiff and mildly stiff problems when derivative evaluations are inexpensive. This method should generally not be used when the user is demanding high accuracy.

"fix":

Same solver as "rkf", but the user interface is very simple, i.e. only `rtol` and `atol` parameters can be passed to the solver. This is the simplest method to try.

"root":

ODE solver with rootfinding capabilities. The `lsodar` solver of package ODEPACK is used. It is a variant of the `lsoda` solver where it finds the roots of a given vector function. See help on `ode_root` for more details.

"discrete":

Discrete time simulation. See help on `ode_discrete` for more details.

In this help we only describe the use of `ode` for standard explicit ODE systems.

- The simplest call of `ode` is: `y=ode(y0,t0,t,f)` where `y0` is the vector of initial conditions, `t0` is the initial time, `t` is the vector of times at which the solution `y` is computed and `y` is matrix of solution vectors `y=[y(t(1)),y(t(2)),...]`.

The input argument `f` defines the RHS of the first order differential equation: $dy/dt=f(t,y)$. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list:

- If `f` is a Scilab function, its syntax must be `ydot = f(t,y)`, where `t` is a real scalar (time) and `y` a real vector (state) and `ydot` a real vector (dy/dt)
- If `f` is a character string, it refers to the name of a Fortran subroutine or a C function, i.e. if `ode(y0,t0,t,"fex")` is the command, then the subroutine `fex` is called.

The Fortran routine must have the following calling sequence: `fex(n,t,y,ydot)`, with `n` an integer, `t` a double precision scalar, `y` and `ydot` double precision vectors.

The C function must have the following prototype: `fex(int *n,double *t,double *y,double *ydot)`

`t` is the time, `y` the state and `ydot` the state derivative (dy/dt)

This external can be build in a OS independant way using `ilib_for_link` and dynamically linked to Scilab by the `link` function.

- The `f` argument can also be a list with the following structure: `lst=list(real_f,u1,u2,...,un)` where `real_f` is a Scilab function with syntax: `ydot = f(t,y,u1,u2,...,un)`

This syntax allows to use parameters as the arguments of `real_f`.

The function `f` can return a $p \times q$ matrix instead of a vector. With this matrix notation, we solve the $n=p+q$ ODE's system $dY/dt=F(t,Y)$ where Y is a $p \times q$ matrix. Then initial conditions, Y_0 , must also be a $p \times q$ matrix and the result of `ode` is the $p \times q(T+1)$ matrix $[Y(t_0), Y(t_1), \dots, Y(t_T)]$.

- Optional input parameters can be given for the error of the solution: `rtol` and `atol` are threshold for relative and absolute estimated errors. The estimated error on $y(i)$ is: `rtol(i)*abs(y(i))+atol(i)`

and integration is carried out as far as this error is small for all components of the state. If `rtol` and/or `atol` is a constant `rtol(i)` and/or `atol(i)` are set to this constant value. Default values for `rtol` and `atol` are respectively `rtol=1.d-5` and `atol=1.d-7` for most solvers and `rtol=1.d-3` and `atol=1.d-4` for "rkf" and "fix".

- For stiff problems, it is better to give the Jacobian of the RHS function as the optional argument `jac`. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If `jac` is a function the syntax should be `J=jac(t,y)`

where t is a real scalar (time) and y a real vector (state). The result matrix J must evaluate to df/dx i.e. $J(k,i) = df_k/dx_i$ with $f_k = k$ th component of f .

If `jac` is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence:

Fortran case:

```
subroutine fex(n,t,y,ml,mu,J,nrpd)
integer n,ml,mu,nrpd
double precision t,y(*),J(*)
```

C case:

```
void fex(int *n,double *t,double *y,int *ml,int *mu,double *J,int *nrpd,)
```

`jac(n,t,y,ml,mu,J,nrpd)`. In most cases you have not to refer `ml`, `mu` and `nrpd`.

If `jac` is a list the same conventions as for `f` apply.

- Optional arguments `w` and `iw` are vectors for storing information returned by the integration routine (see `ode_optional_output` for details). When these vectors are provided in RHS of `ode` the integration re-starts with the same parameters as in its previous stop.
- More options can be given to ODEPACK solvers by using `%ODEOPTIONS` variable. See `odeoptions`.

Examples

```
// ----- Simple one dimension ODE (Scilab function external)
// dy/dt=y^2-y sin(t)+cos(t), y(0)=0
function ydot=f(t,y),ydot=y^2-y*sin(t)+cos(t),endfunction
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,f)
```

```

plot(t,y)

// ----- Simple one dimension ODE (C coded external)
ccode=['#include <math.h>'
      'void myode(int *n,double *t,double *y,double *ydot)'
      '{'
      '  ydot[0]=y[0]*y[0]-y[0]*sin(*t)+cos(*t);'
      '}']
mputl(ccode,TMPDIR+'/myode.c') //create the C file
ilib_for_link('myode','myode.c',[],'c',TMPDIR+'/Makefile',TMPDIR+'/loader.sce')
exec(TMPDIR+'/loader.sce') //incremental linking
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,'myode');

// ----- Simulation of dx/dt = A x(t) + B u(t) with u(t)=sin(omega*t),
// x0=[1;0]
// solution x(t) desired at t=0.1, 0.2, 0.5, 1.
// A and u function are passed to RHS function in a list.
// B and omega are passed as global variables
function xdot=linear(t,x,A,u),xdot=A*x+B*u(t),endfunction
function ut=u(t),ut=sin(omega*t),endfunction
A=[1 1;0 2];B=[1;1];omega=5;
ode([1;0],0,[0.1,0.2,0.5,1],list(linear,A,u))

// ----- Matrix notation Integration of the Riccati differential equation
// Xdot=A'*X + X*A - X'*B*X + C , X(0)=Identity
// Solution at t=[1,2]
function Xdot=ric(t,X),Xdot=A'*X+X*A-X'*B*X+C,endfunction
A=[1,1;0,2]; B=[1,0;0,1]; C=[1,0;0,1];
t0=0;t=0:0.1:%pi;
X=ode(eye(A),0,t,ric)

// ----- Matrix notation, Computation of exp(A)
A=[1,1;0,2];
function xdot=f(t,x),xdot=A*x;,endfunction
ode(eye(A),0,1,f)
ode("adams",eye(A),0,1,f)

// ----- Matrix notation, Computation of exp(A) with stiff matrix, Jacobian
A=[10,0;0,-1];
function xdot=f(t,x),xdot=A*x,endfunction
function J=Jacobian(t,y),J=A,endfunction
ode("stiff",[0;1],0,1,f,Jacobian)

```

See Also

ode_discrete, ode_root, dassl, impl, odedc, odeoptions, csim, ltitr, rtitr

Authors

Alan C. Hindmarsh
 , mathematics and statistics division, l-316 livermore, ca 94550.19

Bibliography

Alan C. Hindmarsh, lsode and lsodi, two new initial value ordinary differential equation solvers, acm-signum newsletter, vol. 15, no. 4 (1980), pp. 10-11.

Used Functions

The associated routines can be found in `SCI/modules/differential_equations/src/fortran` directory :

`lsode.f` `lsoda.f` `lsodar.f`

Name

ode_discrete — ordinary differential equation solver, discrete time simulation

```
y=ode("discrete",y0,k0,kvect,f)
```

Parameters

y0
real vector or matrix (initial conditions).

t0
real scalar (initial time).

f
external i.e. function or character string or list.

k0
integer (initial time).

kvect
integer vector.

Description

With this syntax (first argument equal to "discrete") ode computes recursively $y(k+1)=f(k,y(k))$ from an initial state $y(k_0)$ and returns $y(k)$ for k in **kvect**. **kvect(1)** must be greater than or equal to **k0**.

Other arguments and other options are the same as for ode, see the ode help.

Examples

```
y1=[1;2;3]; deff("yp=a_function(k,y)","yp=A*y+B*u(k)")
A=diag([0.2,0.5,0.9]); B=[1;1;1];u=1:10;n=5;
y=ode("discrete",y1,1,1:n,a_function);
y(:,2)-(A*y1+B*u(1))

// Now y evaluates at [y3,y5,y7,y9]
y=ode("discrete",y1,1,3:2:9,a_function)
```

See Also

ode

Name

ode_optional_output — ode solvers optional outputs description

Description

This page describes the the most important values returned in the optional lhs ode function arguments `w` and `iw`. These are valid only for the `lsode` `lsoda` and `lsodar` ode solver. For more details, one can look at the solvers fortran code comments in `SCI/modules/differential_equations/src/fortran/lsod*.f`.

`w(11)`

the step size in `t` last used (successfully).

`w(12)`

the step size to be attempted on the next step.

`w(13)`

the current value of the independent variable which the solver has actually reached, i.e. the current internal mesh point in `t`. on output, `tcurl` will always be at least as far as the argument `t`, but may be farther (if interpolation was done).

`w(14)`

a tolerance scale factor, greater than 1.0, computed when a request for too much accuracy was detected (`istate` = -3 if detected at the start of the problem, `istate` = -2 otherwise). if `itol` is left unaltered but `rtol` and `atol` are uniformly scaled up by a factor of `tolssf=w(14)` for the next call, then the solver is deemed likely to succeed. (the user may also ignore `tolssf` and alter the tolerance parameters in any other way appropriate.)

`w(15)`

the value of `t` at the time of the last method switch, if any. This value is not significant with `lsode` solver.

`iw(10)`

the number of `g` evaluations for the problem so far. This value is only significant for `lsodar` solver.

`iw(11)`

the number of steps taken for the problem so far.

`iw(12)`

the number of `f` evaluations for the problem so far.

`iw(13)`

the number of jacobian evaluations (and of matrix lu decompositions) for the problem so far.

`iw(14)`

the method order last used (successfully).

`iw(15)`

the order to be attempted on the next step.

`iw(16)`

the index of the component of largest magnitude in the weighted local error vector (`e(i)/ewt(i)`), on an error return with `istate` = -4 or -5.

`iw(17)`

the length of `w` actually required, assuming that the length of `rwork` is to be fixed for the rest of the problem, and that switching may occur. this is defined on normal returns and on an illegal input return for insufficient storage.

iw(18)

the length of `iw` actually required, assuming that the length of `iw` is to be fixed for the rest of the problem, and that switching may occur. this is defined on normal returns and on an illegal input return for insufficient storage.

iw(19)

the method indicator for the last successful step.. 1 means adams (nonstiff), 2 means bdf (stiff). This value is not significant with `lsode` solver.

iw(20)

the current method indicator.. 1 means adams (nonstiff), 2 means bdf (stiff). this is the method to be attempted on the next step. thus it differs from `iw(19)` only if a method switch has just been made. This value is not significant with `lsode` solver.

Name

ode_root — ordinary differential equation solver with root finding

```
y,rd[,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
```

Parameters

y0
real vector or matrix (initial conditions).

t0
real scalar (initial time).

t
real vector (times at which the solution is computed).

f
external i.e. function or character string or list.

rtol,atol
real constants or real vectors of the same size as **y**.

jac
external i.e. function or character string or list.

w,iw
real vectors.

ng
integer.

g
external i.e. function or character string or list.

Description

With this syntax (first argument equal to "root") **ode** computes the solution of the differential equation $dy/dt=f(t,y)$ until the state $y(t)$ crosses the surface $g(t,y)=0$.

g should give the equation of the surface. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If **g** is a function the syntax should be as follows:

```
z = g(t,y)
```

where **t** is a real scalar (time) and **y** a real vector (state). It returns a vector of size **ng** which corresponds to the **ng** constraints. If **g** is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence: **g(n,t,y,ng,gout)** where **ng** is the number of constraints and **gout** is the value of **g** (output of the program). If **g** is a list the same conventions as for **f** apply (see **ode** help).

Output **rd** is a $1 \times k$ vector. The first entry contains the stopping time. Other entries indicate which components of **g** have changed sign. **k** larger than 2 indicates that more than one surface ((**k**-1) surfaces) have been simultaneously traversed.

Other arguments and other options are the same as for **ode**, see the **ode** help.

Examples

```
// Integration of the differential equation
// dy/dt=y , y(0)=1, and finds the minimum time t such that y(t)=2
deff("[ydot]=f(t,y)", "ydot=y")
deff("[z]=g(t,y)", "z=y-2")
y0=1;ng=1;
[y,rd]=ode("roots",y0,0,2,f,ng,g)

deff("[z]=g(t,y)", "z=y-[2;2;33]")
[y,rd]=ode("roots",1,0,2,f,3,g)
```

See Also

dasrt, ode

Name

odedc — discrete/continuous ode solver

```
yt=odedc(y0,nd,stdel,t0,t,f)
```

Parameters

y_0

real column vector (initial conditions), $y_0=[y_{0c};y_{0d}]$ where y_{0d} has nd components.

nd

integer, dimension of y_{0d}

$stdel$

real vector with one or two entries, $stdel=[h, \delta]$ (with $\delta=0$ as default value).

t_0

real scalar (initial time).

t

real (row) vector, instants where y_t is calculated .

f

external i.e. function or character string or list with calling sequence: $yp=f(t,yc,yd,flag)$.

Description

$y=odedc([y_{0c};y_{0d}],nd,[h,\delta],t_0,t,f)$ computes the solution of a mixed discrete/continuous system. The discrete system state y_{d_k} is embedded into a piecewise constant $y_d(t)$ time function as follows:

```
y_d(t) = y_{d\_k} for t in  
[t_k=delay+k*h, t_{(k+1)}=delay+(k+1)*h[ (with delay=h*delta).
```

The simulated equations are now:

```
dy_c/dt = f(t,y_c(t),y_d(t),0), for t in [t_k,t_{(k+1)}[  
y_c(t_0) = y_{0c}
```

and at instants t_k the discrete variable y_d is updated by:

```
y_d(t_{k+}) = f(y_c(t_{k-}),y_d(t_{k-}),1)
```

Note that, using the definition of $y_d(t)$ the last equation gives

```
y_{d\_k} = f(t_k,y_c(t_{k-}),y_d(t_{(k-1)}),1) (y_c is time-continuous: y_c(t_{k-})=y_c(t_k))
```

The calling parameters of f are fixed: $y_{cd}=f(t,yc,yd,flag)$; this function must return either the derivative of the vector yc if $flag=0$ or the update of yd if $flag=1$.

`ycd=dot(yc)` must be a vector with same dimension as `yc` if `flag=0` and `ycd=update(yd)` must be a vector with same dimension as `yd` if `flag=1`.

`t` is a vector of instants where the solution `y` is computed.

`y` is the vector `y=[y(t(1)),y(t(2)),...]`. This function can be called with the same optional parameters as the `ode` function (provided `nd` and `stde1` are given in the calling sequence as second and third parameters). In particular integration flags, tolerances can be set. Optional parameters can be set by the `odeoptions` function.

An example for calling an external routine is given in directory `SCIDIR/default/fydot2.f`

External routines can be dynamically linked (see [link](#)).

Examples

```
//Linear system with switching input
deff('xdu=phis(t,x,u,flag)','if flag==0 then xdu=A*x+B*u; else xdu=1-u;end');
x0=[1;1];A=[-1,2;-2,-1];B=[1;2];u=0;nu=1;stde1=[1,0];u0=0;t=0:0.05:10;
xu=odedc([x0;u0],nu,stde1,0,t,phis);x=xu(1:2,:);u=xu(3,:);
nx=2;
plot2d1('onn',t',x',[1:nx],'161');
plot2d2('onn',t',u',[nx+1:nx+nu],'000');
//Fortran external( see fydot2.f):
norm(xu-odedc([x0;u0],nu,stde1,0,t,'phis'),1)

//Sampled feedback
//
//      |      xcdot=fc(t,xc,u)
// (system) |
//      |      y=hc(t,xc)
//
//
//      |      xd+=fd(xd,y)
// (feedback) |
//      |      u=hd(t,xd)
//
deff('xcd=f(t,xc,xd,iflag)',...
['if iflag==0 then '
'  xcd=fc(t,xc,e(t)-hd(t,xd));'
'else '
'  xcd=fd(xd,hd(t,xc));'
'end']);
A=[-10,2,3;4,-10,6;7,8,-10];B=[1;1;1];C=[1,1,1];
Ad=[1/2,1;0,1/20];Bd=[1;1];Cd=[1,1];
deff('st=e(t)','st=sin(3*t)')
deff('xdot=fc(t,x,u)','xdot=A*x+B*u')
deff('y=hc(t,x)','y=C*x')
deff('xp=fd(x,y)','xp=Ad*x + Bd*y')
deff('u=hd(t,x)','u=Cd*x')
h=0.1;t0=0;t=0:0.1:2;
x0c=[0;0;0];x0d=[0;0];nd=2;
xcd=odedc([x0c;x0d],nd,h,t0,t,f);
norm(xcd-odedc([x0c;x0d],nd,h,t0,t,'fcd1')) // Fast calculation (see fydot2.f)
plot2d([t',t',t'],xcd(1:3,:));
xset("window",2);plot2d2("gnn",[t',t'],xcd(4:5,:));
xset("window",0);
```



See Also

ode, odeoptions, csim, external

Name

odeoptions — set options for ode solvers

```
odeoptions()
```

Description

This function interactively displays a command which should be executed to set various options of ode solvers. The global variable %ODEOPTIONS sets the options.

CAUTION: the `ode` function checks if this variable exists and in this case it uses it. For using default values you should clear this variable. Note that `odeoptions` does not create this variable. To create it you must execute the command line displayed by `odeoptions`.

The variable %ODEOPTIONS is a vector with the following elements:

```
[itask, tcrit, h0, hmax, hmin, jactyp, mxstep, maxordn, maxords, ixpr, ml, mu]
```

The default value is:

```
[1, 0, 0, %inf, 0, 2, 500, 12, 5, 0, -1, -1]
```

The meaning of the elements is described below.

itask 1 : normal computation at specified times 2 : computation at mesh points (given in first row of output of `ode`) 3 : one step at one internal mesh point and return 4 : normal computation without overshooting **tcrit** 5 : one step, without passing **tcrit**, and return

tcrit assumes **itask** equals 4 or 5, described above

h0 first step tried

hmax max step size

hmin min step size

jactype 0 : functional iterations, no jacobian used ("adams" or "stiff" only) 1 : user-supplied full jacobian 2 : internally generated full jacobian 3 : internally generated diagonal jacobian ("adams" or "stiff" only) 4 : user-supplied banded jacobian (see **ml** and **mu** below) 5 : internally generated banded jacobian (see **ml** and **mu** below)

maxordn maximum non-stiff order allowed, at most 12

maxords maximum stiff order allowed, at most 5

ixpr print level, 0 or 1

ml, mu If **jactype** equals 4 or 5, **ml** and **mu** are the lower and upper half-bandwidths of the banded jacobian: the band is the i, j 's with $i - ml \leq j \leq ny - 1$. If **jactype** equals 4 the jacobian function must return a matrix **J** which is $ml + mu + 1 \times ny$ (where $ny = \dim$ of **y** in $\dot{y} = f(t, y)$) such that column 1 of **J** is made of **mu** zeros followed by $df_1/dy_1, df_2/dy_1, df_3/dy_1, \dots$ ($1 + ml$ possibly non-zero entries) column 2 is made of **mu**-1 zeros followed by $df_1/dx_2, df_2/dx_2, \dots$

See Also

`ode`

Part III. Elementary Functions

Name

abs — absolute value, magnitude

```
t=abs(x)
```

Parameters

x
real or complex vector or matrix

t
real vector or matrix

Description

`abs(x)` is the absolute value of the elements of `x`. When `x` is complex, `abs(x)` is the complex modulus (magnitude) of the elements of `x`.

Examples

```
abs([1,%i,-1,-%i,1+%i])
```

Name

`acos` — element wise cosine inverse

```
t=acos(x)
```

Parameters

`x`
real or complex vector

`t`
real or complex vector

Description

The components of vector `t` are cosine inverse of the corresponding entries of vector `x`. Definition domain is $[-1, 1]$.

Examples

```
x=[1,%i, -1, -%i]  
cos(acos(x))
```

Name

`acosd` — element wise cosine inverse, result in degree.

```
t=acosd(x)
```

Parameters

`x`
Real or complex array.

`t`
Real or complex array.

Description

The components of vector `t` are cosine inverse, in degree, of the corresponding entries of vector `x` (`t=acos(x)*180/%pi`). For real data in `[-1, 1]`. The results are real in `[0 180]`.

Examples

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];  
acosd(x)
```

See Also

`acos`

Name

acosh — hyperbolic cosine inverse

```
[ t ]=acosh(x)
```

Parameters

x
real or complex vector

t
real or complex vector

Description

the components of vector t are the ArgCosh of the corresponding entries of vector x . Definition domain is $]1,+\infty[$.

Examples

```
x=[0,1,%i];  
cosh(acosh(x))
```

Name

acoshm — matrix hyperbolic inverse cosine

```
t=acoshm(x)
```

Parameters

x,t
real or complex square matrix

Description

acoshm is the matrix hyperbolic inverse cosine of the matrix x. Uses the formula $t = \logm(x + (x + \text{eye}()) * \text{sqrtm}((x - \text{eye}()) / (x + \text{eye}())))$ For non symmetric matrices result may be inaccurate.

Examples

```
A=[1,2;3,4];  
coshm(acoshm(A))  
A(1,1)=A(1,1)+%i;  
coshm(acoshm(A))
```

See Also

acosh, logm, sqrtm

Name

acosm — matrix wise cosine inverse

```
t=acosm(x)
```

Parameters

x
real or complex square matrix

t
real or complex square matrix

Description

t are cosine inverse of the x matrix. Diagonalization method is used. For nonsymmetric matrices result may be inaccurate. One has $t = -i \cdot \logm(x + i \cdot \sqrt{m(\text{eye}() - x \cdot x)})$

Examples

```
A=[1,2;3,4];  
cosm(acosm(A))
```

See Also

acos, sqrtm, logm

Name

`acot` — computes the element-wise inverse cotangent of the argument.

```
y = acot(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element-wise inverse cotangent of the argument. For real argument the result is real.

The following equalities hold: $\text{acot}(z) = \pi - \text{acot}(-z) = \pi/2 - \text{atan}(z) = i \cdot \text{acoth}(i \cdot z) + \pi/2 \cdot (1 - \text{csgn}(z + i))$

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];  
acot(x)/%pi
```

See Also

`cotg`, `acotd`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joint IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`acotd` — computes the element-wise inverse cotangent of the argument result in degree.

```
y = acotd(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element-wise inverse cotangent of the argument and returns the results in degree. For real argument `w` the result is real.

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];  
acotd(x)
```

See Also

`cotd`, `acot`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`acoth` — element wise hyperbolic cotangent inverse.

```
y = acoth(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element wise hyperbolic cotangent inverse of the argument.

Examples

```
x=-30:0.1:30;  
x(abs(x)<=1)=%nan;  
plot(x,acoth(x))
```

See Also

`atanh`, `coth`

Authors

Serge Steer, INRIA

Used Functions

this function is based on the `atanh` function.

Name

`acsc` — computes the element-wise inverse cosecant of the argument.

```
y = acsc(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element-wise inverse cotsecant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: $\text{acsc}(z) = -\text{acsc}(-z) = \text{asin}(1/z) = \pi/2 - \text{asec}(z) = i \cdot \text{acsch}(i \cdot z)$

Examples

```
x=linspace(1,20,200);  
x=[-x($:-1:1) %nan x];  
plot(x,acsc(x))
```

See Also

`csc`, `acscd`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`acscd` — computes the element-wise inverse cosecant of the argument, results in degree.

```
y = acscd(x)
```

Parameters

`x`
Real array.

`y`
Real or complex array.

Description

Computes the element-wise inverse cosecant of the argument and return the results in degree. For real argument with absolute value greater than 1 the result is real.

Examples

```
x=linspace(1,20,200);  
x=[-x($:-1:1) %nan x];  
plot(x,acscd(x))
```

See Also

`cscd`, `acsc`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`acsch` — computes the element-wise inverse hyperbolic cosecant of the argument.

```
y = acsch(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element-wise inverse hyperbolic cotsecant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: $\operatorname{acsch}(z) = -\operatorname{acsch}(-z) = \operatorname{asinh}(1/z) = \operatorname{csgn}(i+1/z) * \operatorname{asech}(-i*z) - i*\pi/2 = i*\operatorname{acsc}(i*z)$

Examples

```
x=linspace(1,20,200);  
x=[-x($:-1:1) %nan x];  
plot(x,acsch(x))
```

See Also

`csch`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

adj2sp — converts adjacency form into sparse matrix.

Parameters

xadj

integer vector of length (n+1).

adjncy

integer vector of length nz containing the row indices for the corresponding elements in anz

anz

column vector of length nz, containing the non-zero elements of A

mn

row vector with 2 entries, mn=size(A) (optional).

A

real or complex sparse matrix (nz non-zero entries)

Description

sp2adj converts an adjacency form representation of a matrix into its standard Scilab representation (utility fonction).
xadj, adjncy, anz = adjacency representation of A i.e:

xadj(j+1) - xadj(j) = number of non zero entries in row j. adjncy = column index of the non zeros entries in row 1, row 2,..., row n. anz = values of non zero entries in row 1, row 2,..., row n. xadj is a (column) vector of size n+1 and adjncy is an integer (column) vector of size nz=nnz(A). anz is a real vector of size nz=nnz(A).

Examples

```
A = sprand(100,50,.05);  
[xadj,adjncy,anz]= sp2adj(A);  
[n,m] = size(A);  
p = adj2sp(xadj,adjncy,anz,[n,m]);  
A-p
```

See Also

sp2adj, spcompact

Name

amell — Jacobi's am function

```
[sn]=amell(u,k)
```

Parameters

u
real scalar or vector

k
scalar

sn
real scalar or vector

Description

Computes Jacobi's elliptic function $\text{am}(u, k)$ where k is the parameter and u is the argument. If u is a vector sn is the vector of the (element wise) computed values. Used in function %sn.

See Also

delip , %sn , %asn

Name

`and` — logical and of the elements of an array

```
b=and(A), b=and(A, '*')  
b=and(A, 'r'), b=and(A, 1)  
b=and(A, 'c'), b=and(A, 2)
```

Description

`and(A)` is the logical AND of elements of the boolean matrix `A`. `and(A)` returns %T ("true") iff all entries of `A` are %T.

`y=and(A, 'r')` (or, equivalently, `y=and(A, 1)`) is the rowwise and. It returns in each entry of the row vector `y` the and of the rows of `x` (The and is performed on the row index : `y(j)=and(A(i, j), i=1, m)`).

`y=and(A, 'c')` (or, equivalently, `y=and(A, 2)`) is the columnwise and. It returns in each entry of the column vector `y` the and of the columns of `x` (The and is performed on the column index: `y(i)=and(A(i, j), j=1, n)`)).

See Also

not, and operator (&) , or

Name

& — logical and operator

A&B

Description

A&B gives the element-wise logical and of the booleans matrices A and B .A and B must be matrices with the same dimensions or one from them must be a single boolean.

See Also

not, and, or operator (|)

Name

asec — computes the element-wise inverse secant of the argument.

```
y = asec(x)
```

Parameters

x
Real or complex array.

y
Real or complex array.

Description

Computes the element-wise inverse secant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: $\text{asec}(z) = -\text{acsc}(-z) = \text{asin}(1/z) = \pi/2 - \text{asec}(x)$
 $= i * \text{acsch}(i * z)$

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];  
asec(x)/%pi
```

See Also

sec, asecd

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`asecd` — computes the element-wise inverse secant of the argument, results in degree.

```
y = asecd(x)
```

Parameters

`x`
Real or complex array

`y`
Real or complex array

Description

Computes the element-wise inverse secant of the argument, results in degree. For real input data with absolute value greater than 1 the results are real and in $[-90\ 90]$.

`asecd(x)` is equal to `asec(x)*180/%pi`.

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];  
asecd(x)
```

See Also

`asec`, `secd`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

`asech` — computes the element-wise inverse hyperbolic secant of the argument.

```
y = asech(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Computes the element-wise inverse hyperbolic secant of the argument. If the argument is real and have absolute value less than one the result is real.

The following equalities hold: $\text{asech}(x) = \text{acosh}(1 ./ x) = i * \text{csgn}(i * (1 - 1 ./ x)) * \text{asec}(x) = \text{csgn}(i * (1 - 1 ./ x)) * (\pi/2 * (i + \text{acsch}(i * x)))$

Examples

```
asech(1)
```

See Also

`sech`

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joining IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

asin — sine inverse

```
[t]=asin(x)
```

Parameters

x
real or complex vector/matrix

t
real or complex vector/matrix

Description

The entries of t are sine inverse of the corresponding entries of x . Definition domain is $[-1, 1]$.

Examples

```
A=[1,2;3,4]
sin(asin(A))
```

See Also

sin, sinm, asinm

Name

asind — sine inverse, results in degree

```
t=asind(x)
```

Parameters

x
real vector/matrix. Elements must be in $[-1 \ 1]$.

t
real vector/matrix with same dimensions as **x**

Description

The entries of **t** are sine inverse of the corresponding entries of **x**. Definition domain is $[-1, 1]$. The results are in $[-90 \ 90]$;

Examples

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];  
asind(x)
```

See Also

sin, sind, asinm

Name

`asinh` — hyperbolic sine inverse

```
[t]=asinh(x)
```

Parameters

`x`
real or complex vector/matrix

`t`
real or complex vector/matrix

Description

The entries of `t` are the hyperbolic sine inverse of the corresponding entries of `x`. Definition domain is $]-1,i[$.

Examples

```
A=[1,2;2,3]
sinh(asinh(A))
```

Name

asinhm — matrix hyperbolic inverse sine

```
t=asinhm(x)
```

Parameters

x,t
real or complex square matrix

Description

asinhm is the matrix hyperbolic inverse sine of the matrix x. Uses the formula $t = \logm(x + \sqrt{tm(x*x + \text{eye}())})$. Results may be not reliable for non-symmetric matrix.

Examples

```
A=[1,2;2,3]
sinhm(asinhm(A))
```

See Also

asinh, logm, sqrtm

Name

asinm — matrix wise sine inverse

```
t=asinm(x)
```

Parameters

x
real or complex square matrix

t
real or complex square matrix

Description

t are sine inverse of the x matrix. Diagonalization method is used. For non symmetric matrices result may be inaccurate.

Examples

```
A=[1,2;3,4]
sinm(asinm(A))
asinm(A)+%i*logm(%i*A+sqrtm(eye()-A*A))
```

See Also

asin, sinm

Name

atan — 2-quadrant and 4-quadrant inverse tangent

```
phi=atan(x)
phi=atan(y,x)
```

Parameters

x
real or complex scalar, vector or matrix

phi
real or complex scalar, vector or matrix

x, y
real scalars, vectors or matrices of the same size

phi
real scalar, vector or matrix

Description

The first form computes the 2-quadrant inverse tangent, which is the inverse of `tan(phi)`. For real `x`, `phi` is in the interval $(-\pi/2, \pi/2)$. For complex `x`, `atan` has two singular, branching points $\pm i$, $-i$ and the chosen branch cuts are the two imaginary half-straight lines $[i, i\infty)$ and $(-\infty, -i]$.

The second form computes the 4-quadrant arctangent (`atan2` in Fortran), this is, it returns the argument (angle) of the complex number $x+i*y$. The range of `atan(y, x)` is $(-\pi, \pi]$.

For real arguments, both forms yield identical values if `x>0`.

In case of vector or matrix arguments, the evaluation is done element-wise, so that `phi` is a vector or matrix of the same size with `phi(i,j)=atan(x(i,j))` or `phi(i,j)=atan(y(i,j),x(i,j))`.

Examples

```
// examples with the second form
x=[1,%i, -1,%i]
phases=atan(imag(x),real(x))
atan(0, -1)
atan(-%eps, -1)

// branch cuts
atan(-%eps + 2*%i)
atan(+%eps + 2*%i)
atan(-%eps - 2*%i)
atan(+%eps - 2*%i)

// values at the branching points
ieee(2)
atan(%i)
atan(-%i)
```

See Also

tan, ieee

Authors

B.P.

L.V.D. (authors of the complex atan function).

Name

`atand` — 2-quadrant and 4-quadrant element-wise inverse tangent, result in degree.

```
phi=atand(x)
phi=atand(y,x)
```

Parameters

`x`
real scalar, vector or matrix

`phi`
real scalar, vector or matrix

`x, y`
real scalars, vectors or matrices of the same size

`phi`
real scalar, vector or matrix

Description

The first form computes the 2-quadrant inverse tangent, which is the inverse of `tand(phi)`. The `phi` elements are in the interval $[-90, 90]$.

The second form computes the 4-quadrant arctangent (`atan2` in Fortran), this is, it returns the argument (angle) of the complex number $x+i*y$. The range of `atand(y, x)` is $[-180, 180i]$.

Both forms yield identical values if $x>0$.

Examples

```
// example with the second form
x=[0,1/sqrt(3),1,sqrt(3),%inf,0]
atand(x)
```

See Also

`tan`, `tand`

Authors

Serge Steer, INRIA

Name

atanh — hyperbolic tangent inverse

```
t=atanh(x)
```

Parameters

x
real or complex vector/matrix

t
real or complex vector/matrix

Description

The components of vector **t** are the hyperbolic tangent inverse of the corresponding entries of vector **x**. Definition domain is $[-1, 1]$ for the real function (see Remark).

Remark

In Scilab (as in some others numerical software) when you try to evaluate an elementary mathematical function outside its definition domain in the real case, then the complex extension is used (with a complex result). The more famous example being the `sqrt` function (try `sqrt(-1)` !). This approach have some drawbacks when you evaluate the function at a singular point which may led to different results when the point is considered as real or complex. For the `atanh` this occurs for -1 and 1 because the at these points the imaginary part do not converge and so `atanh(1) = +Inf + i NaN` while `atanh(1) = +Inf` for the real case (as $\lim_{x \rightarrow 1^-} \text{atanh}(x)$). So when you evaluate this function on the vector `[1 2]` then like 2 is outside the definition domain, the complex extension is used for all the vector and you get `atanh(1) = +Inf + i NaN` while you get `atanh(1) = +Inf` with `[1 0.5]` for instance.

Examples

```
// example 1
x=[0,%i,-%i]
tanh(atanh(x))

// example 2
x = [-%inf -3 -2 -1 0 1 2 3 %inf]
ieee(2)
atanh(tanh(x))

// example 3 (see Remark)
ieee(2)
atanh([1 2])
atanh([1 0.5])
```

See Also

tanh, ieee

Name

atanhm — matrix hyperbolic tangent inverse

```
t=atanhm(x)
```

Parameters

x
real or complex square matrix

t
real or complex square matrix

Description

`atanhm(x)` is the matrix hyperbolic tangent inverse of matrix `x`. Results may be inaccurate if `x` is not symmetric.

Examples

```
A=[1,2;3,4];  
tanhm(atanhm(A))
```

See Also

`atanh`, `tanhm`

Name

atanm — square matrix tangent inverse

```
[t]=atanm(x)
```

Parameters

x
real or complex square matrix

t
real or complex square matrix

Description

`atanm(x)` is the matrix arctangent of the matrix `x`. Result may be not reliable if `x` is not symmetric.

Examples

```
tanm(atanm([1,2;3,4]))
```

See Also

`atan`

Name

base2dec — conversion from base b representation to integers

```
d=base2dec(s,b)
```

Parameters

- d
matrix of integers
- s
matrix of character strings corresponding to base b representation
- b
integer, the base

Description

base2dec(x,b) returns the matrix of numbers corresponding to the base b representation.

Examples

```
base2dec(['ABC','0','A'],16)
```

See Also

bin2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

bin2dec — integer corresponding to a binary form

```
y=bin2dec(str)
```

Parameters

str

a string or a vector/matrix of strings containing only characters '1' and '0'

y

a scalar or a vector/matrix of positives integers

Description

Given str a binary string, this function returns y the decimal number whose the binary representation is given by str (y and str have the same size).

Examples

```
// example 1 :  
// '1010110' : is the binary representation of 86  
str='1010110';  
y=bin2dec(str)  
  
// example 2 :  
// '1011011' : is the binary representation of 91  
// '1010010' : is the binary representation of 82  
str=['1011011'; '1010010']  
y=bin2dec(str)
```

See Also

base2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

binomial — binomial distribution probabilities

```
pr=binomial(p,n)
```

Parameters

pr
row vector with n+1 components

p
real number in [0,1]

n
an integer >= 1

Description

`pr=binomial(p,n)` returns the binomial probability vector, i.e. `pr(k+1)` is the probability of `k` success in `n` independent Bernoulli trials with probability of success `p`. In other words : `pr(k+1) = probability(X=k)` , with `X` a random variable following the `B(n,p)` distribution, and numerically :

$$\text{pr}(k+1) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \text{ with } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Examples

```
// first example
n=10;p=0.3; clf(); plot2d3(0:n,binomial(p,n));

// second example
n=50;p=0.4;
mea=n*p; sigma=sqrt(n*p*(1-p));
x=( 0:n)-mea )/sigma;
clf()
plot2d(x, sigma*binomial(p,n));
deff('y=Gauss(x)', 'y=1/sqrt(2*pi)*exp(-(x.^2)/2)')
plot2d(x, Gauss(x), style=2);

// by binomial formula (Caution if big n)
function pr=binomial2(p,n)
x=poly(0, 'x');pr=coeff((1-p+x)^n).*horner(x^(0:n),p);
endfunction
p=1/3;n=5;
binomial(p,n)-binomial2(p,n)

// by Gamma function: gamma(n+1)=n! (Caution if big n)
p=1/3;n=5;
Cnks=gamma(n+1)./(gamma(1:n+1).*gamma(n+1:-1:1));
x=poly(0, 'x');
pr=Cnks.*horner(x.^(0:n).*(1-x)^(n:-1:0),p);
pr-binomial(p,n)
```

See Also

[cdfbin](#), [grand](#)

Name

bitand — AND applied to binary representation of input arguments

```
[z]=bitand(x,y)
```

Parameters

x
scalar/vector/matrix of positives integers

y
scalar/vector/matrix of positives integers

z
scalar/vector/matrix of positives integers

Description

Given x and y two positives integers, this function returns z the decimal number whose the binary form is the AND of the binary representations of x and y. (X, y, z have the same size. If dimension of x (and y) is superior than 1 then $z(i)$ is equal to $\text{bitand}(x(i), y(i))$).

Examples

```
// example 1 :  
// '1010110' : is the binary representation of 86  
// '1011011' : is the binary representation of 91  
// '1010010' : is the binary representation for the AND of binary representation  
// so the decimal number corresponding to the AND applied to binary forms 86 and 91 is 82  
x=86; y=91  
z=bitand(x,y)  
  
// example 2 :  
x=[12,45],y=[25,49]  
z=bitand(x,y)
```

See Also

bitor, bin2dec, dec2bin

Name

bitor — OR applied to binary representation of input arguments

```
[z]=bitor(x,y)
```

Parameters

x
scalar/vector/matrix of positives integers

y
scalar/vector/matrix of positives integers

z
scalar/vector/matrix of positives integers

Description

Given x and y two positives integers, this function returns z the decimal number whose the binary form is the OR of the binary representations of x and y (x, y and z have the same size). If dimension of x is superior than 1 then $z(i)$ is equal to $\text{bitor}(x(i), y(i))$.

Examples

```
// example 1 :  
// '110000' : is the binary representation of 48  
// '100101' : is the binary representation of 37  
// '110101' : is the binary representation for the OR applied to the binary forms  
// so the decimal number corresponding to the OR applied to binary forms 48 and 37  
x=48; y=37  
z=bitor(x,y)  
  
// example 2 :  
x=[12,45]; y=[25,49]  
z=bitor(x,y)
```

See Also

bitand, bin2dec, dec2bin

Name

bloc2exp — block-diagram to symbolic expression

```
[str]=bloc2exp(blocd)
[str,names]=bloc2exp(blocd)
```

Parameters

blocd
list

str
string

names
string

Description

given a block-diagram representation of a linear system `bloc2exp` returns its symbolic evaluation. The first element of the list `blocd` must be the string `'blocd'`. Each other element of this list (`blocd(2), blocd(3), ...`) is itself a list of one the following types :

```
list('transfer','name_of_linear_system')
```

```
list('link','name_of_link',
      [number_of_upstream_box,upstream_box_port],
      [downstream_box_1,downstream_box_1_portnumber],
      [downstream_box_2,downstream_box_2_portnumber],
      ...)
```

The strings `'transfer'` and `'links'` are keywords which indicate the type of element in the block diagram.

Case 1 : the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (`syslin` list) or in transfer form (matrix of rationals).

To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2 : the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair `[number_of_upstream_box,upstream_box_port]`, to different inputs of other blocks. Each such input is represented by the pair `[downstream_box_i,downstream_box_i_portnumber]`.

The different elements of a block-diagram can be defined in an arbitrary order.

For example

[1] `S1*S2` with unit feedback.

There are 3 transfers `S1` (number `n_s1=2`), `S2` (number `n_s2=3`) and an adder (number `n_add=4`) with symbolic transfer function `['1','1']`.

There are 4 links. The first one (named 'U') links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one link respectively (output)port 1 of the adder to (input)port 1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named 'Y') links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.

```
//Initialization
syst=list('blocd'); l=1;

//Systems
l=l+1;n_s1=l;syst(l)=list('transfer','S1'); //System 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2'); //System 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']); //adder

//Links

// Inputs -1 --> input 1
l=l+1;syst(l)=list('link','U',[-1],[n_adder,1]);
// Internal
l=l+1;syst(l)=list('link','',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link','',[n_s1,1],[n_s2,1]);

// Outputs // -1 -> output 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);

//Evaluation call
w=bloc2exp(syst);
```

The result is the character string: $w = -(s_2 * s_1 - \text{eye}()) \backslash s_2 * s_1$.

Note that invoked with two output arguments, `[str,names]= blocd(syst)` returns in `names` the list of symbolic names of named links. This is useful to set names to inputs and outputs.

[2] second example

```
//Initialization
syst=list('blocd'); l=1;

//System (2x2 blocks plant)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']));

//Controller
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);

//Evaluation call
w=bloc2exp(syst);
```

In this case the result is a formula equivalent to the usual one:

$P_{11} + P_{12} * \text{invr}(\text{eye}() - K * P_{22}) * K * P_{21};$

See Also

[bloc2ss](#)

Authors

S. S., F. D. (INRIA)

Name

bloc2ss — block-diagram to state-space conversion

```
[sl]=bloc2ss(blocd)
```

Parameters

blocd
list

sl
list

Description

Given a block-diagram representation of a linear system `bloc2ss` converts this representation to a state-space linear system. The first element of the list `blocd` must be the string `'blocd'`. Each other element of this list is itself a list of one the following types :

```
list('transfer','name_of_linear_system')
```

```
list('link','name_of_link',  
      [number_of_upstream_box,upstream_box_port],  
      [downstream_box_1,downstream_box_1_portnumber],  
      [downstream_box_2,downstream_box_2_portnumber],  
      ...)
```

The strings `'transfer'` and `'links'` are keywords which indicate the type of element in the block diagram.

Case 1 : the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (`syslin` list) or in transfer form (matrix of rationals).

To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2 : the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair `[number_of_upstream_box,upstream_box_port]`, to different inputs of other blocks. Each such input is represented by the pair `[downstream_box_i,downstream_box_i_portnumber]`.

The different elements of a block-diagram can be defined in an arbitrary order.

For example

[1] `S1*S2` with unit feedback.

There are 3 transfers `S1` (number `n_s1=2`), `S2` (number `n_s2=3`) and an adder (number `n_add=4`) with symbolic transfer function `['1','1']`.

There are 4 links. The first one (named `'U'`) links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one link respectively (output)port 1 of the adder to (input)port

1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named 'Y') links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.

```
//Initialization
syst=list('blocd'); l=1;

//Systems
l=l+1;n_s1=l;syst(l)=list('transfer','S1'); //System 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2'); //System 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1'])); //adder

//Links
// Inputs -1 --> input 1
l=l+1;syst(l)=list('link','U1',[-1],[n_adder,1]);

// Internal
l=l+1;syst(l)=list('link','',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link','',[n_s1,1],[n_s2,1]);

// Outputs // -1 -> output 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);
```

With `s=poly(0,'s');` `S1=1/(s+1);S2=1/s;` the result of the evaluation call `sl=bloc2ss(syst);` is a state-space representation for $1/(s^2+s-1)$.

[2] LFT example

```
//Initialization
syst=list('blocd'); l=1;

//System (2x2 blocks plant)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']));

//Controller
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);
```

With

```
P=syslin('c',A,B,C,D);
P11=P(1,1);
P12=P(1,2);
P21=P(2,1);
P22=P(2,2);
K=syslin('c',Ak,Bk,Ck,Dk);
```

`bloc2exp(syst)` returns the evaluation the lft of P and K.

See Also

[bloc2exp](#)

Authors

S. S., F. D. (INRIA)

Name

cat — concatenate several arrays

```
y=cat(dims,A1,A2,...,An)
```

Parameters

dims

a positive real scalar.

A1,A2,...An

scalars, vectors, matrices or mutlti-arrays, or cells arrays. A1,A2,...,An must have the same size (excluding the dimension number dims). size(A1,i)=size(A2,i)=...=size(An,i) for i different of dims and size(A1,dims), size(A2,dims),...,size(An,dims) can be different.

y

a scalar, vector, matrix or mutlti-array, y has the same type than A1,A2,...,An.

Description

`y=cat(dims,A1,A2,...,An)` : y is the result of the concatenation of the input arguments A1,A2,...,An. if dims=1 then the concatenation is done according to the rows : if dims=2 then concatenation is done according to the columns of the input arguments,...

if dims=1, then the concatenation is done according to the rows

A1=[1 2 3 ; 4 5 6]; A2=[7 8 9 ; 10 11 12]; y=cat(1,A1,A2) => y=[1 2 3 ; 4 5 6 ; 7 8 9 ; 10 11 12]

if dims=2, then the concatenation is done according to the columns of the input arguments

A1=[1 2 3;4 5 6]; A2=[7 8 9;10 11 12]; y=cat(2,A1,A2) => y=[1 2 3 7 8 9 ; 4 5 6 10 11 12]

Examples

```
// first example : concatenation according to the rows
dims=1; A1=[1 2 3]; A2=[4 5 6 ; 7 8 9]; A3=[10 11 12]; y=cat(dims,A1,A2,A3)

// second example : concatenation according to the columns
dims=2; A1=[1 2 3]'; A2=[4 5;7 8;9 10]; y=cat(dims,A1,A2)

// third example : concatenation according to the 3th dimension
dims=3; A1=matrix(1:12,[2,2,3]); A2=[13 14;15 16]; A3=matrix(21:36,[2,2,4]); y=
```

See Also

permute, matrix

Authors

Farid Belahcene

Name

ceil — rounding up

```
[y]=ceil(x)
```

Parameters

x

a real matrix

y

integer matrix

Description

`ceil(x)` returns an integer matrix made of rounded up elements.

Examples

```
ceil([1.9 -2.5])-[2,-2]  
ceil(-%inf)  
x=rand()*10^20;ceil(x)-x
```

See Also

round, floor, int

Name

`cell2mat` — convert a cell array into a matrix

```
x=cell2mat(c)
```

Parameters

`c`
cell, the components of `c` must have the same type and can be scalars or matrices

`x`
matrix

Description

Returns a matrix which is the concatenation of all components of the cell `c`.

`cell2mat(c)`
all components of `c` must have the same data type (strings or doubles or integers or booleans). For each row `i` of `c`, `cell2mat` concatenates all the components of the `i`th row of the cell `c`

Note that if the components of the cell input `c` are strings then `cell2mat(c)` returns a column vector of strings concatenation.

Examples

```
c=makecell([2,2],[1 2 3; 6 7 8],[4 5;9 10],[11 12;16 17],[14 13 15;18 19 20])  
cell2mat(c)
```

See Also

`cell`

Name

`cellstr` — convert strings vector (or strings matrix) into a cell of strings

```
c=cellstr(s)
```

Parameters

`s`
strings vector, or strings matrix

Description

returns a cell array of strings

- If `s` is a line vector of strings then `cellstr(s)` returns a (one-by-one) cell which contains one component (the concatenation of all columns components of `s`)
- If `s` is a column vector of strings then `cellstr(s)` convert `s` into a cell which have the same size : (size(`s`,1)-by-one) cell of strings
- If `s` is a matrix of strings then for each row `i` of `s`, `cellstr(s)` concatenates all the components of the `i`th rows of the matrix `s` (i.e `s(i,1)`, `s(i,2)`, `s(i,3)`,...) and returns a (size(`s`,1)-by-one) cell of strings

Examples

```
cellstr("foo")  
cellstr(["sci","lab"])  
cellstr(["abc","def",'gh';"i","j","klm"])
```

See Also

`cell`, `string`

Name

char — char function

```
y=char( x)
y=char(st1,st2,st3,...)
```

Parameters

x
a cell of strings arrays, or an array of ascii codes

st1,st2,st3
strings arrays

y:
a strings vector(column)

Description

One input argument :

Given a cell of strings arrays **x**, this function returns a strings vector **y** in which the rows are the components of the strings cell

Given an array of ascii codes **x**, this function returns a strings array **y** corresponding into ascii codes. If `dims(x)=[n1,n2,n3,n4,...]`, then returned value have same size as input value instead of second dims, `dims(y)=[n1,n3,n4,..]`

More than one input argument :

Given strings arrays **st1,st2,st3,...**, this function returns a vector of strings in which the rows are the components of **st1,st2,st3,...**. In the vector **y** the length of all strings **sti** is completed by blanks, to have the same length than the `lengthmax` of **sti**.

Examples

```
//Example with a hypermatrix of ascii codes :
x=hypermat([4,2,3],61:84);
y=char(x)
size(x)
size(y)

//Example with more than one argument :
st1="zeros";
st2=["one","two"];
st3=["three"];
y=char(st1,st2,st3)
size(y)

//all strings rows are completed by 'blanks' to have the same length : 6
length(y)
```

See Also

[asciimat](#)

Authors

F.B

Name

conj — conjugate

```
[y]=conj(x)
```

Parameters

x,y
real or complex matrix.

Description

conj(x) is the complex conjugate of x.

Examples

```
x=[1+%i, -%i;%i, 2*%i];  
conj(x)  
x'-conj(x) //x' is conjugate transpose
```

Name

cos — cosine function

```
[y]=cos(x)
```

Parameters

x
real or complex vector/matrix

Description

For a vector or a matrix, `COS(x)` is the cosine of its elements . For matrix cosine use `COSM(X)` function.

Examples

```
x=[0,1,%i]  
acos(cos(x))
```

See Also

`cosm`

Name

`cosd` — element-wise cosine function, argument in degree

```
y=cosd(x)
```

Parameters

`x`
real vector/matrix

Description

For a vector or a matrix `x` of angles given in degree, `cosd(x)` is the cosine of its elements. The results are in `[-1 1]`. For input elements which are equal to $n \cdot 90$ with n integer and odd, the result is exactly zero.

Examples

```
x=[0,30 45 60 90 360];  
cosd(x)
```

See Also

`cos`

Authors

Serge Steer, INRIA

Name

cosh — hyperbolic cosine

```
[t]=cosh(x)
```

Parameters

x,t
real or complex vectors/matrices

Description

The elements of **t** are the hyperbolic cosine of the corresponding entries of vector **x**.

Examples

```
x=[0,1,%i]  
acosh(cosh(x))
```

See Also

cos, acosh

Name

coshm — matrix hyperbolic cosine

```
t=coshm(x)
```

Parameters

x,t
real or complex square matrix

Description

coshm is the matrix hyperbolic cosine of the matrix x. $t = (\expm(x) + \expm(-x))/2$. Result may be inaccurate for nonsymmetric matrix.

Examples

```
A=[1,2;2,4]
acoshm(coshm(A))
```

See Also

cosh, expm

Name

cosm — matrix cosine function

```
t=cosm(x)
```

Parameters

x
real or complex square matrix

Description

$\text{cosm}(x)$ is the matrix cosine of the x matrix. $t=0.5*(\text{expm}(\%i*x)+\text{expm}(-\%i*x))$.

Examples

```
A=[1,2;3,4]
cosm(A)-0.5*(expm(%i*A)+expm(-%i*A))
```

See Also

cos, expm

Name

cotd — cotangent

```
y=cotd(x)
```

Parameters

x

Real array.

y

Real array with same dimensions as x.

Description

The entries of y are the cotangents of the corresponding entries of x supposed to be given in degree. $t = \cos(x) ./ \sin(x)$. For entries equal to $n \cdot 180$ with n integers the results are infinite, whereas $\cotg(n \cdot \pi)$ is large but finite because π cannot be represented exactly. For entries equal to $n \cdot 90$ with n integers and odd the results are exactly 0.

Examples

```
x=[30 45 60 90];  
cotd(x)
```

See Also

[cotg](#)

Authors

Serge Steer, INRIA

Name

cotg — cotangent

```
[t]=cotg(x)
```

Parameters

x,t
real or complex vectors/matrices

Description

The elements of **t** are the cotangents of the corresponding entries of **x**. $t=\cos(x) ./ \sin(x)$

Examples

```
x=[1,%i];  
cotg(x)-cos(x)./sin(x)
```

See Also

tan

Name

`coth` — hyperbolic cotangent

```
[t]=coth(x)
```

Description

the elements of vector `t` are the hyperbolic cotangent of elements of the vector `x`.

Examples

```
x=[1,2*i]
t=exp(x);
(t-ones(x)./t).\ (t+ones(x)./t)
coth(x)
```

See Also

`cotg`

Name

cothm — matrix hyperbolic cotangent

```
[t]=cothm(x)
```

Description

cothm(x) is the matrix hyperbolic cotangent of the square matrix x.

Examples

```
A=[1,2;3,4];  
cothm(A)
```

See Also

coth

Name

`csc` — Computes the element-wise cosecant of the argument.

```
y = csc(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array with same dimensions as `x`.

Description

Computes the element-wise cosecant of the argument. The cosecant is a periodic function defined as $1/\sin$. For real data the results are real and in $]-\infty, -1] \cup [1, \infty[$.

Examples

```
x=linspace(0.01,%pi-0.01,200)
clf();
plot(-x,csc(-x),x,csc(x))
```

See Also

`sec`, `cscd`

Authors

Serge Steer, INRIA

Name

`cscd` — Computes the element-wise cosecant of the argument given in degree.

```
x = cscd(x)
```

Parameters

`x`
Real or complex array.

`x`
Real or complex array.

Description

The entries of `y` are the cosecant $1/\sin$ of the entries of `x` given in degree. The results are real and in $]-\infty, -1] \cup [1, \infty[$. or entries equal to $n \cdot 180$ with n integer, the result is infinite (or an error depending on ieee mode). For entries equal to $n \cdot 90$ with n integer and odd the result is exactly 1 or -1.

Examples

```
csc(%pi/4)
cscd(90)
```

See Also

`secd`, `csc`, `sind`

Authors

Serge Steer, INRIA

Name

`csch` — Computes the element-wise hyperbolic cosecant of the argument.

```
y = csch(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array with same dimensions as `x`.

Description

Computes the element-wise hyperbolic cosecant of the argument. For real data the results are real.

Examples

```
x=linspace(0.01,4,200);x=[-x($:-1:1) %nan x];  
clf();  
plot(x,csch(x))
```

See Also

`csc`, `acsch`

Authors

Serge Steer, INRIA

Nom

`csgn` — Returns the sign of a vector of real or complex values.

```
s = csgn(z)
```

Parameters

`z`

The vector of values on which we want to compute the sign.

`s`

If the real part is not equal to zero:

- +1 if the real part of an element is positive
- -1 if the real part of an element is negative

If the real part is equal to zero:

- +1 if the imaginary part of an element is positive
- -1 if the imaginary part of an element is negative

if the element is equal to zero, then returns %nan

Description

Returns the sign of a vector of real or complex values.

Examples

```
A = [1 1+%i 0 -1 1-%i -1-%i];  
disp(csgn(A))
```

Authors

Y. Collette

Name

cumprod — cumulative product

```
y=cumprod(x)
y=cumprod(x, 'r') or y=cumprod(x,1)
y=cumprod(x, 'c') or y=cumprod(x,2)
y=cumprod(x, 'm')
```

Parameters

x
vector or matrix (real or complex)

y
vector or matrix (real or complex)

Description

For a vector or a matrix **x**, **y=cumprod(x)** returns in **y** the cumulative product of all the entries of **x** taken columnwise.

y=cumprod(x, 'c') (or, equivalently, **y=cumprod(x,2)**) returns in **y** the cumulative elementwise product of the columns of **x**: **y(i,:)=cumprod(x(i,:))**

y=cumprod(x, 'r') (or, equivalently, **y=cumprod(x,1)**) returns in **y** the cumulative elementwise product of the rows of **x**: **y(:,i)=cumprod(x(:,i))**.

y=cumprod(x, 'm') is the cumulative product along the first non singleton dimension of **x** (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];

cumprod(A)
cumprod(A, 'r')
cumprod(A, 'c')

rand('seed',0);
a=rand(3,4);
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;
    w(k,:)=w(k-1,:).*a(k,:);
end;
w-cumprod(a, 'r')
```

See Also

cumsum, sum, prod

Name

cumsum — cumulative sum

```
y=cumsum(x)
y=cumsum(x, 'r') or y=cumsum(x,1)
y=cumsum(x, 'c') or y=cumsum(x,2)
```

Parameters

x
vector or matrix (real or complex)

y
vector or matrix (real or complex)

Description

For a vector or a matrix **x**, **y=cumsum(x)** returns in **y** the cumulative sum of all the entries of **x** taken columnwise.

y=cumsum(x, 'c') (or, equivalently, **y=cumsum(x, 2)**) returns in **y** the cumulative sum of the columns of **x**: **y(i, :)=cumsum(x(i, :))**

y=cumsum(x, 'r') (or, equivalently, **y=cumsum(x, 1)**) returns in **y** the cumulative sum of the rows of **x**: **y(:, i)=cumsum(x(:, i))**

y=cumsum(x, 'm') is the cumulative sum along the first non singleton dimension of **x** (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];

cumsum(A)
cumsum(A, 'r')
cumsum(A, 'c')

a=rand(3,4)+%i;
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;
    w(k,:)=w(k-1,:)+a(k,:);
end;
w-cumsum(a, 'r')
```

See Also

cumprod, sum

Name

dec2bin — binary representation

```
[str]=dec2bin(x[,n])
```

Parameters

x
scalar/vector/matrix of positives integers

n
a positive integer

str
a string or a vector of string

Description

Given **x**, a positive (or a vector/matix of integers) integer, this function returns a string (or a column vector of strings) which is the binary representation of **x**. If dimension of **x** is superior than 1 then each component of the columns vector **str** is the binary representation of the **x** components (i.e **str(i)** is the binary representation of **x(i)**). If the components length of **str** is less than **n** (i.e length **str(i)** < **n**), then add to **str** components the characters '0' on the left in order to have componants length equal to **n**.

Examples

```
// example 1 :
x=86;
str=dec2bin(x)

// example 2 :
// the binary representation of 86 is: '1010110'
// its length is 7(less than n), so we add to str, 8 times the character '0' (
x=86;n=15;
str=dec2bin(x,n)

// example 3 :
x=[12;45;135]
z=dec2bin(x)
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2oct, dec2hex

Name

dec2hex — hexadecimal representation of integers

```
h=dec2hex(d)
```

Parameters

d
matrix of non negative integers

h
matrix of character strings

Description

dec2hex(x) returns the hexadecimal representation of a matrix of integers.

Examples

```
dec2hex([2748 10;11 3])
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2oct

Name

dec2oct — octal representation of integers

```
o=dec2oct(d)
```

Parameters

d
matrix of non negative integers

o
matrix of character strings

Description

dec2oct(x) returns the octal representation of a matrix of integers.

Examples

```
dec2oct([2748 10;11 3])
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2hex

Name

delip — complete and incomplete elliptic integral of first kind

```
[r]=delip(x,ck)
```

Parameters

- x**
real vector with non negative elements
- ck**
real number between -1 and 1
- r**
real or complex number (or vector) with the same size as **x**

Description

The elliptic integral of the first kind with parameter **ck** is defined as follow:

$$\int_0^x \frac{dt}{\sqrt{(1-t^2)(1-ck^2t^2)}}$$

Where **x** is real and positive, **ck** is in $[-1 \ 1]$.

If **x** is less than 1 the result is real.

When called with **x** a vector **r** is evaluated for each entry of **x**.

Examples

```
ck=0.5;  
delip([1,2],ck)  
deff('y=f(t)','y=1/sqrt((1-t^2)*(1-ck^2*t^2))')  
intg(0,1,f)    //OK since real solution!
```

See Also

amell, %asn, %sn

Name

diag — diagonal including or extracting

```
[y]=diag(vm, [k])
```

Parameters

vm
vector or matrix (full or sparse storage)

k
integer (default value 0)

y
vector or matrix

Description

for vm a (row or column) n-vector `diag(vm)` returns a diagonal matrix with entries of vm along the main diagonal.

`diag(vm, k)` is a $(n+abs(k)) \times (n+abs(k))$ matrix with the entries of vm along the kth diagonal. $k=0$ is the main diagonal $k>0$ is for upper diagonals and $k<0$ for lower diagonals.

For a matrix vm, `diag(vm, k)` is the column vector made of entries of the kth diagonal of vm. `diag(vm)` is the main diagonal of vm. `diag(diag(x))` is a diagonal matrix.

If vm is a sparse matrix `diag(vm, k)` returns a sparse matrix.

To construct a diagonal linear system, use `sysdiag`.

Note that `eye(A) .* A` returns a diagonal matrix made with the diagonal entries of A. This is valid for any matrix (constant, polynomial, rational, state-space linear system,...).

Examples

```
diag([1,2])

A=[1,2;3,4];
diag(A) // main diagonal
diag(A,1)

diag(sparse(1:10)) // sparse diagonal matrix

// form a tridiagonal matrix of size 2*m+1
m=5;diag(-m:m) + diag(ones(2*m,1),1) +diag(ones(2*m,1),-1)
```

See Also

`sysdiag`, `sparse`

Name

diff — Difference and discrete derivative

```
y=diff(x)
y=diff(x [,n [,dim]])
```

Parameters

x
vector or matrix (real, complex, sparse or polynomial)

n
integer the order of differentiation

dim
integer or character string with values "r", "c" and "*"

y
scalar or vector

Description

`y=diff(x)` compute the difference function $y=x(2:)-x(1:-1)$

`diff(x,n,dim)` is the n th difference function along dimension `dim`.

`diff(x,n,"*")` is equivalent to `diff(x(:),n)`.

Default value for `n` is 1. Default value for `dim` is "*".

`dim='r'` is equivalent to `dim=1` and `dim='c'` is equivalent to `dim=2`.

Examples

```
v=(1:8)^3;
diff(v)
diff(v,3)

A=[(1:8)^2
    (1:8)^3
    (1:8)^4];

diff(A,3,2)

//approximate differentiation
step=0.001
t=0:step:10;
y=sin(t);
dy=diff(sin(t))/step; //approximate differentiation of sine function
norm(dy-cos(t(1:$-1)),%inf)
```

Name

double — conversion from integer to double precision representation

```
y=double(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

Parameters

X
matrix of floats or integers

y
matrix of floats

Description

converts data stored using one, two or four bytes integers into double precision floating point representation. If X entries are already double precision floats, nothing is done.

Examples

```
x=int8([0 12 140])
double(x)
```

See Also

int8, inttype, type

Name

dsearch — binary search (aka dichotomous search in french)

```
[ind, occ, info] = dsearch(X, val [, ch ])
```

Parameters

X

a real vector or matrix

val

a real (row or column) vector with n components in strictly increasing order $\text{val}(1) < \text{val}(2) < \dots < \text{val}(n)$

ch

(optional) a character "c" or "d" (default value "c")

ind

a real vector or matrix with the same dimensions than X

occ

a real vector with the same format than val (but with n-1 components in the case ch="c")

info

integer

Description

This function is useful to search in an ordered table and/or to count the number of components of a vector falling in some classes (a class being an interval or a value).

By default or when ch="c", this is the interval case, that is, for each X(i) search in which of the n-1 intervals it falls, the intervals being defined by:

```
I1 = [val(1), val(2)]  
Ik = (val(k), val(k+1)] for 1 < k <= n-1 ;
```

and:

ind(i)

is the interval number of X(i) (0 if X(i) is not in [val(1),val(n)])

occ(k)

is the number of components of X which are in Ik

info

is the number of components of X which are not in [val(1),val(n)]

When ch="d" case, this is the discrete case, that is, for each X(i) search if it is equal to one val(k) and:

ind(i)

is equal to the index of the component of val which matches X(i) (ind(i) = k if X(i)=val(k)) or 0 if X(i) is not in val.

occ(k)

is the number of components of X equal to val(k)

info

is the number of components of X which are not in the set {val(1),...,val(n)}

Examples

```
// example #1 (elementary stat for U(0,1))
m = 50000 ; n = 10;
X = grand(m,1,"def");
val = linspace(0,1,n+1)';
[ind, occ] = dsearch(X, val);
clf() ; plot2d2(val, [occ/m;0]) // no normalisation : y must be near 1/n

// example #2 (elementary stat for B(N,p))
N = 8 ; p = 0.5; m = 50000;
X = grand(m,1,"bin",N,p); val = (0:N)';
[ind, occ] = dsearch(X, val, "d");
Pexp = occ/m; Pexa = binomial(p,N);
clf() ; hm = 1.1*max(max(Pexa),max(Pexp));
plot2d3([val val+0.1], [Pexa' Pexp],[1 2],"111", ...
        "Pexact@Pexp", [-1 0 N+1 hm],[0 N+2 0 6])
xlabel( "binomial distribution B("+string(N)+","+string(p)+") : " ...
        +" exact probability versus experimental ones")

// example #3 (piecewise Hermite polynomial)
x = [0 ; 0.2 ; 0.35 ; 0.5 ; 0.65 ; 0.8 ; 1];
y = [0 ; 0.1 ; -0.1 ; 0 ; 0.4 ; -0.1 ; 0];
d = [1 ; 0 ; 0 ; 1 ; 0 ; 0 ; -1];
X = linspace(0, 1, 200)';
ind = dsearch(X, x);

// define Hermite base functions
deff("y=Ll(t,k,x)","y=(t-x(k+1))./(x(k)-x(k+1))") // Lagrange left on Ik
deff("y=Lr(t,k,x)","y=(t-x(k))./(x(k+1)-x(k))") // Lagrange right on Ik
deff("y=Hl(t,k,x)","y=(1-2*(t-x(k))./(x(k)-x(k+1))).*Ll(t,k,x).^2")
deff("y=Hr(t,k,x)","y=(1-2*(t-x(k+1))./(x(k+1)-x(k))).*Lr(t,k,x).^2")
deff("y=Kl(t,k,x)","y=(t-x(k)).*Ll(t,k,x).^2")
deff("y=Kr(t,k,x)","y=(t-x(k+1)).*Lr(t,k,x).^2")

// plot the curve
Y = y(ind).*Hl(X,ind) + y(ind+1).*Hr(X,ind) + d(ind).*Kl(X,ind) + d(ind+1).*Kr(X,ind);
clf(); plot2d(X,Y,2) ; plot2d(x,y,-9,"000")
xlabel("an Hermite piecewise polynomial")
// NOTE : you can verify by adding these ones :
// YY = interp(X,x,y,d); plot2d(X,YY,3,"000")
```

See Also

find, tabul

Authors

B.P.

Name

eval — evaluation of a matrix of strings

```
[H]= eval(Z)
```

Description

returns the evaluation of the matrix of character strings Z.

Examples

```
a=1; b=2; Z=['a','sin(b)'] ; eval(Z) //returns the matrix [1,0.909];
```

See Also

evstr, execstr

Name

`exp` — element-wise exponential

`exp(X)`

Parameters

`X`
scalar, vector or matrix with real or complex entries.

Description

`exp(X)` is the (element-wise) exponential of the entries of `X`.

Examples

```
x=[1, 2, 3+%i];  
log(exp(x)) //element-wise  
2^x  
exp(x*log(2))
```

See Also

`coff`, `log`, `expm`

Name

eye — identity matrix

```
X=eye(m,n)
X=eye(A)
X=eye()
```

Parameters

A,X
matrices or `syslin` lists

m,n
integers

Description

according to its arguments defines an $m \times n$ matrix with 1 along the main diagonal or an identity matrix of the same dimension as A .

Caution: `eye(10)` is interpreted as `eye(A)` with $A=10$ i.e. 1. (It is NOT a ten by ten identity matrix!).

If A is a linear system represented by a `syslin` list, `eye(A)` returns an `eye` matrix of appropriate dimension: (number of outputs x number of inputs).

`eye()` produces a identity matrix with undefined dimensions. Dimensions will be defined when this identity matrix is added to a matrix with fixed dimensions.

Examples

```
eye(2,3)
A=rand(2,3);eye(A)
s=poly(0,'s');A=[s,1;s,s+1];eye(A)
A=[1/s,1;s,2];eye(A);
A=ssrand(2,2,3);eye(A)
[1 2;3 4]+2*eye()
```

See Also

ones, zeros

Name

factor — factor function

```
[y]=factor(x)
```

Parameters

x

real scalar

y

vector of primes numbers

Description

Given a real x , `factor(x)` returns in a vector y the primes numbers decomposition of x . Particular cases: `factor(0)` returns 0, and `factor(1)` returns 1.

Examples

```
x=620  
y=factor(x)
```

See Also

[primes](#)

Name

fix — rounding towards zero

```
[y]=fix(x)
```

Parameters

x
a real matrix

y
integer matrix

Description

`fix(x)` returns an integer matrix made of nearest rounded integers toward zero,i.e,
`y=sign(x).*floor(abs(x)).` Same as `int`.

See Also

`round` , `floor` , `ceil`

Name

flipdim — flip x components along a given dimension

```
y=flipdim(x,dim)
```

Parameters

x
a scalar, a vector or an array of reals

dim
a positive integer

y
a scalar, a vector or an array of reals

Description

Given x, a scalar/vector/array of reals and dim a positive integer, this function flips the x components along the dimension number dim of x (x and y have the same size)

Examples

```
// example 1: flip x components along the first dimension
x=[1 2 3 4; 5 6 7 8];
dim=1;
y=flipdim(x,dim)

// example 2: flip x components along the second dimension
dim=2;
y=flipdim(x,dim)

// example 3: flip x components along the third dimension
x=matrix(1:48,[3 2,4,2]);
dim=3;
y=flipdim(x,dim)
```

Authors

F.Belahcene

Name

floor — rounding down

```
[y]=floor(x)
```

Parameters

x
a real matrix

y
integer matrix

Description

`floor(x)` returns an integer matrix made of nearest rounded down integers.

Examples

```
floor([1.9 -2.5])-[1, -3]  
floor(-%inf)  
x=rand()*10^20;floor(x)-x
```

See Also

round, fix, ceil

Name

`frexp` — dissect floating-point numbers into base 2 exponent and mantissa

```
[f,e]=frexp(x)
```

Parameters

- `x`
real vector or matrix
- `f`
array of real values, usually in the range $0.5 \leq \text{abs}(f) < 1$.
- `e`
array of integers that satisfy the equation: $x = f \cdot 2.^e$

Description

This function corresponds to the ANSI C function `frexp()`. Any zeros in `x` produce `f=0` and `e=0`.

Examples

```
[f,e]=frexp([1,%pi,-3,%eps])
```

See Also

`log`, `hat`, `ieee`, `log2`

Name

gsort — sorting by quick sort algorithm

```
[B [,k]]=gsort(A)
[B [,k]]=gsort(A,option)
[B [,k]]=gsort(A,option,direction)
```

Parameters

A

a real, an integer or a character string vector/matrix or a sparse vector.

option

a character string. It gives the type of sort to perform:

- 'r': each column of A is sorted
- 'c': each row of A is sorted
- 'g': all elements of A are sorted. It is the default value.
- 'lr': lexicographic sort of the rows of A
- 'lc': lexicographic sort of the columns of A

direction

a character string. It gives the ordering direction: 'i' stand for increasing and 'd' for decreasing order (default).

B

an array with same type and dimensions as A.

k

a real array with integer values and same dimensions as A. Contains the origin indices.

Description

gsort implements a "quick sort" algorithm for various data types.

- $B = \text{gsort}(A, 'g')$, $B = \text{gsort}(A, 'g', 'd')$ and $B = \text{gsort}(A)$ sort the elements of the array A, seen as $A(:)$ in a decreasing order.

$B = \text{gsort}(A, 'g', 'i')$ sort the elements of the array A in the increasing order.

- $B = \text{gsort}(A, 'lr')$ sorts the rows of A in lexical decreasing order. B is obtained by a permutation of the rows of matrix A in such a way that the rows of B verify $B(i,:) \geq B(j,:)$ if $i < j$.

$B = \text{gsort}(A, 'lr', 'i')$ works similarly for increasing lexical order.

- $B = \text{gsort}(A, 'lc')$ sorts the columns of A in lexical decreasing order. B is obtained by a permutation of the columns of matrix A in such a way that the columns of B verify $B(:,i) \geq B(:,j)$ if $i < j$.

$B = \text{gsort}(A, 'lc', 'i')$ works similarly for increasing lexical order.

If required the second return argument k contains the indices of the sorted values in A. If $[B, k] = \text{gsort}(A, 'g')$ one has $B == A(k)$. **The algorithm preserve the relative order of records with equal values.**

When v is complex, the elements are sorted by magnitude, i.e., $\text{abs}(v)$. Only 'g' as second argument is managed with complex.

if v have %nan or %inf as elements. gsort places these at the beginning with 'i' or at the end with 'd' argument.

Examples

```
alr=[1,2,2;
      1,2,1;
      1,1,2;
      1,1,1];
[alr1,k]=gsort(alr,'lr','i')
[alr1,k]=gsort(alr,'lc','i')

v=int32(alr)

gsort(v)
gsort(v,'lr','i')
gsort(v,'lc','i')

v=['Scilab' '2.6'
   'Scilab' '2.7'
   'xcos' '2.7'
   'Scilab' '3.1'
   'xcos' '3.1'
   'xcos' '4.0'
   'Scilab' '4.0']

gsort(v,'lr','i')
gsort(v,'lc','i')
```

See Also

[find](#)

Bibliography

Quick sort algorithm from Bentley & McIlroy's "Engineering a Sort Function". Software---Practice and Experience, 23(11):1249-1265

Name

hex2dec — conversion from hexadecimal representation to integers

```
d=hex2dec(h)
```

Parameters

d

matrix of integers

h

matrix of character strings corresponding to hexadecimal representation

Description

hex2dec(x) returns the matrix of numbers corresponding to the hexadecimal representation.

Examples

```
hex2dec(['ABC','0','A'])
```

See Also

base2dec, bin2dec, oct2dec, dec2bin, dec2oct, dec2hex

Name

imag — imaginary part

```
[y]=imag(x)
```

Parameters

x
real or complex vector or matrix.

y
real vector or matrix.

Description

`imag(x)` is the imaginary part of x . (See `%i` to enter complex numbers).

See Also

[real](#)

Name

`imult` — multiplication by i the imaginary unitary

```
y=imult(x)
```

Parameters

`x`
real or complex scalar, vector or matrix

`y`
complex scalar, vector or matrix

Description

`imult(x)` is a more efficient way to multiply `x` by i than `y = %i*x`, without the problems occurring when `x` comprises "special" floating point numbers as `%inf` and `%nan`.

Examples

```
z1 = imult(%inf)
z2 = %i * %inf
```

Authors

B.P.

Name

ind2sub — linear index to matrix subscript values

```
[i1,i2,...] =ind2sub(dims,I)
Mi = ind2sub(dims,I)
```

Parameters

dims

vector: the matrix dimensions

I

vector: the given linear index

i1,i2,..

the subscript values (same matrix shape as **I**)

Mi

matrix whose columns contains the subscript values.

Description

`ind2sub` is used to determine the equivalent subscript values corresponding to a given single index into an array. `[i1,i2,...] = ind2sub(dims,I)` returns the arrays `i1`, `i2`, ... containing the equivalent row, column, ... subscripts corresponding to the index matrix **I** for a matrix of size **dims**. `Mi=ind2sub(dims,I)` returns a matrix **Mi** whose columns are the arrays `i1(:)`, `i2(:)`, ...

Examples

```
ind2sub([2,3,2],1:12)
[i,j,k]=ind2sub([2,3,2],1:12)
```

See Also

`sub2ind`, `extraction`, `insertion`

Authors

Serge Steer, INRIA

Name

`int` — integer part

```
[y]=int(X)
```

Parameters

`X`
real matrix

`y`
integer matrix

Description

`int(X)` returns the integer part of the real matrix `X`. Same as `fix`.

See Also

`round` , `floor` , `ceil`

Name

`intersect` — returns the vector of common values of two vectors

```
[v [, ka, kb]]=intersect(a,b)
[v [, ka, kb]]=intersect(a,b,orient)
```

Parameters

a
vector of numbers or strings

b
vector of numbers or strings

orient
flag with possible values : 1 or "r", 2 or "c"

v
row vector of numbers or strings

ka
row vector of integers

kb
row vector of integers

Description

`intersect(a,b)` returns a sorted row vector of common values of two vectors of **a** and **b**.

`[v, ka, kb]=intersect(a,b)` also returns index vectors **ka** and **kb** such that $v=a(ka)$ and $v=b(kb)$.

`intersect(a,b,"r")` or `intersect(a,b,1)` returns the matrix formed by the intersection of the unique rows of **a** and **b** sorted in lexicographic ascending order. In this case matrices **a** and **b** must have the same number of columns.

`[v, ka, kb]=intersect(a,b,"r")` also returns index vectors **ka** and **kb** such that $v=a(ka,:)$ and $v=b(kb,:)$.

`intersect(a,b,"c")` or `intersect(a,b,2)` returns the matrix formed by the intersection of the unique columns of **a** and **b** sorted in lexicographic ascending order. In this case matrices **a** and **b** must have the same number of rows.

`[v, ka, kb]=intersect(a,b,"c")` also returns index vectors **ka** and **kb** such that $v=a(:,ka)$ and $v=b(:,kb)$.

Remark

NaN are considered as different from themselves so they are excluded out of intersection in case of vector intersection.

Examples

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

intersect(A,B)
[N, ka, kb]=intersect(A,B)

intersect('a'+string(A), 'a'+string(B))

intersect(int16(A),int16(B))

//with matrices
A = [0,0,1,1 1;
     0,1,1,1,1;
     2,0,1,1,1;
     0,2,2,2,2;
     2,0,1,1,1;
     0,0,1,1,%nan];
B = [1,0,1;
     1,0,2;
     1,2,3;
     2,0,4;
     1,2,5;
     %nan,0,6];

[v, ka, kb] = intersect(A,B, 'c')
A(:, ka)
```

See Also

[unique](#), [gsort](#), [union](#)

Name

inttrap — integration of experimental data by trapezoidal interpolation

```
v = inttrap([x,] s)
```

Parameters

x
vector of increasing x coordinate data. Default value is `1:size(y, 'r')`

s
vector of y coordinate data

v
value of the integral

Description

computes :

Where f is a function described by a set of experimental value:

$s(i)=f(x(i))$ and $x0=x(1)$, $x1=x(n)$

Between mesh points function is interpolated linearly.

Examples

```
t=0:0.1:%pi  
inttrap(t, sin(t))
```

See Also

intg, intc, intl, integrate, intsplin, splin

Name

`isdef` — checks variable existence

```
isdef(name [,where])
```

Parameters

`name`

a character string

`where`

an optional character string with default value `'all'`

Description

`isdef(name)` returns %T if the variable named `name` exists and %F otherwise.

Caveats: a function which uses `isdef` may return a result which depends on the environment!

`isdef(name, 'local')` returns %T if the variable named `name` exists in the local environment of the current function and %F otherwise.

`isdef(name, 'nolocal')` returns %T if the variable named `name` exists in the full calling environment (including the global level) of the current function and %F otherwise.

Examples

```
A=1;
isdef('A')
clear A
isdef('A')

function level1()
    function level2()
        disp(isdef("a","all"));
        disp(isdef("a","local"));
        disp(isdef("a","nolocal"));
    endfunction
    level2()
endfunction
function go()
    a=1;
    level1()
endfunction
go()
```

See Also

`exists` `isglobal`, `whereis`, `type`, `typeof`, `clear`

Name

`isempty` — check if a variable is an empty matrix or an empty list

```
t=isempty(x)
```

Parameters

`x`
vector or matrix or list

`t`
a boolean

Description

`isempty(x)` returns true if `x` is an empty matrix or an empty list.

Examples

```
a=1  
isempty(a(2:$))  
isempty(find(rand(1:10)==5))
```

Name

isequal — objects comparison

```
t=isequal(a,b)
t=isequal(a,b,...)
```

Parameters

a, b , ...
variables of any types

t
boolean

Description

`isequal` compares its arguments. If all of them are equals function returns `%t` and in the other case it returns `%f` .

When comparing list's, structures,... the comparison is made recursively, the order of the fields matters.

floating point data are compared according to IEEE rule, i.e. NaN values are not equal. See `isequal-bitwise` for bitwise comparisons.

Examples

```
a=[1 2]
isequal(a,[1 2])
isequal(a,1)
```

See Also

`isequalbitwise`, `equal`, `less`

Name

isequalbitwise — bitwise comparison of variables

```
t=isequalbitwise(a,b)
t=isequalbitwise(a,b,...)
```

Parameters

a, b , ...
variables of any types

t
boolean

Description

`isequalbitwise` compares its arguments. If all of them are equals function returns `%t` and in the other case it returns `%f`.

When comparing list's, structures,... the comparison is made recursively, the order of the fields matters.

floating point data are compared bitwise, i.e. NaN values are not equal, `double(1)` and `int32(1)` are not equal. See `isequal` for IEEE comparisons.

Examples

```
a=list(1:5,%s+1,'ABCDEFGH');
isequalbitwise(a,a)
```

See Also

`isequal`

Name

isinf — check for infinite entries

```
r=isinf(x)
```

Parameters

x
real or complex vector or matrix r : boolean vector or matrix

Description

`isinf(x)` returns a boolean vector or matrix which contains true entries corresponding with infinite x entries and false entries corresponding with finite x entries.

Examples

```
isinf([1 0.01 -%inf %inf])
```

See Also

isnan

Name

isnan — check for "Not a Number" entries

```
r=isnan(x)
```

Parameters

x
real or complex vector or matrix r : boolean vector or matrix

Description

`isnan(x)` returns a boolean vector or matrix which contains true entries corresponding with "Not a Number" x entries and false entries corresponding with regular x entries.

Examples

```
isnan([1 0.01 -%nan %inf-%inf])
```

See Also

[isinf](#)

Name

`isreal` — check if a variable as real or complex entries

```
t=isreal(x)
t=isreal(x,eps)
```

Parameters

`x`
vector or matrix with floating point entries or coefficients

`t`
a boolean

Description

`isreal(x)` returns true if `x` is stored as a real variable and false if `x` is stored with an (eventually zero) imaginary part.

`isreal(x,eps)` returns true if `x` is stored as a real variable or if maximum absolute value of imaginary floating points is less or equal than `eps`.

Examples

```
isreal([1 2])
isreal(1+0*i)
isreal(1+0*i,0)
isreal(1+s)
isreal(sprand(3,3,0.1))
```

Name

isvector — check if a variable is a vector

```
t=isvector(x)
```

Parameters

x
vector or matrix

t
a boolean

Description

isvector(x) returns true if x is a vector (one of its dimension is different from 1).

Examples

```
isvector(ones(10,1))  
isvector(1)
```

Name

kron — Kronecker product (.*.)

```
kron(A,B)  
A.*.B
```

Description

`kron(A,B)` or `A.*.B` returns the Kronecker tensor product of two matrices A and B. The resulting matrix has the following block form:

$$A.*.B = \begin{pmatrix} A(1,1) \cdot B & \cdots & A(1,n) \cdot B \\ \vdots & & \vdots \\ A(m,1) \cdot B & \cdots & A(m,n) \cdot B \end{pmatrix}$$

If A is a $m \times n$ matrix and B a $p \times q$ matrix then `A.*.B` is a $(m \cdot p) \times (n \cdot q)$ matrix.

A and B can be sparse matrices.

Examples

```
A=[1,2;3,4];  
kron(A,A)  
A.*.A  
sparse(A).*sparse(A)  
A(1,1)=%i;  
kron(A,A)
```

Name

lex_sort — lexicographic matrix rows sorting

```
[N, [k]]=lex_sort(M [,sel] [, 'unique'])
```

Parameters

M
real matrix

N
real matrix

k
column vector of integers

Description

the `lex_sort` function is now obsolete. It can be replaced by functions `gsort` and `unique`.

`N=lex_sort(M)` sorts the rows (as a group) of the matrix `M` in ascending order. If required the output argument `k` contains the ordering: `[N, k]=lex_sort(M)` returns `k` such as `N` is uequal to `M(k, :)`.

`N=lex_sort(M, sel [, 'unique'])` produces the same result as the following sequence of instructions:

```
[N,k]=lex_sort(M(:,sel) [, 'unique']);  
N=M(k, :)
```

The 'unique' flag has to be given if one wants to retain only unique rows in the result. Note that `lex_sort(M, sel, 'unique')` retains only rows such that `M(:,sel)` are unique.

Examples

```
M=round(2*rand(20,3));  
  
lex_sort(M)  
lex_sort(M, 'unique')  
[N,k]=lex_sort(M, [1 3], 'unique')
```

See Also

`gsort`, `unique`

Name

linspace — linearly spaced vector

```
[v]=linspace(x1,x2 [,n])
```

Parameters

x1,x2

real or complex scalars

n

integer (number of values) (default value = 100)

v

real or complex row vector

Description

Linearly spaced vector. `linspace(x1, x2)` generates a row vector of n (default value=100) linearly equally spaced points between x1 and x2. If x1 or x2 are complex then `linspace(x1, x2)` returns a row vector of n complexes, the real (resp. imaginary) parts of the n complexes are linearly equally spaced between the real (resp. imaginary) parts of x1 and x2.

Examples

```
linspace(1,2,10)  
linspace(1+%i,2+2*%i,10)
```

See Also

logspace

Name

log — natural logarithm

```
y=log(x)
```

Parameters

x
constant vector or constant matrix

Description

$\log(x)$ is the "element-wise" logarithm. $y(i,j)=\log(x(i,j))$. For matrix logarithm see `logm`.

Examples

```
exp(log([1,%i,-1,-%i]))
```

See Also

`exp`, `logm`, `log10`, `ieee`

Name

log10 — logarithm

```
y=log10(x)
```

Parameters

x
vector or matrix

Description

decimal logarithm.If x is a vector $\log_{10}(x)=[\log_{10}(x_1), \dots, \log_{10}(x_n)]$.

Examples

```
10.^log10([1,%i,-1,-%i])
```

See Also

log, logm, hat, iieee

Name

log1p — computes with accuracy the natural logarithm of its argument added by one

```
y=log1p(x)
```

Parameters

x
real scalar, vector or matrix

y
real scalar, vector or matrix

Description

$\text{log1p}(x)$ is the "element-wise" $\log(1+x)$ function. $y(i,j)=\log(1 + x(i,j))$. This function, defined for $x > -1$, must be used if we want to compute $\log(1+x)$ with accuracy for $|x| \ll 1$.

Examples

```
format("e",24)
log(1.001)
log1p(0.001)
log(1 + 1.e-7)
log1p(1.e-7)
log(1 + 1.e-20)
log1p(1.e-20)
format("v") //reset default format
```

See Also

log

Authors

B.P.

Name

log2 — base 2 logarithm

```
y=log2(x)
```

Parameters

x
vector or matrix

Description

decimal logarithm.If x is a vector $\log_2(x)=[\log_2(x_1), \dots, \log_2(x_n)]$.

Examples

```
2.^log2([1,%i,-1,-%i])
```

See Also

log, hat, ieee, log10, frexp

Name

logm — square matrix logarithm

```
y=logm(x)
```

Parameters

x
square matrix

Description

$\text{logm}(x)$ is the matrix logarithm of x . The result is complex if x is not positive or definite positive. If x is a symmetric matrix, then calculation is made by schur form. Otherwise, x is assumed diagonalizable. One has $\text{expm}(\text{logm}(x))=x$.

Examples

```
A=[1,2;3,4];  
logm(A)  
expm(logm(A))  
  
A1=A*A';  
logm(A1)  
expm(logm(A1))  
  
A1(1,1)=%i;  
expm(logm(A1))
```

See Also

expm, log

Name

logspace — logarithmically spaced vector

```
logspace(d1, d2, [n])
```

Parameters

d1,d2
real or complex scalar (special meaning for %pi)

n
integer (number of values) (default value = 50)

Description

returns a row vector of n logarithmically equally spaced points between 10^{d1} and 10^{d2} . If d2= %pi then the points are between 10^{d1} and pi.

Examples

```
logspace(1, 2, 10)
```

See Also

linspace

Name

lstsize — list, tlist, mlist numbers of entries

```
n=lstsize(x)
```

Parameters

- l
a list, tlist or mlist object
- n
an integer, the number of entries

Description

`lstsize(x)` returns the number of entries for list, list, mlist objects. This function is more efficient than the `size` function and works similarly with all list types while `size` is overloaded for `mlist` objects.

Examples

```
lstsize(list(1, 'aqsd'))  
x=ssrand(3,2,4);  
[ny,nu]=size(x)  
lstsize(x)
```

See Also

length, size, list, tlist, mlist

Name

max — maximum

```
[m [, k]]=max(A)
[m [, k]]=max(A, 'c')
[m [, k]]=max(A, 'r')
[m [, k]]=max(A, 'm')
[m [, k]]=max(A1,A2,...,An)
[m [, k]]=max(list(A1,A2,...,An))
```

Parameters

A
real vector or matrix.

A1,...,An
a set of real vectors or matrices, all of the same size or scalar.

Description

For A, a real vector or matrix, `max(A)` is the largest element A. `[m, k]=max(A)` gives in addition the index of the maximum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that `m(j)` contains the maximum of the jth column of A (`A(:, j)`), `k(j)` gives the row indice which contain the maximum for column j. 'c' is used for the dual operation on the rows of A. 'm' is used for compatibility with Matlab.

`m=max(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes, returns a vector or a matrix m of size `size(m)=size(A1)` such that `m(i)= max( Aj(i))`, `j=1,...,n`. `[m, k]=max(A1,A2,...,An)` gives in addition the vector or matrix k. for a fixed i, `k(i)` is the number of the first `Aj(i)` achieving the maximum.

`[m, k]=max(list(A1,...,An))` is an equivalent syntax of `[m, k]=max(A1,A2,...,An)`

Examples

```
[m, n]=max([1, 3, 1])
[m, n]=max([3, 1, 1], [1, 3, 1], [1, 1, 3])
[m, n]=max([3, -2, 1], 1)
[m, n]=max(list([3, 1, 1], [1, 3, 1], [1, 1, 3]))
[m, n]=max(list(1, 3, 1))
```

See Also

gsort, find, mini

Name

maxi — maximum

```
[m [, k]]=maxi(A)
[m [, k]]=maxi(A, 'c')
[m [, k]]=maxi(A, 'r')
[m [, k]]=maxi(A, 'm')
[m [, k]]=maxi(A1,A2,...,An)
[m [, k]]=maxi(list(A1,A2,...,An))
```

Parameters

A
real vector or matrix.

A1,...,An
a set of real vectors or matrices, all of the same size or scalar.

Description

For A, a real vector or matrix, `maxi(A)` is the largest element A. `[m, k]=maxi(A)` gives in addition the index of the maximum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that `m(j)` contains the maximum of the jth column of A (`A(:, j)`), `k(j)` gives the row indice which contain the maximum for column j. 'c' is used for the dual operation on the rows of A. 'm' is used for compatibility with Matlab.

`m=maxi(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes, returns a vector or a matrix m of size `size(m)=size(A1)` such that `m(i)= maxi( Aj(i))`, `j=1,...,n`. `[m, k]=maxi(A1,A2,...,An)` gives in addition the vector or matrix k. for a fixed i, `k(i)` is the number of the first `Aj(i)` achieving the maximum.

`[m, k]=maxi(list(A1,...,An))` is an equivalent syntax of
`[m, k]=maxi(A1,A2,...,An)`

Examples

```
[m, n]=maxi([1, 3, 1])
[m, n]=maxi([3, 1, 1], [1, 3, 1], [1, 1, 3])
[m, n]=maxi([3, -2, 1], 1)
[m, n]=maxi(list([3, 1, 1], [1, 3, 1], [1, 1, 3]))
[m, n]=maxi(list(1, 3, 1))
```

See Also

gsort, find, mini

Name

meshgrid — create matrices or 3-D arrays

```
[X, Y] = meshgrid(x)
[X, Y] = meshgrid(x,y)
[X, Y, Z] = meshgrid(x,y,z)
```

Parameters

x, y, z
vectors

X, Y, Z
matrices in case of 2 input arguments, else 3-D arrays in case of 3 input arguments

Description

Create matrices or 3-D arrays.

Examples

```
x = -1:0.1:1;
y = -1:0.1:1;

[X,Y] = meshgrid(x,y);

for i=1:size(X,1)
    for j=1:size(X,2)
        Z(i,j) = sinc(2*pi*X(i,j)*Y(i,j));
    end
end

surf(X,Y,Z)
```

See Also

ndgrid

Authors

Farid Belahcene

Name

min — minimum

```
[m [,k]]=min(A)
[m [,k]]=min(A, 'c')
[m [,k]]=min(A, 'r')
[m [,k]]=min(A, 'm')
[m [,k]]=min(A1,A2,...,An)
[m [,k]]=min(list(A1,A2,...,An))
```

Parameters

A
real vector or matrix.

A1,...,An
a set of real vectors or matrices, all of the same size or scalar.

Description

For A, a real vector or matrix, `min(A)` is the largest element A. `[m,k]=min(A)` gives in addition the index of the minimum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that `m(j)` contains the minimum of the jth column of A (`A(:,j)`), `k(j)` gives the row indice which contain the minimum for column j. 'c' is used for the dual operation on the rows of A. 'm' is used for compatibility with Matlab.

`m=min(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes, returns a vector or a matrix m of size `size(m)=size(A1)` such that `m(i)= min( Aj(i))`, `j=1,...,n`. `[m,k]=min(A1,A2,...,An)` gives in addition the vector or matrix k. for a fixed i, `k(i)` is the number of the first `Aj(i)` achieving the minimum.

`[m,k]=min(list(A1,...,An))` is an equivalent syntax of `[m,k]=min(A1,A2,...,An)`

Examples

```
[m,n]=min([1,3,1])
[m,n]=min([3,1,1],[1,3,1],[1,1,3])
[m,n]=min([3,-2,1],1)
[m,n]=min(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=min(list(1,3,1))
```

See Also

gsort, find, mini

Name

mini — minimum

```
[m [,k]]=mini(A)
[m [,k]]=mini(A, 'c')
[m [,k]]=mini(A, 'r')
[m [,k]]=mini(A, 'm')
[m [,k]]=mini(A1,A2,...,An)
[m [,k]]=mini(list(A1,A2,...,An))
```

Parameters

A
real vector or matrix.

A1,...,An
a set of real vectors or matrices, all of the same size or scalar.

Description

For A, a real vector or matrix, `mini(A)` is the largest element A. `[m,k]=mini(A)` gives in addition the index of the minimum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that `m(j)` contains the minimum of the jth column of A (`A(:,j)`), `k(j)` gives the row indice which contain the minimum for column j. 'c' is used for the dual operation on the rows of A. 'm' is used for compatibility with Matlab.

`m=mini(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes, returns a vector or a matrix m of size `size(m)=size(A1)` such that `m(i)= mini( Aj(i))`, `j=1,...,n`. `[m,k]=mini(A1,A2,...,An)` gives in addition the vector or matrix k. for a fixed i, `k(i)` is the number of the first `Aj(i)` achieving the minimum.

`[m,k]=mini(list(A1,...,An))` is an equivalent syntax of `[m,k]=mini(A1,A2,...,An)`

Examples

```
[m,n]=mini([1,3,1])
[m,n]=mini([3,1,1],[1,3,1],[1,1,3])
[m,n]=mini([3,-2,1],1)
[m,n]=mini(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=mini(list(1,3,1))
```

See Also

gsort, find, min

Name

minus — (-) subtraction operator, sign changes

```
X-Y  
-Y
```

Parameters

X
scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

Y
scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

Description

Substraction

For numeric operands subtraction as its usual meaning. If one of the operands is a matrix and the other one a scalar the the operation is performed element-wise. if `Y==[]` X is returned; if `X==[]` -Y is returned.

Substraction may also be defined for other data types through "soft-coded" operations.

Examples

```
[1, 2] - 1  
[] - 2  
  
%s - 2  
1/%s - 2  
"cat"+"enate"
```

See Also

`addf`, `mtlb_mode`

Name

`modulo` — symmetric arithmetic remainder modulo `m`
`pmodulo` — positive arithmetic remainder modulo `m`

```
i=modulo(n,m)
i=pmodulo(n,m)
```

Parameters

`n,m`
integers

Description

`modulo` computes $i = n \pmod{m}$ i.e. remainder of `n` divided by `m` (`n` and `m` integers).

$i = n - m \cdot \text{int}(n / m)$. Here the answer may be negative if `n` or `m` are negative.

`pmodulo` computes $i = n - m \cdot \text{floor}(n / m)$, the answer is positive or zero.

Examples

```
n=[1,2,10,15];m=[2,2,3,5];
modulo(n,m)

modulo(-3,9)
pmodulo(-3,9)
```

Name

ndgrid — arrays for multidimensional function evaluation on grid

```
[X, Y] = ndgrid(x,y)
[X, Y, Z] = ndgrid(x,y,z)
[X, Y, Z, T] = ndgrid(x,y,z,t)
[X1, X2, ..., Xm] = ndgrid(x1,x2,...,xm)
```

Parameters

$x, y, z, \dots$
vectors

$X, Y, Z, \dots$
matrices in case of 2 input arguments, or else hypermatrices

Description

This is an utility routine useful to create arrays for function evaluation on 2, 3, ..., n dimensional grids. For instance in 2d, a grid is defined by two vectors, x and y of length n_x and n_y , and you want to evaluate a function (says f) on all the grid points, that is on all the points of coordinates $(x(i), y(j))$ with $i=1, \dots, n_x$ and $j=1, \dots, n_y$. In this case, this function can compute the two matrices X, Y of size $n_x \times n_y$ such that :

```
X(i,j) = x(i)    for all i in [1,nx]
Y(i,j) = y(j)    and j in [1,ny]
```

and the evaluation may be done with $Z=f(X, Y)$ (at the condition that you have coded f for evaluation on vector arguments, which is done (in general) by using the element-wise operators $.*$, $./$ and $.^$ in place of $*$, $/$ and $^$).

In the 3d case, considering 3 vectors x, y, z of length n_x, n_y and n_z , X, Y, Z are 3 hypermatrices of size $n_x \times n_y \times n_z$ such that :

```
X(i,j,k) = x(i)
Y(i,j,k) = y(j)    for all (i,j,k) in [1,nx]x[1,ny]x[1,nz]
Z(i,j,k) = z(k)
```

In the general case of m input arguments $x_1, x_2, \dots, x_m$, then the m output arguments $X_1, X_2, \dots, X_m$ are hypermatrices of size $n_{x1} \times n_{x2} \times \dots \times n_{xm}$ and :

```
Xj(i1,i2,...,ij,...,im) = xj(ij)
for all (i1,i2,...,im) in [1,nx1]x[1,nx2]x...x[1,nxm]
```

Examples

```
// create a simple 2d grid
nx = 40; ny = 40;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
```

```
[X,Y] = ndgrid(x,y);

// compute a function on the grid and plot it
//deff("z=f(x,y)","z=128*x.^2.*(1-x).^2.*y.^2.*(1-y).^2");
deff("z=f(x,y)","z=x.^2 + y.^3")
Z = f(X,Y);
clf()
plot3d(x,y,Z, flag=[2 6 4]); xselect()

// create a simple 3d grid
nx = 10; ny = 6; nz = 4;
x = linspace(0,2,nx);
y = linspace(0,1,ny);
z = linspace(0,0.5,nz);
[X,Y,Z] = ndgrid(x,y,z);

// try to display this 3d grid ...
XF=[]; YF=[]; ZF=[];

for k=1:nz
    [xf,yf,zf] = nf3d(X(:,:,k),Y(:,:,k),Z(:,:,k));
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end

for j=1:ny
    [xf,yf,zf] = nf3d(matrix(X(:,j,:),[nx,nz]),...
                      matrix(Y(:,j,:),[nx,nz]),...
                      matrix(Z(:,j,:),[nx,nz]));
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end

clf()
plot3d(XF,YF,ZF, flag=[0 6 3], leg="X@Y@Z")
xtitle("A 3d grid !"); xselect()
```

See Also

[kron](#)

Authors

B. Pincon

Name

`ndims` — number of dimensions of an array

Parameters

`A`

an array

`n`

integer, the number of dimensions of the array

Description

`n=ndims(A)` return the number of dimension of the array `A`. `n` is greater than or equal to 2.

Examples

```
A=rand(2,3);  
ndims(A)  
  
A=rand(2,3,2);  
size(A),ndims(A)  
  
H=[1/%s,1/(%s+1)]  
ndims(H)
```

See Also

[size](#)

Authors

S. Steer

Name

nearfloat — get previous or next floating-point number

```
xnear = nearfloat(dir, x)
```

Parameters

dir
string ("succ" or "pred")

x
real scalar, vector or matrix

xnear
real scalar, vector or matrix

Description

This function computes, in the element wise meaning, the corresponding neighbours of the elements of `x` (in the underlying floating point set, see `number_properties`), the successors if `dir = "succ"` and the predecessors if `dir = "pred"`.

Examples

```
format("e",22)
nearfloat("succ",1) - 1
1 - nearfloat("pred",1)
format("v") //reset default format
```

See Also

`number_properties`, `frexp`

Authors

B.P.

Name

nextpow2 — next higher power of 2.

```
t=nextpow2(x)
```

Parameters

x
real vector or matrix

p
integer vector or matrix

Description

If **x** is a scalar, `nextpow2(x)` returns the first **p** such that $2^p \geq \text{abs}(x)$. if **x** is a vector or a matrix `nextpow2(x)` applies element-wise.

Examples

```
nextpow2(127)
nextpow2(128)
nextpow2(0:10)
```

See Also

`frexp`

Name

norm — matrix norms

```
[y]=norm(x [,flag])
```

Parameters

x
real or complex vector or matrix (full or sparse storage)

flag
string (type of norm) (default value =2)

Description

For matrices

norm(x)
or norm(x,2) is the largest singular value of x (max(svd(x))).

norm(x,1)
The l₁ norm x (the largest column sum : max_i(sum(abs(x), 'r'))).

norm(x,'inf'),norm(x,%inf)
The infinity norm of x (the largest row sum : max_i(sum(abs(x), 'c'))).

norm(x,'fro')
Frobenius norm i.e. sqrt(sum(diag(x'*x)))

For vectors

norm(v,p)
l_p norm (sum(v(i)^p))^(1/p).

norm(v)
=norm(v,2) : l₂ norm

norm(v,'inf')
max(abs(v(i))).

Examples

```
A=[1,2,3];  
norm(A,1)  
norm(A,'inf')  
A=[1,2;3,4]  
max(svd(A))-norm(A)  
  
A=sparse([1 0 0 33 -1])  
norm(A)
```

See Also

h_norm, dhnorm, h2norm, abs

Name

not — ($\sim$) logical not

```
~A
```

Description

$\sim A$ gives the element-wise negation of the elements of the boolean matrix A .

Examples

```
~[%t %t %f]
```

See Also

and, or, find

Name

number_properties — determine floating-point parameters

```
pr = number_properties(prop)
```

Parameters

prop
string

pr
real or boolean scalar

Description

This function may be used to get the characteristic numbers/properties of the floating point set denoted here by $F(b, p, e_{\min}, e_{\max})$ (usually the 64 bits float numbers set prescribe by IEEE 754). Numbers of F are of the form:

```
sign * m * b^e
```

e is the exponent and m the mantissa:

```
m = d_1 b^(-1) + d_2 b^(-2) + .... + d_p b^(-p)
```

d_i the digits are in $[0, b-1]$ and e in $[e_{\min}, e_{\max}]$, the number is said "normalised" if $d_1 \neq 0$. The following may be gotten:

prop = "radix"
then pr is the radix b of the set F

prop = "digits"
then pr is the number of digits p

prop = "huge"
then pr is the max positive float of F

prop = "tiny"
then pr is the min positive normalised float of F

prop = "denorm"
then pr is a boolean (%t if denormalised numbers are used)

prop = "tiniest"
then if denorm = %t, pr is the min positive denormalised number else pr = tiny

prop = "eps"
then pr is the epsilon machine (generally $(b^{(1-p)})/2$) which is the relative max error between a real x (such than $|x|$ in $[tiny, huge]$) and $fl(x)$, its floating point approximation in F

prop = "minexp"
then pr is $e_{\min}$

```
prop = "maxexp"  
then pr is emax
```

Remarks

This function uses the lapack routine dlamch to get the machine parameters (the names (radix, digit, huge, etc...) are those recommended by the LIA 1 standard and are different from the corresponding lapack's ones) ; CAUTION: sometimes you can see the following definition for the epsilon machine : $\text{eps} = b^{(1-p)}$ but in this function we use the traditionnal one (see prop = "eps" before) and so $\text{eps} = (b^{(1-p)})/2$ if normal rounding occurs and $\text{eps} = b^{(1-p)}$ if not.

Examples

```
b = number_properties("radix")  
eps = number_properties("eps")
```

See Also

nearfloat, frexp

Authors

Bruno Pincon

Name

oct2dec — conversion from octal representation to integers

```
d=oct2dec(o)
```

Parameters

d

matrix of integers

o

matrix of character strings corresponding to octal representation

Description

oct2dec(x) returns the matrix of numbers corresponding to the octal representation.

Examples

```
oct2dec(["1" "756115"; "0" "23"])
```

See Also

base2dec, bin2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

ones — matrix made of ones

```
y=ones(m1,m2,...)
y=ones(x)
y=ones()
```

Parameters

x,y
matrices

m1, m2,...
integers

Description

Returns a matrix made of ones.

ones(m1,m2)
returns a (m1, m2) matrix full of ones.

ones(m1,m2,...,mn)
creates a (m1, m2, . . . , mn) matrix full of ones.

ones(x)
returns a matrix full of ones with the same size that x.

ones(x)
is also valid for x a `syslin` list.

Note that `ones(3)` is `ones(a)` with `a=3` i.e it is NOT a 3x3 matrix!

`ones()` is equivalent to `ones(1,1)`.

Examples

```
ones(3)
ones(3,3)
ones(2,3,2)
```

See Also

eye, zeros

Name

or — logical or of the elements of an array

```
or(A), or(A, '*')
or(A, 'r'), or(A, 1)

or(A, 'c'), or(A, 2)
```

Description

`or(A)` gives the or of the elements of the boolean matrix `A`. `or(A)` is true (%t) iff at least one entry of `A` is %t.

`y=or(A, 'r')` (or, equivalently, `y=or(A, 1)`) is the rowwise or. It returns in each entry of the row vector `y` the or of the rows of `x` (The or is performed on the row index : `y(j)=or(A(i, j), i=1, m)`).

`y=or(A, 'c')` (or, equivalently, `y=or(A, 2)`) is the columnwise or. It returns in each entry of the column vector `y` the or of the columns of `x` (The or is performed on the column index: `y(i)=or(A(i, j), j=1, n)`)).

Examples

```
or([%t %t %f])
```

See Also

or operator (`()`), and, not, find

Name

| — logical or operator

$A \mid B$

Description

$A \mid B$ gives the element-wise logical or of the booleans matrices A and B . A and B must be matrices with the same dimensions or one from them must be a single boolean.

Examples

```
[%t %t %f] | [%f %t %t]
[%t %t %f] | %f
```

See Also

or, and, and operator (&), not, find

Name

pen2ea — pencil to E,A conversion

```
[E,A]=pen2ea(Fs)
```

Parameters

Fs
matrix pencil $sE - A$

E,A
two matrices such that $FS = sE - A$

Description

Utility function. Given the pencil $FS = sE - A$, returns the matrices E and A.

Examples

```
E=[1,0];A=[1,2];s=poly(0,'s');  
[E,A]=pen2ea(sE-A)
```

Name

perms — all permutations of vector components

```
y=perms(x)
```

Parameters

x
scalar or vector

y
matrix

Description

Given a vector x of length n , `perms` returns all the permutations of the n components of x (i.e $n!$ permutations). The size of y is $n! \times n$.

Examples

```
x=[4, 7, 10]
y=perms(x)
x=[1, 5, 2, 5]
y=perms(x)
```

Name

permute — permute the dimensions of an array

```
y=permute(x,dims)
```

Parameters

dims

a scalar or a vector of positive reals.

x

a scalar, a vector, a matrix or a mutlti-array.

Description

Permute the dimensions of an array.

Examples

```
//example 1:  
x=[1 2 3;4 5 6];  
y=permute(x,[2 1]);  
  
//example 2:  
x=matrix(1:12,[2,3,2]);  
y=permute(x,[3 1 2]);
```

See Also

pertrans, quote, cat

Authors

Farid Belahcene

Name

pertrans — pertranspose

```
[Y]=pertrans(X)
```

Parameters

X
real or complex matrix

Y
real or complex matrix

Description

`Y=pertrans(X)` returns the pertranspose of X, i.e. the symmetric of X w.r.t the second diagonal (utility function).

Examples

```
A=[1,2;3,4]  
pertrans(A)
```

Name

primes — primes function

```
[y]=primes(x)
```

Parameters

x
real scalar

y
vector

Description

Given a real x , `primes(x)` returns in a vector y all the primes numbers included between 1 and x .
If $x < 2$ then `primes(x)` returns an empty matrix.

Examples

```
x=35  
y=primes(x)
```

See Also

`factor`

Name

prod — product

```
y=prod(x)
y=prod(x, 'r') or y=prod(x,1)
y=prod(x, 'c') or y=prod(x,2)
y=prod(x, 'm')
```

Parameters

x
real or complex vector or matrix

y
real or complex scalar or matrix

Description

For a vector or a matrix **x**, **y=prod(x)** returns in the scalar **y** the prod of all the entries of **x**, , e.g. **prod(1:n)** is **n!**

y=prod(x, 'r') (or, equivalently, **y=prod(x,1)**) computes the rows elementwise product of **x**. **y** is the row vector: **y(1,j)=prod(x(:,j))**.

y=prod(x, 'c') (or, equivalently, **y=prod(x,2)**) computes the columns elementwise product of **x**. **y** is the column vector: **y(i,1)=prod(x(i,:))**.

y=prod(x, 'm') is the product along the first non singleton dimension of **x** (for compatibility with Matlab).

prod is not implemented for sparse matrices.

Examples

```
A=[1,2;0,100];
prod(A)
prod(A, 'c')
prod(A, 'r')
```

See Also

sum, cumprod

Name

rand — random number generator

```
rand(m1,m2,...[,key])
rand(x[,key])
rand()

rand(key)
rand("seed" [,n])
rand("info")
```

Parameters

m1
integers

key
character string with value in "uniform", "normal"

x
a matrix. Only its dimensions are taken into account.

Description

random matrix generator.

Without key argument the syntaxes below produce random matrices with the current random generator (default is "uniform")

rand(m1,m2)
is a random matrix of dimension **m1** by **m2**.

rand(m1,m2,...,mn)
is a random matrix of dimension **m1** by **m2**,... by **mn**.

rand(a)
is a random matrix of same size as **a**. **rand(a)** is complex if **a** is a complex matrix.

rand() : with no arguments gives a scalar whose value changes each time it is referenced.

If present, the key argument allows to specify an other random distribution.

rand('uniform')
The current random generator is set to a uniform random generator. Random numbers are uniformly distributed in the interval (0,1).

rand('normal')
The current random generator is set to a Gaussian (with mean 0 and variance 1) random number generator.

str=rand('info')
return the type of the default random generator ('uniform' or 'normal')

It is possible to (re-)initialize the seed of the rand generator:

rand('seed')
returns the current value of the seed.

`rand('seed',n)`
puts the seed to `n`. (by default `n=0` at first call).

Remark

Use the more powerful function `grand` instead.

Examples

```
x=rand(10,10,'uniform')
rand('normal')
rand('info')
y=rand(x,'normal');
x=rand(2,2,2)
```

See Also

`grand`, `ssrand`

Name

rat — Floating point rational approximation

```
[N,D]=rat(x [,tol])  
y=rat(x [,tol])
```

Parameters

x
real vector or matrix

n
integer vector or matrix

d
integer vector or matrix

y
real vector or matrix

Description

`[N,D] = rat(x,tol)` returns two integer matrices so that $N./D$ is close to x in the sense that $\text{abs}(N./D - X) \leq \text{tol} * \text{abs}(x)$. The rational approximations are generated by truncating continued fraction expansions. `tol = 1.e-6*norm(X,1)` is the default. `y = rat(x,tol)` return the quotient $N./D$

Examples

```
[n,d]=rat(%pi)  
[n,d]=rat(%pi,1.d-12)  
n/d-%pi
```

See Also

int, round

Name

real — real part

```
[y]=real(x)
```

Parameters

x
real or complex vector or matrix

y
real matrix

Description

`real(x)` is the real part of x (See `%i` to enter complex numbers).

See Also

`imag`

Name

`resize_matrix` — create a new matrix with a different size

```
resMat = resize_matrix(mat,nbRow,nbCol,[typeOfMat])
```

Parameters

mat
input matrix from which the resized matrix will be created.

nbRow
number of row of the resized matrix.

nbCol
number of column of the resized matrix.

typeOfMat
character string, type name of the resized matrix.

resMat
resized matrix.

Description

Create a matrix of size `nbRow` x `nbCol` and whose elements `(i,j)` are `mat(i,j)` if `(i,j)` is in the range of the input matrix. Otherwise elements `(i,j)` are 0 for real or integer matrices, %f for boolean matrices and an empty string for string matrices.

The type of the output matrix may be modified by specifying the `typeOfMat` argument. In this case, be sure that the input matrix type is compatible with this one.

For now, only real, integer matrices, boolean and character string matrices are supported. This means that `typeOfMat` must be chosen within: 'constant', 'boolean', 'string' or any integer type ('int8', 'int16',...).

Examples

```
// number matrix
myMat = 5 * rand( 3, 4 )
myMat = resize_matrix( myMat, 3, 3 ) // reduce the matrix size
myMatInteger = resize_matrix( myMat, 4, 4, 'int32' ) // create a integer matrix
myMatBoolean = resize_matrix( myMat, 2, 2, 'boolean' )
myMatBoolean = resize_matrix( myMatBoolean, 3, 5 )

// string matrix
myMatString = ["Scilab","the";"Open Source","Scientific";"Software","Package"]
myMatString = resize_matrix( myMatString, 3, 1 )
```

See Also

`matrix`, `size`, `typeof`

Authors

Jean-Baptiste Silvy

Name

round — rounding

```
y=round(x)
```

Parameters

x
real or complex matrix

y
integer or complex (with integer real and imag) matrix

Description

round(x) rounds the elements of x to the nearest integers.

Examples

```
round([1.9 -2.5])-[2, -3]  
round(1.6+2.1*i)-(2+2*i)  
round(-%inf)  
x=rand()*10^20;round(x)-x
```

See Also

int, floor, ceil

Name

`sec` — Compute the element-wise secant of the argument.

```
y = sec(x)
```

Parameters

`x`
Real or complex array.

`y`
Real or complex array.

Description

Compute the element-wise secant of the argument. The secant is a periodic function defined as $1/\cos$. For real data the results are real and in $] -\infty \ -1] \cup [1 \ \infty[$.

Examples

```
x=[0 %pi/3 2*%pi/3 %pi/4 3*%pi/4 %pi/6 5*%pi/6 %pi];
sec(x)
x=linspace(-%pi,%pi,100)
plot(x,sec(x))
```

See Also

`cos`, `secd`

Authors

Serge Steer, INRIA

Used Functions

This function uses the `COS` function.

Name

secd — Compute the element-wise secant of the argument given in degree.

```
y = secd(x)
```

Parameters

x
Real array.

y
Real array.

Description

The entries of y are the secant $1/\cos$ of the entries of x given in degree. The results are real and in $]-\infty, -1] \cup [1, \infty[$. For entries equal to $n \cdot 180$ with n integer, the result is exactly -1 or $+1$. For entries equal to $n \cdot 90$ with n integer and odd the result is infinite (or an error depending on IEEE mode).

Examples

```
secd(90)  
sec(%pi/2)
```

See Also

cosd, sec

Authors

Serge Steer, INRIA

Name

sech — Compute the element-wise hyperbolic secant of the argument.

```
y = sech(x)
```

Parameters

x
Real or complex array.

y
Real or complex array.

Description

Compute the element-wise hyperbolic secant of the argument. The hyperbolic secant is defined as $1/\cosh$. For real data the results are real and in $[0\ 1]$.

Examples

```
x=linspace(-10,10,1000)  
plot(x,sech(x))
```

See Also

cosh, asech

Authors

Serge Steer, INRIA

Name

`setdiff` — returns components of a vector which do not belong to another one

```
v=setdiff(a,b)
[v,ka]=setdiff(a,b)
```

Parameters

- a**
vector of real numbers or strings
- b**
vector of real numbers or strings
- v**
vector of real numbers or strings with same orientation than **a**
- ka**
row vector of integers, `ka(i)` is the location of `v(i)` in **a**

Description

`setdiff(a,b)` returns a sorted vector which retains the **a** entries which are not in **b**

`[v,ka]=setdiff(a,b)` returns a sorted vector which retains the **a** entries which are not in **b** and the location of these entries in **a**.

Examples

```
a = [223;111;2;4;2;2];
b = [2;3;21;223;123;22];
setdiff(a,b)
[v,k]=setdiff(string(a),string(b))
```

See Also

`unique`, `gsort`, `union`

Name

sign — sign function

Description

$X = \text{sign}(A)$ returns the matrix made of the signs of $A(i, j)$. For complex A , $\text{sign}(A) = A ./ \text{abs}(A)$. function.

Examples

```
sign(rand(2,3))  
sign(1+%i)
```

See Also

[abs](#)

Name

signm — matrix sign function

Description

For square and Hermitian matrices $X = \text{signm}(A)$ is matrix sign function.

Examples

```
A=rand(4,4);B=A+A';X=signm(B);spec(B),spec(X)
```

See Also

sign

Name

`sin` — sine function

```
[t]=sin(x)
```

Parameters

`x`
real or complex vector or matrix

Description

For a vector or a matrix, `sin(x)` is the sine of its elements. For matrix sine use `sinm(X)` function.

Examples

```
asin(sin([1,0,%i]))
```

See Also

`sinm`

Name

sinc — sinc function

```
t=sinc(x)
```

Parameters

x
real or complex vector or matrix

t
real or complex vector or matrix

Description

If x is a vector or a matrix, $t=\text{sinc}(x)$ is the vector or matrix such that $t(i)=\sin(x(i))/x(i)$ if $x(i)\neq 0$ and $t(i)=1$ if $x(i)=0$.

Examples

```
x=linspace(-10,10,3000);  
plot2d(x,sinc(x))
```

See Also

sin, cos

Name

sind — sine function, argument in degree.

```
t=sind(x)
```

Parameters

x
real vector or matrix

t
real vector or matrix with same dimensions as x

Description

For a vector or a matrix x , $\text{sind}(x)$ is the sine of its elements supposed to be given in degree. The results are in $[-1\ 1]$. For integers n , $\text{sind}(n*180)$ is exactly zero.

Examples

```
x=[0,30 45 60 90 360];  
sind(x)
```

See Also

[sin](#)

Authors

Serge Steer, INRIA

Name

`sinh` — hyperbolic sine

```
[t]=sinh(x)
```

Parameters

`x,t`
real or complex vectors/matrices

Description

the elements of vector `t` are the hyperbolic sine of elements of vector `x`.

Examples

```
asinh(sinh([0,1,%i]))
```

See Also

`asinh`

Name

sinhm — matrix hyperbolic sine

```
t=sinhm(x)
```

Parameters

x,t
real or complex square matrix

Description

sinhm(x) is the matrix hyperbolic sine of the matrix x. $t = (\expm(x) - \expm(-x))/2$.

Examples

```
A=[1,2;2,3]
asinhm(sinhm(A))
A(1,1)=%i;sinhm(A)-(expm(A)-expm(-A))/2 //Complex case
```

See Also

sinh

Name

sinm — matrix sine function

```
t=sinm(x)
```

Parameters

x
real or complex square matrix

Description

$\text{sinm}(x)$ is matrix sine of x matrix.

Examples

```
A=[1,2;2,4];  
sinm(A)+0.5*i*(expm(i*A)-expm(-i*A))
```

See Also

sin, asinm

Name

size — size of objects

```
y=size(x [,sel])  
[nr,nc]=size(x)
```

Parameters

x
matrix (including transfer matrix) or list or linear system (`syslin`)

y
1x2 integer vector or integer number

sel
a scalar or a character string

nr,nc
two integers

Description

Applied to :

a matrix (constant, polynomial, string, boolean, rational) **x**, with only one lhs argument `size` returns a 1x2 vector [number of rows, number of columns]. Called with LHS=2, returns **nr, nc** = [number of rows, number of columns]. `sel` may be used to specify what dimension to get:

1 or 'r'
to get the number of rows

2 or 'c'
to get the number of columns

'*'
to get the product of rows and column numbers

Applied to:

a list it returns the number of elements. In this case only `y=size(x)` syntax can be used

Applied to:

a linear system, `y=size(x)` returns in **y** the (row) vector [number of outputs, number if inputs] i.e. the dimension of the corresponding transfer matrix. The syntax `[nr,nc]=size(x)` is also valid (with `(nr,nc)=(y(1),y(2))`). If **x** is a linear system in state-space form, then `[nr,nc,nx]=size(x)` returns in addition the dimension **nx** of the **A** matrix of **x**.

an hypermatrix, `y=size(x)` returns the vector of hypermatrix dimensions. `[n1,n2,...nn]=size(x)` returns the hypermatrix dimensions. `ni=size(x,i)` returns the *i*th dimension and `size(x,'*')` returns the product of dimensions.

Examples

```
[n,m]=size(rand(3,2))  
[n,m]=size(['a','b';'c','d'])  
x=ssrand(3,2,4);[ny,nu]=size(x)  
[ny,nu]=size(ss2tf(x))  
[ny,nu,nx]=size(x)
```

```
// Returns the number of rows
n=size(rand(3,2),"r")
// Returns the number of columns
m=size(rand(3,2),"c")
// Returns the product of the dimensions
nm=size(rand(3,2),"*")
```

See Also

length, syslin

Name

solve — symbolic linear system solver

```
[x]=solve(A,b)
```

Parameters

A,b,x
matrix (resp. vectors) of character strings

Description

solves $A \cdot x = b$ when A is an upper triangular matrix made of character strings.

Examples

```
A=['1','a';'0','2']; //Upper triangular
b=['x';'y'];

w=solve(A,b)

a=1;x=2;y=5;
evstr(w)
inv([1,1;0,2])*[2;5]
```

See Also

trianfml

Name

sort — stable sorting by "quick sort" algorithm (DEPRECATED see gsort)

```
[s, [k]]=sort(v)
[s, [k]]=sort(v, 'r')
[s, [k]]=sort(v, 'c')
```

Parameters

v
real or complex vector/matrix; sparse vector; character string vector/matrix

s
real or complex vector or matrix; sparse vector; character string vector/matrix

k
vector or matrix of integers

Description

the `sort` implements a "bubble sort algorithm".

`sort` will be removed in Scilab 5.3. see `gsort`

`s=sort(v)` sorts `v` in decreasing order. If `v` is a matrix, sorting is done columnwise, `v` being seen as the stacked vector `v(:)`. If `v` is a string, sort is increasing order. `[s, k]=sort(v)` gives in addition the indices of entries of `s` in `v`, i.e. `v(k(:))` is the vector `s`.

`s=sort(v, 'r')` sorts the rows of `v` in decreasing order i.e. each column of `s` is obtained from each column of `v` by reordering it in decreasing order. `[s, k]=sort(v, 'r')` returns in addition in each column of `k` the indices such that `v(k(:, i), i)=s(:, i)` for each column index `i`.

`s=sort(v, 'c')` sorts the columns of `v` in decreasing order i.e. each row of `s` is obtained from each row of `v` by reordering it in decreasing order. `[s, k]=sort(v, 'c')` returns in addition in each row of `k` the indices such that `v(i, k(i, :))=s(i, :)` for each row index `i`.

Complex matrix or vectors are sorted by their magnitude. Column/row sorting is not implemented for complex matrices.

`y=sort(A)` is valid when `A` is a sparse vector. Column/row sorting is not implemented for sparse matrix.

Remark : `sort` is now obsolete it may be replaced by **`gsort`**.

Examples

```
[s,p]=sort(rand(1,10));
//p is a random permutation of 1:10
A=[1,2,5;3,4,2];
[Asorted,q]=sort(A);A(q(:))-Asorted(:)
v=1:10;
sort(v)
sort(v')
sort(v, 'r') //Does nothing for row vectors
sort(v, 'c')
```



See Also

find, gsort

Name

sp2adj — converts sparse matrix into adjacency form

Parameters

A
real or complex sparse matrix (nz non-zero entries)

xadj
integer vector of length (n+1).

adjncy
integer vector of length nz containing the row indices for the corresponding elements in anz

anz
column vector of length nz, containing the non-zero elements of A

Description

sp2adj converts a sparse matrix into its adjacency form (utility fonction).
A = n x m sparse matrix. xadj, adjncy, anz = adjacency representation of A i.e

$xadj(j+1) - xadj(j)$ = number of non zero entries in row j. adjncy = column index of the non zeros entries in row 1, row 2,..., row n. anz = values of non zero entries in row 1, row 2,..., row n. xadj is a (column) vector of size n+1 and adjncy is an integer (column) vector of size $nz=nnz(A)$. anz is a real vector of size $nz=nnz(A)$.

Examples

```
A = sprand(100,50,.05);  
[xadj,adjncy,anz]= sp2adj(A);  
[n,m]=size(A);  
p = adj2sp(xadj,adjncy,anz,[n,m]);  
A-p
```

See Also

adj2sp, sparse, spcompact, spget

Name

speye — sparse identity matrix

```
Isp=speye(nrows,ncols)
Isp=speye(A)
```

Parameters

nrows

integer (number of rows)

ncols

integer (number os columns)

A

sparse matrix

sp

sparse identity matrix

Description

`Isp=speye(nrows,ncols)` returns a sparse identity matrix `Isp` with `nrows` rows, `ncols` columns. (Non square identity matrix have a maximal number of ones along the main diagonal).

`Isp=speye(A)` returns a sparse identity matrix with same dimensions as `A`. If `[m,n]=size(A)`, `speye(m,n)` and `speye(A)` are equivalent. In particular `speye(3)` is not equivalent to `speye(3,3)`.

Examples

```
eye(3,3)-full(speye(3,3))
```

See Also

`sparse`, `full`, `eye`, `spzeros`, `spones`

Name

spones — sparse matrix

```
sp=spones(A)
```

Parameters

A
sparse matrix

sp
sparse matrix

Description

`sp=spones(A)` generates a matrix with the same sparsity structure as A, but with ones in the nonzero positions.

Examples

```
A=sprand(10,12,0.1);  
sp=spones(A)  
B = A~=0  
bool2s(B)
```

See Also

`sparse`, `full`, `eye`, `speye`, `spzeros`

Name

sprand — sparse random matrix

```
sp=sprand(nrows,ncols,fill [,typ])
```

Parameters

nrows

integer (number of rows)

ncols

integer (number of columns)

fill

filling coefficient (density)

typ

character string ('uniform' (default) or 'normal')

sp

sparse matrix

Description

`sp=sprand(nrows,ncols,fill)` returns a sparse matrix `sp` with `nrows` rows, `ncols` columns and approximately `fill*nrows*ncols` non-zero entries.

If `typ='uniform'` uniformly distributed values on `[0,1]` are generated. If `typ='normal'` normally distributed values are generated (mean=0 and standard deviation=1).

Examples

```
w=sprand(100,1000,0.001);
```

See Also

sparse, full, rand, speye

Name

spzeros — sparse zero matrix

```
sp=spzeros(nrows,ncols)
sp=spzeros(A)
```

Parameters

nrows

integer (number of rows)

ncols

integer (number os columns)

A

sparse matrix

sp

sparse zero matrix

Description

`sp=spzeros(nrows,ncols)` returns a sparse zero matrix `sp` with `nrows` rows, `ncols` columns. (Equivalent to `sparse([],[],[nrow,ncols])`)

`sp=spzeros(A)` returns a sparse zero matrix with same dimensions as `A`. If `[m,n]=size(A)`, `spzeros(m,n)` and `spzeros(A)` are equivalent. In particular `spzeros(3)` is not equivalent to `spzeros(3,3)`.

Examples

```
sum(spzeros(1000,1000))
```

See Also

`sparse`, `full`, `eye`, `speye`, `spones`

Name

sqrt — square root

```
y=sqrt(x)
```

Parameters

x
real or complex scalar or vector

Description

`sqrt(x)` is the vector of the square root of the x elements. Result is complex if x is negative.

Examples

```
sqrt([2,4])  
sqrt(-1)
```

See Also

hat, sqrtm

Name

sqrtm — matrix square root

```
y=sqrtm(x)
```

Parameters

x
real or complex square matrix

Description

$y=\text{sqrt}(x)$ is the matrix square root of the $x \times x$ matrix ($x=y^2$) Result may not be accurate if x is not symmetric.

Examples

```
x=[0 1;2 4]
w=sqrtm(x);
norm(w*w-x)
x(1,2)=%i;
w=sqrtm(x);norm(w*w-x,1)
```

See Also

expm, sqrt

Name

squarewave — generates a square wave with period 2π

```
x=squarewave(t [,percent])
```

Parameters

t

real vector, time discretization

x

real vector, the wave value at each time point in set $(-1,+1)$

percent

real positive scalar, the percent of the period in which the signal is positive. Defaut value is 50

Description

`squarewave(t)` generates the vector of the values of the square wave with period 2π at each date given in the `t` vector.

`squarewave(t, %)` generates a square wave such that % is the percent of the period in which the signal is positive.

Examples

```
t=(0:0.1:5*pi)';  
plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
```

See Also

sin, cos

Name

ssrand — random system generator

```
sl=ssrand(nout,nin,nstate)
[sl,U]=ssrand(nout,nin,nstate,flag)
```

Parameters

nout
integer (number of output)

nin
integer (number of input)

nstate
integer (dimension of state-space)

flag
list made of one character string and one or several integers

sl
list (syslin list)

U
square (nstate x nstate) nonsingular matrix

Description

`sl=ssrand(nout,nin,nstate)` returns a random strictly proper ($D=0$) state-space system of size `[nout,nint]` represented by a syslin list and with `nstate` state variables.

`[sl,U]=ssrand(nout,nin,nstate,flag)` returns a test linear system with given properties specified by `flag`. `flag` can be one of the following:

```
flag=list('co',dim_cont_subspace)
flag=list('uo',dim_unobs_subspace)
flag=list('ncno',dim_cno,dim_ncno,dim_co,dim_nco)
flag=list('st',dim_cont_subspace,dim_stab_subspace,dim_stab0)
flag=list('dt',dim_inst_unob,dim_instb0,dim_unobs)
flag=list('on',nr,ng,ng0,nv,rk)
flag=list('ui',nw,nwu,nwui,nwuis,rk)
```

The complete description of the Sys is given in the code of the `ssrand` function (in SCIDIR/macros/UTIL). For example with `flag=list('co',dim_cont_subspace)` a non-controllable system is returned and `dim_cont_subspace` is the dimension of the controllable subspace of Sys. The character strings 'co', 'uo', 'ncno', 'st', 'dt', 'on', 'ui' stand for "controllable", "unobservable", "non-controllable-non-observable", "stabilizable", "detectable", "output-nulling", "unknown-input".

Examples

```
//flag=list('st',dim_cont_subspace,dim_stab_subspace,dim_stab0)
//dim_cont_subspace<=dim_stab_subspace<=dim_stab0
//pair (A,B) U-similar to:
```

```
//      [*,*,*,*;      [*;  
//      [0,s,*,*;      [0;  
//A=    [0,0,i,*;      B=[0;  
//      [0,0,0,u]      [0]  
//  
// (A11,B1) controllable s=stable matrix i=neutral matrix u=unstable matrix  
[Sl,U]=ssrand(2,3,8,list('st',2,5,5));  
w=ss2ss(Sl,inv(U)); //undo the random change of basis => form as above  
[n,nc,u,sl]=st_ility(Sl);n,nc
```

See Also

syslin

Name

sub2ind — matrix subscript values to linear index

```
I = sub2ind(dims,i1,i2,...)
J = sub2ind(dims,Mi)
```

Parameters

dims

vector: the matrix dimensions

i1,i2,...

the subscript value arrays(same matrix shape as **I**)

Mi

matrix whose columns contains the subscript values.

I

the linear index array

Description

`sub2ind` is used to determine the equivalent single index corresponding to a given set of subscript values. `I = sub2ind(dims,i1,i2,...)` returns the linear index equivalent to the row, column, ... subscripts in the arrays `i1,i2,..` for an matrix of size `dims`. In this case `i1,i2,..` must have the same shape and the result `I` has the same matrix shape. `I = sub2ind(dims,Mi)` returns the linear index equivalent to the subscripts in the columns of the matrix `Mi` for a matrix of size `dims`. in this case `I` is a column vector.

Examples

```
i=[1 2 1 1 2 1 1];
j=[1 2 3 1 2 3 3];
k=[1 2 1 2 1 2 1];
sub2ind([2,3,2],i,j,k)

sub2ind([2,3,2],[i',j',k'])
```

See Also

`ind2sub`, `extraction`, `insertion`

Authors

Serge Steer, INRIA

Name

sum — sum (row sum, column sum) of vector/matrix entries

```
y=sum(x)
y=sum(x, 'r') or y=sum(x,1)

y=sum(x, 'c') or y=sum(x,2)

y=sum(x, 'm')
```

Parameters

x
vector or matrix (real, complex, sparse or polynomial)

y
scalar or vector

Description

For a vector or a matrix x , $y=\text{sum}(x)$ returns in the scalar y the sum of all the entries of x .

$y=\text{sum}(x, 'r')$ (or, equivalently, $y=\text{sum}(x, 1)$) is the rowwise sum: $y(j) = \text{sum}(x(:, j))$.
 y is a row vector

$y=\text{sum}(x, 'c')$ (or, equivalently, $y=\text{sum}(x, 2)$) is the columnwise sum. It returns in each entry of the column vector y the sum: $y(i) = \text{sum}(x(i, :))$.

$y=\text{sum}(x, 'm')$ is the sum along the first non singleton dimension of x (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];
trace(A)-sum(diag(A))
sum(A, 'c')-A*ones(2,1)
sum(A+%i)
A=sparse(A);sum(A, 'c')-A*ones(2,1)
s=poly(0, 's');
M=[s,%i+s;s^2,1];
sum(M), sum(M,2)
```

See Also

cumsum, prod

Name

sysconv — system conversion

```
[s1, s2]=sysconv(s1, s2)
```

Parameters

s1,s2
list (linear `syslin` systems)

Description

Converts `s1` and `s2` into common representation in order that system interconnexion operations can be applied. Utility function for experts. The conversion rules in given in the following table.

"c"
continuous time system

"d"
discrete time system

n
sampled system with sampling period n

[]
system with undefined time domain For mixed systems `s1` and `s2` are put in state-space representation.

s1\s2	"c"	"d"	n2	[]
"c"	nothing	uncompatible	c2e(s1,n2)	c(s2)
"d"	uncompatible	nothing	e(s1,n2)	d(s2)
n1	c2e(s2,n1)	e(s2,n1)	n1<>n2 uncomp n1=n2 nothing	e(s2,n1)
[]	c(s1)	d(s1)	e(s1,n2)	nothing

With the following meaning:

n1,n2
sampling period

c2e(s,n)
the continuous-time system `s` is transformed into a sampled system with sampling period `n`.

c(s)
conversion to continuous (time domain is "c")

d(s)
conversion to discrete (time domain is "d")

e(s,n)
conversion to samples system with period `n`

Examples

```
s1=ssrand(1,1,2);  
s2=ss2tf(s1);  
[s1,s2]=sysconv(s1,s2);
```

See Also

syslin, ss2tf, tf2ss

Name

sysdiag — block diagonal system connection

```
r=sysdiag(a1,a2,...,an)
```

Description

Returns the block-diagonal system made with subsystems put in the main diagonal

a_i
subsystems (i.e. gains, or linear systems in state-space or transfer form)

Used in particular for system interconnections.

Remark

At most 17 arguments.

Examples

```
s=poly(0,'s')
sysdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sysdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
```

See Also

brackets, insertion, feedback

Name

syslin — linear system definition

```
[sl]=syslin(dom,A,B,C [,D [,x0] ])  
[sl]=syslin(dom,N,D)  
[sl]=syslin(dom,H)
```

Parameters

dom

character string ('c', 'd'), or [] or a scalar.

A,B,C,D

matrices of the state-space representation (D optional with default value zero matrix). For improper systems D is a polynomial matrix.

x0

vector (initial state; default value is 0)

N, D

polynomial matrices

H

rational matrix or linear state space representation

sl

tlist("syslin" list) representing the linear system

Description

syslin defines a linear system as a list and checks consistency of data.

dom specifies the time domain of the system and can have the following values:

dom='c' for a continuous time system, dom='d' for a discrete time system, n for a sampled system with sampling period n (in seconds).

dom=[] if the time domain is undefined

State-space representation:

```
sl=syslin(dom,A,B,C [,D [,x0] ])
```

represents the system :

$$\begin{aligned} s \cdot x &= A \cdot x + B \cdot u \\ y &= C \cdot x + D \cdot u \\ x(0) &= x0 \end{aligned}$$

The output of syslin is a list of the following form:
sl=tlist(['lss', 'A', 'B', 'C', 'D', 'x0', 'dt'], A, B, C, D, x0, dom) Note that D is allowed to be a polynomial matrix (improper systems).

Transfer matrix representation:

```
sl=syslin(dom,N,D)
sl=syslin(dom,H)
```

The output of `syslin` is a list of the following form :
`sl=tlist(['r','num','den','dt'],N,D,dom)` or
`sl=tlist(['r','num','den','dt'],H(2),H(3),dom).`

Linear systems defined as `syslin` can be manipulated as usual matrices (concatenation, extraction, transpose, multiplication, etc) both in state-space or transfer representation.

Most of state-space control functions receive a `syslin` list as input instead of the four matrices defining the system.

Examples

```
A=[0,1;0,0];B=[1;1];C=[1,1];
S1=syslin('c',A,B,C) //Linear system definition
S1("A") //Display of A-matrix
S1("X0"), S1("dt") // Display of X0 and time domain
s=poly(0,'s');
D=s;
S2=syslin('c',A,B,C,D)
H1=(1+2*s)/s^2, S1bis=syslin('c',H1)
H2=(1+2*s+s^3)/s^2, S2bis=syslin('c',H2)
S1+S2
[S1,S2]
ss2tf(S1)-S1bis
S1bis+S2bis
S1*S2bis
size(S1)
```

See Also

`tlist`, `lsslist`, `rlist`, `ssrand`, `ss2tf`, `tf2ss`, `dscr`, `abcd`

Name

tan — tangent

```
[t]=tan(x)
```

Parameters

x
vector or matrix

t
vector or matrix

Description

The elements of **t** are the tangent of the elements of **x**.

Examples

```
x=[1,%i, -1, -%i]  
tan(x)  
sin(x)./cos(x)
```

See Also

atan, tanm

Name

tand — tangent, argument in degree.

```
t=tand(x)
```

Parameters

x
real vector or matrix

t
real vector or matrix

Description

The elements of **t** are the tangent of the elements of **x**.

Examples

```
mod=ieee();ieee(2);  
x=[0,30 45 60 90 360];  
tand(x)
```

See Also

atand, tan

Name

`tanh` — hyperbolic tangent

```
t=tanh(x)
```

Description

the elements of `t` are the hyperbolic tangents of the elements of `x`.

Examples

```
x=[1,%i, -1, -%i]
tanh(x)
sinh(x)./cosh(x)
```

See Also

`atanh`, `tan`, `tanhm`

Name

`tanhm` — matrix hyperbolic tangent

```
t=tanhm(x)
```

Parameters

`x,t`
real or complex square matrix

Description

`tanhm` is the matrix hyperbolic tangent of the matrix `x`.

Examples

```
A=[1,2;3,4];  
tanhm(A)
```

See Also

`tan`, `tanm`, `expm`, `sinm`, `cosm`, `atanhm`

Name

tanm — matrix tangent

```
[t]=tanm(x)
```

Parameters

x
square real or complex matrix

t
square matrix

Description

tanm(x) is the matrix tangent of the square matrix x.

Examples

```
A=[1,2;3,4];  
tanm(A)
```

See Also

tan, expm, sinm, atanm

Name

toeplitz — toeplitz matrix

```
A=toeplitz(c [,r])
```

Parameters

a,c,r
constant, polynomial or string matrices

Description

returns the Toeplitz matrix whose first row is **r** and first column is **c**. **c(1)** must be equal to **r(1)**.
toeplitz(c) returns the symmetric Toeplitz matrix.

Examples

```
A=toeplitz(1:5);  
  
T=toeplitz(1:5,1:2:7);T1=[1 3 5 7;2 1 3 5;3 2 1 3;4 3 2 1;5 4 3 2];  
T-T1  
  
s=poly(0,'s');  
t=toeplitz([s,s+1,s^2,1-s]);  
t1=[s,1+s,s*s,1-s;1+s,s,1+s,s*s;s*s,1+s,s,1+s;1-s,s*s,1+s,s]  
t-t1  
  
t=toeplitz(['1','2','3','4']);  
t1=['1','2','3','4';'2','1','2','3';'3','2','1','2';'4','3','2','1']
```

See Also

matrix

Name

trfmod — poles and zeros display

```
[hm]=trfmod(h [,job])
```

Description

To visualize the pole-zero structure of a SISO transfer function h .

job='p'
visualization of polynomials (default)

job='f'
visualization of natural frequencies and damping

Interactive simplification of h . `trfmod` opens a dialog window.

See Also

poly , simp

Name

trianfml — symbolic triangularization

```
[f [,sexp]]=trianfml(f [,sexp])
```

Description

Symbolic triangularization of the matrix **f** ; triangularization is performed by elementary row operations; **sexp** is a set of common expressions stored by the algorithm.

Examples

```
A=['1','2';'a','b']  
W=trianfml([A,string(eye(2,2))])  
U=W(:,3:4)  
a=5;b=6;  
A=evstr(A)  
U=evstr(U)  
U*A  
evstr(W(:,1:2))
```

See Also

addf, mulc, solve, trisolve

Name

`tril` — lower triangular part of matrix

```
tril(x [,k])
```

Parameters

`x`
matrix (real, complex, polynomial, rational)

`k`
integer (default value 0)

Description

Lower triangle part of a matrix. `tril(x,k)` is made by entries below the k th diagonal : $k>0$ (upper diagonal) and $k<0$ (diagonals below the main diagonal).

Examples

```
s=poly(0,'s');  
tril([s,s;s,1])  
tril([1/s,1/s;1/s,1])
```

See Also

`triu`, `ones`, `eye`, `diag`

Name

trisolve — symbolic linear system solver

```
[x [,sexp]] = trisolve(A,b [,sexp])
```

Parameters

A,b
matrices of strings

Description

symbolically solves $A \cdot x = b$, A being assumed to be upper triangular.

sexp is a vector of common subexpressions in A, b, x.

Examples

```
A=['x','y';'0','z'];b=['0';'1'];  
w=trisolve(A,b)  
x=5;y=2;z=4;  
evstr(w)  
inv(evstr(A))*evstr(b)
```

See Also

trianfml, solve

Authors

F.D, S.S

Name

triu — upper triangle

Description

Upper triangle. See tril.

Examples

```
s=poly(0, 's');  
triu([s,s;s,1])  
triu([1/s,1/s;1/s,1])
```

See Also

tril, ones, eye, diag

Name

typeof — object type

```
[t]=typeof(object)
```

Parameters

object
Scilab object

t
string

Description

`t=typeof(object)` returns one of the following strings:

"constant"
if object is a real or complex constant matrix.

"polynomial"
if object is a polynomial matrix.

"function"
if object is a function (Scilab code).

"handle"
if object is an handle.

"string"
if object is a matrix made of character strings.

"boolean"
if object is a boolean matrix.

"list"
if object is a list.

"rational"
if object is a rational matrix (transfer matrix).

"state-space"
if object is a state-space model (see `syslin`).

"sparse"
if object is a (real) sparse matrix.

"boolean sparse"
if object is a boolean sparse matrix.

"hypermat"
if object is an hypermatrix (N-dimension array with $N \geq 3$).

"st"
if object is a structure array.

"ce"
if object is a cell array.

the first string in the first list entry
if object is a tlist or mlist.

"fptr"
if object is a Scilab intrinsic (C or Fortran code).

"pointer"
if object is a pointer (See lufact).

"size implicit"
if object is a size implicit polynom used for indexing.

"library"
if object is function library.

"int8" or "uint8" or "int16" or "uint16" or "int32" or "uint32"
if object is a matrix of [unsigned] integers stored on 8, 16 or 32 bits. See int8)

Examples

```
typeof(1)
typeof(poly(0, 'x'))

typeof(1/poly(0, 'x'))
typeof(%t)

w=sprand(100,100,0.001);
typeof(w)
typeof(w==w)

deff('y=f(x)', 'y=2*x');
typeof(f)

L=tlist(['V', 'a', 'b'], 18, 'Scilab');
typeof(L)

typeof(corelib)
```

See Also

type, strings, syslin, poly

Name

union — extract union components of a vector

```
[v [,ka, kb] ] = union(a,b)
[v [,ka, kb] ] = union(a,b,orient)
```

Parameters

a
vector or matrix of numbers or strings

b
vector of real numbers or strings

orient
flag with possible values : 1 or "r", 2 or "c".

v
row vector or matrix of numbers or strings

ka
row vector of integers

kb
row vector of integers

Description

`union(a,b)` returns a sorted row vector which retains the unique entries of `[a(:);b(:)]`.

`union(a,b,"r")` or `union(a,b,1)` returns the matrix formed by the union of the unique rows of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of columns.

`union(a,b,"c")` or `union(a,b,2)` returns the matrix formed by the union of the unique columns of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of rows.

`[v, ka, kb]=union(a,b)` also returns index vectors `ka` and `kb` such that `v` is a sorted combination of the entries `a(ka)` and `b(kb)`.

Examples

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

union(A,B)
[N, ka, kb]=union(A,B)

union('a'+string(A), 'b'+string(B))
```

See Also

unique, gsort

Name

unique — extract unique components of a vector or matrices

```
[N [,k]]=unique(M)
[N [,k]]=unique(M ,orient)
```

Parameters

M
vector or matrix of numbers or strings

orient
flag with possible values : 1 or "r", 2 or "c"

N
vector or matrix of numbers or strings

k
vector of integers

Description

`unique(M)` returns a vector which retains the unique entries of **M** in ascending order.

`unique(M, "r")` or `unique(M, 1)` returns the unique rows of **M** in lexicographic ascending order.

`unique(M, "c")` or `unique(M, 2)` returns the unique columns of **M** in lexicographic ascending order.

If required the output argument **k** contains the position of the first encountered unique entries.

Examples

```
M=round(2*rand(20,1));

unique(M)
[N,k]=unique(M)

unique(string(M))
[N,k]=unique(string(M))

A = [0,0,1,1;
      0,1,1,1;
      2,0,1,1;
      0,2,2,2;
      2,0,1,1;
      0,0,1,1];
T='x'+string(A);

//unique rows

[m,k]=unique(A, 'r')
unique(T, 'r')
```

```
//unique columns  
[m,k]=unique(T, 'c')  
unique(A, 'c')
```

See Also

[union](#), [intersect](#), [gsort](#), [lex_sort](#)

Name

vectorfind — finds in a matrix rows or columns matching a vector

```
ind = vectorfind(m,v,job)
```

Parameters

m
a matrix of any type

v
vector of any type

job
string flag with possible values "r" to look for matching rows or "c" to look for matching columns

ind
row vector containing indices of matching rows or columns

Description

finds in a matrix rows or columns matching a vector.

Examples

```
alr=[1,2,2;  
     1,2,1;  
     1,1,2;  
     1,1,1;  
     1,2,1];  
  
ind = vectorfind(alr,[1,2,1],'r')  
ind = vectorfind(string(alr),string([1,2,1]),'r')
```

See Also

find, gsort

Authors

R. Nikoukhah, S. Steer INRIA

Name

zeros — matrix made of zeros

```
y=zeros()  
y=zeros(x)  
y=zeros(m1,m2,...)
```

Parameters

x,y
matrices

m1, m2,...
integers

Description

Creates matrix of zeros (same as $0 \times \text{ones}$).

zeros(m1,m2)
for an (m1, m2) matrix.

zeros(m1,m2,...,mn)
creates a (m1, m2, . . . , mn) matrix filled with zeros

zeros(A)
for a matrix of same size of A.

zeros(3)
is zeros(a) with a=3 i.e it is NOT a 3x3 matrix!

zeros()
returns a single zero

If x is a `syslin` list (linear system in state-space or transfer form), `zeros(x)` is also valid and returns a zero matrix.

Examples

```
zeros(3)  
zeros(3,3)  
zeros(2,3,2)
```

See Also

eye, ones, spzeros

Part IV. Functions

Nom

`add_profiling` — Adds profiling instructions to a function.

```
add_profiling(funname)
```

Parameters

`funname`

A character string, the name of the function

Description

`add_profiling(funname)` Adds profiling instructions to the function named `funname`. Then when this function is run the number of calls, the time spent is stored for each function line.

Examples

```
function x=foo(a,n)
    x=0;
    for i=1:n
        if x<10 then
            x=x+a
        else
            x=x+1
        end
    end
    x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
```

See Also

`profile`, `plotprofile`, `remove_profiling`, `reset_profiling`

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Nom

bytecode — given a function returns the "bytecode" of a function in a Scilab array and conversely.

```
x = bytecode(f)
f = bytecode(X)
```

Parameters

f
A scilab function.

x
an int32 row vector

Description

`x = bytecode(f)` returns the "bytecode" of the function `f` in the Scilab integer array `x`.

`f = bytecode(x)` returns in `f` the function associated with the "bytecode" given in the Scilab integer array `x`. Warning the validity of `x` is not checked.

Remark

The bytecode of Scilab function will evolve drastically in the future, So the use of this function should be restricted to the profiling instruction handling.

Examples

```
function a=foo(),a=sin(3),endfunction
bytecode(foo)
```

See Also

`add_profiling` , `bytecodewalk` , `macr2lst` , `macr2tree`

Authors

Serge Steer, INRIA

Nom

bytecodewalk — walk in function bytecode applying transformation.

```
c1 = bytecodewalk(code, query, job)
```

Parameters

code

int32 vector: input byte code array

query

integer, the opcode to look for

job

the operation to perform, for the requested opcode

c1

int32 vector: output byte code array

Description

walk in function bytecode applying transformation.

See Also

bytecode

Authors

Serge Steer INRIA

Name

deff — on-line definition of function

```
deff(' [s1, s2, ...]=newfunction(e1,e2,...)',text [,opt])
```

Parameters

e1,e2,...,

input variables.

s1,s2,...,

output variables.

text

matrix of character strings

opt

optional character string

'c'

function is "compiled" to be more efficient (default)

'p'

function is "compiled" and prepared for profiling (see profile)

'n'

function is not "compiled"

Description

`deff` can be used to define functions from sequences of instructions written in text strings. The resulting function object has the same properties of any other function defined in a text file and loaded with `exec` or `execf`.

Quotes in the instructions (delimiting strings or meaning matrix transposition) have to be doubled to be interpreted correctly (see `help quote`). This can make writing up a little awkward. An option in such cases is to define functions in files as usual, to load them into Scilab by `exec` (with the `'n'` option) and to use `sci2exp` to get a printout of corresponding `deff` instructions.

Examples

```
deff(' [x]=myplus(y,z)', 'x=y+z')  
deff(' [x]=mymacro(y,z)', ['a=3*y+1'; 'x=a*z+y'])
```

See Also

`comp` , `exec` , `function` , `profile`

Name

`exec` — script file execution

```
exec(path [,mode])
exec(fun [,mode])
ierr=exec(path, 'errcatch' [,mode])
ierr=exec(fun, 'errcatch' [,mode])
```

Parameters

`path`
a string, the path of the script file

`mode`
an integer scalar, the execution mode (see below)

`fun`
a scilab function

`ierr`
integer, 0 or error number

Description

`exec(path [,mode])` executes sequentially the scilab instructions contained in the file given by `path` with an optional execution mode `mode`.

The different cases for `mode` are :

0 : the default value

-1 : nothing is printed

1 : echo of each command line

2 : prompt `-->` is printed

3 : echoes + prompts

4 : stops before each prompt. Execution resumes after a carriage return.

7 : stops + prompts + echoes : useful mode for demos.

`exec(fun [,mode])` executes function `fun` as a script: no input nor output argument nor specific variable environment. This form is more efficient, because script code may be pre-compiled (see `comp`). This method for script evaluation allows to store scripts as function in libraries.

If an error is encountered while executing, if 'errcatch' flag is present `exec` issues no error message, aborts execution of the instructions and resumes with `ierr` equal to the error number. If 'errcatch' flag is not present, standard error handling works.

Remark

`exec` files may now be used to define functions using the inline function definition syntax (see `function`).

Examples

```
// create a script file
mputl('a=1;b=2',TMPDIR+'/myscript')

// execute it
exec(TMPDIR+'/myscript')
whos -name "a "

// create a function
deff('y=foo(x)', 'a=x+1;y=a^2')
clear a b
// call the function
foo(1)
// a is a variable created in the environment of the function foo
// it is destroyed when foo returns
whos -name "a "

x=1 //create x to make it known by the script foo
exec(foo)

// a and y are created in the current environment
whos -name "a "
```

See Also

execstr , evstr , comp , mode , chdir , pwd

Name

`execstr` — execute Scilab code in strings

```
execstr(instr)
ierr=execstr(instr,'errcatch' [,msg])
```

Parameters

`instr`

vector of character strings, Scilab instruction to be executed.

`ierr`

integer, 0 or error number.

`msg`

character string with values 'm' or 'n'. Default value is 'n'.

Description

Executes the Scilab instructions given in argument `instr`.

Note that `instr` should not make use of continuation marks (..)

If the 'errcatch' flag is not present, error handling works as usual.

If the 'errcatch' flag is set, and an error is encountered while executing the instructions defined in `instr`, `execstr` issues no error message, but aborts execution of the `instr` instructions (at the point where the error occurred), and resumes with `ierr` equal to the error number. In this case the display of the error message is controlled by the `msg` option:

"m"

error message is displayed and recorded.

"n"

no error message is displayed, but the error message is recorded (see `lasterror`). This is the default.

`ierr=execstr(instr,'errcatch')` can handle syntactical errors. This is useful for evaluation of instruction obtained by a query to the user.

Examples

```
execstr('a=1') // sets a=1.
execstr('1+1') // does nothing (while evstr('1+1') returns 2)

execstr(['if %t then';
        '  a=1';
        '  b=a+1';
        'else'
        '  b=0'
        'end'])

execstr('a=zzzzzzz','errcatch')
execstr('a=zzzzzzz','errcatch','m')
```

```
//syntax errors
execstr('a=1?02','errcatch')
lasterror(%t)

execstr('a=[1 2 3]','errcatch')
lasterror(%t)

// variable1 does not exist
if execstr('variable1;', 'errcatch')<>0 then disp("Trigger an error"),end

// variable2 exists ... no error is triggered by execstr
variable2=[2,3];
if execstr('variable2;', 'errcatch')<>0 then
    disp("Trigger an error");
else
    disp("execstr is happy");
end
```

See Also

evstr , lasterror , error , try

Name

fun2string — generates ascii definition of a scilab function

```
txt=fun2string(fun,name)
```

Parameters

fun

a function type variable

name

a character string, the generated function name

txt

a column vector of strings, the text giving the scilab instructions

Description

Given a loaded Scilab function pseudo-code fun2string allows to re-generate the code. The generated code is indented and beautified.

The mechanism is similar, but simpler than the mfile2sci one. It may be adapted for syntax translations.

Examples

```
txt=fun2string(asinh,'foo');  
write(%io(2),txt,'(a)')
```

See Also

exec , edit , macrovar

Name

function — opens a function definition
endfunction — closes a function definition

Description

```
function <lhs_arguments>=<function_name><rhs_arguments>  
    <statements>  
endfunction
```

Where

<function_name>
stands for the name of the function

<rhs_arguments>
stands for the input argument list. It may be

- a comma separated sequence of variable names enclosed in parenthesis, like (x1, . . . , xm). Last variable name can be the key word **varargin** (see varargin)
- the sequence () or nothing, if the function has no input argument.

<lhs_arguments>
stands for the output argument list. It may be

- a comma separated sequence of variable names enclosed in brackets, like [y1, . . . , yn]. Last variable name can be the key word **varargout** (see varargout)
- the sequence [] ,if the function has no input argument. In this case the syntax may also be:
function <function_name><rhs_arguments>

<statements>
stands for a set of scilab instructions (statements) This syntax may be used to define function (see functions) inline or in a script file (see exec). For compatibility with old Scilab versions, functions defined in a script file containing only function definitions can be "loaded" into Scilab using the **exec** function.

The function <lhs_arguments>=<function_name><rhs_arguments> sequence cannot be split over several lines. This sequence can be followed by statements in the same line if a comma or a semicolon is added at its end.

function definitions can be nested

Examples

```
//inline definition (see function)  
function [x,y]=myfct(a,b)  
x=a+b  
y=a-b  
endfunction  
  
[x,y]=myfct(3,2)  
  
//a one line function definition
```

```
function y=sq(x),y=x^2,endfunction

sq(3)

//nested functions definition
function y=foo(x)
a=sin(x)
function y=sq(x), y=x^2,endfunction
y=sq(a)+1
endfunction

foo(%pi/3)

// definition in an script file (see exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

See Also

functions , exec , exec

Name

functions — Scilab procedures and Scilab objects

Description

Functions are Scilab procedures ("macro", "function" and "procedure" have the same meaning).

Function definition

Usually, they are defined in files with an editor and loaded into Scilab using the `exec` function or through a library (see `lib` or `genlib`). But They can also be defined on-line (see `deff` or `function`). A function is defined by two components:

- a "syntax definition" part as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
function [y1,...,yn,varargout]=foo(x1,...,xm,varargin)
```

- a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The `yi` are output variables calculated as functions of input variables `xi` and variables existing in Scilab when the function is executed.

Calling function

- Usually function calling syntax is `[y1,...,yn]=foo(x1,...,xm)`. Shorter input or output argument list than definition ones may be used. In such cases, only the first variables from the left are used or set.

The `argn` function may be used to get the actual number of calling arguments.

- It is possible to define function with indeterminate number of input or output maximum number of arguments. This can be done using the `varargin` and `varargout` keywords. See the given links for details.
- It is also possible to use "named argument" to specify input arguments: suppose function `fun1` defined as `function y1=fun1(x1,x2,x3)` then it can be called with a syntax like `y=fun1(x1=33,x3=[1 2 3])` within `fun1` `x2` will be undefined,

it can also be called with a syntax like `y=fun1(x1=33,y='foo')`. In such a case the `y` variable will be available in the context of the function `fun1`. Note that the maximum number of argument must be less or equal to the number of formal input argument used in the function syntax part.

It is possible to check for defined variables with the `exists` function.

- When a function has no left hand side argument and is called only with character string arguments, the calling syntax may be simplified:

```
fun('a','toto','a string')
```

is equivalent to:

```
fun a toto 'a string'
```

Miscellaneous

Functions are Scilab objects (with type numbers 13 or 11). They can be manipulated (built, saved, loaded, passed as arguments,..) as other variable types.

Collections of functions can be collected in libraries. Functions which begin with % sign (e.g. %foo) are often used to overload (see overloading) operations or functions for new data type.

Examples

```
//inline definition (see function)
function [x,y]=myfct(a,b)
    x=a+b
    y=a-b
endfunction

[x,y]=myfct(3,2)

//inline definition (see deff)
deff('[x,y]=myfct(a,b)', ['x=a+b';
                        'y=a-b'])
// definition in an ascii file (see exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

See Also

function , deff , exec , comp , lib , getd , genlib , exists , varargin , varargout

Name

`genlib` — build library from functions in given directory

```
genlib(lib_name [[,dir_name, [ Force [,verb [,Names]]]])  
genlib(lib_name [,path=dir_name] [,verbose=verb] [,force=Force] [,names=Names])
```

Parameters

`lib_name`:

Scilab string. The variable name of the library to (re)create.

`dir_name`:

Scilab string. The name of the directory to look for `.sci`-files.

`Force`

boolean value (default value is %f). Set it to %t to force the sci-files recompilation.

`verb`

boolean values (default value is %f). Set it to %t to get information.

`Names`

a vector of strings, the names of function to include in the library. By default all the sci-files are taken into account

Description

For each `.sci` file in `dir_name` (or only those specified by the `Names` argument), `genlib` executes a `exec` and saves the functions to the corresponding `.bin` file. The `.sci` file must not contain anything but Scilab functions. If a `.bin` file is newer than the associated `.sci` file, `genlib` does not translate and save the file.

This default behaviour can be changed if `force` is given and set to %t. In this latter case the recompilation is always performed for each `.sci` file.

When all `.sci` files have been processed, `genlib` creates a library variable named `lib_name` and saves it in the file `lib` in `dir_name`. If the Scilab variable `lib_name` is not protected (see `predef`) this variable is updated.

If `verbose` is set to %t information are displayed during the build process.

If `dir_name` argument is not given and if `lib_name` Scilab variable exists and it is a library `dir_name` is taken equal to the `lib_name` library path (update mode).

Restrictions

Scilab tacitly assumes that file `foo.sci` defines at least a function named `foo`. If subsidiary functions are included, they are made known to Scilab only after the function `foo` had been referenced.

See Also

`getd` , `exec` , `save` , `lib`

Name

get_function_path — get source file path of a library function

```
path=get_function_path(fun_name)
```

Parameters

fun_name

a string, the name of the function

path

a string, the absolute pathname of the function source file (.sci) or [].

Description

Given the name of a function get_function_path returns the absolute pathname of the function source file if the function is defined in a Scilab library (see lib) or [] if name does not match any library function.

Examples

```
get_function_path('median')
```

See Also

lib , string

Name

getd — getting all functions defined in a directory

```
getd(path)
```

Parameters

`path`
Scilab string. The directory pathname

Description

loads all `.sci` files (containing Scilab functions) defined in the `path` directory.

Examples

```
getd('SCI/modules/cacsd/macros')
```

See Also

`exec` , `lib` , `pwd` , `chdir`

Name

getf — defining a function from a file

```
getf(file-name [,opt])
```

Parameters

filename

Scilab string.

opt

optional character string

"c"

loaded functions are "compiled" to be more efficient (default)

"n"

loaded functions are not "compiled"

"p"

loaded functions are "compiled" and prepared for profiling (see profile)

Description

WARNING: this function has been deprecated (see `exec` as a replacement of `getf`). `getf` will be removed in Scilab 5.3

loads one or several functions (see `functions`) defined in the file '`file-name`'. The string `opt='n'` means that the functions are not compiled (pre-interpreted) when loaded. This can be useful for some debugging purpose (see `comp`). By default, functions are compiled when loaded (i.e. `opt='c'` is used).

In the file a function must begin by a "syntax definition" line as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The following lines contain a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The `yi` are output variables calculated as functions of input variables `xi` and variables existing in Scilab when the function is executed. Shorter input or output argument list may be used.

Many functions may be written in the same file. A function is terminated by an `endfunction` keyword. For compatibility with previous versions a function may also be terminated by the following `function` keyword or the EOF mark. For that reason it is not possible to load function containing nested function definition using the `getf` function.

`getf` is an obsolete way for loading functions into scilab from a file. It is replaced by the function `exec`. Note that functions in a file should be terminated by an `endfunction` keyword. The `exec` function supposes `opt=='c'`.

To prepare a function for profiling please use the `add_profiling` function.

Examples

```
getf('SCI/modules/graphics/macros/plot.sci')  
getf SCI/modules/graphics/macros/plot.sci
```

See Also

functions, function, genlib, getd, exec, edit, comp, add_profiling

Name

`head_comments` — display scilab function header comments

```
head_comments(name)
head_comments(name,%paths)
```

Parameters

`name`

character string, the function name

`%paths`

character string vector, paths where to look for the assoated sci-file

Description

`comments(name)` displays the function header comments (like the Matlab help). The comments are read from the associated sci-file. If `name` is a function in a library the sci-file path is those given by the library path (see `lib`). If `name` is a function which is not in a library, a file with name `name.sci` is searched in the directories given by the variable `%paths`

Warning, most of the scilab predefined functions have no header comments.

Examples

```
head_comments sinc
```

See Also

`help`

Authors

Serge Steer, INRIA

Name

lib — library definition

```
xlib = lib('lib-dir')
```

Parameters

lib-dir
character string

Description

lib-dir is a character string defining a directory that contains compiled Scilab function (**.bin**) files.

In addition to these files **lib-dir** must have a file called **names**, that contains the names of the functions defined in **lib-dir**. On success, all functions in **lib-dir** are available from within Scilab. They are loaded on demand when called for the first time.

Binary files can be created from within Scilab with the command **save**.

Scilab's standard libraries are defined using **lib** on the **SCIDIR/modules/*/macros/*** subdirectories.

A library variable usually is saved for later loading, either on-line or from the user-specific startup file (see startup).

Restrictions

Scilab tacitly assumes that each **xxxx.bin** file defines a variable named **xxxx**.

Examples

```
//define some variables
function z = myplus(x, y), z = x + y,endfunction
function z = yourplus(x, y), x = x - y,endfunction
A=1:10;

//create the *.bin files in libdir
libdir=TMPDIR
save(libdir + '/myplus.bin', myplus);
save(libdir + '/yourplus.bin', yourplus);
save(libdir + '/A.bin', A);

//create the name file
mputl(['myplus';'yourplus';'A'],TMPDIR+'/names');

//build the library containing myplus and yourplus
xlib = lib(libdir+'/')

//erase the variables
clear myplus yourplus A

//Automatic loading and execution
myplus(1,2)
```

A

See Also

library , genlib , save , deff , exec , whereis

Name

librarieslist — get scilab libraries

```
s=librarieslist()
```

Parameters

s
a string matrix

Description

return in S all libraries on scilab stack.

libraries are scanned from the lastest loaded to the first one.

Examples

```
librarieslist()  
// to sort list  
l = gsort(librarieslist(),'r','i')
```

See Also

libraryinfo gsort

Name

library — library datatype description

Description

A library is a data type with type number 14. It contains a path-name and a set of names. It allows automatic loading of variables using the following algorithm:

Suppose the Scilab user references the variable named `foo`. Scilab first looks if `foo` is the name of a primitive or of an already defined variable. If not, it looks for `foo` sequentially (the newest first) in all defined library .

Suppose `foo` belongs to the set of names of the library `xlib` then Scilab tries to load the file `<xlib-path-name>/foo.bin`. `<xlib-path-name>/foo.bin` must have been created using the `save` function

Library are often used for collection of functions, but they can also be used for any collection of scilab variables

If a function is defined in more than one library, the default search algorithm loads thode contained in the newest. It possible to force the use of a specific library using dot notation:

`xlib.foo` loads the variable `foo` contained in `xlib`. if `foo` is a function and `xlib.foo(args)` executes the functions

Examples

```
// elemllib is a predefined library
elementary_functionlib //displays the contents of the library
A=rand(3,3);
cosm(A) //loads cosm and executes it
whos -name cosm // now cosm is a variable
elementary_functionlib.sinm //loads sinm from the library
elementary_functionlib.cosm(A) //reloads cosm and executes it
```

See Also

`lib` , `string` , `load` , `save`

Name

libraryinfo — get macros and path of a scilab library

```
macros = libraryinfo(libraryname)
[macros,path] = libraryinfo(libraryname)
```

Parameters

macros

a string matrix (all main functions of the library)

path

a string (path of library)

libraryname

a string (library name)

Description

get functions names and path of a scilab library. The function names returned correspond to those which correspond to the associated .sci or .bin file names. The other ones are subsidiary functions.

Examples

```
[m,p]=libraryinfo('corelib')
```

See Also

librarieslist

Name

listfunctions — properties of all functions in the workspace

```
[flist,compiled,profilable,called] = listfunctions([scope])
```

Parameters

scope

string, "local" (default) or "global"

flist

string array, names of all the function variables in the specified namespace

compiled

boolean array, true if the corresponding element of flist is of type=13

profilable

boolean array, true if the corresponding element of flist is of type=13, and additionally profiling information is found in the pseudocode of the function

called

uint32 array, number of times the corresponding element of flist has been already called (nonzero only for profilable functions)

Description

- This function checks all the variables in the workspace (given by who) and collects those of type 11 or 13; for the latter, `lst=macr2lst(fun)` is called, in order to check for the magic profiling entry at the end of the first codeline, i.e. `lst(5)(1)=="25"`.

Examples

```
recompilefunction("asinh","p")  
[flist,compiled,profilable,called] = listfunctions();  
flist(profilable)
```

See Also

function , exec , deff , comp , fun2string , profile , recompilefunction

Authors

Enrico Segre

Bibliography

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

Name

macro — Scilab procedure and Scilab object

Description

Macros are Scilab procedures ("macro", "function" and "procedure" have the same meaning). Usually, they are defined in files with an editor and loaded into Scilab by `exec` or through a library.

They can also be defined on-line (see `deff`). A file which contains a macro must begin as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The `yi` are output variables calculated as functions of input variables and variables existing in Scilab when the macro is executed. A macro can be compiled for faster execution. Collections of macros can be collected in libraries. Macros which begin with % sign (e.g. `%foo`) and whose arguments are lists are used to perform specific operations: for example, `z=%rnr(x,y)` is equivalent to `z=x*y` when `x` and `z` are rationals (i.e. `x=list('r',n,d,[...])` with `n` and `d` polynomials).

See Also

`deff` , `exec` , `comp` , `lib`

Name

macrovar — variables of function

```
vars=macrovar(function)
```

Parameters

vars
list list(in,out,nolocal,called,locals)

function
name of a function

Description

Returns in a list the set of variables used by a function. `vars` is a list made of five column vectors of character strings

in input variables (`vars(1)`)

out output variables (`vars(2)`)

no local referenced variables which are not defined inside the function and which are not functions (`vars(3)`)

called names of functions called (`vars(4)`)

locals local variables (`vars(5)`)

Examples

```
deff('y=f(x1,x2)', 'loc=1;y=a*x1+x2-loc')
vars=macrovar(f)
```

See Also

string, macr2lst

Name

plotprofile — extracts and displays execution profiles of a Scilab function

```
plotprofile(fun)
```

Parameters

fun

a Scilab compiled function, or a function name (string), or an array of function names

Description

To use `plotprofile`, the Scilab function must have been prepared for profiling (see `exec`).

When such a function is executed, the system counts how many times each line is executed and how much cpu time is spent executing each line. This data is stored within the function data structure. The function `plotprofile` in an interactive command which displays this results in a graphic window. When a line is clicked, the source of the function is displayed with the selected line highlighted.

NOTE: you have to click on the "Exit" item in the graphics windows to exit from "plotprofile".

The function code is regenerated with `fun2string` and dumped into a temporary file.

Examples

```
//define a function and prepare it for profiling
deff('x=foo(n)', ['if n==0 then'
                  '  x=[]'
                  'else'
                  '  x=0'
                  '  for k=1:n'
                  '    s=svd(rand(n+10,n+10))'
                  '    x=x+s(1)'
                  '  end'
                  'end'], 'p')

//call the function
foo(30)

//get execution profiles
plotprofile(foo) // click on Exit to exit
```

See Also

`profile` , `showprofile` , `fun2string`

Name

profile — extract execution profiles of a Scilab function

```
c=profile(fun)
```

Parameters

fun

a Scilab function

c

a nx3 matrix containig the execution profiles

Description

To use `profile` the Scilab function must have been prepared for profiling (see `exec`).

For such function, When such a function is executed the systems counts how many time each line is executed and how may cpu time is spend for each line execution. These data are stored within the function data structure. The `profile` function allows to extract these data and return them in the two first columns of `c`. The `c` third column gives a measure of interpreter effort for one execution of corresponding line. `I`th line of `C` corresponds to `I`th line of the function (included first)

Note that, due to the precision of the processor clock (typically one micro second), some executed lignes may appear with 0 cpu time even if total cpu time really spend in their execution is large.

Examples

```
//define function and prepare it for profiling
deff('x=foo(n)', ['if n==0 then'
                  '  x=[]'
                  'else'
                  '  x=0'
                  '  for k=1:n'
                  '    s=svd(rand(n+10,n+10))'
                  '    x=x+s(1)'
                  '  end'
                  'end'], 'p')

//call the function
foo(10)

//get execution profiles
profile(foo)

//call the function
foo(20)
profile(foo) //execution profiles are cumulated
```

See Also

`exec` , `deff` , `plotprofile` , `showprofile`

Name

recompilefunction — recompiles a scilab function, changing its type

```
recompilefunction(funname [, kind [, force]])
```

Parameters

funname

string, name of the function to recompile

kind

string: **"n"** (noncompiled, type 11), **"c"** (compiled, type 13) or **"p"** (compiled, type 13, with provision for profiling). Default "c".

force

boolean. If false, the function is recompiled only if its kind changes; if true, it is recompiled even if it keeps the same kind (notably useful to recompile a "p" function, to reset the profiling statistics).

Description

- This function reverse-compiles a function variable via fun2string, and recompiles it to the desired kind with deff.

Examples

```
recompilefunction("asinh","p")
for i=1:100; asinh(rand(100,100)); end
showprofile(asinh)
```

See Also

function , deff , comp , fun2string , profile

Authors

Enrico Segre

Bibliography

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

Nom

`remove_profiling` — Removes profiling instructions to out of a function.

```
remove_profiling(funname)
```

Parameters

`funname`

A character string, the name of the function

Description

`remove_profiling(funname)` Removes profiling instructions (if any) out of the function named `funname`.

Examples

```
function x=foo(a,n)
    x=0;
    for i=1:n
        if x<10 then
            x=x+a
        else
            x=x+1
        end
    end
    x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
remove_profiling("foo")
```

See Also

`profile`, `plotprofile`, `remove_profiling`, `reset_profiling`

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Nom

reset_profiling — Resets profiling counters of a function.

```
reset_profiling(funname)
```

Parameters

funname

A character string, the name of the function

Description

reset_profiling(funname) Resets profiling counters (if any) of the function named **fun-**
name.

Examples

```
function x=foo(a,n)
    x=0;
    for i=1:n
        if x<10 then
            x=x+a
        else
            x=x+1
        end
    end
    x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
reset_profiling("foo")
profile(foo) //extract profile information
```

See Also

profile, plotprofile, add_profiling, reset_profiling, remove_profiling

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Name

showprofile — extracts and displays execution profiles of a Scilab function

```
showprofile(fun)
```

Parameters

fun
a Scilab function

Description

To use `showprofile` the Scilab function must have been prepared for profiling (see `exec`).

For such function, When such a function is executed the systems counts how many time each line is executed and how may cpu time is spend for each line execution. These data are stored within the function data structure. The `showprofile` function outputs profiling results (see `profile`) with text of the function lines.

Function text is rebuild with `fun2string`.

Examples

```
//define function and prepare it for profiling
deff('x=foo(n)', ['if n==0 then'
    ' x=[]'
    ' else'
    ' x=0'
    ' for k=1:n'
    '     s=svd(rand(n+10, n+10))'
    '     x=x+s(1)'
    ' end'
    'end'], 'p')

//call the function
foo(30)

//get execution profiles
showprofile(foo)
```

See Also

`profile` , `plotprofile` , `fun2string`

Name

`varargin` — variable numbers of arguments in an input argument list

Description

A function whose last input argument is `varargin` can be called with more input arguments than indicated in the input argument list. The calling arguments passed from `varargin` keyword onwards may then be retrieved within the function in a list named `varargin`.

Suppose that `varargin` keyword is the n th argument of the formal input argument list, then if the function is called with less than $n-1$ input arguments the `varargin` list is not defined, if the function is called with $n-1$ arguments then `varargin` list is an empty list.

`y = function ex(varargin)` may be called with any number of input arguments. Within function `ex` input arguments may be retrieved in `varargin(i)`, $i=1:\text{length}(\text{varargin})$

If it is not the last input argument of a function, `varargin` is a normal input argument name with no special meaning.

The total number of actual input arguments is given by `argn(2)`.

Remark

Named argument syntax like `foo(..., key=value)` is incompatible with the use of `varargin`. The reason is that the names (i.e. keys) associated with values are not stored in the `varargin` list. Consider for instance:

```
function foo(varargin),disp([varargin(1),varargin(2)]),endfunction
```

```
foo(a=1,b=2)
```

Scilab answers: 1. 2.

```
foo(b=1,a=2)
```

Scilab answers: 1. 2.

Result is the same, but the arguments were inverted.

Examples

```
deff('exempl(a,varargin)',['[lhs,rhs]=argn(0)'  
                           'if rhs>=1 then disp(varargin),end'])  
exempl(1)  
exempl()  
exempl(1,2,3)  
l=list('a',%s,%t);  
exempl(1,l(2:3))
```

See Also

`function`, `varargout`, `list`

Name

varargout — variable numbers of arguments in an output argument list

Description

A function whose output argument list contains `varargout` must be called with more output arguments than indicated in the output argument list. The calling arguments passed from `varargout` keyword onwards are extracted out of the `varargout` list defined in the function.

`varargout= function ex()` may be called with any number of output arguments. Within function `ex` output arguments may be stored in `varargout(i)`.

`[X1,...Xn,varargout]= function ex()` may also be used. In this case the `Xi` variables must be assigned in the function as well as `varargout(i)`.

The actual total number of output argument is given `argn(1)`

Remark

The `varargout` variable must be created within the function and assigned to a list. If `varargout` is the only formal output variable the list must contain at least one entry.

Examples

```
function varargout=exempl()
    varargout=list(1,2,3,4)
endfunction

x=exempl()
[x,y]=exempl()
[x,y,z]=exempl()

function [a,b,varargout]=exempl1()
    a='first'
    b='second'
    varargout=list(1,2,3,4)
endfunction

exempl1()
[a,b]=exempl1()
[a,b,c]=exempl1()
```

See Also

function, varargin, list

Name

whereis — name of library containing a function

```
[libname]=whereis(function-name)
```

Description

returns as a character string the name of the library containing the function `function-name`. The path of the library is returned by typing "`libname`".

See Also

`lib`

Part V. Files : Input/Output functions

Name

basename — strip directory and suffix from filenames

```
files= basename(files[,flag [,flagexpand]])
```

Parameters

files

a string matrix giving a set of file names.

flag,flagexpand

boolean optional parameters. (default value %t).

files

a string matrix.

Description

basename return the basename of the file entries given in files.

If `flag` is true the files are first converted to the target type given by the MSDOS variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or `~` are expanded using environment variables.

Examples

```
files=basename('SCI/modules/fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci.k')
```

See Also

listfiles, pathconvert, fileparts

Name

chdir — changes Scilab current directory
cd — changes Scilab current directory

```
b=chdir(path)
realpath=cd(path)
cd path
```

Parameters

b
a boolean %t if chdir operation is ok.

path
a character string

realpath
a character string, the real path name after pathname conversion (see below)

Description

Change the current Scilab directory to those given by **path**. Note that path conversion is performed and for example `SCI/modules/core/macros` is a valid pattern for both unix and windows. If **path** is empty change to "home" directory.

Examples

```
chdir(TMPDIR);
pwd
cd
cd SCI
```

See Also

pwd

Name

copyfile — Copy file

```
copyfile('source','destination')  
[status,message] = copyfile('source','destination')
```

Description

copyfile('source','destination') copies the file or directory , source (and subdirectories) to the file or directory, destination.

If source is a directory, destination can not be a file. copyfile replaces existing files without warning.

[status, message] = copyfile('source','destination') copies source to destination, returning the status and a message.

Whatever the operating system, if the copy succeeds, the status is 1 and the message is empty ; if the copy fails, the status is 0 and the message is not empty.

The timestamp given to the destination file is identical to that taken from source file.

Examples

```
copyfile(SCI+"/etc/scilab.start",TMPDIR+"/scilab.start")  
[status,message] = copyfile(SCI+"/etc/scilab.start",TMPDIR);
```

See Also

movefile, mdelete

Authors

Allan CORNET

Name

createdir — Make new directory

```
createdir('dirname')
status = createdir('dirname')
```

Description

createdir('dirname') creates the directory dirname in the current directory, if dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = createdir('dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is %T for success and %F otherwise.

createdir is used by mkdir.

Examples

```
createdir(SCIHOME+'/Directory_test')
removedir(SCIHOME+'/Directory_test')
```

See Also

mkdir , rmdir

Authors

A.C

Name

deletefile — delete a file

```
f = deletefile(filename)
```

Parameters

filename

a file name existing or not.

f

%t or %f

Description

delete a file. if file has been deleted, it will return %t else %f.

Examples

```
fd = mopen(TMPDIR+'/filetodelete.txt','wt');  
mclose(fd);  
if (fileinfo(TMPDIR+'/filetodelete.txt') <> []) then deletefile(TMPDIR+'/fileto  
deletefile(TMPDIR+'/notexistingfile')
```

Authors

A.C

Name

dir — get file list

```
dir path
S=dir([path])
```

Parameters

path

a string matrix giving a directory pathname (eventually ended by a pattern built with *). Default value is .

S

a tlist of type dir with fields : name, date and isdir

Description

dir can be used to get the files which match the patterns given by the path argument. Patterns are given to the unix ls or to the windows dir commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example SCI/modules/core/macros/*.sci is a valid pattern for both unix and windows.

The name field of the returned variable is the column vector of the file names.

The date field of the returned variable is the column vector of integers containing a last modification date coded in second from 1 Jan 1970).

The isdir field of the returned variable is the column vector of boolean true if the corresponding name is a directory.

The default display of the returned structure is a column formatted list of files. It can be changed redefining the function %dir_p

Examples

```
dir
dir SCI/modules/core/macros/*.bin
x=dir('SCI/modules/core/macros/*.bin')
dt=getdate(x.date);
mprintf("%s: %04d-%02d-%02d %02d:%02d:%02d\n",x.name,dt(:,[1 2 6 7:9]))
```

See Also

listfiles , findfiles , ls , fileinfo , date

Name

`dirname` — get directory from filenames

```
dirs= dirname(files[,flag [,flagexpand]])
```

Parameters

`files`

a string matrix giving a set of file names.

`flag,flagexpand`

boolean optional parameters. (default value %t).

`files,dir`

string matrices.

Description

`dirname` return the `dirname` of the file entries given in `files`.

If `flag` is true the files are first converted to the target type given by the `MSDOS` variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or `~` are expanded using environment variables.

Note that `dirname(files,%f)` can give erroneous results if pathnames given in `files` do not follow the convention given by the `MSDOS` variable.

Examples

```
files=dirname('SCI/modules/fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci.k')
```

See Also

`basename` , `listfiles` , `pathconvert`

Name

dispfiles — display opened files properties

```
dispfiles([units])
```

Parameters

units

a vector of numbers, the file's logical units. By default all opened files.

Description

dispfiles displays properties of currently opened files.

Examples

```
dispfiles()
```

See Also

file , mopen

Authors

S. Steer

Name

fileext — returns extension for a file path

```
extension = fileext(fullpath)
```

Parameters

fullpath

a character string, the given file path

extension

a character string, the extension part is any or "

Description

extension=fileext(fullpath) splits the fullpath character string in the extension part including the dot.

Examples

```
extension = fileext('SCI/etc/scilab.start')
extension = fileext(['SCI/etc/scilab.start'; 'SCI/etc/scilab.quit'])
```

See Also

fileparts

Authors

Allan CORNET

Name

fileinfo — Provides information about a file

```
[x,ierr] = fileinfo(files)
```

Parameters

files

a character string or a string column vector, file pathname

x

an integer vector of size 13 containing information or an empty matrix if file does not exist.

if 'files' is a string column vector, x will be a matrix of size m x 13.

If a filename does not exist, it will return as output information a line of size 13 with Nan in each element of this line.

ierr

error indicator, 0, if no error has occurred

Description

x = fileinfo(file) returns

x(1)

The file size

x(2)

The file mode (decimal value).

x(3)

The user id

x(4)

The group id

x(5)

The device number

x(6)

The date of last modification

x(7)

The date of last change

x(8)

The date of last access

x(9)

The device type (if inode device)

x(10)

The blocksize for filesystem I/O (always 0 on Windows)

x(11)

The number of blocks allocated (always 0 on Windows)

x(12)

The inode

x(13)

The number of hard links

Reference

This function is an interface to the C function `stat`.

Permissions are typically specified as octal numbers : `dec2oct(x(2))` to convert

Numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Any omitted digits are assumed to be leading zeros. The first digit selects the set user ID (4) and set group ID (2) and sticky (1) attributes. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1); the third selects permissions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

Examples

```
w = fileinfo(SCI+'/etc/scilab.start')
// file permission
dec2oct(w(2))
// file date
getdate(w(6))

// Checks write permission on a file
w = fileinfo(SCI+'/etc/scilab.start')

S_IWRITE = 128; // mask write permission
S_IEXEC = 64; // mask exec permission
S_IREAD = 256; // mask read permission
S_IFCHR = 8192; // mask directory permission

if ( bitand( w(2), S_IWRITE ) <> 0) then
    disp('WRITE PERMISSION on this file.');
```

```
else
    disp('NO WRITE PERMISSION on this file.');
```

```
end

FILES = [SCI;SCIHOME;'not_exist_file';TMPDIR]
[X,ERRS] = fileinfo(FILES)
```

See Also

`getdate`, `file`, `dispfiles`, `newest`, `isdir`

Authors

S. Steer INRIA

A.C

Name

fileparts — returns the path, filename and extension for a file path

```
[path, fname, extension]=fileparts(fullpath)
v=fileparts(fullpath, sel)
```

Parameters

fullpath

a character string, the given file path

sel

a optional character string selector, with posible values: 'path' 'fname' or 'extension'

path

a character string, the path of the directory pointed to by fullpath

fname

a character string, the filename part is any or ''

extension

a character string, the extension part is any or ''

value

a character string, depending on sel value

Description

`[path, fname, extension]=fileparts(fullpath)` splits the fullpath character string in its three parts: the path of the directory pointed to, the filename part, the extension part including the dot.

Examples

```
[path, fname, extension]=fileparts('SCI/etc/scilab.start')
fileparts('SCI/etc/scilab.start', 'extension')
```

See Also

`pathconvert` , `basename` , `fullfile`

Authors

Serge Steer, INRIA

Name

filesep — returns directory separator for current platform

```
s = filesep()
```

Parameters

s
a string

Description

returns directory separator. ('/' on Linux or '\' on Windows)

Examples

```
filesep()
```

Authors

A.C

Name

findfiles — Finding all files with a given filespec

```
f = findfiles()  
f=findfiles(path)  
f=findfiles(path,filespec)
```

Parameters

path

a path

filespec

a spec file. example "*.sce"

f

returns a string matrix of filenames

Description

Finding all files with a given filespec

Examples

```
f=findfiles()  
f=findfiles(SCI)  
f=findfiles(SCI+'/modules/core/macros','*.sci')
```

See Also

listfiles

Authors

A.C

Name

fprintf — Emulator of C language fprintf function

```
fprintf(file,format,value_1,...,value_n)
```

Parameters

`format`

a Scilab string. Specifies a character string combining literal characters with conversion specifications.

`value_i`

Specifies the data to be converted according to the `format` parameter.

`str`

column vector of character strings

`file`

a Scilab string specifying a file name or a logical unit number (see `file`)

Note that if `file=0`, the message will be displayed on standard error stream (`stderr`).

Description

Obsolete function, use preferably the `mfprintf` function which is much more compatible with the C `fprintf` functionalities.

The `fprintf` function converts, formats, and writes its `value` parameters, under control of the `format` parameter, to the file specified by its `file` parameter.

The `format` parameter is a character string that contains two types of objects:

Literal characters

which are copied to the output stream.

Conversion specifications

each of which causes zero or more items to be fetched from the `value` parameter list. see `printf_conversion` for details

If any `values` remain after the entire `format` has been processed, they are ignored.

Examples

```
u=file('open','results','unknown') //open the result file
t=0:0.1:2*pi;
for tk=t
    fprintf(u,'time = %6.3f value = %6.3f',tk,sin(tk)) // write a line
end
file('close',u) //close the result file

fprintf(0,'My error which is going to be displayed on the stderr')
```

See Also

`mfprintf` , `string` , `print` , `write` , `format` , `disp` , `file` , `printf` , `sprintf` , `printf_conversion`

Name

fprintfMat — Write a matrix in a file.

```
fprintfMat(file,M [,format,text])
```

Parameters

fil

a string, the pathname of the file to be written.

M

A matrix of real numbers.

format

a character string, a C like format. This is an optional parameter, the default value is "%f"

text

a string matrix giving non numerical comments stored at the beginning of the file.

Description

The `fprintfMat` function writes a matrix in a formatted file. Each row of the matrix give a line in the file. If `text` is given then the elements of `text` are inserted columnwise at the beginning of the file one element per line.

Examples

```
n=50;  
a=rand(n,n,'u');  
fprintfMat(TMPDIR+'/Mat',a,'%5.2f');  
a1=fscanfMat(TMPDIR+'/Mat');
```

See Also

`fclose` , `feof` , `fprintf` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

`fscanf` — Converts formatted input read on a file

```
[v_1, ...v_n]=fscanf (file,format)
```

Parameters

`format`

Specifies the format conversion.

`file`

Specifies the input file name or file number.

Description

The `fscanf` functions read character data on the file specified by the `file` argument , interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see `scanf_conversion` for details.

See Also

`printf` , `read` , `scanf` , `sscanf` , `mfscanf` , `scanf_conversion`

Name

fscanfMat — Reads a Matrix from a text file.

```
M=fscanfMat(filename);  
[M, text]=fscanfMat(filename);
```

Parameters

filename

a character string giving the name of the file to be scanned.

M

Output variable. A matrix of real numbers.

text

Output variable. A string matrix.

Description

The `fscanfMat` function is used to read a scalar matrix from a text file. The first non-numeric lines of the file are returned in `text` if requested and all the remaining lines must have the same number of columns (column separator are assumed to be white spaces or tab characters). The number of columns of the matrix will follow the number of columns found in the file and the number of lines is fetched by detecting eof in the input file. This function can be used to read back numerical data saved with the `fprintfMat`.

Examples

```
fd=fopen(TMPDIR+'/Mat','w');  
fprintf(fd,'Some text.....\n');  
fprintf(fd,'Some text again\n');  
a=rand(6,6);  
for i=1:6 ,  
    for j=1:6, fprintf(fd,'%5.2f ',a(i,j));end;  
    fprintf(fd,'\n');  
end  
fclose(fd);  
a1=fscanfMat(TMPDIR+'/Mat')
```

See Also

`fclose` , `feof` , `fprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

fullfile — Build a full filename from parts

```
f = fullfile(varargin)
```

Parameters

varargin

all directories and filename used to build the full filename (at least one directory and filename)

f

full filename

Description

`f = fullfile(varargin)` builds a full filename taking care of platform on which it is run and handling the cases when the directories begin or end with a directory separator.

Examples

```
f=fullfile("/home/", "\scilab", "macros", "\util", "fullfile.sci")  
f=fullfile("C:", "\scilab", "macros", "\util", "fullfile.sci")
```

See Also

`pathconvert` , `fileparts`

Authors

V.C.

Name

fullpath — Creates an full path name for the specified relative path name.

```
res = fullpath(relative_path)
```

Parameters

res
a string

relative_path
a string

Description

Creates an full path name for the specified relative path name.

On linux 'relative_path' needs to exist.

Examples

```
mkdir(TMPDIR+'/niv1');  
mkdir(TMPDIR+'/niv1/niv2');  
mputl(' ',TMPDIR+'/niv1/test.txt');  
cd(TMPDIR+'/niv1/niv2');  
fullpath('../test.txt')
```

Authors

A.C

Name

`get_absolute_file_path` — Given an absolute pathname of a file opened in scilab.

```
pathname = get_absolute_file_path(filename)
```

Parameters

filename

A character string : filename

pathname

A character string : the absolute pathname

Description

Given the absolute pathname of a file already opened in scilab.

`get_absolute_file_path` searches, in scilab's internal list of files currently opened, filename and returns its path.

"get_absolute_file_path" seek, in the internal list of the files of scilab currently opened, " filename" and it gives his path.

if file not opened , it will return a error.

WARNING : in previous version (scilab 5.0.x) current directory was returned if file was not found.

This function can be used to find from where (path) is executed a scilab script.

Examples

```
// exec this script

a=mopen(TMPDIR+'test.sce','wt');
disp(get_absolute_file_path('test.sce'));
mclose(a);
```

See Also

`getshortpathname`, `getlongpathname`, `pwd`

Authors

Allan CORNET

Name

getdrives — Get the drive letters of all mounted filesystems on the computer.

```
drives = getdrives()
```

Parameters

drives
a matrix of strings

Description

Get the drive letters of all mounted filesystems on the computer.

returns the roots of all mounted filesystems on the computer as a matrix of strings.

For Linux this list consists solely of the root directory, / .

Examples

```
getdrives()
```

Authors

A.C

Name

getlongpathname — get long path name (Only for Windows)

```
longpath=getlongpathname(shortpath)
[longpath,bOK]=getlongpathname(shortpath)
```

Parameters

shortpath

A character string or matrix of strings the short path

longpath

A character string or matrix of strings the long path

bOK

A boolean %T or a matrix of boolean if path has been converted else %F

Description

The getlongpathname primitive converts the specified path to its long form. If no long path is found, this primitive returns the specified name.

Examples

```
[longpath,bOK]=getlongpathname(SCI)
[longpaths,bOKs]=getlongpathname([SCI;SCI])
```

See Also

getshortpathname

Authors

Allan CORNET

Name

`getrelativefilename` — Given an absolute directory and an absolute filename, returns a relative file name.

```
rel_file = getrelativefilename(abs_dir,abs_file)
```

Parameters

abs dir

A character string : the absolute directory

abs file

A character string : the absolute filename

rel file

A character string : relative filename

Description

Given an absolute directory and an absolute filename, returns a relative file name.

For example, if the current directory is C:\scilab\bin and the filename C:\scilab\modules\help-tools\readme.txt is given, `getrelativefilename` will return `..\modules\helptools\readme.txt`.

Examples

```

if MSDOS then
    getrelativefilename('C:\program file\scilab-4.0\bin', 'C:\program file\scilab-4.0\bin')
    getrelativefilename('C:\program file\scilab-4.0\bin\', 'C:\program file\scilab-4.0\bin\')
    getrelativefilename(SCI+'\bin', SCI+'\modules\helptools\help.dtd')
    getrelativefilename(WSCI+'\bin', WSCI+'\modules\helptools\help.dtd')
    getrelativefilename(pwd(), WSCI+'\bin\scilex')
else
    getrelativefilename('/usr/local/scilab-4.0/bin', '/usr/local/scilab-4.0/module')
    getrelativefilename('/usr/local/scilab-4.0/bin/', '/usr/local/scilab-4.0/module')
    getrelativefilename(SCI+'/bin', SCI+'/modules/helptools/help.dtd')
    getrelativefilename(pwd(), SCI+'/bin/scilex')
end

```

See Also

getshortpathname , getlongpathname , pwd

Authors

Pierre MARECHAL

Name

getshortpathname — get short path name (Only for Windows)

```
shortpath=getshortpathname(longpath)
[shortpath,bOK]=getshortpathname(longpath)
```

Parameters

longpath

A character string or matrix of strings the long path

shortpath

A character string or a matrix of strings the short path

bOK

A boolean %T or a matrix of boolean if path has been converted else %F

Description

The getshortpathname primitive converts the specified path to its short form.

Examples

```
[shortpath,bOK]=getshortpathname(SCI)
[shortpaths,bOKs]=getshortpathname([SCI;SCI])
```

See Also

getlongpathname

Authors

Allan CORNET

Name

`%io` — variable returns standard input and standard output (file descriptor).

```
%io(1)
%io(2)
```

Description

`%io(1)` returns standard input (file descriptor 5).

`%io(2)` returns standard output (file descriptor 6).

file descriptor 0 is the standard error.

Examples

```
mfprintf(%io(2), 'Scilab stdout (C)');
mfprintf(0, 'Scilab stderr (C)');
write(%io(2), 'Scilab stdout (Fortran)');
write(0, 'Scilab stderr (Fortran)');
[units, typ, names]=file()
```

See Also

`write`, `mfprintf`, `file`

Name

isdir — checks if argument is a directory path

```
r=isdir(path)
```

Parameters

path

a character string or a matrix of strings, the file pathname

r

a boolean, true if `path` is a path to a directory.

Description

`r=isdir(path)` checks if `path` is a path to a directory.

Reference

This function is based on the C function `stat`. The `SCI` and `~` shortcuts for Scilab directory and home directory are handled.

Examples

```
isdir(TMPDIR)
isdir SCI/etc/scilab.start
```

See Also

`fileinfo`

Authors

S. Steer INRIA

Name

isfile — checks if argument is a file

```
b = isfile(filenamees)
```

Parameters

files

a character string or a string matrix.

x

an matrix of boolean (%t is filename exists).

Description

x = isfile(filename) checks if filename is existing file or not. (a directory is not a file. see isdir.)

Examples

```
cd SCI
filenames = ls()
isfile(filenamees)

isfile(filenamees)

isfile(SCI + '/etc')
isdir(SCI + '/etc')
```

See Also

fileinfo, isdir

Author

A.C

Name

listfiles — list files

```
files= listfiles(paths [,flag,flagexpand])
```

Parameters

paths

a string matrix giving a set of pathnames (eventually ended by a pattern built with *)

flag,flagexpand

boolean optional parameters. (default value %t).

files

a string matrix.

Description

`listfiles` can be used to list the files which match the patterns given by one of the paths entries. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/core/macros/*.sci` is a valid pattern for both unix and windows.

if `flag` is true the pathnames given by `paths` are converted according to the MSDOS value (See `pathconvert`). Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or `~` are expanded using environment variables.

Examples

```
files=listfiles(['SCI/modules/core/macros/*.sci';'SCI/modules/core/macros/*.bin
```

See Also

`findfiles` , `basename` , `pathconvert`

Name

listvarinfile — list the contents of a saved data file

```
listvarinfile(filename)
[nams,typs,dims,vols]=listvarinfile(filename)
```

Parameters

filename

character string, the pathname of the file to be inspected

nams

character array, names of the variables saved in the file

dims

list, dimensions of the variables saved in the file

typs

numeric array, types of the variables saved in the file

vols

numeric array, size in bytes of the variables saved in the file

Description

- This utility function lists "a la whos" the variables contained in a Scilab data file produced by save.

Remark: hypermatrices are reported as plain mlists; rationals and state-spaces are reported as plain tlists; graphic handles are not recognized.

Examples

```
a=eye(2,2); b=int16(ones(a)); c=rand(2,3,3);
save("vals.dat",a,b,c)
listvarinfile("vals.dat")
```

See Also

whos , save , load , save_format , type

Authors

Serge Steer

31 Jan 2001; reediting by Enrico Segre

Name

ls — show files

```
ls path options  
files=ls( [path] )
```

Parameters

path

a string matrix giving a directory pathname (eventually ended by a pattern built with *). Default value is .

files

a string column vector. By default it contains a column formatted output. if one of the option is '-1', `files` contains an entry for each files

Description

ls can be used to list the files which match the patterns given by the path argument. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/modules/core/macros/*.sci` is a valid pattern for both unix and windows.

If you want to get a vector of all files matching a pattern use preferably the `listfiles` or the `dirfunction`.

Please note that starting from the version 5.0 of Scilab, the second input argument has been removed (a sequence of strings which can be added under Unix systems: the Unix `ls` command options). This option has been removed mainly for security and portability reasons.

Examples

```
ls  
ls SCI/modules/core/macros/*.sci  
x=ls('SCI/modules/core/macros/*.sci')
```

See Also

`listfiles` , `findfiles` , `dir` , `fileinfo`

Name

`maxfiles` — sets the limit for the number of files a scilab is allowed to have open simultaneously.

```
r= maxfiles(newnumbermax)
```

Parameters

`newnumbermax`
a integer the new value

`r`
effective new value.

Description

sets the limit for the number of files a scilab is allowed to have open simultaneously.

Minimum : 20

Maximum : 100

Default : 20

Examples

```
r=maxfiles(50);
```

See Also

`mopen`

Name

`mclearerr` — reset binary file access errors

```
mclearerr([fd])
```

Parameters

`fd`

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

Description

The function `clearerr` is used to resets the error indicator and EOF indicator to zero.

See Also

`merror` , `mclose` , `mopen` , `mput` , `mget` , `mgetstr` , `mputstr` , `meof` , `mseek` , `mtell`

Name

`mclose` — close an opened file

```
err=mclose([fd])  
mclose('all')
```

Parameters

`fd`

scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

`err`

a scalar. Error indicator : vector

Description

`mclose` must be used to close a file opened by `mopen`. If `fd` is omitted `mclose` closes the last opened file.

`mclose('all')` closes all files opened by `file('open',...)` or `mopen`. Be careful with this use of `mclose` because when it is used inside a Scilab script file, it also closes the script and Scilab will not execute commands written after `mclose('all')`.

See Also

`mEOF` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetl` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputl` , `mputstr` , `mscanf` , `mseek` , `mtell` , `file` , `mdelete`

Name

mdelete — Delete file(s)

```
mdelete(filename)
```

Parameters

`filename`

a character string. The pathname of the file(s) to delete.

Description

`mdelete` may be used to delete a file or a set of files if `filename` contains meta-characters.

Note that `mdelete` does not ask for confirmation when you enter the delete command. To avoid accidentally losing files, make sure that you have accurately specified the items you want deleted.

See Also

`mopen` , `mclose` , `meof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell`

Name

`feof` — check if end of file has been reached

```
err=feof(fd)
```

Parameters

`fd`

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

`err`

scalar. Error indicator

Description

The function `feof` will return a non null value if end of file has been reached in a previous call to `mget` or `mgetstr`. The function `clearerr` is used to reset the error flag and EOF flag to zero.

See Also

`mclose` , `feof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

`merror` — tests the file access errors indicator

```
err = merror([fd])  
[err,msg] = merror([fd])
```

Parameters

`fd`

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

`err`

scalar. returns the error status number `errno` of the most recent file I/O operation associated with the specified file.

If the most recent I/O operation performed on the specified file was successful, the value of `msg` is empty and `merror` returns an `err` value of 0.

`msg`

string. returns the error string message.

Description

The function `merror` is used to tests the file access errors indicator. returning non-zero if it is set. The error indicator can only be reset by the `mclearerr` function.

A nonzero `err` indicates that an error occurred in the most recent file I/O operation. The value of `message` is a string that can contain information about the nature of the error. If the message is not helpful, consult the C run-time library manual for your host operating system for further details.

Examples

```
fd = mopen(TMPDIR + '/filetxt.txt', 'wt');  
[err,msg] = merror(fd)  
if (err <> 0) then  
    printf('Problem\n');  
end  
mclose(fd);
```

See Also

`mclearerr`, `mclose`, `mopen`, `mput`, `mget`, `mgetstr`, `mputstr`, `meof`, `mseek`, `mtell`

Name

`mfprintf` — converts, formats, and writes data to a file

```
mfprintf(fd, format, a1, ..., an);
```

Parameters

`fd`

scalar, file descriptor given by `mopen` (it's a positive integer).

if `fd` equals 0 redirection in `stderr`.

if `fd` equals 6 redirection in `stdout`.

OBSOLETE :The value `-1` refers to the default file (i.e the last opened file).

`format`

a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C `printf` format operand syntax.

`str`

a character string, string to be scanned.

`a1,...,an`

Specifies the data to be converted and printed according to the format parameter.

Description

The `mfprintf` function is a interface for C-coded version of `fprintf` functions.

The `mfprintf` function writes formatted operands to the file specified by the file descriptor `fd`. The argument operands are formatted under control of the format operand.

this function may be used to output column vectors of numbers and string vectors without an explicit loop on the elements. In that case this function iterates on the rows. The shortest vector gives the number of time the format has to be iterated.

An homogeneous sequence of identical type parameters can be replaced by a matrix

Examples

```
fd = mopen(TMPDIR+'/text.txt', 'wt');
mfprintf(fd, 'hello %s %d.\n', 'world', 1);
mfprintf(fd, 'hello %s %d.\n', 'scilab', 2);
mclose(fd);
if (isdef('editor') | (funptr('editor')<>0)) then
    editor(TMPDIR+'/text.txt')
end
mfprintf(0, 'stderr output.\n');
mfprintf(6, 'stdout output.\n');
```

See Also

`mclose`, `meof`, `fprintfMat`, `mfscanf`, `fscanfMat`, `mget`, `mgetstr`, `mopen`, `mprintf`, `mput`, `mputstr`, `mscanf`, `mseek`, `mtell`, `mdelete`, `printf_conversion`

Name

`mscanf` — reads input from the standard input (interface to the C `scanf` function)
`mfscanf` — reads input from the stream pointer stream (interface to the C `fscanf` function)
`msscanf` — reads its input from the character string (interface to the C `sscanf` function)

```
[n,v_1,...v_n]=mfscanf([niter,]fd,format)
L=mfscanf([niter,] fd,format)

[n,v_1,...v_n]=mscanf([niter,] format)
L=mscanf([niter,]format)

[n,v_1,...v_m]=msscanf([niter,]str,format)
L=msscanf([niter,] str,format)
```

Parameters

`format`

a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C `printf` format operand syntax as described in `scanf_conversion`.

`fd`

The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer). The value -1 refers to the last opened file.

`str`

a Scilab string or string vector.

`niter`

an integer, the number of times the format as to be used.

`n`

an integer, the number of data read or -1 if EOL has been encountered before any datum has been read.

`v_i`

Each function reads characters, interprets them according to a format, and stores the results in its output arguments. If more than `$n$` output arguments are provided, the last ones `v_n+1, . . . v_m` are set to empty matrices.

`L`

if all data are homogeneous they are stored in a unique vector which is returned, otherwise subsequences of same data type are stored in matrices and an `mlist` (with type `cblock`) containing all the built matrices is returned.

Description

The `mfscanf` function reads characters from the stream `fd`.

The `mscanf` function reads characters from Scilab window.

The `msscanf` function reads characters from the Scilab string `str`.

The `niter` optional argument specifies how many time the format has to used. One iteration produces one line in the output matrix. If `niter==-1` the function iterates up to the end of file. The `niter` default value is 1.

comments about precision :

mfscanf is based on C function fscanf. If you use '%f', '%g', '%e' as format your datas will be cast to float and returned in a scilab variable.

This scilab variable is a double then you can have some precision errors. In this case, it is better to use '%lg' format.

Examples

```
//-----
//--      Simple use                                --
//-----
s='1 1.3' //a string
[n,a,b]=msscanf(s,"%i %e")
L=msscanf(s,"%i %e")

//-----
//--      Formats samples                            --
//-----

msscanf(" 12\n", '%c%c%c%c') //scan characters

msscanf('0xabc', '%x') //scan with hexadecimal format

msscanf('012345abczoo', '%[0-9abc]%s') //[] notation

// reading float and double
msscanf('4345.988', '%g')-4345.988 // scan as float
msscanf('4345.988', '%lg')-4345.988 // scan as double

//-----
//--      scanning multi-line data files            --
//-----
//create a file with data
u=mopen(TMPDIR+'/foo', 'w');
t=(0:0.1:%pi)';mfprintf(u, "%6.3f %6.3f\n", t, sin(t))
mclose(u);

u=mopen(TMPDIR+'/foo', 'r'); // open the file for reading
//read the file line by line
[n,a,b]=mfscanf(u, '%e %e') //first line using mutiple LHS syntax
l=mfscanf(u, '%e %e')      //second one using single LHS syntax
//use niter to read 5 more lines
l=mfscanf(5,u, '%e %e')

//use niter=-1 to read up to the end of file
l=mfscanf(-1,u, '%e %e')

mclose(u); //close the file

//-----
//--      scanning multi-line strings vectors        --
//-----
//use niter to scan a string vector
[n,Names,Ages]=msscanf(-1, ["Alain 19";"Pierre 15";"Tom 12"], '%s %d')
D=msscanf(-1, ["Alain 19";"Pierre 15";"Tom 12"], '%s %d')
typeof(D)
Names=D(:,1) //strings
```

```
Age=D(:,2) //numerical values
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete, scanf_conversion

Name

`mget` — reads byte or word in a given binary format and convert to double
`mgeti` — reads byte or word in a given binary format return an int type

```
x=mget([n,type,fd])  
x=mgeti([n,type,fd])
```

Parameters

- `n`
a positive scalar: The number of items to be read.
- `fd`
a scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.
- `type`
a string. Give the binary format used to write all the entries of `x`.
- `x`
a vector of floating point or integer type numbers

Description

The `mget` function reads data in the input specified by the stream parameter `fd` and returns a vector of floating point data. The `mgeti` function reads data in the input specified by the stream parameter `fd` and returns a vector of integer data.

Data is read at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

Note , On Windows, default behavior is to skip byte 13 (0x0D). `mopen` should be called with the ``b`` option, e.g. `fd1=mopen(file1,'rb')` , so that all bytes will be read without exception.

Data type:

- `d`
double
- `f`
float
- `l`
long
- `i`
int
- `s`
short
- `c`
character

Optional flag:

- u..
 unsigned (in combination with one of the above types)
- ..l
 little endian (in combination with one of the above types)
- ..b
 big endian (in combination with one of the above types)

Bytes read are automatically swapped if necessary (by checking little=endian status).

This default swapping behavior can be suppressed by adding a flag in the mopen function.

Formats "l", "d", and "f" are only valid with the mget function.

Examples

```
file1 = 'test1.bin';
file2 = 'test2.bin';
fd1=mopen(file1, 'wb');
fd2=mopen(file2, 'wb');
mput(1996, 'ull', fd1);
mput(1996, 'ull', fd2);
mclose(fd1);
mclose(fd2);

fd1=mopen(file1, 'rb');
if 1996<>mget(1, 'ull', fd1) ;write(%io(2), 'Bug');end;
fd2=mopen(file2, 'rb');
if 1996<>mget(1, 'ull', fd2) ;write(%io(2), 'Bug');end;
mclose(fd1);
mclose(fd2);
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

Name

`mgetl` — read lines from an ascii file

```
txt=mgetl(file_desc [,m])
```

Parameters

`file_desc`

a character string giving the file name or a logical unit returned by `mopen`

`m`

an integer scalar. number of lines to read. Default value is -1.

`txt`

a column vector of string

Description

`mgetl` function allows to read a lines from an ascii file.

If `m` is omitted or is -1 all lines till end of file occurs are read.

If `m` is given `mgetl` tries to read exactly `m` lines. This option is useful to sequentially read part of a file. In this case if an end of file (EOF) occurs before `m` lines are read the read lines are returned (it is possible to check if EOF had occured using the `meof` function) issued.

`mgetl` allows to read files coming from Unix, Windows, or Mac operating systems.

Examples

```
mgetl('SCI/etc/scilab.start',5)

mgetl SCI/modules/elementary_functions/macros/erf.sci

fd=mopen('SCI/etc/scilab.start','r')
mgetl(fd,10)
mclose(fd)
```

See Also

`mputl` , `mclose` , `mfscanf` , `mget` , `mput` , `mgetstr` , `mopen` , `read`

Authors

S. Steer

Name

mgetstr — read a character string

```
str=mgetstr(n [,fd] )
```

Parameters

n

a positive scalar: The number of character to read.

fd

scalar. The fd parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str

a character string

Description

mgetstr function allows to read a character string in a binary file. If EOF is reached before read completion only the properly read values will be returned.

Examples

```
// open file descriptor as text with read mode
fd_r = mopen(SCI+'/ACKNOWLEDGEMENTS','rt')

// get 100 characters of fd_r
strs_1 = mgetstr(100, fd_r)

// get 200 next characters of fd_r
strs_2 = mgetstr(200, fd_r)

// close file descriptor
mclose(fd_r);
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete

Name

mkdir — Make new directory

```
mkdir('dirname')
mkdir('parentdir','newdir')
status=mkdir( ... )
[status,msg]=mkdir( ... )
```

Description

mkdir('dirname') creates the directory dirname in the current directory, if dirname represents a relative path. Otherwise, dirname represents an absolute path and mkdir attempts to create the absolute directory dirname

mkdir('parentdir','dirname') creates the directory dirname in the existing directory parentdir, where parentdir is an absolute or relative pathname.

[status,message] = mkdir(...,'dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is 1 for success, 2 if it already exists, -2 if it is a filename and 0 otherwise.

Examples

```
// Absolute pathname
mkdir(TMPDIR+"/mkdir_example_1")
status_2 = mkdir(TMPDIR+"/mkdir_example_2")
[status_3,msg_3] = mkdir(TMPDIR+"/mkdir_example_3")

// Absolute pathname (parentdir + dirname)
[status_4,msg_4] = mkdir(TMPDIR,"mkdir_example_4")

// Relative pathname
cd TMPDIR;
[status_5,msg_5] = mkdir("mkdir_example_5")
[status_6,msg_6] = mkdir("mkdir_example_5/mkdir_example_6")
```

See Also

cd, dir, rmdir

Authors

A.C

Name

mopen — open a file

```
[fd,err]=mopen(file [, mode, swap ])
```

Parameters

file

a character string. The pathname of the file to open.

mode

a character string that controls whether the file is opened for reading (r), writing (w), or appending (a) and whether the file is opened for updating (+). The mode can also include a b parameter to indicate a binary file.

swap

a scalar. If `swap` is present and `swap=0` then automatic bytes swap is disabled.

err

a scalar. Error indicator. see `merror`.

fd

scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

Description

`mopen` may be used to open a file in a way compatible with the C `fopen` procedure. Without `swap` argument the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor.

The mode parameter controls the access allowed to the stream. The parameter can have one of the following values. In this list of values, the b character indicates a binary file

r

Opens the file for reading.

rb

Opens a binary file for reading.

rt

Opens a text file for reading.

w

Creates a new file for writing, or opens and truncates a file to zero length.

wb

Creates a new binary file for writing, or opens and truncates a file to zero length.

wt

Creates a text binary file for writing, or opens and truncates a file to zero length.

a or ab

Appends (opens a file for writing at the end of the file, or creates a file for writing).

r+ or r+b

Opens a file for update (reading and writing).

w+ or w+b

Truncates to zero length or creates a file for update.

a+ or a+b

Appends (opens a file for update, writing at the end of the file, or creates a file for writing).

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without a file-positioning operation (`mseek()` function). Also, an input operation cannot be directly followed by an output operation without an intervening file positioning operation, unless the input operation encounters the end of the file.

When you open a file for append (that is, when the mode parameter is `a` or `a+`), it is impossible to overwrite information already in the file. You can use the `fseek()` function to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is ignored. All output is written at the end of the file and the file pointer is repositioned to the end of the output.

To open files in a way compatible with Fortran like functions use function `file`.

```
// open a SCI+'/ACKNOWLEDGEMENTS' as text and read only
fd_r = mopen(SCI+'/ACKNOWLEDGEMENTS','rt')
```

```
// read five lines of fd_r
mgetl(fd_r, 5)
```

```
// another way to read file
// here read five words
mfscanf(5,fd_r,'%s')
```

```
// close file descriptor associated to SCI+'/ACKNOWLEDGEMENTS' as text and read
mclose(fd_r);
```

```
// open a file as text with write property
fd_w = mopen(TMPDIR+'/write.txt','wt');
```

```
// write a line in fd_w
mputl('This is a line of text', fd_w);
mclose(fd_w);
```

```
// read text
fd_r2 = mopen(TMPDIR+'/write.txt','rt');
mgetl(fd_r2)
mclose(fd_r2);
```

```
// read write a file as binary
```

```
// first we write file
fd_wb = mopen(TMPDIR+'/writeread.bin','wb')
```

```
// put values as binary
mput(2003, 'l', fd_wb);
mput(2008, 'i', fd_wb);
mput(2012, 's', fd_wb);
mput(98, 'c', fd_wb);
```

```
// close file descriptor associated to TMPDIR+'/writeread.bin'
mclose(fd_wb);
```

```
// we read file
fd_rb = mopen(TMPDIR+'/writeread.bin','rb')

mget(fd_rb, 'l')
mget(fd_rb, 'i')
mget(fd_rb, 's')
mget(fd_rb, 'c')

mclose(fd_rb)
```

See Also

mclose, merror, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

Name

movefile — Move file or directory

```
movefile('source','destination')  
[status,message] = movefile('source','destination')
```

Description

movefile('source','destination') moves the file or directory , source (and subdirectories) to the file or directory, destination.

If source is a directory, destination can not be a file. movefile replaces existing files without warning.

[status, message] = movefile('source','destination') moves source to destination, returning the status and a message.

Whatever the operating system, if the move succeeds, the status is 1 and the message is empty ; if the move fails, the status is 0 and the message is not empty.

Examples

```
copyfile(SCI+"/etc/scilab.start",TMPDIR+"/scilab.start")  
[status,message] = movefile(TMPDIR+"/scilab.start",TMPDIR+"/renamed_scilab.star
```

See Also

copyfile

Authors

Allan CORNET

Name

`mput` — writes byte or word in a given binary format

```
mput(x [, type, fd])
```

Parameters

`x`

a vector of floating point or integer type numbers

`fd`

scalar. The `fd` parameter returned by the function. Default value is -1 which stands for the last (mopen) opened file.

`type`

a string. Give the binary format used to write all the entries of `x`.

Description

The `mput` function writes data to the output specified by the stream parameter `fd`. Data is written at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

"l", "i", "s", "ul", "ui", "us", "d", "f", "c", "uc"

for writing respectively a long, an int, a short, an unsigned long, an unsigned int, an unsigned short, a double, a float, a char and an unsigned char. The bytes which are wrote are automatically swapped if necessary (by checking little-endian status) in order to produce machine independent binary files (in little-endian mode). This default swapping mode can be suppressed by adding a flag in the `mopen` function.

".l" or ".b"

It is also possible to write in little-endian or big-endian mode by adding a 'l' or 'b' character at the end of a type specification. For example "db" will write a double in big-endian mode.

Examples

```
filen = 'test.bin';
mopen(filen, 'wb');
mput(1996, 'l');mput(1996, 'i');mput(1996, 's');mput(98, 'c');

// force little-endian
mput(1996, 'll');mput(1996, 'il');mput(1996, 'sl');mput(98, 'cl');

// force big-endian
mput(1996, 'lb');mput(1996, 'ib');mput(1996, 'sb');mput(98, 'cb');

mclose();
mopen(filen, 'rb');
if 1996<>mget(1, 'l') then pause,end
if 1996<>mget(1, 'i') then pause,end
if 1996<>mget(1, 's') then pause,end
if 98<>mget(1, 'c') then pause,end
```

```
// force little-endian
if 1996<>mget(1,'ll') then pause,end
if 1996<>mget(1,'il') then pause,end
if 1996<>mget(1,'sl') then pause,end
if 98<>mget(1,'cl') then pause,end

// force big-endian
if 1996<>mget(1,'lb') then pause,end
if 1996<>mget(1,'ib') then pause,end
if 1996<>mget(1,'sb') then pause,end
if 98<>mget(1,'cb') then pause,end

mclose();
```

See Also

`mclose` , `meof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetl` , `mgetstr` , `mopen` , `mprintf` , `mputl` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

`mputl` — writes strings in an ascii file

```
r = mputl(txt ,file_desc)
```

Parameters

- `r`
returns %t or %f to check if function has correctly written on the file.
- `file_desc`
A character string giving the name of the file or a logical unit returned by `mopen`.
- `txt`
a vector of strings.

Description

`mputl` function allows to write a vector of strings as a sequence of lines in an ascii file.

Examples

```
fd = mopen(TMPDIR+'/text_mputl.txt','wt');  
mputl('Hello World',fd);  
mclose(fd);  
  
fd = mopen(TMPDIR+'/text_mputl.txt','rt');  
disp(mgetl(fd));  
mclose(fd);
```

See Also

`mget`, `mgetl`, `mclose`, `mfprintf`, `mput`, `mputstr`, `mopen`, `write`

Authors

S. Steer

Allan CORNET

Name

mputstr — write a character string in a file

```
mputstr(str [, fd]);
```

Parameters

fd

scalar. The fd parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str

a character string

Description

mputstr function allows to write a character string in a binary file.

See Also

`mclose` , `meof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

`mseek` — set current position in binary file.

```
mseek(n [,fd, flag])
```

Parameters

`n`

a positive scalar: The offset from origin in number of bytes.

`fd`

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

`flag`

a string, specifies the origin. Default value 'set'.

Description

The function `mseek()` sets the position of the next input or output operation on the stream `fd`. The new position is at the signed distance given by `n` bytes from the beginning, from the current position, or from the end of the file, according to the `flag` value which can be 'set', 'cur' or 'end'.

`mseek()` allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. `mseek()`, by itself, does not extend the size of the file.

Examples

```
file3='test3.bin'
fd1= mopen(file3,'wb');
for i=1:10, mput(i,'d'); end
mseek(0);
mput(678,'d');
mseek(0,fd1,'end');
mput(932,'d');
mclose(fd1)
fd1= mopen(file3,'rb');
res=mget(11,'d')
res1=[1:11]; res1(1)=678;res1($)=932;
if res1<>res ;write(%io(2),'Bug');end;
mseek(0,fd1,'set');

// trying to read more than stored data
res1=mget(100,'d',fd1);
if res1<>res ;write(%io(2),'Bug');end;
meof(fd1)
mclearerr(fd1)
mclose(fd1);
```

See Also

`mclose` , `meof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

mtell — binary file management

```
mtell([fd])
```

Parameters

`fd`

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

Description

The function `mtell()` returns the offset of the current byte relative to the beginning of the file associated with the named stream `fd`.

See Also

`mclose` , `meof` , `mfprintf` , `fprintfMat` , `mfscanf` , `fscanfMat` , `mget` , `mgetstr` , `mopen` , `mprintf` , `mput` , `mputstr` , `mscanf` , `mseek` , `mtell` , `mdelete`

Name

newest — returns newest file of a set of files

```
k=newest(paths)
k=newest(path1,path2,...,pathn)
```

Parameters

k
the index of the newest file

paths
a character string vector, paths(i) is the pathname of ith file

pathi
a character string, the pathname of ith file

Description

Given a set of pathnames newest returns the index of the newest one. Non existant files are supposed to be the oldest.

Examples

```
newest('SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bode
newest(['SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bod
newest('SCI/modules/graphics/macros/bode.'+['sci','bin'])
```

See Also

fileinfo

Name

`pathconvert` — pathnames conversion between posix and windows.

```
paths=pathconvert(paths [,flagtrail [,flagexpand [,type]]])
```

Parameters

`paths`

a string matrix giving a set of pathnames

`flagtrail`

boolean optional parameters. Its default value is TRUE.

`flagexpand`

boolean optional parameter. Its default value depends on the MSDOS variable.

`type`

a string 'u' or 'w'.

Description

`pathconvert` can be used to convert a set of pathnames (given by a string matrix `paths`) from windows native filename to posix-style pathnames and back. The target style is given by the optional string `type` which can be 'u' for Unix or 'w' for Windows. The default style is set according to the value of MSDOS. If MSDOS is TRUE (resp. FALSE) then default type is 'w' (resp. 'u').

Windows pathnames starting with `name:` are converted to pathnames starting with `/cygdrive/name/` using the cygwin convention.

`flagtrail` is an optional boolean parameter. When its value is TRUE (default value) a trailing separator ('\' or '/') is added at the end of the path if it was missing. If `flagtrail` is set to FALSE, the trailing separator is removed.

`flagexpand` is an optional boolean parameter. When its value is TRUE leading strings like HOME, SCI or ~ are expanded using environment variables.

Examples

```
pathconvert("SCI/modules/fileio\macros/foo.sci",%f,%f,"u")
pathconvert("SCI/modules/fileio\macros/foo.sci",%f,%f,"w")
pathconvert("SCI/modules/fileio/macros/foo.sci",%f,%t,"w")
pathconvert("HOME/modules/fileio/macros/foo.sci",%t,%t,"w")
pathconvert("c:/tmp",%f,%t,"u")
pathconvert("/cygdrive/c/tmp",%f,%f,"w")
```

See Also

`basename` , `listfiles`

Name

pathsep — returns path separator for current platform

```
s = pathsep()
```

Parameters

s
a string

Description

returns path separator. (':' on Linux or ';' on Windows)

Examples

```
pathsep()
```

Authors

A.C

Name

pwd — print Scilab current directory
pwd — get Scilab current directory

```
pwd  
x=pwd( )
```

Description

pwd returns in `ans` the Scilab current directory. `x=pwd( )` returns in `x` the Scilab current directory.

Examples

```
pwd  
x=pwd( )
```

See Also

chdir , cd

Name

removedir — Remove a directory

```
removedir('dirname')  
[status] = removedir('dirname','s')
```

Description

removedir('dirname') removes the directory dirname from the current directory. If the directory is not empty, files and subdirectories are removed. If dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = removedir('dirname') removes the directory dirname and its contents from the current directory, returning the status. Here, status is %T for success and is %F for error.

removedir is used by rmdir.

Examples

```
createdir(SCIHOME+'/Directory_test')  
removedir(SCIHOME+'/Directory_test')
```

See Also

mkdir , rmdir

Authors

A.C

Name

rmdir — Remove a directory

```
rmdir('dirname')
rmdir('dirname','s')
[status,message] = rmdir('dirname','s')
```

Description

`rmdir('dirname')` removes the directory `dirname` from the current directory. If the directory is not empty, you must use the `s` argument. If `dirname` is not in the current directory, specify the relative path to the current directory or the full path for `dirname`.

`rmdir('dirname','s')` removes the directory `dirname` and its contents from the current directory.

`[status,message] = rmdir('dirname','s')` removes the directory `dirname` and its contents from the current directory, returning the status, and a message. Here, `status` is 1 for success and is 0 for error.

Examples

```
mkdir(SCI,'Directory')
rmdir(SCI+'/Directory')
```

See Also

`cd` , `dir` , `mkdir`

Authors

A.C

Name

save_format — format of files produced by "save"

Description

Variables are saved by Scilab with the save function in the following format:

each variable record is appended consecutively to the file. The variable record begins with 6 long integer holding the variable name in encoded format (see the Remarks section below),

After that comes the variable type (long integer), then, depending on it, for:

Floating matrices (type 1)

row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (a long integer in {0,1}),

data (n*m*(it+1) doubles)

Polynomials (type 2) and Size implicit indices (type 129)

row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (long integer in {0,1}),

formal variable name (16 bytes),

index_table (m*n+1 long integers);

data ((N-1)*(it+1) doubles) , where N is the value of the last entry of the index_table

Booleans (type 4)

row_size m (a long integer),

column_size n (a long integer);

data (n*m long integers)

Floating sparse matrices (type 5)

row_size m (a long integer),

column_size n (a long integer),

real/complex_flag it (a long integer in {0,1}),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_row (m long integers),

column_index_non_zero_elements (nel long integers),

non_zero_values (nel*(it+1) doubles)

Boolean sparse matrices (type 6)

row_size m (a long integer),

column_size n (a long integer),

unused it (a long integer),

total_number_of_non_zero_elements nel (a long integer),
 number_of_non_zero_elements_per_row (m long integers),
 column_index_non_zero_elements (nel long integers)

Matlab sparse matrix (type 7)
 row_size m (a long integer),
 column_size n (a long integer),
 real/complex_flag it (a long integer in {0,1}),
 total_number_of_non_zero_elements nel (a long integer),
 number_of_non_zero_elements_per_column (n long integers),
 row_index_non_zero_elements (nel long integers),
 non_zero_values (nel*(it+1) doubles)

Integer matrices (type 8)
 row_size m (a long integer),
 column_size n (a long integer),
 integer_type (a long integer): 1,2,4, or 11,12,14 for signed and unsigned 1,2,4 bytes integers;
 data (n*m bytes for integer_type 1 or 11, n*m short integers for integer_type 2 or 12, n*m long integers for integer_type 4 or 14)

handles (type 9)
 version (4 bytes)
 row_size m (a byte),
 column_size n (a byte),
 data (m*n serialization_records)

A serialization_record is a flat representation of the C data structure associated with the corresponding graphic object. Each graphic object is defined by a (recursive) set of properties (see the get) function).

The saved serialization_record of a graphic object is structured as follow
 type_length n (a byte)

type (n bytes, the ascii codes of the type name)
 property_values record (variable length)

Strings (type 10)
 row_size m (a long integer),
 column_size n (a long integer),
 index_table (n*m+1 long integers);
 data (N long integers, the Scilab encoding of the characters (see code2str), where N is the value of the last entry of the index_table

Uncompiled functions (type 11)
 nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),
nin (long integer),
rhs_names (6*nin long integers, see the Remarks section below);
code_length N (a long integer),
code (N long integers)

Compiled functions (type 13)
nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),
nin (long integer),
rhs_names (6*nin long integers, see the Remarks section below),
pseudo_code_length N (a long integer),
pseudo_code (N long integers)

Libraries (type 14)
path_length np (a long integer),
path_name (np long integers: the path character codes sequence, (see code2str)),
number of names nn (long integer),
names (6*nn long integers, see the Remarks section below);

Lists (type 15), tlists (type 16), mlists (type 17)
number of fields n (a long integer),
index (n+1 long integers);
variables_sequence (n variables, each one written according to its format)

Pointers (type 128)
Not handled

Function pointers (type 130)
function_ptr (a long integer,(see funptr))
function_name_code (6 long integers,see the Remarks section below);

Remarks

Numbers (long interger, short integers, double) are stored using the little endian convention.

The variable names are stored as a sequence of 6 long integers, with a specific encoding. see the cvname.f file for details.

See Also

save, load, listvarinfile, type, typeof

Authors

compiled by Enrico Segre

Name

scanf — Converts formatted input on standard input

```
[v_1, . . . v_n]=scanf (format);
```

Parameters

format

Specifies the format conversion.

Description

The scanf functions get character data on standard input (%io(1)), interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see `scanf_conversion` for details.

See Also

`printf` , `read` , `fscanf` , `sscanf` , `scanf_conversion`

Name

scanf_conversion — scanf, sscanf, fscanf conversion specifications

Description

Each conversion specification in the format parameter contains the following elements:

- +
The character % (percent sign)
- +
The optional assignment suppression character *
- +
An optional numeric maximum field width
- +
A conversion code

The conversion specification has the following syntax:

`[*][width][size]convcode.`

The results from the conversion are placed in `v_i` arguments unless you specify assignment suppression with `*` (asterisk). Assignment suppression provides a way to describe an input field that is to be skipped. The input field is a string of nonwhite-space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates how to interpret the input field. You should not specify the `v_i` parameter for a suppressed field. You can use the following conversion codes:

- %
Accepts a single % (percent sign) input at this point; no assignment is done.
- d, i
Accepts a decimal integer;
- u
Accepts an unsigned decimal integer;
- o
Accepts an octal integer;
- x
Accepts a hexadecimal integer;
- e, f, g
Accepts a floating-point number. The next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float. The input format for floating-point numbers is a string of digits, with the following optional characteristics:
 - +
It can be a signed value.
 - +
It can be an exponential value, containing a decimal point followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.
 - +
It can be one of the special values INF, NaN,

- s
Accepts a string of characters.
- c
character value is expected. The normal skip over white space is suppressed.
- %lg
get value as a double.

See Also

scanf , scanf , fscanf

Name

`sscanf` — Converts formatted input given by a string

```
[v_1, ...v_n]=sscanf (string,format)
```

Parameters

`format`

Specifies the format conversion.

`string`

Specifies input to be read.

Description

This function is obsolete, use preferably the `msscanf` function which is more efficient and is more compatible with the C `sscanf` procedure.

The `sscanf` functions interpret character string according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see `scanf_conversion` for details.

See Also

`mprintf` , `msscanf` , `mfscanf` , `scanf_conversion`

Part VI. Graphics Library

Name

GlobalProperty — to customize the objects appearance (curves, surfaces...) in a plot or surf command.

Description

The GlobalProperty is an optional argument that can be used inside a plot or surf command. It allows a global customization of all the new plotted lines (respectively surfaces). It has to be given as a couple {PropertyName, PropertyValue}. Several couples can be set at the same time in a plot or surf call.

PropertyName must be a string defining the property to set. The PropertyValue can be a real, integer or string (scalar or matrix) depending on the type of property used. For example, to specify a red (color) longdash-dot (line style) with diamond marker (marker), the sequence should be : 'Color','red','LineStyle','-.-','Marker','diam'.

As you can see, a full complete spelling of each property name and value is not required but those arguments, specified in any order, must remain unambiguous. Furthermore, the string specification is not case sensitive. GlobalProperty is predominant on all LineSpec previously stated.

Here is a complete list of the **PropertyName** you can specify (using plot or surf) and their available associated **PropertyValue**. If not specified, those properties are available for both **Polyline** and **Fac3d** objects (created respectively by plot or surf) and, as previously said, they are applied to the new created objects (lines or surfaces).

Sometimes, you may have two PropertyName corresponding to one property : the first one is the equivalent default Matlab name, the second is the default name used by Scilab (i.e.: **Color** or **Foreground** for a line, see below).

CData or ColorData:

a real matrix specifying the color at every points defined by Z matrix. This property is linked to the object's `data.color` property (see `surface_properties`). Note that this property is available for surfaces only.

CDataMapping or ColorDataMapping:

a string with value 'scaled' or 'direct'. If a `data.color` is set, each index color data specifies a single value for each vertex. `cdata_mapping` determines whether those indices are scaled to map linearly into the current colormap ('scaled' mode) or point directly into this colormap ('direct' mode). This property is useful when `color_flag` equals 2,3 or 4. Note that this property exists only with **Fac3d** entities. Note also that `plot3d` has 'direct' mode by default and `surf` has 'scaled' mode by default.

Clipping:

a string "on" or "off" defining the clipping mode ("on" by default). It is equivalent to the `clip_state` property. This field contains the `visible` property (see `polyline_properties`). Note that this property is not yet available for surface entities.

Color or Foreground:

a string defining a known color (see `color_list`) or a 1x3 (or 3x1) RGB vector defining a color number. Color number is given as a 3-uple R, G, B corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's **foreground** property (see `polyline_properties`). **Warning** : Color is not available for surfaces objects. The **Foreground** property exists for surfaces objects but is linked to the Matlab **EdgeColor** property (see `surface_properties`).

EdgeColor or Foreground:

a string defining a known color (see `color_list`) or a 1x3 (or 3x1) RGB vector defining a color number. Color number is given as a 3-uple R, G, B corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the surface **foreground** property (see `surface_properties`). **Warning** : For polyline objects, the **Foreground** property exists with a different meaning (see above) and **EdgeColor** does not exist at all.

FaceColor:

a string with value 'none', 'flat' or 'interp' specifying the way the facet's color are rendered. When 'none' is selected, a mesh of the surface is drawn; if 'flat' (default mode) is set, the `Fac3d color . data` values determine one color per facet using the color of the first vertex of the facet. If the value is 'interp', an interpolated shading is done on the surface using `color . data` to determine a color at each vertex of each facet.

LineStyle:

This property value should be a string defining a line style. This property is linked to the object's `line_style` property (see `polyline_properties` or `surface_properties`).

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dotted line
none	No line

Marker or MarkStyle:

A string defining the marker type. Note that if you specify a marker without a line style, both line (with default solid mode enabled) and marker are drawn. This property is linked to the object's `mark_style` and `mark_mode` properties (see `polyline_properties` or `surface_properties`).

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram'	Five-pointed star (pentagram)
'none'	No marker (default)

MarkerEdgeColor or MarkForeground:

a string defining a known color (see `color_list`) or a 1x3 (or 3x1) RGB vector defining a color number. Color number is given as a 3-uple R, G, B corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's `mark_foreground` property (see `polyline_properties` or `surface_properties`).

MarkerFaceColor or MarkBackground:

a string defining a known color (see `color_list`) or a 1x3 (or 3x1) RGB vector defining a color number. Color number is given as a 3-uple R, G, B corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's `mark_background` property (see `polyline_properties` or `surface_properties`).

MarkerSize or MarkSize:

a scalar defining the marker size in `point` unit. This property is linked to the object's `mark_size` property with `mark_size_unit` enabled to `"point"` (see `polyline_properties` or `surface_properties`).

Visible:

a string `"on"` or `"off"` defining the visibility mode (`"on"` by default). This property is linked to the object's `visible` property (see `polyline_properties` or `surface_properties`).

X data:

a real vector or matrix (re-)defining the given data for all the plotted lines or surfaces. Concerning dimensions, note that this new data must match all the previous specified X data : that is to say all those data matrices must be of the same dimensions. This property is linked to the object's `data.x` property (see `polyline_properties` or `surface_properties`).

Y data:

a real vector or matrix (re-)defining the given data for all the plotted lines or surfaces. Concerning dimensions, note that this new data must match all the previous specified Y data : that is to say all those data matrices must be of the same dimensions. This property is linked to the object's `data.y` property (see `polyline_properties` or `surface_properties`).

Z data:

when used with `plot`, a real vector or matrix adding a

Z data for all the plotted lines ; with `surf`, a real matrix (re-)defining the given data for all the surfaces. Concerning dimensions, note that this new data must match all the previous specified X and Y data. This property is linked to the object's `data.z` property (see `polyline_properties` or `surface_properties`).

Examples

```
// -----  
// With the plot command :  
// -----  
x=1:10; // Init.  
plot(x,sin(x),'color','red','linest','-.','marker','>','markeredge','cyan','markerface','cyan')  
clf();  
  
// combinations' order in {PropertyName,PropertyValue} does not matter  
plot(x,sin(x),'marker','p','markerface','cyan','markersize',10)  
clf();  
  
// combination of LineSpec and GlobalProperty shows the GlobalProperty predominance  
plot(x,x.*x,'*cya--','color','gr','linestyle','-','marker','sq','markersize',6,  
      'markerface','cyan')  
clf();  
  
//multiple plots with different LineSpecs and finally some global GlobalProperty  
clf();  
t=0:%pi/20:2*pi;  
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo','markstyle','diam')  
  
// -----  
// With the plot2d command :  
// -----  
function draw_marks(title)  
    a=gca();  
    a.title.text=title;  
    a.mark_size=8;
```

```

a.data_bounds=[-1.5 1.5 -1.5 1.5];

theta=(1/15)*(2*pi)*[0:15];
plot2d(cos(theta),sin(theta));

for i=0:14 do
    theta=(i/15)*(2*pi);
    plot2d(cos(theta),sin(theta),style=-i);
end
endfunction

clf();
subplot(2,2,1)
draw_marks("black foreground / white background")

subplot(2,2,2)
a=gca();
a.mark_foreground=-1;
a.mark_background=4;
draw_marks("black foreground / cyan background")

subplot(2,2,3)
a=gca();
a.mark_background=0;
draw_marks("black foreground / invisible background")

subplot(2,2,4)
a=gca();
a.mark_foreground=0;
a.mark_background=4;
draw_marks("invisible foreground / cyan background")

// -----
// With the surf command :
// -----

Z= [ 0.0001 0.0013 0.0053 -0.0299 -0.1809 -0.2465 -0.1100 -0.
    0.0005 0.0089 0.0259 -0.3673 -1.8670 -2.4736 -1.0866 -0.160
    0.0004 0.0214 0.1739 -0.3147 -4.0919 -6.4101 -2.7589 -0.277
    -0.0088 -0.0871 0.0364 1.8559 1.4995 -2.2171 -0.2729 0.836
    -0.0308 -0.4313 -1.7334 -0.1148 3.0731 0.4444 2.6145 2.441
    -0.0336 -0.4990 -2.3552 -2.1722 0.8856 -0.0531 2.6416 2.406
    -0.0137 -0.1967 -0.8083 0.2289 3.3983 3.1955 2.4338 1.212
    -0.0014 -0.0017 0.3189 2.7414 7.1622 7.1361 3.1242 0.663
    0.0002 0.0104 0.1733 1.0852 2.6741 2.6725 1.1119 0.197
    0.0000 0.0012 0.0183 0.1099 0.2684 0.2683 0.1107 0.019

clf();
f=gcf();
f.figure_size = [610,724];
subplot(211)
surf(Z,'facecol','interp','ydat',101:110,'edgecol','magenta')
subplot(212)
surf(Z,'edgeco','b','marker','d','markersiz',9,'markerfac','k','xdata',-50:-41)

```

See Also

LineSpec , plot , surf , clf , polyline_properties , surface_properties

Authors

F.Leray

Name

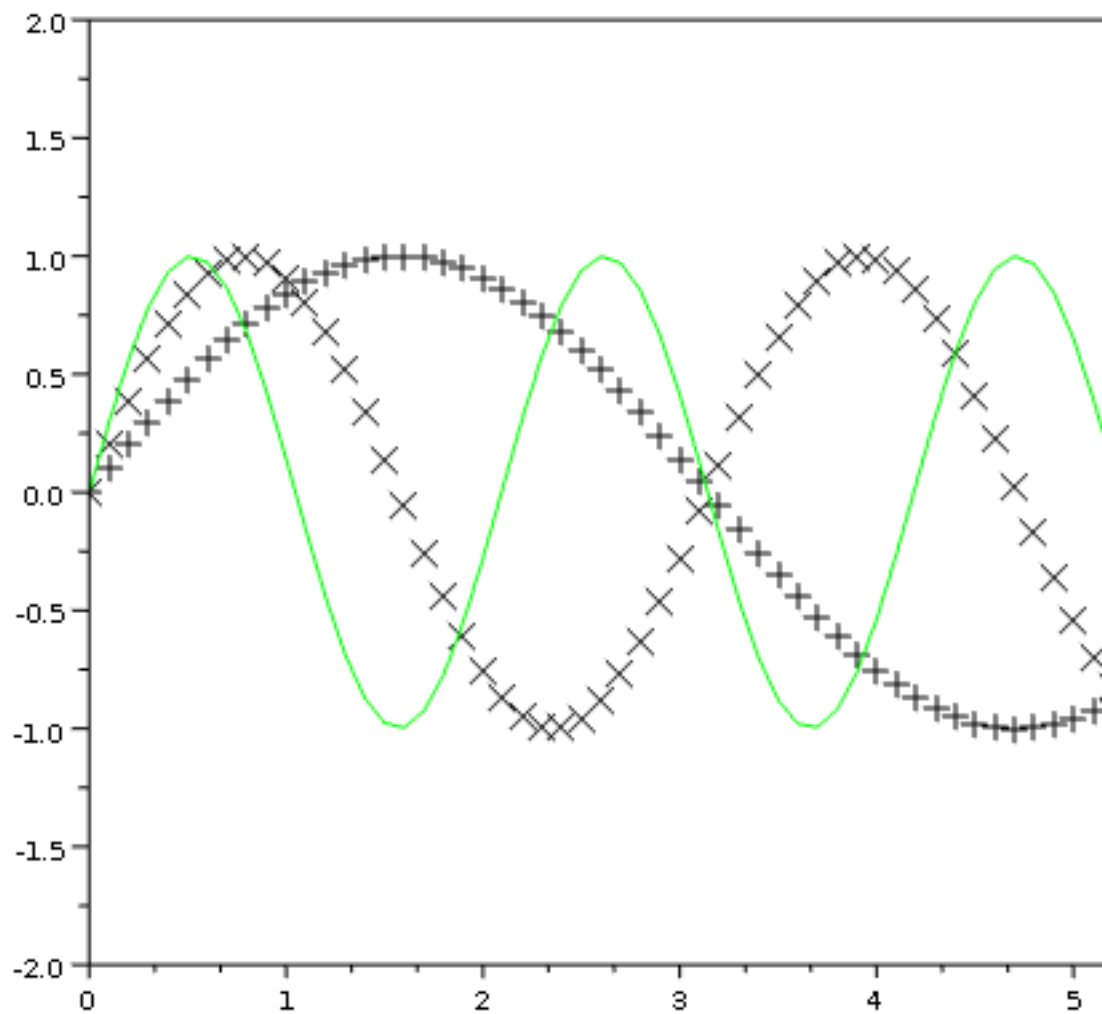
Graphics — graphics library overview

2d plotting

`plot2d`

Plot a curve

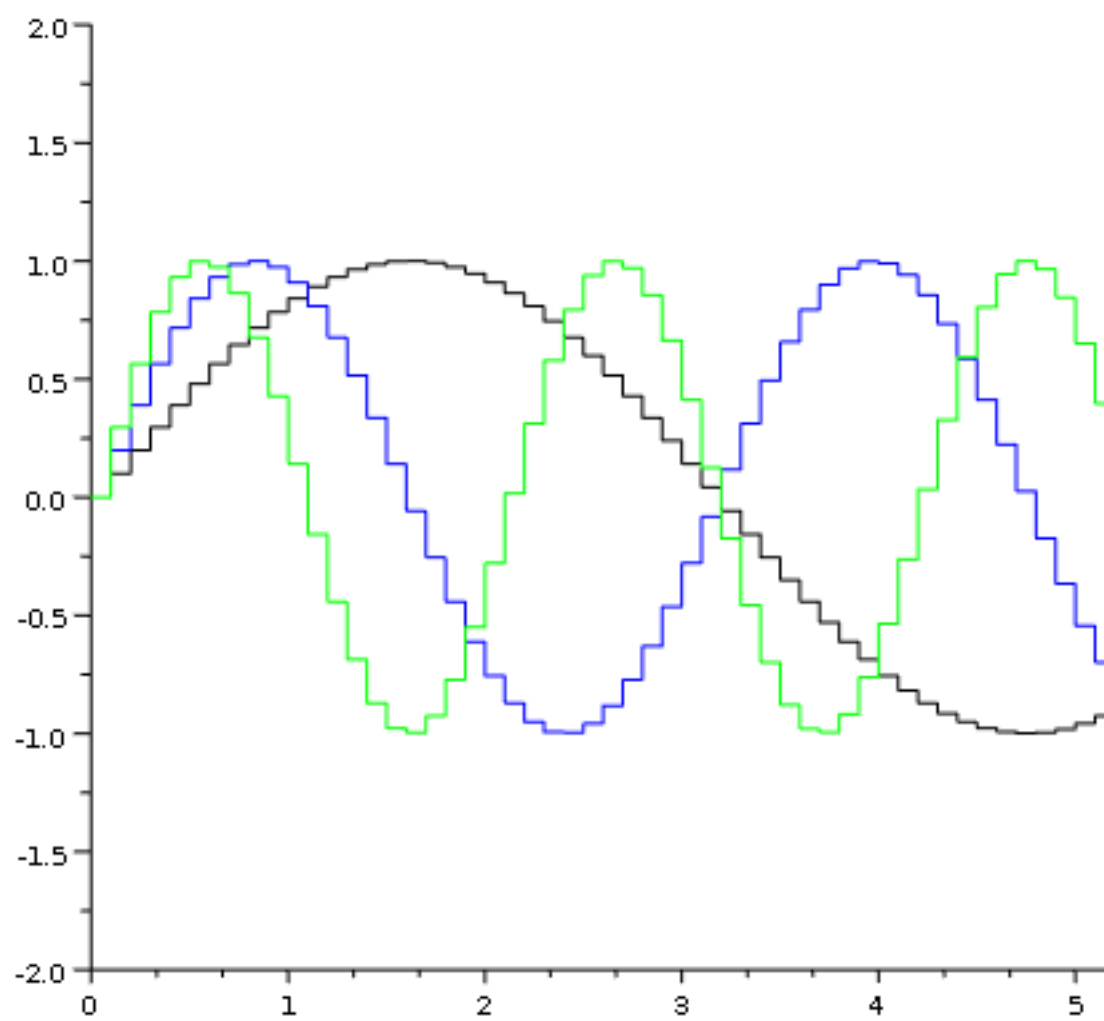
Example:



`plot2d2`

Plot a curve as step function

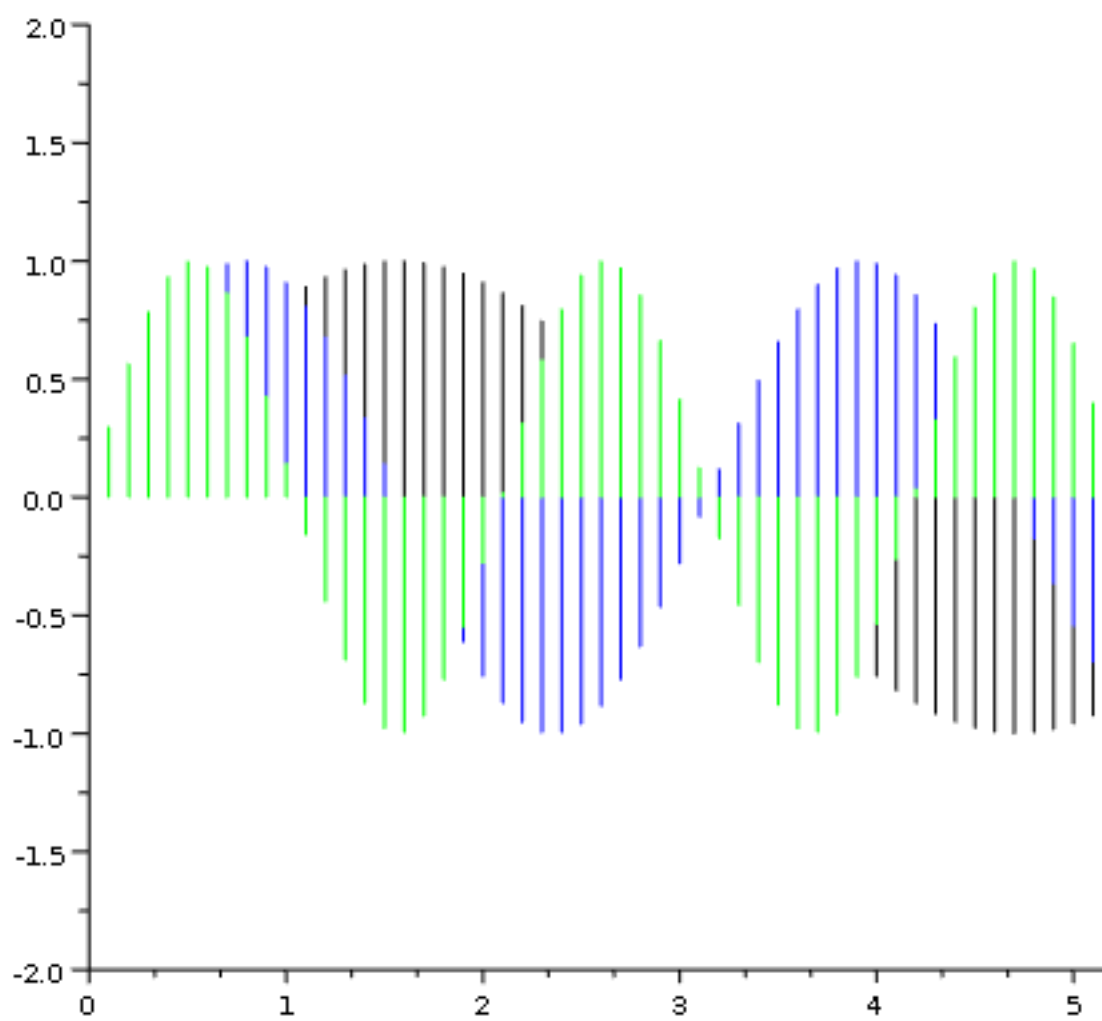
Example:



`plot2d3`

Plot a curve with vertical bars

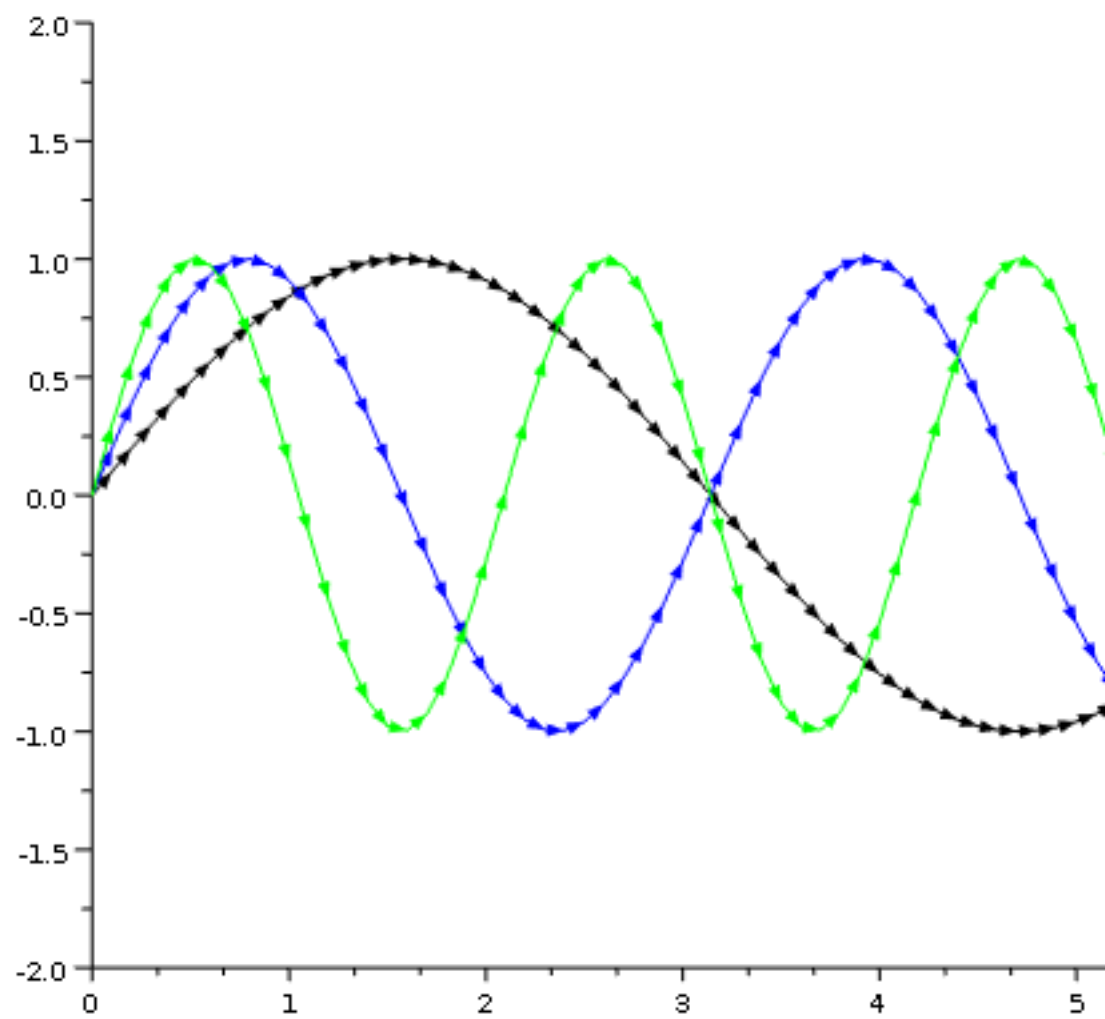
Example:



`plot2d4`

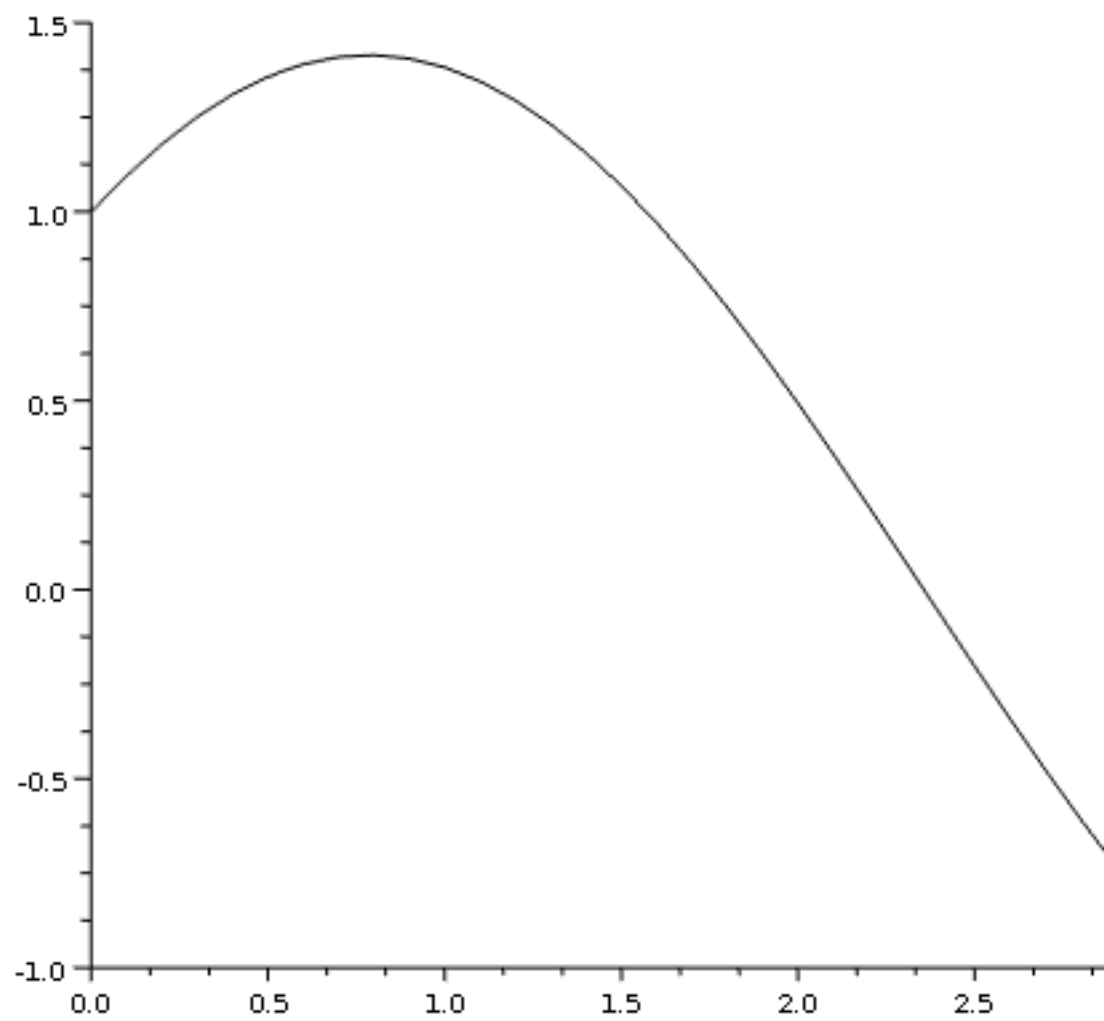
Plot a curve with arrows

Example:



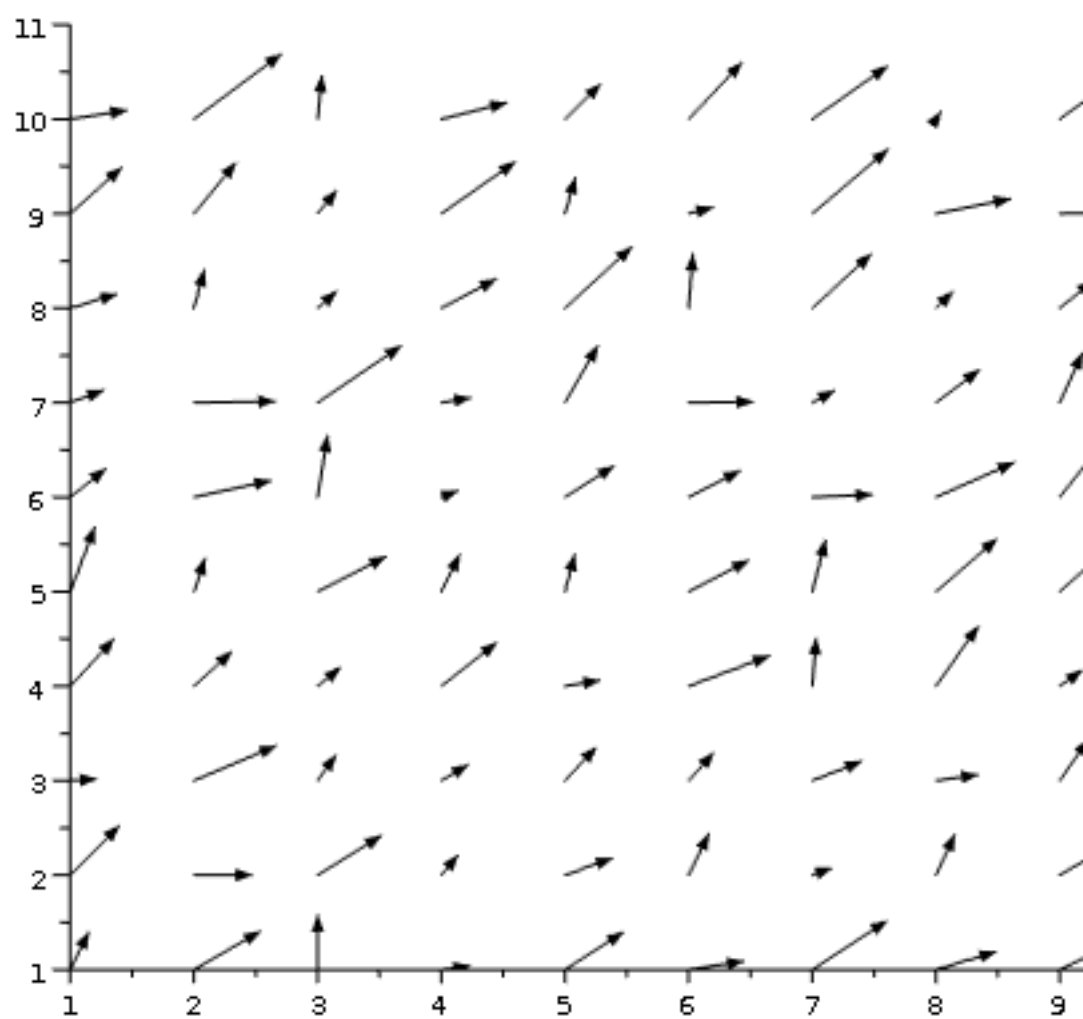
`fplot2d`
Plot a curve defined by a function

Example:



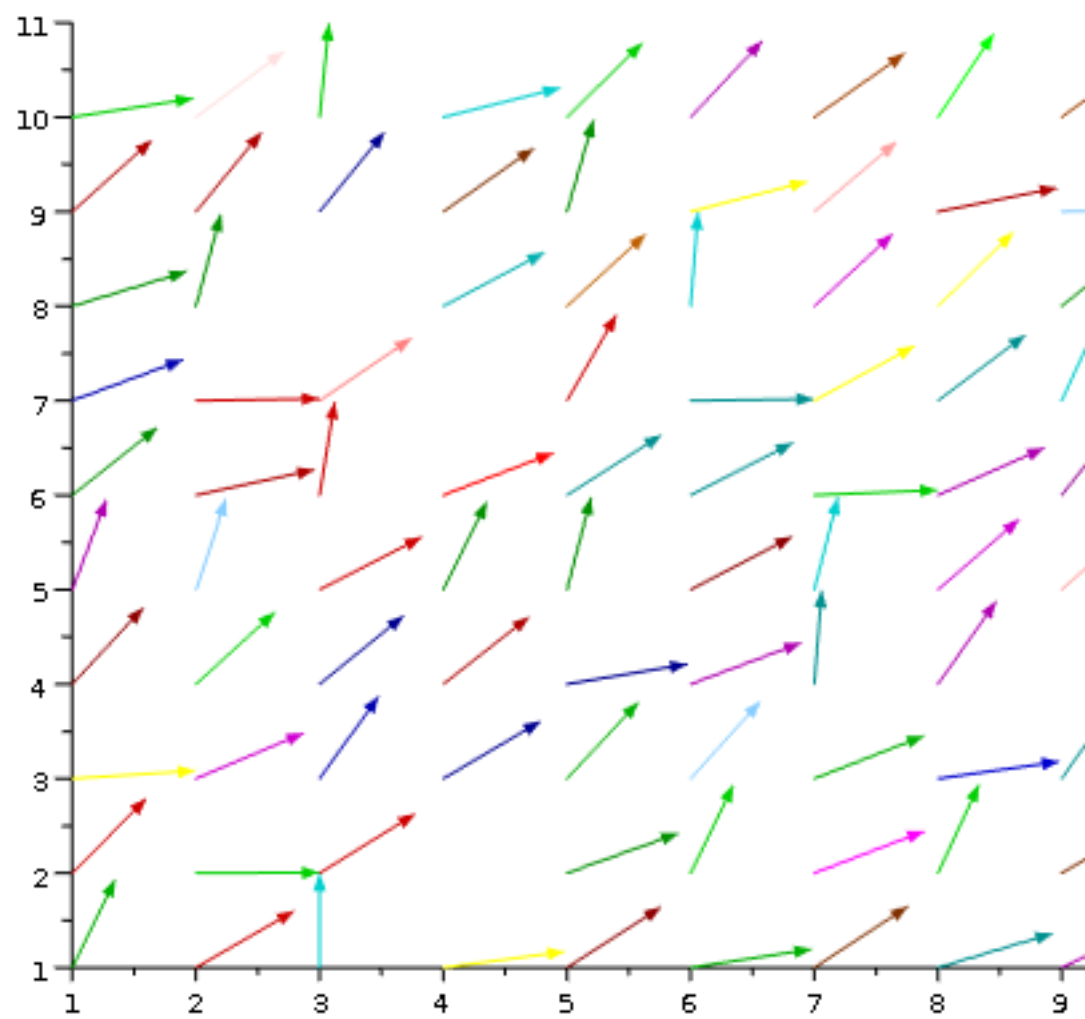
champ
2D vector field

Example:



champ1
2D vector field with colored arrows

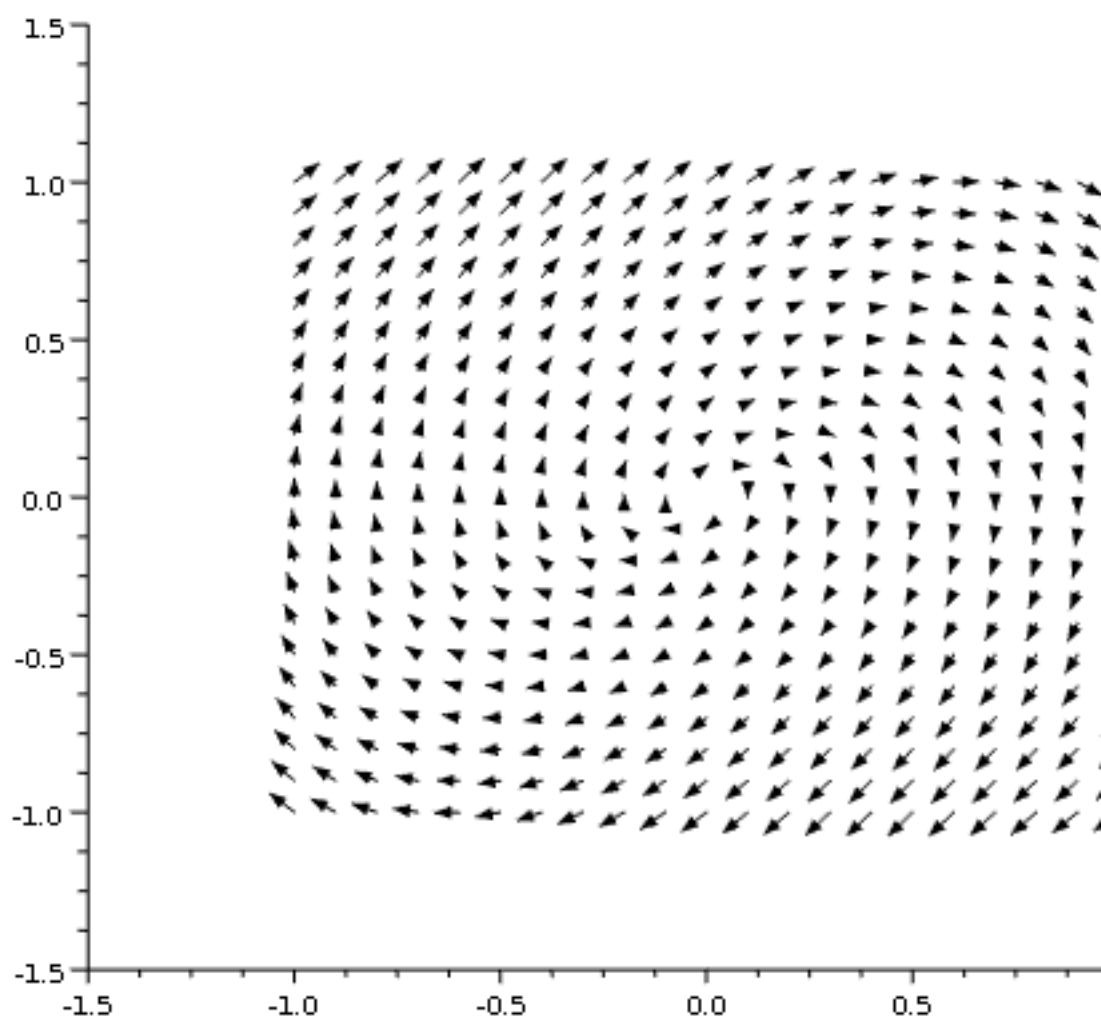
Example:



fchamp

Direction field of a 2D first order ODE

Example:



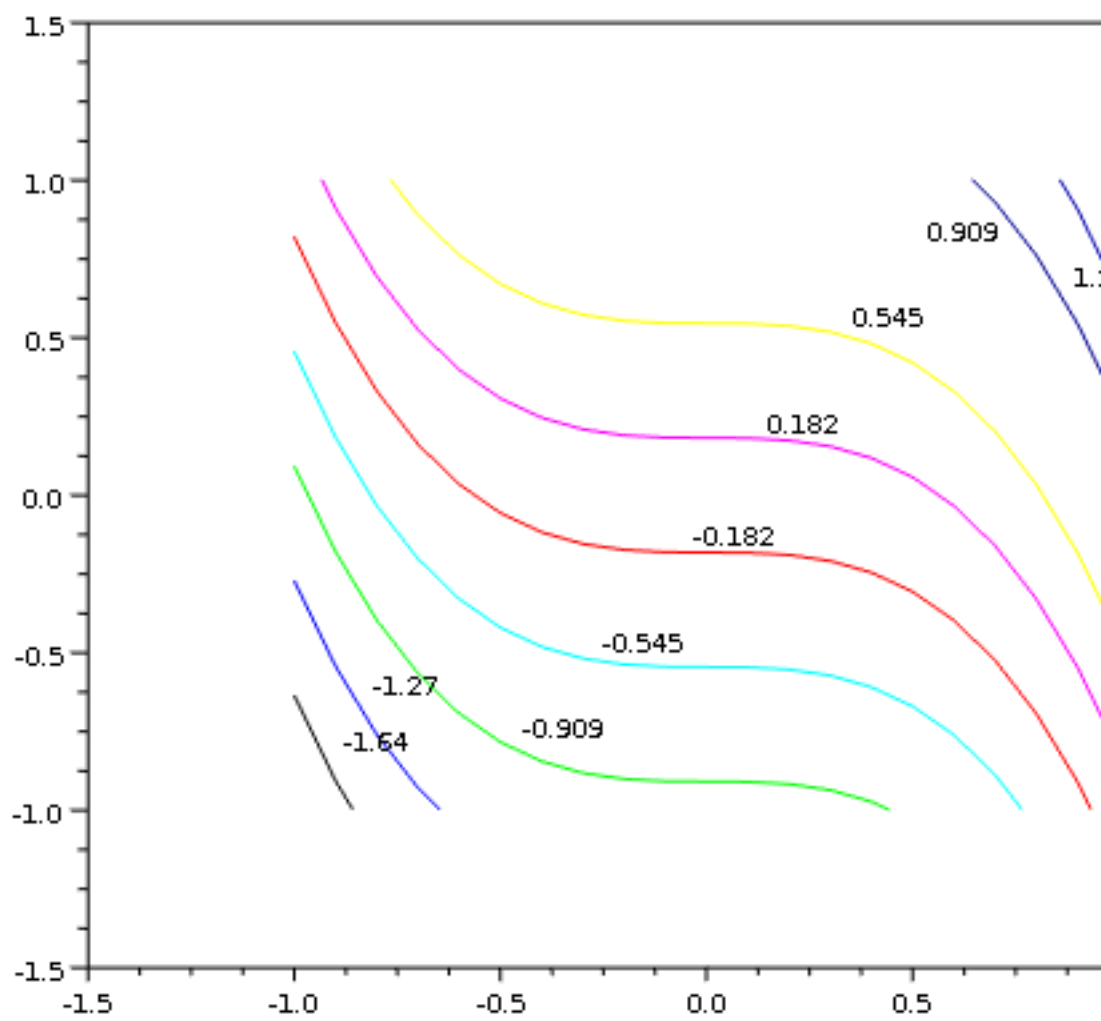
`contour2d`

Level curves of a surface on a 2D plot

`fcontour2d`

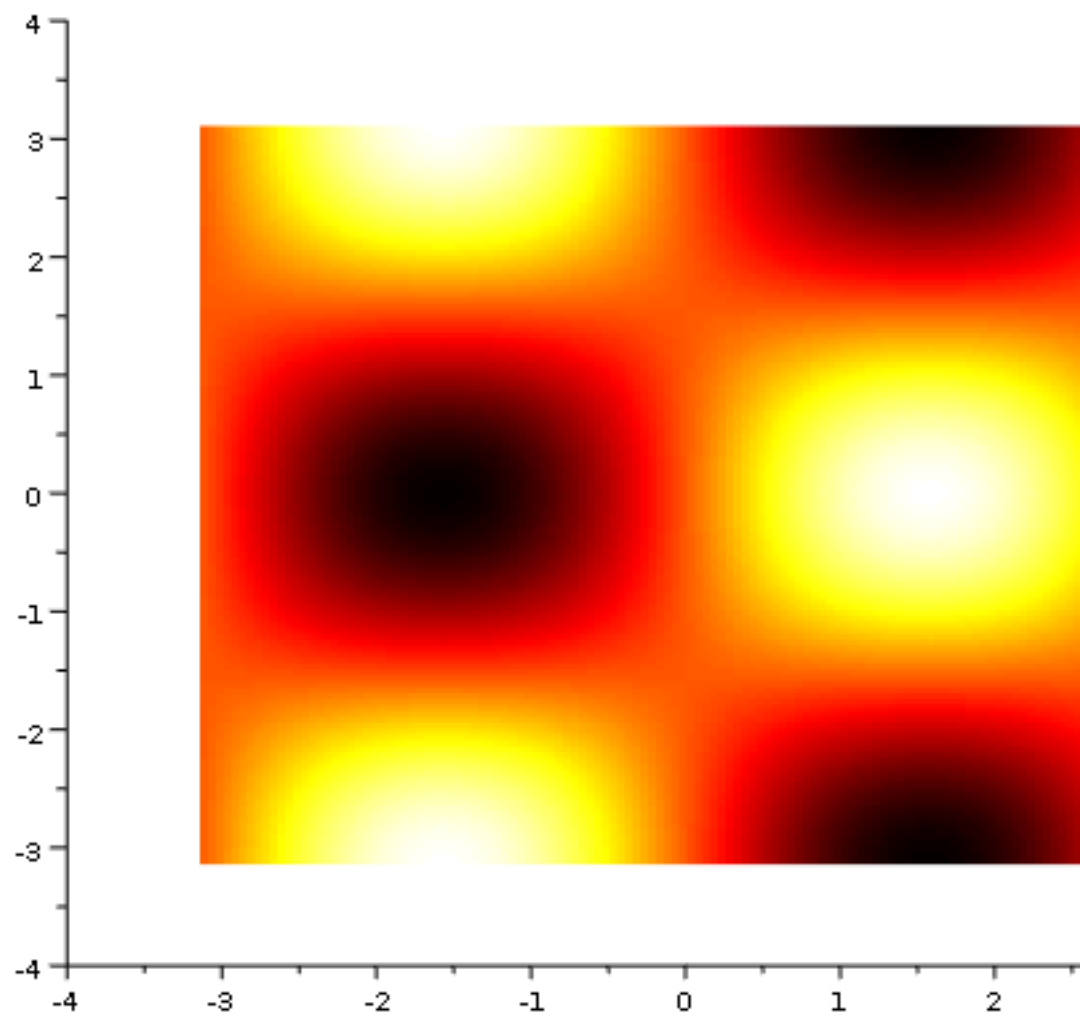
Level curves of a surface defined by a function on a 2D plot

Example:



grayplot
2D plot of a surface using colors

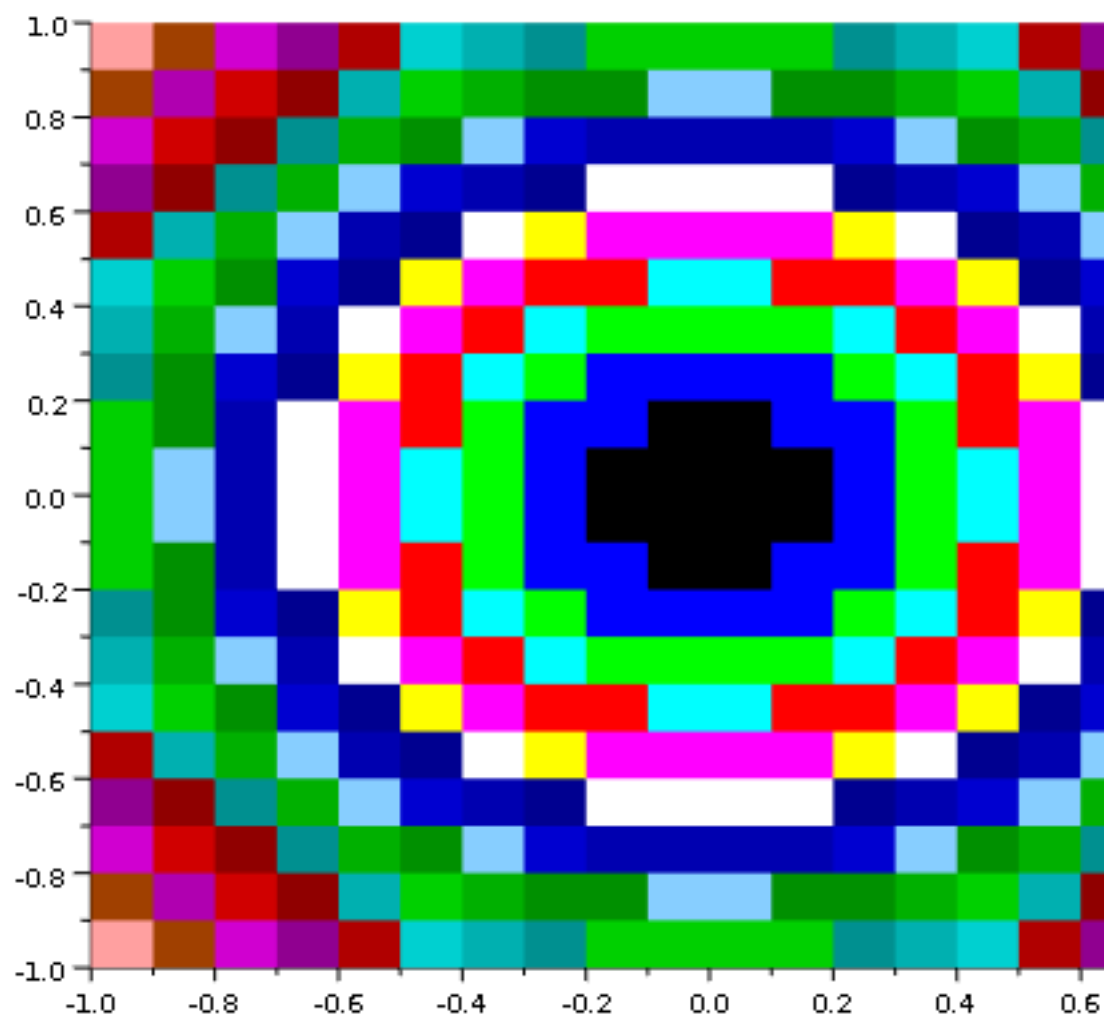
Example:



`fgrayplot`

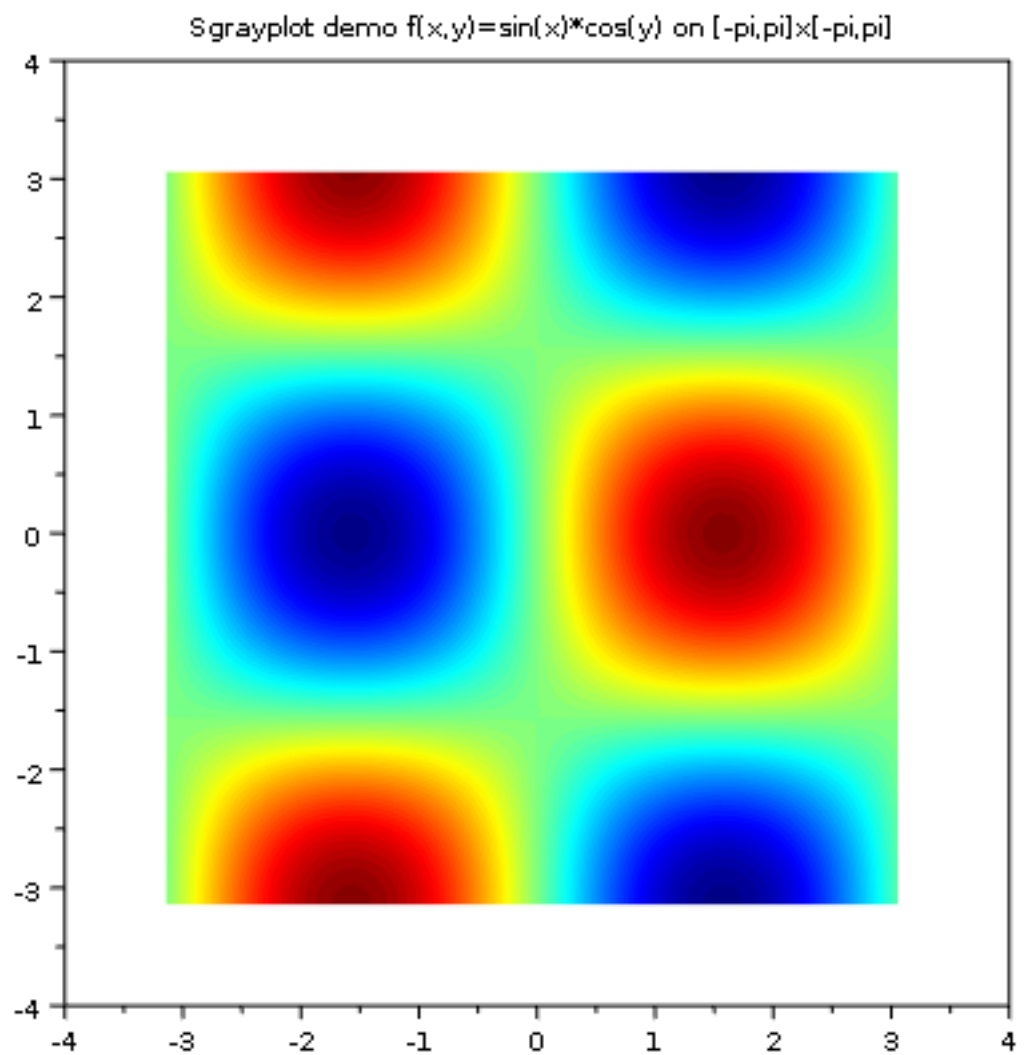
2D plot of a surface defined by a function using colors

Example:



Sgrayplot
Smooth 2D plot of a surface using colors

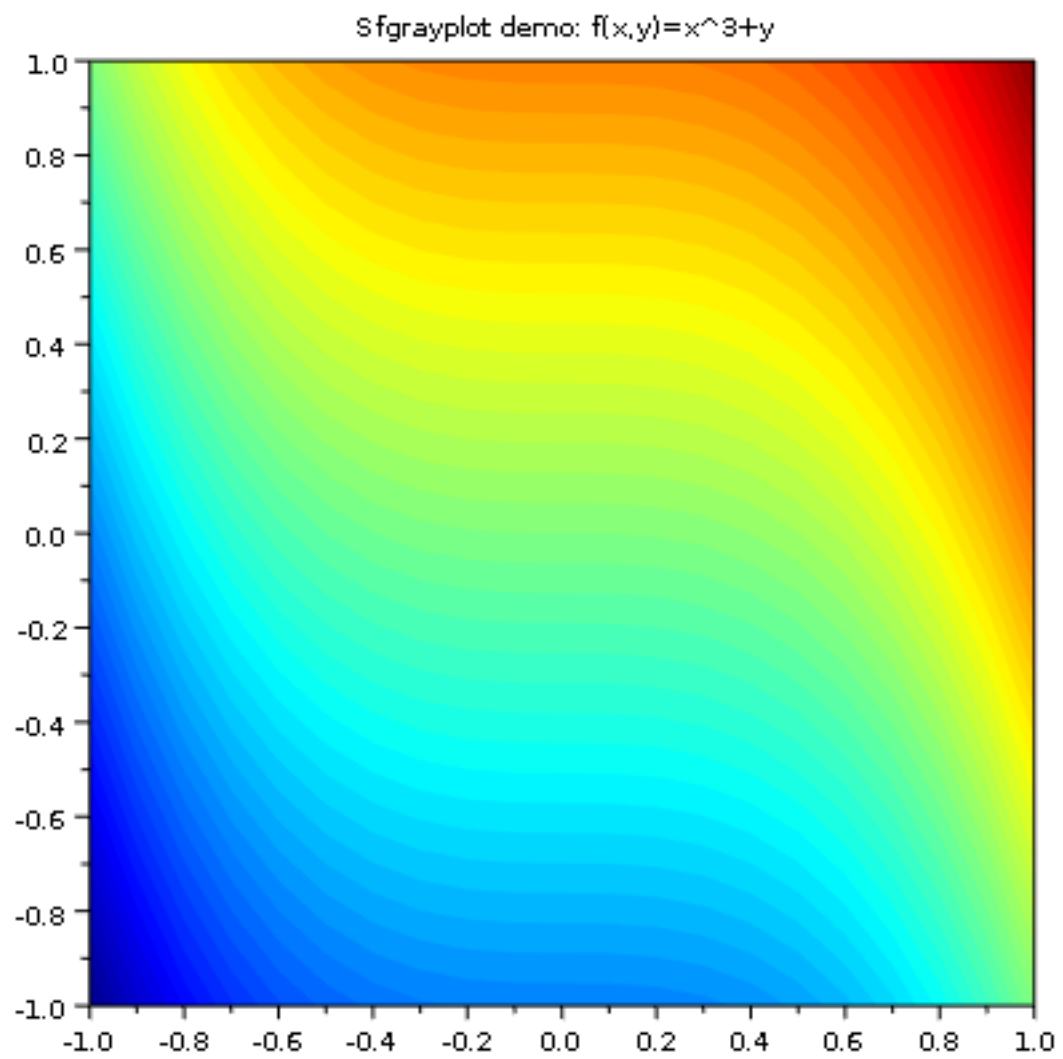
Example:



Sfgrayplot

Smooth 2D plot of a surface defined by a function using colors

Example:



xgrid

Add a grid on a 2D plot

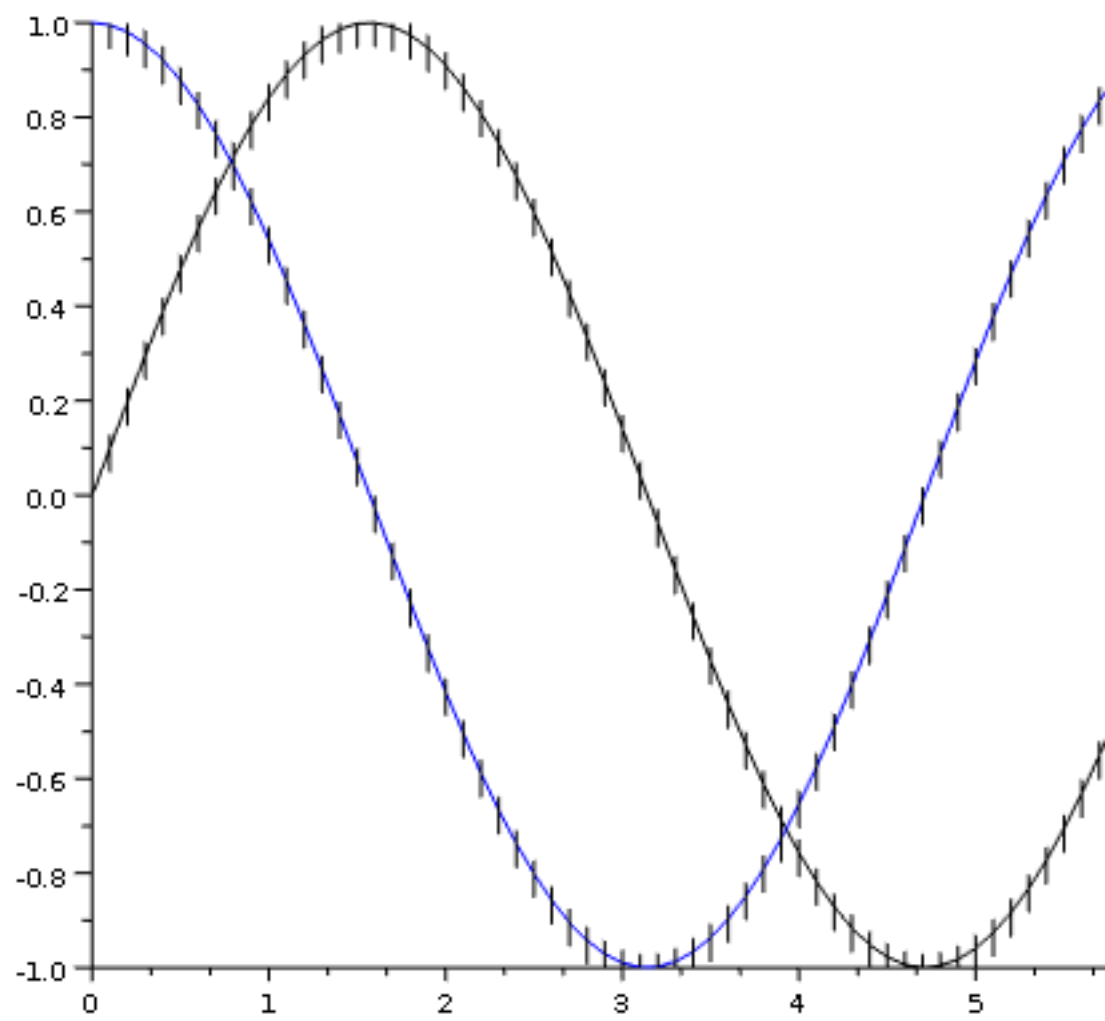
Example:



`errbar`

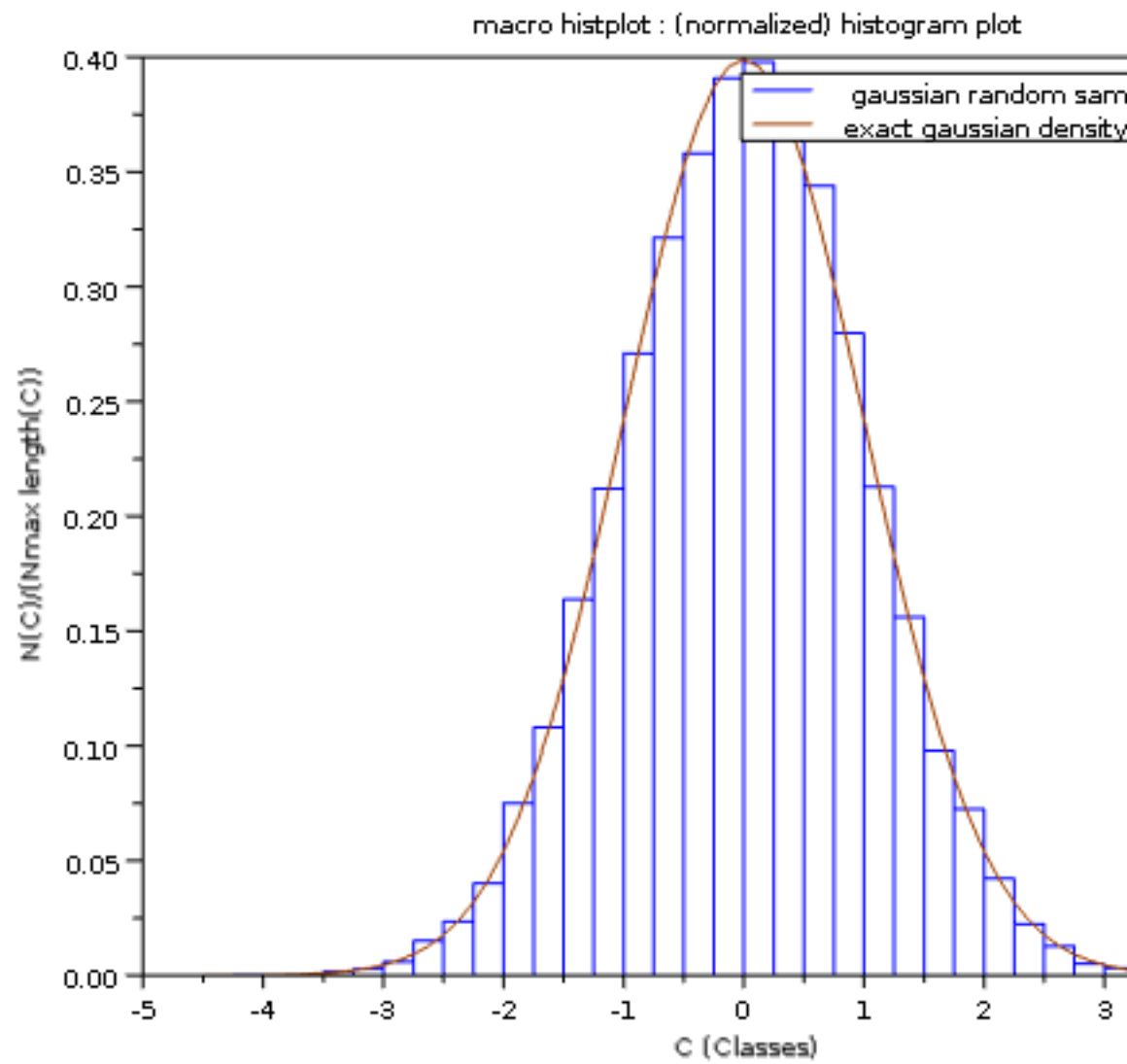
Add vertical error bars on a 2D plot

Example:



histplot
Plot a histogram

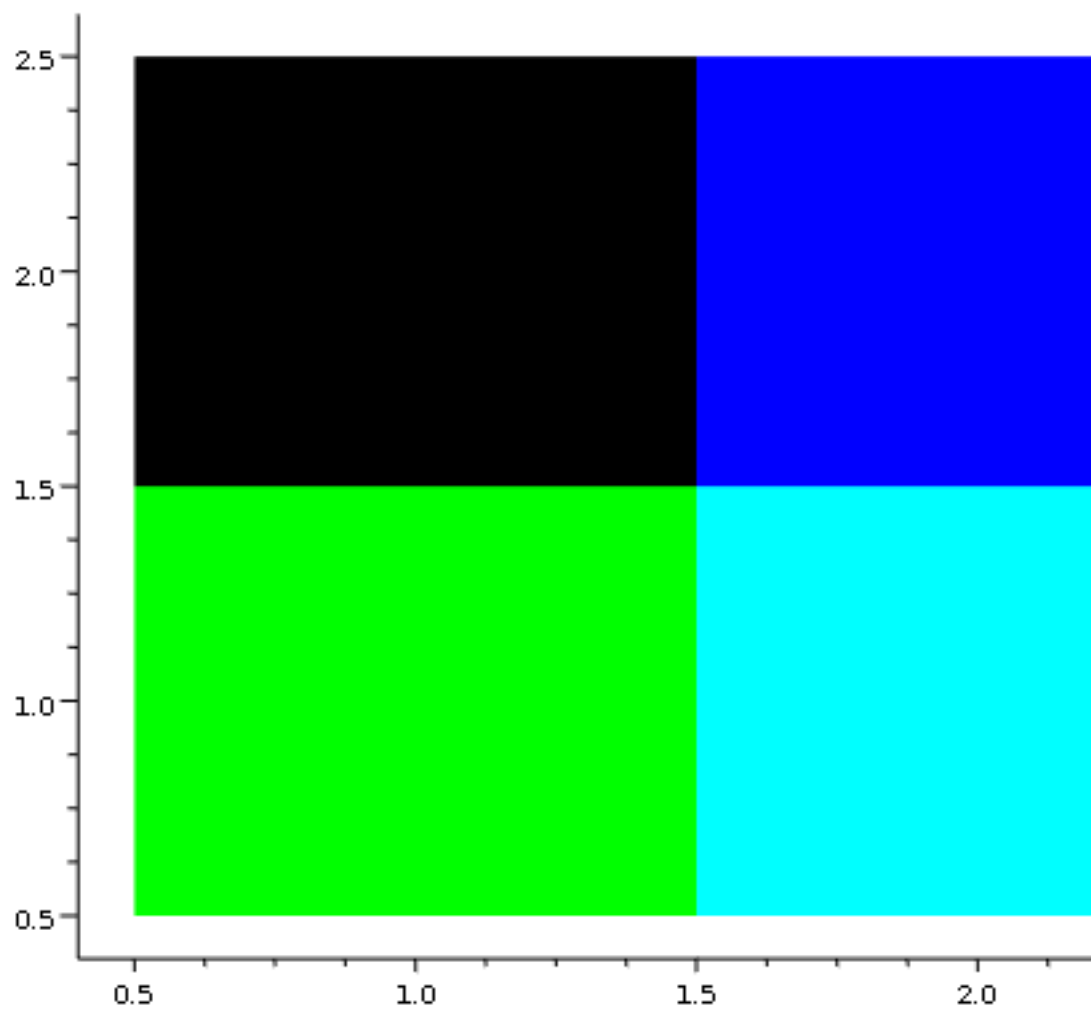
Example:



Matplot

2D plot of a matrix using colors

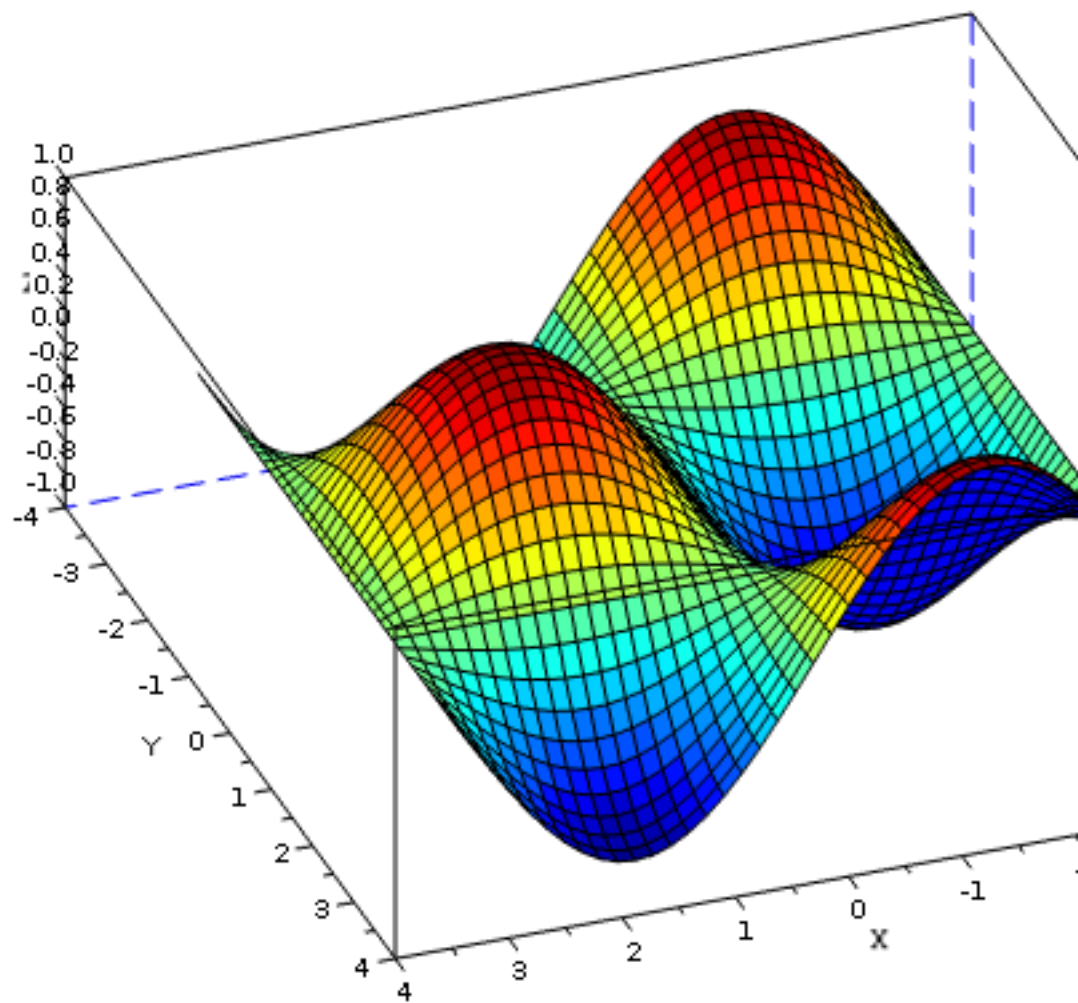
Example:



3d plotting

`plot3d`
Plot a surface

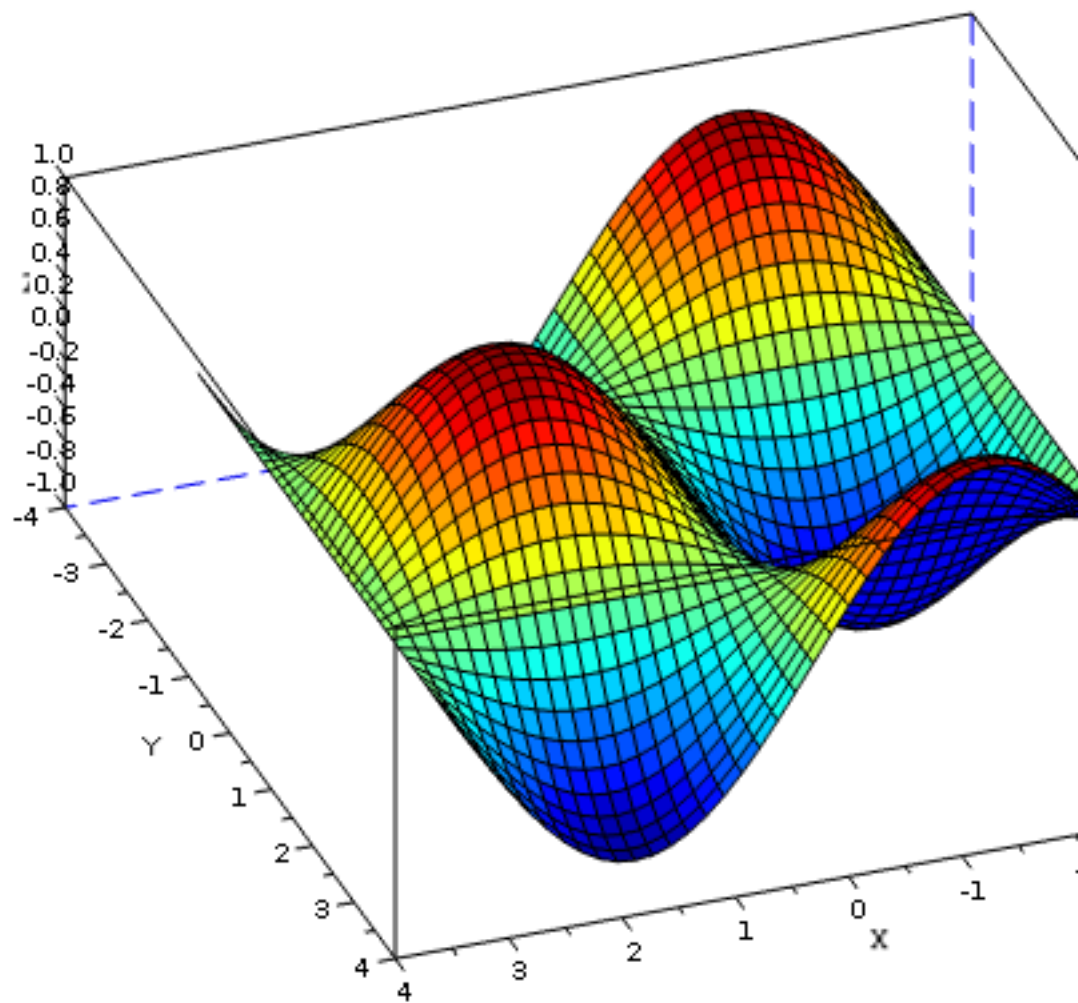
Example:



`plot3d1`

Plot a surface with gray or color level

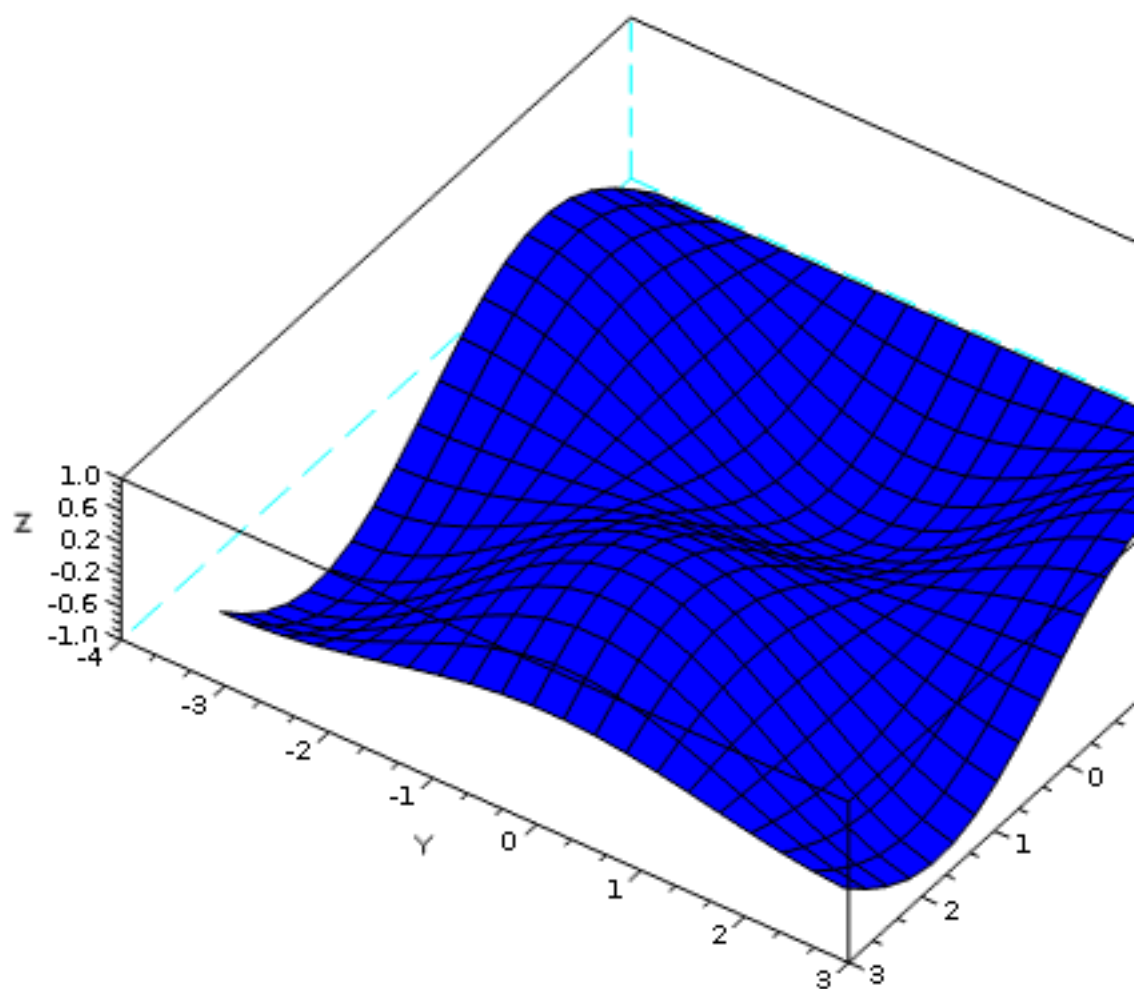
Example:



`fplot3d`

Plot a surface defined by a function

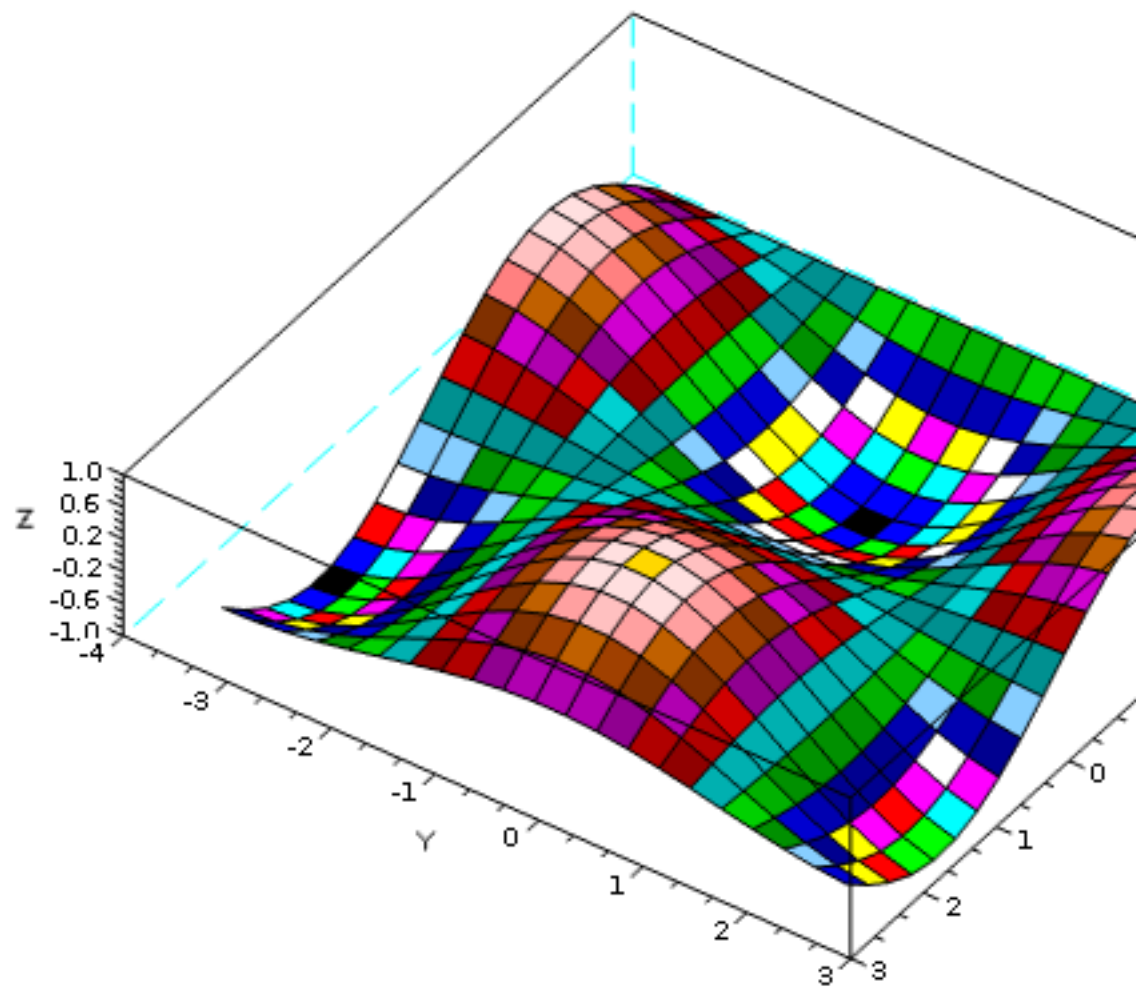
Example:



`fplot3d1`

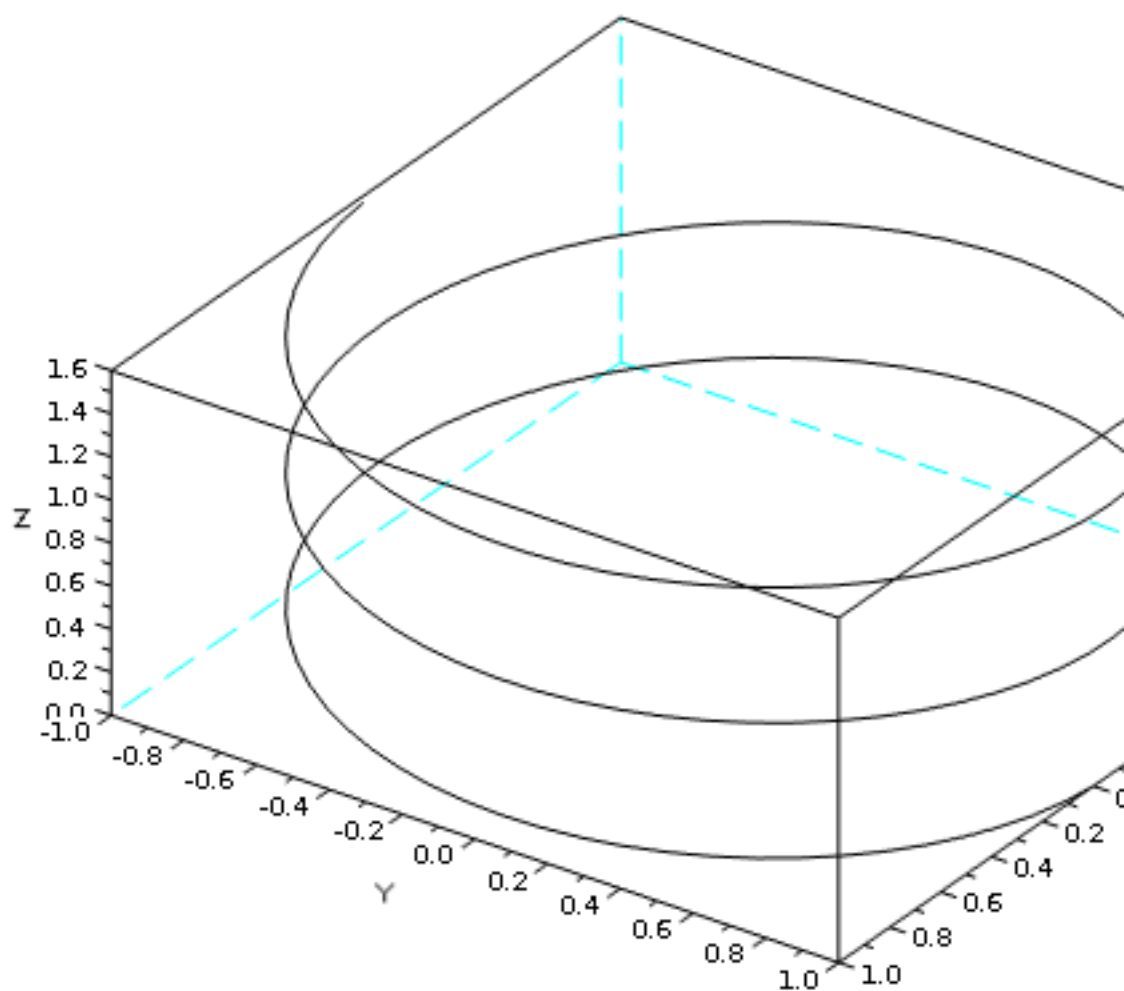
Plot a surface defined by a function with gray or color level

Example:



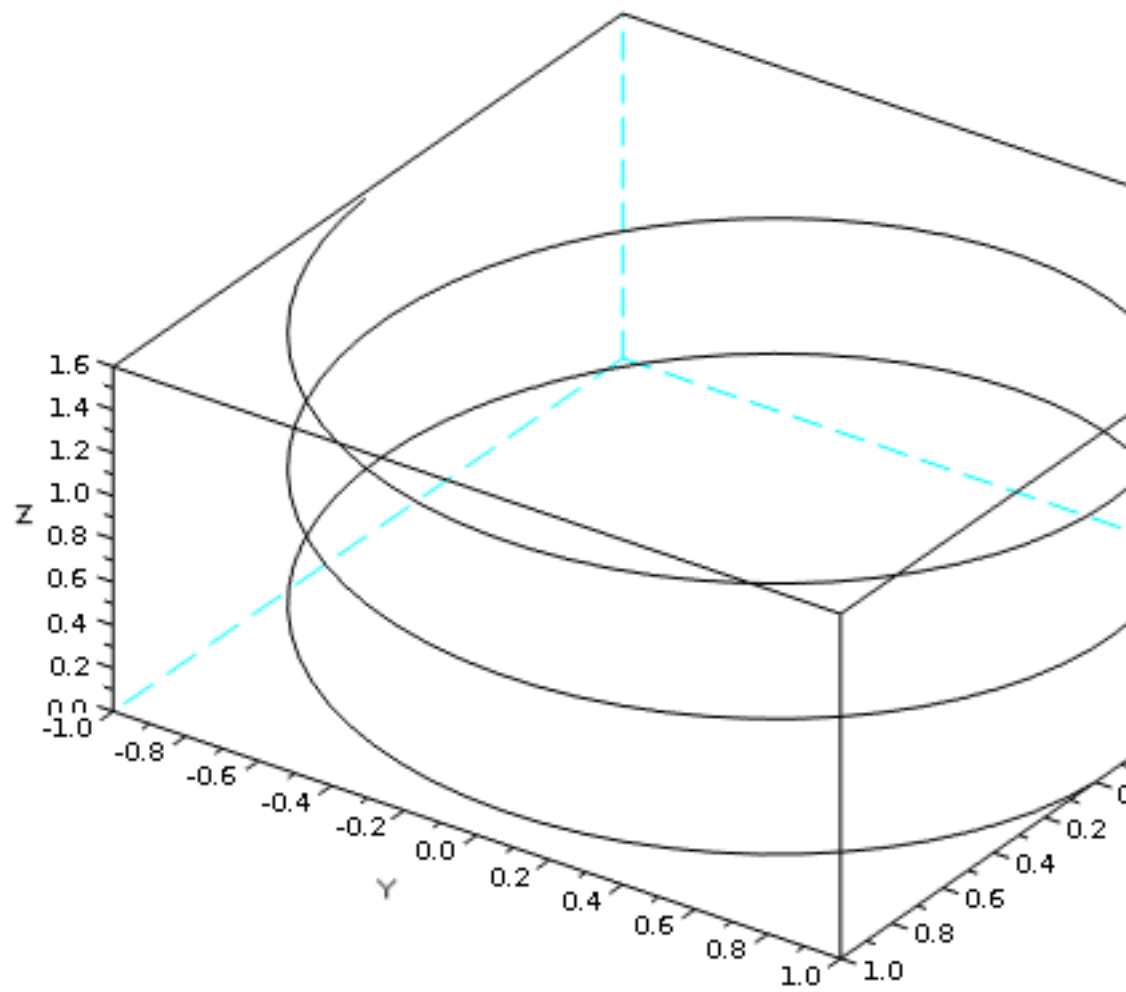
param3d
Plot one curve

Example:



param3d1
Plots curves

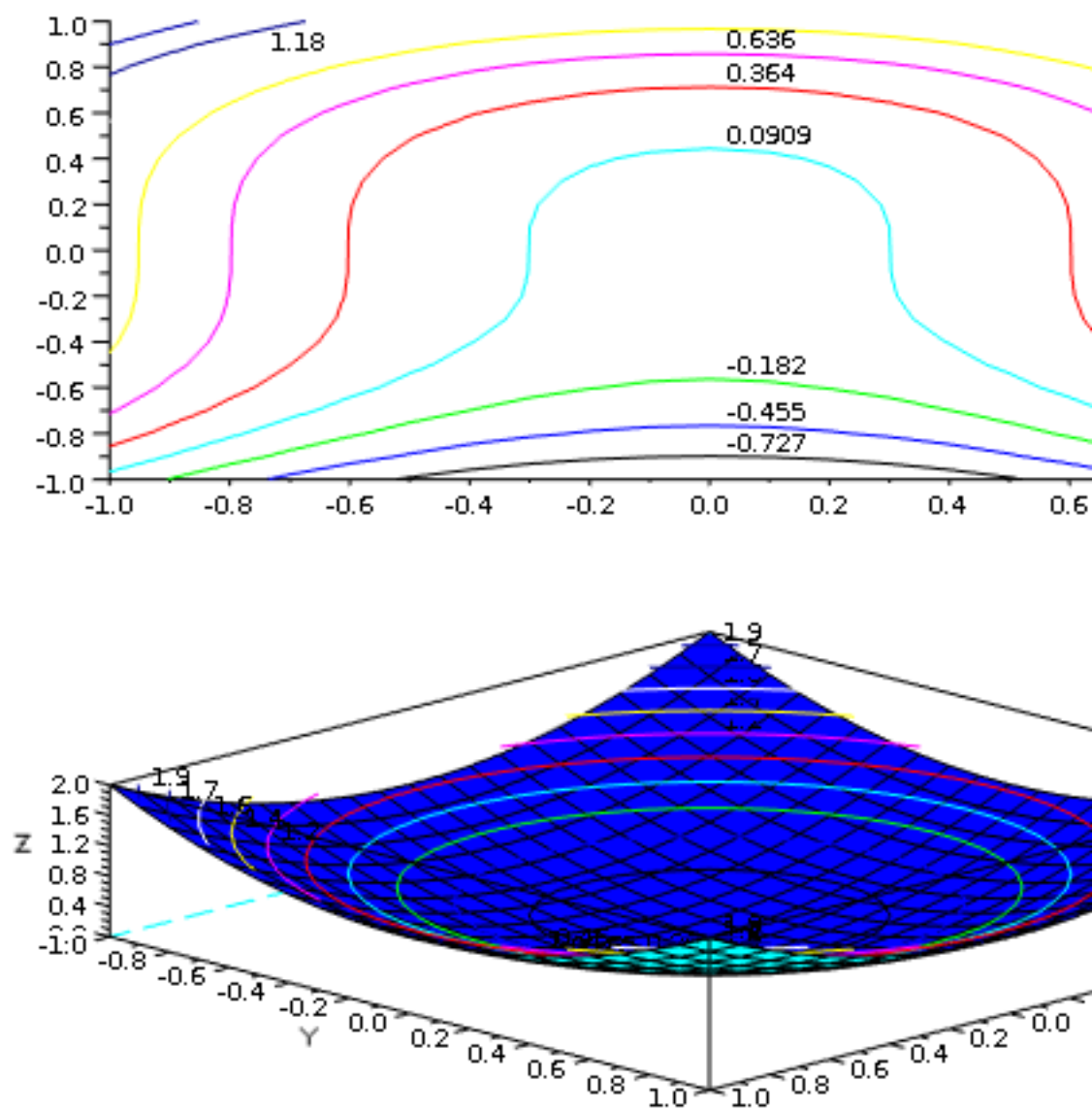
Example:



contour

Level curves on a 3D surface

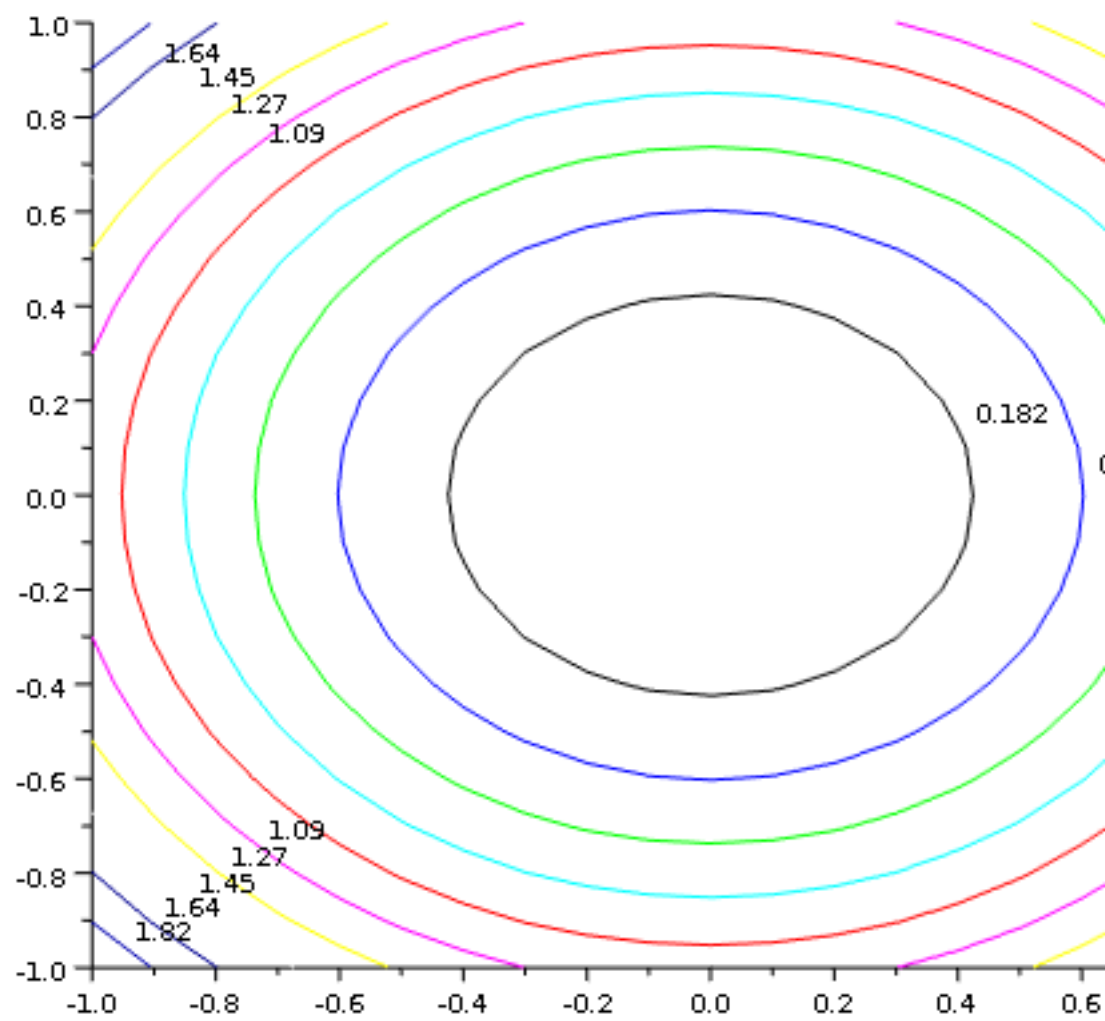
Example:



`fcontour`

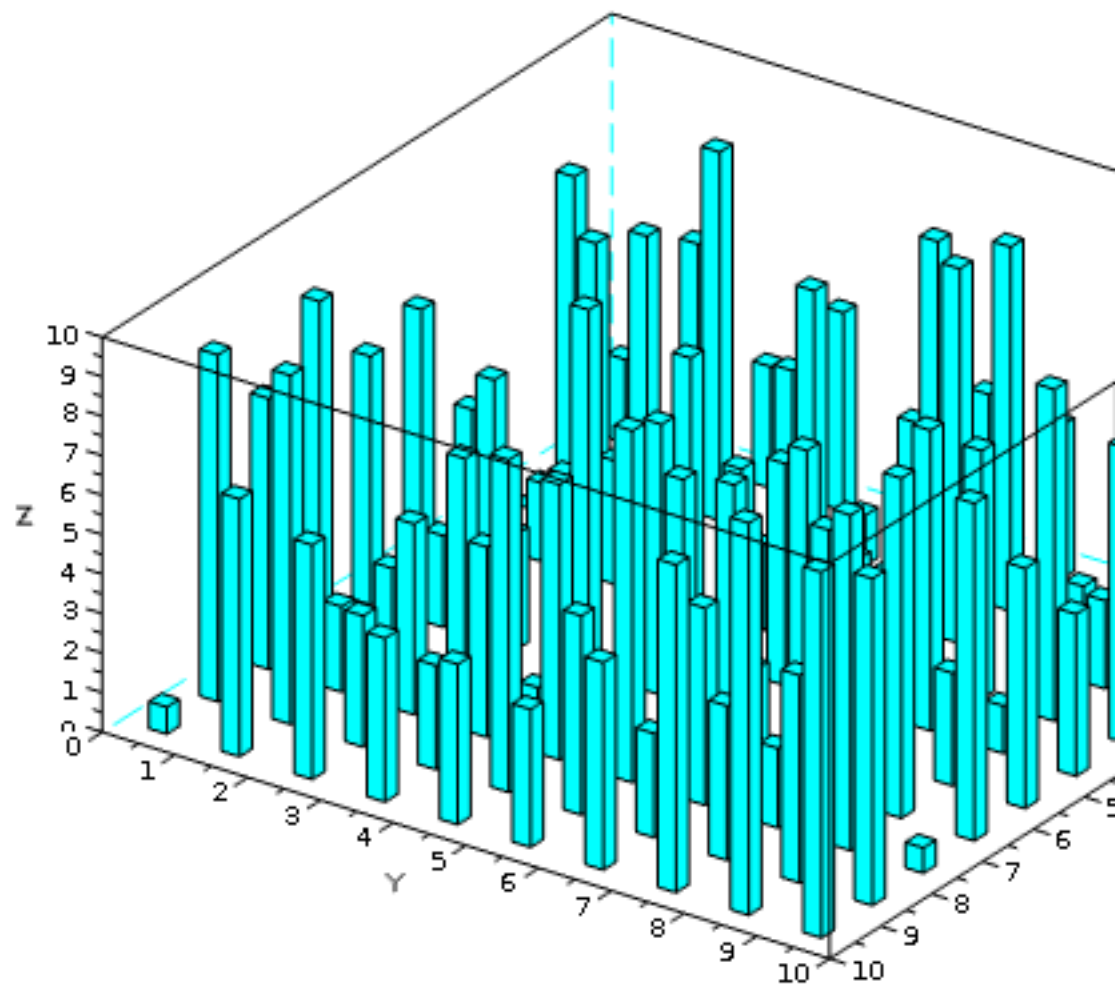
Level curves on a 3D surface defined by a function

Example:



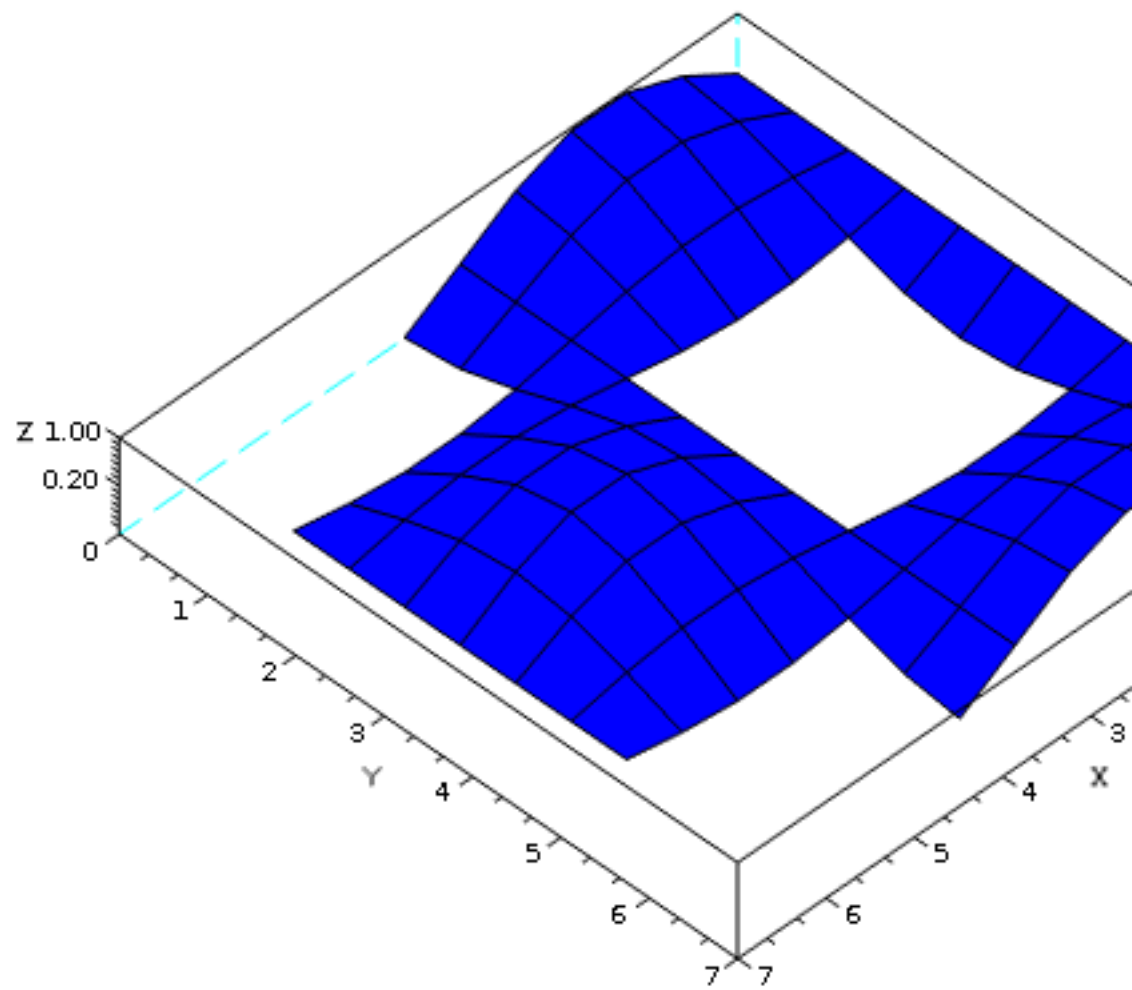
hist3d
3D representation of a histogram

Example:



genfac3d
Compute facets of a 3D surface

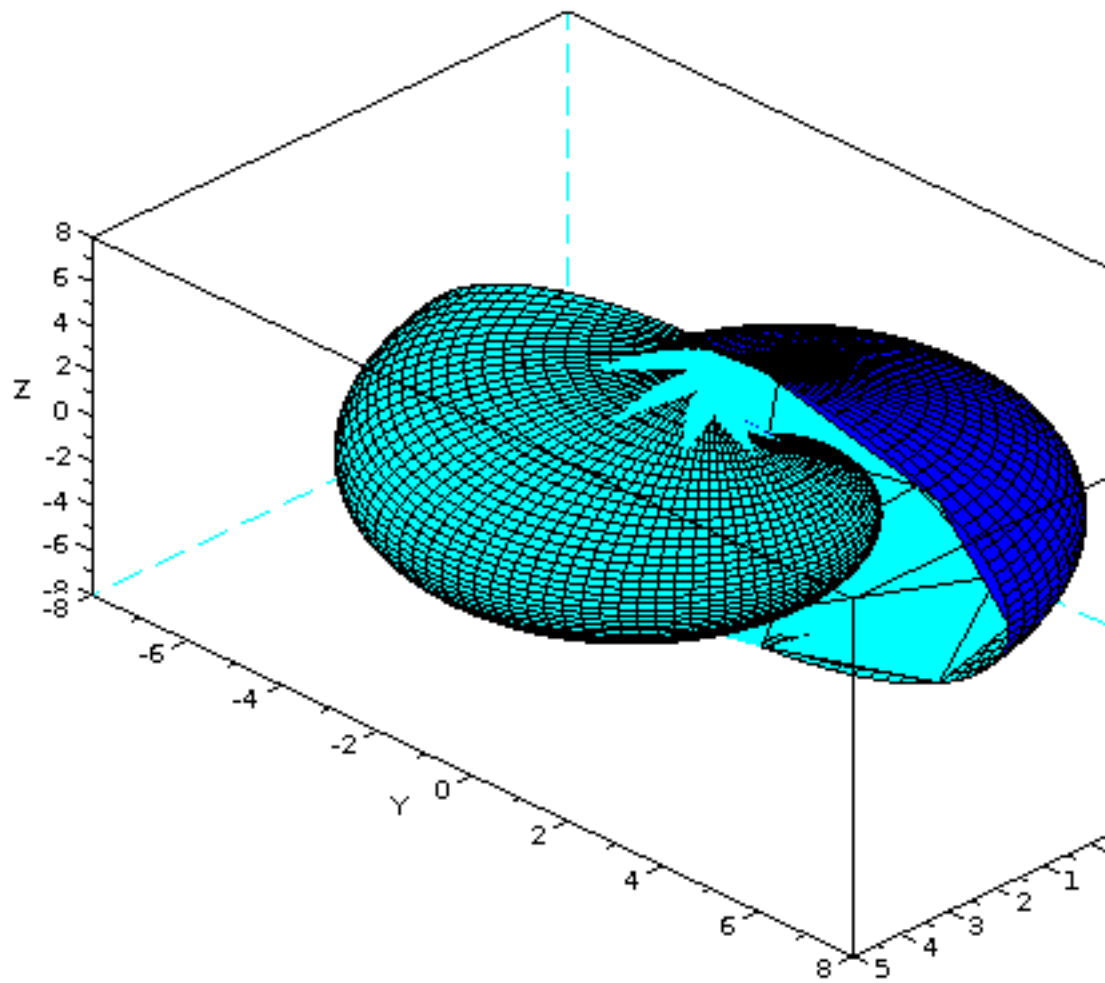
Example:



eval3dp

Compute facets of a 3D surface

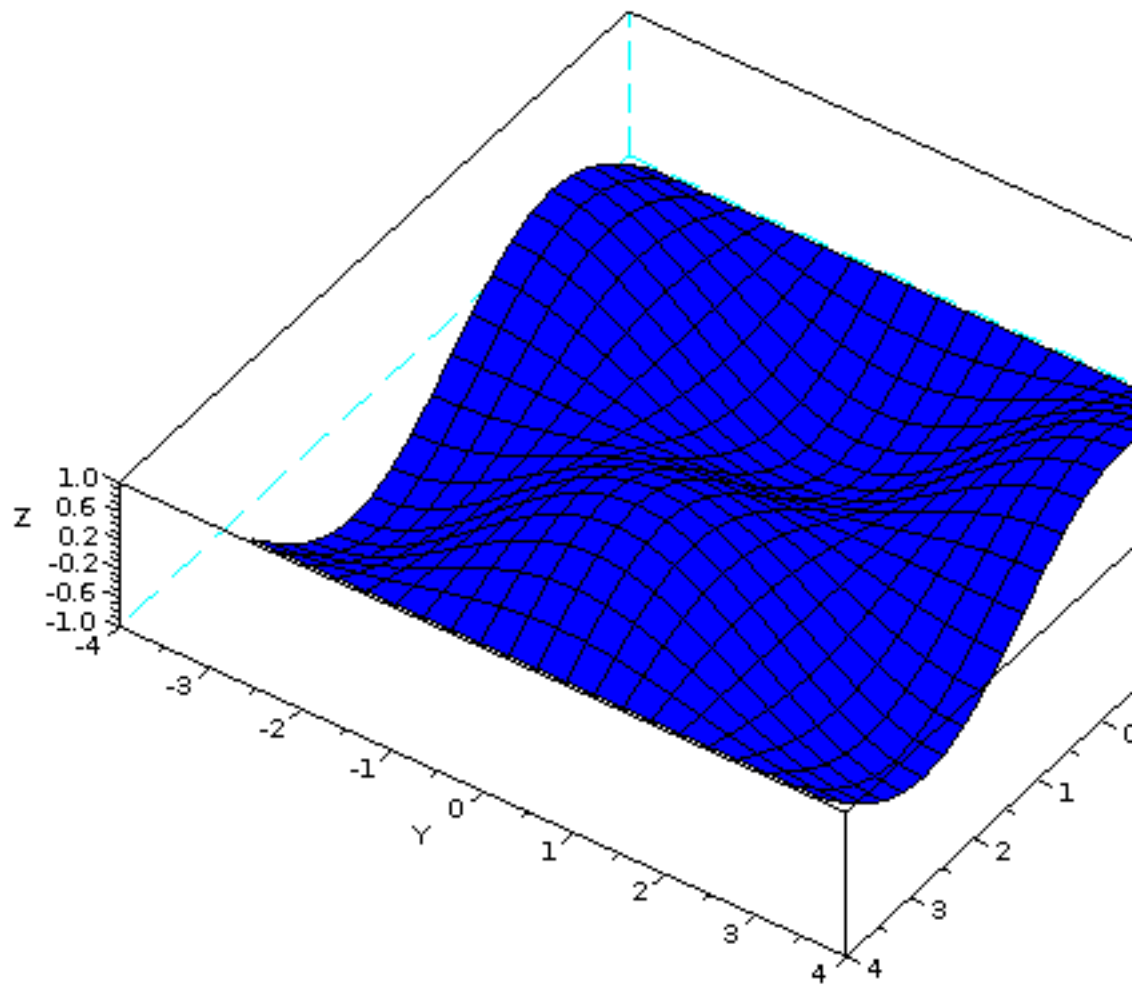
Example:



geom3d

Projection from 3D on 2D after a 3D plot

Example:



Line and polygon plotting

- `xpoly`
Draw a polyline or a polygon
- `xpolys`
Draw a set of polylines or polygons
- `xrpoly`
Draw a regular polygon
- `xsegs`
Draw unconnected segments
- `xfpoly`
Fill a polygon

xfpolys
Fill a set of polygons

Rectangle plotting

xrect
Draw a rectangle

xfrect
Fill a rectangle

xrects
Draw or fill a set of rectangles

Arc plotting

xarc
Draw a part of an ellipse

xarcs
Draw parts of a set of ellipses

xfarc
Fill a part of an ellipse

xfarcs
Fill parts of a set of ellipses

Arrow plotting

xarrows
Draw a set of arrows

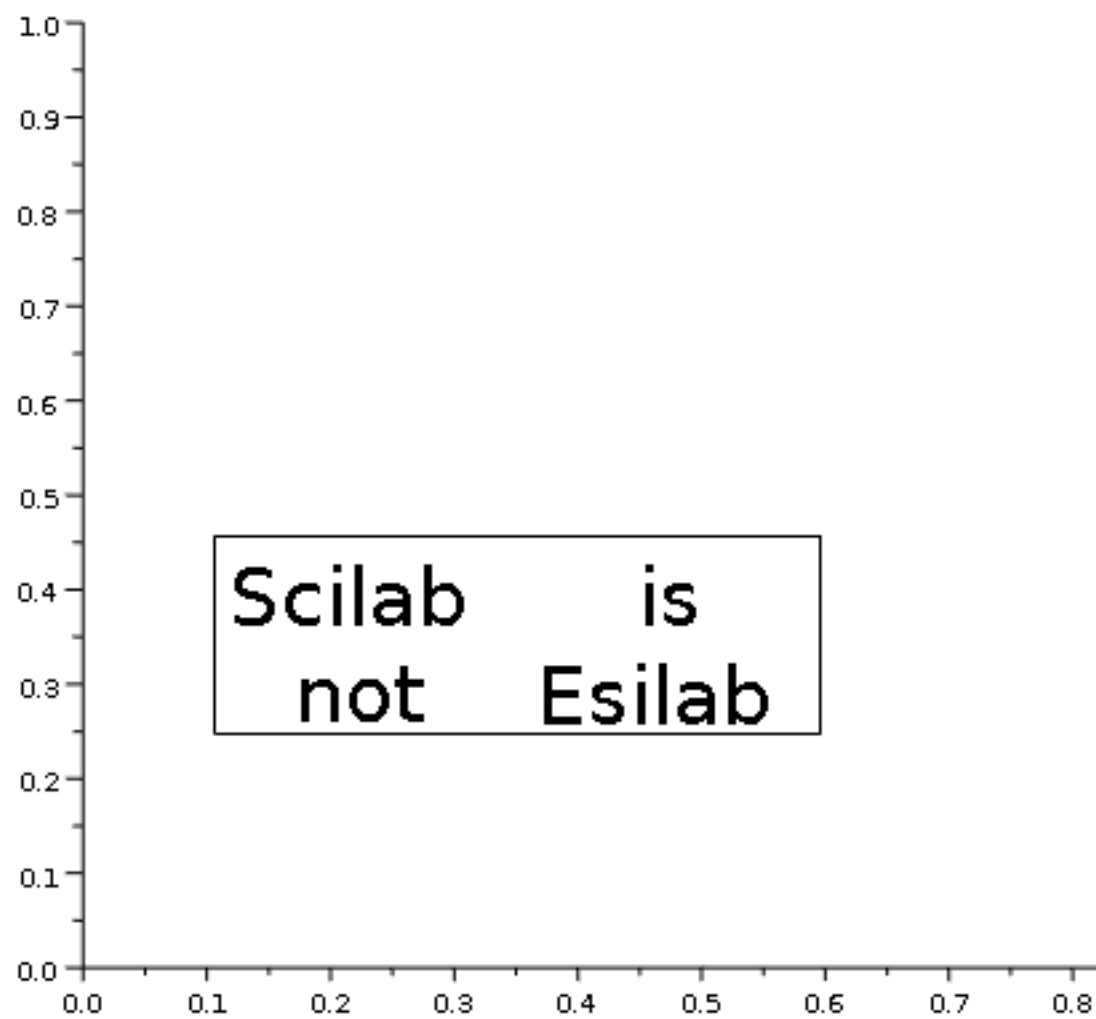
Strings

xstring
Draw strings

xstringl
Compute a box which surrounds strings

xstringb
Draw strings into a box

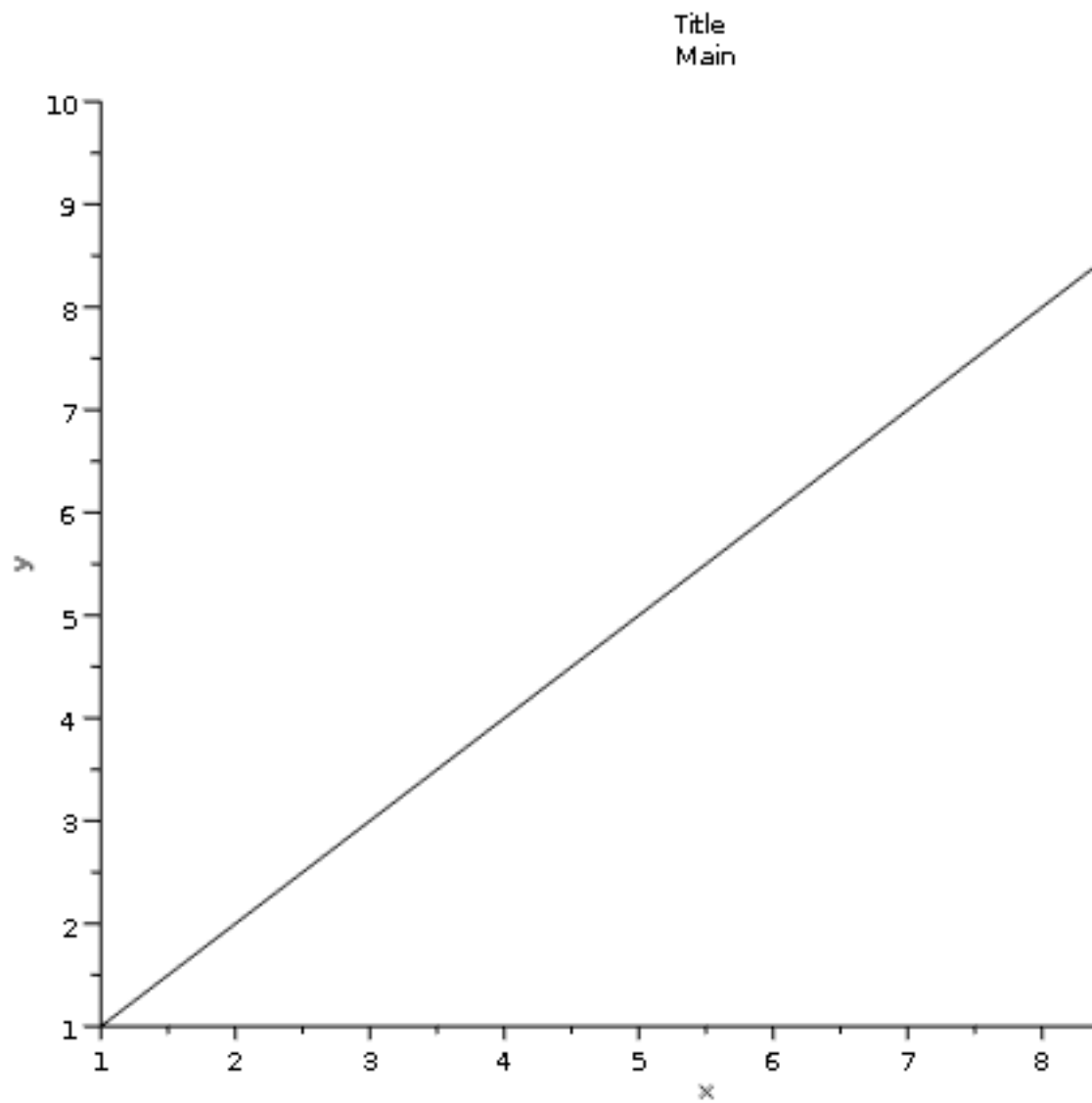
Example:



`xtitle`

Add titles on a graphics window

Example:



`titlepage`
Add a title in the middle of a graphics window

`xinfo`
Draw an info string in the message subwindow

Frames and axes

`drawaxis`
Draw an axis

`graduate`
Pretty axis graduations

`plotframe`
Plot a frame with scaling and grids

Coordinates transformations

- isoview
Set scales for isometric plot (do not change the size of the window)
- square
Set scales for isometric plot (change the size of the window)
- scaling
Affine transformation of a set of points
- rotate
Rotation of a set of points
- xsetech
Set the sub-window of a graphics window for plotting
- subplot
Divide a graphics window into a matrix of sub-windows
- xgetech
Get the current graphics scale
- xchange
Transform real to pixel coordinates

Colors

- colormap
Using colormaps
- getcolor
Dialog to select colors in the current colormap
- addcolor
Add new colors to the current colormap
- graycolormap
Linear gray colormap
- hotcolormap
Red to yellow colormap

Graphics context

- xset
Set values of the graphics context
- xget
Get current values of the graphics context
- xlfont
Load a font in the graphics context or query loaded font
- getsymbol
Dialog to select a symbol and its size

Save and load

- xsave
Save graphics into a file

xload
Load a saved graphics

xs2fig
Send graphics to a file in Xfig syntax

xs2gif
Send graphics to a file in Gif syntax

xs2ppm
Send graphics to a file in PPM syntax

Graphics primitives

xbasc
Clear a graphics window and erase the associated recorded graphics

xclear
Clear a graphics window

driver
Select a graphics driver

xinit
Initialisation of a graphics driver

xend
Close a graphics session

xbasr
Redraw a graphics window

replot
Redraw the current graphics window with new boundaries

xpause
Suspend Scilab

xselect
Raise the current graphics window

xdel
Delete a graphics window

winsid
Return the list of graphics windows

xname
Change the name of the current graphics window

Mouse position

xclick
Wait for a mouse click

locate
Mouse selection of a set of points

xgetmouse
Get the current position of the mouse

Interactive editor

`edit_curv`

Interactive graphics curve editor

`sd2sci`

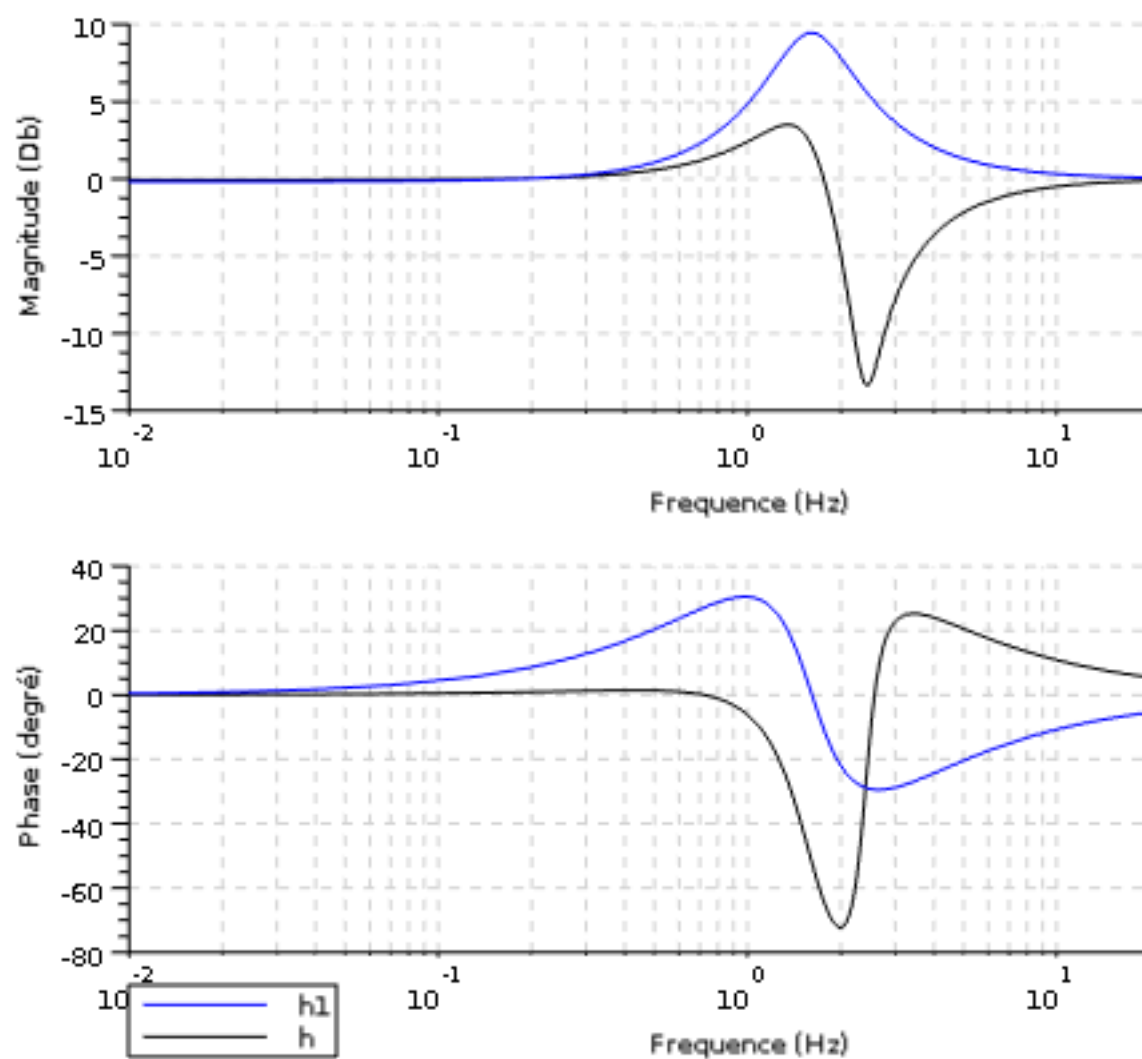
`gr_menu` structure to scilab instruction convertor

Graphics functions for automatic control

`bode`

Bode plot

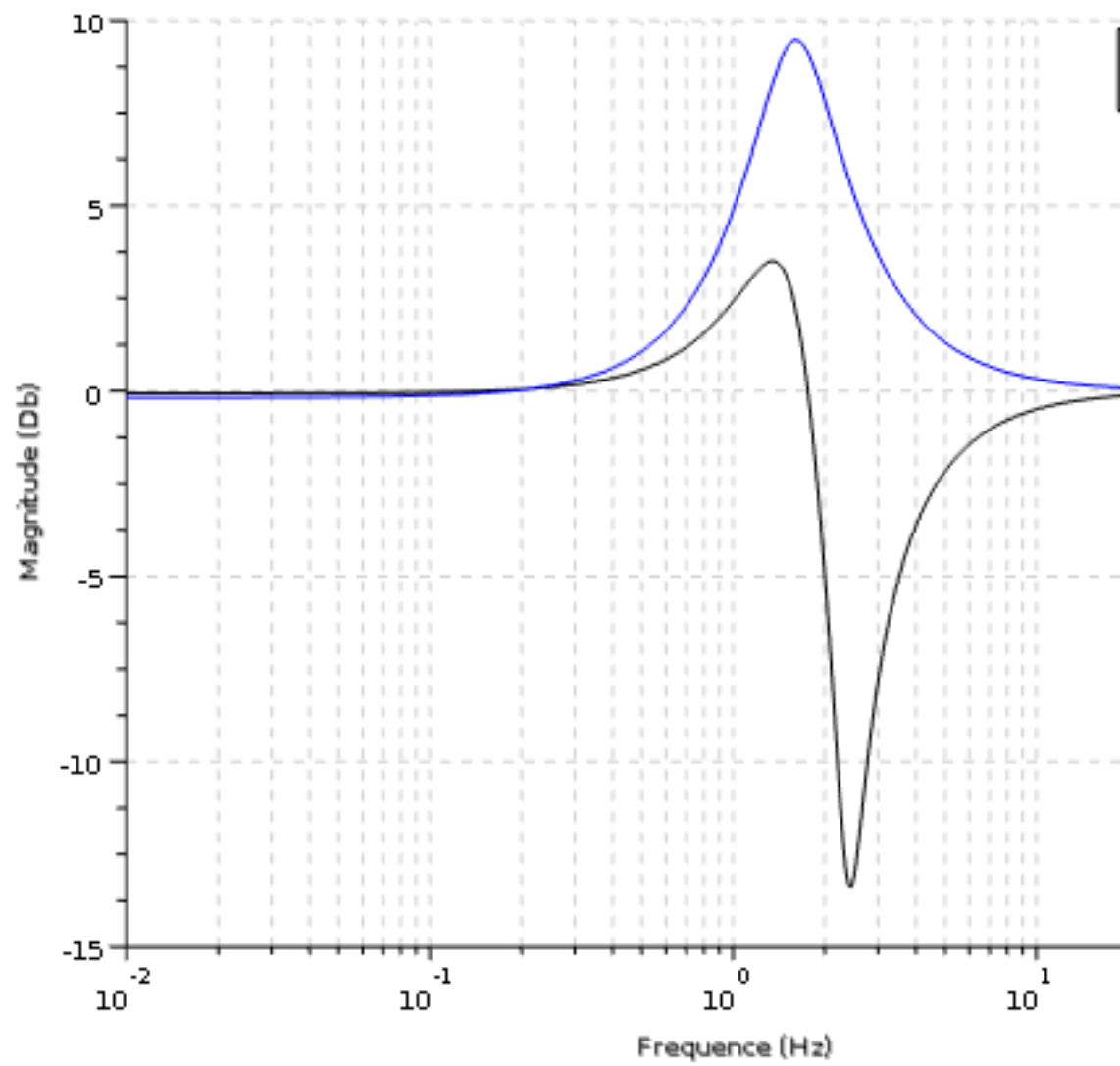
Example:



`gainplot`

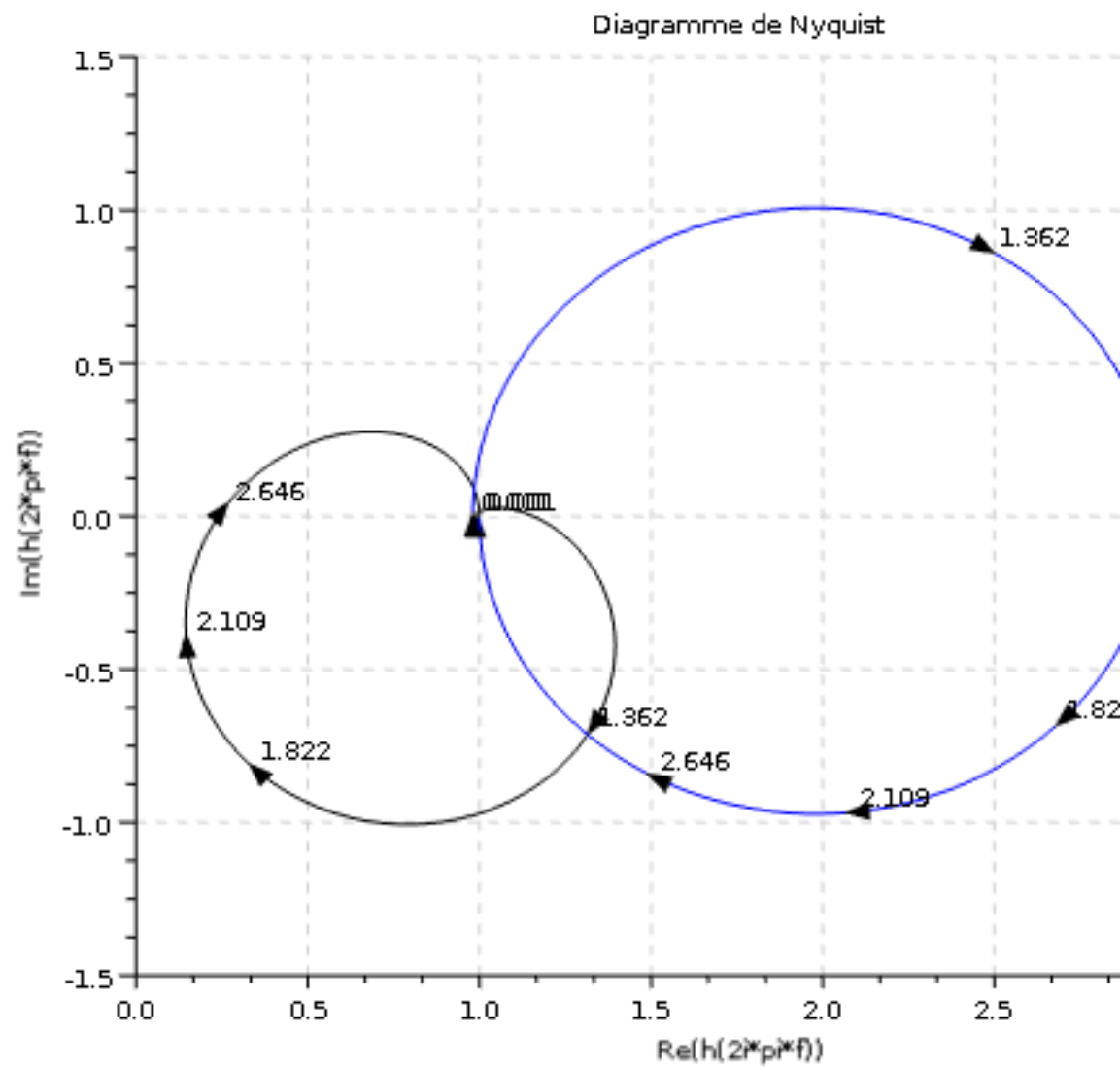
Magnitude plot

Example:



nyquist
Nyquist plot

Example:



m_circle
M-circle plot

Example:

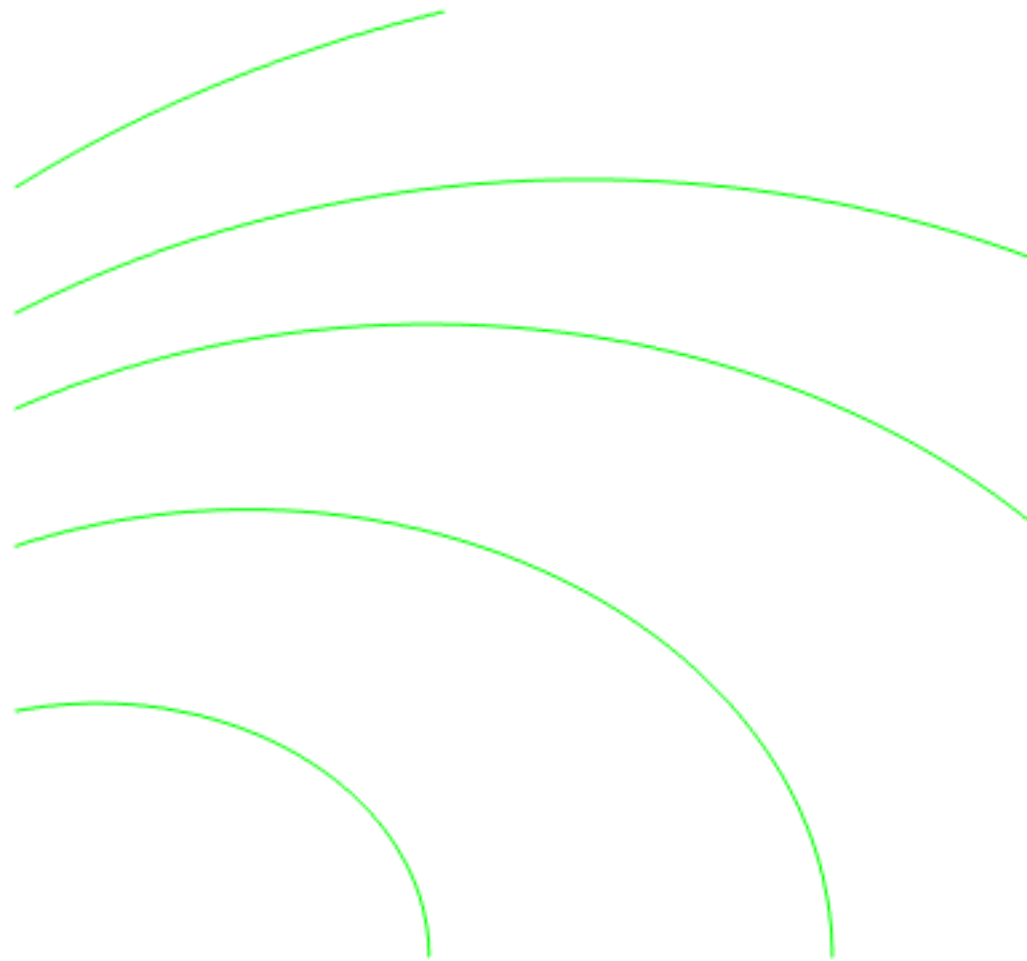
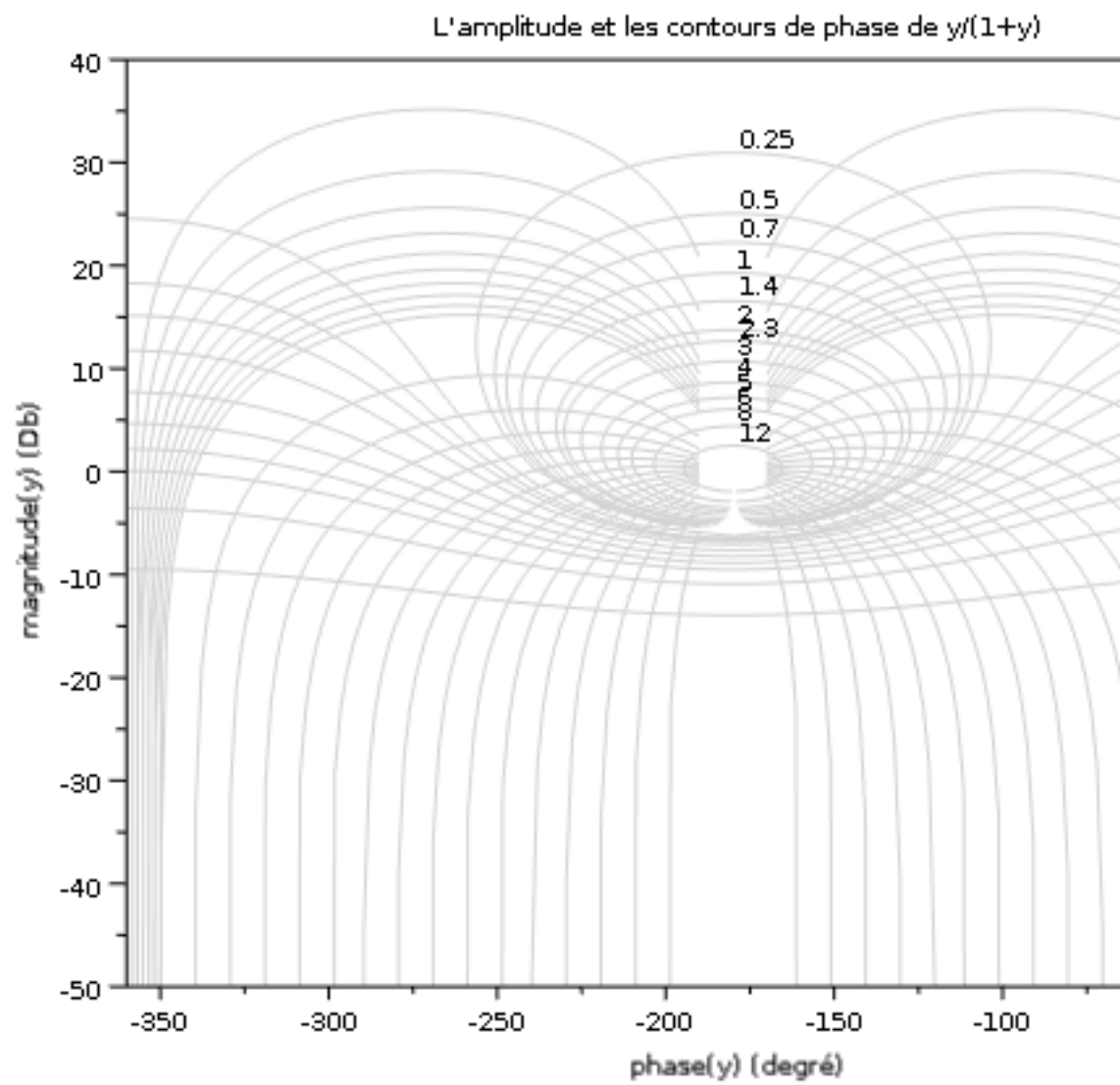


chart
Nichols chart

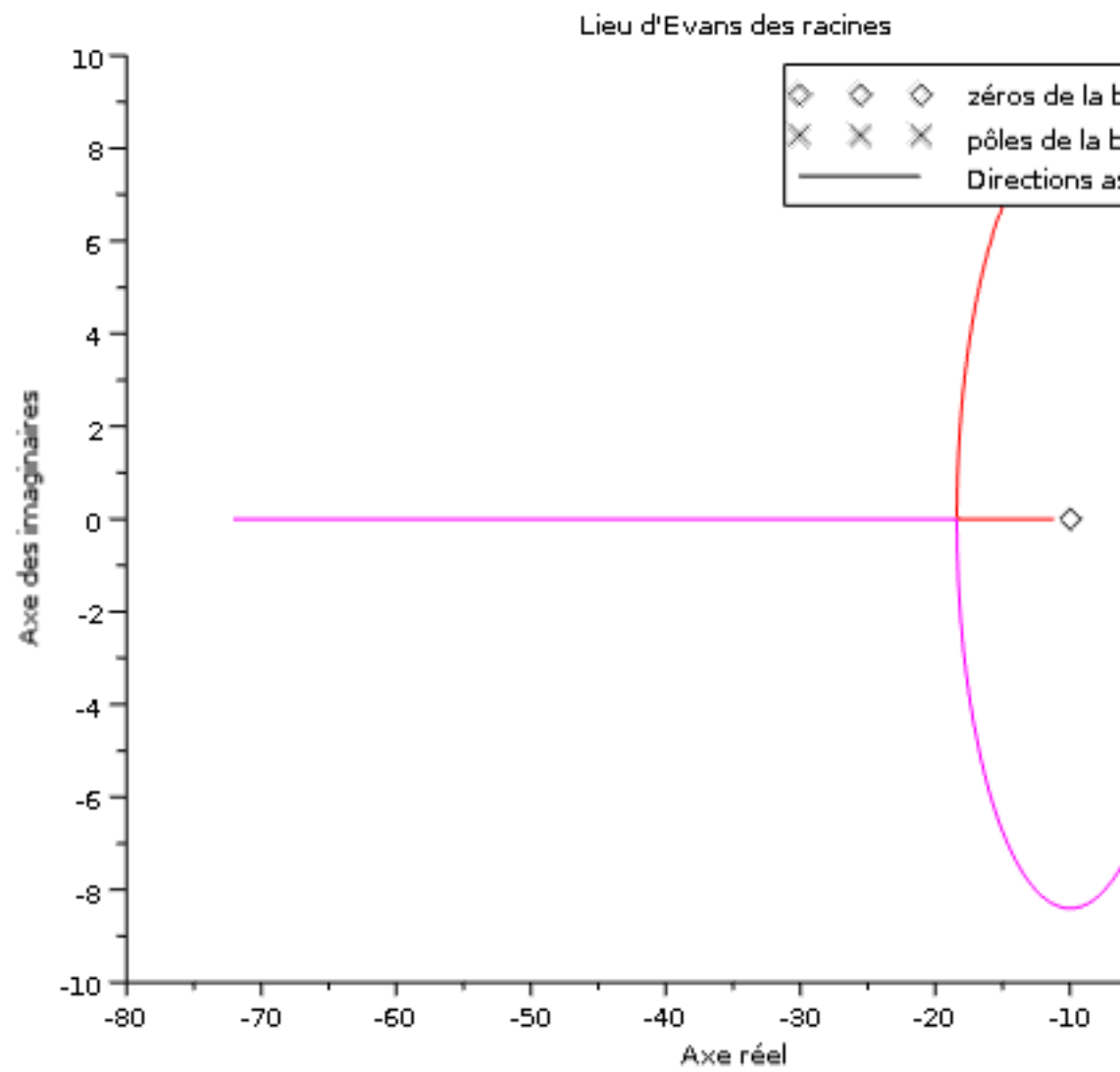
Example:



black

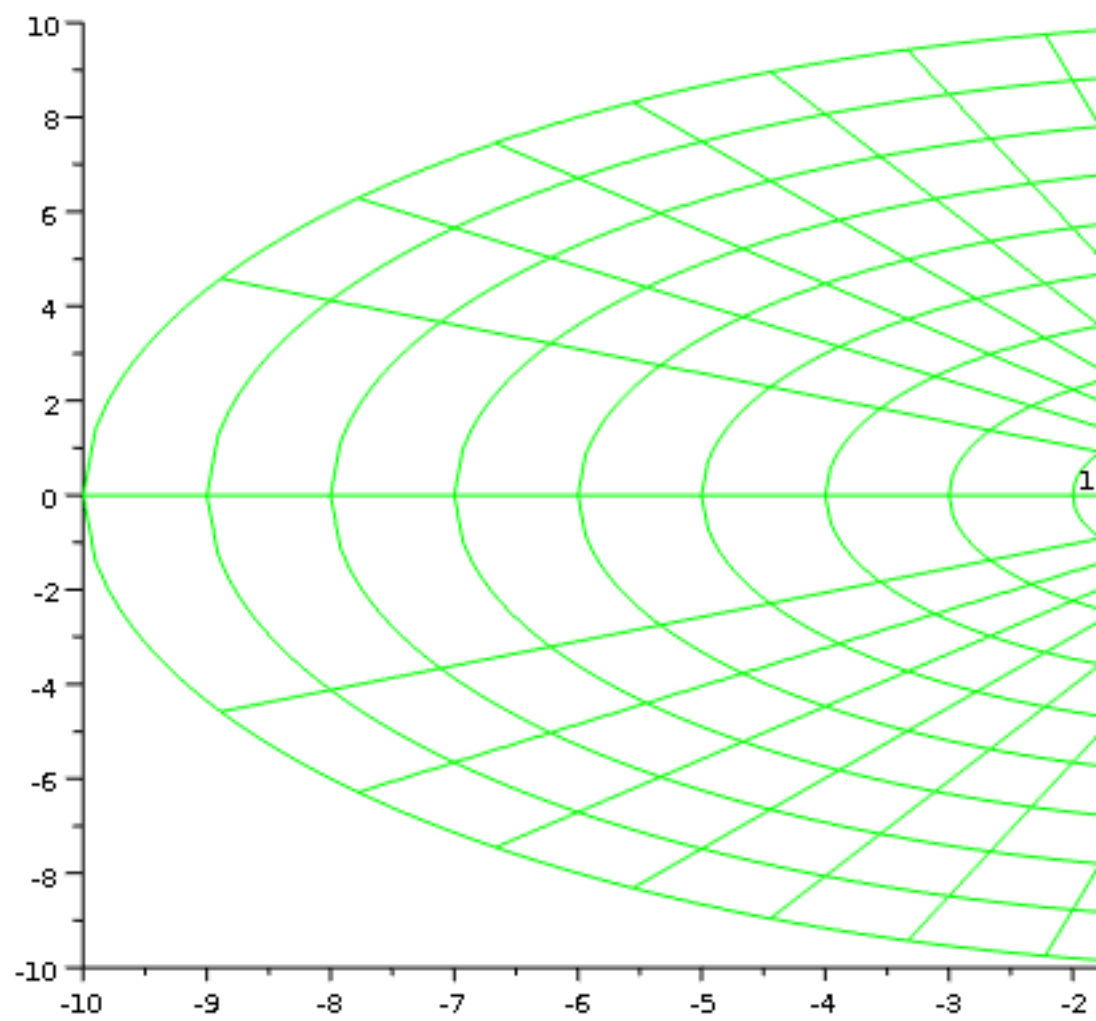
Black's diagram

Example:



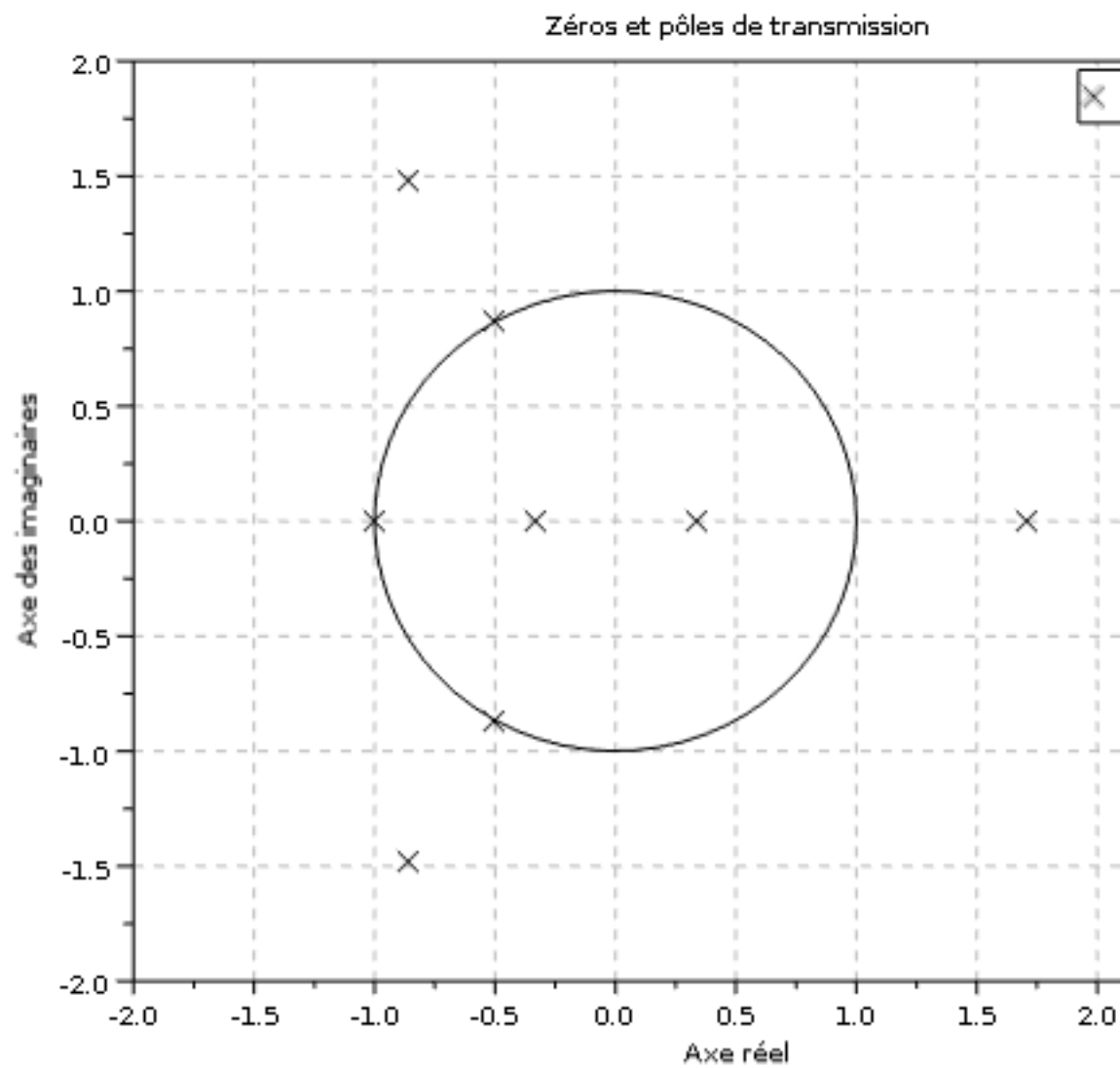
sgrid
s-plane grid lines

Example:



plzr
pole-zero plot

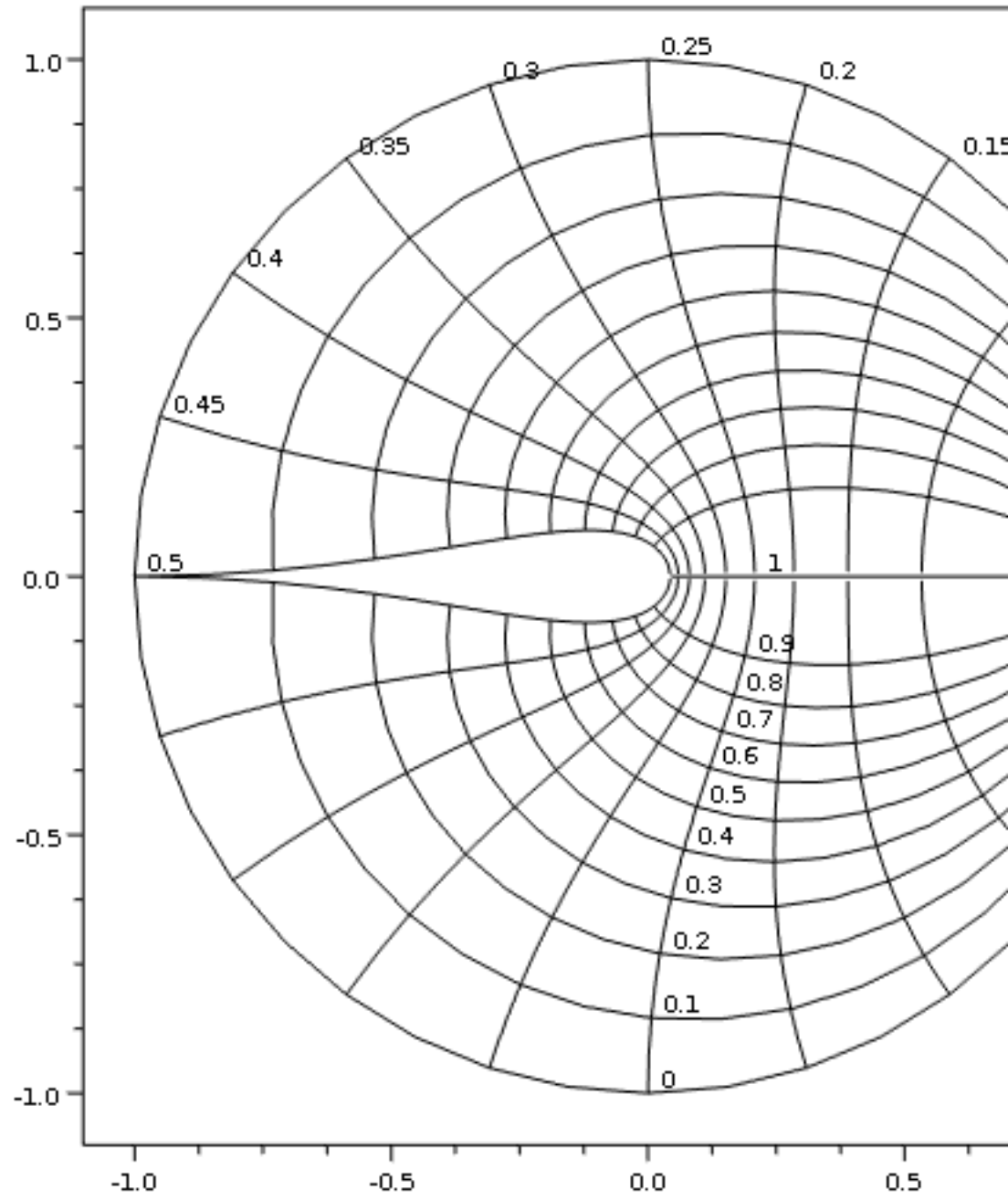
Example:



zgrid
zgrid plot

Example:

loci with constant damping and constant frequencies
in discrete plane



Name

LineStyle — to quickly customize the lines appearance in a plot

Description

The LineSpec is an optional argument that can be used inside a plot command to customize each new line aspect. It has to be given as a concatenated string containing information about color, line style or markers. It is very usefull to quickly specify such basic line properties.

To specify a red longdash-dot with diamond marker, the string can be 'r-.diam'. As you can see, a full complete spelling of each property value is not required but the string, which is a concatenation (in any order) of these three types of properties , must remain unambiguous. Furthermore, the string specification is not case sensitive.

Here is a complete list of the **LineStyle** types you can specify (using plot).

LineStyle:

a string defining the line style. This property is linked to the object's `line_style` property (see `polyline_properties`).

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-. .	Dash-dotted line

Color:

a string defining the line color. This property is linked to the object's `foreground` property (see `polyline_properties`).

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

A default color table is used to color plotted curves if you do not specify a color (neither with **LineStyle** nor with **GlobalProperty**). When drawing multiple lines, the plot command automatically cycles through this table. Here are the used colors:

R	G	B
0.	0.	1.
0.	0.5	0.
1.	0.	0.
0.	0.75	0.75

0.75	0.	0.75
0.75	0.75	0.
0.25	0.25	0.25

Marker type:

A string defining the marker type. note that if you specify a marker wihtout a line style, only the marker is drawn. This property is linked to the object's `mark_style` and `mark_mode` properties (see `polyline_properties`).

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram'	Five-pointed star (pentagram)
'none'	No marker (default)

Examples

```
x=1:0.1:10; // Init.
plot(x,sin(x),'r.->') // plots a dash-dotted line with a right-pointing triangle
clf();

// If you specify a marker wihtout a line style, only the marker is drawn
plot(x,sin(x),'d') // plots a dash-dotted line with a right-pointing triangle c

x=1:10; // Init.
// combinations' order does not matter
plot(x,x.*x,'*cya--')

//multiple plots with different LineSpecs
clf();
t=0:%pi/20:2*pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
```

See Also

GlobalProperty , plot , clf

Authors

F.Leray

Name

Matplot — 2D plot of a matrix using colors

```
Matplot(a, [strf, rect, nax])  
Matplot(a, <opt_args>)
```

Parameters

a

real matrix of size (n1,n2).

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following:

rect

sets the bounds of the plot. If this key is given and neither `frameflag` nor `strf` is given then the `y` character of `strf` is supposed to be 7. See below for value.

nax

sets the grids definition. If this key is given and neither `axesflag` nor `strf` is given then the `z` character of `strf` is supposed to be 1. See below for value.

frameflag

specifies how the frame of the plot is computed. The value is an integer ranging from 0 to 8. It corresponds to the `y` character of `strf`. See below.

axesflag

specifies what kind of axes are drawn around the plot. The value is an integer ranging from 0 to 5. It corresponds to the `z` character of `strf`. See below.

strf

is a string of length 3 "xyz".

default

The default is "081".

x

controls the display of captions.

x=0

no caption.

x=1

captions are displayed. They are given by the optional argument `leg`.

y

controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

y=0

no computation, the plot use the previus (or default) scale

y=1

from the rect arg

y=2

from the min/max of the x, y datas

- y=3
built for an isometric scale from the rect arg
- y=4
built for an isometric plot from the min/max of the x, y datas
- y=5
enlarged for pretty axes from the rect arg
- y=6
enlarged for pretty axes from the min/max of the x, y datas
- y=7
like y=1 but the previous plot(s) are redrawn to use the new scale
- y=8
like y=2 but the previous plot(s) are redrawn to use the new scale

z
controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the **nax** optional argument.

- z=0
nothing is drawn around the plot.
- z=1
axes are drawn, the y=axis is displayed on the left.
- z=2
the plot is surrounded by a box without tics.
- z=3
axes are drawn, the y=axis is displayed on the right.
- z=4
axes are drawn centred in the middle of the frame box, with the box disabled.
- z=5
axes are drawn centred in the middle of the frame box, with the box enabled.

rect
This argument is used when the second character y of argument **strf** is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: **rect**=[xmin,ymin,xmax,ymax].

nax
This argument is used when the third character z of argument **strf** is 1. It is a row vector with four entries [nx,Nx,ny,Ny] where nx (ny) is the number of subgraduations on the x (y) axis and Nx (Ny) is the number of graduations on the x (y) axis.

Description

The entries of matrix **int(a)** are used as colormap entries in the current colormap. The color associated to **a(i,j)** is used to draw a small square of size 1 with center at location $(x=j, y=(n1-i+1))$. If a matrix entry is outside the colormap, the corresponding rectangle is not displayed.

Enter the command **Matplot()** to see a demo.

Examples

```
Matplot([1 2 3;4 5 6])  
clf()  
// draw the current colormap  
Matplot((1:xget("lastpattern")))
```

See Also

[colormap](#) , [plot2d](#) , [Matplot1](#) , [Matplot_properties](#)

Authors

J.Ph.C.

Name

Matplot1 — 2D plot of a matrix using colors

```
Matplot1(a, rect)
```

Parameters

a
real matrix of size (n1,n2).

rect
[xmin,ymin,xmax,ymax]

Description

The entries of matrix `int(a)` are used as colormap entries in the current colormap. `rect` specify a rectangle in the current scale and the matrix is drawn inside this rectangle. Each matrix entry will be rendered as a small rectangle filled with its associated color. If a matrix entry is outside the colormap, the corresponding rectangle is not displayed.

Examples

```
//--- first example
clf();
ax=gca();//get current axes handle
ax.data_bounds=[0,0;10,10];//set the data_bounds
ax.box='on'; //draw a box
a=5*ones(11,11); a(2:10,2:10)=4; a(5:7,5:7)=2;
// first matrix in rectangle [1,1,3,3]
Matplot1(a,[1,1,3,3])
a=ones(10,10); a= 3*tril(a)+ 2*a;
// second matrix in rectangle [5,6,7,8]
Matplot1(a,[5,6,7,8])

//--- second example (animation)
n=100;

clf();
f=gcf();//get current figure handle
f.pixmap='on';//double buffer mode
ax=gca();//get current axes handle
ax.data_bounds=[0,0;10,10];//set the data_bounds
ax.box='on'; //draw a box
show_pixmap()
for k=-n:n,
    a=ones(n,n);
    a= 3*tril(a,k)+ 2*a;
    a= a + a';
    k1= 3*(k+100)/200;
    if k>-n then delete(gcf()),end
    Matplot1(a,[k1,2,k1+7,9])
    show_pixmap() //send double buffer to screen
end
```

See Also

[colormap](#) , [plot2d](#) , [Matplot](#) , [grayplot](#) , [Matplot_properties](#)

Authors

J.Ph.C.

Name

Matplot_properties — description of the Matplot entities properties

Description

The Matplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images (see `Matplot` and `Matplot1`).

parent:

This property contains the handle of the parent. The parent of the Matplot entity should be of the type "Axes".

children:

This property contains a vector with the `children` of the handle. However, `Matplot` handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity . It should be "on" or "off" . By default, the plot is visible, the value's property is "on". If "off" the plot is not drawn on the screen.

data:

This field defines a `[m×n] color` data matrix using the current colormap. The color associated to `color(i,j)` is used to draw a small square of length 1 with center at location $(x=j, y=(m-i+1))$.

clip_state:

This field contains the `clip_state` property value for the Matplot. It should be :

- "off" this means that the Matplot is not clipped.
- "clipgrf" this means that the Matplot is clipped outside the Axes box.
- "on" this means that the Matplot is clipped outside the rectangle given by property `clip_box`.

clip_box:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the Matplot to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the Matplot data structure, and to retrieve it.

Examples

```
Matplot((1:xget("lastpattern")))
e=gce(); // get current entity

e.data=e.data($:-1:1) // reverse order
```

See Also

`set` , `get` , `delete` , `grayplot` , `Matplot` , `Matplot1` , `graphics_entities` , `grayplot_properties`

Authors

F.Leray

Name

Sfgrayplot — smooth 2D plot of a surface defined by a function using colors

```
Sfgrayplot(x,y,f,<opt_args>)  
Sfgrayplot(x,y,f [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

Parameters

x,y

real row vectors of size n1 and n2.

f

scilab function ($z=f(x,y)$)

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `strf, rect, nax, zminmax, colminmax, mesh, colout` (see `plot2d` for the 3 first and `fec` for the 4 last).

strf,rect,nax

see `plot2d`.

zminmax, colminmax, mesh, colout

see `fec`.

Description

`Sfgrayplot` is the same as `fgrayplot` but the plot is smoothed. The function `fec` is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid (here with $n1=5, n2=3$):

```
| / | / | / | / |  
| / _ | / _ | / _ | / _ |  
| / | / | / | / |  
| / _ | / _ | / _ | / _ |
```

The function `colorbar` may be used to see the color scale (but you must know (or compute) the min and max values).

Instead of `Sfgrayplot`, you can use `Sgrayplot` and this may be a little faster.

Enter the command `Sfgrayplot( )` to see a demo.

Examples

```
// example #1: plot 4 surfaces  
function z=surf1(x,y), z=x*y, endfunction  
function z=surf2(x,y), z=x^2-y^2, endfunction  
function z=surf3(x,y), z=x^3+y^2, endfunction  
function z=surf4(x,y), z=x^2+y^2, endfunction  
clf()  
xset("colormap",[jetcolormap(64);hotcolormap(64)])  
x = linspace(-1,1,60);  
y = linspace(-1,1,60);
```

```
drawlater() ;
subplot(2,2,1)
    colorbar(-1,1,[1,64])
    Sfgrayplot(x,y,surf1,strf="041",colminmax=[1,64])
    xtitle("f(x,y) = x*y")
subplot(2,2,2)
    colorbar(-1,1,[65,128])
    Sfgrayplot(x,y,surf2,strf="041",colminmax=[65,128])
    xtitle("f(x,y) = x^2-y^2")
subplot(2,2,3)
    colorbar(-1,2,[65,128])
    Sfgrayplot(x,y,surf3,strf="041",colminmax=[65,128])
    xtitle("f(x,y) = x^3+y^2")
subplot(2,2,4)
    colorbar(0,2,[1,64])
    Sfgrayplot(x,y,surf4,strf="041",colminmax=[1,64])
    xtitle("f(x,y) = x^2+y^2")
drawnow() ;
xselect()

// example #2: plot surf3 and add some contour lines
function z=surf3(x,y), z=x^3+y^2, endfunction
clf()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",hotcolormap(128))
drawlater() ;
colorbar(-1,2)
Sfgrayplot(x,y,surf3,strf="041")
fcontour2d(x,y,surf3,[-0.1, 0.025, 0.4],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2")
drawnow() ;
xselect()

// example #3: plot surf3 and use zminmax and colout optional arguments
//             to restrict the plot for -0.5<= z <= 1
function z=surf3(x,y), z=x^3+y^2, endfunction
clf()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",jetcolormap(128))
drawlater() ;
zminmax = [-0.5 1]; colors=[32 96];
colorbar(zminmax(1),zminmax(2),colors)
Sfgrayplot(x, y, surf3, strf="041", zminmax=zminmax, colout=[0 0], colminmax=co
fcontour2d(x,y,surf3,[-0.5, 1],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2, with parts under z = -0.5 and upper z = 1 removed")
drawnow() ;
xselect()
```

See Also

fec , fgrayplot , grayplot , Sgrayplot

Authors

J.Ph.C.

Name

Sgrayplot — smooth 2D plot of a surface using colors

```
Sgrayplot(x,y,z,<opt_args>)  
Sgrayplot(x,y,z [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

Parameters

x,y

real row vectors of size n1 and n2.

z

real matrix of size (n1,n2). $z(i,j)$ is the value of the surface at the point $(x(i),y(j))$.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `strf, rect, nax, zminmax, colminmax, mesh, colout`.

strf

is a string of length 3 "xyz" (by default `strf= "081"`)

x

controls the display of captions.

x=0

no caption.

x=1

captions are displayed. They are given by the optional argument `leg`.

y

controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

y=0

no computation, the plot use the previus (or default) scale

y=1

from the `rect` arg

y=2

from the min/max of the x, y datas

y=3

built for an isometric scale from the `rect` arg

y=4

built for an isometric plot from the min/max of the x, y datas

y=5

enlarged for pretty axes from the `rect` arg

y=6

enlarged for pretty axes from the min/max of the x, y datas

y=7

like `y=1` but the previus plot(s) are redrawn to use the new scale

y=8

like `y=2` but the previus plot(s) are redrawn to use the new scale

z
controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the **nax** optional argument.

z=0
nothing is drawn around the plot.

z=1
axes are drawn, the y=axis is displayed on the left.

z=2
the plot is surrounded by a box without tics.

z=3
axes are drawn, the y=axis is displayed on the right.

z=4
axes are drawn centred in the middle of the frame box, with the box disabled.

z=5
axes are drawn centred in the middle of the frame box, with the box enabled.

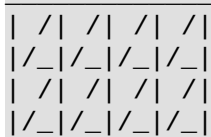
rect
This argument is used when the second character **y** of argument **strf** is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: **rect**=[**xmin**,**ymin**,**xmax**,**ymax**].

nax
This argument is used when the third character **z** of argument **strf** is 1. It is a row vector with four entries [**nx**,**Nx**,**ny**,**Ny**] where **nx** (**ny**) is the number of subgraduations on the x (y) axis and **Nx** (**Ny**) is the number of graduations on the x (y) axis.

zminmax, **colminmax**, **mesh**, **colout**
See **fec**.

Description

Sgrayplot is the same as **grayplot** but the plot is smoothed. The function **fec** is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid (here with **n1=5**, **n2=3**):



The function **colorbar** may be used to see the color scale.

The parameter **zminmax** is useful for animation purpose (see an example after) and the parameter **colminmax** lets the user choose a part of the current colormap (see the **fec** help page).

Enter the command **Sgrayplot ()** to see a demo.

Examples

```
// example #1
x=-10:10; y=-10:10;m =rand(21,21);
```

```
clf()
xset("colormap",hotcolormap(64))
Sgrayplot(x,y,m, strf="011", rect=[-20,-20,20,20])

// example #2
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
clf()
xset("colormap",jetcolormap(64))
colorbar(-1,1)
Sgrayplot(t,t,m, strf="041")

// example #3: an animation display cos(t)*sin(x)sin(y).
n = 30;
nt = 100;
x = linspace(0,2*%pi,n);
y = linspace(0,%pi,n/2);
z = sin(x')*sin(y);
t = linspace(0,4*%pi,nt);
xselect(); clf()
f=gcf();
f.color_map=jetcolormap(64);
f.pixmap='on';
colorbar(-1,1)
Sgrayplot(x,y,cos(t(1))*z, strf="042", zminmax=[-1,1])
c=gce(),e=c.children
xtitle("Kaa's eyes")
for i = 1:nt
    e.data(:,3)=matrix(cos(t(i))*z,-1,1);
    show_pixmap()
end
f.pixmap='off';
```

See Also

`fec` , `fgrayplot` , `grayplot` , `Sfgrayplot` , `colorbar`

Authors

J.Ph.C.

Name

`addcolor` — add new colors to the current colormap

```
new=addcolor(c)
```

Parameters

`new`

ids of the colors defined in `c` in the new color table.

`c`

matrix with 3 columns, RGB color definition.

Description

`addcolor` adds new colors given in the `c` argument to the current colormap. `c` must be a matrix with 3 columns [`R` `G` `B`] `R` is red component, `G` is green component, `B` is blue component). Each entry in `c` must be a non negative number less or equal to 1.

The ids of the new colors are returned into `new`.

If a color defined in `c` is already present in the current colormap it is not added.

Example

```
plot3d();  
h = gcf();  
h.color_map = jetcolormap(16);  
addcolor(name2rgb('grey')/255);
```

See Also

`colormap`, `name2rgb`

Name

alufunctions — pixel drawing functions

Description

src is the source ie the "value of the pixel" which we want to draw. dst is the destination ie "value of the pixel" which is already drawn.

- 0
clear ie "0"
- 1
and ie "src AND dst"
- 2
and reverse ie "src AND NOT dst"
- 3
copy ie "src"
- 4
and inverted ie "(NOT src) AND dst"
- 5
noop ie "dst"
- 6
xor ie "src XOR dst"
- 7
or ie "src OR dst"
- 8
nor ie "(NOT src) AND (NOT dst)"
- 9
equiv ie "(NOT src) XOR dst"
- 10
invert ie "NOT dst"
- 11
or reverse ie "src OR (NOT dst)"
- 12
copy inverted ie "NOT src"
- 13
or inverted ie "(NOT src) OR dst"
- 14
nand ie "(NOT src) OR (NOT dst)"
- 15
set ie "1"

Name

arc_properties — description of the Arc entity properties

Description

The Arc entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for ellipses and part of ellipses and the filled ones.

parent:

This field contains the handle of the parent. The parent of the arc entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the children of the handle. However, arc handles currently do not have any children.

thickness:

This property is a positive real specifying the line width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole thickness value is considered.

line_style:

The line_style property value should be an integer in [0 8]. 0 and 1 stands for solid, the other value stands for a selection of dashes (see getlinestyle).

line_mode:

This property allows to display or not the line representing the arc. The value must be "on" or "off".

fill_mode:

If fill_mode property value is "on" , the arc is filled with the background color.

foreground:

This field contains the default foreground property value used to draw the outside of the arc. It should be a color index (relative to the current colormap).

background:

This field contains the color used to fill the arc. It should be a color index (relative to the current colormap).

data:

This property is to return the coordinates of the upper-left point, the width and the height of the inclosing rectangle as well as the boundary angles of the sector. It is the matrix in user coordinates [xleft, yup, [zup], width, height, a1, a2] where a1 and a2 are the sector boundary angles in degree.

Warning: in Scilab versions up to 4.1.2 a1 and a2 were given in degree/64.

visible:

This field contains the visible property value for the entity . It should be "on" or "off". If "on" the arc is drawn, If "off" the arc is not displayed on the screen.

arc_drawing_method:

This field controls the kind of discretisation used to render the arc. Its value must be either "nurbs" or "lines". If "nurbs" is selected then the arc is rendered using nurbs curves and surfaces. This results in the display of a perfect ellipse part whatever the view point is. If "lines" is selected then the arc is approximated with a constant number of lines. This reduce drawing time but some sharp edges may appear upon zooming. The use of "lines" value is discouraged and should only be used if a loss in framerate is noticed when using "nurbs" value.

clip_state:

This field contains the `clip_state` property value for the arc. Clip_state value should be :

- "off" this means that the arc is not clipped
- "clipgrf" this means that the arc is clipped outside the Axes box.
- "on" this means that the arc is clipped outside the arc given by property clip_box.

clip_box:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the arc to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the arc data structure, and to retrieve it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xarc(-1.5,1.5,3,3,0,360*64)

arc=get("hdl");//get handle on current entity (here the arc entity)
arc.fill_mode="on";
arc.foreground=5;
arc.data(:,[3 6])=[2 270*64];
xfarc(-.5,1,.4,.6,0,360*64);
arc.visible="off";
```

See Also

set, get, delete, xarc, xarcs, xfarc, xfarc, graphics_entities

Authors

Djalel ABDEMOUCHE

Jean-Baptiste SILVY

Name

autumncolormap — red through orange to yellow colormap

```
cmap=autumncolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

autumncolormap computes a colormap with n colors varying from red through orange to yellow.

Examples

```
f = scf();  
plot3d1();  
f.color_map = autumncolormap(32);
```

See Also

colormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

axes_properties — description of the axes entity properties

Description

The Axes entity is the second level of the graphics entities hierarchy. This entity defines the parameters allowing the change of coordinates and the axes drawing as well as the parameters' default values for the children creation.

Axes properties

parent:

This field contains the handle of the parent figure.

children:

FA vector containing the handles of all graphics objects children of the axes. These graphics objects are of type "Compound", "Rectangle", "Polyline", "Segs", "Arc", "Grayplot",... (see Compound_properties, rectangle_properties, champ_properties, axis_properties, polyline_properties, segs_properties, grayplot_properties, surface_properties, fec_properties, text_properties, legend_properties)

visible:

This field contains the `visible` property value for axes. Its value should be "on" or "off". By default, axes is visible "on" in case all "visible" children are displayed on the screen, If "off" the axes and all its children are not drawn.

axes_visible:

A 1x3 string vector. This property specifies whether each axis is drawn or not. Its value should be "on" or "off" for a global setting. To act on a single axis, the syntax is `axes_visible(N)` where N is 1, 2 or 3 corresponding to the x, y or z axis. The scaling data and if required the grids are drawn if the value is "on". Note that when creating a simple axes entity using the `gca()` (shortcut for `get('current_axes')`) or `gcf()` (shortcut for `get('current_figure')`) commands, the axes visibility is set to "off".

axes_reverse:

A 1x3 string vector corresponding to the three axes (X,Y,Z). For each axis, the property specifies the direction of the increasing values. If "off", the default direction is used. If "on", the direction is reverse. It is also possible to use only one string, "on" or "off", to set simultaneously the three data.

grid:

The field value is a vector [`x-grid`, `y-grid`, `z-grid`] where `x-grid` controls a grid drawing for the x-axis and `y-grid`, `z-grid` respecting to the y-axis, z-axis. The default values are -1 grids are not drawn, else the grids are drawn using the color given indexed by the grid value.

grid_position:

This character string specifies the grid position compared with other graphic entities. Its value can be either "foreground" to draw the grid ahead other graphic entities or "background" to draw the grid behind.

x_location:

Specify the location of the x-axis. The possible values are:

- "bottom". In this case the x axis is drawn at the bottom of the axes rectangle.
- "top". In this case the x axis is drawn at the top of the axes rectangle.

- "middle". In this case the x axis is drawn at the centered position.
- "origin". In this case the x axis is drawn at the origin.

y_location:

Specify the location of the y-axis. The possible values are:

- "left". In this case the y axis is drawn at the left of the axes rectangle.
- "right". In this case the y axis is drawn at the right of the axes rectangle.
- "middle". In this case the y axis is drawn at the centered position.
- "origin". In this case the y axis is drawn at the origin.

title:

An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a title with options on this label.

x_label:

An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on x axis with options on this label.

y_label:

An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on y axis with options on this label.

z_label:

An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on z axis with options on this label.

auto_ticks:

A 1x3 string vector giving the auto_ticks status for each axis. This property specifies whether each axis is graduated using a computational algorithm or not (graduations are set by the user). Its value should be "on" or "off" for a global setting. To act on a single axis, the syntax is `auto_ticks(N)` where N is 1, 2 or 3 corresponding to the x, y or z axis. Note that editing ticks (text and/or locations) via `x_ticks`, `y_ticks` or `z_ticks` automatically set `auto_ticks` to "off" for the corresponding axes.

x_ticks.locations:

A real vector containing the locations for the graduations on x axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `x_ticks` tlist defining a corresponding `x_ticks.labels` string vector too.

y_ticks.locations:

A real vector containing the locations for the graduations on y axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `y_ticks` tlist defining a corresponding `y_ticks.labels` string vector too.

z_ticks.locations:

A real vector containing the locations for the graduations on z axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `z_ticks` tlist defining a corresponding `z_ticks.labels` string vector too.

x_ticks.labels:

A string vector containing the labels for the graduations on x axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations,

man can act on the `x_ticks` `tlist` defining a corresponding `x_ticks.locations` real vector too.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

`y_ticks.labels`:

A string vector containing the labels for the graduations on y axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations, man can act on the `y_ticks` `tlist` defining a corresponding `y_ticks.locations` real vector too.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

`z_ticks.labels`:

A string vector containing the labels for the graduations on z axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations, man can act on the `z_ticks` `tlist` defining a corresponding `z_ticks.locations` real vector too.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

`box`:

This property specifies whether to enclose the axes in a box. Its value can be either "off", "hidden_axes", "back_half" or "on". If the property is "off", the box is not draw. If the property is "hidden_axes", only the back frame is drawn. If the property is "back_half", the X, Y and Z axis are also drawn. If the property is "on" the whole box is drawn.

`filled`:

This property specifies whether the axes background should be drawn or not. Its value can be either "off" or "on". If the property is "off", the background is not drawn, the axes box is transparent. If the property is "on" the background is drawn using the color specified by the `background` property.

`sub_ticks`:

This field sets the number of tics to draw between two main tics. The field value is the vector `[nx, ny]` where `nx` is the number of sub tics for the x-axis and `ny` respecting to the y-axis.

`font_style`:

Specifies the font used for displaying tics labels. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see `graphics_fonts`.

`font_size`:

It is a scalar specifying the character size of tics labels. If `fractional_font` property is "off" only the integer part of the value is used. For more information see `graphics_fonts`.

`font_color`:

This property determines the color of the tics labels.

`fractional_font`:

This property specify whether ticks labels are displayed using fractional font sizes. Its value must be either "on" or "off". If "on" the floating point value of `font_size` is used for display and the font is anti-aliased. If "off" only the integer part is used and the font is not smoothed.

`isoview`:

This property is used to have isometric scales on the x, y and z axes (for exemple to make the display of the curve $\sin(x)$ versus $\cos(x)$ be a circle not an ellipse). Its value should be "on"

or "off". If the value is "on", the axes `data_bounds` automatically change according to the corresponding figure `figure_size` property values.

cube_scaling:

This property is used in `3d mode` to have a rescaling of the x, y and z axes. Indeed, it allows the `data` to fit into a 1x1x1 cube ; the goal is to better display 3d graphics in case axes scales are very different from one to another. Its value should be "on" or "off" (which is the default value). In most cases, it helps generating Matlab-like 3d view.

view:

This field is related to the graphics universe. It takes "3d" as value corresponding to the three-dimensional views. In the other case its value can be "2d" for initial 2d plotting (default value). This flag also depends on the plots the user enters : a `plot3d` command, for example, will switch the `view` flag from "2d" to "3d".

rotation_angles:

This field is the vector `[alpha, theta]`. These two values give the spherical coordinates of the observation points (in degree).

log_flags:

3 character string that sets the scale (linear or logarithmic) along the axes. Each character specifies the scale for respectively the X, Y and Z axes. They should take a value between 'n' for linear scale or 'l' for logarithmic scale.

tight_limits:

If this property value is "on" axes adapt to fit exactly with the minima and maxima values of the `data_bounds`. If this field value is "off", axes may enlarge boundaries such as to produce pretty tics labels.

data_bounds:

This field contains the boundary values for the x,y and z coordinates. It is the matrix `[xmin,ymin,zmin;xmax,ymax,zmax]` or `[xmin,ymin;xmax,ymax]`. Note that, to strictly have the specified data bounds, `tight_limits` must be set to "on" value (see above).

zoom_box:

This field contains the current zoom box if any coordinates are given. It is an empty matrix (no zoom) or the vector `[xmin,ymin,xmax,ymax,zmin,zmax]` (defines a smaller axes box).

margins:

A vector `[margin_left,margin_right,margin_top,margin_bottom]` specifying the margins portion for this axes. This vector is composed of numbers between [0 1] with default: [0.125 0.125 0.125 0.125]. These numbers are ratios relative to associated values of the `axes_bounds` property, which are width for `margin_left` and `margin_right`, and height for `margin_top` and `margin_bottom`.

axes_bounds:

A vector `[x_left,y_up,width,height]` specifying the portion of figure used by this axes. Where `x_left`, `y_up`, `width` and `height` are numbers in [0 1] give respectively the position of the upper-left corner and the dimension of the axes (these numbers are ratio relative to the corresponding figure `figure_size` property values).

hidden_axis_color:

This property defines the color of the hidden axis. It takes an index relative to the current colormap.

user_data:

This field can be used to store any scilab variable in the axes data structure, and to retrieve it.

Properties for high level functions

The `plot`, `plot2dx`, `grayplot` and `matplot` functions use the following properties to decide how to merge consecutive plots if this is not stated by the `frameflag` calling argument. The result of the merge is decided through these two following properties:

`auto_clear`:

If this property value is equal to "on", a call to a high level graphic will re-initialize the current axes and erase all its children before performing the drawing. If the value is "off" the drawings will be added to current axes according to "auto_scale" property.

`auto_scale`:

A property to update the axes data boundary. If value is "on", a new plot will adapt the current axes properties to fit with previous and current plots. If its value is "off" the new plot will be drawn in the current axes data boundary.

Children's default values:

`hiddencolor`:

This property controls the hidden parts' color. It takes as value an index relative to the current colormap. In another case, if it is a negative value, the hidden parts take the same colors as the surface

`line_mode`:

This field contains the default `line_mode` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. Its value should be "on" (default) or "off".

`line_style`:

This field contains the default `line_style` property value for Segs, Arcs, Rectangle and Polyline objects. `line_style` selects the type of line to be used to draw lines. Its value should be an integer in [0 8]. 0 and 1 stand for solid, the other values stand for a selection of dashes (dash, dash dot, longdash dot, bigdash dot, bigdash longdash, dot, double dot).

`thickness`:

This field contains the default `thickness` property value for all objects using line drawing. It is a positive real specifying the line width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

`mark_mode`:

This field contains the default `mark_mode` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. Its value should be "on" or "off" (default).

`mark_style`:

This field contains the default `mark_style` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. `mark_style` selects the type of mark to be displayed. Its value should be an integer in [0 9] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, trefle and circle.

`mark_size_unit`:

This field contains the default `mark_size_unit` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. If `mark_size_unit` is set to "point", then the `mark_size` value is directly given in points. When `mark_size_unit` is set to "tabulated", `mark_size` is computed relative to the font size array; therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that `plot2d` and pure `scilab` functions use `tabulated` mode as default; when using `plot` function, the `point` mode is automatically enabled.

`mark_size`:

This field contains the default `mark_size` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. `mark_size` selects the font size of mark to be displayed.

Its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt (see `getmark`).

`mark_foreground`:

This field contains the default `mark_foreground` property value for all objects created under this axes. Polyline, rectangle, legend, surface, segment and axis objects are using this property to specify a foreground (edge) color for their marks. Its value should be a color index (relative to the current `color_map`) or 0 for transparent edge. Note that the default value is -1 (default black) and, even if you change the `color_map`, this -1 value will always point onto the default black color.

`mark_background`:

This property controls the default `mark_background` property value for all objects created under this axes. Polyline, rectangle, legend, surface, segment and axis objects are using this property to specify a background (face) color for their marks. It takes as value an index relative to the current colormap or 0 for transparent face. Note that the default value is -2 (default white) and, even if you change the `color_map`, this -2 value will always point onto the default white color.

`foreground`:

This field contains the default `foreground` property value for axes and all objects created under this axes. Its value should be a color index (relative to the current `color_map`). Note that the default value is -1 (default black) and, even if you change the `color_map`, this -1 value will always point onto the default black color.

`background`:

This property controls the default `background` property value for axes and all objects created under this axes. It takes as value an index relative to the current colormap. Note that the default value is -2 (default white) and, even if you change the `color_map`, this -2 value will always point onto the default white color.

`arc_drawing_mode`:

This property controls the default `arc_drawing_mode` property value for all created Arc objects under this Axes object. Its value should be either "nurbs" or "lines".

`clip_state`:

This field contains the default `clip_state` property value for all objects. Its value should be :

- "off" this means that all objects created after that are not clipped (default value).
- "clipgrf" this means that all objects created after that are clipped outside the Axes boundaries.
- "on" this means that all objects created after that are clipped outside the rectangle given by property `clip_box`.

`clip_box`:

This field contains the default `clip_box` property value for all objects. Its value should be an empty matrix if `clip_state` is "off". Other case the clipping is given by the vector [x, y, w, h] (upper-left point width height).

Note on default values :

All these listed properties and fields inherit from default values stored in an axes model. These default values can be seen and changed. To do so, use the `get("default_axes")` command : it returns a graphic handle on the axes model. Note that no graphic window is created by this command. The next created axes will inherit from this model (see "Example on axes model" below).

Examples

```

lines(0) // disables vertical paging
a=get("current_axes");//get the handle of the newly created axes
a.axes_visible="on"; // makes the axes visible
a.font_size=3; //set the tics label font size
a.x_location="top"; //set the x axis position
a.data_bounds=[-100,-2,-1;100,2,1]; //set the boundary values for the x, y and z
a.sub_tics=[5,0];
a.labels_font_color=5;
a.grid=[2,2];
a.box="off";

// Example with 3D axes
clf(); //clear the graphics window
x=0.1:0.1:2*pi;plot2d(x-.3,sin(x)*7+.2);
a=gca(); // get the handle of the current axes
a.grid=[1 -1 -1]; //make x-grid
a.rotation_angles=[70 250]; //turn the axes with giving angles
a.grid=[1 6 -1]; //make y-grid
a.view="2d"; //return te the 2d view
a.box="back_half";
a.labels_font_color=5;
a.children.children.thickness=4;
a.children.children.polyline_style=3;
a.view="3d"; //return te the 3d view
a.children.children.thickness=1;
a.children.children.foreground=2;
a.grid=[1 6 3]; //make z-grid
a.parent.background=4;
a.background=7;
plot2d(cos(x)+1,3*sin(x)-3);
plot2d(cos(x)+7,3*sin(x)+3);
a.children(2).children.polyline_style=2;
a.children(1).children.polyline_style=4;
a.children(1).children.foreground=5;
a.children(2).children.foreground=14;
a.parent.figure_size= [1200,800];
a.box="on";
a.labels_font_size=4;
a.parent.background=8;
a.parent.figure_size= [400,200];
a.rotation_angles=[0 260];
delete(a.children(2));
delete(); // delete current object

a = gca();
a.labels_font_size=1;
a.auto_clear= "on";
x=0:0.1:2.5*pi;plot2d(10*cos(x),sin(x));

a.data_bounds(:,1) = [1;15] ; // set positive bounds for X axe
a.log_flags = "lnn" ; // set X axes to logarithmic scale
a.log_flags = "nnn" ; // switch back to linear scale

a=gca();

```

```

a.rotation_angles=[45 45];
a.data_bounds=[-20, -3, -2; 20 3 ,2];
xrect([-4 0.5 8 1]);
a.auto_clear = "off" ;
a.isoview="on"; // isoview mode
xrect([-2 0.25 4 0.5]);
a.children(1).fill_mode="on";
a.axes_visible="off";
a.children(1).data=[-2 0.25 -1 4 0.5];
a.children(2).data=[-4 0.5 1 8 1];
x=2*pi*(0:7)/8;
xv=[.2*sin(x);.9*sin(x)];yv=[.2*cos(x);.9*cos(x)];
xsegs(10*xv,yv,1:8)
s=a.children(1);
s.arrow_size=1;
s.segs_color=5;
a.data_bounds //the boundary values for the x,y and z coordinates
a.view="2d";
a.data_bounds=[-10,-1; 10,1]; // set the boundary values for the two-dimensiona

// Example on axes model
da=gda() // get the handle on axes model to view and edit the fields
// title by default
da.title.text="My Default@Title"
da.title.foreground = 12;
da.title.font_size = 4;
// x labels default
da.x_label.text="x";
da.x_label.text="x"; // Latex or MathML can be used here also
da.x_label.font_style = 8;
da.x_label.font_size = 2;
da.x_label.foreground = 5;
da.x_location = "middle";
// y labels default
da.y_label.text="y";
da.y_label.font_style = 3;
da.y_label.font_size = 5;
da.y_label.foreground = 3;
da.y_location = "right";
da.thickness = 2;
da.foreground = 7;
// the plot
x=(0:0.1:2*pi)';
plot2d(x,[sin(x),sin(2*x),sin(3*x)],style=[1,2,3],rect=[0,-2,2*pi,2]);
sda() // back to default axes model

// Example with LaTeX / MathML ticks:
plot2d();
a=gca();

mathml="<mrow><mfrac><mrow><mi>d</mi> <mi>y</mi></mrow><mrow> <mi>d</mi> <mi>
<mfrac><mn>1</mn><msup> <mi>y</mi> <mn>2</mn></msup> </mfrac> </mrow>";
// Only LaTeX expression
a.x_ticks.labels=["0";"1";"$\sin(x)$";"3";"$\cos(a) - test$";"5";"6";"7"];

// Mixed expression: MathML and LaTeX
a.y_ticks.labels=["0";"1";"2";"3";"$\cos(a)$";"5";"6";mathml;"8"];

```

See Also

lines , set , get , gca , gda , gcf , sda , sdf , scf , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

axis_properties — description of the axis entity properties

Description

The Axis entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for axis scaling and appearance.

Axis properties

parent:

This property contains the handle of the parent. The parent of the axis entity should be of the type "Axes" or "Compound".

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, the axis entity is visible, the value's property is "on". If "off", the axis is not drawn on the screen.

tics_direction:

Specify the direction of the tics drawn under the horizontal axis and the vertical axis. The possible values of this property are:

- "top". In this case, tics are drawn at the top of the horizontal axis.
- "bottom". In this case, tics are drawn at the bottom of the horizontal axis.
- "left". In this case, tics are going left on the vertical axis.
- "right". In this case, tics are going right on the vertical axis.

The defaults values are "top" for the horizontal axis and "right" for vertical axis.

xtics_coord:

This field represents the x-coordinate of the axis. It is a row vector containing values increasing from left to right which give tics positions for a horizontal axis. Other case, the entity is a vertical axis, this property contain a scale which defines the x-origin of the axis.

yticks_coord:

This field represents the y-coordinate of the axis. It is a row vector containing values increasing from bottom to top which give tics positions for a vertical axis. Other case, the entity is a horizontal axis, this property contain a scale which defines the y-origin of the axis.

tics_color:

The value of this properties is index of the color used to draw the axis'lines and tics.

tics_segment:

This field contains a flag which controls the display of the base segment of the axis. The default is "on", else if to not display it, the property takes "off" as value.

tics_style:

This property describes how the tics are given. It is a string flag which can have these possible values:

- "v". It's the default value, In this case, tics positions are given by the row factor `xtics_coord` for horizontal axis (`yticks_coord` for the vertical one).
- "r". In this case, tics positions are given by the vector `[min,max,n]` where n is the number of intervals.

- "i". In this case the vector given tics positions is of size 4, [k1, k2, a, n] then values are increasing between $k1 \cdot 10^a$ and $k2 \cdot 10^a$, n is the number of intervals.

sub_tics:

This field sets the number of tics to draw between two main tics.

tics_labels:

This field is a string matrix, which contains the strings to be drawn along the axis at tics positions.

format_n:

This property is a character string which specifies the floating-point display format of the tics labels numbers, when relevant. It uses the format syntax of the C language printf function (for example "%.3f"). If equal to "", a default display format is used.

labels_font_color:

This property determines the color of the tics labels.

labels_font_size:

It is a scalar specifying the character size of tics labels. If fractional_font property is "off" only the integer part of the value is used. For more information see graphics_fonts.

fractional_font:

This property specifies whether ticks labels are displayed using fractional font sizes. Its value must be either "on" or "off". If "on" the floating point value of font_size is used for display and the font is anti-aliased. If "off" only the integer part is used and the font is not smoothed.

clip_state:

This field contains the clip_state property value for the arc. Clip_state value should be :

- "off" this means that the axis is not clipped
- "clipgrf" this means that the axis is clipped outside the Axes box.
- "on" this means that the axis is clipped outside the arc given by property clip_box.

clip_box:

This field is to determinate the clip_box property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector [x, y, w, h] (upper-left point width height) defines the portions of the axis to display, however clip_state property value will be changed.

user_data:

This field can be used to store any scilab variable in the axis data structure, and to retrieve it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-1, -1;10,10];

drawaxis(x=2:7,y=4,dir='u');
a1=a.children(1)
a1.xtics_coord=[1 4 5 8 10];
a1.tics_color=2;
a1.labels_font_size=3;
a1.tics_direction="bottom";
a1.tics_labels= [" February" "May" "june" "August" "October"];
```



```
drawaxis(x=1.2:1:10,y=5,dir='u',textcolor=13);
a2=get("hdl")
a2.sub_tics=0;
a2.tics_segment="off";
a2.ytics_coord=4;

drawaxis(x=-1,y=0:1:7,dir='r',fontsize=10,textcolor=5,ticscolor=6,sub_int=10)
a3=get("hdl");
a3.tics_labels= 'B' +string(0:7);
a3.tics_direction="left";
```

See Also

set , get , delete , drawaxis , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

bar — bar histogram

```
bar(y)
bar(x,y)
bar([h],x,y [,width [,color [,style]]])
```

Parameters

- h**
an axes handle, (default: `h=gca()`).
- y**
a real scalar, vector of size N, or a matrix N*M.
- x**
a real scalar or a vector of size N, (default: if y is a vector then x is a vector and x length equals to y length. If y is a matrix then x is a vector and x length equals to the lines number of y).
- width**
(optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8, i.e 80%).
- color**
(optional), a string (default: 'blue'), specifying the inside color bar.
- style**:
a string, 'grouped' or 'stacked' (default: 'grouped').

Description

`bar(y, ...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6. If y is a vector, bar draws vector y versus vector `1:size(y,*)`. If y is a matrix N*M, bar draws M polylines (type 6), each polyline corresponds to a column of y versus vector `x=1:size(y,1)`.

`bar(x,y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6, where x length = y length. If y is a matrix NxM then bar function draws M polylines which have the type 6. Each polyline corresponds to a column of y versus vector x.

`bar(h, ...)` : defines the current axes where the drawing is performed.

`bar(...,width,...)` : defines the width of the bar(s) in percentage (generally: $0 < \text{width} \leq 1$).

`bar(...,style,...)` : defines how the bar is drawn. If y is a matrix N*M (so M polylines of type 6) then there are two ways to draw the M bars. the style option = 'grouped' allows to center the M polylines versus each components of x, and the style option 'stacked' allows to stack them.

`bar(...,color,...)` : defines the bar color. Bar functions uses the same colormap than in the `plot` function.

If there are several bar calls, the `barhomogenize` function allows to homegenize the width and style of all bars (i.e polylines of type 6) included in the current working axes.

Examples

```
// First example: draw a bar (i.e a polyline with polyline_style type =6) with
```

```
// width=0.5 and color='yellow' and default style='grouped', x=1:length(y).
scf(0);
y=[1 -3 5];
bar(y,0.5,'yellow');

// Second example: draw 3 bars (i.e 3 polylines with polyline_style type =6),de
scf(1);
x=[1 2 5]; y=[1 -5 6;3 -2 7;4 -3 8];
bar(x,y);

// Third example : style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
bar(x,y, 'stacked');

// Fourth example: width=0.2;color='green'; default style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
bar(x,y,0.2,'green');
```

See Also

[barh](#) , [barhomogenize](#) , [plot](#) , [polyline_properties](#)

Authors

Farid Belahcene

Name

barh — horizontal display of bar histogram

```
barh(y)
barh(x,y)
barh([h],x,y [,width [,color [,style]]])
```

Parameters

- h**
an axes handle, (default: `h=gca()`).
- y**
a real scalar, vector of size N, or a matrice N*M.
- x**
a real scalar or a vector of size N, (default: if y is a vector then x is a vector and x length equals to y length. If y is a matrix then x is a vector and x length equals to the lines number of y.
- width**
(optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8, i.e=80%).
- color**
(optional), a string (default: 'blue'), specifying the inside color bar.
- style:**
a string, 'grouped' or 'stacked' (default: 'grouped').

Description

`barh(y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6. If y is a vector, bar draws vector y versus vector `x=1:size(y,*)` . If y is a matrix N*M, bar draws M polylines (type 6), each polyline corresponds to a column of y versus vector `x=1:size(y,1)`.

`barh(x,y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6, where x length = y length. If y is a matrix NxM then bar function draws M polylines which have the type 6. Each polyline corresponds to a column of y versus vector x.

`barh(h,...)` : defines the current axes where the drawing is performed.

`barh(...,width,...)` : defines the width of the bar(s) in percentage (generally: $0 < \text{width} < 1$).

`barh(...,style,...)` : defines how the bar is drawn. If y is a matrix N*M (so M polylines of type 6) then there are two ways to draw the M bars. the style option = 'grouped' allows to center the M polylines versus each components of x, and the style option = 'stacked' allows to stack them.

`barh(...,color,...)` : defines the bar color. Bar functions uses the same colormap than in the `plot` function.

If there are several bar calls, the `barhomogenize` function allows to homegenize the width and style of all bars (i.e polylines of type 6) included in the current working axes.

Examples

```
// First example: draw a bar (i.e a polyline with polyline_style type =6),default
scf(0);
y=[1 -3 5];
barh(y,0.5,'yellow');

// Second example: draw 3 bars (i.e 3 polylines with polyline_style type =6),default
scf(1);
x=[1 2 5]; y=[1 -5 6;3 -2 7;4 -3 8];
barh(x,y);

// Third example : style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
barh(x,y,'stacked');

// Fourth example: width=0.2;color='green'; default style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
barh(x,y,0.2,'green');
```

See Also

[bar](#) , [barhomogenize](#) , [plot](#) , [polyline_properties](#)

Authors

Farid Belahcene

Name

barhomogenize — homogenize all the bars included in the current working axes

```
barhomogenize()  
barhomogenize([h[, 'style'[, 'width']]])
```

Parameters

h
an axes handle, (default: `h=gca()`).

style
a string, 'grouped' or 'stacked' (default: 'grouped').

width
(optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8).

Description

If there are several bar calls, the `barhomogenize` function allows to homogenize the width and style of all bars (i.e which has the `polyline_style` type 6) included in the current working axes. These bars must have the same x data.

`barhomogenize( )` : takes the default values, `h=gca()`, `width=0.8`, `style='grouped'`.

`barhomogenize(h, ...)` : defines the current axes where the drawing is performed.

`barhomogenize(..., width, ...)` : defines the width of the bar(s) in percentage (generally: $0 < \text{width} \leq 1$).

`barhomogenize(..., style, ...)` : defines how the bars are drawn. The 'grouped' option allows to center the M polylines versus each components of x, and the 'stacked' option allows to stack them.

Examples

```
// First example: creation of 1 yellow bar (i.e 1 polyline with polyline_style=6)  
subplot(2,3,1)  
xtitle('ex1: creation of 1 yellow bar and 3 bars ')  
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];  
bar(x,y1,'yellow');bar(x,y2);  
// grouped homogenisation of these 4 bars  
subplot(2,3,2)  
xtitle('grouped homogenisation')  
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];  
bar(x,y1,'yellow');bar(x,y2);  
barhomogenize();  
// stacked homogenisation of thes 4 bars  
subplot(2,3,3)  
xtitle('stacked homogenisation')  
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];  
bar(x,y1,'yellow');bar(x,y2);  
barhomogenize('stacked',1);  
  
// Second example : creation of 1 red bar (i.e 1 polyline with polyline_style=6)
```

```
subplot(2,3,4)
xtitle('ex2: creation of 1 bar and 2 polylines')
x=1:10; y=sin(x)/2;
bar(x,y, 'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
// modify the polyline_style type of the second polyline from plot (this polyli
subplot(2,3,5)
xtitle('transformation of the second polyline to bar')
x=1:10; y=sin(x)/2;
bar(x,y, 'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
// homogenisation of the first bar (from bar function) and second bar (from the
subplot(2,3,6)
xtitle('grouped homogenisation')
x=1:10; y=sin(x)/2;
bar(x,y, 'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
barhomogenize();
// change the style and the width
//barhomogenize('stacked',0.5);
//barhomogenize('stacked',1);
```

See Also

[bar](#) , [polyline_properties](#)

Authors

Farid Belacehne

Name

bonecolormap — gray colormap with a light blue tone

```
cmap=bonecolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

bonecolormap computes a gray colormap with a light blue tone.

Examples

```
f = scf();  
plot3d1();  
f.color_map = bonecolormap(32);
```

See Also

colormap , autumncolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

captions — draw graph captions

```
hl=captions(h, strings [,location])
```

Parameters

h

vector of handles on polyline entities.

strings

n vector of strings, strings(i) is the caption of the ith polyline

hl

a handle of type "Legend", points to the structure containing all the captions information (see legend_properties).

location

a character string with possible values:

- "in_upper_right" : captions are drawn in the upper right corner of the axes box.
- "in_upper_left": captions are drawn in the upper left corner of the axes box.
- "in_lower_right": captions are drawn in the lower right corner of the axes box.
- "in_lower_left": captions are drawn in the lower left corner of the axes box.
- "out_upper_right": captions are drawn at the right of the upper right corner of the axes box.
- "out_upper_left": captions are drawn at the left of the upper left corner of the axes box.
- "out_lower_right": captions are drawn at the right of the lower right corner of the axes box.
- "out_lower_left": captions are drawn at the left of the lower left corner of the axes box.
- "upper_caption": captions are drawn above the upper left corner of the axes box.
- "lower_caption": captions are drawn below the lower left corner of the axes box. This option correspond to the `leg` argument of `plot2d`
- "by_coordinates": the upper left corner of the captions box is given by the "position" field of the associated data structure. The x and y positions are given as fractions of the axes_bounds sizes.

Description

Puts captions on the current plot at the in the bottom left corner of the graphic window using the specified strings as labels. `captions` prepends labels by a recall of the corresponding polylines. The type and properties are recovered from the given handles:

The `captions` function creates a Legend data structure.

There is at most one Legend associated with each axes. If the caption function is recalled while a Legend still exist the old one is replaced.

Examples

```
t=0:0.1:2*pi;
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];
a.margins(4)=0.2;

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[1,2 3]);
e=gce();
e.children(1).thickness=3;
e.children(2).line_style=4;

hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)']);
hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)'],'in_upper_right');

hl.legend_location='in_upper_right'
hl.fill_mode='on';
```

See Also

[plot2d](#), [legend](#), [polyline_properties](#), [legend_properties](#)

Name

champ — 2D vector field plot

```
champ(x,y,fx,fy,[arfact,rect,strf])
champ(x,y,fx,fy,<opt_args>)
```

Parameters

x,y

two vectors which define the grid.

fx

a matrix which describes the x component of the vector field. $fx(i,j)$ is the x component of the vector field at point $(x(i),y(j))$.

fy

a matrix which describes the y component of the vector field. $fy(i,j)$ is the y component of the vector field at point $(x(i),y(j))$.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `arfact, rect, strf` (see below).

arfact

an optional argument of type `real` which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).

rect

a vector `rect=[xmin,ymin,xmax,ymax]` which gives the boundaries of the graphics frame to use.

strf

a string of length 3 "xyz" which has the same meaning as the `strf` parameter of `plot2d`. The first character `x` has no effect with `champ`.

Description

`champ` draws a 2D vector field. The length of the arrows is proportional to the intensity of the field.

If you want colored arrows with the color of the arrows depending on the intensity of the field, use `champ1`.

Enter the command `champ( )` to see a demo.

Examples

```
// using rect as plot boundaries
champ(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
// using (x,y) to get boundaries
clf()
champ(-5:5,-5:5,rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")
```

See Also

`champ1` , `fchamp` , `plot2d`

Authors

J.Ph.C.

Name

champ1 — 2D vector field plot with colored arrows

```
champ1(x,y,fx,fy,[arfact,rect,strf])
```

Parameters

x,y

two vectors which define the grid.

fx

a matrix which describes the x component of the vector field. $fx(i,j)$ is the x component of the vector field at point $(x(i),y(j))$.

fy

a matrix which describes the y component of the vector field. $fy(i,j)$ is the y component of the vector field at point $(x(i),y(j))$.

arfact

an optional argument of type real which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).

rect

a vector `rect=[xmin,ymin,xmax,ymax]` which gives the boundaries of the graphics frame to use.

frameflag

controls the computation of the actual coordinate ranges from the minimal requested values. The associated value should be an integer ranging from 0 to 8.

axesflag

specifies how the axes are drawn. The associated value should be an integer ranging from 0 to 5.

strf

a string of length 3 "xyz" which has the same meaning as the `strf` parameter of `plot2d`. The first character x has no effect with `champ1`.

Description

`champ1` draws a 2D vector field with colored arrows. The color of the arrows depends on the intensity of the field.

If you want arrows proportional to the intensity of the field, use `champ`.

Enter the command `champ1()` to see a demo.

Examples

```
champ1(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
```

See Also

`champ` , `fchamp` , `plot2d`

Authors

J.Ph.C.

Name

champ_properties — description of the 2D vector field entity properties

Description

The Champ entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a 2D vector field.

visible:

This properties contains the `visible` property value for the entity . It should be "on" or "off" . If "on" the vector field is drawn, If "off" the vector field is not displayed on the screen.

data:

This field defines a `tlist` data structure of type "champdata" composed of a row and column indices of each element : the x and y grid coordinates are contained respectively in `data.x` and `data.y`. The complementary fields named `data.fx` and `data.fy` are matrices which describe respectively the x and y component of the vector field at point $(x(i), y(j))$.

user_data:

This field can be use to store any scilab variable in the champ data structure, and to retrieve it.

line_style:

The `line_style` property value should be an integer in [0 8]. 0 and 1 stands for solid, the other value stands for a selection of dashes (see `getlinestyle`).

thickness:

This property is a positive real specifying the arrow width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

colored:

If this this property value is "on", fields vectors are drawn using a color proportional to the intensity of the field.

clip_state:

This field contains the `clip_state` property value for the champ. It should be :

- "off" this means that the vector field is not clipped
- "clipgrf" this means that the vector field is clipped outside the Axes box.
- "on" this means that the vector field is clipped outside the rectangle given by property `clip_box`.

clip_box:

This property contains the `clip_box` property. Its value should be an empty matrix if `clip_state` is "off" .Other cases the vector `[x, y, w, h]` (upper-left point width height) defines the portions of the vector field to display, however `clip_state` property value will be changed.

parent:

This property contains the handle of the parent. The parent of the 2D vector field entity should be of the type "Axes" or "Compound".

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-10,-10;10,10];
```

```
champ(-5:5, -5:5, rand(11,11), rand(11,11))  
  
c=a.children  
  
c.colored="on";  
c.thickness=2;  
c.data // display the tlist of type "scichampdata"  
a.data_bounds=[-5, -5;5,5];
```

See Also

set , get , delete , champ , champ1 , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

`clear_pixmap` — erase the pixmap buffer

```
clear_pixmap()
```

Description

If a graphic window `pixmap` property is "on" the drawings are send to a pixmap memory instead of the screen display.

The `clear_pixmap()` instruction erase the pixmap, but not the screen.

The pixmap mode can be used to obtain smooth animations.

See Also

`figure_properties` , `show_pixmap`

Authors

Serge Steer INRIA

Name

`clf` — clear or reset the current graphic figure (window) to default values

```
clf(<opt_job_arg>)
clf(h,<opt_job_arg>)
clf(num,<opt_job_arg>)
```

Parameters

`h`
a handle, the figure handle

`num`
a number, the `figure_id`

`<opt_job_arg>`
a string ('clear' or 'reset') specifying the job for `clf`.

Description

The `clf` command is used to reset a figure to its default values and/or to delete all its children.

If `opt_job_arg` string value is 'clear' then all children of the specified figure are deleted.

If `opt_job_arg` string value is 'reset' then not only all children of the specified figure are deleted but the figure properties are reset to their default values using the default figure model (see `gdf`). The only exception are the `axes_size` and `figure_size` properties which can not be reset if the figure is docked with other elements.

`clf(num)` clear or reset the figure with `figure_id==num`.

`clf(h)` clear or reset the figure pointed to by the handle `h`.

`clf()` clear or reset the current figure.

`clf` function delete every children of specified windows including menus and uicontrols added by user. To prevent menus and uicontrols from being deleted, the `delete(gca())` command might be used instead.

Examples

```
f4=scf(4); //creates figure with id==4 and make it the current one
f4.color_map = jetcolormap(64);
f4.figure_size = [400, 200];
plot2d()
clf(f4,'reset')

f0=scf(0); //creates figure with id==0 and make it the current one
f0.color_map=hotcolormap(128); // change color map
t=-%pi:0.3:%pi;
plot3d1(t,t,sin(t)*cos(t));

clf() // equivalent to clf(gcf(),'clear')
plot3d1(t,t,sin(t)*cos(t)); // color_map unchanged

clf(gcf(),'reset')
```

```
plot3d1(t,t,sin(t)'*cos(t)); // color_map changed (back to the default one with
```

See Also

`set` , `get` , `gcf` , `gdf` , `scf` , `graphics_entities`

Authors

S. Steer & F.Leray INRIA

Name

`color` — returns the color id of a color

```
id=color(name)
id=color(r,g,b)
```

Parameters

`name`

name of a color.

`r,g,b`

RGB integer values of a color.

`id`

id of the color.

Description

`color` returns the color id corresponding to its argument:

- `name` must be the name of a known color (see `color_list`).
- `r`, `g` and `b` must be integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color.

If the requested color does not exist in the current colormap it is added to the colormap.

This function can be used to specify the foreground or background colors when plotting.

Examples

```
x=linspace(-2*pi,2*pi,100)';
// using existing colors
plot2d(x,[sin(x),cos(x)],style=[color("red"),color("green")]);
// new colors: there are added to the colormap
e=gce(); p1=e.children(1); p2=e.children(2);
p1.foreground=color("purple"); p2.foreground=color("navy blue");
// using RGB values
p1.foreground=color(255,128,128);
```

See Also

`colormap` , `color_list` , `getcolor`

Name

color_list — list of named colors

Description

You will find below the names of the colors known by Scilab. The RGB values are given after the name. Note that sometimes colors have more than 1 name.

scilab blue4	0	0	144
scilabblue4	0	0	144
scilab blue3	0	0	176
scilabblue3	0	0	176
scilab blue2	0	0	208
scilabblue2	0	0	208
scilab green4	0	144	0
scilabgreen4	0	144	0
scilab green3	0	176	0
scilabgreen3	0	176	0
scilab green2	0	208	0
scilabgreen2	0	208	0
scilab cyan4	0	144	144
scilabcyan4	0	144	144
scilab cyan3	0	176	176
scilabcyan3	0	176	176
scilab cyan2	0	208	208
scilabcyan2	0	208	208
scilab red4	144	0	0
scilabred4	144	0	0
scilab red3	176	0	0
scilabred3	176	0	0
scilab red2	208	0	0
scilabred2	208	0	0
scilab magenta4	144	0	144
scilabmagenta4	144	0	144
scilab magenta3	176	0	176
scilabmagenta3	176	0	176
scilab magenta2	208	0	208
scilabmagenta2	208	0	208
scilab brown4	128	48	0
scilabbrown4	128	48	0
scilab brown3	160	64	0
scilabbrown3	160	64	0
scilab brown2	192	96	0
scilabbrown2	192	96	0

scilab pink4	255	128	128
scilabpink4	255	128	128
scilab pink3	255	160	160
scilabpink3	255	160	160
scilab pink2	255	192	192
scilabpink2	255	192	192
scilab pink	255	224	224
scilabpink	255	224	224

snow	255	250	250
ghost white	248	248	255
ghostwhite	248	248	255
white smoke	245	245	245
whitesmoke	245	245	245
gainsboro	220	220	220
floral white	255	250	240
floralwhite	255	250	240
old lace	253	245	230
oldlace	253	245	230
linen	250	240	230
antique white	250	235	215
antiquewhite	250	235	215

papaya whip	255	239	213
papayawhip	255	239	213
blanched almond	255	235	205
blanchedalmond	255	235	205
bisque	255	228	196

peach puff	255	218	185
peachpuff	255	218	185
navajo white	255	222	173
navajowhite	255	222	173
moccasin	255	228	181
cornsilk	255	248	220
ivory	255	255	240
lemon chiffon	255	250	205
lemonchiffon	255	250	205
seashell	255	245	238
honeydew	240	255	240
mint cream	245	255	250
mintcream	245	255	250
azure	240	255	255
alice blue	240	248	255

aliceblue	240	248	255
lavender	230	230	250
lavender blush	255	240	245
lavenderblush	255	240	245

misty rose	255	228	225
mistyrose	255	228	225
white	255	255	255
black	0	0	0
dark slate gray	47	79	79
darkslategray	47	79	79
dark slate grey	47	79	79
darkslategrey	47	79	79
dim gray	105	105	105
dimgray	105	105	105
dim grey	105	105	105
dimgrey	105	105	105
slate gray	112	128	144
slategray	112	128	144
slate grey	112	128	144
slategrey	112	128	144
light slate gray	119	136	153
lightslategray	119	136	153
light slate grey	119	136	153
lightslategrey	119	136	153
gray	190	190	190
grey	190	190	190
light grey	211	211	211
lightgrey	211	211	211
light gray	211	211	211
lightgray	211	211	211

midnight blue	25	25	112
midnightblue	25	25	112
navy	0	0	128
navy blue	0	0	128
navyblue	0	0	128
cornflower blue	100	149	237
cornflowerblue	100	149	237
dark slate blue	72	61	139
darkslateblue	72	61	139
slate blue	106	90	205
slateblue	106	90	205

medium slate blue	123	104	238
mediumslateblue	123	104	238
light slate blue	132	112	255
lightslateblue	132	112	255
medium blue	0	0	205
mediumblue	0	0	205
royal blue	65	105	225
royalblue	65	105	225
blue	0	0	255
dodger blue	30	144	255
dodgerblue	30	144	255
deep sky blue	0	191	255
deepskyblue	0	191	255
sky blue	135	206	235
skyblue	135	206	235
light sky blue	135	206	250
lightskyblue	135	206	250
steel blue	70	130	180
steelblue	70	130	180
light steel blue	176	196	222
lightsteelblue	176	196	222
light blue	173	216	230
lightblue	173	216	230
powder blue	176	224	230
powderblue	176	224	230

pale turquoise	175	238	238
paleturquoise	175	238	238
dark turquoise	0	206	209
darkturquoise	0	206	209
medium turquoise	72	209	204
mediumturquoise	72	209	204
turquoise	64	224	208
cyan	0	255	255
light cyan	224	255	255
lightcyan	224	255	255
cadet blue	95	158	160
cadetblue	95	158	160
medium aquamarine	102	205	170
mediumaquamarine	102	205	170
aquamarine	127	255	212

dark green	0	100	0

darkgreen	0	100	0
dark olive green	85	107	47
darkolivegreen	85	107	47
dark sea green	143	188	143
darkseagreen	143	188	143
sea green	46	139	87
seagreen	46	139	87
medium sea green	60	179	113
mediumseagreen	60	179	113
light sea green	32	178	170
lightseagreen	32	178	170
pale green	152	251	152
palegreen	152	251	152
spring green	0	255	127
springgreen	0	255	127
lawn green	124	252	0
lawngreen	124	252	0
green	0	255	0
chartreuse	127	255	0
medium spring green	0	250	154
mediumspringgreen	0	250	154
green yellow	173	255	47
greenyellow	173	255	47
lime green	50	205	50
limegreen	50	205	50
yellow green	154	205	50
yellowgreen	154	205	50
forest green	34	139	34
forestgreen	34	139	34
olive drab	107	142	35
olivedrab	107	142	35

dark khaki	189	183	107
darkkhaki	189	183	107
khaki	240	230	140
pale goldenrod	238	232	170
palegoldenrod	238	232	170
light goldenrod yellow	250	250	210
lightgoldenrodyellow	250	250	210
light yellow	255	255	224
lightyellow	255	255	224
yellow	255	255	0

gold	255	215	0
light goldenrod	238	221	130
lightgoldenrod	238	221	130
goldenrod	218	165	32
dark goldenrod	184	134	11
darkgoldenrod	184	134	11

rosy brown	188	143	143
rosybrown	188	143	143
indian red	205	92	92
indianred	205	92	92
saddle brown	139	69	19
saddlebrown	139	69	19
sienna	160	82	45
peru	205	133	63
burlywood	222	184	135
beige	245	245	220
wheat	245	222	179
sandy brown	244	164	96
sandybrown	244	164	96
tan	210	180	140
chocolate	210	105	30
firebrick	178	34	34
brown	165	42	42
dark salmon	233	150	122
darksalmon	233	150	122
salmon	250	128	114
light salmon	255	160	122
lightsalmon	255	160	122

orange	255	165	0
dark orange	255	140	0
darkorange	255	140	0
coral	255	127	80
light coral	240	128	128
lightcoral	240	128	128
tomato	255	99	71
orange red	255	69	0
orangered	255	69	0
red	255	0	0
hot pink	255	105	180
hotpink	255	105	180
deep pink	255	20	147

deeppink	255	20	147
pink	255	192	203
light pink	255	182	193
lightpink	255	182	193
pale violet red	219	112	147
palevioletred	219	112	147
maroon	176	48	96
medium violet red	199	21	133
mediumvioletred	199	21	133
violet red	208	32	144
violetred	208	32	144
magenta	255	0	255
violet	238	130	238
plum	221	160	221
orchid	218	112	214
medium orchid	186	85	211
mediumorchid	186	85	211
dark orchid	153	50	204
darkorchid	153	50	204
dark violet	148	0	211
darkviolet	148	0	211
blue violet	138	43	226
blueviolet	138	43	226
purple	160	32	240
medium purple	147	112	219
mediumpurple	147	112	219
thistle	216	191	216
snow1	255	250	250
snow2	238	233	233
snow3	205	201	201
snow4	139	137	137
seashell1	255	245	238
seashell2	238	229	222
seashell3	205	197	191
seashell4	139	134	130
antiquewhite1	255	239	219
antiquewhite2	238	223	204
antiquewhite3	205	192	176
antiquewhite4	139	131	120
bisque1	255	228	196
bisque2	238	213	183

bisque3	205	183	158
bisque4	139	125	107
peachpuff1	255	218	185
peachpuff2	238	203	173
peachpuff3	205	175	149
peachpuff4	139	119	101
navajowhite1	255	222	173
navajowhite2	238	207	161
navajowhite3	205	179	139
navajowhite4	139	121	94
lemonchiffon1	255	250	205
lemonchiffon2	238	233	191
lemonchiffon3	205	201	165
lemonchiffon4	139	137	112
cornsilk1	255	248	220
cornsilk2	238	232	205
cornsilk3	205	200	177
cornsilk4	139	136	120
ivory1	255	255	240
ivory2	238	238	224
ivory3	205	205	193
ivory4	139	139	131
honeydew1	240	255	240
honeydew2	224	238	224
honeydew3	193	205	193
honeydew4	131	139	131

lavenderblush1	255	240	245
lavenderblush2	238	224	229
lavenderblush3	205	193	197
lavenderblush4	139	131	134
mistyrose1	255	228	225
mistyrose2	238	213	210
mistyrose3	205	183	181
mistyrose4	139	125	123

azure1	240	255	255
azure2	224	238	238
azure3	193	205	205
azure4	131	139	139
slateblue1	131	111	255
slateblue2	122	103	238
slateblue3	105	89	205

slateblue4	71	60	139
royalblue1	72	118	255
royalblue2	67	110	238
royalblue3	58	95	205
royalblue4	39	64	139
blue1	0	0	255
blue2	0	0	238
blue3	0	0	205
blue4	0	0	139
dodgerblue1	30	144	255
dodgerblue2	28	134	238
dodgerblue3	24	116	205
dodgerblue4	16	78	139
steelblue1	99	184	255
steelblue2	92	172	238
steelblue3	79	148	205
steelblue4	54	100	139
deepskyblue1	0	191	255
deepskyblue2	0	178	238
deepskyblue3	0	154	205
deepskyblue4	0	104	139
skyblue1	135	206	255
skyblue2	126	192	238
skyblue3	108	166	205
skyblue4	74	112	139
lightskyblue1	176	226	255
lightskyblue2	164	211	238
lightskyblue3	141	182	205
lightskyblue4	96	123	139

slategray1	198	226	255
slategray2	185	211	238
slategray3	159	182	205
slategray4	108	123	139
lightsteelblue1	202	225	255
lightsteelblue2	188	210	238
lightsteelblue3	162	181	205
lightsteelblue4	110	123	139
lightblue1	191	239	255
lightblue2	178	223	238
lightblue3	154	192	205
lightblue4	104	131	139

lightcyan1	224	255	255
lightcyan2	209	238	238
lightcyan3	180	205	205
lightcyan4	122	139	139

paleturquoise1	187	255	255
paleturquoise2	174	238	238
paleturquoise3	150	205	205
paleturquoise4	102	139	139
cadetblue1	152	245	255
cadetblue2	142	229	238
cadetblue3	122	197	205
cadetblue4	83	134	139
turquoise1	0	245	255
turquoise2	0	229	238
turquoise3	0	197	205
turquoise4	0	134	139
cyan1	0	255	255
cyan2	0	238	238
cyan3	0	205	205
cyan4	0	139	139

darkslategray1	151	255	255
darkslategray2	141	238	238
darkslategray3	121	205	205
darkslategray4	82	139	139
aquamarine1	127	255	212
aquamarine2	118	238	198
aquamarine3	102	205	170
aquamarine4	69	139	116
darkseagreen1	193	255	193
darkseagreen2	180	238	180
darkseagreen3	155	205	155
darkseagreen4	105	139	105
seagreen1	84	255	159
seagreen2	78	238	148
seagreen3	67	205	128
seagreen4	46	139	87
palegreen1	154	255	154
palegreen2	144	238	144
palegreen3	124	205	124
palegreen4	84	139	84
springgreen1	0	255	127

springgreen2	0	238	118
springgreen3	0	205	102
springgreen4	0	139	69
green1	0	255	0
green2	0	238	0
green3	0	205	0
green4	0	139	0
chartreuse1	127	255	0
chartreuse2	118	238	0
chartreuse3	102	205	0
chartreuse4	69	139	0
olivedrab1	192	255	62
olivedrab2	179	238	58
olivedrab3	154	205	50
olivedrab4	105	139	34
darkolivegreen1	202	255	112
darkolivegreen2	188	238	104
darkolivegreen3	162	205	90
darkolivegreen4	110	139	61
khaki1	255	246	143
khaki2	238	230	133
khaki3	205	198	115
khaki4	139	134	78

lightgoldenrod1	255	236	139
lightgoldenrod2	238	220	130
lightgoldenrod3	205	190	112
lightgoldenrod4	139	129	76
lightyellow1	255	255	224
lightyellow2	238	238	209
lightyellow3	205	205	180
lightyellow4	139	139	122
yellow1	255	255	0
yellow2	238	238	0
yellow3	205	205	0
yellow4	139	139	0
gold1	255	215	0
gold2	238	201	0
gold3	205	173	0
gold4	139	117	0
goldenrod1	255	193	37
goldenrod2	238	180	34

goldenrod3	205	155	29
goldenrod4	139	105	20
darkgoldenrod1	255	185	15
darkgoldenrod2	238	173	14
darkgoldenrod3	205	149	12
darkgoldenrod4	139	101	8

rosybrown1	255	193	193
rosybrown2	238	180	180
rosybrown3	205	155	155
rosybrown4	139	105	105
indianred1	255	106	106
indianred2	238	99	99
indianred3	205	85	85
indianred4	139	58	58
sienna1	255	130	71
sienna2	238	121	66
sienna3	205	104	57
sienna4	139	71	38
burlywood1	255	211	155
burlywood2	238	197	145
burlywood3	205	170	125
burlywood4	139	115	85

wheat1	255	231	186
wheat2	238	216	174
wheat3	205	186	150
wheat4	139	126	102
tan1	255	165	79
tan2	238	154	73
tan3	205	133	63
tan4	139	90	43

chocolate1	255	127	36
chocolate2	238	118	33
chocolate3	205	102	29
chocolate4	139	69	19
firebrick1	255	48	48
firebrick2	238	44	44
firebrick3	205	38	38
firebrick4	139	26	26
brown1	255	64	64
brown2	238	59	59

brown3	205	51	51
brown4	139	35	35
salmon1	255	140	105
salmon2	238	130	98
salmon3	205	112	84
salmon4	139	76	57
lightsalmon1	255	160	122
lightsalmon2	238	149	114
lightsalmon3	205	129	98
lightsalmon4	139	87	66

orange1	255	165	0
orange2	238	154	0
orange3	205	133	0
orange4	139	90	0
darkorange1	255	127	0
darkorange2	238	118	0
darkorange3	205	102	0
darkorange4	139	69	0
coral1	255	114	86
coral2	238	106	80
coral3	205	91	69
coral4	139	62	47
tomato1	255	99	71
tomato2	238	92	66
tomato3	205	79	57
tomato4	139	54	38
orangered1	255	69	0
orangered2	238	64	0
orangered3	205	55	0
orangered4	139	37	0
red1	255	0	0
red2	238	0	0
red3	205	0	0
red4	139	0	0
deeppink1	255	20	147
deeppink2	238	18	137
deeppink3	205	16	118
deeppink4	139	10	80
hotpink1	255	110	180
hotpink2	238	106	167
hotpink3	205	96	144

hotpink4	139	58	98
pink1	255	181	197
pink2	238	169	184
pink3	205	145	158
pink4	139	99	108
lightpink1	255	174	185
lightpink2	238	162	173
lightpink3	205	140	149
lightpink4	139	95	101

palevioletred1	255	130	171
palevioletred2	238	121	159
palevioletred3	205	104	137
palevioletred4	139	71	93
maroon1	255	52	179
maroon2	238	48	167
maroon3	205	41	144
maroon4	139	28	98
violetred1	255	62	150
violetred2	238	58	140
violetred3	205	50	120
violetred4	139	34	82

magenta1	255	0	255
magenta2	238	0	238
magenta3	205	0	205
magenta4	139	0	139
orchid1	255	131	250
orchid2	238	122	233
orchid3	205	105	201
orchid4	139	71	137
plum1	255	187	255
plum2	238	174	238
plum3	205	150	205
plum4	139	102	139
mediumorchid1	224	102	255
mediumorchid2	209	95	238
mediumorchid3	180	82	205
mediumorchid4	122	55	139
darkorchid1	191	62	255
darkorchid2	178	58	238
darkorchid3	154	50	205
darkorchid4	104	34	139

purple1	155	48	255
purple2	145	44	238
purple3	125	38	205
purple4	85	26	139
mediumpurple1	171	130	255
mediumpurple2	159	121	238
mediumpurple3	137	104	205
mediumpurple4	93	71	139

thistle1	255	225	255
thistle2	238	210	238
thistle3	205	181	205
thistle4	139	123	139

gray0	0	0	0
grey0	0	0	0
gray1	3	3	3
grey1	3	3	3
gray2	5	5	5
grey2	5	5	5
gray3	8	8	8
grey3	8	8	8
gray4	10	10	10
grey4	10	10	10
gray5	13	13	13
grey5	13	13	13
gray6	15	15	15
grey6	15	15	15
gray7	18	18	18
grey7	18	18	18
gray8	20	20	20
grey8	20	20	20
gray9	23	23	23
grey9	23	23	23
gray10	26	26	26
grey10	26	26	26
gray11	28	28	28
grey11	28	28	28
gray12	31	31	31
grey12	31	31	31
gray13	33	33	33
grey13	33	33	33
gray14	36	36	36

grey14	36	36	36
gray15	38	38	38
grey15	38	38	38
gray16	41	41	41
grey16	41	41	41
gray17	43	43	43
grey17	43	43	43
gray18	46	46	46
grey18	46	46	46
gray19	48	48	48
grey19	48	48	48
gray20	51	51	51
grey20	51	51	51
gray21	54	54	54
grey21	54	54	54
gray22	56	56	56
grey22	56	56	56
gray23	59	59	59
grey23	59	59	59
gray24	61	61	61
grey24	61	61	61
gray25	64	64	64
grey25	64	64	64
gray26	66	66	66
grey26	66	66	66
gray27	69	69	69
grey27	69	69	69
gray28	71	71	71
grey28	71	71	71
gray29	74	74	74
grey29	74	74	74
gray30	77	77	77
grey30	77	77	77
gray31	79	79	79
grey31	79	79	79
gray32	82	82	82
grey32	82	82	82
gray33	84	84	84
grey33	84	84	84
gray34	87	87	87
grey34	87	87	87
gray35	89	89	89

grey35	89	89	89
gray36	92	92	92
grey36	92	92	92
gray37	94	94	94
grey37	94	94	94
gray38	97	97	97

grey38	97	97	97
gray39	99	99	99
grey39	99	99	99
gray40	102	102	102
grey40	102	102	102
gray41	105	105	105
grey41	105	105	105
gray42	107	107	107
grey42	107	107	107
gray43	110	110	110
grey43	110	110	110
gray44	112	112	112
grey44	112	112	112
gray45	115	115	115
grey45	115	115	115
gray46	117	117	117
grey46	117	117	117
gray47	120	120	120
grey47	120	120	120
gray48	122	122	122
grey48	122	122	122
gray49	125	125	125
grey49	125	125	125
gray50	127	127	127
grey50	127	127	127
gray51	130	130	130
grey51	130	130	130
gray52	133	133	133
grey52	133	133	133
gray53	135	135	135
grey53	135	135	135
gray54	138	138	138
grey54	138	138	138
gray55	140	140	140
grey55	140	140	140

gray56	143	143	143
grey56	143	143	143
gray57	145	145	145
grey57	145	145	145
gray58	148	148	148
grey58	148	148	148
gray59	150	150	150
grey59	150	150	150
gray60	153	153	153
grey60	153	153	153

gray61	156	156	156
grey61	156	156	156
gray62	158	158	158
grey62	158	158	158
gray63	161	161	161
grey63	161	161	161
gray64	163	163	163
grey64	163	163	163
gray65	166	166	166
grey65	166	166	166
gray66	168	168	168
grey66	168	168	168
gray67	171	171	171
grey67	171	171	171
gray68	173	173	173
grey68	173	173	173
gray69	176	176	176
grey69	176	176	176
gray70	179	179	179
grey70	179	179	179
gray71	181	181	181
grey71	181	181	181
gray72	184	184	184
grey72	184	184	184
gray73	186	186	186
grey73	186	186	186
gray74	189	189	189
grey74	189	189	189

gray75	191	191	191
grey75	191	191	191
gray76	194	194	194

grey76	194	194	194
gray77	196	196	196
grey77	196	196	196
gray78	199	199	199
grey78	199	199	199
gray79	201	201	201
grey79	201	201	201
gray80	204	204	204
grey80	204	204	204
gray81	207	207	207
grey81	207	207	207
gray82	209	209	209
grey82	209	209	209
gray83	212	212	212
grey83	212	212	212
gray84	214	214	214
grey84	214	214	214
gray85	217	217	217
grey85	217	217	217
gray86	219	219	219
grey86	219	219	219
gray87	222	222	222
grey87	222	222	222

gray88	224	224	224
grey88	224	224	224
gray89	227	227	227
grey89	227	227	227
gray90	229	229	229
grey90	229	229	229
gray91	232	232	232
grey91	232	232	232
gray92	235	235	235
grey92	235	235	235
gray93	237	237	237
grey93	237	237	237
gray94	240	240	240
grey94	240	240	240
gray95	242	242	242
grey95	242	242	242
gray96	245	245	245
grey96	245	245	245

gray97	247	247	247
grey97	247	247	247
gray98	250	250	250
grey98	250	250	250
gray99	252	252	252
grey99	252	252	252
gray100	255	255	255
grey100	255	255	255

dark grey	169	169	169
darkgrey	169	169	169
dark gray	169	169	169
darkgray	169	169	169
dark blue	0	0	139
darkblue	0	0	139
dark cyan	0	139	139
darkcyan	0	139	139
dark magenta	139	0	139
darkmagenta	139	0	139
dark red	139	0	0
darkred	139	0	0
light green	144	238	144
lightgreen	144	238	144

See Also

color , name2rgb , rgb2name

Name

colorbar — draw a colorbar

```
colorbar(umin, umax [, colminmax, fmt])
```

Parameters

umin

real scalar, the minimum value associated with the plot

umax

real scalar, the maximum value associated with the plot

colminmax

(optional) a vector with 2 integer components

fmt

(optional) a string to set up the display format for colorbar graduations

Description

Draw a colorbar for a plot3d, fec, Sgrayplot, etc... The function may be called **BEFORE** the plot3d, fec, Sgrayplot,... because its sets and changes the frame for the plot. This way the colorbar is not part of the associated plot and so is not modified by a zoom or a rotation.

The optional argument `colminmax` may be used to precise the first color (associated with `umin`) and the the last color (associated with `umax`) of the current colormap. By default `colminmax=[1 nb_colors]` where `nb_colors` is the number of colors of the current colormap.

The optional argument `fmt` is a string containing a C-format, like "%.2f", "%e", etc...

For the 2 optional arguments you can use the syntax `keyword=value` (see the last example).

Examples

```
// example 1
x = linspace(0,1,81);
z = cos(2*pi*x)'*sin(2*pi*x);
zm = min(z); zM = max(z);
clf()
xset("colormap",jetcolormap(64))
colorbar(zm,zM)
Sgrayplot(x,x,z)
xtitle("The function cos(2 pi x)sin(2 pi y)")

// example 2
x = linspace(0,1,81);
z = cos(2*pi*x)'*sin(2*pi*x);
zm = min(z); zM = max(z);
zz = abs(0.5*cos(2*pi*x)'*cos(2*pi*x));
zzm = min(zz); zzM = max(zz);
clf();
xset("colormap",jetcolormap(64))

drawlater() ;
subplot(2,2,1)
```



```
colorbar(zm,zM)
Sgrayplot(x,x,z, strf="031", rect=[0 0 1 1])
xtitle("a Sgrayplot with a colorbar")
subplot(2,2,2)
colorbar(zm,zM)
plot3d1(x,x,z)
xtitle("a plot3d1 with a colorbar")
subplot(2,2,3)
colorbar(zzm,zzM)
plot3d1(x,x,zz)
xtitle("a plot3d1 with a colorbar")
subplot(2,2,4)
colorbar(zzm,zzM)
Sgrayplot(x,x,zz, strf="031", rect=[0 0 1 1])
xtitle("a Sgrayplot with a colorbar")
drawnow() ;

// example 3
x = linspace(0,1,81);
zz = abs(0.5*cos(2*pi*x))*cos(2*pi*x);
zzm = min(zz); zzM = max(zz);
[xf,yf,zf]=genfac3d(x,x,zz);
nb_col = 64;
clf()
xset("colormap",hotcolormap(nb_col))
drawlater() ;
colorbar(zzm,zzM,[1, nb_col],fmt="%.1f")
nbcol = xget("lastpattern")
zcol = dsearch(zf, linspace(zzm, zzM, nb_col+1));
plot3d(xf, yf, list(zf, zcol), flag = [-2 6 4])
xtitle("a plot3d with shaded interpolated colors")
drawnow() ;
xselect()
```

See Also

[colormap](#)

Authors

B. Pincon, S. Steer

Name

`colordef` — Set default color values to display different color schemes

```
colordef(color_scheme)
colordef(f,color_scheme)
colordef('new',color_scheme)
```

Parameters

`color_scheme`

a character string with possible value: 'white', 'black', 'none'

`f`

a handle on a graphic figure

Description

- `colordef('white')` sets the default figure (see `gdf`) colormap to `jetcolormap(64)`, the default figure background color to light gray and the default axes (see `gda`) background color to white, the axes lines foreground and font color to black.
- `colordef('black')` sets the default figure (see `gdf`) colormap to `jetcolormap(64)`, the default figure background color to dark gray and the default axes (see `gda`) background color to black, the axes lines foreground and font color to white.
- `colordef('none')` sets the default figure (see `gdf`) colormap to `hsvcolormap(64)`, the default figure background color to dark gray and the default axes (see `gda`) background color to black, the axes lines foreground and font color to white.
- `colordef(f,color_scheme)` sets the color properties of figure given by the handle `f` as well as the color properties of its current axes.
- `colordef('new',color_scheme)` creates a new graphic window and its color properties as well as the properties of its axes.

Examples

See Also

`gdf`, `gda`, `figure_properties`, `axes_properties`

Authors

S. Steer
INRIA

Name

colormap — using colormaps

Description

A colormap `cmap` is defined by a $m \times 3$ matrix. m is the number of colors. Color number i is given as a 3-uple `cmap(i,1)`, `cmap(i,2)` `cmap(i,3)` corresponding respectively to red, green and blue intensity between 0 and 1.

At the beginning, 32 colors are defined in the colormap. You can change the colormap of a figure by using `set(f, "color_map", cmap)` where `f` is the handle of the figure.

Each color in the colormap has an id you have to use to specify color in most plot functions. To see the ids, use function `getcolor`.

The functions `hotcolormap`, `jetcolormap` and `graycolormap` provide colormaps with continuous variation of the colors.

You can get the default colormap by `cmap=get(sdf(), "color_map")`.

Examples

```
n=64;
r=linspace(0,1,n)';
g=linspace(1,0,n)';
b=ones(r);
cmap=[r g b];
f=gcf(); f.color_map=cmap;
plot3d1()
f.color_map=get(sdf(), "color_map");
```

See Also

`autumncolormap` , `bonecolormap` , `coolcolormap` , `coppercolormap` , `graycolormap` , `hotcolormap` , `hsvcolormap` , `jetcolormap` , `oceancolormap` , `pinkcolormap` , `rainbowcolormap` , `springcolormap` , `summercolormap` , `whitecolormap` , `wintercolormap` , `color` , `getcolor`

Name

Compound_properties — description of the Compound entity properties

Description

The Compound entity is a third of the graphics entities hierarchy. This entity defines interdependencies of the various graphics entities and their global visibility property.

parent:

This property contains the handle of the parent. The parent of the text entity should be of the type "Axes" or "Compound".

children:

A vector containing the handles of all graphics objects children of the Compound. These graphics objects can be of type "Compound", "Rectangle", "Polyline", "Patch", "Segs", "Arc", "Grayplot",...

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, value is "on" where graphics entities children of the Compound are drawn according to their visibility property. If "off" all children of Compound are not displayed on the screen.

user_data:

This field can be used to store any scilab variable in the figure data structure, and to retrieve it.

See Also

glue , unglue , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

contour — level curves on a 3D surface

```
contour(x,y,z,nz,[theta,alpha,leg,flag,ebox,zlev])
contour(x,y,z,nz,<opt_args>)
```

Parameters

x,y

two real row vectors of size n1 and n2.

z

real matrix of size (n1,n2), the values of the function or a Scilab function which defines the surface $z=f(x,y)$.

nz

the level values or the number of levels.

-

If **nz** is an integer, its value gives the number of level curves equally spaced from **zmin** to **zmax** as follows:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

Note that the **zmin** and **zmax** levels are not drawn (generically they are reduced to points) but they can be added with

```
[im,jm] = find(z == zmin);      // or zmax
plot2d(x(im)',y(jm)',-9,"000")
```

-

If **nz** is a vector, **nz(i)** gives the value of the *i*th level curve. Note that it can be useful in order to see **zmin** and **zmax** level curves to add an epsilon tolerance: **nz=[zmin+%eps, ..., zmax-%eps]**.

<opt_args>

a sequence of statements **key1=value1, key2=value2, ...** where keys may be **theta,alpha,leg,flag,ebox,zlev** (see below). In this case, the order has no special meaning.

theta, alpha

real values giving in degree the spherical coordinates of the observation point.

leg

string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

flag

a real vector of size three **flag=[mode, type, box]**.

mode

string representation mode.

mode=0:

the level curves are drawn on the surface defined by (x,y,z).

mode=1:
the level curves are drawn on a 3D plot and on the plan defined by the equation $z=z_{lev}$.

mode=2:
the level curves are drawn on a 2D plot.

type
an integer (scaling).

type=0
the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3
3d isometric with box bounds given by optional `ebox`, similarly to `type=1`

type=4
3d isometric bounds derived from the data, to similarly `type=2`

type=5
3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`

type=6
3d expanded isometric bounds derived from the data, similarly to `type=2`

box
an integer (frame around the plot).

box=0
nothing is drawn around the plot.

box=1
unimplemented (like `box=0`).

box=2
only the axes behind the surface are drawn.

box=3
a box surrounding the surface is drawn and captions are added.

box=4
a box surrounding the surface is drawn, captions and axes are added.

ebox
used when `type` in `flag` is 1. It specifies the boundaries of the plot as the vector `[xmin, xmax, ymin, ymax, zmin, zmax]`.

zlev
real number.

Description

`contour` draws level curves of a surface $z=f(x,y)$. The level curves are drawn on a 3D surface. The optional arguments are the same as for the function `plot3d` (except `zlev`) and their meanings are

the same. They control the drawing of level curves on a 3D plot. Only `flag(1)=mode` has a special meaning.

`mode=0`

the level curves are drawn on the surface defined by (x,y,z) .

`mode=1`

the level curves are drawn on a 3D plot and on the plan defined by the equation $z=z_{lev}$.

`mode=2`

the level curves are drawn on a 2D plot.

You can change the format of the floating point number printed on the levels by using `xset("fpf",string)` where `string` gives the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format and Use `string=" "` to suppress printing.

Usually we use `contour2d` to draw levels curves on a 2D plot.

Enter the command `contour( )` to see a demo.

Examples

```
t=linspace(-%pi,%pi,30);
function z=my_surface(x,y),z=x*sin(x)^2*cos(y),endfunction

contour(t,t,my_surface,10)
// changing the format of the printing of the levels
xset("fpf","%.1f")
clf()
contour(t,t,my_surface,10)
// 3D
clf()
z=feval(t,t,my_surface);
plot3d(t,t,z);contour(t,t,z+0.2*abs(z),20,flag=[0 2 4]);
```

See Also

`contour2d` , `plot3d`

Authors

J.Ph.C.

Name

contour2d — level curves of a surface on a 2D plot

```
contour2d(x,y,z,nz,[style,strf,leg,rect,nax])  
contour2d(x,y,z,nz,<opt_args>)
```

Parameters

x,y

two real row vectors of size n1 and n2: the grid.

z

real matrix of size (n1,n2), the values of the function on the grid or a Scilab function which defines the surface $z=f(x,y)$.

nz

the level values or the number of levels.

If **nz** is an integer, its value gives the number of level curves equally spaced from **zmin** to **zmax** as follows:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

Note that the **zmin** and **zmax** levels are not drawn (generically they are reduced to points) but they can be added with

```
[im,jm] = find(z == zmin); // or zmax  
plot2d(x(im)',y(jm)',-9,"000")
```

If **nz** is a vector, **nz(i)** gives the value of the *i*th level curve.

<opt_args>

This represents a sequence of statements **key1=value1, key2=value2,...** where **key1, key2, . . .** can be one of the following: **style, leg, rect, nax, strf** or **axesflag** and **frameflag** (see **plot2d**)

style,strf,leg,rect,nax

see **plot2d**. The argument **style** gives the dash styles or colors which are to be used for level curves. It must have the same size as the number of levels.

Description

contour2d draws level curves of a surface $z=f(x,y)$ on a 2D plot. The values of $f(x,y)$ are given by the matrix **z** at the grid points defined by **x** and **y**.

You can change the format of the floating point number printed on the levels by using **xset("fpf",string)** where **string** gives the format in C format syntax (for example **string="%.3f"**). Use **string=""** to switch back to default format and Use **string=" "** to suppress printing. This last feature is useful in conjunction with legends to display the level numbers in a legend and not directly onto the level curves as usual (see Examples).

The optional arguments **style, strf, leg, rect, nax**, can be passed by a sequence of statements **key1=value1, key2=value2, ...** where keys may be **style, strf, leg, rect, nax**. In this case, the order has no special meaning.

Use `contour` to draw levels curves on a 3D surface.

Examples

```
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])
// changing the format of the printing of the levels
xset("fpf", "%.2f")
clf()
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])

// now an example with level numbers drawn in a legend
// Caution there are a number of tricks...
x = linspace(0,4*pi,80);
z = cos(x')*cos(x);
clf(); f=gcf();
xset("fpf", " ") // trick 1: this implies that the level numbers are not
                  // drawn on the level curves
f.color_map=jetcolormap(7);
// trick 2: to be able to put the legend on the right without
//           interfering with the level curves use rect with a xmax
//           value large enough
contour2d(x,x,z,-0.75:0.25:0.75,frameflag=3,rect=[0,0,5*pi,4*pi])
// trick 3: use legends (note that the more practical legend function
//           will not work as soon as one of the level is formed by 2 curves)
legends(string(-0.75:0.25:0.75),1:7,"lr");
xtitle("Some level curves of the function cos(x)cos(y)")
```

See Also

`contour` , `contour2di` , `plot2d`

Authors

J.Ph.C.

Name

contour2di — compute level curves of a surface on a 2D plot

```
[xc,yc]=contour2di(x,y,z,nz)
```

Parameters

x,y

two real row vectors of size n1 and n2: the grid.

z

real matrix of size (n1,n2), the values of the function.

nz

the level values or the number of levels.

If nz is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

Note that the zmin and zmax levels are not drawn (generically they are reduced to points) but they can be added with

```
[im,jm] = find(z == zmin);      // or zmax
plot2d(x(im)',y(jm)',-9,"000")
```

If nz is a vector, nz(i) gives the value of the ith level curve.

xc,yc

vectors of identical sizes containing the contours definitions. See below for details.

Description

contour2di computes level curves of a surface $z=f(x,y)$ on a 2D plot. The values of $f(x,y)$ are given by the matrix Z at the grid points defined by x and y.

xc(1) contains the level associated with first contour path, yc(1) contains the number N1 of points defining this contour path and (xc(1+(1:N1)), yc(1+(1:N1))) contain the coordinates of the paths points. The second path begin at xc(2+N1) and yc(2+N1) and so on.

Examples

```
[xc,yc]=contour2di(1:10,1:10,rand(10,10),5);
k=1;n=yc(k);c=1;
while k+yc(k)<size(xc,'*')
    n=yc(k);
    plot2d(xc(k+(1:n)),yc(k+(1:n)),c)
    c=c+1;
    k=k+n+1;
end
```

See Also

`contour` , `fcontour` , `fcontour2d` , `contour2d` , `plot2d` , `xset`

Authors

J.Ph.C.

Name

contourf — filled level curves of a surface on a 2D plot

```
contourf(x,y,z,nz,[style,strf,leg,rect,nax])
```

Parameters

x,y

two real row vectors of size n1 and n2: the grid.

z

real matrix of size (n1,n2), the values of the function.

nz

the level values or the number of levels.

-

If nz is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

Note: that the zmin and zmax levels are not drawn (generically they are reduced to points) but they can be added with

```
[im,jm] = find(z == zmin);      // or zmax
plot2d(x(im)',y(jm)',-9,"000")
```

-

If nz is a vector, nz(i) gives the value of the ith level curve.

style,strf,leg,rect,nax

see plot2d. The argument style gives the colors which are to be used for level curves. It must have the same size as the number of levels.

Description

contourf paints surface between two consecutives level curves of a surface $z=f(x,y)$ on a 2D plot. The values of $f(x,y)$ are given by the matrix Z at the grid points defined by x and y.

You can change the format of the floating point number printed on the levels by using `xset("fpf",string)` where string gives the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format.

Enter the command `contourf()` to see a demo.

Examples

```
contourf(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11])

function z=peaks(x,y)
x1=x(:).*ones(1,size(y,'*'));

```

```
y1=y(:)'.*ones(size(x, ' '),1);
z = (3*(1-x1).^2).*exp(-(x1.^2) - (y1+1).^2) ...
    - 10*(x1/5 - x1.^3 - y1.^5).*exp(-x1.^2-y1.^2) ...
    - 1/3*exp(-(x1+1).^2 - y1.^2)
endfunction

function z=peakit()
x=-4:0.1:4;y=x;z=peaks(x,y);
endfunction

z=peakit();

levels=[-6:-1, -logspace(-5,0,10), logspace(-5,0,10),1:8];
m=size(levels, ' ');
n = fix(3/8*m);
r = [(1:n)'/n; ones(m-n,1)];
g = [zeros(n,1); (1:n)'/n; ones(m-2*n,1)];
b = [zeros(2*n,1); (1:m-2*n)'/(m-2*n)];
h = [r g b];
xset('colormap',h);
xset('fpf', ' ');
clf();
contourf([],[],z,[-6:-1, -logspace(-5,0,10), logspace(-5,0,10),1:8],0*ones(1,m))

xset('fpf', '');
clf();
contourf([],[],z,[-6:-1, -logspace(-5,0,10), logspace(-5,0,10),1:8]);
```

See Also

[contour](#), [fcontour](#), [fcontour2d](#), [contour2di](#), [plot2d](#), [xset](#)

Authors

J.Ph.C.

Name

coolcolormap — cyan to magenta colormap

```
cmap=coolcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

coolcolormap computes a colormap with n colors varying from cyan to magenta.

Examples

```
f = scf();  
plot3d1();  
f.color_map = coolcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

coppercolormap — black to a light copper tone colormap

```
cmap=coppercolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

coppercolormap computes a colormap with *n* colors varying from black to a light copper tone.

Examples

```
f = scf();  
plot3d1();  
f.color_map = coppercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

copy — copy a graphics entity.

```
copy(h)
copy(h, h_axes)
h_copy=copy(h)
```

Parameters

h
a handle, the handle of the entity to copy.

h_axes
a handle, the handle of the parent entity for the destination. It should be an axes entity.

h_copy
a handle, the handle on the entity copied.

Description

This routine can be used to make a copy of a graphics entity with all its properties'values, it returns the handle on this new entity. **h_axes** omitted, graphics entity is cloned and it's copied in the same parent axes entity.

Examples

```
subplot(211);a1=gca();
plot2d()
e=gce();
p1=e.children(1);
p2=copy(p1);p2.data(:,2)=p2.data(:,2)-0.5;
subplot(212);a2=gca();
a2.data_bounds=a1.data_bounds;
copy(p1,a2);
```

See Also

get , set , delete , move , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

delete — delete a graphic entity and its children.

```
delete(h)
delete(h,"callback")
delete()
delete("all")
```

Parameters

h

a handle, the handle of the entity to delete. **h** can be a vector of handles, in which case all objects identified by **h(i)** will be deleted.

"all"

string keyword (optional).

Description

This routine can be used to delete a graphics entity identified by the handle given as argument. In this case, All children of this graphics entity will be deleted. Without any argument **delete** removes the current entity. With **"all"** argument it deletes all the entities of the current figure.

The **"callback"** argument is not yet handled.

Examples

```
subplot(211);
t=1:10;plot2d(t,t.^2),
subplot(223);
plot3d();
subplot(224);
plot2d();
xfrect(1,0,3,1);
a=get("current_axes")
delete(); //delete the graphics object newly created
delete(a.children); //delete all children of the current axes
delete(a); //delete the axes
delete("all"); //delete all the graphics objects of the figure
```

See Also

[get](#) , [set](#) , [copy](#) , [move](#) , [is_handle_valid](#) , [graphics_entities](#)

Authors

Djalel ABDEMOUCHE

Name

dragrect — Drag rectangle(s) with mouse

```
[final_rect, btn]=dragrect(initial_rect)
```

Parameters

initial_rect

4 4xn matrix containing the initial rectangles definition. Each column contains [x_left; y_top; width; height]. If only one rectangle is present the initial_rect may also be a vector.

final_rect

a rectangle defined by [x_left, y_top, width, height]

btn

an integer, the number of the mouse button clicked

Description

dragrect drags one or more rectangles anywhere on the screen. The 4xn matrix rect defines the rectangles. Each column of initial_rect must contain the initial rectangle position as [left;top;width;height] values. When a button is clicked dragrect returns the final rectangles definition in final_Rect.

Examples

```
xsetech(frect=[0,0,100,100])  
r=dragrect([10;10;30;10])  
xrect(r)
```

See Also

xrect , xrects , xclick , xgetmouse

Name

draw — draw an entity.

```
draw()  
draw(h)
```

Parameters

h

a handle, the handle on an entity to draw. h can be a vector of handles, in which case all objects identified by h(i) will be drawn.

Description

This function can be used to draw entities specified by h even if its parent figure `immediate_drawing` property is "off". If no argument is specified, the current object is drawn. Note that the window must not be resized ; if not, those objects are hidden back.

Examples

```
subplot(212)  
a=gca();  
plot2d  
drawlater  
  
subplot(211)  
plot2d1 // default drawing mode  
  
e=gce();  
draw(e.children(2)) // draw a single polyline of the second axes  
  
e.children(1).visible='off'; // We can choose to make a line invisible  
  
draw(e) // draw Compound and its children <=> draw all the visible polylines  
  
drawnow // ...now!  
  
e.children(1).visible='on';
```

See Also

`get` , `set` , `drawnow` , `drawlater` , `graphics_entities`

Authors

Djalel ABDEMOUCHE, F.Leray

Name

drawaxis — draw an axis

```
drawaxis([options])  
// options: x,y,dir,sub_int,fontsize,format_n,seg,textcolor,ticscolor,tics
```

Parameters

dir=string

used to specify the tics direction. **string** can be chosen among 'u','r','d','l' and 'l' is the default value. the values 'u','r','d','l' stands respectively for up, right, down, left

tics=string

A flag which describes how the tics are given. **string** can be chosen among 'v','r', and 'i', and, 'v' is the default value

x,y

two vectors which give tics positions.

val= string matrix

A string matrix, which, when given, gives the string to be drawn along the axis at tics positions.

fontsize=int

specifies the fontsize to use for displaying values along the axis. Default value is -1 which stands for current fontsize

format_n=string

format to use for displaying numbers along the axis, where **string** gives the format according to the C language printf function format syntax (for example **string**="%.3f").

seg= 1 or 0

A flag which controls the display of the base segment of the axis (default value is 1).

sub_int=integer

an integer which gives the number of sub-intervals to draw between large tics.

textcolor=integer

specify the color to use for displaying values along the axis. Default value is -1 which stands for current color.

ticscolor=integer

specify the color to use for tics drawing. Default value is -1 which stands for current color.

Description

drawaxis is used to draw an axis in vertical or horizontal direction. the direction of the axis is given by **dir** **dir** = 'u' or 'd' gives a horizontal axis with tics going up ('u') or down ('d'). **dir** = 'r' or 'l' give a vertical axis with tics going right ('r') or left ('l').

x and **y** give the axis tics positions. If the axis is horizontal then **y** must be a scalar or can be omitted and **x** is a Scilab vector. The meaning of **x** is controlled by **tics**.

If **tics**='v' then **x** gives the tics positions along the x-axis.

If **tics**='r' then **x** must be of size 3. **x**=[**xmin**, **xmax**, **n**] and **n** gives the number of intervals.

If **tics**='i' then **x** must be of size 4, **x**=[**k1**, **k2**, **a**, **n**]. then **xmin**=**k1***10^a, **xmax**=**k2***10^a and **n** gives the number of intervals

If `y` is omitted then the axis will be positioned at the top of the frame if `dir='u'` or at the bottom if `dir='d'`

By default, numbers are drawn along the axis. They are drawn using a default format which can be changed with `format_n`. It is also possible to display given strings and not numbers, this is done if `val` is provided. The size of `val` must match the number of tics.

Examples

```
plot2d(1:10,1:10,1,"020")
// horizontal axis
drawaxis(x=2:7,y=4,dir='u',tics='v')
// horizontal axis on top of the frame
drawaxis(x=2:7,dir='u',tics='v')
// horizontal axis at the bottom of the frame
drawaxis(x=2:7,dir='d',tics='v')

// horizontal axis given by a range
drawaxis(x=[2,7,3],y=4,dir='d',tics='r')

// vertical axis
drawaxis(x=4,y=2:7,dir='r',tics='v')
drawaxis(x=2,y=[2,7,3],dir='l',tics='r')
drawaxis(y=2:7,dir='r',tics='v')
drawaxis(y=2:7,dir='l',tics='v')

// horizontal axis with strings displayed at tics positions
drawaxis(x=2:7,y=8,dir='u',tics='v',val='A'+string(1:6));
// vertical axis with strings displayed at tics positions
drawaxis(x=8,y=2:7,dir='r',tics='v',val='B'+string(1:6));

// horizontal axis given with a 'i' range.
drawaxis(x=[2,5,0,3],y=9,dir='u',tics='i');
drawaxis(x=9,y=[2,5,0,3],dir='r',tics='i',sub_int=5);

// horizontal axis again
drawaxis(x=2:7,y=4,dir='u',tics='v',fontsize=10,textcolor=9,ticscolor=7,seg=0,s
```

See Also

[axis_properties](#)

Authors

J.Ph.C.

Name

drawlater — makes axes children invisible.

```
drawlater()
```

Description

This function can be used not to display immediately onto the `current` figure the next created graphics objects - i.e. by calling high level functions such as plotting functions or setting properties to existent objects. The `immediate_drawing` property of the current figure is set to `'off'` in order to postpone the next drawings.

It can specially be used with the `drawnow` function.

To enable back the `immediate_drawing` for the current figure, you can use `drawnow` function.

Warning : note that between `drawlater` and `drawnow` calls, the current figure may have changed. Therefore, this must be used carefully.

Examples

```
//Example : one axes / one figure
drawlater();
xfarc(.25,.55,.1,.15,0,64*360);
xfarc(.55,.55,.1,.15,0,64*360);
xfrect(.3,.8,.3,.2);
xfrect(.2,.7,.5,.2);
xfrect(.32,.78,.1,.1);
xfrect(.44,.78,.14,.1);
xfrect(-.2,.4,1.5,.8);
xstring(0.33,.9,"A Scilab Car");
a=get("current_axes");
a.children(1).font_size=4;
a.children(1).font_style=4;
a.children(1).background=5;
a.children(3).background=8;
a.children(4).background=8;
a.children(5).background=17;
a.children(6).background=17;
a.children(7).background=25;
a.children(8).background=25;
xclick();drawnow();

//Example 2:: two axes / one figure

subplot(212)
a=gca();
drawlater // what will be present in this axes will be displayed later
plot2d // draw these axes and children later...

subplot(211) // Warning: we change the axes
plot2d1 // default drawing mode

draw(a) // ...axes only is visible providing you not redraw the window
drawnow() // all is visible
```

See Also

get , set , drawnow , graphics_entities

Authors

Djalel ABDEMOUCHE, F.Leray

Name

drawnow — draw hidden graphics entities.

```
drawnow()
```

Description

Considering the current figure, this function can be used to draw the hidden graphics entities and all its children, that may have been postponed by a previous call to `drawlater`. The `immediate_drawing` property of the current `figure` is set to "on" .

It can specially be used with the `drawlater` function.

Examples

```
f=get("current_figure") // handle of the current figure

drawlater()
subplot(221);
t=1:10;plot2d(t,t.^2)
subplot(222);
x=0:1:7;plot2d(x,cos(x),2)
subplot(234);
plot2d(t,cos(t),3);
subplot(235);
plot2d(x,sin(2*x),5);
subplot(236);
plot2d(t,tan(2*t));

draw(gca()); //draw the current axes and its children
draw(f.children([3 4])); // draw the specified axes and their children
drawnow(); // draw the current figure
```

See Also

`get` , `set` , `drawlater` , `graphics_entities`

Authors

Djalel ABDEMOUCHE, F.Leray

Name

`edit_curv` — interactive graphic curve editor

```
[x,y,ok,gc] = edit_curv(y)
[x,y,ok,gc] = edit_curv(x,y)
[x,y,ok,gc] = edit_curv(x,y,job)
[x,y,ok,gc] = edit_curv(x,y,job,tit)
[x,y,ok,gc] = edit_curv(x,y,job,tit,gc)
```

Parameters

`x`
vector of x coordinates

`y`
vector of y coordinates

`job`
a character string formed by one to three of the characters 'a','x','y'

 'a'
 to add points to the edited curve

 'x'
 to modify x coordinates of the edited curve points

 'y'
 to modify y coordinates of the edited curve points

`tit`
a vector of three character strings which give the curve legend

`gc`
a list of graphic window parameters: `gc=list(rect,nax)`

`rect`
 bounds of the graphics (see `plot2d` for details)

`nax`
 graduation parameters (see `plot2d` for details)

`ok`
indicator if `ok==%t` user as returned with 'ok' menu else user as returned with 'abort' menu : list
(graphical objects created under `edit_curv`)

Description

`edit_curv` is an interactive graphic curve editor. To add a new point simply click at the desired location, the added point will be connected to the nearest end-point. to move a point click on it, drag the mouse to the new position and click to set the new position

Authors

Serge Steer ; ; ; ;

Name

errbar — add vertical error bars on a 2D plot

```
errbar(x,y,em,ep)
```

Parameters

x,y,em,ep
four matrices of the same size.

Description

errbar adds vertical error bars on a 2D plot. x and y have the same meaning as in plot2d. em(i,j) and ep(i,j) stands for the error interval on the value y(i,j): [y(i,j)-em(i,j),y(i,j)+ep(i,j)].

Enter the command errbar() to see a demo.

Examples

```
t=[0:0.1:2*pi]';  
y=[sin(t) cos(t)]; x=[t t];  
plot2d(x,y)  
errbar(x,y,0.05*ones(x),0.03*ones(x))
```

See Also

plot2d

Authors

J.Ph.C.

Name

eval3d — values of a function on a grid

```
[z]=eval3d(fun,x,[y])
```

Parameters

fun

function accepting vectors as arguments.

x,y

2 vectors of size (1,n1) and (1,n2). (default value for y : y=x).

z

matrix of size (n1,n2).

Description

This function returns a matrix $z(n1,n2)$. $z(i,j)=fun(x(i),y(j))$. If the function `fun` doesn't accept arguments of type vector use the primitive `feval`.

Examples

```
x=-5:5;y=x;
deff(' [z]=f(x,y)', ['z= x.*y']);
z=eval3d(f,x,y);
plot3d(x,y,z);

deff(' [z]=f(x,y)', ['z= x*y']);
z=feval(x,y,f);
plot3d(x,y,z);
```

See Also

[feval](#)

Authors

Steer S.; ;

Name

eval3dp — compute facets of a 3D parametric surface

```
[Xf,Yf,Zf]=eval3dp(fun,p1,p2)
```

Parameters

Xf,Yf,Zf

matrices of size (4,n-1*m-1). Xf(:,i), Yf(:,i) and Zf(:,i) are respectively the x-axis, y-axis and z-axis coordinates of the 4 points of the ith four sided facet.

fun

a Scilab function.

p1

a vector of size n.

p2

a vector of size m.

Description

eval3dp computes a four sided facets representation of a 3D parametric surface defined by the function fun. fun(p1,p2) computes the x-axis,y-axis and z-axis coordinates of the corresponding points on the surface, as [x(i),y(i),z(i)]=fun(p1(i),p2(i)). This is used for efficiency.

Examples

```
p1=linspace(0,2*pi,10);
p2=linspace(0,2*pi,10);
deff("[x,y,z]=scp(p1,p2)","x=p1.*sin(p1).*cos(p2);..
                           \"y=p1.*cos(p1).*cos(p2);..
                           \"z=p1.*sin(p2)\"");
[Xf,Yf,Zf]=eval3dp(scp,p1,p2);
plot3d(Xf,Yf,Zf)
```

See Also

genfac3d, plot3d

Name

event handler functions — Prototype of functions which may be used as event handler.

```
event_handler_function(win,x,y,ibut)
```

Parameters

win

window number where the event had occurred.

x

X coordinate in pixels of the mouse pointer when the events occurred.

y

Y coordinate in pixels of the mouse pointer when the events occurred.

ibut

number corresponding to the event type.

Description

When the event handler mode is enable, Scilab will call the specified event handler function each time an event occurs in the graphic window. The event handler function must comply with the above prototype. Each time an event occurred, the function is called and its four parameters are set accordingly to the window number, mouse position and type of event.

The `ibut` parameter takes one of the following value depending on event type:

`ibut==0`

Left mouse button has been pressed

`ibut==1`

Middle mouse button has been pressed

`ibut==2`

Right mouse button has been pressed

`ibut==3`

Left mouse button has been clicked

`ibut==4`

Middle mouse button has been clicked

`ibut==5`

Right mouse button has been clicked

`ibut==10`

Left mouse button has been double-clicked

`ibut==11`

Middle mouse button has been double-clicked

`ibut==12`

Right mouse button has been double-clicked

`ibut==-5`

Left mouse button has been released

```
ibut==4
    Middle mouse button has been released

ibut==3
    Right mouse button has been released

ibut==1
    mouse pointer has moved

ibut >=32
    key with ascii code ascii(ibut) has been pressed

ibut <=32
    key with ascii code ascii(-ibut) has been released

ibut >=1000+32
    key with ascii code ascii(ibut-1000) has been pressed while CTRL key pressed

ibut==1000
    graphic window has been closed
```

For example, let say that the name of the event handler function is `fooHandler` for window number 0. A left click in the window at position [100,150] (in pixels) will be equivalent as calling `fooHandler( 0, 100, 150, 3 )`.

See `figure_properties` or `seteventhandler` for information on how to specify the `event_handler` name.

Examples

```
function my_eventhandler(win,x,y,ibut)
    if ibut==1000 then return,end
    [x,y]=xchange(x,y,'i2f')
    xinfo(sprintf('Event code %d at mouse position is (%f,%f)',ibut,x,y))
endfunction

plot2d()
fig = gcf() ;
fig.event_handler = 'my_eventhandler' ;
fig.event_handler_enable = "on" ;
//now:
// - move the mouse over the graphic window
// - press and release keys shifted or not with Ctrl pressed or not
// - press button, wait a little release
// - press and release button
// - double-click button

fig.event_handler_enable = "off" ; //suppress the event handler
```

See Also

`figure_properties` , `seteventhandler` , `xgetmouse` , `xclick`

Authors

Jean-Baptiste Silvy

Name

fac3d — 3D plot of a surface (obsolete)

```
fac3d(x,y,z,[theta,alpha,leg,flag,ebox])  
fac3d1(x,y,z,[theta,alpha,leg,flag,ebox])
```

Description

These functions are obsolete and have been replaced by `plot3d` and `plot3d1`.

See Also

`plot3d` , `plot3d1`

Name

fchamp — direction field of a 2D first order ODE

```
fchamp(f,t,xr,yr,[arfact,rect,strf])  
fchamp(f,t,xr,yr,<opt_args>)
```

Parameters

f

An external (function or character string) or a list which describes the ODE.

-

It can be a function name **f**, where **f** is supposed to be a function of type $y=f(t, xy [p1, \dots, pn])$. **f** returns a column vector of size 2, **y**, which gives the value of the direction field **f** at point $xy=[x, y]$ and at time **t**.

-

It can also be an object of type list, `list(f,P1,...,Pn)` where **f** is a function of type $y=f(t, xy, p1, \dots, pn)$ and **Pi** gives the value of the parameter **pi**.

t

The selected time.

xr,yr

Two row vectors of size **n1** and **n2** which define the grid on which the direction field is computed.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2, ...` where **key1**, **key2**, ... can be one of the following: **arfact**, **rect**, **strf** (see below).

arfact,rect,strf

Optional arguments, see **champ**.

Description

fchamp is used to draw the direction field of a 2D first order ODE defined by the external function **f**. Note that if the ODE is autonomous, argument **t** is useless, but it must be given.

Enter the command **fchamp()** to see a demo.

Examples

```
deff("[xdot] = derpol(t,x)",..  
    ["xd1 = x(2)";..  
    "xd2 = -x(1) + (1 - x(1)**2)*x(2)";..  
    "xdot = [ xd1 ; xd2 ]")  
xf= -1:0.1:1;  
yf= -1:0.1:1;  
fchamp(derpol,0,xf,yf)  
clf()  
fchamp(derpol,0,xf,yf,1,[-2,-2,2,2],"011")
```

See Also

champ, **champ1**

Authors

J.Ph.C.

Name

fcontour — level curves on a 3D surface defined by a function

```
fcontour(xr,yr,f,nz,[theta,alpha,leg,flag,ebox,zlev])  
fcontour(xr,yr,f,nz,<opt_args>)
```

Description

This function is obsolete. It is now included in the contour function.

Authors

J.Ph.C.

Name

fcontour2d — level curves of a surface defined by a function on a 2D plot

```
fcontour2d(xr, yr, f, nz, [style, strf, leg, rect, nax])  
fcontour2d(xr, yr, f, nz, <opt_args>)
```

Description

This function is obsolete. It is now included in the contour2d function.

Authors

J.Ph.C.

Name

fec — pseudo-color plot of a function defined on a triangular mesh

```
fec(x,y,triangles,func,<opt_args>)
fec(x,y,triangles,func,[strf,leg,rect,nax,zminmax,colminmax,colout,mesh])
```

Parameters

x,y

two vectors of size n , $(x(i), y(i))$ gives the coordinates of node i

func

a vector of size n : $func(i)$ gives the value at node i of the function for which we want the pseudo-color plot.

triangles

is a $[Ntr, 5]$ matrix. Each line of `triangles` specifies a triangle of the mesh `triangle(j) = [number, node1, node2, node3, flag]`. `node1, node2, node3` are the number of the nodes which constitutes the triangle. `number` is the number of the triangle and `flag` is an integer not used in the `fec` function

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `strf, leg, rect, nax, zminmax, colminmax, colout, mesh` (see `plot2d` for the 4 first ones).

strf,leg,rect,nax

see `plot2d`

zminmax

vector with 2 components $[zmin\ zmax]$ (useful in particular for animation)

colminmax

vector of 2 positives integers $[colmin\ colmax]$

colout

vector of 2 integers $[under\_min\_col\ upper\_max\_col]$

mesh

boolean scalar, default value %f (must be true if you want also display the mesh)

Description

This function is the good one to draw linear triangular finite element solutions or simply to display a function defined on a triangulation. The color interpolation is done through software computation and so it is not too fast.

The function `colorbar` may be used to see the color scale (see the example section).

The `zminmax` argument gives the z values associated with the first and the last color (of the current colormap). More exactly if the colormap have nc colors and if we note $dz = (zmax-zmin)/nc$, then the part of the triangulation where $zmin + (i-1)dz \leq z < zmin + i\ dz$ is filled with the color i . By default $zmin = \min(func)$ and $zmax = \max(func)$. If you want to do an animation with `func` values that varie in time, take for `zmin` and `zmax` the global minimum and maximum or something close.

The `colout` argument lets the user choosing the colors for the 2 extremes regions $\{func < zmin\}$ and $\{func > zmax\}$, `under_min_col` and `upper_max_col` may be equal (independantly) to:

-1
in this case the same color than in the neighbouring zone is used (CAUTION: you don't see that the corresponding limit is crossed), this is the default case.

0
in this case the extreme region is not painting at all.

k
(k being a valid index to a color of the current colormap) the extreme region is painting in color k.

If you don't want to use the complete colormap you may use the `colminmax` argument with $1 \leq \text{colmin} < \text{colmax} \leq \text{nc}$ (nc being the number of colors of the current colormap) so as to use only the `[colmin,colmax]` sub-part of the colormap. (by default all the colors of the colormap are used).

See the demo files `demos/fec`:

`fec.ex1` is a simple demo file in which a mesh and a function on that mesh is completely built in Scilab syntax

`fec.ex2` is an example for which the mesh and the function value where computed by an external mesh builder (amdba type mesh) and an external program. A set of macros (provided in file `macros.sci`) can be used to read the data files in Scilab and plot the results.

Examples

```
// define a mini triangulation (4 vertices, 2 triangles)
x = [0 1 0 -1];
y = [0 0 1 1];
T = [1 1 2 3 1;
     2 3 4 1 1];
z = [0 1 0 -1]; // values of the func at each vertices
clf()
xset("colormap",jetcolormap(64))
subplot(1,2,1)
colorbar(-1,1)
fec(x,y,T,z,strf="040",mesh=%t)
xtitle("fec example (with the mesh)")
subplot(1,2,2)
colorbar(-1,1)
fec(x,y,T,z,strf="040") // rmq: mesh=%f by default
xtitle("fec example (without the mesh)")
xselect()

// this example shows the effect of zminmax and uses the
// previous example datas (you have to execute the it before)
clf()
xset("colormap",jetcolormap(64))
colorbar(-0.5,0.5) // be careful colorbar must be set by hands !
fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], mesh=%t)
xtitle("fec example : using zminmax argument")
xselect()

// this example shows the effect of zminmax and colout. It uses
// also the datas of the first example (you have to execute the it before)
clf()
xset("colormap",jetcolormap(64))
subplot(2,2,1)
colorbar(-0.5,0.5)
```

```

    fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 0], mesh=%t)
    xtitle("fec example : using zminmax and colout =[0 0]")
subplot(2,2,2)
    colorbar(-0.5,0.5)
    fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[32 32], mesh=%t)
    xtitle("fec example : using zminmax and colout =[32 32]")
subplot(2,2,3)
    colorbar(-0.5,0.5)
    fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[-1 0], mesh=%t)
    xtitle("fec example : using zminmax and colout =[-1 0]")
subplot(2,2,4)
    colorbar(-0.5,0.5)
    fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 -1], mesh=%t)
    xtitle("fec example : using zminmax and colout =[0 -1]")
xselect()

// this example shows a feature from colminmax:
// playing with 2 colormaps for 2 subplots. It
// uses also the data of the first example.
clf()
xset("colormap",[hotcolormap(64);jetcolormap(64)])
subplot(1,2,1)
    colorbar(-1,1,[1 64])
    fec(x,y,T,z,strf="040", colminmax=[1 64], mesh=%t)
    xtitle("fec using the hot colormap")
subplot(1,2,2)
    colorbar(-1,1,[65 128])
    fec(x,y,T,z,strf="040", colminmax=[65 128], mesh=%t)
    xtitle("fec using the jet colormap")
xselect()

```

See Also

colorbar , Sfgrayplot , Sgrayplot

Name

fec_properties — description of the fec entities properties

Description

The Fec entity is a leaf of the graphics entities hierarchy. It represents 2D finite elements plots (see `fec`, `Sgrayplot`).

parent:

This property contains the handle of the parent. The parent of the fec entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the children of the handle. However, `Fec` handles currently do not have any children.

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, the plot is visible, the value's property is "on". If "off" the plot is not drawn on the screen.

data:

This is a three column matrix $[x, y, f]$, where $x(i)$ and $y(i)$ are the coordinates of the i 'th node. $f(i)$ is the value associated to the node i .

triangles:

This is a five column matrix $[tn, n1, n2, n3, flag]$. $tn(j)$ is the triangle number. $n1(j)$, $n2(j)$ and $n3(j)$ are the index of the nodes which constitute the triangle. ($flag(j)$ is not used).

z_bounds:

This vector of size 2, $[zmin, zmax]$, gives the z values associated with the first and the last color (of the current colormap). More exactly if the colormap have nc colors and if we note $dz = (zmax - zmin)/nc$, then the part of the triangulation where $zmin + (i-1)dz \leq z < zmin + i \cdot dz$ is filled with the color i . By default the `z_bounds` property value is $[0, 0]$. In this case, the `zmin` and `zmax` are automatically set to the minimum and maximum of the `func` argument.

outside_color:

This vector of size 2, $[cmin, cmax]$, defines the color used when nodes values are outside the `z_bounds = [zmin, zmax]` interval. When node values are lower than `zmin` the color with index `cmin` is used. When node values are greater than `zmax` the color with index `cmax` is used. By default, the `outside_color` property value is $[0, 0]$. In this case, `cmin` and `cmax` are automatically set to the two bounds of the colormap. If `cmin` or `cmax` are negative, then values outside `z_bounds` interval are not displayed, they appear transparent.

color_range:

This vector of size 2, $[rmin, rmax]$, allows to use only a part of the colormap for display. `rmin` and `rmax` stand for colormap indices. If they are both greater than 1, then the actual colormap used to display the fec entity is `colormap(rmin:rmax)` where `colormap` is the colormap of the parent figure. By default, the `color_range` property value is $[0, 0]$. In this case, the whole colormap is used.

line_mode:

If "on", the wireframe enclosing triangles is drawn. If "off", only the inside of triangles are drawn.

foreground:

This color index specifies the color of the mesh. If `line_mode` property is "on", the wireframe is drawn using this color.

`clip_state`:

This field contains the `clip_state` property value for the fec. It should be :

- "off" this means that the fec is not clipped.
- "clipgrf" this means that the fec is clipped outside the Axes box.
- "on" this means that the fec is clipped outside the rectangle given by property `clip_box`.

`clip_box`:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the fec to display, however `clip_state` property value will be changed.

`user_data`:

This field can be use to store any scilab variable in the fec data structure, and to retrieve it.

Examples

```
x=-10:10; y=-10:10;m =rand(21,21);
Sgrayplot(x,y,m);
a=get("current_axes");
f=a.children.children(1)
f.data(:,3)=(1:size(f.data,1))';
a.parent.color_map=hotcolormap(64);
```

See Also

`set` , `get` , `delete` , `fec` , `Sgrayplot` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

fgrayplot — 2D plot of a surface defined by a function using colors

```
fgrayplot(x,y,f,[strf,rect,nax])  
fgrayplot(x,y,f,<opt_args>)
```

Parameters

x,y

real row vectors.

f

external of type $y=f(x,y)$.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `rect, nax, strf` or `axesflag` and `frameflag` (see `plot2d`).

strf,rect,nax

see `plot2d`.

Description

`fgrayplot` makes a 2D plot of the surface given by $z=f(x,y)$ on a grid defined by *x* and *y*. Each rectangle on the grid is filled with a gray or color level depending on the average value of *z* on the corners of the rectangle.

Enter the command `fgrayplot()` to see a demo.

Examples

```
t=-1:0.1:1;  
deff("[z]=my_surface(x,y)","z=x**2+y**2")  
fgrayplot(t,t,my_surface,rect=[-2,-2,2,2])
```

See Also

`grayplot` , `plot2d` , `Sgrayplot` , `Sfgrayplot`

Authors

J.Ph.C.

Name

figure_properties — description of the graphics figure entity properties

Description

The figure entity is the top level of the graphics entities hierarchy. This entity contain a number of properties designed to control many aspects of displaying Scilab graphics objects. These properties fall into two categories. Properties that contain information about figure itself and others related to set default values for the children creation.

Figure properties:

children:

This handles represent the vector of the figure's children . Note that all figure children are of type "Axes". Also keep in mind that, when creating a figure entity (using scf command), an Axes entity is simultaneously built too.

figure_style:

The value of this field defines the figure style. Scince Scilab 5.0, old graphic mode has been disable. This property is kept for compatibility but can not be modified.

figure_position:

This field contains the position in pixel of the graphic window on the screen. This is a vector $[x, y]$ defining the position of the upper-left corner of the window. The position $[0, 0]$ is the upper-left corner of the screen.

The initial position of graphic windows is taken from the default figure entity (see gdf). The only exception is when default figure `figure_position` value is $[-1, -1]$. In this case, the initial position of graphic windows is automatically set by the windowing system.

figure_size:

This property controls the size in pixel of the screen's graphics window. The size is the vector $[width, height]$.

axes_size:

Used to Specifies the size in pixel of the virtual graphics window. The size is the vector $[width, height]$. The virtual graphic window should be bigger than the part really visible on the screen. This property could not be modified if the figure is docked with other elements.

auto_resize:

This property determines if graphics window is resized. If the value is "on" then the `axes_size` property is equaled to the `figure_size` and the axes children are zoomed accordingly. If the value is "off" that indicate that `axes_size` cannot be resized when `figure_size` is changed.

viewport:

Postion of the visible part of graphics in the panner.

figure_name:

This field contains the name of the figure. This name is a character string displayed at the top of the graphics_window. The name can contain a single substring %d which will be replaced by the `figure_id`. No other instance of the % character is allowed inside the name.

figure_id:

This field contains the identifier of the figure. This is an integer number which is set at figure creation and cannot be changed after.

info_message:

This character string set the text displayed in the info bar of the graphic window.

color_map:

Property which defines the colormap used by this figure. The colormap is a m by 3 matrix. m is the number of colors. Color number i is given as a 3-uple R, G, B corresponding respectively to red, green and blue intensity between 0 and 1.

pixmap:

This property controls the pixmap status of a Graphic Window. If this property value is "off" the graphics are directly displayed on the screen. If it is "on" the graphics are done on a pixmap and are sent to the graphics window with the command `show_pixmap()`.

pixel_drawing_mode:

This field defines the bitwise operation used to draw the pixel on the screen. The default mode is `copy` what is to say the pixel is drawn as required. More generally the bitwise operation is performed between the color of the source pixel and the color of the pixel already present on the screen. Operations are : "clear", "and", "andReverse", "copy", "andInverted", "noop", "xor", "or", "nor", "equiv", "invert", "orReverse", "copyInverted", "orInverted", "nand", "set",

anti_aliasing:

This property controls the anti-aliasing level used to improve graphic quality. If the property is set to "off", anti-aliasing is disable. To enable anti-aliasing the property must set to either "2x", "4x", "8x" or "16x". In this case, it stands for the anti-aliasing level. For example, "16x" is a higher quality level than "2x". Specifying a higher anti-aliasing level improves image quality but also decreases graphic performances.

immediate_drawing:

This property controls the figure display. Its value can be "on" (default mode) or "off". It is used to delay a huge succession of graphics commands (implying several drawings or redrawings). Note that, when using `drawlater` or `drawnow` commands, it affects the property value of the current figure (which is respectively turned to 'off' or 'on').

background:

This property controls the figure window background color. It takes as value an index relative to the current colormap.

event_handler

A character string. The name of the Scilab function which is intended to handle the events. Not that setting an empty string will disable the event handler. For more information about event handler functions see the event handler functions help.

event_handler_enable

Enable or disable the event handler. Its value must be either "on" or "off".

user_data:

This field can be use to store any scilab variable in the figure data structure, and to retrieve it.

tag:

This field can be use to store a character string generally used to identify the control. It allows to give it a "name". Mainly used in conjontion with `findobj()`.

Children's default values:**visible:**

This field ules if the contents of the figure as to be redrawn. Its value should be "on" or "off".

rotation_style:

This field is related to the "3D Rot" button. It takes `unary` as value (default) in the aim to rotate only selected 3D plot. In the other case its value can be `multiple` : all 3D plots are rotated.

Note on default values :

All these listed properties and fields inherit from default values stored in a figure model. These default values can be seen and changed. To do so, use the `get("default_figure")` command : it returns a graphic handle on the figure model. Note that no graphic window is created by this command. The next created figures will inherit from this model (see example 2 below).

Examples

```
lines(0) // disables vertical paging

//Example 1
f=get("current_figure") //get the handle of the current figure :
                        //if none exists, create a figure and return the corres
f.figure_position
f.figure_size=[200,200]
f.background=2
f.children // man can see that an Axes entity already exists
delete(f);
f=gcf(); // macro shortcut <=> f=get("current_figure")
f.pixmap = "on" // set pixmap status to on
plot2d() // nothing happens on the screen...
show_pixmap() // ...display the pixmap on screen

//Example 2 : default_figure settings
df=get("default_figure") // get the default values (shortcut is gdf() )
// Let's change the defaults...
df.color_map=hotcolormap(128)
df.background= 110 // set background to a kind of yellow (Note that we
                  // are using a color index inside the color_map previously r
scf(122); // creates new figure number 122 with the new default
plot2d()
scf(214);
t=-%pi:0.3:%pi;
plot3d(t,t,sin(t))*cos(t),35,45,'X@Y@Z',[15,2,4]);
```

See Also

`lines`, `set`, `get`, `scf`, `gcf`, `gdf`, `gca`, `gda`, `axes_properties`, `show_pixmap`, `clear_pixmap`, `hotcolormap`, `event handler functions`

Authors

Djalel ABDEMOUCHE

Name

fplot2d — 2D plot of a curve defined by a function

```
fplot2d(xr,f,[style,strf,leg,rect,nax])  
fplot2d(xr,f,<opt_args>)
```

Parameters

xr

vector.

f

external of type $y=f(x)$ i.e. a scilab function or a dynamically linked routine referred to as a string.

style,strf,leg,rect,nax
see plot2d

<opt_args>
see plot2d

Description

fplot2d plots a curve defined by the external function f. The curve is approximated by a piecewise linear interpolation using the points $(xr(i), f(xr(i)))$. The values of $f(x)$ are obtained by feval(xr, f).

Enter the command fplot2d() to see a demo.

Examples

```
deff("[y]=f(x)","y=sin(x)+cos(x)")  
x=[0:0.1:10]*%pi/10;  
fplot2d(x,f)  
clf();  
fplot2d(1:10,'parab')
```

See Also

plot2d , feval , paramfplot2d

Authors

J.Ph.C.

Name

`fplot3d` — 3D plot of a surface defined by a function

```
fplot3d(xr,yr,f,[theta,alpha,leg,flag,ebox])  
fplot3d(xr,yr,f,<opt_args>)
```

Parameters

`xr`
row vector of size `n1`.

`yr`
row vector of size `n2`.

`f`
external of type `z=f(x,y)`.

`theta,alpha,leg,flag,ebox`
see `plot3d`.

`<opt_args>`
see `plot3d`.

Description

`fplot3d` plots a surface defined by the external function `f` on the grid defined by `xr` and `yr`.

Enter the command `fplot3d()` to see a demo.

Examples

```
deff('z=f(x,y)', 'z=x^4-y^4')  
x=-3:0.2:3 ;y=x ;  
clf() ;fplot3d(x,y,f,alpha=5,theta=31)
```

See Also

`plot3d`

Authors

J.Ph.C.

Name

fplot3d1 — 3D gray or color level plot of a surface defined by a function

```
fplot3d1(xr,yr,f,[theta,alpha,leg,flag,ebox])  
fplot3d1(xr,yr,f,<opt_args>)
```

Parameters

xr
row vector of size n1.

yr
row vector of size n2.

f
external of type $z=f(x,y)$.

theta,alpha,leg,flag,ebox
see plot3d1.

<opt_args>
see plot3d.

Description

fplot3d1 plots a 3D gray or color level plot of a surface defined by the external function **f** on the grid defined by **xr** and **yr**.

Enter the command `fplot3d1()` to see a demo.

Examples

```
def f('z=f(x,y)', 'z=x^4-y^4')  
x=-3:0.2:3 ;y=x ;  
clf() ;fplot3d1(x,y,f,alpha=5,theta=31)
```

See Also

plot3d1

Authors

J.Ph.C.

Name

gca — Return handle of current axes.

```
a = gca()
```

Parameters

a
handle, the handle of the current axes entity.

Description

This routine returns the handle of the current axes for the current figure.

Examples

```
subplot(211)
a=gca() //get the current axes
a.box="off";
t=-%pi:0.3:%pi;plot3d(t,t,sin(t)'*cos(t),80,50,'X@Y@Z',[5,2,4]);
subplot(212)
plot2d(); //simple plot
a=gca() //get the current axes
a.box="off";
a.x_location="middle";
a.parent.background=4;
delete(gca()) // delete the current axes
xdel(0) //delete a graphics window
```

See Also

gda , gcf , gce , get , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

gce — Get current entity handle.

```
e = gce()
```

Parameters

e
handle, the handle of the current entity.

Description

This routine returns the handle of the last created (and still existent) entity.

Examples

```
a=gca() //get the handle of the newly created axes
a.data_bounds=[1,1;10,10];
a.axes_visible = 'on' ;
for i=1:5
    xfrect(7-i,9-i,3,3);
    e=gce();
    e.background=i;
end
delete(); // delete current entity
delete(gce()); // delete current entity
delete(gcf()); // delete current figure
```

See Also

gcf , gca , get , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

gcf — Return handle of current graphic window.

```
h = gcf()
```

Parameters

h
handle.

Description

This routine returns the handle of the current graphic window.

Examples

```
f=gcf(); // Create a figure
f.figure_size= [610,469]/2;
f.figure_name= "Foo";

f=figure(); // Create a figure
h=uicontrol(f,"style","listbox","position", [10 10 150 160]); // Create a listbox
set(h, "string", "item 1|item 2|item3");// fill the list
set(h, "value", [1 3]); // select item 1 and 3 in the list
gcf()

scf(0); // Make graphic window 0 the current figure
gcf() // Return the graphic handle of the current figure

figure(f) // Make window f the current figure
gcf() // Return the graphic handle of the current figure
```

See Also

[gdf](#) , [gca](#) , [gce](#) , [get](#) , [scf](#) , [set](#) , [graphics_entities](#) , [uicontrol](#)

Authors

Serge Steer, INRIA

Name

`gda` — Return handle of default axes.

```
a = gda()
a = get("default_axes")
```

Parameters

`a`
handle, the handle of the default axes.

Description

The default axes is a graphic entity which is never drawn. It is used as a reference for the axes properties default values. These default properties values are used to initialize new axes inside figures.

The `gda` function returns the handle on the default axes. The user can use this handle to set or get the axes properties default values.

Note that an equivalent default graphic entity exists for figure entities too (see `gdf`).

Examples

```
a=gda() // get the handle of the model axes
// set its' properties
a.box="off";
a.foreground=2;
a.labels_font_size=3;
a.labels_font_color=5;
a.sub_tics=[5 5 3];
a.x_location="top";

//now used the model axes for drawing
subplot(211) //create an axes entity
plot2d()

// set other model's properties
a.background=color('gray')
a.grid=[5 5 5];
subplot(212)
t=0:0.1:5*pi;
plot2d(sin(t),cos(t))

set(a,"default_values",1); // return to the default values of the model
                           // see sda() function

clf()
plot2d(sin(t),cos(t))
```

See Also

`gdf` , `sda` , `sdf` , `clf` , `gca` , `get` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

`gdf` — Return handle of default figure.

```
f = gdf()
f = get("default_figure")
```

Parameters

`f`
handle, the handle of the default figure.

Description

The default figure is a graphic entity which is never drawn. It is used as a reference for the figure properties default values. These default properties values are used to initialize new figures.

The `gdf` function returns the handle on the default figure. The user can use this handle to set or get the figure properties default values.

Note that an equivalent default graphic entity exists for axes entities too (see `gda`).

Examples

```
f=gdf() // get the handle of the model figure
// setting its' properties
f.background=7;
f.figure_name="Function gdf()";
f.figure_position=[-1 100];
f.auto_resize="off";
f.figure_size=[300 461];
f.axes_size=[600 400];
plot2d() //create a figure
scf(1);
plot3d() //create a second figure
set(f,"default_values",1); // return to the default values of figure's model
                           // see sdf() function
scf(2);
plot2d()
```

See Also

`gda` , `sdf` , `sda` , `gcf` , `get` , `scf` , `set` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

ged — Scilab Graphic Editor

```
ged(action, fignum)
```

Parameters

action

Real: action to be executed on graphic window given by **fignum**:

- 1: Select window **fignum** as current figure.
- 2: Redraw window **fignum**.
- 3: Clear window **fignum**.
- 4: Ask the user to select a graphic entity to copy.
- 5: Paste last graphic entity copied using action 4.
- 6: Ask the user to select a graphic entity and then move it.
- 7: Ask the user to select the graphic entity to delete.
- 8: Start a GUI to edit window properties.
- 9: Start a GUI to edit current axes properties.
- 10: Start an entity picker to select a graphic object and edit it using Graphic Editor GUI.
- 11: Stop the entity picker.

fignum

Real: Graphic window number, the figure to edit.

Description

ged starts Scilab Graphic Editor on figure number **fignum** and execute action given by **action**.

Authors

V.C.

Name

genfac3d — compute facets of a 3D surface

```
[xx,yy,zz]=genfac3d(x,y,z,[mask])
```

Parameters

xx,yy,zz

matrices of size (4,n-1xm-1). $xx(:, i)$, $yy(:, i)$ and $zz(:, i)$ are respectively the x-axis, y-axis and z-axis coordinates of the 4 points of the i th four sided facet.

x

x-axis coordinates vector of size m.

y

y-axis coordinates vector of size n.

z

matrix of size (m,n). $z(i, j)$ is the value of the surface at the point $(x(i), y(j))$.

mask

boolean optional matrix with same size as z used to select the entries of z to be represented by facets.

Description

genfac3d computes a four sided facets representation of a 3D surface $z=f(x, y)$ defined by x , y and z .

Examples

```
t=[0:0.3:2*pi]'; z=sin(t)*cos(t');  
[xx,yy,zz]=genfac3d(t,t,z);  
plot3d(xx,yy,zz)
```

See Also

eval3dp , plot3d

Name

geom3d — projection from 3D on 2D after a 3D plot

```
[x,y]=geom3d(x1,y1,z1)
```

Parameters

$x1, y1, z1$

real vectors of the same size (points in 3D).

x, y

real vectors of the same size as $x1$, $y1$ and $z1$.

Description

After having used a 3D plot function such as `plot3d`, `plot3d1` or `param3d`, `geom3d` gives the mapping between a point in 3D space ($x1(i), y1(i), z1(i)$) and the corresponding point ($x(i), y(i)$) in the projected 2D plan. Then all the 2D graphics primitives working on (x, y) can be used for superposition on the 3D plot.

Examples

```
deff("[z]=surface(x,y)", "z=sin(x)*cos(y)")
t=%pi*(-10:10)/10;
// 3D plot of the surface
fplot3d(t,t,surface,35,45,"X@Y@Z")
// now (t,t,sin(t).*cos(t)) is a curve on the surface
// which can be drawn using geom3d and xpoly
[x,y]=geom3d(%pi/2,0,surface(%pi/2,0))
```

Authors

J.Ph.C.

Name

`get` — Retrieve a property value from a graphics entity or an User Interface object.

```
h=get(prop)
val=get(h,prop)
val=h.prop
```

Parameters

- `h`
handle, the handle of the entity to retrieve a property. `h` can be a vector of handles, in which case `get` returns the property value for all objects contained in `h`. `h` can also be `0` to get the root object properties.
- `prop`
character string name of the property.
- `val`
value of the property.

Description

This routine can be used to retrieve the value of a specified property from a graphics entity or a GUI object. In this case it is equivalent to use the dot operator on an handle. For example, `get(h, "background")` is equivalent to `h.background`.

Property names are character strings. To get the list of all existing properties see `graphics_entities` or `uicontrol` for User Interface objects.

`get` function can be also called with only a property as argument. In this case, the property must be one of the following:

- `current_entity` or `hdl`
returns a handle on the lastly created (and still existent) entity. `get('current_entity')` and `get('hdl')` are equivalent to `gce`.
- `current_figure`
returns a handle on the current graphic figure. `get('current_figure')` is equivalent to `gcf`.
- `current_axes`
returns a handle on the current axes entity. `get('current_axes')` is equivalent to `gca`.
- `default_figure`
returns a handle on the default figure entity. `get('default_figure')` is equivalent to `gdf`.
- `default_axes`
returns a handle on the default axes entity. `get('default_axes')` is equivalent to `gda`.
- `figures_id`
returns a row vector containing ids of all opened graphic figures. `get('figures_id')` is equivalent to `winsid`.

Examples

```
// for graphics entities
clf()
```



```
// simple graphics objects
subplot(121);
x=[-.2:0.1:2*pi]';
plot2d(x-2,x.^2);
subplot(122);
xrect(.2,.7,.5,.2);
xrect(.3,.8,.3,.2);
xfarc(.25,.55,.1,.15,0,64*360);
xfarc(.55,.55,.1,.15,0,64*360);
xstring(0.2,.9,"Example <<A CAR>>");

h=get("current_entity") //get the newly object created
h.font_size=3;

f=get("current_figure") //get the current figure
f.figure_size
f.figure_size=[700 500];
f.children
f.children(2).type
f.children(2).children
f.children(2).children.children.thickness=4;

a=get("current_axes") //get the current axes
a.children.type
a.children.foreground //get the foreground color of a set of graphics objects
a.children.foreground=9;

// for User Interface objects
h=uicontrol('string','Button'); // Opens a window with a button.
p=get(h,'position'); // get the geometric aspect of the button
disp('Button width: ' + string(p(3))); // print the width of the button
close(); // close figure
```

See Also

uicontrol , root_properties , graphics_entities , set

Authors

Djalel ABDEMOUCHE

Name

`get_figure_handle` — get a figure handle from its id

```
f = get_figure_handle(figure_id)
```

Parameters

`figure_id`

Integer, id of the figure to retrieve.

`f`

Handle of the corresponding figure.

Description

`get_figure_handle` function allows to retrieve the handle of a graphic figure knowing its id. If a figure with the specified id exists the function returns it. Otherwise it returns an empty matrix.

Examples

```
// create some figures
scf(0);
scf(5);
scf(12);

// get handle on the figure with id 5
f5 = get_figure_handle(5);
// current figure remains the one with id 12
gcf()
// get a non existing figure
f42 = get_figure_handle(42);
```

See Also

`set` , `get` , `gcf` , `scf` , `graphics_entities`

Authors

Jean-Baptiste Silvy INRIA

Name

`getcolor` — opens a dialog to show colors in the current colormap

```
c=getcolor(title,[cini])  
c=getcolor()
```

Parameters

`title`

string, dialog title.

`cini`

initial selected color id. Default value is 1.

`c`

selected color id or [] if the selection is cancelled.

Description

`getcolor` opens a window displaying the palette of the current colormap. The user can click on a color to show its id and RGB values. `getcolor` returns the id of the selected color or [] if the "Cancel" button has been clicked or the window closed.

See Also

`color` , `colormap` , `getmark` , `getfont`

Name

getfont — dialog to select font . **Obsolete function.**

```
[fId,fSize]=getfont()  
[fId,fSize]=getfont(str)  
fnt=getfont()  
fnt=getfont(str)  
fnt=getfont(S=str,font=[fId,fSize])
```

Parameters

str
character (e.g. "a")

fId
integer, the number of the selected font

fSize
integer, the size of the selected font

fnt
vector [fId,fSize]

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

getfont opens a graphic window to select a font. User has to select a font and a size clicking on the corresponding displayed character. Killing a keyboard key changes the displayed character.

Examples

```
[fId,fSize]=getfont();  
xset("font",fId,fSize)  
plot2d(0,0,rect=[0 0 10 10],axesflag=0)  
xstring(5,5,"string")
```

See Also

ged, text_properties

Name

getlinestyle — dialog to select linestyle. **Obsolete function.**

```
k=getlinestyle()
```

Parameters

k
integer, selected linestyle or [] if the "Cancel" button has been clicked.

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

getlinestyle opens a graphic window to select a line style.

Examples

```
x=0:0.1:10;  
plot2d(x,sin(x))  
e=gce(); // store the Compound containing the plot  
e.children(1).line_style = getlinestyle();
```

See Also

ged, set, segs_properties, segs_properties

Name

getmark — dialog to select mark (symbol). **Obsolete function**

```
[mark, mrkSize]=getmark()
```

Parameters

mark
integer, the number of the selected mark

mrkSize
integer, the size of the selected mark

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

getmark opens a graphic window to select a mark (symbol).

Examples

```
x=0:0.1:10;  
[mark, mrkSize]=getmark();  
plot2d(x, sin(x), style=-mark);  
clf();  
plot2d(x, sin(x))  
e=gce(); // store the Compound containing the plot  
[mark, mrkSize]=getmark();  
e.children(1).mark_style = mark;
```

See Also

ged, set, segs_properties, segs_properties

Name

getsymbol — dialog to select a symbol and its size. **Obsolete function**

```
c=getsymbol([title])
```

Parameters

title

string, dialog title.

c

vector of size 2 [n, sz].

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

getsymbol opens a dialog choice box with title title if given where the user can select a symbol and its size. getsymbol returns the id of the mark n and the id of its size sz.

See Also

ged, set, segs_properties, segs_properties

Name

glue — glue a set of graphics entities into an Compound.

```
glue(H)
h_agreg=glue(H)
```

Parameters

H
a vector of handle.

h_agreg
a handle, the handle on the Compound entity.

Description

Given a vector of handles, this function glue the corresponding entities in a single Compound and returns the handle on this new entity.

Examples

```
scf(1);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2]);
xrect(-1,1,2,2);
xarc(-0.5,0.5,2,2,0,360*64);
a=gca();
r=a.children(1);
c=a.children(2);
nc=glue([r,c]);
// a.children(1) and nc now both correspond to the
// newly created compound object,
a.children(1)
// which is then translated.
move(nc,[-1,-0.5]);
```

See Also

get , set , move , unglue , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

graduate — pretty axis graduations

```
[xi,xa,np]=graduate( xmi, xma,n1,n2)
[xi,xa,np]=graduate( xmi, xma)
```

Parameters

xmi,xma
real scalars

n1, n2
integers with default values 3,10

xi, xa
real scalars

np
integer

Description

graduate looks for the minimum interval $[xi, xa]$ and a number of tics np such that:

$$xi \leq xmi \leq xma \leq xa$$
$$xa - xi / np = k(10^n), k \text{ in } [1 \ 3 \ 5] \text{ for an integer } n$$
$$n1 < np < n2$$

Examples

```
y=(0:0.33:145.78)';
clf();plot2d1('enn',0,y)
[ymn,ymx,np]=graduate(mini(y),maxi(y))
rect=[1,ymn,prod(size(y)),ymx];
clf();plot2d1('enn',0,y,1,'011',' ',rect,[10,3,10,np])
```

See Also

xsetech , plot2d

Authors

S. Steer 1992;

Name

graphics_entities — description of the graphics entities data structures

Description

In Scilab, graphics window and the drawing it contains are represented by hierarchical entities. The hierarchy top level is the Figure. Each Figure defines at least one child of type Axes. Each Axes entity contains a set of leaf entities which are the basic graphics objects like Polylines, Rectangles, Arcs, Segs,... It can also contain an Compound type which are recursive sets of entities. The main interest of the graphic mode is to make property change easier. This graphics' mode provides a set of high-level graphing routines (see `set`, `get`) used to control objects' properties such as data, coordinates and scaling, color and appearances without requiring to replay the initial graphics commands.

Graphics entities are associated to Scilab variables of type `handle`. The handle is a unique identifier which is associated to each instance of a created graphical entity. Using this handle, it will be possible to reach entities' properties through "set" and "get" routines. The handles are also used to manipulate graphics objects, to move them, to make copies or delete them.

Figure:

The figure entity is the top level of the graphics entities hierarchy. This entity defines the parameters for the figure itself as well as the parameters' default values for the children creation. The figure children are the Axes entities.

The handle on the current figure (the figure used where the drawing are sent) may be got using `get("current_figure")` and it may be set using `set("current_figure", h)`, where `h` is either a handle on a figure or a `figure_id` in this last case if the figure does not already exists, it is created

See `figure_properties` for details.

Axes:

The Axes entity is the second level of the graphics entities hierarchy. This entity defines the parameters for the change of coordinates and the axes drawing as well as the parameters' default values for its children creation. See `axes_properties` for details. The handle on the current Axes may be got using `get("current_axes")`.

Compound:

The Compound entity is just a vector of children and with a single property (visibility property). It is used to glue a set of entities together.

See `glue`, `unglue` and `Compound_properties` functions.

Axis:

The Axis entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for axis scaling and appearance.

See `axis_properties` for details.

Polyline:

The polyline entity is a leaf of the graphics entities hierarchy. It defines 2D and 3D polylines and polylines extensions drawing properties.

See `polyline_properties` for details.

Arc:

The Arc entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for ellipses and part of ellipses.

See `arc_properties` for details.

Rectangle:

The Rectangle entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for rectangles and filled rectangles.

See rectangle_properties for details.

Surface:

The Surface entity is a leaf of the graphics entities hierarchy. It has sub types Fac3d or Plot3d. This entity defines the parameters for 3d surface plots.

See surface_properties for details.

Fec:

The Fec entity is a leaf of the graphics entities hierarchy. It represents 2D finite elements plots .

See fec_properties for details.

Grayplot:

The Grayplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images.

See grayplot_properties for details.

Matplot:

The Matplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots using integer matrices.

See Matplot_properties for details.

Segs:

The Segs entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a set of colored segments or colored arrows.

See segs_properties for details.

Champ:

The Champ entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a 2D vector field.

See champ_properties for details.

Text:

The Text entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for string drawing.

See text_properties for details.

Label:

The Labels entity are children of the Axes graphics entity. This entity defines the parameters for the 3 x,y and z labels and title drawn on a graphics window.

See label_properties for details.

Legend:

The Legend entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for legends drawn below plot2dx graphs. This entity requires further developments.

See legend_properties for details.

Examples



```
//Play this example line per line

scf() //create a figure in entity mode

//get the handle on the Figure entity and display its properties
f=get("current_figure")
a=f.children // the handle on the Axes child
x=(1:10)'; plot2d(x,[x.^2 x.^1.5])
e=a.children //Compound of 2 polylines

p1=e.children(1) //the last drawn polyline properties
p1.foreground=5; // change the polyline color
e.children.thickness=5; // change the thickness of the two polylines

delete(e.children(2))

move(e.children,[0,30]) //translate the polyline

a.axes_bounds=[0 0 0.5 0.5];

subplot(222) //create a new Axes entity
plot(1:10);
a1=f.children(1); //get its handle
copy(e.children,a1); //copy the polyline of the first plot in the new Axes
a1.data_bounds=[1 0;10 100]; //change the Axes bounds

set("current_figure",10) //create a new figure with figure_id=10
plot3d() //the drawing are sent to figure 10
set("current_figure",f) //make the previous figure the current one
plot2d(x,x^3) //the drawing are sent to the initial figure
```

See Also

set , get , move , draw , delete , object_editor , plot , surf

Name

graphics_fonts — description of fonts used in graphic figures

Description

Some Graphic entities such as Text, Axes, Label or Legend entities display one or more character strings in graphic figures. The appearance of the displayed strings can be modified by specifying different fonts and character sizes.

Changing font

Fonts used in graphic figures are selected from a set of fonts called loaded fonts. In other words, the loaded fonts are the ones currently available in graphic figures. The list of these fonts can be obtained using the `xlfont` function without argument. By default, Scilab contains a set of 11 loaded fonts. This set can be modified and extended using the `xlfont` function with a font name as argument. The added font can either be loaded from a file or be one of the system. To know the list of fonts available on the system use the `xlfont('AVAILABLE_FONTS')` command. For more information on how to manipulate fonts see `xlfont`.

Here is the list of the 11 default fonts.

Font number	Font Family	Bold	Italic
0	Monospaced	No	No
1	ScilabSymbols	No	No
2	Serif	No	No
3	Serif	No	Yes
4	Serif	Yes	No
5	Serif	Yes	Yes
6	SansSerif	No	No
7	SansSerif	No	Yes
8	SansSerif	Yes	No
9	SansSerif	Yes	Yes
10	SansSerif	Yes	Yes

The font used by a graphic entities can be modified with its `font_style` property. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font.

The `fractional_font` controls the font anti-aliasing. Its value can be either 'on' or 'off'. If its value is 'on' the font is smoothed, otherwise it's not.

Changing character size

The text size of a graphic entity is modified using the `font_size` property. It is a scalar specifying the displayed character size.

The Scilab character size is different from the Java size. Here is the equivalence between the two scales.

Scilab Size	Java Size
0	8
1	10
2	12

3	14
4	18
5	24
6	30
7	36
8	42
9	48
10	54

The character size might not be an integer. In this case, the result depends on the entities `fractional_font` property. If `fractional_font` property is 'on' then the displayed font size is interpolated between the two closest integer. For example, a font size of 2.5 displays characters with a Java size of 13. If `fractional_font` property is 'off' then the displayed font size is truncated to its integer part. For example, a font size of 2.5 displays characters using a Java size of 12.

See Also

`xlfont` , `graphics_entities`

Name

graycolormap — linear gray colormap

```
cmap=graycolormap(n)
```

Parameters

n
integer ≥ 1 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

graycolormap computes a colormap with *n* gray colors varying linearly from black to white.

Examples

```
f = scf();  
plot3d1();  
f.color_map = graycolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap , xset

Name

grayplot — 2D plot of a surface using colors

```
grayplot(x,y,z,[strf,rect,nax])  
grayplot(x,y,z,<opt_args>)
```

Parameters

x,y

real row vectors of size n1 and n2.

z

real matrix of size (n1,n2). $z(i,j)$ is the value of the surface at the point $(x(i),y(j))$.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, ...` can be one of the following: `rect`, `nax`, `strf` or `axesflag` and `frameflag` (see `plot2d` and `plot2d_old_version`).

strf,rect,nax

see `plot2d`.

Description

`grayplot` makes a 2D plot of the surface given by `z` on a grid defined by `x` and `y`. Each rectangle on the grid is filled with a gray or color level depending on the average value of `z` on the corners of the rectangle. If `z` contains %nan values, the surrounding rectangles are not displayed.

Enter the command `grayplot()` to see a demo.

Examples

```
x=-10:10; y=-10:10;m =rand(21,21);  
grayplot(x,y,m,rect=[-20,-20,20,20])  
t=-%pi:0.1:%pi; m=sin(t)'+cos(t);  
clf()  
grayplot(t,t,m)
```

See Also

`fgrayplot` , `plot2d` , `Sgrayplot` , `Sfgrayplot`

Authors

J.Ph.C.

Name

grayplot_properties — description of the grayplot entities properties

Description

The Grayplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images (see `grayplot`, `Sgrayplot`, `fgrayplot` and `Sfgrayplot`).

parent:

This property contains the handle of the parent. The parent of the grayplot entity should be of the type "Axes".

children:

This property contains a vector with the `children` of the handle. However, `grayplot` handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity . It should be "on" or "off" . By default, the plot is visible, the value's property is "on". If "off" the plot is not drawn on the screen.

data:

This field defines a `tlst` data structure of type "grayplotdata" composed of a row and column indices of each element : the x and y grid coordinates are contained respectively in `data.x` and `data.y`. The complementary field named `data.z` is the value of the surface at the point $(x(i), y(j))$ (scaled mode) or the centered value of the surface defined between two consecutive $x(i)$ and $y(j)$ (direct mode). If `data_mapping` (see below) is set to "scaled", the entire z data is used to perform a color interpolation whereas, if `data_mapping`'s value is "direct", the last line and column of the z data indices are ignored and the color is determined directly in the colormap by the indices of the submatrix `data.z(1:$-1, 1:$-1)`.

data_mapping:

By default the value of this property is "scaled" : the indices of painting colors are proportional to the value z coordinates. In the other case, the property takes as value "direct" where the plot is a grayplot and the indices of painting colors are given by the data (see above).

clip_state:

This field contains the `clip_state` property value for the grayplot. It should be :

- "off" this means that the grayplot is not clipped.
- "clipgrf" this means that the grayplot is clipped outside the Axes box.
- "on" this means that the grayplot is clipped outside the rectangle given by property `clip_box`.

clip_box:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector $[x, y, w, h]$ (upper-left point width height) defines the portions of the grayplot to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the grayplot data structure, and to retrieve it.

Examples

```
m=5;n=5;
```

```
M=round(32*rand(m,n));
grayplot(1:m,1:n,M)

a=get("current_axes");
a.data_bounds= [-1,-1;7,7]
h=a.children

h.data_mapping="direct";

// A 2D plotting of a matrix using colors
clf()
a=get("current_axes");
a.data_bounds= [0,0;4,4];

b=5*ones(11,11); b(2:10,2:10)=4; b(5:7,5:7)=2;
Matplot1(b,[1,1,3,3]) ;

h=a.children
for i=1:7
    xclick(); // click the mouse to sets Matplot data
    h.data=h.data+4;
end
```

See Also

[set](#) , [get](#) , [delete](#) , [grayplot](#) , [Matplot](#) , [Matplot1](#) , [graphics_entities](#) , [Matplot_properties](#)

Authors

Djalel ABDEMOUCHE & F.Leray

Name

graypolarplot — Polar 2D plot of a surface using colors

```
graypolarplot(theta, rho, z, [strf, rect])
```

Parameters

theta

a vector with size n1, the discretization of the the angle in radian.

rho

a vector with size n2, the discretization of the radius

z

real matrix of size (n1,n2). $z(i, j)$ is the value of the surface at the point (theta(i),rho(j)).

strf

is a string of length 3 "xy0".

default

The default is "030".

x

controls the display of captions.

x=0

no captions.

x=1

captions are displayed. They are given by the optional argument `leg`.

y

controls the computation of the frame.

y=0

the current boundaries (set by a previous call to another high level plotting function) are used. Useful when superposing multiple plots.

y=1

the optional argument `rect` is used to specify the boundaries of the plot.

y=2

the boundaries of the plot are computed using min and max values of x and y.

y=3

like y=1 but produces isoview scaling.

y=4

like y=2 but produces isoview scaling.

y=5

like y=1 but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=6

like y=2 but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=7

like y=5 but the scale of the new plot is merged with the current scale.

y=8

like y=6 but the scale of the new plot is merged with the current scale.

leg

a string. It is used when the first character x of argument `strf` is 1. `leg` has the form "`leg1@leg2@. . .`" where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is "".

rect

This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

Description

Takes a 2D plot of the surface given by `z` on a polar coordinate grid defined by `rho` and `theta`. Each grid region is filled with a gray or color level depending on the average value of `z` on the corners of the grid.

Examples

```
rho=1:0.1:4;theta=(0:0.02:1)*2*pi;
z=30+round(theta'*(1+rho^2));
f=gcf();
f.color_map= hotcolormap(128);
clf();graypolarplot(theta,rho,z)
```

Name

havewindow — return scilab window mode

```
havewindow()
```

Description

returns %t if scilab has it own window and %f if not, i.e. if scilab has been invoked by "scilab -nw".
(nw stands for "no-window").

Name

hist3d — 3D representation of a histogram

```
hist3d(f,[theta,alpha,leg,flag,ebox])
hist3d(f,<opt_args>)
hist3d(list(f,x,y),[theta,alpha,leg,flag,ebox])
hist3d(list(f,x,y),<opt_args>)
```

Parameters

mtx

matrix of size (m,n) defining the histogram $mtx(i,j)=F(x(i),y(j))$, where x and y are taken as $0:m$ and $0:n$.

list(mtx,x,y)

where mtx is a matrix of size (m,n) defining the histogram $mtx(i,j)=F(x(i),y(j))$, with x and y vectors of size (1,m+1) and (1,n+1).

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `theta, alpha, leg, flag, ebox`. See `plot3d`.

`theta,alpha,leg,flag,ebox`
see `plot3d`.

Description

`hist3d` represents a 2d histogram as a 3D plot. The values are associated to the intervals $[x(i), x(i+1)[$ $\times$ $[y(i), y(i+1)[$.

Enter the command `hist3d()` to see a demo.

Examples

```
hist3d(10*rand(10,10));

Z = zeros(100,5);
A = abs(rand(40,5));
Z(1:40,:) = A;
scf();
hist3d(Z);

Index = find(Z==0);
Z(Index) = %nan;
scf();
hist3d(Z);

A = abs(rand(10,5));
Z(91:100,:) = A;
scf();
hist3d(Z);
```

See Also

`histplot` , `plot3d`

Authors

Steer S. & JPhilippe C.

Name

histplot — plot a histogram

```
histplot(n, data, <opt_args>)  
histplot(x, data, <opt_args>)
```

Parameters

n
positive integer (number of classes)

x
increasing vector defining the classes (x may have at least 2 components)

data
vector (datas to be analysed)

<opt_args>
This represents a sequence of statements `key1=value1, key2=value2 ,...` where `key1, key2, ...` can be any optional plot2d parameter (`style, strf, leg, rect, nax, logflag, frameflag, axesflag`) or normalization. For this last one the corresponding value must be a boolean scalar (default value %t).

Description

This function plot an histogram of the **data** vector using the classes **x**. When the number **n** of classes is provided instead of **x**, the classes are choosen equally spaced and $x(1) = \min(data) < x(2) = x(1) + dx < \dots < x(n+1) = \max(data)$ with $dx = (x(n+1)-x(1))/n$.

The classes are defined by $C1 = [x(1), x(2)]$ and $Ci = (x(i), x(i+1)]$ for $i \geq 2$. Noting N_{max} the total number of **data** ($N_{max} = \text{length}(\text{data})$) and Ni the number of **data** components falling in Ci , the value of the histogram for **x** in Ci is equal to $Ni/(N_{max} (x(i+1)-x(i)))$ when `normalization` is true (default case) and else, simply equal to Ni . When normalization occurs the histogram verifies:

```
x(n+1)  
/  
|  h(x) dx = 1,  when x(1)<=min(data) and max(data) <= x(n+1))  
/  
x(1)
```

Any plot2d (optional) parameter may be provided; for instance to plot an histogram with the color number 2 (blue if `std colormap` is used) and to restrict the plot inside the rectangle `[-3,3]x[0,0.5]`, you may use `histplot(n,data, style=2, rect=[-3,0,3,0.5])`.

Enter the command `histplot()` to see a demo.

Examples

```
// example #1: variations around an histogram of a gaussian random sample  
d=rand(1,10000,'normal'); // the gaussian random sample  
clf();histplot(20,d)  
clf();histplot(20,d,normalization=%f)  
clf();histplot(20,d,leg='rand(1,10000,\'normal\')',style=5)
```



```
clf();histplot(20,d,leg='rand(1,10000,\'normal\')',style=16, rect=[-3,0,3,0.5])

// example #2: histogram of a binomial (B(6,0.5)) random sample
d = grand(1000,1,"bin", 6, 0.5);
c = linspace(-0.5,6.5,8);
clf()
subplot(2,1,1)
    histplot(c, d, style=2)
    xtitle("normalized histogram")
subplot(2,1,2)
    histplot(c, d, normalization=%f, style=5)
    xtitle("non normalized histogram")

// example #3: histogram of an exponential random sample
lambda = 2;
X = grand(100000,1,"exp", 1/lambda);
Xmax = max(X);
clf()
histplot(40, X, style=2)
x = linspace(0,max(Xmax),100)';
plot2d(x,lambda*exp(-lambda*x),strf="000",style=5)
legend(["exponential random sample histogram" "exact density curve"]);
```

See Also

hist3d , plot2d , dsearch

Name

hotcolormap — red to yellow colormap

```
cmap=hotcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

hotcolormap computes a colormap with n hot colors varying from red to yellow.

Examples

```
f = scf();  
plot3d1();  
f.color_map = hotcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

hsv2rgb — Converts HSV colors to RGB

```
[r,g,b] = hsv2rgb(h,s,v)
rgb = hsv2rgb(h,s,v)
[r,g,b] = hsv2rgb(hsv)
rgb = hsv2rgb(hsv)
```

Parameters

h
a vector of size n. The "hue" values.

s
a vector of size n. The "saturation" values.

v
a vector of size n. The "value" values

hsv
a n x 3 matrix. Each row contains a [hue saturation value] tripple.

r
a column vector of size n. The associated "red" values.

g
a column vector of size n. The associated "green" values.

b
a column vector of size n. The associated "blue" values.

rgb
a n x 3 matrix. Each row contains a [red green blue] tripple.

Description

The function `hsv2rgb` converts colormaps between the RGB and HSV color spaces. As hue varies from 0 to 1.0, the corresponding colors vary from red through yellow, green, cyan, blue, magenta, and back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter.

Examples

```
t=[0:0.3:2*pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
f=gcf();f.pixmap='on';
for h=0:0.1:1
    hsv=[h*ones(32,1) linspace(0,1,32)' 0.7*ones(32,1)];
    f.color_map=hsv2rgb(hsv);
    show_pixmap()
    xpause(10000)
end
for v=0:0.1:1
    hsv=[ones(32,1) linspace(0,1,32)' v*ones(32,1)];
```

```
f.color_map=hsv2rgb(hsv);  
show_pixmap()  
xpause(10000)  
end
```

Authors

Serge Steer
INRIA

Name

hsvcolormap — Hue-saturation-value colormap

```
cmap = hsvcolormap(n)
```

Parameters

n
integer ≥ 1 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

hsvcolormap computes a colormap with `ncolors`. This colormap varies the hue component of the hsv color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The map is particularly useful for displaying periodic functions.

Examples

```
t=[0:0.1:2*pi]'; z=sin(t)*cos(t');  
f=gcf();f.color_map=hsvcolormap(64);  
plot3d1(t,t,z,35,45,"X@Y@Z",[-2,2,2])
```

Authors

Serge Steer
INRIA

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

`is_handle_valid` — Check whether a set of graphic handles is still valid.

```
isValid = is_handle_valid(h)
```

Parameters

`h`
Matrix of graphic handles

`isValid`
Matrix of boolean with the same size as `h`

Description

`is_handle_valid` function tests whether a set of graphic handle is still valid. A valid handle is a handle which has not been deleted. The result, `isValid`, is a boolean matrix such as `isValid(i, j)` is true if `h(i, j)` is valid and false otherwise.

Examples

```
// check that current objects are valid
is_handle_valid([gcf(), gca(), gce()])

// create 11 polylines
plot([0:10; 0:10; 0:10], [0:10; 0:0.5:5; 0:2:20]);

// check polylines validity
axes = gca();
polylines = axes.children(1).children
is_handle_valid(polylines)

// delete some polylines
delete(polylines(3:7));
// print validity
is_handle_valid(polylines)
```

See Also

`delete`, `graphics_entities`

Authors

Jean-Baptiste Silvy INRIA

Name

`isoview` — set scales for isometric plot (do not change the size of the window)

```
isoview(xmin, xmax, ymin, ymax)
```

Parameters

`xmin, xmax, ymin, ymax`
four real values

Description

This function is obsolete, use preferably the `frameflag=4` `plot2d` option which enable window resizing.

`isoview` is used to have isometric scales on the x and y axes. It does not change the size of the graphics window. The rectangle `xmin, xmax, ymin, ymax` will be contained in the computed frame of the graphics window. `isoview` set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance `strf="x0z"` in `plot2d`).

Examples

```
t=[0:0.1:2*pi]';
plot2d(sin(t),cos(t))
clf()
isoview(-1,1,-1,1)
plot2d(sin(t),cos(t),1,"001")
xset("default")

plot2d(sin(t),cos(t),frameflag=4)
```

See Also

`square` , `xsetech`

Authors

Steer S.

Name

jetcolormap — blue to red colormap

```
cmap=jetcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

jetcolormap computes a colormap with n colors varying from blue, lightblue, green, yellow, orange then red

Examples

```
f = scf();  
plot3d1();  
f.color_map = jetcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

label_properties — description of the Label entity properties

Description

The Label entity is a child of an `AXES` entity. When an Axes entity is built, the Title and Labels handles come with it and are part of the Axes properties. Therefore, the properties of those sub-objects are editable via the Axes handle (see `gca` and `gda`). This entity defines the parameters for label drawing:

parent:

This property contains the handle of the parent. The parent of the label entity should be of type `"Axes"`.

Note that, for now, Label entity is exclusively used in `title`, `x_label`, `y_label` and `z_label` building.

visible:

This field contains the `visible` property value for the entity. It should be `"on"` or `"off"`. By default, the label is visible, the value's property is `"on"`. If `"off"` the label is not displayed on the screen.

text:

The matrix containing the strings of the object. The rows of the matrix are displayed horizontally and the columns vertically.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

font_foreground:

This field contains the color used to display the label `text`. Its value should be a color index (relative to the current colormap).

foreground:

This field contains the color used to display the line around the box if any. Its value should be a color index (relative to the current colormap).

background:

This field contains the color used to fill the box if any. Its value should be a color index (relative to the current colormap).

fill_mode:

This field takes the values `"on"` or `"off"`. If `"on"` a box is draw around the text with a line on its edge and a background.

font_style:

Specifies the font used to display the label. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see `graphics_fonts`.

font_size:

It is a scalar specifying the displayed characters size. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see `graphics_fonts`.

fractional_font:

This property specify whether text is displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

font_angle:

This scalar allows you to turn the label. The font is turned clockise with the angle given in degrees. Note that changing the `font_angle` will automatically disable the `auto_rotation` property.

auto_rotation:

If "on", Scilab computes automatically the best angle to turn the label for the display. If "off", the label can be manually turned with the `font_angle` property.

position:

This 2d vector allows you to place manually the label on the screen. The position is stored in the data units of the axes. Note that changing the `font_angle` will automatically disable the `auto_position` property.

auto_position:

If "on", Scilab computes automatically the best position in the graphic window for the display. If "off", the label can be manually places with the `position` property.

Examples

```
a=get("current_axes");
a.title
type(a.title)
plot3d()
a.x_label
a.y_label
a.z_label
xtitle("My title","my x axis label", "Volume","Month")

t=a.title;
t.foreground=9;
t.font_size=4;
t.font_style=5;
t.text="SCILAB";

x_label=a.x_label;
x_label.text=" Weight"
x_label.font_style= 5;
a.y_label.foreground = 12;

// Starting with Scilab 5.2, it is now possible to write LaTeX or MathML:
t.text="\sqrt{SCILAB}";
x_label.font_size= 5;
x_label.text="<mrow><mfrac><mrow><mn>1</mn></mrow><mrow><mn>2</mn></mrow></mfrac>
<mrow><mfrac><mrow><mfrac><mrow><mi>a</mi></mrow><mrow><mi>b</mi></mrow></mfrac>
</mfrac></mrow></mfrac></mrow></mfenced></mrow>"
```

See Also

`set` , `get` , `delete` , `xtitle` , `graphics_entities` , `axes_properties` , `text_properties`

Authors

Djalel ABDEMOUCHE

Name

legend — draw graph legend

```
hl=legend([h,] string1,string2, ... [,pos] [,boxed])  
hl=legend([h,] strings [,pos] [,boxed])
```

Parameters

h

graphic handle on an Axes entity or vector of handles on polyline entities. The default value is the handle on `current_axes`.

string1,string2, ...

character strings strings_i is the legend of the *i*th curve

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

strings

n vector of strings, strings(*i*) is the legend of the *i*th curve

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

pos

(optional) specify where to draw the legend; this parameter may be a integer flag (or equivalently a string flag) or a vector [x,y] which gives the coordinates of the upper left corner of the legend box. In the first case the possible values are:

1

the legend is drawn in the upper right-hand corner (default).

2

the legend is drawn in the upper left-hand corner.

3

the legend is drawn in the lower left-hand corner.

4

the legend is drawn in the lower right-hand corner.

5

interactive placement with the mouse .

-1

the legend is drawn at the right of the upper right-hand corner.

-2

the legend is drawn at the left of the upper left-hand corner.

-3

the legend is drawn at the left of the lower left-hand corner.

-4

the legend is drawn at the right of the lower right-hand corner.

-5

the legend is drawn above the upper left-hand corner.

-6

the legend is drawn below the lower left-hand corner.

boxed

a boolean (default value %t) which sets or not the drawing of the box.

hl

a handle, points to the Compound containing all the legend .

Description

Puts a legend on the current plot using the specified strings as labels. legend prepends labels by a recall of the corresponding line or patch. The recall type and properties are recovered from the given handles:

when called without handle argument (or with a handle on a axes entity) the function first build the vectors of handle on polylines entities which are the children of the given axes.

In the interactive placement mode (opt=5) move the legend box with the mouse and press the left button to release it.

Examples

```
t=linspace(0,%pi,20);
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-5,2 3]);
e=gce();
e1=e.children(1);e1.thickness=2;e1.polyline_style=4;e1.arrow_size_factor = 1/2;
e.children(2).line_style=4;
e3=e.children(3);e3.line_mode='on';e3.mark_background=5;

hl=legend(['cos(t)';'cos(2*t)';'cos(3*t)']);
```

See Also

plot2d, xstring, captions, polyline_properties

Name

legend_properties — description of the Legend entity properties.

Description

The Legend entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for legends drawn below `plot2dx` graphs or created by the `captions` function. For selected line plotted, the legend shows a sample of the line type, marker symbol, and color.

parent:

This property contains the handle of the parent. The parent of the legend entity should be of the type "Compound". This Compound entity contains also the remainder of the graph's entities.

children:

This property contains a vector with the `children` of the handle. However, `legend` handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity . It should be "on" or "off" . If "on" the legend is drawn , If "off" the legend is not displayed on the screen.

text:

This field is the character string vector which contains the legends for each annotated objects.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

font_size:

It is a scalar specifying the displayed characters size. If `fractional_font` property is "off" only the integer part of the value is used. For more information see `graphics_fonts`.

font_style:

Specifies the font used to display the legend labels. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see `graphics_fonts`.

font_color

A color index, this property determines the color of the text.

fractional_font:

This property specify whether text is displayed using fractional font sizes. Its value must be either "on" or "off". If "on" the floating point value of `font_size` is used for display and the font is anti-aliased. If "off" only the integer part is used and the font is not smoothed.

links:

A row array of handles. They refer to the associated polylines.

legend_location

A character string, specifies the location of the Legend.

- "in_upper_right" : captions are drawn in the upper right corner of the axes box.
- "in_upper_left": captions are drawn in the upper left corner of the axes box.
- "in_lower_right": captions are drawn in the lower right corner of the axes box.
- "in_lower_left": captions are drawn in the lower left corner of the axes box.
- "out_upper_right": captions are drawn at the right of the upper right corner of the axes box.
- "out_upper_left": captions are drawn at the left of the upper left corner of the axes box.

- "out_lower_right": captions are drawn at the right of the lower right corner of the axes box.
- "out_lower_left": captions are drawn at the left of the lower left corner of the axes box.
- "upper_caption": captions are drawn above the upper left corner of the axes box.
- "lower_caption": captions are drawn below the lower left corner of the axes box. This option correspond to the `leg` argument of `plot2d`
- "by_coordinates": the upper left corner of the captions box is given by the "position" field of the associated data structure. The x and y positions are given as fractions of the `axes_bounds`

position

The coordinates of the upper left corner of the legend. The x and y positions are given as fractions of the `axes_bounds` sizes. This field may be set if `legend_location=="by_coordinates"` or get for the other `legend_location` settings.

line_mode

This field specifies if a rectangle is drawn around the legend or not. It should be "on" or "off". If "on" the rectangle is drawn using the following properties.

thickness

This property is a positive real specifying the surrounding rectangle shape width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

foreground

This field gives the color index of the line used to draw the rectangle shape.

fill_mode

This field specifies if the legend background is painted or not. It should be "on" or "off". If "on" the background is painted using the color index set in the `background` field.

background

This field gives the color index of the line used to paint the rectangle area.

clip_state:

This field contains the default `clip_state` property value for all objects. Its value should be :

- "off" this means that all objects created after that are not clipped (default value).
- "clipgrf" this means that all objects created after that are clipped outside the Axes boundaries.
- "on" this means that all objects created after that are clipped outside the rectangle given by property `clip_box`.

clip_box:

This field contains the default `clip_box` property value for all objects. Its value should be an empty matrix if `clip_state` is "off". Other case the clipping is given by the vector `[x, y, w, h]` (upper-left point width height).

user_data:

This field can be use to store any scilab variable in the text data structure, and to retrieve it.

Examples

```
// x initialisation
x=[0:0.1:2*pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)],...
    [1,2,3],leg="L1@L2@L3")
a=get("current_axes");
l=a.children(2);
l.links
l.text=["sin(x)";"sin(2*x)";"sin(3*x)"];
l.visible="off"; // invisible
l.font_size = 2;
l.font_style = 5;
l.visible='on';
// Starting from Scilab 5.2, Latex:
l.text=["$\sin(x)$";"$\sin(2*x)$";"$\sin(3*x)$"];
```

See Also

plot2d, graphics_entities

Authors

Djalel ABDEMOUCHE

Name

legends — draw graph legend

```
legends(strings, style, <opt_args>)
```

Parameters

strings

n vector of strings, strings(i) is the legend of the ith curve

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

style

integer row vector of size n (the plot styles, third parameter of plot2d) or an integer 2 x n matrix, style(1,k) contains the plot style for the kth curve and style(2,k) contains the line style (if style(1,k)>0) or mark color (if style(1,k)<0).

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following:

opt

specify where to draw the legends; this parameter may be a integer flag (or equivalently a string flag) or a vector [x,y] which gives the coordinates of the upper left corner of the legend box. In the first case the possible values are:

1 or "ur"

the legends are drawn in the upper right-hand corner.

2 or "ul"

the legends are drawn in the upper left-hand corner.

3 or "ll"

the legends are drawn in the lower left-hand corner.

4 or "lr"

the legends are drawn in the lower right-hand corner.

5 or "?"

interactive placement with the mouse (default).

6 or "below"

the legends are drawn under the graph (which is resized accordingly).

with_box

a boolean (default value %t) which sets ot not the drawing of the box.

font_size

an integer (default value 1) which sets the size of the font used for the names in the legend.

Description

Puts a legend on the current plot using the specified strings as labels.

In the interactive placement (`opt=5` or `opt="?"`) move the legend box with the mouse and press the left button to release it.

This function allows more flexible placement of the legends than the `leg` plot2d argument.

Examples

```
// Example 1
t=0:0.1:2*pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3],opt="lr")

scf() ;
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt="?")

// Example 2
scf() ;
subplot(221)
t=0:0.1:2*pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=3 )

subplot(222)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=6 )

subplot(223)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=1, font_size=2 )

subplot(224)
t=0:0.1:2*pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=2, font_size=1 )
```

See Also

`plot2d` , `xstring` , `xtitle` , `legend`

Name

locate — mouse selection of a set of points

```
x=locate([n,flag])
```

Parameters

x
matrix of size (2,n1). n1=n if the parameter n is given.

n,flag
integer values.

Description

`locate` is used to get the coordinates of one or more points selected with the mouse in a graphics window. The coordinates are given using the current graphics scale.

If $n > 0$, n points are selected and their coordinates are returned in the matrix **x**.

If $n \leq 0$, points are selected until the user clicks with the left button of the mouse which stands for stop. The last point (clicked with the left button) is not returned.

`x=locate()` is the same as `x=locate(-1)`.

If `flag=1` a cross is drawn at the points where the mouse is clicked.

See Also

`xclick` , `xgetmouse`

Authors

S.S. & J.Ph.C

Name

Math rendering in Scilab graphics — Display mathematical equations in Scilab graphics through the LaTeX or MathML languages.

Usage

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

LaTeX texts must start and end by \$ (dollar symbol) while MathML texts must start by < and end by > and being syntactically valide.

On the first use (these libraries are loading on the fly only when needed), note that the MathML engine is slower to load than LaTeX.

```
// Example with LaTeX / MathML ticks:
plot2d();
a=gca();

mathml="<mrow>;<mfrac><mrow><mi>d</mi><mi>y</mi></mrow><mrow><mi>d</mi><mi>x</mi></mrow></mfrac><mn>1</mn><msup><mi>y</mi><mn>2</mn></msup></mfrac></mrow>";
// LaTeX and MathML mixed expression
a.x_ticks.labels=[mathml;"1";"$\sin(x)$";"3";"$\cos(a) - test$";"5";"6";"7"];
```

LaTeX description

The rendering engine is based on the Java library JLaTeXMath. JLaTeXMath is an implementation of the mathematic mode of LaTeX. All LaTeX base commands are handle (don't hesitate to submit a bug report if missing). On the contrary, TeX commands, like `\over` are not supported.

Since Scilab 5.2.1, greek, cyrillic and complete unicode for latin alphabets characters are handled in the Scilab graphics

```
xtitle('$\textstyle\sum_{n=1}^{+\infty}\frac{1}{n^2}=\frac{\pi^2}{6}$')
xtitle('$\big(\bigg)$')
xtitle('$\mbox{Vector field for }\ddot{\theta}=\sin\theta$')
xtitle('$\JLaTeXMath\ \mathfrak{and}\ \mathtt{Scilab}$')
xstring(0.5,0.5,"$\overbrace{\#####\ \ \ \ }\underbrace{\#####\ \ \ \mu\#}$")
```

JLaTeXMath provides several fonts with the commands `\mathbb`, `\mathscr`, `\mathcal`, `\mathbf`, `\mathit`, `\mathsf`, `\mathtt`, `\mathfrak`, `\mathds`, `\mathrm`, with their bold versions when they are available with the command `\boldsymbol` :

```
xtitle('$\mathbb{SCILAB}\ \mathsf{or}\ \boldsymbol{\mathfrak{Scilab}}$')
xtitle('$\mathscr{C}\mbox{ n'est pas }\boldsymbol{\mathcal{C}}$')
```

Different LaTeX packages are available: *amsmath*, *amssymb*, *stmaryrd*, *amsxtra* and *ac-cent*s with some commands of *graphics*. Most of the commands of these packages are available (some of *amsmath* are missing for example).

```
xtitle('$\sideset{_\alpha^\beta}{_\gamma^\delta}\prod$')
xtitle('$\hat{\accentset{\star}{\hat h}}\undertilde{ABC}$')
```

```

xtitle('$\begin{pmatrix}\mathfrak{a}&\alpha\\\mathbb{A}&\mathcal{A}\end{pmatrix}\n
\begin{bmatrix}\mathfrak{a}&\alpha\\\mathbb{A}&\mathcal{A}\end{bmatrix}$')
xstring(0.5,0.5,'$\left(\frac{\pi}{\sqrt{3}}\right)^2\middle|\sqrt{\frac{1+\frac{1}{x}}{x}}$')
xtitle('$\doublecup\ddag\fatbslash\lll\oplus\ovee\circledcirc\circlearrowright$')
xtitle('$\rotatebox{180}{\boxed{JLaTeXMath}}\ \reflectbox{JLaTeXMath}$')
xtitle('$\scalebox{0.6}{\sum_{n=1}^{+\infty}\frac{1}{n^2}=\frac{\pi^2}{6}}$')
xtitle('$\fcolorbox{black}{Tan}{JLaTeXMath}$')
xtitle('$\textcolor{Magenta}{\mathfrak{Scilab}}\mbox{ and }\textcolor{Green}{\mathfrak{M}}$')

```

It is also possible to define new commands or new environments:

```

xtitle('$\newcommand{\op}{\left(}\newcommand{\cp}{\right)}\ \op\frac{12}{cp}$')
xtitle('$\newcommand{\myfrac}[2]{\frac{\mathfrak{#1}}{\mathcal{#2}}}\myfrac{A}{B}$')

```

MathML description

The MathML rendering is based on Jeulid. Jeulid is a MathML implementation which covers the whole specification. Therefore, all the MathML language is supported within Scilab.

Due to the size of the Jeulid library, on the first use, it can take up to a few seconds to load. However, next uses are much faster.

```

plot3d();
a=get("current_axes");
a.x_label.font_size= 5;
a.x_label.text="<mrow><mfrac><mrow><mn>1</mn></mrow><mrow><mn>2</mn></mrow></mfrac></mrow><mfrac><mrow><mfrac><mrow><mi>a</mi></mrow><mrow><mi>b</mi></mrow></mfrac></mrow><mfrac><mrow><mi>c</mi></mrow><mrow><mi>d</mi></mrow></mfrac></mrow></mfrac></mrow></mfenced>"

```

See Also

xtitle, axes_properties, label_properties, legend_properties, text_properties, xstringb, xstringl, xstring

Name

mesh — 3D mesh plot

```
mesh(Z)
mesh(X,Y,Z)
mesh(...,<GlobalProperty>)
mesh(...,<color>,<GlobalProperty>)
mesh(<axes_handle>,...)
```

Parameters

Z

a real matrix defining the surface height. It can not be omitted. The Z data is a $m \times n$ matrix.

X,Y

two real matrices : always set together, these data defines a new standard grid. This new X and Y components of the grid must match Z dimensions (see description below).

color

an optional real matrix defining a color value for each $(X(j), Y(i))$ point of the grid (see description below).

<GlobalProperty>

This optional argument represents a sequence of couple statements `{PropertyName, PropertyValue}` that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see GlobalProperty).

<axes_handle>

This optional argument forces the plot to appear inside the selected axes given by `axes_handle` rather than the current axes (see gca).

Description

`mesh` draws a parametric surface using a rectangular grid defined by X and Y coordinates (if `{X, Y}` are not specified, this grid is determined using the dimensions of the Z matrix) ; at each point of this grid, a Z coordinate is given using the Z matrix. `mesh` is based on the `surf` command with default option `color_mode = white index` (inside the current colormap) and `color_flag = 0`.

Data entry specification :

In this paragraph and to be more clear, we won't mention `GlobalProperty` optional arguments as they do not interfere with entry data (except for "Xdata", "Ydata" and "Zdata" property, see GlobalProperty). It is assumed that all those optional arguments could be present too.

If Z is the only matrix specified, (Z) plots the matrix Z versus the grid defined by `1:size(Z,2)` along the x axis and `1:size(Z,1)` along the y axis.

Remarks

To enable the transparency mode you should set the `color_mode` option to 0.

Examples

```
[X,Y]=meshgrid(-1:.1:1,-1:.1:1);
Z=X.^2-Y.^2;
```

```
xtitle('z=x2-y2');  
mesh(X,Y,Z);
```

See Also

[surf](#) , [meshgrid](#) , [plot2d](#) , [LineSpec](#) , [GlobalProperty](#)

Authors

F.Belahcene

Name

milk_drop — milk drop 3D function

```
z=milk_drop(x,y)
```

Parameters

x,y
two row vectors of size n1 and n2.

z
matrix of size (n1,n2).

Description

milk_drop is a function representing the surface of a milk drop falling down into milk. It can be used to test functions `eval3d` and `plot3d`.

Examples

```
x=-2:0.1:2; y=x;  
z=eval3d(milk_drop,x,y);  
plot3d(x,y,z)
```

See Also

`eval3d` , `plot3d`

Authors

Steer S.

Name

move — move, translate, a graphic entity and its children.

```
move(h, t)
move(h, t, "alone")
```

Parameters

h

a handle, the handle of the entity to move.

t

an array, either [dx, dy] or [dx,dy,dz], which gives the translation vector to apply.

"alone"

string keyword (optional).

Description

This routine can be used to apply a translation to a graphics entity. If the entity has children, they will be also translated.

Given the keyword "alone" , only the specified entity needs to be redrawn. It must specially be used with the `pixel_drawing_mode` property of the figure entity (see draw objects under "xor" drawing mode).

Examples

```
plot3d();
a=gca();
p=a.children(1);
t=[1, 0, -1.25];
move(p, t);
```

See Also

`get` , `set` , `draw` , `figure_properties` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

name2rgb — returns the RGB values of a named color

```
rgb=name2rgb(name)
```

Parameters

name

name of the color.

rgb

vector of RGB integer values of a color.

Description

name2rgb returns the RGB values of a color given by its name. The result is a vector $[r, g, b]$ where r , g and b are integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color.

If no color is found $[]$ is returned.

The list of all known colors is given by color_list.

Examples

```
rgb=name2rgb("gold")  
rgb2name(rgb)
```

See Also

color , color_list , rgb2name

Name

`newaxes` — Creates a new Axes entity

```
a=newaxes()
```

Parameters

`a`
a handle, the handle on the newly created Axes entity

Description

`newaxes()` is used to create a new `AXES` entity (see `graphics_entities`) in the current figure. The properties of this entity are inherited from the `default_axes` entity (see `gda`)

Examples

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t),'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))
legend('sinh')

sca(a1); //make first axes current
plot(t,asinh(t),'g')
legend(['acosh','asinh'])
```

See Also

`subplot` , `gda` , `sca`

Authors

S. Steer, INRIA

Name

nf3d — rectangular facets to plot3d parameters

```
[xx,yy,zz]=nf3d(x,y,z)
```

Parameters

x,y,x,xx,yy,zz
6 real matrices

Description

Utility function. Used for transforming rectangular facets coded in three matrices x,y,z to scilab code for facets accepted by plot3d.

Examples

```
//A sphere...  
u = linspace(-%pi/2,%pi/2,40);  
v = linspace(0,2*%pi,20);  
x= cos(u)'*cos(v);  
y= cos(u)'*sin(v);  
z= sin(u)'*ones(v);  
//plot3d2(x,y,z) is equivalent to...  
[xx,yy,zz]=nf3d(x,y,z); plot3d(xx,yy,zz)
```

See Also

plot3d , plot3d2

Name

`object_editor` — description of the graphic object editor capacities

`graphic` — description of the graphic object editor capacities

`menus` — description of the graphic object editor capacities

Description

Scilab graphics allow the user to have interaction with graphics before and after having them drawn. Each graphics window and the drawing it contains are represented by hierarchical entities. The hierarchy top level is the Figure. Each Figure defines at least one child of type Axes. Each Axes entity contains a set of leaf entities which are the basic graphics objects like Polylines, Rectangles, Arcs, Segs,... It can also contain a Compound type which are recursive sets of entities.

The main interest of the graphic mode is to make property changes easier. This graphics mode provides a set of high-level graphing routines (see `set`, `get`) used to control objects' properties such as data, coordinates and scaling, color and appearances without requiring to replay the initial graphics commands.

Graphics entities are associated to Scilab variables of type `handle`. The handle is a unique identifier which is associated to each instance of a created graphical entity. Using this handle, it will be possible to reach entities' properties through "set" and "get" routines. The handles are also used to manipulate graphics objects, to move them, to make copies or delete them.

To complete and use the graphics handle capacity at its maximum, a graphic object editor has been created too. It is a set of Tcl/Tk interfaces available for each kind of graphics objects (see `graphics_entities` for more details) that can be enabled for each graphic window. To make it work, select the `Edit` menu in the graphic window. Seven graphics editing operations are available :

Select figure as current:

Let this

figure be the current one.

Redraw figure:

Redraw the content of the graphics window.

Erase figure:

Erase the content of the graphics window. Its action corresponds to `clf` routine.

Copy object:

Using the mouse, it allows the user to select a 2D object (like a curve, a rectangle...) and put it in the clipboard. Thus, by a next call to `Paste object`, the object is copied in the selected current axes.

Paste object:

Allow the user to paste a previous object put into in the clipboard inside the selected current axes.

Move object:

Using the mouse, it allows the user to move a 2D object (like a curve, a rectangle...) inside the selected current axes.

Delete object:

Using the mouse, it allows the user to pick up a 2D object (like a curve, a rectangle...) inside the selected current axes and to delete it instantly.

Figure Properties:

Launch the Tcl/Tk interface for the Figure object applied to the figure handle of the graphics window.

Current Axes Properties:

Launch the Tcl/Tk interface for the Axes object applied to the current axes handle of the graphics window.

Start Entity Picker:

Start an event handler on the graphics window to catch the mouse clicks on graphics objects and launch the corresponding Tcl/Tk interface. The left mouse-click allows object edition and the right click performs a move of the selected object. Note that, for now, this feature is applied to 2D objects only.

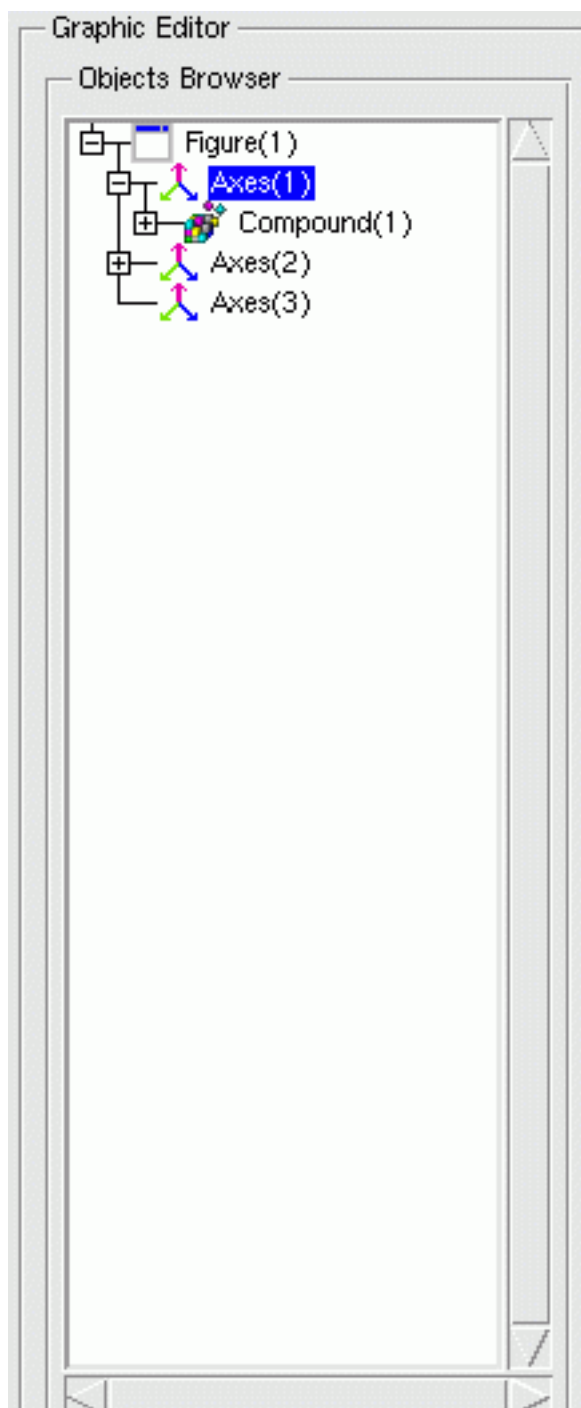
Stop Entity Picker:

Stop the action of the Entity Picker by terminating the event handler on the graphics window.

Once the graphic interface is enabled (using the **Figure Properties** or **Current Axes Properties** options), two main areas appear :

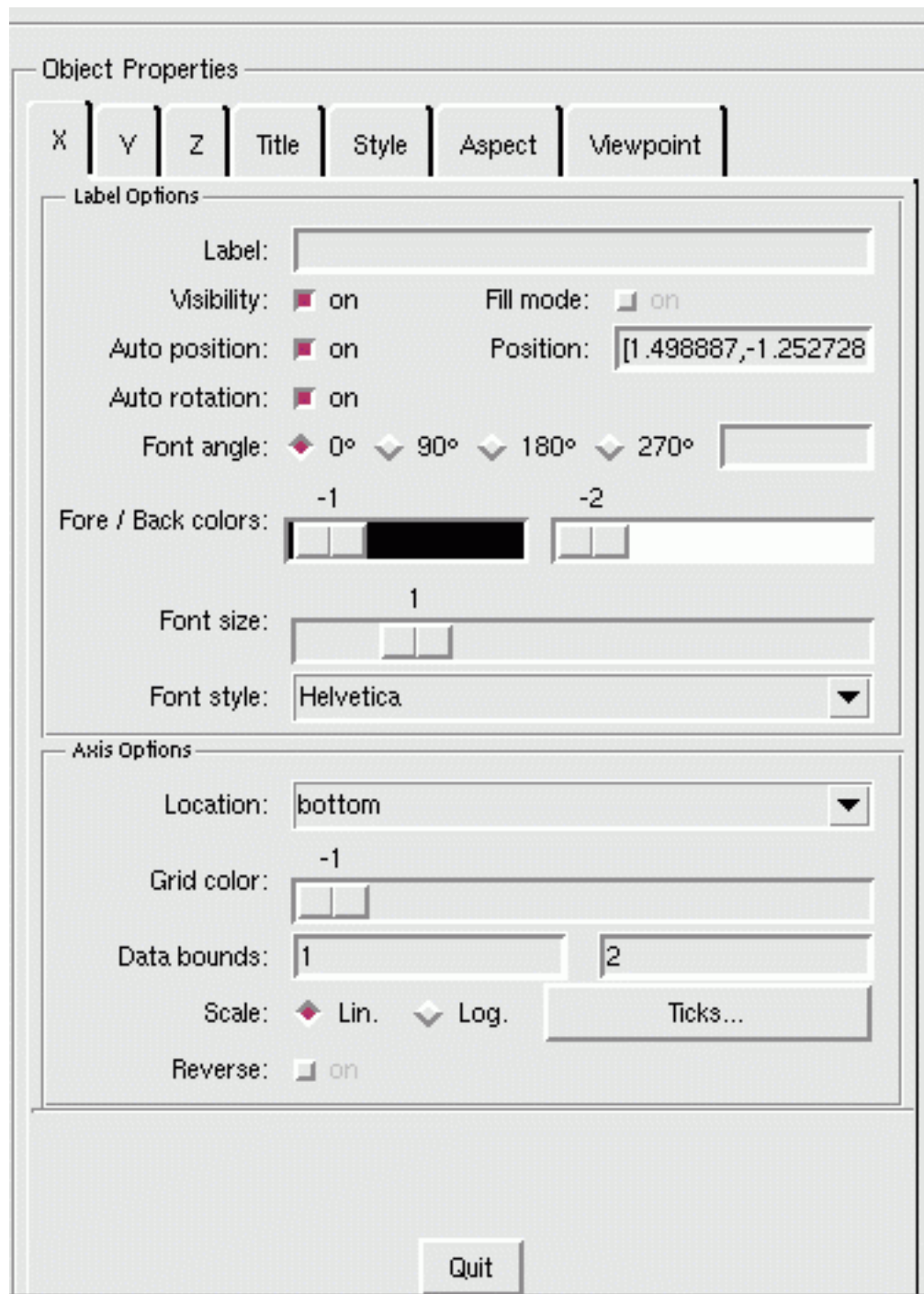
A tree selector:

Placed on the left side of the graphical editor, a hierarchical tree selector specifies which object is currently edited. It can be used to switch from a graphic object to another provided that they are in the same graphic window.



A notebook:

The second area represents a notebook composed with different properties pages (like **Style**, **Data**, **Clipping...**) depending on the selected graphic object. Using this editor, man can edit more easily the whole properties set of each graphic object (like through the "set" and "get" commands). Here is an example of the axes' notebook displaying axes properties:



Furthermore, you can legend/annotate your figure using sample primitives given inside the **Insert** menu in the graphic window. Using the mouse and following the instruction in the message subwindow, you can add a:

Line:

Draw a line between 2 left mouse clicks. The line lives in the axes where the first point was selected.

Polyline:

Draw a polyline by clicking on the left button to define the line path and right click at last to complete the drawing. The polyline lives in the axes where the first point was selected.

Arrow:

Draw an arrow between 2 left mouse clicks. The arrow lives in the axes where the first point was selected.

Double arrow:

Draw a double-sided arrow between 2 left mouse clicks. The double arrow lives in the axes where the first point was selected.

Text:

Open a dialog box to enter the text, then click on the figure window to place it. The text lives in the axes where the point was selected.

Rectangle:

Draw a rectangle : 2 left mouse clicks define respectively the upper left corner and the lower-right corner of the rectangle. The rectangle lives in the axes where the first point was selected.

Circle:

Draw a circle : 2 left mouse clicks define respectively the upper left corner and the lower-right corner of the bounding-box where the circle lives. The rectangle lives in the axes where the first point was selected.

See Also

graphics_entities , set , get , clf , plot

Authors

F.Leray INRIA

Name

oceancolormap — linear blue colormap

```
cmap=oceancolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

oceancolormap computes a colormap with with n blue colors varying linearly from black to white.

Examples

```
f = scf();  
plot3d1();  
f.color_map = oceancolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

oldplot — simple plot (old version)

```
oldplot(x,y,[xcap,ycap,caption])  
oldplot(y)
```

Parameters

x,y
two vectors with same sizes

xcap,ycap,caption
character strings or string matrices

Description

Plot *y* as function of *x*. *xcap* and *ycap* are captions for x-axis and y-axis respectively and *caption* is the caption of the plot.

Invoked with only one argument, `oldplot(y)` plots the *y* vector or, if *y* is a matrix, it plots all its row vectors on the same plot. This plot is done with respect to the vector 1:<number of columns of *y*>.

`oldplot` is obsolete. Use `plot2d` or `plot` instead.

Examples

```
x=0:0.1:2*pi;  
// simple plot  
oldplot(sin(x))  
// using captions  
clf()  
oldplot(x,sin(x),"sin","time","plot of sinus")  
// plot 2 functions  
clf()  
oldplot([sin(x);cos(x)])
```

See Also

`plot2d` , `plot`

Authors

J.Ph.C.

Name

param3d — 3D plot of a parametric curve

```
param3d(x, y, z, [theta, alpha, leg, flag, ebox])
```

Parameters

x,y,z

three vectors of the same size (points of the parametric curve).

theta, alpha

real values giving in degree the spherical coordinates of the observation point. *The default values are 35 and 45 degree.*

leg

string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag=[type,box]

type and **box** have the same meaning as in `plot3d`:

type

an integer (scaling).

type=0

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data. *This is the default value.*

type=3

3d isometric with box bounds given by optional `ebox`, similarly to `type=1`.

type=4

3d isometric bounds derived from the data, similarly to `type=2`.

type=5

3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`.

type=6

3d expanded isometric bounds derived from the data, similarly to `type=2`. Note that axes boundaries can be customized through the axes entity properties (see `axes_properties`).

box

an integer (frame around the plot).

box=0

nothing is drawn around the plot.

box=1

unimplemented (like `box=0`).

box=2

only the axes behind the surface are drawn.

`box=3`

a box surrounding the surface is drawn and captions are added.

`box=4`

a box surrounding the surface is drawn, captions and axes are added. Note that axes aspect can also be customized through the axes entity properties (see `axes_properties`). *This is the default value.*

`ebox`

It specifies the boundaries of the plot as the vector `[xmin, xmax, ymin, ymax, zmin, zmax]`. This argument is used together with `type` in `flag` : if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into account. Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see `axes_properties`). The `ebox` default value is `[0,1,0,1,0,1]`.

Description

`param3d` is used to plot a 3D curve defined by its coordinates `x`, `y` and `z`. Note that data can also be got or modified through the surface entity properties (see `surface_properties`).

Note that properties like `rotation_angles`, `colors` and `thickness` of the plotted curves can also be got or modified through the polyline entity properties (see `polyline_properties`).

Use `param3d1` to do multiple plots.

Enter the command `param3d( )` to see a demo.

Examples

```
t=0:0.1:5*pi;
param3d(sin(t),cos(t),t/10,35,45,"X@Y@Z",[2,3])

e=gce() //the handle on the 3D polyline

e.foreground=color('red');

a=gca(); //the handle on the axes
a.rotation_angles=[10 70];
```

See Also

`param3d1`, `plot3d`

Authors

J.Ph.C.

Name

param3d1 — 3D plot of parametric curves

```
param3d1(x,y,z,[theta,alpha,leg,flag,ebox])  
param3d1(x,y,list(z,colors),[theta,alpha,leg,flag,ebox])
```

Parameters

x,y,z

matrices of the same size (nl,nc).

Each column *i* of the matrices corresponds to the coordinates of the *i*th curve. You can give a specific color for each curve by using `list(z,colors)` instead of `z`, where `colors` is a vector of size `nc`. If `color(i)` is negative the curve is plotted using the mark with id `abs(style(i))`; if `style(i)` is strictly positive, a plain line with color id `style(i)` or a dashed line with dash id `style(i)` is used.

theta,alpha

real values giving in degree the spherical coordinates of the observation point. *The default values are 35 and 45 degree.*

leg

string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

flag=[type,box]

`type` and `box` have the same meaning as in `plot3d`:

type

an integer (scaling).

type=0

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data. *This is the default value.*

type=3

3d isometric with box bounds given by optional `ebox`, similarly to `type=1`.

type=4

3d isometric bounds derived from the data, similarly to `type=2`.

type=5

3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`.

type=6

3d expanded isometric bounds derived from the data, similarly to `type=2`. Note that axes boundaries can be customized through the axes entity properties (see `axes_properties`).

box

an integer (frame around the plot).

box=0
nothing is drawn around the plot.

box=1
unimplemented (like box=0).

box=2
only the axes behind the surface are drawn.

box=3
a box surrounding the surface is drawn and captions are added.

box=4
a box surrounding the surface is drawn, captions and axes are added. Note that axes aspect can also be customized through the axes entity properties (see axes_properties). *This is the default value.*

ebox
It specifies the boundaries of the plot as the vector [xmin, xmax, ymin, ymax, zmin, zmax]. This argument is used together with type in flag : if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If flag is missing, ebox is not taken into account. Note that, when specified, the ebox argument acts on the data_bounds field that can also be reset through the axes entity properties (see axes_properties). The ebox default value is [0, 1, 0, 1, 0, 1].

Description

param3d1 is used to plot 3D curves defined by their coordinates x, y and z. Note that data can also be got or modified through the surface entity properties (see surface_properties).

Note that properties like rotation angles, colors and thickness of the plotted curves can also be got or modified through the polyline entity properties (see polyline_properties).

Enter the command param3d1() to see a demo.

Examples

```
xset('window',20) // create a window number 20
t=[0:0.1:5*pi]';
param3d1([sin(t), sin(2*t)], [cos(t), cos(2*t)], ...
         list([t/10, sin(t)], [3, 2]), 35, 45, "X@Y@Z", [2, 3])

xdel(20);
a=get("current_axes");//get the handle of the newly created axes
t=[0:0.1:5*pi]';
param3d1([sin(t), sin(2*t)], [cos(t), cos(2*t)], [t/10, sin(t)])
a.rotation_angles=[65, 75];
a.data_bounds=[-1, -1, -1; 1, 1, 2]; //boundaries given by data_bounds
a.thickness = 2;
h=a.children //get the handle of the param3d entity: an Compound composed of 2
h.children(1).foreground = 3 // first curve
curve2 = h.children(2);
curve2.foreground = 6;
curve2.mark_style = 2;
```

See Also

param3d, plot3d, plot2d, gca, xdel, delete

Authors

J.Ph.C.

Name

paramfplot2d — animated 2D plot, curve defined by a function

```
paramfplot2d(f,x,theta)
paramfplot2d(f,x,theta,flag)
paramfplot2d(f,x,theta,flagrect)
```

Parameters

x
real vector.

f
function $y=f(x, t)$. f is a Scilab function or a dynamically linked routine (referred to as a string).

theta
real vector (set of parameters).

flag
string 'no' or 'yes': If "yes" screen is cleared between two consecutive plots.

rect
"rectangle" [xmin, xmax, ymin, ymax] (1 x 4 real vector),

Description

Animated plot of the function $x \rightarrow f(x,t)$ for $t=\text{theta}(1), \text{theta}(2), \text{etc.}$ f can be a either Scilab function or a dynamically linked routine since $y=f(x,t)$ is evaluated as $y=\text{feval}(x(:),t,f)$. See feval. f : mapping $x,t \rightarrow f(x,t) = \mathbb{R}^N$ valued function for $x = \text{vector of } \mathbb{R}^N$ and $t = \text{real number}$. x is a N -vector of x -values and for each t in theta , $f(x,t) = N$ -vector of y -values.

Examples

```
deff('y=f(x,t)', 'y=t*sin(x)')
x=linspace(0, 2*pi, 50); theta=0:0.05:1;
paramfplot2d(f,x,theta);
```

See Also

plot2d , feval , fplot2d

Name

pie — draw a pie

```
pie(x)
pie(x[, sp[, txt]])
```

Parameters

x
a scalar or a vector of positive reals.

sp
a real scalar or a vector of reals.

txt
a cell or a vector of strings.

Description

pie(x): if size of **x** is **N** then pie function draws a pie with **N** parts, the area of the **ith** part is equal to $(x(i)/\text{sum}(x)) \times (\text{surface of the unit cercle})$.

pie(x, sp): the **sp** vector allows to cut one or several parts of the pie, (the size of **sp** must be equal to **N**). if the value of the **ith** index of **sp** is different of zero then the **ith** part is separated from the others by a space, else if it's equal to zero then it is attached to the others.

pie(x, txt): the **txt** vector allows to write a text for each part of the pie, the **ith** component of **txt** corresponds to the **ith** part (default : it's written the percentages which corresponds to the parts surface). The size of **txt** must be equal to **N**.

Examples

```
// First example : one input argument  x=[1 2 5]
scf(0);
pie([1 2 5]);

// Second example : two input arguments x=[5 9 4 6 3], sp=[0 1 0 1 0], the second part is separated
scf(1);
pie([5 9 4 6 3],[0 1 0 1 0]);

// Third example : three input arguments, x=[3 4 6 2], sp=[0 1 0 0], txt=["part1","part2","part3","part4"]
scf(2);
pie([3 4 6 2],[0 1 0 0],["part1","part2","part3","part4"]);
```

See Also

xfpolys

Authors

Farid Belahcene

Name

pinkcolormap — sepia tone colorization on black and white images

```
cmap=pinkcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

pinkcolormap computes a colormap that provides sepia tone colorization on black and white images

Examples

```
f = scf();  
plot3d1();  
f.color_map = pinkcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

plot — 2D plot

```
plot(y,<LineStyle>,<GlobalProperty>)
plot(x,y,<LineStyle>,<GlobalProperty>)
plot(x1,y1,<LineStyle1>,x2,y2,<LineStyle2>,...xN,yN,<LineStyleN>,<GlobalProperty1>
plot(<axes_handle>,...)
```

Parameters

x

a real matrice or vector. If omitted, it is assumed to be the vector `1:n` where `n` is the number of curve points given by the `y` parameter.

y

a real matrice or vector. `y` can also be a function defined as a macro or a primitive.

<LineStyle>

This optional argument must be a string that will be used as a shortcut to specify a way of drawing a line. We can have one `LineStyle` per `y` or `{x,y}` previously entered. `LineStyle` options deals with `LineStyle`, `Marker` and `Color` specifiers (see `LineStyle`). Those specifiers determine the line style, mark style and color of the plotted lines.

<GlobalProperty>

This optional argument represents a sequence of couple statements `{PropertyName, PropertyValue}` that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see `GlobalProperty`).

<axes_handle>

This optional argument forces the plot to appear inside the selected axes given by `axes_handle` rather than the current axes (see `gca`).

Description

`plot` plots a set of 2D curves. `plot` has been rebuild to better handle Matlab syntax. To improve graphical compatibility, Matlab users should use `plot` (rather than `plot2d`).

Data entry specification :

In this paragraph and to be more clear, we won't mention `LineStyle` nor `GlobalProperty` optional arguments as they do not interfere with entry data (except for `"Xdata"`, `"Ydata"` and `"Zdata"` property, see `GlobalProperty`). It is assumed that all those optional arguments could be present too.

If `y` is a vector, `plot(y)` plots vector `y` versus vector `1:size(y, ' * ')`.

If `y` is a matrix, `plot(y)` plots each columns of `y` versus vector `1:size(y, 1)`.

If `x` and `y` are vectors, `plot(x,y)` plots vector `y` versus vector `x`. `x` and `y` vectors should have the same number of entries.

If `x` is a vector and `y` a matrix `plot(x,y)` plots each columns of `y` versus vector `x`. In this case the number of columns of `y` should be equal to the number of `x` entries.

If `x` and `y` are matrices, `plot(x,y)` plots each columns of `y` versus corresponding column of `x`. In this case the `x` and `y` sizes should be the same.

Finally, if only `x` or `y` is a matrix, the vector is plotted versus the rows or columns of the matrix. The choice is made depending on whether the vector's row or column dimension matches the matrix row or column dimension. In case of a square matrix (on `x` or `y` only), priority is given to columns rather than lines (see examples below).

y can also be a function defined as a macro or a primitive. In this case, x data must be given (as a vector or matrix) and the corresponding computation $y(x)$ is done implicitly.

The `LineSpec` and `GlobalProperty` arguments could be used to customize the plot. Here is a complete list of the available options.

LineSpec

This option may be used to specify, in a short and easy manner, how the curves are drawn. It must always be a string containing references to `LineStyle`, `Marker` and `Color` specifiers.

These references must be set inside the string (order is not important) in an unambiguous way. For example, to specify a red long-dashed line with the diamond mark enabled, you can write : `'r--d'` or `'--dire'` or `'--reddiam'` or another unambiguous statement... or to be totally complete `'diamondred--'` (see `LineSpec`).

Note that the line style and color, marks color (and sizes) can also be (re-)set through the polyline entity properties (see `polyline_properties`).

GlobalProperty

This option may be used to specify how all the curves are plotted using more option than via `LineSpec`. It must always be a couple statement constituted of a string defining the `PropertyName`, and its associated value `PropertyValue` (which can be a string or an integer or... as well depending on the type of the `PropertyName`). Using `GlobalProperty`, you can set multiple properties : every properties available via `LineSpec` and more : the marker color (foreground and background), the visibility, clipping and thickness of the curves. (see `GlobalProperty`)

Note that all these properties can be (re-)set through the polyline entity properties (see `polyline_properties`).

Remarks

By default, successive plots are superposed. To clear the previous plot, use `clf()`. To enable `auto_clear` mode as the default mode, edit your default axes doing:

```
da=gda();
```

```
da.auto_clear = 'on'
```

For a better display `plot` function may modify the `box` property of its parent Axes. This happens when the parent Axes were created by the call to `plot` or were empty before the call. If one of the axis is centered at origin, the box is disabled. Otherwise, the box is enabled.

For more information about box property and axis positionning see `axes_properties`

A default color table is used to color plotted curves if you do not specify a color. When drawing multiple lines, the plot command automatically cycles through this table. Here are the used colors:

R	G	B
0.	0.	1.
0.	0.5	0.
1.	0.	0.
0.	0.75	0.75
0.75	0.	0.75
0.75	0.75	0.
0.25	0.25	0.25

Enter the command `plot` to see a demo.

Examples

```
// x initialisation
x=[0:0.1:2*pi]';
//simple plot
plot(sin(x))
clf()
plot(x,sin(x))
//multiple plot
clf()
plot(x,[sin(x) sin(2*x) sin(3*x)])
clf()

// axis on the right
plot(x,sin(x))
a=gca(); // Handle on current axes entity
a.y_location = "right";
clf()

// axis centered at (0,0)
plot(x-4,sin(x),x+2,cos(x))
a=gca(); // Handle on axes entity
a.x_location = "origin";
a.y_location = "origin";

// Some operations on entities created by plot ...
a=gca();
a.isoview='on';
a.children // list the children of the axes : here it is an Compound child comp
poly1= a.children.children(2); //store polyline handle into poly1
poly1.foreground = 4; // another way to change the style...
poly1.thickness = 3; // ...and the tickness of a curve.
poly1.clip_state='off' // clipping control
a.isoview='off';

//LineSpec and GlobalProperty examples:
clf();
t=0:%pi/20:2*pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
scf(2)
plot([t ;t],[sin(t) ;cos(t)],'xdat',[1:2])
scf(3)
axfig3 = gca();
scf(4) // should remain blank
plot(axfig3,[t ;t],[sin(t) ;cos(t)],'zdat',[1:2],'marker','d','markerfac','green')
xdel(winsid())

//Data specification
t=-%pi:0.1:%pi;
size(t)
plot(t) // simply plots y versus t vector size
clf(); // clear figure

plot(t,sin(t)); // plots sin(t) versus t
clf();
```

```

t=[1      1      1      1
   2      3      4      5
   3      4      5      6
   4      5      6      7];

plot(t) // plots each t column versus row size
clf();

subplot(221)
plot(t,sin(t)); // plots sin(t) versus t column by column this time
xlabel("sin(t) versus t")
subplot(222)
plot(t,sin(t)')
xlabel("sin(t)'' versus t")
subplot(223)
plot(t',sin(t))
a=gca();
a.data_bounds=[0 -1;7 1]; // to see the vertical line hidden by the y axis
xlabel("sin(t) versus t'")
subplot(224)
plot(t',sin(t)')
xlabel("sin(t)'' versus t'")

clf();

//Special case 1
//x : vector ([5 6 7 8]) and y : matrix (t)
x=[5 6 7 8]
plot(x,t);
plot(x',t); // idem, x is automatically transposed to match t (here the columns)
clf()

// Only one matching possibility case : how to make 4 identical plots in 4 manners
// x is 1x4 (vector) and y is 4x5 (non square matrix)
subplot(221);
plot(x,[t [8;9;10;12]]');
subplot(222);
plot(x',[t [8;9;10;12]]');
subplot(223);
plot(x,[t [8;9;10;12]]');
subplot(224);
plot(x',[t [8;9;10;12]]');
clf()

//Special case 2
// Case where only x or y is a square matrix
//x : matrix (t) and y : vector ([1 2 3 4])
plot(t,[1 2 3 4]) // equivalent to plot(t,[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t,[1;2;3;4]) // the same plot
clf();

// t is transposed : notice the priority given to the columns treatment
plot(t',[1 2 3 4]) // equivalent to plot(t',[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t',[1 2 3 4]') // the same plot
clf();

// y is a function defined by..
// ..a primitive

```

```
plot(1:0.1:10,sin) // equivalent to plot(1:0.1:10,sin(1:0.1:10))
clf();

// ..a macro:
deff('[y]=toto(x)', 'y=x.*x')
plot(1:10,toto)
```

See Also

plot2d , surf , scf , clf , xdel , delete , LineSpec , GlobalProperty

Authors

F.Leray

Name

plot2d — 2D plot

```
plot2d([x],y)
plot2d([x],y,<opt_args>)
```

Parameters

- x**
a real matrix or vector. If omitted, it is assumed to be the vector $1:n$ where n is the number of curve points given by the **y** parameter.
- y**
a real matrix or vector.
- <opt_args>**
This represents a sequence of statements **key1=value1, key2=value2,...** where **key1, key2, . . .** can be one of the following:
- style**
sets the style for each curve. The associated value should be a real vector with integer (positive or negative) values.
- rect**
sets the minimal bounds requested for the plot. The associated value should be a real vector with four entries: **[xmin, ymin, xmax, ymax]**.
- logflag**
sets the scale (linear or logarithmic) along the axes. The associated value should be a string with possible values: "nn", "nl", "ln" and "ll".
- frameflag**
controls the computation of the actual coordinate ranges from the minimal requested values. The associated value should be an integer ranging from 0 to 8.
- axesflag**
specifies how the axes are drawn. The associated value should be an integer ranging from 0 to 5.
- nax**
sets the axes labels and tics definition. The associated value should be a real vector with four integer entries **[nx, Nx, ny, Ny]**.
- leg**
sets the curves captions. The associated value should be a character string.

Description

plot2d plots a set of 2D curves. If you are familiar with Matlab `plot` syntax, you should use `plot`.

If **x** and **y** are vectors, `plot2d(x,y,<opt_args>)` plots vector **y** versus vector **x**. **x** and **y** vectors should have the same number of entries.

If **x** is a vector and **y** a matrix `plot2d(x,y,<opt_args>)` plots each columns of **y** versus vector **x**. In this case the number of columns of **y** should be equal to the number of **x** entries.

If **x** and **y** are matrices, `plot2d(x,y,<opt_args>)` plots each columns of **y** versus corresponding column of **x**. In this case the **x** and **y** sizes should be the same.

If `y` is a vector, `plot2d(y,<opt_args>)` plots vector `y` versus vector `1:size(y, 's')`.

If `y` is a matrix, `plot2d(y,<opt_args>)` plots each columns of `y` versus vector `1:size(y, 1)`.

The `<opt_args>` arguments should be used to customize the plot

`style`

This option may be used to specify how the curves are drawn. If this option is specified, the associated value should be a vector with as many entries as curves.

- if `style(i)` is strictly positive, the curve is drawn as plain line and `style(i)` defines the index of the color used to draw the curve (see `getcolor`). Note that the line style and the thickness can be set through the `polyline` entity properties (see `polyline_properties`).

Piecewise linear interpolation is done between the given curve points.

- if `style(i)` is negative or zero, the given curve points are drawn using marks, `abs(style(i))` defines the mark with id used. Note that the marks color and marks sizes can be set through the `polyline` entity properties (see `polyline_properties`).

`logflag`

This option may be used to set the scale (linear or logarithmic) along the axes. The associated value should be a string with possible values: "nn", "nl", "ln" and "ll". "l" stands for logarithmic scale and graduations and "n" for normal scale.

`rect`

This option may be used to set the minimal bounds requested for the plot. If this option is specified, the associated value should be a real vector with four entries:

[`xmin`, `ymin`, `xmax`, `ymax`]. `xmin` and `xmax` defines the bounds on the abscissae while `ymin` and `ymax` defines the bounds on the ordinates.

This argument may be used together with the `frameflag` option to specify how the axes boundaries are derived from the given `rect` argument. If the `frameflag` option is not given, it is supposed to be `frameflag=7`.

The axes boundaries can also be customized through the axes entity properties (see `axes_properties`).

`frameflag`

This option may be used to control the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

`frameflag=0`

no computation, the plot use the previous (or default) scale.

`frameflag=1`

The actual range is the range given by the `rect` option.

`frameflag=2`

The actual range is computed from the min/max of the `x` and `y` data.

`frameflag=3`

The actual range is the range given by the `rect` option and enlarged to get an isometric scale.

`frameflag=4`

The actual range is computed from the min/max of the `x` and `y` data and enlarged to get an isometric scale.

`frameflag=5`

The actual range is the range given by the `rect` option and enlarged to get pretty axes labels.

`frameflag=6`

The actual range is computed from the min/max of the x and y data and enlarged to get pretty axes labels.

`frameflag=7`

like `frameflag=1` but the previous plot(s) are redrawn to use the new scale. Used to add the current graph to a previous one.

`frameflag=8`

like `frameflag=2` but the previous plot(s) are redrawn to use the new scale. Used to add the current graph to a previous one.

`frameflag=9`

like `frameflag=8` but the range is enlarged to get pretty axes labels. This is the default value.

The axes boundaries can also be customized through the axes entity properties (see `axes_properties`)

`axesflag`

This option may be used to specify how the axes are drawn. The associated value should be an integer ranging from 0 to 5 :

`axesflag=0`

nothing is drawn around the plot (`axes_visible=["off" "off"]`; `box="off"`).

`axesflag=1`

axes are drawn, the y-axis is displayed on the left (`axes_visible=["on" "on"]`; `box="on"`, `y_location="left"`).

`axesflag=2`

the plot is surrounded by a box without tics (`axes_visible=["off" "off"]`; `box="on"`).

`axesflag=3`

axes are drawn, the y-axis is displayed on the right (`axes_visible=["on" "on"]`; `box="off"`, `y_location="right"`).

`axesflag=4`

axes are drawn centered in the middle of the frame, the box being not drawn (`axes_visible=["on" "on"]`; `box="off"`, `x_location="middle"`, `y_location="middle"`).

`axesflag=5`

axes are drawn centered in the middle of the frame similarly to `axesflag=4`, the difference being that the box is drawn (`axes_visible=["on" "on"]`; `box="on"`, `x_location="middle"`, `y_location="middle"`).

`axesflag=9`

axes are drawn, the y-axis is displayed on the left (`axes_visible=["on" "on"]`; `box="off"`, `y_location="left"`). This is the default value

The axes aspect can also be customized through the axes entity properties (see `axes_properties`).

`nax`

This option may be used to specify the axes labels and tics definition. The associated value should be a real vector with four integer entries `[nx, Nx, ny, Ny]`.

`Nx` gives the number of main tics to be used on the x-axis (to use autoticks set it to -1), `nx` gives the number of subtics to be drawn between two main x-axis tics.

`Ny` and `ny` give similar information for the y-axis.

If `axesflag` option is not set `nax` option supposes that `axesflag` option has been set to 9.

`leg`

This option may be used to sets the curve captions. It must be a string with the form "`leg1@leg2@...`" where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is " ".

The curve captions are drawn on below the x-axis. This option is not flexible enough, use the captions or legend functions preferably.

More information

To get more information on the plot2d old syntax , use the `plot2d_old_version` help document.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d()` to see a demo.

Other high level plot2d functions exist:

- `plot2d2` same as `plot2d` but the curve is supposed to be piecewise constant.
- `plot2d3` same as `plot2d` but the curve is plotted with vertical bars.
- `plot2d4` same as `plot2d` but the curve is plotted with vertical arrows.

Examples

```
// x initialisation
x=[0:0.1:2*pi]';
//simple plot
plot2d(sin(x));
clf();
plot2d(x,sin(x));
//multiple plot
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5]);
//multiple plot with captions and given tics + style
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],...
        [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*pi,2]);
// isoview
clf();
plot2d(x,sin(x),1,frameflag= 4);
// scale
clf();
plot2d(x,sin(x),1,frameflag= 6);
// auto scaling with previous plots + style
clf();
plot2d(x,sin(x),-1);
plot2d(x,2*sin(x),12);
plot2d(2*x,cos(x),3);
// axis on the right
clf();
plot2d(x,sin(x),leg="sin(x)");
```

```
a=gca(); // Handle on axes entity
a.y_location="right";
// axis centered at (0,0)
clf();
plot2d(x-4,sin(x),1,leg="sin(x)");
a=gca(); // Handle on axes entity
a.x_location="origin";
a.y_location="origin";
// Some operations on entities created by plot2d ...
a=gca();
a.isoview='on';
a.children // list the children of the axes.
// There are a compound made of two polylines and a legend
poly1= a.children(1).children(1); //store polyline handle into poly1
poly1.foreground = 4; // another way to change the style...
poly1.thickness = 3; // ...and the tickness of a curve.
poly1.clip_state='off'; // clipping control
leg = a.children(2); // store legend handle into leg
leg.font_style = 9;
leg.line_mode = "on";
a.isoview='off';
```

See Also

[plot](#), [plot2d1](#), [plot2d2](#), [plot2d3](#), [plot2d4](#), [clf](#), [xdel](#), [delete](#)

Authors

Name

plot2d1 — 2D plot (logarithmic axes) (obsolete)

```
plot2d1(str,x,y,[style,strf,leg,rect,nax])
```

Parameters

str

is a string of length three "abc".

a

can have the following values: e, o or g.

e

means "empty". It specifies the fact that the value of x is not used (the x values are supposed to be regularly spaced, ie 1:<number of rows of y>). The user must anyway give a value for x, 1 for instance: `plot2d1("enn",1,y)`.

o

means "one". If there are many curves, they all have the same x-values: x is a column vector of size nl and y is a matrix of size (nl,nc). For example : `x=[0:0.1:2*pi]'; plot2d1("onn",x,[sin(x) cos(x)])`.

g

means "general". x and y must have the same size (nl,nc). Each column of y is plotted with respect to the corresponding column of x. nc curves are plotted using nl points.

b, c

can have the values n (normal) or l (logarithmic).

b=l

a logarithmic axis is used on the x-axis

c=l

a logarithmic axis is used on the y-axis

x,y,[style,strf,leg,rect,nax]

these arguments have the same meaning as in the `plot2d` function.

opt_args

these arguments have the same meaning as in the `plot2d` function.

Description

This function is obsolete. USE `plot2d` INSTEAD !!

`plot2d1` plots a set of 2D curves. It is the same as `plot2d` but with one more argument `str` which enables logarithmic axis. Moreover, it allows to specify only one column vector for x when it is the same for all the curves.

By default, successive plots are superposed. To clear the previous plot, use `clf`.

Enter the command `plot2d1()` to see a demo.

Examples

```
// multiple plot without giving x
```

```
x=[0:0.1:2*pi]';  
plot2d1("enn",1,[sin(x) sin(2*x) sin(3*x)])  
// multiple plot using only one x  
clf()  
plot2d1("onn",x,[sin(x) sin(2*x) sin(3*x)])  
// logarithmic plot  
x=[0.1:0.1:3]'; clf()  
plot2d1("oll",x,[exp(x) exp(x^2) exp(x^3)])
```

See Also

plot2d , plot2d2 , plot2d3 , plot2d4 , clf

Authors

J.Ph.C.

Name

plot2d2 — 2D plot (step function)

```
plot2d2([x],y)
plot2d2([x],y,<opt_args>)
plot2d2([logflag],x,y,[style,strf,leg,rect,nax])
```

Parameters

args

see `plot2d` for a description of parameters.

Description

`plot2d2` is the same as `plot2d` but the functions given by (x, y) are supposed to be piecewise constant.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d2()` to see a demo. Note that all the modes proposed by `plot2dxx` ($xx = 1$ to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```
// plots a step function of value i on the segment [i,i+1]
// the last segment is not drawn
plot2d2([1:4],[1:4],1,"111","step function",[0,0,5,5])
// compare the following with plot2d
x=[0:0.1:2*pi]';
clf()
plot2d2(x,[sin(x) sin(2*x) sin(3*x)])
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=2;
e.children(2).polyline_style=2;
e.children(3).polyline_style=2;
```

See Also

`plot2d` , `plot2d3` , `plot2d4` , `subplot` , `clf` , `polyline_properties`

Authors

J.Ph.C.

Name

plot2d3 — 2D plot (vertical bars)

```
plot2d3([logflags,] x,y,[style,strf,leg,rect,nax])
plot232(y)
plot2d3(x,y <,opt_args>)
```

Parameters

args

see `plot2d` for a description of parameters.

Description

`plot2d3` is the same as `plot2d` but curves are plotted using vertical bars.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d3()` to see a demo. Note that all the modes proposed by `plot2dxx` ($xx = 1$ to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```
// compare the following with plot2d1
x=[0:0.1:2*pi]';
plot2d3(x,[sin(x) sin(2*x) sin(3*x)])
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=3;
e.children(2).polyline_style=3;
e.children(3).polyline_style=3;
```

See Also

`plot2d` , `plot2d2` , `plot2d4` , `clf` , `polyline_properties`

Authors

J.Ph.C.

Name

plot2d4 — 2D plot (arrows style)

```
plot2d4([logflag,] x,y,[style,strf,leg,rect,nax])
plot2d4(y)
plot2d4(x,y <,opt_args>)
```

Parameters

args

see `plot2d` for a description of parameters.

Description

`plot2d4` is the same as `plot2d` but curves are plotted using arrows style. This can be useful when plotting solutions of an ODE in a phase space.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d4()` to see a demo. Note that all the modes proposed by `plot2dxx` ($xx = 1$ to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```
// compare the following with plot2d1
x=[0:0.1:2*pi]';
plot2d4(x,[sin(x) sin(2*x) sin(3*x)])
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=4;
e.children(2).polyline_style=4;
e.children(3).polyline_style=4;
```

See Also

`fchamp` , `plot2d` , `plot2d2` , `plot2d3` , `subplot` , `clf` , `polyline_properties`

Authors

J.Ph.C.

Name

plot2d_old_version — **The syntaxes described below are obsolete**

```
plot2d([logflag],x,y,[style,strf,leg,rect,nax])
```

Parameters

x,y

two matrices (or column vectors).

- in the usual way **x** is a matrix of the same size than **y** (the column **j** of **y** is plotted with respect to column **j** of **x**)
- if all the columns of **x** are equal (ie the abscissae of all the curves are the same), **x** may be simply the (column) vector of these abscissae (**x** is then a column vector of length equal to the row dimension of **y**).
- when **x** is not given, it is supposed to be the column vector [1; 2; ...; row dimension of **y**].

style

is a real row vector of size **nc**. The style to use for curve **i** is defined by **style(i)**. The default style is 1:nc (1 for the first curve, 2 for the second, etc.).

- if **style(i)** is negative or zero, the curve is plotted using the mark with id **abs(style(i))**; use **xset()** to set the mark id and **xget('mark')** to get the current mark id.
- if **style(i)** is strictly positive, a plain line with color id **style(i)** or a dashed line with dash id **style(i)** is used; use **xset()** to see the color ids.

strf

is a string of length 3 "xyz" (by default **strf= "081"**)

x

controls the display of captions.

x=0

no caption.

x=1

captions are displayed. They are given by the optional argument **leg**.

y

controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

y=0

no computation, the plot use the previus (or default) scale

y=1

from the **rect** arg

y=2

from the min/max of the **x, y** datas

y=3

built for an isometric scale from the **rect** arg

y=4

built for an isometric plot from the min/max of the **x, y** datas

y=5
enlarged for pretty axes from the rect arg

y=6
enlarged for pretty axes from the min/max of the x, y datas

y=7
like y=1 but the previous plot(s) are redrawn to use the new scale

y=8
like y=2 but the previous plot(s) are redrawn to use the new scale

z
controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the nax optional argument.

z=0
nothing is drawn around the plot.

z=1
axes are drawn, the y=axis is displayed on the left.

z=2
the plot is surrounded by a box without tics.

z=3
axes are drawn, the y=axis is displayed on the right.

z=4
axes are drawn centred in the middle of the frame box.

z=5
axes are drawn so as to cross at point $(0, 0)$. If point $(0, 0)$ does not lie inside the frame, axes will not appear on the graph.

leg
a string. It is used when the first character x of argument strf is 1. leg has the form "leg1@leg2@. . . ." where leg1, leg2, etc. are respectively the captions of the first curve, of the second curve, etc. The default is "".

rect
This argument is used when the second character y of argument strf is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: rect=[xmin, ymin, xmax, ymax].

nax
This argument is used when the third character z of argument strf is 1. It is a row vector with four entries [nx, Nx, ny, Ny] where nx (ny) is the number of subgraduations on the x (y) axis and Nx (Ny) is the number of graduations on the x (y) axis.

logflag
a string formed by two characters h (for horizontal axis) and v (for vertical axis) each of these characters can take the values "n" or "l". "l" stands for logarithmic graduation and "n" for normal graduation. For example "ll" stands for a log-log plot. Default value is "nn".

Description

plot2d plots a set of 2D curves. Piecewise linear plotting is used.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

See the meaning of the parameters above for a complete description.

Enter the command `plot2d()` to see a demo.

Other high level plot2d function exists:

- `plot2d2`: same as `plot2d` but the curve is supposed to be piecewise constant.
- `plot2d3`: same as `plot2d` but the curve is plotted with vertical bars.
- `plot2d4`: same as `plot2d` but the curve is plotted with vertical arrows.

Examples

```
//simple plot
x=[0:0.1:2*pi]';
plot2d(sin(x))
clf()
plot2d(x,sin(x))
//multiple plot
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
// old syntax and new syntax
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011","",[0,0,6,0.5])
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
//multiple plot with captions and given tics
// old syntax and new syntax
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
    [1,2,3],"111","L1@L2@L3",[0,-2,2*pi,2],[2,10,2,10]);
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
    [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*pi,2])
// isoview
clf()
plot2d(x,sin(x),1,"041")
// scale
clf()
plot2d(x,sin(x),1,"061")
// auto scaling with previous plots
clf()
plot2d(x,sin(x),1)
plot2d(x,2*sin(x),2)
plot2d(2*x,cos(x),3)
// axis on the right
clf()
plot2d(x,sin(x),1,"183","sin(x)")
// centered axis
clf()
plot2d(x,sin(x),1,"184","sin(x)")
// axis centered at (0,0)
clf()
plot2d(x-4,sin(x),1,"185","sin(x)")
```

See Also

[plot2d1](#), [plot2d2](#), [plot2d3](#), [plot2d4](#), [clf](#), [xset](#)

Authors

J.Ph.C.

Name

plot3d — 3D plot of a surface

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(x,y,z,<opt_args>)

plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,<opt_args>)

plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),<opt_args>)
```

Parameters

x,y

row vectors of sizes n_1 and n_2 (x-axis and y-axis coordinates). These coordinates must be monotone.

z

matrix of size (n_1,n_2) . $z(i,j)$ is the value of the surface at the point $(x(i),y(j))$.

xf,yf,zf

matrices of size (n_f,n) . They define the facets used to draw the surface. There are n facets. Each facet i is defined by a polygon with n_f points. The x-axis, y-axis and z-axis coordinates of the points of the i th facet are given respectively by $xf(:,i)$, $yf(:,i)$ and $zf(:,i)$.

colors

a vector of size n giving the color of each facets or a matrix of size (n_f,n) giving color near each facet boundary (facet color is interpolated).

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `theta, alpha, leg, flag, ebox` (see definition below).

theta, alpha

real values giving in degree the spherical coordinates of the observation point.

leg

string defining the labels for each axis with `@` as a field separator, for example `"X@Y@Z"`.

flag

a real vector of size three. `flag=[mode, type, box]`.

mode

an integer (surface color).

mode>0

the surface is painted with color `"mode"`; the boundary of the facet is drawn with current line style and color.

mode=0:

a mesh of the surface is drawn.

mode<0:

the surface is painted with color `"-mode"`; the boundary of the facet is not drawn.

Note that the surface color treatment can be done using `color_mode` and `color_flag` options through the surface entity properties (see `surface_properties`).

type

an integer (scaling).

type=0:

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:

3d isometric with box bounds given by optional `ebox`, similarly to `type=1`.

type=4:

3d isometric bounds derived from the data, to similarly `type=2`.

type=5:

3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`.

type=6:

3d expanded isometric bounds derived from the data, similarly to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see `axes_properties`).

box

an integer (frame around the plot).

box=0:

nothing is drawn around the plot.

box=1:

unimplemented (like `box=0`).

box=2:

only the axes behind the surface are drawn.

box=3:

a box surrounding the surface is drawn and captions are added.

box=4:

a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see `axes_properties`).

ebox

It specifies the boundaries of the plot as the vector `[xmin, xmax, ymin, ymax, zmin, zmax]`. This argument is used together with `type` in `flag` : if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into account.

Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see `axes_properties`).

Description

`plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` draws the parametric surface $z=f(x,y)$.

`plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])` draws a surface defined by a set of facets. You can draw multiple plots by replacing `xf`, `yf` and `zf` by multiple matrices assembled by rows as `[xf1 xf2 ...]`, `[yf1 yf2 ...]` and `[zf1 zf2 ...]`. Note that data can also be set or get through the surface entity properties (see `surface_properties`).

You can give a specific color for each facet by using `list(zf,colors)` instead of `zf`, where `colors` is a vector of size `n`. If `colors(i)` is positive it gives the color of facet `i` and the boundary of the facet is drawn with current line style and color. If `colors(i)` is negative, color id `-colors(i)` is used and the boundary of the facet is not drawn.

It is also possible to get interpolated color for facets. For that the color argument must be a matrix of size `nfxn` giving the color near each boundary of each facets. In this case positive values for colors mean that the boundary are not drawn. Note that colors can also be set through the surface entity properties (via `tlist` affectations) and edited using `color_flag` option (see `surface_properties`).

The optional arguments `theta`, `alpha`, `leg`, `flag`, `ebox`, can be passed by a sequence of statements `key1=value1`, `key2=value2`, ... In this case, the order has no special meaning. Note that all these optional arguments except `flag` can be customized through the axes entity properties (see `axes_properties`). As described before, the `flag` option deals with surface entity properties for `mode` (see `surface_properties`) and axes properties for `type` and `box` (see `axes_properties`).

You can use the function `genfac3d` to compute four sided facets from the surface $z=f(x,y)$. `eval3dp` can also be used.

Enter the command `plot3d()` to see a demo.

Examples

```
// simple plot using z=f(x,y)
t=[0:0.3:2*pi]';
z=sin(t)*cos(t');
plot3d(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf()
plot3d(xx,yy,zz)
// multiple plots
clf()
plot3d([xx xx],[yy yy],[zz 4+zz])
// multiple plots using colors
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
// simple plot with viewpoint and captions
clf()
plot3d(1:10,1:20,10*rand(10,20),alpha=35,theta=45,flag=[2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)","x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
      "y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
```



```

    "z=r*sin(alp)+orig(3)*ones(tet)"]);
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
clf();plot3d(xx,yy,zz)
clf();
f=gcf();
f.color_map = hotcolormap(128);
r=0.3;orig=[1.5 0 0];
[xx1,yy1,zz1]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
cc=(xx+zz+2)*32;cc1=(xx1-orig(1)+zz1/r+2)*32;
clf();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),theta=70,alpha=80,flag=

delete(gcf());
t=[0:0.3:2*pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
e=gce();
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,6*rand(f.z)); // random color
e.data = TL;
TL2 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,4*rand(1,800)); // random co
e.data = TL2;
TL3 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,[20*ones(1,400) 6*ones(1,400)
e.data = TL3;
TL4 = tlist(["3d" "x" "y" "z"],f.x,f.y,f.z); // no color
e.data = TL4;
e.color_flag=1 // color index proportional to altitude (z coord.)
e.color_flag=2; // back to default mode
e.color_flag= 3; // interpolated shading mode (based on blue default color)
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=gce(); //get handle on current entity (here the surface)
a=gca(); //get current axes
a.rotation_angles=[40,70];
a.grid=[1 1 1]; //make grids
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off"; //axes are hidden
a.axes_bounds=[.2 0 1 1];
h.color_flag=1; //color according to z
h.color_mode=-2; //remove the facets boundary by setting color_mode to white c
h.color_flag=2; //color according to given colors
h.color_mode = -1; // put the facets boundary back by setting color_mode to bla
f=gcf();//get the handle of the parent figure
f.color_map=hotcolormap(512);
c=[1:400,1:400];
TL.color = [c;c+1;c+2;c+3];
h.data = TL;
h.color_flag=3; // interpolated shading mode

```

We can use the plot3d function to plot a set of patches (triangular, quadrangular, etc).

```

// The plot3d function to draw patches:
// patch(x,y,[z])
// patch(x,y,[list(z,c)])
// The size of x : number of points in the patches x number of patches
// y and z have the same sizes as x

```

```
// c:
// - a vector of size number of patches: the color of the patches
// - a matrix of size number of points in the patches x number of
//   patches: the color of each points of each patches

// Example 1: a set of triangular patches

x = [0 0;
     0 1;
     1 1];

y = [1 1;
     2 2;
     2 1];

z = [1 1;
     1 1;
     1 1];

tcolor = [2 3]';

subplot(2,2,1);
plot3d(x,y,list(z,tcolor));
xtitle('A triangle set of patches');

// Example 2: a mixture of triangular and quadrangular patches

xquad = [5, 0;
         10,0;
         15,5;
         10,5];

yquad = [15,0;
         20,10;
         15,15;
         10,5];

zquad = ones(4,2);

xtri = [ 0,10,10, 5, 0;
        10,20,20, 5, 0;
        20,20,15,10,10];

ytri = [ 0,10,20, 5,10;
        10,20,20,15,20;
        0, 0,15,10,20];

ztri = zeros(3,5);

subplot(2,2,3);
plot3d(xquad,yquad,zquad);
plot3d(xtri,ytri,ztri);
xtitle('Mixing triangle and quadrangle set of patches');

// Example 3: some rabbits

rabxtri = [ 5, 5, 2.5, 7.5, 10;
           5, 15, 5, 10, 10];
```

```

        15, 15, 5, 10, 15];

rabytri = [10, 10, 9.5, 2.5, 0;
           20, 10, 12, 5, 5;
           10 0 7 0 0];
rabztri = [0,0,0,0,0;
           0,0,0,0,0;
           0,0,0,0,0];

rabtricolor_byface = [2 2 2 2 2];

rabtricolor = [2,2,2,2,2;
               3,3,3,3,3;
               4,4,4,4,4];

rabxquad = [0, 1;
            0, 6;
            5,11;
            5, 6];

rabyquad = [18,23;
            23,28;
            23,28;
            18,23];

rabzquad = [1,1;
            1,1;
            1,1;
            1,1];

rabquadcolor_byface = [2 2];

rabquadcolor = [2,2;
                3,3;
                4,4;
                5,5];

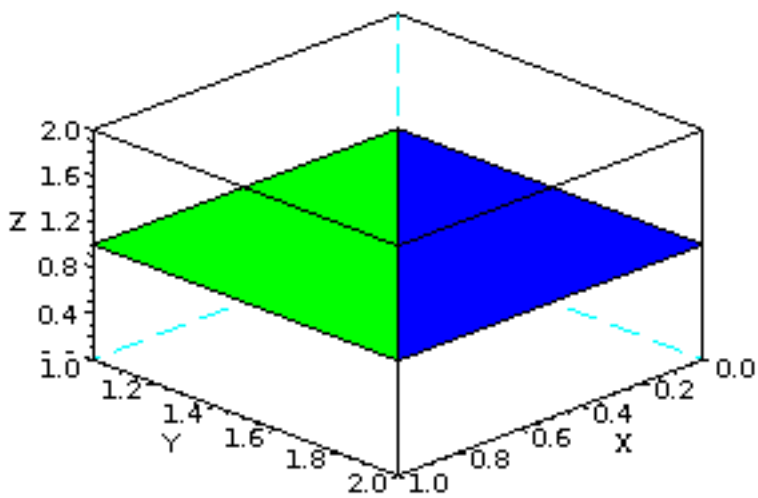
subplot(2,2,2);
plot3d(rabxtri, rabytri, list(rabztri,rabtricolor));
plot3d(rabxquad,rabyquad,list(rabzquad,rabquadcolor));
h = gcf();
h.children(1).background = 1;
xtitle('A psychedelic rabbit set of patches');

subplot(2,2,4);
plot3d(rabxtri, rabytri, list(rabztri,rabtricolor_byface));
plot3d(rabxquad,rabyquad,list(rabzquad,rabquadcolor_byface));
h = gcf();
h.children(1).background = 1;
xtitle('A standard rabbit set of patches');

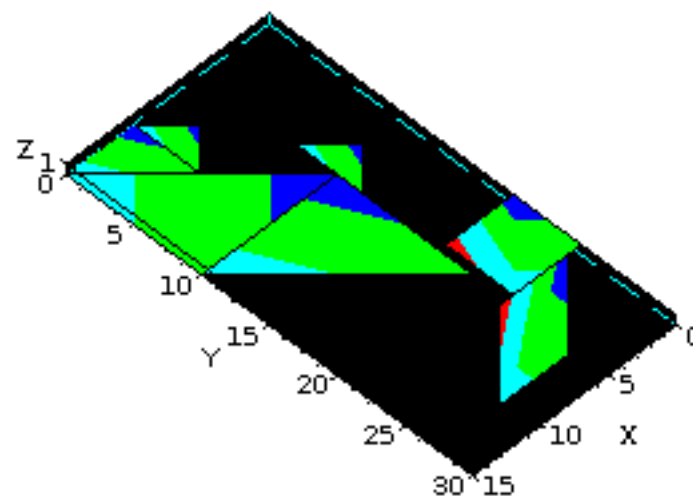
```

The result of the preceding example:

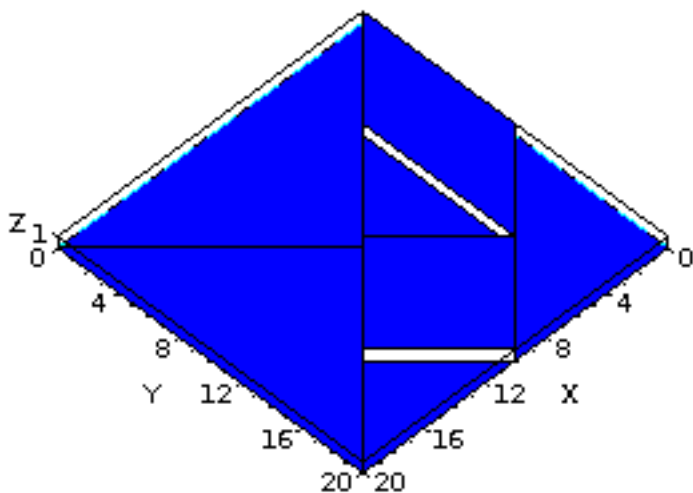
A triangle set of patches



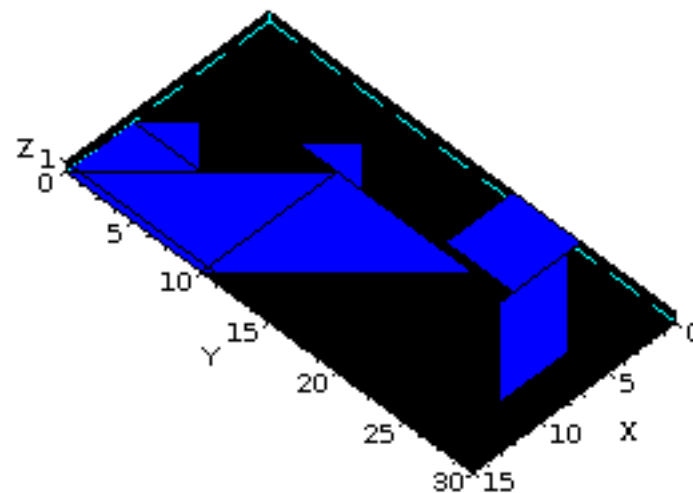
A psychedelic rabbit set of patches



Mixing triangle and quadrangle set of patches



A standard rabbit set of patches



We can also use the plot3d function to plot a set of patches using vertex and faces.

```
// Vertex / Faces example: 3D example

// The vertex list contains the list of unique points composing each patch
// The points common to 2 patches are not repeated in the vertex list

vertex = [0 1 1;
          0 2 2;
          1 2 3;
          1 1 4];

// The face list indicates which points are composing the patch.
face = [1 2 3;
        1 3 4];

tcolor = [2 3]';
```

```
// The formula used to translate the vertex / face representation into x, y, z
xvf = matrix(vertex(face,1),size(face,1),length(vertex(face,1))/size(face,1))';
yvf = matrix(vertex(face,2),size(face,1),length(vertex(face,1))/size(face,1))';
zvf = matrix(vertex(face,3),size(face,1),length(vertex(face,1))/size(face,1))';

scf();
subplot(2,1,1);
plot3d(xvf,yvf,list(zvf,tcolor));
xtitle('A triangle set of patches - vertex / face mode - 3d');

// 2D test
// We use the 3D representation with a 0 Z values and then switch to 2D representation

// Vertex / Faces example: 3D example

// The vertex list contains the list of unique points composing each patch
// The points common to 2 patches are not repeated in the vertex list

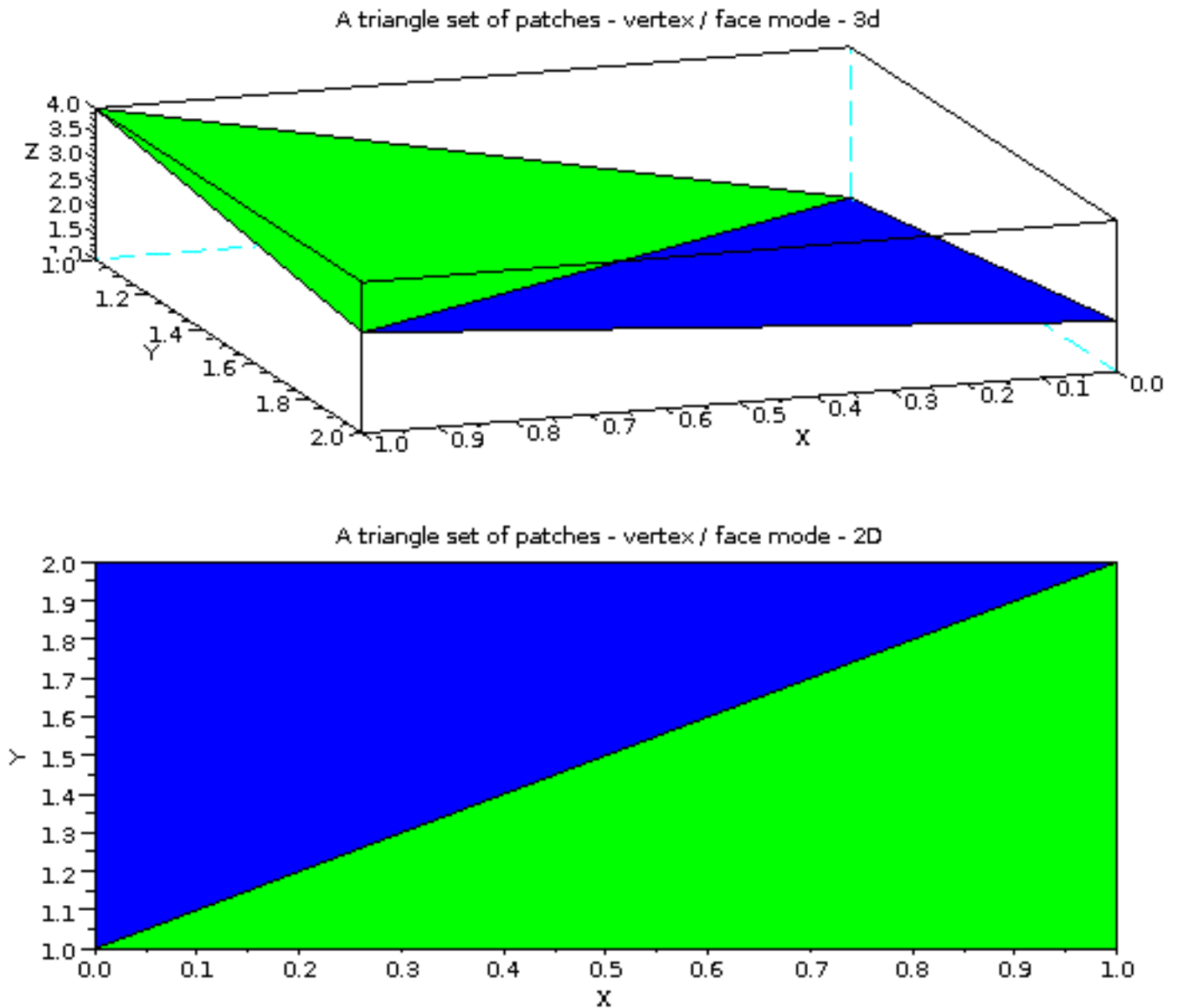
vertex = [0 1;
          0 2;
          1 2;
          1 1];

// The face list indicates which points are composing the patch.
face = [1 2 3;
        1 3 4];

// The formula used to translate the vertex / face representation into x, y, z
xvf = matrix(vertex(face,1),size(face,1),length(vertex(face,1))/size(face,1))';
yvf = matrix(vertex(face,2),size(face,1),length(vertex(face,1))/size(face,1))';
zvf = matrix(zeros(vertex(face,2)),size(face,1),length(vertex(face,1))/size(face,1))';

subplot(2,1,2);
plot3d(xvf,yvf,list(zvf,tcolor));
xtitle('A triangle set of patches - vertex / face mode - 2D');
a = gca();
a.view = '2d';
```

The result of the preceding example:



How to set manually some ticks

```
plot3d();
h = gca();
h.x_ticks = tlist(['ticks','locations','labels'],[-2,-1,0,1,2],['-2','-1','0','1','2']);
h.y_ticks = tlist(['ticks','locations','labels'],[-4,-3-2,-1,0,1,2,3,4],['-4','-3','-2','-1','0','1','2','3','4']);
h.z_ticks = tlist(['ticks','locations','labels'],[-1,0,1],['Point 1','Point 2','Point 3']);
```

See Also

eval3dp, genfac3d, geom3d, param3d, plot3d1, clf, gca, gcf, xdel, delete, axes_properties

Authors

J.Ph.C.

Name

plot3d1 — 3D gray or color level plot of a surface

```
plot3d1(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d1(xf,yf,zf,[theta,alpha,leg,flag,ebox])

plot3d1(x,y,z,<opts_args>)
plot3d1(xf,yf,zf,<opts_args>)
```

Parameters

x,y

row vectors of sizes $n1$ and $n2$ (x-axis and y-axis coordinates). These coordinates must be monotone.

z

matrix of size $(n1,n2)$. $z(i,j)$ is the value of the surface at the point $(x(i),y(j))$.

xf,yf,zf

matrices of size (nf,n) . They define the facets used to draw the surface. There are n facets. Each facet i is defined by a polygon with nf points. The x-axis, y-axis and z-axis coordinates of the points of the i th facet are given respectively by $xf(:,i)$, $yf(:,i)$ and $zf(:,i)$.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `theta, alpha, leg, flag, ebox` (see definition below).

theta, alpha

real values giving in degree the spherical coordinates of the observation point.

leg

string defining the labels for each axis with `@` as a field separator, for example `"X@Y@Z"`.

flag

a real vector of size three. `flag=[mode, type, box]`.

mode

an integer (surface color).

mode>0

the surface is painted with color `"mode"` ; the boundary of the facet is drawn with current line style and color.

mode=0:

a mesh of the surface is drawn.

mode<0:

the surface is painted with color `"-mode"` ; the boundary of the facet is not drawn.

Note that the surface color treatment can be done using `color_mode` and `color_flag` options through the surface entity properties (see `surface_properties`).

type

an integer (scaling).

type=0:

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument **ebox**.

type=2:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:
3d isometric with box bounds given by optional **ebox**, similarly to **type=1**.

type=4:
3d isometric bounds derived from the data, to similarly **type=2**.

type=5:
3d expanded isometric bounds with box bounds given by optional **ebox**, similarly to **type=1**.

type=6:
3d expanded isometric bounds derived from the data, similarly to **type=2**.

Note that axes boundaries can be customized through the axes entity properties (see **axes_properties**).

box
an integer (frame around the plot).

box=0:
nothing is drawn around the plot.

box=1:
unimplemented (like **box=0**).

box=2:
only the axes behind the surface are drawn.

box=3:
a box surrounding the surface is drawn and captions are added.

box=4:
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see **axes_properties**).

ebox
It specifies the boundaries of the plot as the vector **[xmin, xmax, ymin, ymax, zmin, zmax]**. This argument is used together with **type** in **flag** : if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If **flag** is missing, **ebox** is not taken into account.

Note that, when specified, the **ebox** argument acts on the **data_bounds** field that can also be reset through the axes entity properties (see **axes_properties**).

Description

plot3d1 plots a surface with colors depending on the z-level of the surface. This special plot function can also be enabled setting **color_flag=1** after a **plot3d** (see **surface_properties**)

Enter the command **plot3d1()** to see a demo.

Examples

```
// simple plot using z=f(x,y)
t=[0:0.3:2*pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf();
plot3d1(xx,yy,zz)
// multiple plots
clf();
plot3d1([xx xx],[yy yy],[zz 4+zz])
// simple plot with viewpoint and captions
clf();
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// same plot without grid
clf();
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[-2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)","x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
"y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
"z=r*sin(alp)+orig(3)*ones(tet)");
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-pi/2,pi/2,40),linspace(0,pi*2,20));
clf()

plot3d(xx,yy,zz)
e=gce();
e.color_flag=1;
scf(2);
plot3d1(xx,yy,zz) // the 2 graphics are similar
```

See Also

plot3d , gca , gce , scf , clf

Authors

J.Ph.C.

Name

plot3d2 — plot surface defined by rectangular facets

```
plot3d2(X,Y,Z [, vect, theta, alpha, leg, flag, ebox])  
plot3d2(X,Y,Z, <opt_args>)
```

Parameters

X, Y, Z:

3 real matrices defining a data structure.

vect

a real vector.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `theta, alpha, leg, flag, ebox` (see definition below).

theta, alpha

real values giving in degree the spherical coordinates of the observation point.

leg

string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag

a real vector of size three. `flag=[mode, type, box]`.

mode

an integer (surface color).

mode>0

the surface is painted with color "mode" ; the boundary of the facet is drawn with current line style and color.

mode=0:

a mesh of the surface is drawn.

mode<0:

the surface is painted with color "-mode" ; the boundary of the facet is not drawn.

Note that the surface color treatment can be done using `color_mode` and `color_flag` options through the surface entity properties (see `surface_properties`).

type

an integer (scaling).

type=0:

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:

3d isometric with box bounds given by optional `ebox`, similarly to `type=1`.

`type=4:`

3d isometric bounds derived from the data, to similarly `type=2`.

`type=5:`

3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`.

`type=6:`

3d expanded isometric bounds derived from the data, similarly to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see `axes_properties`).

`box`

an integer (frame around the plot).

`box=0:`

nothing is drawn around the plot.

`box=1:`

unimplemented (like `box=0`).

`box=2:`

only the axes behind the surface are drawn.

`box=3:`

a box surrounding the surface is drawn and captions are added.

`box=4:`

a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see `axes_properties`).

`ebox`

It specifies the boundaries of the plot as the vector `[xmin, xmax, ymin, ymax, zmin, zmax]`. This argument is used together with `type` in `flag` : if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into account.

Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see `axes_properties`).

Description

`plot3d2` plots a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a surface. The surface is composed of four sided polygons.

The X-coordinates of a facet are given by `X(i,j)`, `X(i+1,j)`, `X(i+1,j+1)` and `X(i,j+1)`. Similarly Y and Z matrices contain Y and Z-coordinates.

The `vect` vector is used when multiple surfaces are coded in the same (X,Y,Z) matrices. `vect(j)` gives the line at which the coding of the jth surface begins. Like in `plot3d`, the same properties are editable (see `surface_properties` and `axes_properties`).

Examples

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
```

```
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);
e=gce();
e.color_mode=4; // change color
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,10*(f.z)+1);
e.data=TL;
e.color_flag=2;
```

See Also

[plot3d](#) , [genfac3d](#)

Name

plot3d3 — mesh plot surface defined by rectangular facets

```
plot3d3(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])  
plot3d3(X,Y,Z, <opt_args>)
```

Parameters

X, Y, Z:

3 real matrices defining a data structure.

vect

a real vector.

<opt_args>

This represents a sequence of statements `key1=value1, key2=value2,...` where `key1, key2, . . .` can be one of the following: `theta, alpha, leg, flag, ebox` (see definition below).

theta, alpha

real values giving in degree the spherical coordinates of the observation point.

leg

string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag

a real vector of size four. `flag=[vertical_color, horizontal_color, type, box]`.

vertical_color

an integer (surface color), default is 3.

Colormap index defining the color used to draw vertical edges.

horizontal_color

an integer (surface color), default is 4.

Colormap index defining the color used to draw horizontal edges.

type

an integer (scaling) default is 2.

type=0:

the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2:

rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:

3d isometric with box bounds given by optional `ebox`, similarly to `type=1`.

type=4:

3d isometric bounds derived from the data, to similarly `type=2`.

`type=5:`
3d expanded isometric bounds with box bounds given by optional `ebox`, similarly to `type=1`.

`type=6:`
3d expanded isometric bounds derived from the data, similarly to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see `axes_properties`).

`box`
an integer (frame around the plot), default is 4.

`box=0:`
nothing is drawn around the plot.

`box=1:`
unimplemented (like `box=0`).

`box=2:`
only the axes behind the surface are drawn.

`box=3:`
a box surrounding the surface is drawn and captions are added.

`box=4:`
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see `axes_properties`).

`ebox`
It specifies the boundaries of the plot as the vector `[xmin, xmax, ymin, ymax, zmin, zmax]`. This argument is used together with `type` in `flag`: if it is set to 1, 3 or 5 (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into account. Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see `axes_properties`).

Description

`plot3d3` performs a mesh plot of a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a surface. The surface is composed of four sided polygons.

The X-coordinates of a facet are given by `X(i,j)`, `X(i+1,j)`, `X(i+1,j+1)` and `X(i,j+1)`. Similarly Y and Z matrices contain Y and Z-coordinates.

The `vect` vector is used when multiple surfaces are coded in the same (X,Y,Z) matrices. `vect(j)` gives the line at which the coding of the `j`th surface begins. See `plot3d2` for a full description. As a mesh plot, the only available property you can edit is the `visible` option (see `axes_properties`).

Examples

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d3(X,Y,Z);
```

```
e=gce(); // get the current entity
e.visible='off';
e.visible='on'; // back to the mesh view
```

See Also

plot3d2 , plot3d , param3d

Name

plotframe — plot a frame with scaling and grids. **This function is obsolete.**

```
plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])
plotframe(rect,<opts_args>)
```

Parameters

rect

vector [xmin,ymin,xmax,ymax].

tics

vector [nx,mx,ny,my] where mx, nx (resp. my, ny) are the number of x-axis (resp. y-axis) intervals and subintervals.

arg_optX

optional arguments up to three and choosen among.

flags

vector [wantgrids,findbounds] where wantgrids is a boolean variable (%t or %f) which indicates gridding. findbounds is a boolean variable. If findbounds is %t, the bounds given in rect are allowed to be slightly modified (in fact always increased) in order to have simpler graduations: then tics(2) and tics(4) are ignored.

Captions

vector of 3 strings [title,x-leg,y-leg] corresponding respectively to the title of the plot and the captions on the x-axis and the y-axis. Warning: the upper-case "C" is important.

subwin

a vector of size 4 defining the sub window. The sub window is specified with the parameter subwin=[x,y,w,h] (upper-left, width, height). The values in subwin are specified using proportion of the width or height of the current graphics window (see xsetech).

<opts_args>

This represents a sequence of statements **key1=value1, key2=value2,...** where **key1, key2,...** can be one of the following: **tics, flags, captions** ou **subwin**. These arguments have the same meaning as the ones used in the first form of the routine.

Description

plotframe is used with 2D plotting functions plot2d, plot2d1,... to set a graphics frame. It must be used before plot2d which should be invoked with the "000" superposition mode.

This function is obsolete.

Examples

```
x=[-0.3:0.8:27.3]';
y=rand(x);
rect=[min(x),min(y),max(x),max(y)];
tics=[4,10,2,5]; //4 x-intervals and 2 y-intervals
plotframe(rect,tics,[%f,%f],["My plot","x","y"],[0,0,0.5,0.5])
plot2d(x,y,2,"000")
plotframe(rect,tics=tics,flags=[%t,%f],Captions=["My plot with grids","x","y"],
plot2d(x,y,3,"000")
plotframe(rect,tics,[%t,%t],..
```



```
["My plot with grids and automatic bounds","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,4,"000")
plotframe(rect,flags=[%f,%t],tics=tics,..
          Captions=["My plot without grids but with automatic bounds ","x","y"]
          subwin=[0.5,0.5,0.5,0.5])
plot2d(x,y,5,"000")
```

See Also

[plot2d](#), [graduate](#), [xsetech](#)

Name

plzr — pole-zero plot

```
plzr(sl)
```

Parameters

sl
list (syslin)

Description

produces a pole-zero plot of the linear system sl (syslin list)

Examples

```
s=poly(0, 's');  
n=[1+s    2+3*s+4*s^2    5; 0    1-s    s];  
d=[1+3*s    5-s^3    s+1;1+s    1+s+s^2    3*s-1];  
h=syslin('c',n./d);  
plzr(h);
```

See Also

trzeros , roots , syslin

Name

polarplot — Plot polar coordinates

```
polarplot(theta, rho, [style, strf, leg, rect])  
polarplot(theta, rho, <opt_args>)
```

Parameters

rho

a vector, the radius values

theta

a vector with same size than rho, the angle values.

<opt_args>

a sequence of statements `key1=value1, key2=value2, ...` where keys may be `style, leg, rect, strf` or `frameflag`

style

is a real row vector of size `nc`. The style to use for curve `i` is defined by `style(i)`. The default style is `1:nc` (1 for the first curve, 2 for the second, etc.).

- if `style(i)` is negative, the curve is plotted using the mark with id `abs(style(i))+1`; use `xset()` to see the mark ids.
- if `style(i)` is strictly positive, a plain line with color id `style(i)` or a dashed line with dash id `style(i)` is used; use `xset()` to see the color ids.
- When only one curve is drawn, `style` can be the row vector of size 2 [`sty, pos`] where `sty` is used to specify the style and `pos` is an integer ranging from 1 to 6 which specifies a position to use for the caption. This can be useful when a user wants to draw multiple curves on a plot by calling the function `plot2d` several times and wants to give a caption for each curve.

strf

is a string of length 3 "xy0".

default

The default is "030".

x

controls the display of captions,

x=0

no captions.

x=1

captions are displayed. They are given by the optional argument `leg`.

y

controls the computation of the frame. same as `frameflag`

y=0

the current boundaries (set by a previous call to another high level plotting function) are used. Useful when superposing multiple plots.

y=1

the optional argument `rect` is used to specify the boundaries of the plot.

y=2

the boundaries of the plot are computed using min and max values of `x` and `y`.

y=3

like `y=1` but produces isoview scaling.

y=4

like `y=2` but produces isoview scaling.

y=5

like `y=1` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=6

like `y=2` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=7

like `y=5` but the scale of the new plot is merged with the current scale.

y=8

like `y=6` but the scale of the new plot is merged with the current scale.

`leg`

a string. It is used when the first character `x` of argument `strf` is 1. `leg` has the form "`leg1@leg2@...`" where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is "".

`rect`

This argument is used when the second character `y` of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

Description

`polarplot` creates a polar coordinate plot of the angle `theta` versus the radius `rho`. `theta` is the angle from the `x`-axis to the radius vector specified in radians; `rho` is the length of the radius vector specified in dataspace units.

Examples

```
t= 0:.01:2*pi;
clf();polarplot(sin(7*t),cos(8*t))

clf();polarplot([sin(7*t') sin(6*t')],[cos(8*t') cos(8*t')],[1,2])
```

Name

polyline_properties — description of the Polyline entity properties

Description

The Polyline entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for polylines.

parent:

This field contains the handle of the parent. The parent of the polyline entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the `children` of the handle. However, polyline handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, the polyline is visible, the value's property is "on". If "off" the polyline is not drawn on the screen.

data:

This field contains the values for the x and y coordinates. Component Z is to be added in the case of three-dimensional axes. It is a two (three) column matrix $[x, y, [z]]$ of points.

closed:

This field determines whether the polyline is closed or not: its value can be "on" or "off" (no default value, it depends on the primitive used to create the polyline).

line_mode:

This field contains the default `line_mode` property value for the polyline. Its value should be "on" (line drawn) or "off" (no line drawn).

fill_mode:

If the `polyline_style` field is different of 5, fill the background of the curve with color defined by the `background` property.

line_style:

The `line_style` property value should be an integer in $[0, 8]$. 0 and 1 stands for solid, the other value stands for a selection of dashes (see `getlinestyle`).

thickness:

This property is a positive real specifying the line width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

arrow_size_factor:

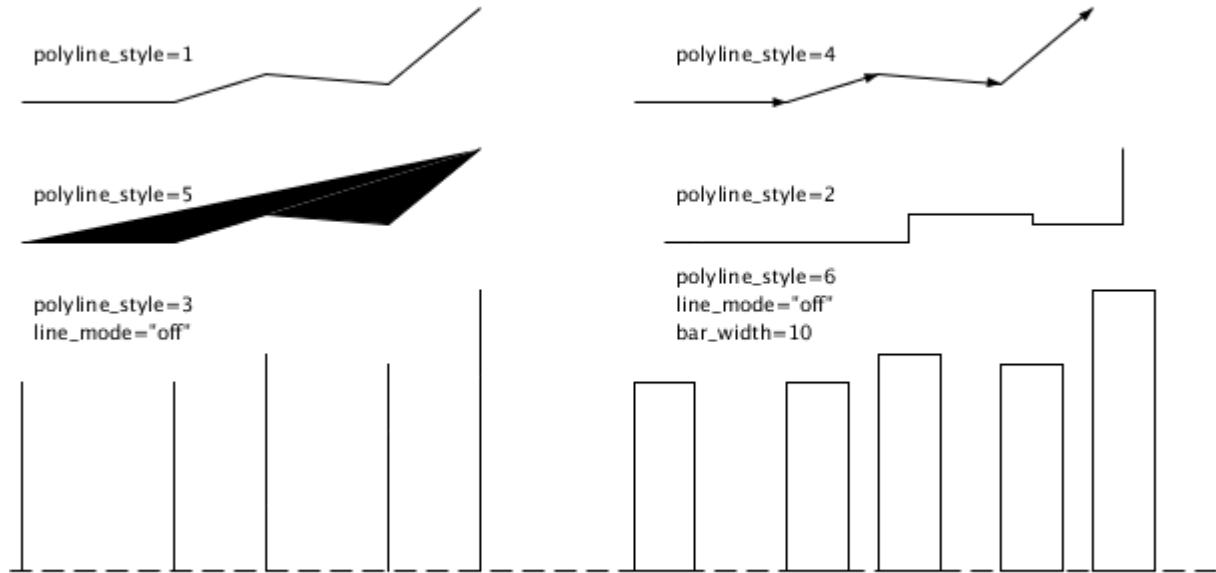
This integer allows to set the size of arrows drawn on the polyline. The actual size of arrows is the product of the `thickness` and the size factor.

polyline_style:

This property sets several polyline drawing mode:

- If the value is 0 or 1 lines are drawn between two consecutive points.
- If the value is 2 the polyline produces a staircase plot. Two consecutive points are linked by an horizontal line followed by a vertical line.
- If the value is 3 the polyline produces a bar plot. For each given point (x,y) a vertical line is drawn from (x,y) to (x,0).

- If the value is 4 arrows are drawn between two consecutives points.
- If the value is 5 the polyline is filled (patch).
- If the value is 6 the polyline is a Matlab-like bar object. The properties `bar_shift` and `bar_width` command its appearance.

**foreground:**

This field contains the default **foreground** property used to draw the polyline. Its value should be a color index (relative to the current colormap).

background:

This field contains the color used to fill the background of the polyline. Its value should be a color index (relative to the current colormap).

interp_color_vector:

This field contains the vector of color indices used to fill in the polyline when the `interp_color_mode` property is set to "on". It defines the intervals of colormap indices used to fill each segment. For instance, the first segment will be filled by every colors whose index is between the first two elements of the vecor. It is only applicable if the polyline is defined by 3 or 4 points. Therefore, the size of the vector must match this dimension.

interp_color_mode:

This field determines if we are using the interpolated shading mode to fill the polyline : its value can be "on" or "off". Note that an `interp_color_vector` must be defined before switching to "on" value (see above).

mark_mode:

This field contains the default `mark_mode` property value for the polyline. Its value should be "on" (marks drawn) or "off" (no marks drawn).

mark_style:

The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is "on". The value should be an integer in [0 14] which stands for: dot, plus, cross, star,

filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



`mark_size_unit:`

This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to "point", then the `mark_size` value is directly given in points. When `mark_size_unit` is set to "tabulated", `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that `plot2d` and pure `scilab` functions use `tabulated` mode as default ; when using `plot` function, the `point` mode is automatically enabled.

`mark_size:`

The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is "on". Its value should be an integer between 0 and 5 which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

`mark_foreground:`

This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent edge.

`mark_background:`

This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent face.

`x_shift:`

This field contains the offset computed by a call to the `bar` function (or re-computed by a call to `barhomogenize`) and is used to perform a nice vertical bar representation. Note that this offset is also taken into account for all the other `polyline_style`. The unit is expressed in user coordinates.

`y_shift:`

This field contains the offset computed by a call to the `bar` function (or re-computed by a call to `barhomogenize`) and is used to perform a nice horizontal bar representation. Note that this offset is also taken into account for all the other `polyline_style`. The unit is expressed in user coordinates.

`z_shift:`

This field contains the offset the user may specify. Note that this offset is taken into account for all the `polyline_style`. The unit is expressed in user coordinates.

`bar_width:`

This field determines the width of the selected polyline when its `polyline_style` is set to `bar` mode (case 6) : the unit is expressed in user coordinates.

`clip_state:`

This field contains the `clip_state` property value for the polyline. It should be :

- "off" this means that the polyline is not clipped.
- "clipgrf" this means that the polyline is clipped outside the Axes box.
- "on" this means that the polyline is clipped outside the rectangle given by property `clip_box`.

clip_box:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the polyline to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the polyline data structure, and to retrieve it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xpoly(sin(2*pi*(0:5)/5),cos(2*pi*(0:5)/5),"lines",0)
p=get("hdl"); //get handle on current entity (here the polyline entity)
p.foreground=2;
p.thickness=3;
p.mark_style=9;
d=p.data;d(1,:)=[0 0];p.data=d;
a.rotation_angles=[0 45];

p.data=[(-2:0.1:2)' sin((-2:0.1:2)*pi)']
p.mark_mode="off";
p.polyline_style=3;
p.line_style=4;
```

See Also

`set` , `get` , `delete` , `xpoly` , `xfpoly` , `xpolys` , `xfpolys` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

rainbowcolormap — red through orange, yellow, green, blue to violet colormap

```
cmap=rainbowcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

rainbowcolormap computes a colormap with *n* colors varying from red through orange, yellow, green, blue to violet.

Examples

```
f = scf();  
plot3d1();  
f.color_map = rainbowcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

rectangle_properties — description of the Rectangle entity properties

Description

The Rectangle entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for rectangles and filled rectangles.

parent:

This field contains the handle of the parent. The parent of the rectangle entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the children of the handle. However, rectangle handles currently do not have any children.

mark_mode:

This field contains the default mark_mode property value for the polyline. Its value should be "on" (marks drawn) or "off" (no marks drawn).

mark_style:

The mark_style property value is used to select the type of mark to use when mark_mode property is "on". The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the mark_style and the mark_foreground and mark_background properties.



mark_size_unit:

This field contains the default mark_size_unit property value. If mark_size_unit is set to "point", then the mark_size value is directly given in points. When mark_size_unit is set to "tabulated", mark_size is computed relative to the font size array: therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that xrect and pure scilab functions use tabulated mode as default ; when using plot function, the point mode is automatically enabled.

mark_size:

The mark_size property is used to select the type of size of the marks when mark_mode property is "on". Its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:

This field contains the mark_foreground property value which is the marks' edge color. Its value should be a color index (relative to the current color_map) or 0 for transparent edge.

mark_background:

This field contains the mark_background property value which is the marks' face color. Its value should be a color index (relative to the current color_map) or 0 for transparent face.

line_mode:

This field contains the default line_mode property value for the rectangle. Its value should be "on" (line drawn) or "off" (no line drawn).

fill_mode:

If `fill_mode` property value is "on" , the rectangle is filled with the foreground color, the `mark_mode` must also have the value "off". if not and the value's property is "off" only the shape of the rectangle is drawn using the foreground color.

line_style:

The `line_style` property value should be an integer in [0 8]. 0 and 1 stands for solid, the other value stands for a selection of dashes (see `getlinestyle`).

thickness:

This property is a positive real specifying the rectangle width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

foreground:

This field contains the color used to draw the outline of the rectangle. Its value should be a color index (relative to the current colormap).

background:

This field contains the color used to fill the rectangle. Its value should be a color index (relative to the current colormap).

data:

This property is to return the coordinates of the upper-left point of the rectangle and its width and height in user coordinates. The result is the matrix `[xleft,yup,[zup],width,height]`

visible:

This field contains the `visible` property value for the entity . It should be "on" or "off" . By default, the rectangle is visible, the value's property is "on". If "off" the rectangle is not drawn on the screen.

clip_state:

This field contains the `clip_state` property value for the rectangle. It should be :

- "off" this means that the rectangle is not clipped.
- "clipgrf" this means that the rectangle is clipped outside the Axes box.
- "on" this means that the rectangle is clipped outside the rectangle given by property `clip_box`.

clip_box:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the rectangle to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the rectangle data structure, and to retrieve it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xrect(-1,1,2,2)

r=get("hdl");//get handle on current entity (here the rectangle entity)
r.type
r.parent.type
```

```
r.foreground=13;  
r.line_style=2;  
r.fill_mode="on";  
r.background=color('red');  
r.clip_box=[-1 1;1 1];  
r.data(:,[3 4])=[1/2 1/2];  
r.data(:,[1 2])=[1/2 1/2];  
r.clip_state="off"
```

See Also

[set](#) , [get](#) , [delete](#) , [xrect](#) , [xfrect](#) , [xrects](#) , [graphics_entities](#)

Authors

Djalel ABDEMOUCHE

Name

relocate_handle — Move handles inside the graphic hierarchy.

```
relocate_handle( movedHandles, parent )
```

Parameters

movedHandles

Vector of relocated handles.

parent

New parent of the handles.

Description

The `relocate_handle` function allows to move handles from their current locations in the graphical hierarchy to another. All the moved entities are relocated under the same parent handle specified with the **parent** parameter.

Since not every handles are compatible with each others, some restrictions exist when relocating handles. For examples, it is not allowed to move an axes handle under a polyline. More information about compatibility can be found in the `graphics_entities` page.

This routine is particularly useful to move an object from an axes entity to an other or to move axes from figures handles.

Examples

```
x = 0:10 ;

// plot a first polyline
plot(x,x^2) ;
axes1 = gca() ;
poly1 = gce() ;

// plot a second in an other window
scf() ;
plot( x,x ) ;
axes2 = gca() ;
poly2 = gce() ;
poly2bis = copy( poly2 ) ; // make a copy of the polyline

// put both polyline in the same window
relocate_handle( poly2bis, axes1 ) ;
```

See Also

`graphics_entities` , `copy` , `delete` , `swap_handles`

Authors

Jean-Baptiste Silvy

Name

replot — redraw the current graphics window with new boundaries

```
replot(rect,[handle])
```

Parameters

rect

row vector of size 4.

handle

optional argument. Graphics handle(s) of type Axes to select one or several given Axes.

Description

replot is used to redraw the content of the current graphics window with new boundaries defined by `rect=[xmin,ymin,xmax,ymax]`. Under old graphics syntax, it works only with the driver "Rec".

This transformation can be applied to specific axes given by Axes graphics handles via the handle argument. If handle is not specified, the new boundaries are applied to the current axes of the current figure. The transformation changes the `data_bounds` value of those axes. Note that the axes property `tight_limits` must also be set to "on" to strictly select those bounds (see `axes_properties`).

Examples

```
backupstyle='?'

// First Example
x=[0:0.1:2*pi]';
plot2d(x,sin(x))
replot([-1,-1,10,2])

// Second Example
xdel(winsid());
plot() // plot demo
f=gcf();
replot([-1,-1,10,2],f.children(1)) // specify axes handle's value
replot([-3,-2,8,4],f.children(2))
```

See Also

`xbasr`, `clf`

Authors

J.Ph.C.

Name

rgb2name — returns the name of a color

```
names=rgb2name(r,g,b)
names=rgb2name(rgb)
```

Parameters

r,g,b

RGB integer values of a color.

rgb

vector of RGB integer values of a color.

names

names of the color.

Description

rgb2name returns the color name corresponding to the RGB values given by its argument. A vector of color names can also be returned if the color has more than 1 name. *r*, *g* and *b* must be integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color. RGB values can also be given by a vector [*r*, *g*, *b*].

If no color is found [] is returned.

The list of all known colors is given by color_list.

Examples

```
rgb2name(255,128,128)
rgb2name([255 215 0])
// get color #10 of the current colormap and find its name
cmap=get(gcf(),"color_map");
rgb2name(cmap(10,:)*255)
```

See Also

color , color_list , name2rgb

Name

rotate — rotation of a set of points

```
xy1=rotate(xy,[theta,orig])
```

Parameters

xy

matrix of size (2,.).

xy1

matrix of size (2,.).

theta

real, angle in radians; default value is 0.

orig

center of the rotation [xc;yc]; default value is [0;0].

Description

rotate performs a rotation with angle theta:

$$xy1(:,i) = M(\text{theta}) * (xy(:,i) - \text{orig}) + \text{orig}$$

where M stands for the corresponding rotation matrix.

Examples

```
xsetech([0,0,1,1],[-1,-1,1,1])
xy=[(0:0.1:10);sin(0:0.1:10)]/10;
for i=2*pi*(0:10)/10,
    [xy1]=rotate(xy,i);
    xpoly(xy1(1,:),xy1(2:,:), "lines")
end
```

Name

`rotate_axes` — Interactive rotation of an Axes handle.

```
rotate_axes()  
rotate_axes(h)
```

Parameters

`h`

Figure or Axes handle. Specify on which Axes the rotation will apply.

Description

`rotate_axes` function is used to perform an interactive rotation of an Axes object. When the function is called, the user is requested to click twice on the graphic window. The first click initializes the rotation and the second ends it.

If an Axes handle is specified as input argument, the rotation will apply on it. If a figure handle is specified, the first click determines the Axes object to rotate. If the function is called with no argument, the rotation apply on the current figure.

Examples

```
clf();  
// create two axes in the figure  
subplot(2, 1, 1);  
plot2d;  
subplot(2, 1, 2);  
plot3d;  
  
// rotate only the second axes  
axes2 = gca();  
rotate_axes(axes2);  
  
// rotate the selected one  
rotate_axes();  
// or  
rotate_axes(gcf());
```

See Also

`zoom_rect` , `axes_properties`

Authors

Jean-Baptiste Silvy INRIA

Name

rubberbox — Rubberband box for rectangle selection

```
[final_rect, btn]=rubberbox()  
[final_rect, btn]=rubberbox(initial_rect)  
[final_rect, btn]=rubberbox(edition_mode)  
[final_rect, btn]=rubberbox(initial_rect, edition_mode)
```

Parameters

initial_rect

vector with two or four entries. With four entries it gives the initial rectangle defined by [x_left, y_top, width, height], with two entries width and height are supposed to be 0.

edition_mode

a boolean, if edition_mode is %t button press selects the first corner, release selects the opposite corner. If edition_mode is %f, a button press or click selects the first corner, a click is requested to select the opposite corner. The default value is %f.

final_rect

a rectangle defined by [x_left, y_top, width, height]

btn

an integer, the number of the mouse button clicked

Description

`rubberbox(initial_rect)` tracks a rubberband box in the current graphic window, following the mouse. When a button is clicked `rubberbox` returns the final rectangles definition in `final_Rect`. If the argument `initial_rect` is not specified, a click is needed to fix the initial corner position.

Examples

```
xsetech(frect=[0,0,100,100])  
[x,y]=xclick();r=rubberbox([x;y;30;10])  
xrect(r)  
r=rubberbox()
```

See Also

`xrect` , `xrects` , `xclick` , `xgetmouse` , `dragrect`

Name

sca — set the current axes entity

```
a=sca(a)
```

Parameters

a
a handle, the handle on the Axes entity

Description

sca(a) is used to set the current Axes entity (see graphics_entities) to the one pointed to by the handle a. The drawing functions generally use the current axes entity.

Examples

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t), 'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))

sca(a1); //make first axes current
plot(t,asinh(t), 'g')
legend(['acosh', 'asinh'])
sca(a2); //make secoçnd axes current
legend('sinh')
```

See Also

subplot , gda , newaxes

Authors

S. Steer, INRIA

Name

scaling — affine transformation of a set of points

```
xy1=scaling(xy, factor, [orig])
```

Parameters

xy1

matrice of size (2,.).

xy

matrice of size (2,.).

factor

real scalar, coefficient of the linear transformation.

orig

shift vector; default value is [0;0].

Description

scaling performs an affine transformation on the set of points defined by the coordinates xy:

$$xy1(:,i) = factor * xy(:,i) + orig.$$

Name

`scf` — set the current graphic figure (window)

```
f = scf()  
f = scf(h)  
f = scf(num)
```

Parameters

`h`
a handle, the figure handle

`num`
a number, the figure_id

`f`
the handle of the current figure

Description

The current figure is the destination of the graphic drawing. The `scf` function allows to change this current figure or to create it if it does not already exist.

`scf(num)` set the figure with `figure_id==num` as the current figure. If it does not already exist it is created.

`scf(h)` set the figure pointed to by the handle `h` as the current figure. If it does not already exist it is created.

`scf()` is equivalent to `scf(max(winsid())+1)` . It may be used to create a new graphic window.

Examples

```
f4=scf(4); //creates figure with id==4 and make it the current one  
f0=scf(0); //creates figure with id==0 and make it the current one  
plot2d() //draw in current figure (id=0)  
scf(f4); // set first created figure as current one  
plot3d() //draw in current figure (id=4)
```

See Also

`set` , `get` , `gcf` , `clf` , `get_figure_handle` , `graphics_entities`

Authors

S. Steer INRIA

Name

sd2sci — gr_menu structure to scilab instruction convertor

```
txt=sd2sci(sd [,sz [,orig]])
```

Parameters

sd

data structure build by gr_menu.

sz

vector of number or strings with two components, give the x and y zoom factors

orig

vector of number or strings with two components, give the origin translation vector

Description

given a sd data structure generated by gr_menu sd2sci forms a vector of scilab instructions corresponding to the graphic edited by gr_menu.

The optional parameters sz and orig allows to zoom and shift the initial graphic.

If sz or orig are given by strings generated instructions are relative use then as formal expressions.

See Also

execstr

Authors

Serge Steer INRIA 1988

Name

sda — Set default axes.

```
sda()  
a = gda(); set(a, "default_values", 1)
```

Parameters

a
handle, the handle of the default axes.

Description

This routine resets the axes' model to default values.

Examples

```
x=[0:0.1:2*pi]';  
f=get("default_figure"); // get the handle of the model figure  
a=get("default_axes");   // get the handle of the model axes  
                           // setting its' properties  
f.figure_size=[1200 900];  
f.figure_position=[0 0];  
a.background=4;  
a.box="off";  
a.foreground=5;  
a.labels_font_color=25;  
a.labels_font_size=4;  
a.sub_tics=[7 3];  
a.x_location="middle";  
a.y_location="middle";  
a.tight_limits="on";  
a.thickness=2;  
a.grid=[-1 24];  
subplot(221);  
plot2d(x-2, sin(x))  
subplot(222);  
plot2d(x-6, [2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)], [-1, -2, -3 -4])  
sda() // return to the default values of the axes' model  
subplot(223);  
plot2d(x-2, sin(x))  
subplot(224);  
plot2d(x-6, [2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)], [-1, -2, -3 -4])  
xdel(0)  
plot2d(x-2, sin(x))
```

See Also

sdf , gda , gdf , set , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

sdf — Set default figure.

```
sdf()  
f = gdf(); set(f,"default_values",1)
```

Parameters

f
handle, the handle of the default figure.

Description

This routine resets the figure's model to default values.

Examples

```
x=[0:0.1:2*pi]';  
f=get("default_figure"); // get the handle of the model figure  
a=get("default_axes");   // get the handle of the model axes  
                           // setting its' properties  
f.background=4;  
f.auto_resize="off";  
f.figure_size=[400 300];  
f.axes_size=[600 400];  
f.figure_position=[0 -1];  
a.x_location="top";  
a.y_location="left";  
for (i=1:6)  
    xset("window",i) // create a figure with the identifier i  
    plot2d(x,[sin(x) cos(x)],[i -i])  
    xclick();  
    if i == 4, sdf(); end // return to the default values of the figure's model  
end
```

See Also

sda , gdf , gda , set , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

secto3d — 3D surfaces conversion

```
[m[,x]]=secto3d(seclist,npas)
[m]=secto3d(seclist ,x)
```

Parameters

seclist
a list whose elements are (2,.) matrices

npas
an integer

m
a matrix

x
a vector

Description

Considering a surface given through a list `seclist` of sections in the (x,z) plane `[m[,x]]=secto3d(seclist [,npas])` returns a matrix `m` which contains a regular discretization of the surface.

- The i -th row of the matrix `m` corresponds to the i -th section
- The j -th column of `m` corresponds to the `x(j)`

Each section `seclist(i)` is described by a (2,.) matrix which gives respectively the x and z coordinates of points.

`[m]=secto3d(seclist ,x)` in that case the x -vector gives the discretization of the x -axis for all the sections

See Also

`plot3d`

Authors

Steer S.; ;

Name

segs_properties — description of the Segments entity properties

Description

The Segs entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a set of colored segments or colored arrows.

parent:

This property contains the handle of the parent. The parent of the segment entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the children of the handle. However, segs handles currently do not have any children.

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, the segments are visible, the value's property is "on". If "off" the segments are not drawn on the screen.

data:

This field is two column matrix $[x, y, [z]]$ which gives the coordinates of the segments boundary. If $xv=matrix(x, 2, -1)$ and $yv=matrix(y, 2, -1)$ then $xv(:, k)$ and $yv(:, k)$ are the boundary coordinates of the segment numbered k.

line_mode:

This field contains the default `line_mode` property value for the segs. Its value should be "on" (line drawn) or "off" (no line drawn).

line_style:

The `line_style` property value should be an integer in [0 8]. 0 and 1 stands for solid, the other value stands for a selection of dashes (see `getlinestyle`).

thickness:

This property is a positive real specifying the arrow width in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

arrow_size:

Factor that specify the size of a arrowheads. With a negative value, the size is also dependent of the arrows length. TO draw segment, the value must be set to 0.

segs_color:

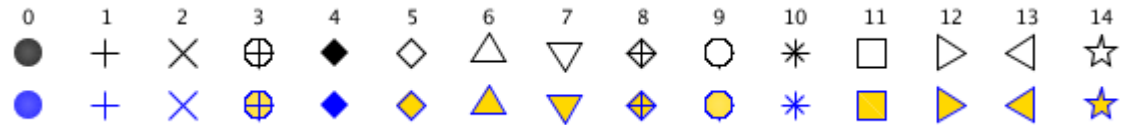
This field contains the vector of colors to use to draw each segment. Each element is a color index relative to the current colormap.

mark_mode:

This field contains the default `mark_mode` property value for the polyline. Its value should be "on" (marks drawn) or "off" (no marks drawn).

mark_style:

The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is "on". The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:

This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to "point", then the `mark_size` value is directly given in points. When `mark_size_unit` is set to "tabulated", `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that `plot2d` and pure scilab functions use `tabulated` mode as default ; when using `plot` function, the `point` mode is automatically enabled.

mark_size:

The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is "on". Its value should be an integer between 0 and 5 which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:

This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent edge.

mark_background:

This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent face.

clip_state:

This field contains the `clip_state` property value for the segments. It should be :

- "off" this means that the segments is not clipped.
- "clipgrf" this means that the segments is clipped outside the Axes box.
- "on" this means that the segments is clipped outside the rectangle given by the property `clip_box`.

clip_box:

This field contains the `clip_box` property. By default segment are not clipped, `clip_state` is "off", so the value should be an empty matrix. Other cases the vector `[x, y, w, h]` (upper-left point width height) defines the portions of the segments to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the `segs` data structure, and to retrieve it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-10,-10;10,10];
x=2*%pi*(0:7)/8;
xv=[2*sin(x);9*sin(x)];
yv=[2*cos(x);9*cos(x)];
xsegs(xv,yv,1:8)

s=a.children
s.arrow_size=1;
s.segs_color=15:22;
```

```
for j=1:2
    for i=1:8
        h=s.data(i*2,j);
        s.data(i*2,j)=s.data(i*2-1,j);
        s.data(i*2-1,j)= h;
    end
end

s.segs_color=5; //set all the colors to 5

s.clip_box=[-4,4,8,8];
a.thickness=4;
xrect(s.clip_box);
```

See Also

set , get , delete , xsegs , graphics_entities

Authors

Djalel ABDMOUCHE

Name

set — set a property value of a graphic entity object or of a User Interface object.

```
set(prop, val)
set(h, prop)
set(h, prop, val)
h.prop=val
```

Parameters

h
graphic handle of the entity which to set the named property. **h** can be a vector of handles, in which case set modifies the property for all entities contained in **h**.

prop
character string, name of the property to set.

val
value to give to the property.

Description

This routine can be used to modify the value of a specified property from a graphics entity or a GUI object. In this case it is equivalent to use the dot operator on an handle. For exemple, `set(h, "background", 5)` is equivalent to `h.background = 5`.

Property names are character strings. The type of the set values depends on the handle type and property.

To get the list of all existing properties see `graphics_entities` or `uicontrol` for User Interface objects.

`set` function can be also called with only a property and a value as argument. In this case, the property must be one of the following:

`current_entity` or `hdl`
`set('current_entity', h)` or `set('hdl', h)` sets a new entity as current. In this case, the value must be a graphic handle.

`current_figure`
`set('current_figure', fig)` sets a new graphic figure as current. It is equivalent to `scf`. In this case, the value must be a Figure handle.

`current_axes`
`set('current_axes', axes)` sets a new axes entity as current. It is equivalent to `sca`. In this case, the value must be an Axes handle.

`set` can also be called with a graphic handle and property as arguments. The handle must be either a handle on the default figure or the default axes entities. The property must be `"default_values"`. In this case, the default entity is reset to the value it had at Scilab startup. `set("default_values", h)` is equivalent to `sda` or `sdf`.

Examples

```
clf()
set("auto_clear", "off") ;
// Exemple of a Plot 2D
```

```
x=[-.2:0.1:2*pi]';
plot2d(x-.3,[sin(x-1) cos(2*x)],[1 2] );
a=get("current_axes");
p1=a.children.children(1);
p2=a.children.children(2);
// set the named properties to the specified values on the objects
set(p2,"foreground",13);
set(p2,"polyline_style",2);
set(a,'tight_limits',"on");
set(a,"box","off");
set(a,"sub_ticks",[ 7 0 ]);
set(a,"y_location","middle")
set(p2,'thickness',2);
set(p1,'mark_mode',"on");
set(p1,'mark_style',3);
plot2d(x-2,x.^2/20);
p3= a.children(1).children;
set([a p1 p2 p3],"foreground",5)
```

See Also

[get](#) , [delete](#) , [copy](#) , [move](#) , [graphics_entities](#) , [uicontrol](#)

Authors

Djalel ABDEMOUCHE

Name

seteventhandler — set an event handler for the current graphic window

```
seteventhandler(sfun_name)
seteventhandler('')
```

Parameters

sfun_name

a character string. The name of the Scilab function which is intended to handle the events

Description

The function allows the user to set a particular event handler for the current graphic window. `seteventhandler('')` removes the handler.

For more information about event handler functions see the event handler functions help.

Examples

```
function my_eventhandler(win,x,y,ibut)
    if ibut==-1000 then return,end
    [x,y]=xchange(x,y,'i2f')
    xinfo(msprintf('Event code %d at mouse position is (%f,%f)',ibut,x,y))
endfunction

plot2d()
seteventhandler('my_eventhandler')
//now:
// - move the mouse over the graphic window
// - press and release keys shifted or not with Ctrl pressed or not
// - press button, wait a little release
// - press and release button
// - double-click button

seteventhandler('') //suppress the event handler
```

See Also

addmenu , xgetmouse , xclick , xchange , event handler functions , figure_properties

Name

show_pixmap — send the pixmap buffer to the screen

```
show_pixmap()
```

Description

If a graphic window `pixmap` property is "on" the drawings are send to a pixmap memory instead of the screen display.

The `show_pixmap()` instruction send the pixmap to the screen.

The pixmap mode can be used to obtain smooth animations. This property can be found among the figure entity fields (see `figure_properties`).

Examples

```
f=gcf();f.pixmap='on'; //set the pixmap mode
a=gca();a.data_bounds=[0 0; 10 10];
//construct two rectangles
xrects([0;10;1;1],5);r1=gce();r1=r1.children;
xrects([0;1;1;1],13);r2=gce();r2=r2.children;
//animation loop
for k=1:1000
    //draw the rectangles in the pixmap buffer
    move(r1,[0.01,-0.01]);move(r2,[0.01,0.01])
    //show the pixmap buffer
    show_pixmap()
end
```

See Also

`figure_properties` , `clear_pixmap`

Authors

Serge Steer INRIA

Name

`show_window` — raises a graphics window

```
show_window([figure])
```

Parameters

`figure`
number or handle of the figure to show.

Description

With no parameters, `show_window` raises the current graphics window even if it is iconified. Otherwise, raises the specified window by its number or handle. If no window already exists, one is created.

Authors

J.Ph.C.

Name

springcolormap — magenta to yellow colormap

```
cmap=springcolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

springcolormap computes a colormap with n colors varying from magenta to yellow.

Examples

```
f = scf();  
plot3d1();  
f.color_map = springcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , summercolormap , whitecolormap , wintercolormap

Name

`square` — set scales for isometric plot (change the size of the window)

```
square(xmin, ymin, xmax, ymax)
```

Parameters

`xmin, xmax, ymin, ymax`
four real values

Description

`square` is used to have isometric scales on the x and y axes. The requested values `xmin`, `xmax`, `ymin`, `ymax` are the boundaries of the graphics frame and `square` changes the graphics window dimensions in order to have an isometric plot. `square` set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance `fstrf="x0z"` in `plot2d`).

Examples

```
t=[0:0.1:2*pi]';  
plot2d(sin(t),cos(t))  
clf()  
square(-1,-1,1,1)  
plot2d(sin(t),cos(t))  
xset("default")
```

See Also

`isoview` , `xsetech`

Authors

Steer S.

Name

stringbox — Compute the bounding rectangle of a text or a label.

```
rect = stringbox( string, x, y, [angle, [fontStyle, [fontSize]]] )  
rect = stringbox( Handle )
```

Parameters

rect

a 2x4 matrix containing the 4 vertex coordinates of the bounding rectangle.

string

string matrix to be enclosed.

x,y

real scalars, coordinates of the lower left point of strings.

angle

Rotation angle of the string clockwise and in degrees around the (x,y) point.

fonStyle

integer specifying the type of the font.

fontSize

integer specifying the size of the font.

Handle

a graphic handle of Text or Label type.

Description

stringbox returns the bounding rectangle vertices of a text or label object or a string which will be displayed with a certain way. the coordinates are given with the current graphic scale. the first vertex correspond to the text coordinates (x,y), the lower left point without rotation, the following vertex are given clockwise in the resulting matrix.

The result might not be very accurate with PostScript driver.

Examples

```
// show axes  
axes = gca() ;  
axes.axes_visible = 'on' ;  
axes.data_bounds = [ 1, 1 ; 10, 10 ] ;  
  
// display a labels for axes  
xtitle( 'stringbox', 'X', 'Y' ) ;  
  
// get the bounding box of X label  
stringbox( axes.x_label )  
  
// draw a string  
str = [ "Scilab", "is" , "not", "Skylab" ] ;  
xstring( 4, 9, str ) ;  
  
//modify the text
```

```
e = gce() ;
e.font_angle = 90 ;
e.font_size   = 6 ;
e.font_style  = 7 ;
e.box = 'on' ;

// get its bounding box
stringbox( e )
// or
rect = stringbox( str, 4, 9, 90, 7, 6 )

// click and find if the text was hit
hit = xclick() ;
hit = hit( 2 : 3 ) ;

if hit(1) >= rect(1,1) & hit(1) <= rect(1,2) & hit(2) <= rect(2,2) & hit(2) >
    disp('You hit the text.') ;
else
    disp('You missed it.')
end;
```

See Also

[xstring](#) , [xstringl](#) , [xstringb](#)

Authors

Jean-Baptiste Silvy

Name

subplot — divide a graphics window into a matrix of sub-windows

```
subplot(m,n,p)  
subplot(mnp)
```

Parameters

`m,n,p`
positive integers

`mnp`
an integer with decimal notation `mnp`

Description

`subplot(m,n,p)` or `subplot(mnp)` breaks the graphics window into an `m`-by-`n` matrix of sub-windows and selects the `p`-th sub-window for drawing the current plot. The number of a sub-window into the matrices is counted row by row ie the sub-window corresponding to element `(i,j)` of the matrix has number `(i-1)*n + j`.

Examples

```
subplot(221)  
plot2d()  
subplot(222)  
plot3d()  
subplot(2,2,3)  
param3d()  
subplot(2,2,4)  
hist3d()
```

See Also

`plot2d` , `plot3d` , `xstring` , `xtitle`

Name

summercolormap — green to yellow colormap

```
cmap=summercolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

summercolormap computes a colormap with *n* colors varying from green to yellow.

Examples

```
f = scf();  
plot3d1();  
f.color_map = summercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , whitecolormap , wintercolormap

Name

surf — 3D surface plot

```
surf(Z,<GlobalProperty>)  
surf(Z,color,<GlobalProperty>)  
surf(X,Y,Z,<color>,<GlobalProperty>)  
surf(<axes_handle>,...)
```

Parameters

Z

a real matrix defining the surface height. It can not be omitted. The Z data is a $m \times n$ matrix.

X,Y

two real matrices or vectors: always set together, these data defines a new standard grid. This new X and Y components of the grid must match Z dimensions (see description below).

color

an optional real matrix defining a color value for each $(X(j), Y(i))$ point of the grid (see description below).

<GlobalProperty>

This optional argument represents a sequence of couple statements `{PropertyName, PropertyValue}` that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see GlobalProperty).

<axes_handle>

This optional argument forces the plot to appear inside the selected axes given by `axes_handle` rather than the current axes (see `gca`).

Description

`surf` draws a colored parametric surface using a rectangular grid defined by X and Y coordinates (if `{X, Y}` are not specified, this grid is determined using the dimensions of the Z matrix) ; at each point of this grid, a Z coordinate is given using the Z matrix (only obligatory data). `surf` has been created to better handle Matlab syntax. To improve graphical compatibility, Matlab users should use `surf` (rather than `plot3d`).

Data entry specification :

In this paragraph and to be more clear, we won't mention `GlobalProperty` optional arguments as they do not interfere with entry data (except for "Xdata", "Ydata" and "Zdata" property, see GlobalProperty). It is assumed that all those optional arguments could be present too.

If Z is the only matrix specified, `surf(Z)` plots the matrix Z versus the grid defined by `1:size(Z,2)` along the x axis and `1:size(Z,1)` along the y axis.

If a `{X, Y, Z}` triplet is given, Z must be a matrix with `size(Z)=[m,n]`, X or Y can be :

- a) a vector : if X is a vector, `length(X)=n`. Respectively, if Y is a vector, `length(Y)=m`.
- b) a matrix : in this case, `size(X)` (or `size(Y)`) must equal `size(Z)`.

Color entry specification :

As stated before, the surface is created over a rectangular grid support. Let consider two independant variables `i` and `j` such as :

a) $1 \leq i \leq m$ and $1 \leq j \leq n$.

b) $i-1, j-1$ ---- $i-1, j$ ---- $i-1, j+1$ ---.. |

			i direction
$i, j-1$	----	i, j	----- $i, j+1$ ---.. V
:	:	:	
			----- > j direction

This imaginary rectangular grid is used to build the real surface support onto the XY plane. Indeed, X,Y and Z data have the same size (even if X or Y is vector, see below) and can be considered as 3 functions $x(i, j)$, $y(i, j)$ and $z(i, j)$ specifying the desired surface. If X or Y are vectors, they are internally treated to produce good matrices matching the Z matrix dimension (and the grid is forcibly a rectangular region).

Considering the 3 functions $x(i, j)$, $y(i, j)$ and $z(i, j)$, the portion of surface defining between two consecutive i and j is called a patch.

By default, when no color matrix is added to a surf call, the color parameter is linked to the Z data. When a color matrix is given, it can be applied to the patch in two different ways : at the vertices or at the center of each patch.

That is why, if Z is a $[m \times n]$ matrix, the C color matrix dimension can be $[m \times n]$ (one color defined per vertex) or $[m-1 \times n-1]$ (one color per patch).

Color representation also varies when specifying some GlobalProperty:

The FaceColor property sets the shading mode : it can be 'interp' or 'flat' (default mode). When 'interp' is selected, we perform a bilinear color interpolation onto the patch. If size(C) equals size(Z)-1 (i.e. we provided only one color per patch) then the color of the vertices defining the patch is set to the given color of the patch.

When 'flat' (default mode) is enabled we use a color faceted representation (one color per patch). If size(C) equals size(Z) (i.e. we provided only one color per vertices), the last row and column of C are ignored.

The GlobalProperty arguments could be used to customize the surface. Here is a brief description on how it works:

GlobalProperty

This option may be used to specify how all the surfaces are drawn. It must always be a couple statement constituted of a string defining the **PropertyName**, and its associated value **PropertyValue** (which can be a string or an integer or... as well depending on the type of the **PropertyName**). Note that you can set multiple properties : the face & edge color, color data, color data mapping, marker color (foreground and background), the visibility, clipping and thickness of the edges of the surface... (see GlobalProperty)

Note that all these properties can be (re-)set through the surface entity properties (see surface_properties).

Remarks

By default, successive surface plots are superposed. To clear the previous plot, use `clf()`. To enable `auto_clear` mode as the default mode, edit your default axes doing:

```
da=gda();
```

```
da.auto_clear = 'on'
```

Enter the command `surf` to see a demo.

Examples

```
// Z initialisation

Z= [  0.0001   0.0013   0.0053  -0.0299  -0.1809  -0.2465  -0.1100  -0.
    0.0005   0.0089   0.0259  -0.3673  -1.8670  -2.4736  -1.0866  -0.160
    0.0004   0.0214   0.1739  -0.3147  -4.0919  -6.4101  -2.7589  -0.277
   -0.0088  -0.0871   0.0364   1.8559   1.4995  -2.2171  -0.2729   0.836
   -0.0308  -0.4313  -1.7334  -0.1148   3.0731   0.4444   2.6145   2.441
   -0.0336  -0.4990  -2.3552  -2.1722   0.8856  -0.0531   2.6416   2.406
   -0.0137  -0.1967  -0.8083   0.2289   3.3983   3.1955   2.4338   1.212
   -0.0014  -0.0017   0.3189   2.7414   7.1622   7.1361   3.1242   0.663
    0.0002   0.0104   0.1733   1.0852   2.6741   2.6725   1.1119   0.197
    0.0000   0.0012   0.0183   0.1099   0.2684   0.2683   0.1107   0.019

//simple surface
surf(Z); // Note that X and Y are determined by Z dimensions

//same surface with red face color and blue edges
scf(2); // new figure number 2
surf(Z, 'facecol', 'red', 'edgecol', 'blu")

// X and Y initialisation
// NB: here, X has the same lines and Y the same columns
X = [ -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666
   -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.666

Y= [ -3.0000  -3.0000  -3.0000  -3.0000  -3.0000  -3.0000  -3.0000  -3.
   -2.3333  -2.3333  -2.3333  -2.3333  -2.3333  -2.3333  -2.3333  -2.333
   -1.6667  -1.6667  -1.6667  -1.6667  -1.6667  -1.6667  -1.6667  -1.666
   -1.0000  -1.0000  -1.0000  -1.0000  -1.0000  -1.0000  -1.0000  -1.000
   -0.3333  -0.3333  -0.3333  -0.3333  -0.3333  -0.3333  -0.3333  -0.333
    0.3333   0.3333   0.3333   0.3333   0.3333   0.3333   0.3333   0.333
    1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.000
    1.6667   1.6667   1.6667   1.6667   1.6667   1.6667   1.6667   1.666
    2.3333   2.3333   2.3333   2.3333   2.3333   2.3333   2.3333   2.333
    3.0000   3.0000   3.0000   3.0000   3.0000   3.0000   3.0000   3.000
```

```

// example 1
scf(3)
surf(X,Y,Z)

//example 2
// As you can see, the grid is not necessary rectangular
scf(4)
X(1,4) = -1.5;
Y(1,4) = -3.5;
Z(1,4) = -2;
surf(X,Y,Z)

// example 3
// X and Y are vectors => same behavior as sample 1
// With vectors, the grid is inevitably rectangular
scf(5)// new figure number 5
X=[ -3.0000  -2.3333  -1.6667  -1.0000  -0.3333   0.3333   1.0000   1.6667];
Y=X;
surf(X,Y,Z)

//LineSpec and GlobalProperty examples:
xdel(winsid()) // destroy all existing figures
surf(Z,Z+5) // color array specified
e=gce();
e.cdata_mapping='direct' // default is 'scaled' relative to the colormap
e.color_flag=3; // interpolated shading mode. The default is 4 ('flat' mode) for

scf(2)
surf(X,Y,Z, 'colorda',ones(10,10),'edgeco','cya','marker','penta','markersiz',20)

scf(3)
surf(Z,'cdamapping','direct')
scf(4)
surf(Z,'facecol','interp') // interpolated shading mode (color_flag == 3)

scf(10)
axfig10=gca();
scf(11);
surf(axfig10,Z,'ydat',[100:109],'marker','d','markerfac','green','markeredg','y')

xdel(winsid())

```

See Also

plot2d , clf , xdel , delete , LineSpec , GlobalProperty

Authors

F.Leray

Name

surface_properties — description of the 3D entities properties

Description

The Surface entity is a leaf of the graphics entities hierarchy. Two classes appears under this type of entity : `Plot3d` and `Fac3d` according to the plotting function or the way data is entered. `Fac3d` and `Plot3d` entities are similar but `Fac3d` is more complete and accept more options than `Plot3d`. To always have `Fac3d` entities, simply use `genfac3d` to pre-build matrices before using `plot3d` or use the `surf` command.

Here are the properties contained in a surface entity:

parent:

This property contains the handle of the parent. The parent of the surface entity should be of type "Axes" or "Compound".

children:

This property contains a vector with the `children` of the handle. However, surface handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity . It should be "on" or "off" . By default, surfaces are visibles, the value's property is "on". If "off" the 3D graphics are not displayed on the screen.

surface_mode:

This field contains the default `surface_mode` property value for the surface. Its value should be "on" (surface drawn) or "off" (no surface drawn).

foreground:

If `color_mode` ≥ 0 , this field contains the color index used to draw the edges. If not, foreground is not used at all. The foreground value should be an integer color index (relative to the current colormap).

thickness:

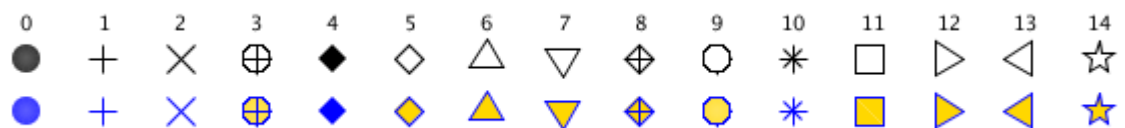
This property is a positive real specifying the width of facets contours in pixels. The displayed width is actually determined by rounding the supplied width to the nearest integer. The only exception is vectorial export where the whole `thickness` value is considered.

mark_mode:

This field contains the default `mark_mode` property value for the surface. Its value should be "on" (marks drawn) or "off" (no marks drawn).

mark_style:

The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is "on". The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:

This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to "point", then the `mark_size` value is directly given in points. When `mark_size_unit`

is set to `"tabulated"`, `mark_size` is computed relative to the font size array: therefore, its value should be an integer in `[0 5]` which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that `plot3d` and pure scilab functions use `tabulated` mode as default ; when using the `surf` (or `plot` for 2D lines) function, the `point` mode is automatically enabled.

`mark_size`:

The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is `"on"`. Its value should be an integer between 0 and 5 which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

`mark_foreground`:

This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent edge.

`mark_background`:

This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current `color_map`) or 0 for transparent face.

`data`:

This field defines a `tlist` data structure of type `"3d"` composed of a row and column indices of each element as the x-, y- and z-coordinates contained respectively in `data.x`, `data.y` and `data.z`. The complementary field named `data.color` is available in case a real color vector or matrix is specified. If none, `data.color` is not listed. The surface is painted according to `color_mode` and `color_flag` properties.

`color_mode`:

an integer between `[-size(colormap) ; size(colormap)]` defining the color of the facet when `color_flag` value is 0. As stated before, if `color_mode > 0`, edges are drawn using foreground color. If `color_mode` is set to 0, a mesh of the surface is drawn: front faces have no colors. Finally, when `color_mode < 0`, front faces are painted with color `-color_mode` but no edges are displayed.

`color_flag`:

This field is used to specify the algorithm used to set facets' colors.

Not that the rules on `color_mode`, `foreground` and `hiddencolor` are still applied to this case.

- `color_flag == 0`
 - All facets are painted using the color index and method defined by `color_mode` (see above).
- `color_flag == 1`
 - All facets are painted using one color index per facet proportional to Z. The minimum Z value is painted using the index 1 color while the maximum Z value is painted using highest color index. The edges of the facets can be additionally drawn depending on the value of `color_mode` (see above).
- The 3 remaining cases (`color_flag == 2, 3 or 4`) are only available only with `Fac3d` entity. Then, the `data.color` value is used to set colors for facets (indices in the current colormap) if it exists. If not, the current `color_mode` is used to paint the facets.
- `color_flag == 2 ('flat' shading)`
 - All facets are painted using the color index given in the `data.color` property (one color per facet is needed). Two cases are then possible :
 - `data.color` contains a color vector : if `color(i)` is positive it gives the color of facet `i` and the boundary of the facet is drawn with current line style and color. If `color(i)` is negative, color id `-color(i)` is used and the boundary of the facet is not drawn.

`data.color` contains a color matrix of size (nf,n) where n stands for the number of facets and nf for the number of points defining the polygonal facet. For the nf vertices defining each facet, the algorithm computes an **average value** of the color index (from the matrix color index) : the nf vertices of the same facet will have the same color index value.

- `color_flag == 3 ('interpolated' shading)`

- Facets painting results of interpolation of vertices colors. The indices of vertices color are given in the `data.color` property (one color per vertex is needed). Two cases are possible :
- `data.color` contains a `colors` vector : then, there are too few data to complete the interpolated shading mode. Indeed, a color matrix of size (nf,n) (where n stands for the number of facets and nf for the number of points defining the polygonal facet) is needed to perform this operation. For each facet, the algorithm copies the single color index value of the facet into the nf color indexes vertices defining the facet's boundary.

`data.color` contains a color matrix of size (nf,n) (see upper for nf and n definitions), the interpolated shading mode can be completed normally using those color indexes.

- `color_flag == 4 (Matlab-like 'flat' shading)`

- Same as `color_flag==2` with a slight difference when `data.color` is a matrix. All facets are painted using the color index given in the `data.color` property (one color per facet is needed). Two cases are then possible :
- `data.color` contains a `color` vector : if `color(i)` is positive it gives the color of facet i and the boundary of the facet is drawn with current line style and color. If `color(i)` is negative, color id `-color(i)` is used and the boundary of the facet is not drawn.

`data.color` contains a color matrix of size (nf,n) where n stands for the number of facets and nf for the number of points defining the polygonal facet. For the nf vertices defining each facet, the algorithm takes the **color of the first vertex** defining the patch (facet).

`cdata_mapping`:

Specific to `Fac3d` handles. A string with value 'scaled' or 'direct'. If a `data.color` is set, each index color data specifies a single value for each vertex. `cdata_mapping` determines whether those indices are scaled to map linearly into the current colormap ('scaled' mode) or point directly into this colormap ('direct' mode). This property is useful when `color_flag` equals 2,3 or 4.

`hiddencolor`:

This field contains the color index used to draw the backward faces of a surface. Its value should be a positive integer (color index relative to the current colormap). If it is a negative integer, the same color than the "visible" face is applied to the rear face.

`clip_state`:

This field contains the `clip_state` property value for the surface. It should be :

- "off" this means that the surface is not clipped.
- "clipgrf" this means that the surface is clipped outside the Axes box.
- "on" this means that the surface is clipped outside the rectangle given by property `clip_box`.

`clip_box`:

This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if `clip_state` is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the surface to display, however `clip_state` property value will be changed.

user_data:

This field can be use to store any scilab variable in the surface data structure, and to retrieve it.

Examples

```
//create a figure
t=[0:0.3:2*pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=get("hdl") //get handle on current entity (here the surface)
a=gca(); //get current axes
a.rotation_angles=[40,70];
a.grid=[1 1 1];
//make grids
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off";
//axes are hidden a.axes_bounds=[.2 0 1 1];
f=get("current_figure");
//get the handle of the parent figure
f.color_map=hotcolormap(64);
//change the figure colormap
h.color_flag=1;
//color according to z
h.color_mode=-2;
//remove the facets boundary
h.color_flag=2;
//color according to given colors
h.data.color=[1+modulo(1:400,64),1+modulo(1:400,64)];
//shaded
h.color_flag=3;

scf(2); // creates second window and use surf command
subplot(211)
surf(z,'cdata_mapping','direct','facecol','interp')

subplot(212)
surf(t,t,z,'edgeco','b','marker','d','markersiz',9,'markeredg','red','markerfac
e=gce();
e.color_flag=1 // color index proportional to altitude (z coord.)
e.color_flag=2; // back to default mode
e.color_flag= 3; // interpolated shading mode (based on blue default color beca
```

See Also

set , get , delete , plot3d , plot3d1 , plot3d2 , surf , graphics_entities

Authors

Djalel ABDEMOUCHE & F.Leray

Name

swap_handles — Permute two handles in the graphic Hierarchy.

```
swap_handle( handle1, handle2 )
```

Parameters

handle1

first handle of the permutation.

handle2

second handle of the permutation.

Description

The **swap_handles** function allows to permute two handles in the graphic hierarchy. The first handle will take the second handle position and vice versa.

Since not every handles are compatible with each others, some restrictions exist when swapping handles. For examples, it is not allowed to swap a polyline with an axes handle, since their would not be compatible with their new parents.. More information about compatibility can be found in the `graphics_entities` page.

This routine may be used on children of the same parent to change their indices..

Examples

```
//-----//
//  First example  //
//-----//

// create a rectangle
xrect( 0.5, 0.5,0.5,0.5) ;
rect = gce() ;

// create a circle
xarc( 0.5, 0.5, 0.5, 0.5, 0, 64 * 360 ) ;
circle = gce() ;

// create an arrow
xpoly([0,1],[0,1]) ;
arrow = gce() ;
arrow.polyline_style = 4 ;
arrow.arrow_size_factor = 4 ;

// get the list of children
axes = gca() ;
axes.children

// change the order
swap_handles( rect, arrow ) ;
swap_handles( arrow, circle ) ;

// get the new order
axes.children
```



```
//-----//  
//  Second example //  
//-----//  
  
// create two windows  
plot2d ;  
axes1 = gca() ;  
  
scf() ;  
fec ;  
axes2 = gca() ;  
  
// swap their axes  
// note that the color map does not change.  
swap_handles( axes1, axes2 ) ;
```

See Also

[graphics_entities](#) , [copy](#) , [delete](#) , [relocate_handle](#)

Authors

Jean-Baptiste Silvy

Name

text_properties — description of the Text entity properties

Description

The Text entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for string drawing

parent:

This property contains the handle of the parent. The parent of the text entity should be of the type "Axes" or "Compound".

children:

This property contains a vector with the `children` of the handle. However, text handles currently do not have any `children`.

visible:

This field contains the `visible` property value for the entity. It should be "on" or "off". By default, the text is visible, the value's property is "on". If "off" the text is not displayed on the screen.

text:

the matrix containing the strings of the object. The rows of the matrix are displayed horizontally and the columns vertically.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

alignment:

Specify how the strings are aligned in their columns. The value must be 'left', 'center' or 'right'.

data:

This field is the vector $[x, y, [z]]$ of the origin of the text in the data units of the axes.

box:

This field takes the values "on" or "off". If "on" a box is draw around the text with a line on its edge and a background.

line_mode:

This boolean property allows to draw or not a line around the box when the `box` property is "on". If `line_mode` is "off", the line of the box is not drawn.

fill_mode:

This boolean property allows to draw or not the background of the box when the `box` property is "on". If `fill_mode` is "off", the background of the box is not transparent.

text_box:

A two dimensionnal vector specifying the size of a rectangle in user coordinates. The rectangle is used when the `text_box_mode` property is set to 'centered' or 'filled'.

text_box_mode:

May have three different value : 'off', 'centered' or 'filled'. If 'off', the strings are displayed using the given font and the data field specifies the position of the lower-left point of the text. If 'centered', the text is displayed in the middle of the rectangle whose size is given by `text_box`. If 'filled' the font size of the strings will be expanded to fill the rectangle.

When using 'off' or 'centered' modes, text size remains constant upon zooming. They are the best modes to create annotations in a graph. On the contrary, when using the 'filled' mode, the text size follow the graphic scale. It is then possible to zoom upon text objects.

font_foreground:

This field contains the color used to display the characters of the text. Its value should be a color index (relative to the current colormap).

foreground:

This field contains the color used to display the line on the edge of the box. Its value should be a color index (relative to the current colormap).

background:

This field contains the color used to fill the box around of the text. Its value should be a color index (relative to the current colormap).

font_size:

It is a scalar specifying the displayed characters size. If **fractional_font** property is "off" only the integer part of the value is used. For more information see **graphics_fonts**.

font_style:

Specifies the font used to display the character strings. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see **graphics_fonts**.

fractional_font:

This property specify whether text is displayed using fractional font sizes. Its value must be either "on" or "off". If "on" the floating point value of **font_size** is used for display and the font is anti-aliased. If "off" only the integer part is used and the font is not smoothed.

font_angle:

This property determines the orientation of the text string. Specify value of rotation in degrees.

clip_state:

This field contains the **clip_state** property value for the text. Its value should be :

- "off" this means that the text is not clipped.
- "clipgrf" this means that the text is clipped outside the Axes box.
- "on" this means that the text is clipped outside the rectangle given by the property **clip_box**.

clip_box:

This field contains the **clip_box** property. Its value should be an empty matrix if **clip_state** is "off" or the vector $[x, y, w, h]$ (upper-left point width height).

user_data:

This field can be use to store any scilab variable in the text data structure, and to retrieve it.

Examples

```
a=get("current_axes");
a.data_bounds=[0,0;1,1];
a.axes_visible = 'on' ;

xstring(0.5,0.5,"Scilab is not esilaB",0,0)

t=get("hdl") //get the handle of the newly created object

t.font_foreground=6; // change font properties
t.font_size=5;
t.font_style=5;
```

```
t.text=["SCILAB","is";"not","esilaB"] ; // change the text
t.font_angle=90 ; // turn the strings
t.text_box = [0,0] ;
t.text_box_mode = 'centered' ; // the text is now centered on [0.5,0.5].
t.alignment = 'center' ;
t.box = 'on' ; // draw a box around the text

// Only valid from Scilab 5.2
mathml="<mrow>          <mfrac>          <mrow>          <mi>d</mi>          <mi>y</m
t.text=["SCILAB","can write LaTeX :", "$\frac{abc}{xyz}$";"or","MathML :",mathml
```

See Also

set , get , delete , xtitle , graphics_entities

Authors

Djalel ABDEMOUCHE, Jean-Baptiste SILVY

Name

`title` — display a title on a graphic window

```
title(my_title)
title(my_title, <Property>)
title(<axes_handle>, <my_title>, <Property>)
```

Parameters

`my_title`

a string, it's the title to display

`<Property>`

This optional argument represents a sequence of couple statements `{PropertyName, PropertyValue}` that defines global objects' properties applied to the created title.

`<axes_handle>`

This optional argument forces the title to appear inside the selected axes given by `axes_handle` rather than the current axes (see `gca`).

Description

`title` displays a title on a graphic window.

The `Property` arguments should be used to customize the title. Here is a complete list of the available options.

`Property` :

`backgroundcolor` : this field contains the color used to fill the box if any. Its value should be a color index (relative to the current colormap).

`color` : this field contains the color used to display the title text. Its value should be a color index (relative to the current colormap).

`edgecolor` : this field contains the color used to display the line around the box if any. Its value should be a color index (relative to the current colormap).

`fontname` : seven different fonts are available : "Courier", "Symbol", "Times", "Times Italic", "Times Bold", "User defined". The `font_size` property is an index in [0 6] which is associated to the previous font names.

`fontsize` : the `fontsize` property is used to select the type of size of the title. Its value should be an integer in between 0 and 5 which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

`position` : this 2d vector allows you to place manually the title on the screen. The position is stored in the data units of the axes.

`rotation` : this scalar allows you to turn the title. The font is turned inverse clockise with the angle given in degrees.

`visible` : this field contains the visible property value for the title. It should be "on" or "off". By default, the label is visible, the value's property is "on" . If "off" the title is not displayed on the screen.

Examples

```
// display a title with several properties
title('my title');
// change the color for the font
title('my title','color','blue');
// change the color for the around the box
title('my title','edgecolor','red');
// change the position of the title
title('my title','position',[0.3 0.8]);
// change the size of the font
title('my title','fontsize',3);
// a rotation
title('my title','rotation',90);

// We can do all these modifications with just the below instruction:
title('my title','color','blue','edgecolor','red','fontsize',3,'rotation',90,'p
```

See Also

[label_properties](#) , [titlepage](#) , [xtitle](#)

Authors

F.Belahcene

Name

`titlepage` — add a title in the middle of a graphics window

```
titlepage(str)
```

Parameters

`str`
matrix of strings

Description

`titlepage` displays the matrix of strings `str` in the middle of the current graphics window with a font as big as possible.

See Also

`xtitle`

Authors

S. S.

Name

`twinkle` — is used to have a graphics entity `twinkle`

```
twinkle(h,[n])
```

Parameters

`h`
handle of a graphics entity.

`n`
integer.

Description

`twinkle` let the graphics entity given by its handle `h` `twinkle`. It can be used to find the graphics object corresponding to a graphics handle in the graphics window. By default the graphics entity twinkles 5 times, but you can change this by using optional argument `n`.

Examples

```
x=linspace(-2*pi,2*pi,100)';  
plot2d(x,[sin(x),cos(x)]);  
e=gce(); p1=e.children(1); p2=e.children(2);  
// cos plot twinkle  
twinkle(p1)  
// sin plot twinkle 10 times  
twinkle(p2,10)  
// axes twinkle  
twinkle(gca())
```

See Also

`graphics_entities`

Name

unglue — unglue a coumpound object and replace it by individual children.

```
unglue(h)  
H=unglue(h)
```

Parameters

h
a handle on an Compound.

H
a vector of handle on the resulting entities after unCompound.

Description

Given a handle on an Compound entity, this function destroys the Compound and unpacks the elementary entities to associated them to its parent. `g lue` returns a vector of handles on these individual children.

See Also

`get` , `set` , `copy` , `glue` , `graphics_entities`

Authors

Djalel ABDEMOUCHE

Name

unzoom — unzoom graphics

```
unzoom()  
unzoom(H)
```

Parameters

H
A vector of Figure or Axes handle.

Description

`unzoom()` is used to remove the zoom effect for all the axes of the current graphic figure:

`unzoom(H)` is used to remove the zoom effect for all the Figures and Axes given by the vector of handles H. Removing zoom effect for a Figure is the equivalent of removing zoom effect on all its Axes children.

Examples

```
clf()  
x=0:0.01:6*pi;  
plot2d(x,sin(x^2))  
zoom_rect([16,-1,18,1])  
unzoom()  
  
//subplots accordingly  
clf()  
x=0:0.01:6*pi;  
subplot(211)  
plot2d(x,cos(x))  
a1=gca();  
subplot(212)  
plot2d(x,cos(2*x))  
a2=gca();  
rect=[3 -2 7 10]; // a rectangle specified in the current axes (last one) coord  
zoom_rect(rect)  
unzoom(a1) // unzoom first plot only  
unzoom(a2) // unzoom second plot only  
zoom_rect(rect) // zoom again  
unzoom(gcf()) // unzoom all the axes, equivalent to unzoom()
```

See Also

`zoom_rect` , `axes_properties`

Authors

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

Name

whitecolormap — completely white colormap

```
cmap=whitecolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

This colormap is completely white

Examples

```
f = scf();  
plot3d1();  
f.color_map = whitecolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , wintercolormap

Name

winsid — return the list of graphics windows

```
x=winsid()
```

Parameters

x
row vector.

Description

winsid is used to get the list of graphics windows as a vector of windows numbers.

Name

wintercolormap — blue to green colormap

```
cmap=wintercolormap(n)
```

Parameters

n
integer ≥ 3 , the colormap size.

cmap
matrix with 3 columns [R, G, B].

Description

wintercolormap computes a colormap with *n* colors varying from blue to green.

Examples

```
f = scf();  
plot3d1();  
f.color_map = wintercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap

Name

`xarc` — draw a part of an ellipse

```
xarc(x, y, w, h, a1, a2)
```

Parameters

`x,y,w,h`

four real values defining a rectangle.

`a1,a2`

real values defining a sector.

Description

`xarc` draws a part of an ellipse contained in the rectangle (x, y, w, h) (upper-left point, width, height), and in the sector defined by the angle `alpha1` and the angle `alpha1+alpha2`. `alpha1` and `alpha2` are given respectively by `a1/64` degrees and `a2/64` degrees. This function uses the current graphics color and user coordinates.

Examples

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xset("color",3)
xarc(-1,1,2,2,0,90*64)
xarc(-1.5,1.5,3,3,0,360*64)
```

See Also

`xsarcs` , `xfarc` , `xfarcs`

Authors

J.Ph.C.

Name

`xarcs` — draw parts of a set of ellipses

```
xarcs(arcs,[style])
```

Parameters

`arcs`

matrix of size (6,n) describing the ellipses.

`style`

row vector of size n giving the style to use.

Description

`xarcs` draws parts of a set of ellipses described by `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2; . . . ]'` where each ellipse is defined by the 6 parameters (`x`, `y`, `w`, `h`, `a1`, `a2`) (see `xarc`).

`x`, `y`, `w`, `h` parameters are specified in user coordinates.

`style(i)` gives the color used to draw ellipse number `i`.

Examples

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
      1.0 0.0 0.5; // upper left y
      0.5 1.0 0.5; // width
      0.5 0.5 1.0; // height
      0.0 0.0 0.0; // angle 1
      180*64 360*64 90*64]; // angle 2
xarcs(arcs,[1,2,3])
```

See Also

`xarc` , `xfarc` , `xfarcs`

Authors

J.Ph.C.;

Name

xarrows — draw a set of arrows

```
xarrows(nx,ny,[arsize,style])
```

Parameters

nx,ny

real vectors or matrices of same size.

arsize

real scalar, size of the arrow head. The default value can be obtained by setting arsize to -1.

style

matrix or scalar. If **style** is a positive scalar it gives the color to use for all arrows. If it is a negative scalar then the current color is used. If it is a vector **style(i)** gives the color to use for arrow **i**.

Description

xarrows draws a set of arrows given by **nx** and **ny**. If **nx** and **ny** are vectors, the *i*th arrow is defined by $(nx(i), ny(i)) \rightarrow (nx(i+1), ny(i+1))$. If **nx** and **ny** are matrices:

```
nx=[xi_1 x1_2 ...; xf_1 xf_2 ...]  
ny=[yi_1 y1_2 ...; yf_1 yf_2 ...]
```

the *k* th arrow is defined by $(xi_k, yi_k) \rightarrow (xf_k, yf_k)$.

xarrows uses the current graphics scale which can be set by calling a high level drawing function such as **plot2d**.

Examples

```
x=2*pi*(0:9)/8;  
x1=[sin(x);9*sin(x)];  
y1=[cos(x);9*cos(x)];  
plot2d([-10,10],[-10,10],[-1,-1],"022")  
xset("clipgrf")  
xarrows(x1,y1,1,1:10)  
xset("clipoff")
```

Authors

J.Ph.C.

Name

`xbasc` — clears a graphics window.

```
xbasc([window-id])
```

Parameters

`window-id`
integer scalar or vector

Description

Without any argument, this function clears the current graphic figure by deleting all its children. Otherwise it clears the graphic figures whose ids are included in the vector `window-id`. For example `xbasc(1:3)` clears windows 1, 2 and 3. If one of the windows does not exist, then it is automatically created.

`xbasc` function delete every children of specified windows including menus and uicontrols added by user. To prevent menus and uicontrols from being deleted, the `delete(gca())` command might be used instead.

The `xbasc` function is obsolete and will be removed in Scilab 5.3. To erase a figure, please use instead the `clf` or `delete` functions.

See Also

`clf` , `xclear`

Name

`xbasr` — redraw a graphics window

```
xbasr(win_num)
```

Description

`xbasr` is used to redraw the content of the graphics window with id `win_num`. It works only with the driver "Rec".

See Also

`driver` , `replot`

Authors

J.Ph.C.

Name

`xchange` — transform real to pixel coordinates

```
[x1,y1,rect]=xchange(x,y,dir)
```

Parameters

`x,y`

two matrices of size (n1,n2) (coordinates of a set of points).

`dir`

parameter used to specify the conversion type (See "Description" for details)

`x1,y1`

two matrices of size (n1,n2) (coordinates of the set of points).

`rect`

a vector of size 4.

Description

After having used a graphics function, `xchange` computes pixel coordinates from real coordinates and conversely, according to the value of the parameter `dir`: "f2i" (float to int) means real to pixel and "i2f" (int to float) means pixel to real. `x1` and `y1` are the new coordinates of the set of points defined by the old coordinates `x` and `y`.

`rect` is the coordinates in pixel of the rectangle in which the plot was done: [upper-left point, width, height].

Examples

```
t=[0:0.1:2*pi]';  
plot2d(t,sin(t))  
[x,y,rect]=xchange(1,1,"f2i")  
[x,y,rect]=xchange(0,0,"i2f")
```

Authors

J.Ph.C.

Name

`xclear` — clears a graphics window

```
xclear([window-id])
```

Parameters

`window-id`
integer scalar or vector

Description

Without any argument, this function clears the current graphic figure by turning its visibility to 'off'. Otherwise it clears the graphics figures whose numbers are included in the vector `window-id`. For example `xclear(1:3)` clears windows 1, 2 and 3. If one of the windows does not exist, then it is automatically created.

Function `xclear` is obsolete. To clear a figure, please use instead the `clf` function or the `visible` property.

See Also

`clf`

Authors

J.Ph.C.

Name

`xclick` — Wait for a mouse click.

```
[ibutton, xcoord, ycoord, iwin, cbmenu]=xclick([flag])
```

Parameters

`ibutton`

Real scalar (integer value): mouse button number, key code... (See description below).

`xcoord`

Real scalar: x-coordinate of the mouse pointer when the clic occurred, in current graphic scale.

`ycoord`

Real scalar: x-coordinate of the mouse pointer when the clic occurred, in current graphic scale.

`iwin`

Real scalar (integer value): number of the window where the action occurred.

`cbmenu`

String: callback associated to a menu if `xclick` returns due to a click on a menu. In this case, `ibutton`, `xcoord`, `ycoord`, and `iwin` take arbitrary values.

`flag`

Real scalar (integer value): If present, the click event queue is not cleared when entering `xclick`.

Description

`xclick` waits for a mouse click in the graphics window.

If it is called with 3 left hand side arguments, it waits for a mouse click in the current graphics window.

If it is called with 4 or 5 left hand side arguments, it waits for a mouse click in any graphics window.

The values of `ibutton` are described below.

`ibutton==0`

Left mouse button has been pressed.

`ibutton==1`

Middle mouse button has been pressed.

`ibutton==2`

Right mouse button has been pressed.

`ibutton==3`

Left mouse button has been clicked.

`ibutton==4`

Middle mouse button has been clicked.

`ibutton==5`

Right mouse button has been clicked.

`ibutton==10`

Left mouse button has been double-clicked.

`ibutton==11`

Middle mouse button has been double-clicked.

`ibutton==12`

Right mouse button has been double-clicked.

`ibutton >=32`

key with ASCII code `ibutton` has been pressed.

`ibutton <=32`

key with ASCII code `-ibutton` has been released.

`ibutton >=1000+32`

key with ASCII code `ibutton-1000` has been pressed while CTRL key pressed.

`ibutton==-1000`

graphic window has been closed.

WARNING: `ibutton` was equal to -100 for graphic window closure up to Scilab 4.1.2, but this code has been changed (in Scilab 5.0) because it was also the code returned for d key release.

`ibutton==-2`

A dynamic menu has been selected and its callback is returned in `cbmenu`.

See Also

`locate`, `xgetmouse`, `seteventhandler`

Authors

J.Ph.C.

V.C.

Name

`xdel` — delete a graphics window

```
xdel([win-nums])
```

Parameters

`win-nums`
integer or integer vector

Description

`xdel` deletes the graphics windows `win-nums` or the current graphics window if no argument is given.

Authors

J.Ph.C.

Name

`xfarc` — fill a part of an ellipse

```
xfarc(x, y, w, h, a1, a2)
```

Parameters

`x,y,w,h`

four real values defining a rectangle.

`a1,a2`

real values defining a sector.

Description

`xfarc` fills a part of an ellipse contained in the rectangle (x, y, w, h) (upper-left point, width, height), and in the sector defined by the angle `alpha1` and the angle `alpha1+alpha2`. `alpha1` and `alpha2` are given respectively by `a1/64` degrees and `a2/64` degrees. This function uses the current color and user coordinates.

Examples

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xfarc(-0.5,0.5,1,1,0,90*64)
xset("color",2)
xfarc(0.5,0.5,1,1,0,360*64)
```

See Also

`xarc` , `xarcs` , `xfarcs`

Authors

J.Ph.C.;

Name

`xfarcs` — fill parts of a set of ellipses

```
xfarcs(arcs,[style])
```

Parameters

`arcs`

matrix of size (6,n) describing the ellipses.

`style`

row vector of size n giving the colors to use.

Description

`xfarcs` fills parts of a set of ellipses described by `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2; . . . ]'` where each ellipse is defined by the 6 parameters (`x,y,w,h,a1,a2`) (see `xfarc`).

`x, y, w, h` parameters are specified in user coordinates.

`style(i)` gives the color number for filling ellipse number `i`.

Examples

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
      1.0 0.0 0.5; // upper left y
      0.5 1.0 0.5; // width
      0.5 0.5 1.0; // height
      0.0 0.0 0.0; // angle 1
      180*64 360*64 90*64]; // angle 2
xfarcs(arcs,[1,2,3])
```

See Also

`xfarc` , `xfarc` , `xfarc`

Authors

J.Ph.C.

Name

xfpoly — fill a polygon

```
xfpoly(xv,yv,[close])
```

Parameters

xv,yv

two vectors of same size (the points of the polygon).

close

integer. If close=1, the polyline is closed; default value is 0.

Description

xfpoly fills a polygon with the current color. If `close` is equal to 1 a point is added to the polyline `xv, yv` to define a polygon.

Examples

```
x=sin(2*pi*(0:4)/5);
y=cos(2*pi*(0:4)/5);
plot2d(0,0,-1,"010", " ", [-2,-2,2,2])
xset("color",5)
xfpoly(x,y)

// News graphics only
e=gce(); // get the current entity (the last created: here the polyline)
e.fill_mode='off';
e.closed = 'off' // the polyline is now open

xset("default")
```

See Also

xfpolys , xpoly , xpolys

Authors

J.Ph.C.

Name

xfpolys — fill a set of polygons

```
xfpolys(xpols,ypols,[fill])
```

Parameters

xpols,ypols

matrices of the same size (p,n) (points of the polygons).

fill

vector of size n or of size (p,n)

Description

xfpolys fills a set of polygons of the same size defined by the two matrices xpols and ypols. The coordinates of each polygon are stored in a column of xpols and ypols.

The polygons may be filled with a given color (flat) or painted with interpolated (shaded) colors.

flat color painting

In this case fill should be a vector of size n. The pattern for filling polygon number i is given by fill(i):

- if fill(i)<0, the polygon is filled with pattern id -fill(i).
- if fill(i)=0, the polygon is drawn with the current dash style (or current color) and not filled.
- if fill(i)>0, the polygon is filled with pattern id fill(i). Then its contour is drawn with the current dash (or color) and closed if necessary.

interpolated color painting

In this case fill should be a matrix with same sizes as xpols and ypols. Note that p must be equal to 3 or 4.

fill(k,i) gives the color at the kth edge of polygon i.

Examples

```
a=gca();a.data_bounds=[0,-10;210,40];a.foreground=color('red');
x1=[0,10,20,30,20,10,0]';
y1=[15,30,30,15,0,0,15]';
xpols=[x1 x1 x1 x1]; xpols=xpols+[0,60,120,180].*.ones(x1);
ypols=[y1 y1 y1 y1];
xfpolys(xpols,ypols,[-1,0,1,2])

// interpolated colors
clf()
f=gcf();
a=gca();a.data_bounds=[0,-10;40,30];a.isoview='on';
x1=[0,10,20,10]';
y1=[10,0,10,20]';
c=linspace(2,100,4)';
xpols=[x1 x1+20 x1+10 x1+10];
ypols=[y1 y1 y1+10 y1-10];
```

```
cols= [c c($:-1:1) c([3 4 1 2]) c]
f.color_map=jetcolormap(max(cols));
xfpolys(xpols,ypols,cols)

// interpolated colors
clf()
f=gcf();
x11=[0;20;20;0];y11=[10;10;30;30];c11=[10;10;30;30];
x12=x11;y12=y11+20;c12=[20;20;1;1];c12=[30;30;10;10];
x21=[0;30;30;0]+22;y21=[20;20;30;30];c21=[20;20;30;30];
x22=x21;y22=y21+10;c22=[30;30;20;20];
x31=[0;40;40;0]+55;y31=[0;0;30;30];c31=[0;0;30;30];
x32=x31;y32=y31+30;c32=[30;30;0;0];
X=[x11 x12 x21 x22 x31 x32];Y=[y11 y12 y21 y22 y31 y32];C=([c11 c12 c21 c22 c31 c32]);
a=gca();a.isoview='on';
a.data_bounds=[min(X),min(Y);max(X),max(Y)];
f=gcf();f.color_map=graycolormap(max(C));
xfpolys(X,Y,C)
```

See Also

[xfpoly](#) , [xpoly](#) , [xpolys](#)

Authors

J.Ph.C.

Name

xfrect — fill a rectangle

```
xfrect(x,y,w,h)
xfrect(rect) // rect =[x,y,w,h]
```

Parameters

x,y,w,h
four real values defining the rectangle.

Description

xrect fills a rectangle defined by $[x, y, w, h]$ (upper-left point, width, height) in user coordinates.

Examples

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xfrect(-1,1,2,2)
xset("default")
```

See Also

xrect , xrects

Authors

J.Ph.C.

Name

xget — get current values of the graphics context. **This function is obsolete.**

```
[x1]=xget(str,[flag])  
xget()
```

Parameters

str
string.

flag
optional. Set to 1 gives a verbose mode.

Description

Warning this function is obsolete. Use the Scilab graphic objects representation instead (see the set and get functions as well as the `graphics_entities` help page).

This function is used to get values from the graphics context on the topic specified by the string `str`. When called with no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

`number=xget("alufunction")`
Get the logical function number used for drawing. See `xset`.

`str=xget("auto clear")`
Get the auto clear status ("on" or "off").

`color=xget("background")`
Get the background color of the current Axes object. The result is a colormap index corresponding to the color.

`rect=xget("clipping")`
Get the clipping zone as a rectangle `rect=[x,y,w,h]` (Upper-Left point Width Height).

`c=xget("color")`
Get the default color for filling, line or text drawing functions. `c` is an integer projected in the interval `[0,whiteid]`. 0 stands for black filling and `whiteid` for white. The value of `whiteid` can be obtained with `xget("white")`.

`cmap=xget("colormap")`
Get the colormap used for the current graphics window as a `m x 3` RGB matrix.

`dash=xget("dashes")`
Get the dash style `dash=[dash_number]` where `dash_number` is the id of the dash. This keyword is obsolete, please use `xget("color")` or `xget("line style")` instead.

`font=xget("font")`
Get `font=[fontid,fontsize]`, the default font and the default size for fonts. size.

`fontsize=xget("font size")`
Get the default size for fonts size.

`color=xget("foreground")`
Get the foreground color of the current Axes object. The result is a colormap index corresponding to the color.

`str=xget("fpf")`

Get the floating point format for number display in contour functions. Note that `str` is "" when default format is used.

`color=xget("hidden3d")`

Get the color number for hidden faces in plot3d.

`pat=xget("lastpattern")`

Get the id of the last available pattern or color, with the current colormap of the current window. In fact `pat+1` and `pat+2` are also available and stand respectively for black and white pattern.

`type=xget("line mode")`

Get the line drawing mode. `type=1` is absolute mode and `type=0` is relative mode. (Warning: the mode `type=0` is has bugs)

`xget("line style")`

Get the default line style (1: solid, >1 for dashed lines).

`mark=xget("mark")`

Get the default mark id and the default marks size. `mark=[markid,marksize]`.

`marksize=xget("mark size")`

Get the default marks size.

`pat=xget("pattern")`

Get the current pattern or the current color. `pat` is an integer in the range `[1, last]`. When one uses black and white, 0 is used for black filling and `last` for white. The value of `last` can be obtained with `xget("lastpattern")`.

`value=xget("thickness")`

Get the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).

`flag=xget("use color")`

Get the flag 0 (use black and white) or 1 (use colors). See `xset`.

`[x,y]=xget("viewport")`

Get the current position of the visible part of graphics in the panner.

`dim=xget("wdim")`

Get the width and the height of the current graphics window `dim=[width,height]`.

`win=xget("window")`

Get the current window number `win`.

`pos=xget("wpos");`

Get the position of the upper left point of the graphics window `pos=[x,y]`.

See Also

`xset`, `getcolor`, `getsymbol`, `ged`, `set`, `graphics_entities`

Authors

J.Ph.C.

Name

xgetech — get the current graphics scale

```
[wrect, frect, logflag, arect]=xgetech()
```

Parameters

wrect, frect
real vectors.

logflag
string of size 2 "xy".

Description

xgetech returns the current graphics scale (of the current window). The rectangle [xmin,ymin,xmax,ymax] given by frect is the size of the whole graphics window. The plotting will be made in the region of the current graphics window specified by wrect.

wrect=[x, y, w, h] (upper-left point, width, height) describes a region inside the graphics window. The values in wrect are specified using proportion of the width and height of the graphics window:

wrect=[0 0 1 1] means that the whole graphics window is used.

wrect=[0.5 0 0.5 1] means that the graphics region is the right half of the graphics window.

logflag is a string of size 2 "xy", where x and y can be "n" or "l". "n" stands for normal (linear) scale and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

arect=[x_left, x_right, y_up, y_down] gives the frame size inside the subwindow. The graphic frame is specified (like wrect) using proportion of the width or height of the current graphic subwindow. Default value is 1/8*[1, 1, 1, 1]. If arect is not given, current value remains unchanged.

Examples

```
// first subwindow
xsetech([0,0,1.0,0.5])
plot2d()
// then xsetech is used to set the second sub window
xsetech([0,0.5,1.0,0.5])
grayplot()
// get the graphic scales of first subwindow
xsetech([0,0,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
// get the graphic scales of second subwindow
xsetech([0,0.5,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
clf();
xset('default')
```

See Also

xsetech

Authors

J.Ph.C.;

Name

`xgetmouse` — get the mouse events and current position

```
[rep [,win]]=xgetmouse([sel])
```

Parameters

`sel`
boolean vector [`getmotion`, `getrelease`]. default value is [%t, %f]

`rep`
vector of size 3, [x,y,ibutton].

`win`
number of the figure where the event occurred.

Description

If the mouse pointer is located in the current graphics window, `xgetmouse` returns in `rep` the current pointer position (x,y) and the value `ibutton`. The `ibutton` value indicates the event type:

`ibutton==0`
Left mouse button has been pressed

`ibutton==1`
Middle mouse button has been pressed

`ibutton==2`
Right mouse button has been pressed

`ibutton==3`
Left mouse button has been clicked

`ibutton==4`
Middle mouse button has been clicked

`ibutton==5`
Right mouse button has been clicked

`ibutton==10`
Left mouse button has been double-clicked

`ibutton==11`
Middle mouse button has been double-clicked

`ibutton==12`
Right mouse button has been double-clicked

`ibutton==-5`
Left mouse button has been released

`ibutton==-4`
Middle mouse button has been released

`ibutton==-3`
Right mouse button has been released

ibutton== -1
pointer has moved

ibutton >= 32
key with ascii code ascii(ibutton) has been pressed

ibutton <= -32
key with ascii code ascii(-ibutton) has been released

ibutton >= 1000+32
key with ascii code ascii(ibutton-1000) has been pressed while CTRL key pressed

ibutton== -1000
graphic window has been closed

WARNING: In previous versions of Scilab (<5.0), the user could give a flag to precise if the mouse click event queue had to be cleared when entering xgetmouse. This option has been removed in Scilab 5.1.

Examples

```
// rectangle selection
clf(); // erase/create window
a=gca();a.data_bounds=[0 0;100 100];//set user coordinates
xtitle(" drawing a rectangle ") //add a title
xselect(); //put the window on the top

[b,xc,yc]=xclick(); //get a point
xrect(xc,yc,0,0) //draw a rectangle entity
r=gce();// the handle of the rectangle
rep=[xc,yc, -1];first=%f;

while rep(3)==-1 do // mouse just moving ...
    rep=xgetmouse();
    xc1=rep(1);yc1=rep(2);
    ox=mini(xc,xc1);
    oy=maxi(yc,yc1);
    w=abs(xc-xc1);h=abs(yc-yc1);
    r.data=[ox,oy,w,h]; //change the rectangle origin, width and height
    first=%f;
end
```

See Also

locate , xclick , seteventhandler

Authors

S. Steer

Name

xgraduate — axis graduation

```
[xi, xa, np1, np2, kMinr, kMaxr, ar]=xgraduate(xmi, xma)
```

Parameters

xmi,xma

real scalars

xi, xa, kMinr, kMaxr, ar

real scalars

np1,np2

integer

Description

xgraduate returns the axis graduations which are used by the plot routines (with pretty print flag enabled). It returns an interval $[xi, xa]$ which contains the given interval $[xmi, xma]$ and such that

$xi = kMinr * 10^{ar}$, $xa = kMaxr * 10^{ar}$ and the interval can be divided into np2 intervals and each interval is divided in np1 sub-intervals.

Examples

```
[x1, xa, np1, np2, kMinr, kMaxr, ar]=xgraduate(-0.3, 0.2)
```

See Also

graduate , plot2d

Authors

J.P.C ; ;

Name

`xgrid` — add a grid on a 2D plot

```
xgrid([style])
```

Parameters

`style`
integer

Description

`xgrid` adds a grid on a 2D plot. `style` is the dash id or the color id to use for the grid plotting. Use `xset()` for the meaning of id.

Examples

```
x=[0:0.1:2*pi]';  
plot2d(sin(x))  
xgrid(2)
```

See Also

`xset` , `plot2d`

Authors

J.Ph.C.

Name

xinfo — draw an info string in the message subwindow

```
xinfo(info)
```

Parameters

info
string

Description

xinfo draws the string `info` in the message subwindow of the current graphics window.

Name

`xlfont` — load a font in the graphic context or query loaded font

```
xlfont(font-name)
xlfont(font-filename)
xlfont('reset')
xlfont(font-name, font-id)
xlfont(font-filename, font-id)
xlfont(font-name, font-id, bold)
xlfont(font-name, font-id, bold, italic)
fonts=xlfont('AVAILABLE_FONTS')
fonts=xlfont()
```

Parameters

font-name
string, name of the font family.

font-filename
string, filename of a true type font.

font-id
integer ≥ 0 .

fonts
a column vector of font names.

bold
a boolean %t if bold

italic
a boolean %t if italic

Description

Without any argument, `xlfont()` returns the list of currently loaded fonts.

`xlfont('AVAILABLE_FONTS')` returns list of fonts available on your system.

`xlfont('reset')` reset to initial index list of fonts.

With arguments, `xlfont` is used to load a new font at different sizes in the graphics context.

Default family fonts are "Monospaced" (0), "Symbol" (1), "Serif" (2), "Serif Italic" (3), "Serif Bold" (4), "Serif Bold Italic" (5), "SansSerif" (6), "SansSerif Italic" (7), "SansSerif Bold" (8), "SansSerif Bold Italic" (9). These default fonts are automatically loaded when needed and so `xlfont` is not really required for them. In fact `xlfont` is essentially useful to load a new font.

Examples

```
xlfont('reset');
xlfont()

// Caution : this example may not work if your system have not Monospaced font.
xlfont("Monospaced",10,%t,%t);
xstring(1,0,'A title');
```

```
figure_entity = gcf();  
axes_entity = figure_entity.children;  
title_entity = axes_entity.children;  
title_entity.font_style = 10;  
  
xlfont('reset');
```

See Also

[getfont](#)

Authors

Allan CORNET

Name

xload — load a saved graphics

```
xload(file_name,[win_num])
```

Parameters

file_name

string, name of the file.

win_num

integer, the graphics window number. If not given, the current graphics window is used.

Description

xload reloads the graphics contained in the file file_name in the graphics window win_num.

Since Scilab 5.0, all uimenu or uicontrol handles are also loaded.

For files containing graphics, the load function can be used instead of xload. xload does not restore the window number, the window size nor the window dimensions.

Examples

```
t=0:0.01:10;
subplot(211),plot2d(t,sin(t))
subplot(212),plot2d(t,sin(3*t))
save(TMPDIR+'/foo.scg',gcf())
clf()
load(TMPDIR+'/foo.scg')

a=gca();
curve=a.children.children; //handle on the curve
save(TMPDIR+'/foo.scg',curve)
delete(curve)
load(TMPDIR+'/foo.scg')
```

See Also

xsave , load , save

Authors

J.Ph.C.

Name

xname — change the name of the current graphics window

```
xname( name )
```

Parameters

name
string, new name of the graphics window.

Description

xname changes the name of the current graphics window.

Authors

J.Ph.C.

Name

xnumb — draw numbers

```
xnumb(x, y, nums, [box, angle])
```

Parameters

x,y,nums

vectors of same size.

box

integer value.

angle

optional vector of same size as x

Description

xnumb draws the value of `nums(i)` at position `x(i)`, `y(i)` in the current scale. If `box` is 1, a box is drawn around the numbers. If `angle` is given, it gives the direction for string drawing.

Examples

```
plot2d([-100, 500], [-100, 600], [-1, -1], "022")
x=0:100:200;
xnumb(x, 500*ones(x), [10, 20, 35], 1)
```

See Also

xstring

Authors

J.Ph.C.

Name

xpause — suspend Scilab

`xpause(microsecs)`

Description

`xpause` suspends the current process for the number of microseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

Authors

J.Ph.C.

Name

xpoly — draw a polyline or a polygon

```
xpoly(xv,yv [, dtype [, close]])
```

Parameters

xv,yv

matrices of the same size (points of the polyline).

dtype

string (drawing style). default value is "lines".

close

integer. If close=1, the polyline is closed; default value is 0.

Description

xpoly draws a single polyline described by the vectors of coordinates xv and yv. If xv and yv are matrices they are considered as vectors by concatenating their columns. dtype can be "lines" for using the current line style or "marks" for using the current mark to draw the polyline.

Examples

```
x=sin(2*pi*(0:4)/5);
y=cos(2*pi*(0:4)/5);

plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xpoly(x,y,"lines",1) // by default closed

// News graphics only
e=gce(); // get the current entity (the last created: here the polyline)
e.closed = 'off' // the polyline is now open
```

See Also

xfpoly , xfpolys , xpolys

Authors

J.Ph.C.

Name

xpolys — draw a set of polylines or polygons

```
xpolys(xpols,ypols,[draw])
```

Parameters

xpols,ypols

matrices of the same size (p,n) (points of the polylines).

draw

vector of size n.

Description

xpolys draws a set of polylines using marks or dashed lines. The coordinates of each polyline are stored in a column of xpols and ypols.

The style of polyline i is given by draw(i):

- If draw(i) is negative, the mark with id -draw(i) is used to draw polyline i (marks are drawn using the current pattern). Use xset() to see the meaning of the ids.
- If draw(i) is strictly positive, the line style (or color) with id draw(i) is used to draw polyline i. Use xset() to see the meaning of the ids.

Examples

```
plot2d(0,0,-1,"012","",[0,0,1,1])
rand("uniform")
xset("color",3)
xpolys(rand(3,5),rand(3,5),[-1,-2,0,1,2])
xset("default")
```

See Also

xfpoly , xfpolys , xpoly

Authors

J.Ph.C.

Name

xrect — draw a rectangle

```
xrect(x,y,w,h)
xrect(rect) // rect =[x,y,w,h]
```

Parameters

x,y,w,h
four real values defining the rectangle.

Description

xrect draws a rectangle defined by $[x, y, w, h]$ (upper-left point, width, height) in user coordinates.

WARNING: please note that height is positive downwards.

Examples

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xrect(-1,1,2,2)
xset("default")
```

See Also

xfrect , xrects

Authors

J.Ph.C.

Name

xrects — draw or fill a set of rectangles

```
xrects(rects,[fill])
```

Parameters

rects
matrix of size (4,n).

fill
vector of size n.

Description

xrects draws or fills a set of rectangles. Each column of `rects` describes a rectangle (upper-left point, width, height) in user coordinates: `rects=[x1 y1 w1 h1;x2 y2 w2 h2;...]`'.

`fill(i)` gives the pattern to use for filling or drawing rectangle `i`:

if `fill(i)<0`, rectangle `i` is drawn using the line style (or color) `-fill(i)`

if `fill(i)>0`, rectangle `i` is filled using the pattern (or color) `fill(i)`

if `fill(i)=0`, rectangle `i` is drawn using the current line style (or color).

WARNING: please note that height is positive downwards.

Examples

```
plot2d([-100,500],[-50,50],[-1,-1],"022")
cols=[-34,-33,-32,-20:5:20,32,33,34];
x=400*(0:14)/14; step=20;
rects=[x;10*ones(x);step*ones(x);30*ones(x)];
xrects(rects,cols)
xnumb(x,15*ones(x),cols)
```

See Also

xfrect , xrect

Authors

J.Ph.C.

Name

xrpoly — draw a regular polygon

```
xrpoly(orig,n,r,[theta])
```

Parameters

orig

vector of size 2.

n

integer, number of sides.

r

real scalar.

theta

real, angle in radian; 0 is the default value.

Description

xrpoly draws a regular polygon with *n* sides contained in the circle of diameter *r* and with the origin of the circle set at point *orig*. *theta* specifies a rotation angle in radian. This function uses the current graphics scales.

Examples

```
plot2d(0,0,-1,"012","",[0,0,10,10])
xrpoly([5,5],5,5)
```

See Also

xrect

Name

xsave — save graphics into a file

```
xsave(filename, [win_num])
```

Parameters

file_name

string, name of the file.

win_num

integer, the graphics window number. If not given, the current graphics window is used.

Description

xsave saves the graphics contained in the graphics window win_num in the binary file file_name. and can be reloaded with xload.

Since Scilab 5.0, all uimenu or uicontrol handles are also saved.

For graphics `xsave(file_name, win_num)` use preferably `save(file_name, scf(win_num))`.

Examples

```
t=0:0.01:10;
subplot(211), plot2d(t, sin(t))
subplot(212), plot2d(t, sin(3*t))
save(TMPDIR+'foo.scg', gcf())
clf()
load(TMPDIR+'foo.scg')

a=gca();
curve=a.children.children; //handle on the curve
save(TMPDIR+'foo.scg', curve)
delete(curve)
load(TMPDIR+'foo.scg')
```

See Also

xload , save , load

Authors

J.Ph.C.

Name

xsegs — draw unconnected segments

```
xsegs(xv,yv,[style])
xsegs(xv,yv,zv,[style])
```

Parameters

xv, *yv*, *zv*

matrices of the same size. If *zv* is not specified, a zero vector is used.

style

vector or scalar. If *style* is a positive scalar, it gives the color to use for all segments. If *style* is a negative scalar, then current color is used. If *style* is a vector, then *style(i)* gives the color to use for segment *i*.

Description

xsegs draws a set of unconnected segments given by *xv*, *yv* and *zv*. If *xv*, *yv* and *zv* are matrices they are considered as vectors by concatenating their columns. The coordinates of the two points defining a segment are given by two consecutive values of *xv*, *yv* and *zv*:

$(xv(i), yv(i), zv(i)) \rightarrow (xv(i+1), yv(i+1), zv(i+1))$.

For instance, using matrices of size (2,n), the segments can be defined by:

```
xv=[xi_1 xi_2 ...; xf_1 xf_2...]
yv=[yi_1 yi_2 ...; yf_1 yf_2...]
zv=[zi_1 zi_2 ...; zf_1 zf_2...]
```

and the segments are $(xi_k, yi_k, zi_k) \rightarrow (xf_k, yf_k, zf_k)$.

Examples

```
// 2D example
x=2*pi*(0:9)/10;
xv=[sin(x);9*sin(x)];
yv=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xsegs(xv,yv,1:10)

// 3D example
clf();
a=gca();
a.view="3d";
f=gcf();
f.color_map=rainbowcolormap(120);
alpha=2*pi*(0:119)/40;
xv=[sin(alpha)/2;sin(alpha)/3];
yv=[cos(alpha)/2;cos(alpha)/3];
```

```
zv=[alpha/8;alpha/8];
xsegs(xv,yv,zv,1:120);
// Now adjust the data_bounds
a.data_bounds = [min(xv) min(yv) min(zv); ...
                  max(xv) max(yv) max(zv)];
// We can add an arrow to each segs
e = gce();
e.arrow_size = 0.4;
```

Authors

J.Ph.C.

Name

`xselect` — raise the current graphics window

```
xselect()
```

Description

`xselect` raises the current graphics window. It creates the window if none exists.

Warning: This function is obsolete and will be removed in Scilab 5.1. It has been replaced by the `show_window` function.

See Also

`show_window`

Authors

J.Ph.C.

Jean-Baptiste Silvy

Name

`xset` — set values of the graphics context. This function is obsolete.

```
xset(choice-name,x1,x2,x3,x4,x5)
xset()
```

Parameters

choice-name
string

x1,...,x5
depending on choice-name

Description

Warning this function is obsolete. Use the Scilab graphic objects representation instead (see the `set` and `get` functions as well as the `graphics_entities` help page).

`xset` is used to set default values of the current window graphic context.

When called no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

Use `xset()` to display or set the current color, mark and fonts used.

`xset("alufunction",number)`

Used to set the logical function for drawing. The logical function used is set by `x1`. Usual values are: 3 for copying (default), 6 for animation and 0 for clearing. See `alufunctions` for more details.

`xset("auto clear","on"|"off")`

Switch "on" or "off" the auto clear mode for graphics. When the auto clear mode is "on", successive plots are not superposed, ie a `clf()` operation (the graphics window is cleared and the associated recorded graphics is erased) is performed before each high level graphics function. Default value is "off".

`xset("background",color)`

Set the background color of the current Axes object. The `color` argument is the colormap index of the color to use.

`xset("clipping",x,y,w,h)`

Set the clipping zone (the zone of the graphics window where plots can be drawn) to the rectangle (x,y,w,h) (Upper-Left point Width Height). This function uses the current coordinates of the plot.

`xset("color",value)`

Set the default color for filling, line or text drawing functions. `value` is an integer projected in the interval [0,whiteid]. 0 is used for black filling and whiteid for white. The value of whiteid can be obtained with `xget("white")`.

`xset("colormap",cmap)`

Set the colormap as a `m x 3` matrix. `m` is the number of colors. Color number `i` is given as a 3-uple `cmap(i,1)`, `cmap(i,2)`, `cmap(i,3)` corresponding respectively to red, green and blue intensity between 0 and 1.

`xset("dashes",i)`

In black and white mode (`xset("use_color",0)`), set the dash style to style `i` (0 for solid line). In color mode (`xset("use_color",1)`) this is used to set line, mark and text color.

This keyword is obsolete, please use `xset('color',i)` or `xset('line style',i)` instead.

`xset("default")`

Reset the graphics context to default values.

`xset("font",fontid,fontsize)` : Set the current font and its current size. Note that `fontsize` applies to all fonts not only `fontid`.

`xset("font size",fontsize)`

Set the fonts size.

`xset("foreground",color)`

Set the foreground color of the current Axes object. The `color` argument is the colormap index of the color to use.

`xset("fpf",string)`

Set the floating point format for number display in contour functions. `string` is a string giving the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format.

`xset("hidden3d",colorid)` : Set the color number for backward facing faces in `plot3d`. `colorid=0` zero suppress the drawing of backward facing faces of 3d objects. This is technically called 'culling' and speeds up the rendering of closed surfaces.

`xset("line mode",type)`

This function is used to set the line drawing mode. Absolute mode is set with `type=1` and relative mode with `type=0`. (Warning: the mode `type=0` has bugs)

`xset("line style",value)`

Set the current line style (1: solid, >1 for dashed lines).

`xset("mark",markid,marksize)`

Set the current mark and the current mark size. Use `xset()` to see the marks. Note that `mark-size` applies to all marks not only `markid`.

`xset("mark size",marksize)`

Set the marks size.

`xset("pattern",value)`

Set the current pattern for filling functions. `value` is an integer projected in the interval `[0,whiteid]`. 0 is used for black filling and `whiteid` for white. The value of `whiteid` can be obtained with `xget("white")`. "pattern" is equivalent to "color".

`xset("pixmap",flag)`

If `flag=0` the graphics are directly displayed on the screen. If `flag=1` the graphics are done on a pixmap and are sent to the graphics window with the command `xset("wshow")`. The pixmap is cleared with the command `xset("wwpc")`. Note that the usual command `clf()` also clears the pixmap.

`xset("thickness",value)`

Set the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).

`xset("use color",flag)`

If `flag=1` then `xset("pattern",.)` or `xset("dashes",.)` will be used so as to change the default color for drawing or for filling patterns. If `flag=0` then we switch back to the gray and dashes mode.

`xset("viewport",x,y)`

Set the position of the panner.

`xset("wdim",width,height)`

Set the width and the height of the current graphics window. This option is not used by the postscript driver.

`xset("wpdim",width,height)`

Sets the width and the height of the current physical graphic window (which can be different from the actual size in mode `wresize 1`). This option is not used by the postscript driver.

`xset("window",window-number)`

Set the current window to the window `window-number` and creates the window if it does not exist.

`xset("wpos",x,y)`

Set the position of the upper left point of the graphics window.

`xset("wresize",flag)`

If `flag=1` then the graphic is automatically resized to fill the graphics window.

```
xdel();
xset("wresize",1);
plot2d();
xset("wdim",1000,500)
```

If `flag=0` the scale of the graphic is left unchanged when the graphics window is resized. Top left panner or keyboard arrows may be used to scroll over the graphic.

```
xdel();
plot2d();
xset("wresize",0);
xset("wdim",1000,500)
```

`xset("wshow")`

See `xset("pixmap",1)` above.

`xset("wwpc")`

See `xset("pixmap",1)` above.

See Also

`xget` , `getcolor`, `getsymbol`, `ged`, `set`, `graphics_entities`

Authors

J.Ph.C.

Name

xsetech — set the sub-window of a graphics window for plotting

```
xsetech(wrect,[frect,logflag])
xsetech(wrect=[...],frect=[..],logflag="..", arect=[...])
xsetech()
```

Parameters

wrect

vector of size 4, defining the sub-window to use.

frect

vector of size 4.

logflag

string of size 2 "xy", where x and y can be "n" or "l". "n" stands for normal and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

arect

vector of size 4.

Description

xsetech is mainly used to set the sub-window of the graphics window which will be used for plotting. The sub-window is specified with the parameter `wrect=[x,y,w,h]` (upper-left point, width, height). The values in `wrect` are specified using proportion of the width or height of the current graphic window. For instance `wrect=[0,0,1,1]` means that the whole graphics window will be used, and `wrect=[0.5,0,0.5,1]` means that the graphics region will be the right half of the graphics window.

xsetech also set the current graphics scales for 2D plotting and can be used in conjunction with graphics routines which request the current graphics scale (for instance `strf="x0z"` or `frameflag=0` in `plot2d`).

`frect=[xmin,ymin,xmax,ymax]` is used to set the graphics scale and is just like the `rect` argument of `plot2d`. If `frect` is not given the current value of the graphic scale remains unchanged. the default value of `rect` is `[0,0,1,1]` (at window creation, when switching back to default value with `xset('default')` or when clearing graphic recorded events `clf()`).

`arect=[x_left, x_right,y_up,y_down]` is used to set the graphic frame inside the sub-window. The graphic frame is specified (like `wrect`) using proportion of the width or height of the current graphic subwindow. Default value is $1/8 * [1,1,1,1]$. If `arect` is not given, current value remains unchanged.

Examples

```
// To get a graphical explanation of xsetech parameters enter:
exec('SCI/modules/graphics/demos/xsetechfig.sce');

// Here xsetech is used to split the graphics window in two parts
// first xsetech is used to set the first sub-window
// and the graphics scale
xsetech([0,0,1.0,0.5],[-5,-3,5,3])
// we call plot2d with the "001" option to use the graphics scale
// set by xsetech
```

```
plot2d([1:10]',[1:10]',1,"001"," ")
// then xsetech is used to set the second sub-window
xsetech([0,0.5,1.0,0.5])
// the graphics scale is set by xsetech to [0,0,1,1] by default
// and we change it with the use of the rect argument in plot2d
plot2d([1:10]',[1:10]',1,"011"," ",[-6,-6,6,6])
// Four plots on a single graphics window
clf()
xset("font",2,0)
xsetech([0,0,0.5,0.5]); plot3d()
xsetech([0.5,0,0.5,0.5]); plot2d()
xsetech([0.5,0.5,0.5,0.5]); grayplot()
xsetech([0,0.5,0.5,0.5]); histplot()
// back to default values for the sub-window
xsetech([0,0,1,1])
// One plot with changed arect
clf()
xset("default")
xsetech(arect=[0,0,0,0])
x=1:0.1:10;plot2d(x',sin(x)')
clf()
xsetech(arect=[1/8,1/8,1/16,1/4])
x=1:0.1:10;plot2d(x',sin(x)')
clf()
xset("default")
```

See Also

[xgetech](#) , [subplot](#) , [isoview](#) , [square](#)

Authors

J.Ph.C.

Name

xsetm — dialog to set values of the graphics context. **Obsolete function.**

```
xsetm()
```

Description

This function as well as the xset one was strongly linked with the old graphic mode which is no more available. The current graphic is much more flexible with respect to parameter setting (see the set and get functions as well as the graphics_entities help page). It is possible to start a more convenient property editor using ged.

See Also

xset, ged, set, graphics_entities

Authors

J.Ph.C. ENPC

Name

xstring — draw strings

```
xstring(x,y,str,[angle,box])
```

Parameters

x,y

real vectors or scalars, coordinates of the lower-left point of the strings.

str

matrix of strings.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

angle

real vector or scalar, clockwise angle in degree; default is 0.

box

integer vector or scalar, default is 0.

Description

If **x** is a real scalar, it's regarded as a vector with the size of **y** whose elements are set to **x**.

If **y** is a real scalar, it's regarded as a vector with the size of **x** whose elements are set to **y**.

xstring draws **n** strings at location $(x[i], y[i])$ in the current graphic scale.

If **str** contain **n** elements, these **n** elements are the **n** drawn strings.

Otherwise, each row of the matrix stands for a line of text and row elements stand for words separated by a white space.

If **angle** is a real scalar, it's regarded as a vector of size **n** whose elements are set to **angle**.

angle(i) gives the slope in degree used for drawing the strings at location $(x[i], y[i])$.

If **box** is a integer scalar, it's regarded as a vector of size **n** whose elements are set to **box**.

If **box(i)** is 1 and **angle(i)** is 0, a box is drawn around the strings at location $(x[i], y[i])$.

Examples

```
plot2d([0;1],[0;1],0)
xstring(0.5,0.5,["$\\overbrace{Scilab}$" "n'est ";"pas" "$\\underbrace{Matlab}$"
// LaTeX rendering (>= Scilab 5.2)

//Other example
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
          "h" "i" "j" "k" "l" "m" "n" ..
          "o" "p" "q" "r" "s" "t" "u" ..
          "v" "w" "x" "y" "z"];
clf()
plot2d([0;1],[0;2],0)
xstring(0.1,1.8,alphabet) // alphabet
xstring(0.1,1.6,alphabet,0,1) // alphabet in a box
```

```
xstring(0.1,1.4,alphabet,20) // angle
xset("font",1,1)           // use symbol fonts
xstring(0.1,0.1,alphabet)
xset("font",1,3)           // change size font
xstring(0.1,0.3,alphabet)
xset("font",1,24); xstring(0.1,0.6,"a") //big alpha
xset("default")
```

See Also

titlepage , xnumb , xstringb , xstringl , xtitle

Authors

J.Ph.C.

Name

xstringb — draw strings into a box

```
xstringb(x,y,str,w,h,[option])
```

Parameters

x,y,w,h

vector of 4 real scalars defining the box.

str

matrix of strings.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

option

string.

Description

`xstringb` draws the matrix of strings `str` centered inside the rectangle `rect=[x,y,w,h]` (lower-left point, width, height) in user coordinates.

If `option` is given with the value `"fill"`, the character size is computed so as to fill as much as possible in the rectangle.

Enter the command `xstringb()` to see a demo.

Examples

```
str=["Scilab" "is";"$\sqrt{not}$" "elisaB"]; // LaTeX rendering (>= Scilab 5.2)
plot2d(0,0,[-1,1],"010"," ",[0,0,1,1]);

r=[0,0,1,0.5];
xstringb(r(1),r(2),str,r(3),r(4),"fill");
xrect(r(1),r(2)+r(4),r(3),r(4));

r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");

r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
```

See Also

`titlepage` , `xstring` , `xstringl` , `xtitle`

Authors

J.Ph.C.

Name

xstringl — compute a box which surrounds strings

```
rect=xstringl(x,y,str,[fontId,fontSize])
```

Parameters

rect

vector of 4 real scalars defining the box.

x,y

real scalars, coordinates of the lower-left point of the strings.

str

matrix of strings.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

fontId

an integer specifying the font type.

fontSize

an integer specifying the font size.

Description

xstringl returns in `rect=[x,y,w,h]` (upper-left point, width, height) the size of a rectangle in the current graphic scale which would surround the strings `str` drawn at location `x,y` (lower-left point).

The result can be approximative when using the Postscript driver.

Examples

```
plot2d([0;1],[0;1],0)
str=["$\underleftrightharpoonup{Scilab}$" "is";"not" "elisaB"]; // Only valid from Scilab 5.2
r=xstringl(0.5,0.5,str)
xrects([r(1) r(2)+r(4) r(3) r(4)]')
xstring(r(1),r(2),str)

plot2d([0;1],[0;1],0)
str=["Scilab" "n'est ";"pas" "Matlab"];
r2 = xstringl(0.5,0.5,str,2,5)
xrects([r2(1) r2(2)+r2(4) r2(3) r2(4)]')
xstring(r2(1),r2(2),str)

txt2=gce();
txt2.font_size = 5;
txt2.font_style = 2;

plot2d([0;1],[0;1],0)
// Only valid from Scilab 5.2
mathml="<mrow>          <mfrac>          <mrow>          <mi>d</mi>          <mi>y</mi>"
txt2=gce();
txt2.font_size = 5;
a=gca();
```

```
a.margins=[0,0;0,0];
str=["SCILAB", "can write LaTeX :", "$\frac{abc}{xyz}$"; "or", "MathML :", mathml] ;
r2 = xstringl(0.5,0.5,str,2,5)
xrects([r2(1) r2(2)+r2(4) r2(3) r2(4)]')
xstring(r2(1),r2(2),str)
```

See Also

[titlepage](#) , [xstring](#) , [xstringl](#) , [xtitle](#) , [stringbox](#)

Authors

J.Ph.C.

Name

`xtitle` — add titles on a graphics window

```
xtitle(title,[x_label,[y_label,[z_label]]],<opts_args>)
```

Parameters

`title,x_label,y_label,z_label`
matrices of strings.

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

`<opt_args>`
a sequence of statements `key1=value1, key2=value2, ...` where keys may be boxed (see below). In this case, the order has no special meaning.

`boxed`
an integer value. If it is 1, a box is drawn around each title.

Description

`xtitle` add titles on a 2D or 3D plot. `title` is the general title and `x_label`, `y_label` and `z_label` are the titles on the three axis. If the arguments are matrices, each line of the matrices is displayed on a different line.

Enter the command `xtitle()` to see a demo.

Examples

```
// draw a surface
plot3d() ;
// puts the titles
xtitle( 'My surface is blue', 'X axis', 'Y axis', 'Z axis' ) ;
// draw a box around the titles
xtitle( 'My surface is blue', 'X axis', 'Y axis', 'Z axis', boxed = 1 );

// With LaTeX & MathML:
mathml="<mrow>          <mfrac>          <mrow>          <mi>d</mi>          <mi>y</mi>
xtitle( 'My surface is blue', 'X axis', '$Y axis$', mathml );
```

See Also

`titlepage` `label_properties`

Authors

J.Ph.C.

Name

zoom_rect — zoom a selection of the current graphic figure

```
zoom_rect()  
zoom_rect(rect)  
zoom_rect(h)  
zoom_rect(h, rect)
```

Parameters

rect

Vector of size 4 [xmin, ymin, xmax, ymax] give the rectangle to be zoomed.

h

Graphic handle of type Figure or Axes. Specify on which Axes the zoom will apply.

Description

zoom_rect function is used to perform a zoom inside a set of Axes Objects.

The h input argument specifies on which Axes the zoom will apply. If h is a Figure handle then the zoom will apply on its Axes children. If h is a Axes handle then the zoom will only apply to this handle. If h is not specified, then the zoom is performed on the current Figure.

If rect input argument is specified then the zoomed Axes zoom_box property is modified by the argument (see axes_properties). Its bounds along X and Y axis are replaced by rect. If rect is not specified zoom_rect is an interactive zoom. User is required to select a rectangle using the mouse. The new zoom_box property of zoomed axes are then computed by finding the intersections of the rectangle with their axes boxe.

Examples

```
clf()  
x=0:0.01:6*pi;  
plot2d(x, sin(x^2))  
zoom_rect([16, -1, 18, 1])  
//more zoom  
zoom_rect([16, 0, 16.2, 1])  
//back to the original  
unzoom()  
// zooming using axes_properties  
a=gca();  
a.zoom_box=[16, 0, 16.2, 1];  
a.zoom_box=[];  
  
//zooming subplots accordingly  
clf()  
x=0:0.01:6*pi;  
subplot(211)  
plot2d(x, cos(x))  
subplot(212)  
plot2d(x, cos(2*x))  
rect=[3 -2 7 10]; //a rectangle specified in the current axes (last one) coordi  
zoom_rect(rect)
```

```
unzoom()
//set the global underlying axes as current
f=gcf();set('current_axes',f.children($))
rect=[0.4 0 0.6 1] //a rectangle specified in ratio of the window size
zoom_rect(rect)
rect=[0.4 0.2 0.6 0.8]; //a rectangle specified in ratio of the window size
zoom_rect(rect)

// interactive zoom on current figure
zoom_rect();
// or
zoom_rect(gcf());
```

See Also

[unzoom](#) , [axes_properties](#)

Authors

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

Part VII. Graphics : exporting and printing

Name

driver — select a graphics driver

```
driver(driver_name)
current_driver=driver()
```

Parameters

driver_name
string, driver to be selected.

Description

This function is used to select a graphics driver, or with no arguments to get the current graphics driver name. The selected driver can be one of the followings:

"X11"
output to the screen of the computer.

"Pos"
output into Postscript format.

"Rec"
output to the screen of the computer. Same as X11.

"Fig"
output into XFig format.

"GIF"
output into Gif format.

"PPM"
output into PPM format.

Remark

To convert "GIF" or "PPM" files to other image format or for building animation one can use the "convert" program for ImageMagic (<http://www.imagemagick.org/>)

To redirect the graphic output to a GIF file (which will be written in the current directory), you can use the following example:

```
driver('GIF');
xinit('mygiffile.gif');
plot3d();
xend();
driver('X11');
```

For example if one has generated a sequence of Gif files named `img*.gif` it is possible to build an animated Gif file (named `anim.gif`) by

```
convert -delay 10 img*.gif anim.gif
```

Authors

J.Ph.C.

Name

xend — close a graphics session

```
xend()
```

Description

xend is used to close a graphics session. Under the Postscript, Xfig or Gif drivers xend closes the file which was opened by `xinit`.

Examples

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

See Also

`xinit`

Authors

J.Ph.C.

Name

`xinit` — Initialization of a graphics driver

```
xinit(FileName)
xinit()
```

Parameters

`FileName`
string: name of the export file.

Description

For the Postscript, Xfig, Gif or PPM driver, `FileName` must be specified. It is the name of the file where all the graphics operations are recorded.

For screen drivers (X11 or Rec), `xinit` should be called without any argument and opens an empty graphic window.

Examples

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

See Also

`driver` , `xend` , `scf`

Authors

J.Ph.C.

Jean-Baptiste Silvy

Name

xs2bmp — send graphics to a BMP file.

```
xs2bmp(win_num, file_name)
xs2bmp(fig, file_name)
```

Parameters

`win_num`
integer, id of the figure to export.

`fig`
handle of the figure to export.

`file_name`
string, name of the exported file.

Description

xs2bmp exports the display of a graphic window into a BMP file.

Examples

```
scf(0)
plot2d()
//BMP export
xs2bmp(0, 'foo.bmp');
```

See Also

xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

A.C

Name

xs2emf — send graphics to an EMF file (Only for Windows).

```
xs2emf(win_num, file_name [,orientation])
xs2emf(fig, file_name [,orientation])
```

Parameters

win_num

integer, id of the figure to export.

fig

handle of the figure to export.

file_name

string, name of the exported file.

orientation

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

xs2emf exports the display of a graphic window into an EMF file.

The export file is obtained by creating an EPS file and converting it into EMF format using pstoeit.

Examples

```
if MSDOS then
    scf(0);
    plot2d();
    //EMF export
    xs2emf(0, 'foo.emf');
end
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig

Authors

A.C

Name

xs2eps — send graphics to an EPS file.

```
xs2eps(win_num, file_name [,orientation])  
xs2eps(fig, file_name [,orientation])
```

Parameters

win_num

integer, id of the figure to export.

fig

handle of the figure to export.

file_name

string, name of the exported file.

orientation

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

xs2eps exports the display of a graphic window into a complete Encapsulated PostScript file.

Examples

```
scf(0)  
plot2d()  
//EPS export  
xs2eps(0, 'foo.eps')  
xs2eps(gcf(), 'foo.eps')
```

See Also

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

`xs2fig` — send graphics to a FIG file.

```
xs2fig(win_num, file_name [,orientation])
xs2fig(fig, file_name [,orientation])
```

Parameters

`win_num`

integer, id of the figure to export.

`fig`

handle of the figure to export.

`file_name`

string, name of the exported file.

`orientation`

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

`xs2fig` exports the display of a graphic window into an FIG file.

The export file is obtained by creating an EPS file and converting it into FIG format using `pstoedit`.

Examples

```
//simple example
scf(0);
plot2d();
xs2fig(0, 'foo.fig');
xs2fig(gcf(), 'foo.fig')
```

See Also

`xs2bmp`, `xs2gif`, `xs2jpg`, `xs2png`, `xs2ppm`, `xs2eps`, `xs2pdf`, `xs2svg`, `xs2ps`, `xs2emf`

Authors

S.K

Name

xs2gif — send graphics to a GIF file.

```
xs2gif(win_num, file_name)
xs2gif(fig, file_name)
```

Parameters

win_num
integer, id of the figure to export.

fig
handle of the figure to export.

file_name
string, name of the exported file.

Description

xs2gif exports the display of a graphic window into a GIF file.

To convert a sequence of "GIF" files to an animated GIF file one can use the "convert" program for ImageMagic (<http://www.imagemagick.org/>)

For example if one has generated a sequence of Gif files named `img*.gif` it is possible to build an animated Gif file (named `anim.gif`) by

```
convert -delay 10 img*.gif anim.gif
```

Examples

```
scf(0);
plot2d();
//GIF export
xs2gif(0, 'foo.gif');
xs2gif(gcf(), 'foo.gif');
```

See Also

xs2bmp, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2jpg — send graphics to a JPG file.

```
xs2jpg(win_num, file_name)
xs2jpg(fig, file_name)
```

Parameters

win_num
integer, id of the figure to export.

fig
handle of the figure to export.

file_name
string, name of the exported file.

Description

xs2jpg exports the display of a graphic window into a JPG file.

Examples

```
scf(0);
plot2d();
//JPG export
xs2jpg(0, 'foo.jpg');
xs2jpg(gcf(), 'foo.jpg');
```

See Also

xs2bmp, xs2gif, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

S.K

Name

xs2pdf — save graphics to a PDF file.

```
xs2pdf(win_num, file_name [,orientation])  
xs2pdf(fig, file_name [,orientation])
```

Parameters

win_num

integer, id of the figure to export.

fig

handle of the figure to export.

file_name

string, name of the exported file.

If the extension is not provided, it is going to be automatically added.

orientation

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

xs2pdf exports the display of a graphic window into an PDF file.

Examples

```
scf(0);  
plot2d();  
//PDF export  
filename='foo';  
xs2pdf(0,filename);  
xs2pdf(gcf(),filename);
```

See Also

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2png — send graphics to a PNG file.

```
xs2png(win_num, file_name)
xs2png(fig, file_name)
```

Parameters

win_num
integer, id of the figure to export.

fig
handle of the figure to export.

file_name
string, name of the exported file.

Description

xs2png exports the display of a graphic window into a PNG file.

Examples

```
scf(0)
plot2d()
//PNG export
xs2png(0, 'foo.png');
xs2png(gcf(), 'foo.png');
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

S.K

Name

xs2ppm — send graphics to a PPM file.

```
xs2ppm(win_num, file_name)
xs2ppm(fig, file_name)
```

Parameters

win_num
integer, id of the figure to export.

fig
handle of the figure to export.

file_name
string, name of the exported file.

Description

xs2ppm exports the display of a graphic window into a PPM file.

Examples

```
scf(0)
plot2d()
//PPM export
filename='foo.ppm';
xs2ppm(0, filename);
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2png, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2ps — send graphics to a PS file.

```
xs2ps(win_num, file_name [,orientation])  
xs2ps(fig, file_name [,orientation])
```

Parameters

win_num

integer, id of the figure to export.

fig

handle of the figure to export.

file_name

string, name of the exported file.

orientation

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

xs2ps exports the display of a graphic window into a PostScript file.

Note that the generated Postscript file cannot be directly printed since it requires a header file. The function `xs2eps` can be used to directly produce an encapsulated Postscript file with a header.

Examples

```
scf(0);  
plot2d();  
// Postscript export  
filename='foo.ps';  
xs2ps(0,filename);
```

See Also

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2fig, xs2emf

Name

xs2svg — save graphics to a SVG file.

```
xs2svg(win_num, file_name [,orientation])  
xs2svg(fig, file_name [,orientation])
```

Parameters

win_num

integer, id of the figure to export.

fig

handle of the figure to export.

file_name

string, name of the exported file.

orientation

optional character, with possible values 'portrait' or 'landscape'. The default value is 'portrait'.

Description

xs2svg exports the display of a graphic window into a SVG file.

Examples

```
scf(0)  
plot2d()  
//SVG export  
xs2svg(0, 'foo.svg')  
xs2svg(gcf(), 'foo.svg');
```

See Also

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2ps, xs2fig, xs2emf

Part VIII. Boolean

Name

`bool2s` — convert boolean matrix to a zero one matrix.

```
bool2s(x)
```

Parameters

`x`
a boolean vector or a boolean matrix or a constant matrix

Description

If `x` is a boolean matrix, `bool2s(x)` returns the matrix where "true" values are replaced by 1 and "false" value by 0.

If `x` is a "standard" matrix, `bool2s(x)` returns the matrix where non-zero values are replaced by 1.

Examples

```
bool2s([%t %t %f %t])
bool2s([2.3 0 10 -1])
```

See Also

`boolean` , `find`

Name

`find` — find indices of boolean vector or matrix true elements

```
[ii]=find(x [,nmax])  
[i1,i2,..]=find(x [,nmax])
```

Parameters

`x`

may be a boolean vector, a boolean matrix, a boolean hypermatrix, a "standard" matrix or hypermatrix

`nmax`

an integer giving the maximum number of indices to return. The default value is -1 which stands for "all". This option can be used for efficiency, to avoid searching all indices.

`ii, i1, i2, ..`

integer vectors of indices or empty matrices

Description

If `x` is a boolean matrix,

`ii=find(x)` returns the vector of indices `i` for which `x(i)` is "true". If no true element found `find` returns an empty matrix.

`[i1,i2,..]=find(x)` returns vectors of indices `i1` (for rows) and `i2` (for columns),.. such that `x(i1(n),i2(n),..)` is "true". If no true element found `find` returns empty matrices in `i1, i2, ...`

if `x` is a standard matrix or hypermatrix `find(x)` is interpreted as `find(x<>0)`

`find([])` returns `[]`

Examples

```
beers=["Desperados", "Leffe", "Kronenbourg", "Heineken"];  
find(beers=="Leffe") // OK  
find(beers=="1664") // KO  
find(beers=="Foster") // KO  
beers=[beers, "Foster"]  
find(beers=="Foster") // OK
```

```
A=rand(1,20);  
w=find(A<0.4)  
A(w)  
w=find(A>100)
```

```
B=rand(1,20);  
w=find(B<0.4,2) //at most 2 returned values
```

```
H=rand(4,3,5); //an hypermatrix  
[i,j,k]=find(H>0.9)
```

```
H(i(1),j(1),k(1))
```

See Also

[boolean](#) , [extraction](#) , [insertion](#) , [vectorfind](#)

Part IX. CACSD

Name

abcd — state-space matrices

```
[A, B, C, D]=abcd( sl )
```

Parameters

sl
linear system (syslin list) in state-space or transfer form

A,B,C,D
real matrices of appropriate dimensions

Description

returns the A, B, C, D matrices from a linear system Sl.

Utility function. For transfer matrices Sl is converted into state-space form by tf2ss.

The matrices A, B, C, D are the elements 2 to 5 of the syslin list Sl, i.e. $[A, B, C, D] = Sl(2:5)$.

Examples

```
A=diag([1,2,3]);B=[1;1;1];C=[2,2,2];
sys=syslin('c',A,B,C);
sys("A")
sys("C")
[A1,B1,C1,D1]=abcd(sys);
A1
systf=ss2tf(sys);
[a,b,c,d]=abcd(systf)
spec(a)
c*b-C*B
c*a*b-C*A*B
```

See Also

syslin , ssrand

Name

abinv — AB invariant subspace

```
[X,dims,F,U,k,Z]=abinv(Sys,alpha,beta,flag)
```

Parameters

Sys

syslin list containing the matrices [A,B,C,D].

alpha

(optional) real number or vector (possibly complex, location of closed loop poles)

beta

(optional) real number or vector (possibly complex, location of closed loop poles)

flag

(optional) character string 'ge' (default) or 'st' or 'pp'

X

orthogonal matrix of size nx (dim of state space).

dims

integer row vector dims=[dimR,dimVg,dimV,noc,nos] with
dimR<=dimVg<=dimV<=noc<=nos. If flag='st', (resp. 'pp'), dims has 4 (resp. 3)
components.

F

real matrix (state feedback)

k

integer (normal rank of Sys)

Z

non-singular linear system (syslin list)

Description

Output nulling subspace (maximal unobservable subspace) for Sys = linear system defined by a syslin list containing the matrices [A,B,C,D] of Sys. The vector dims=[dimR,dimVg,dimV,noc,nos] gives the dimensions of subspaces defined as columns of X according to partition given below. The dimV first columns of X i.e $V=X(:,1:\text{dimV})$, span the AB-invariant subspace of Sys i.e the unobservable subspace of $(A+B^*F, C+D^*F)$. ($\text{dimV}=n_x$ iff $C^*(-1)(D)=X$).

The dimR first columns of X i.e. $R=X(:,1:\text{dimR})$ spans the controllable part of Sys in V, ($\text{dimR} \leq \text{dimV}$). ($\text{dimR}=0$ for a left invertible system). R is the maximal controllability subspace of Sys in kernel(C).

The dimVg first columns of X spans V_g =maximal AB-stabilizable subspace of Sys. ($\text{dimR} \leq \text{dimVg} \leq \text{dimV}$).

F is a decoupling feedback: for $X=[V,X_2]$ ($X_2=X(:,\text{dimV}+1:n_x)$) one has $X_2'*(A+B^*F)*V=0$ and $(C+D^*F)*V=0$.

The zeros of Sys are given by : $X_0=X(:,\text{dimR}+1:\text{dimV})$; $\text{spec}(X_0'*(A+B^*F)*X_0)$ i.e. there are $\text{dimV}-\text{dimR}$ closed-loop fixed modes.

If the optional parameter **alpha** is given as input, the **dimR** controllable modes of $(A+BF)$ in V are set to **alpha** (or to $[\alpha(1), \alpha(2), \dots]$. (**alpha** can be a vector (real or complex pairs) or a (real) number). Default value **alpha**=-1.

If the optional real parameter **beta** is given as input, the **noc-dimV** controllable modes of $(A+BF)$ "outside" V are set to **beta** (or $[\beta(1), \beta(2), \dots]$). Default value **beta**=-1.

In the X, U bases, the matrices $[X'*(A+B*F)*X, X'*B*U; (C+D*F)*X, D*U]$ are displayed as follows:

```
[A11, *, *, *, *, *] [B11 * ]
[0, A22, *, *, *, *] [0  * ]
[0, 0, A33, *, *, *] [0  * ]
[0, 0, 0, A44, *, *] [0  B42]
[0, 0, 0, 0, A55, *] [0  0  ]
[0, 0, 0, 0, 0, A66] [0  0  ]

[0, 0, 0, *, *, *] [0  D2]
```

where the X -partitioning is defined by **dims** and the U -partitioning is defined by **k**.

A11 is (**dimR** $\times$ **dimR**) and has its eigenvalues set to **alpha(i)**'s. The pair (**A11**, **B11**) is controllable and **B11** has **nu**-**k** columns. **A22** is a stable (**dimVg**-**dimR** $\times$ **dimVg**-**dimR**) matrix. **A33** is an unstable (**dimV**-**dimVg** $\times$ **dimV**-**dimVg**) matrix (see **st_ility**).

A44 is (**noc**-**dimV** $\times$ **noc**-**dimV**) and has its eigenvalues set to **beta(i)**'s. The pair (**A44**, **B42**) is controllable. **A55** is a stable (**nos**-**noc** $\times$ **nos**-**noc**) matrix. **A66** is an unstable (**nx**-**nos** $\times$ **nx**-**nos**) matrix (see **st_ility**).

Z is a column compression of **Sys** and **k** is the normal rank of **Sys** i.e **Sys*****Z** is a column-compressed linear system. **k** is the column dimensions of **B42**, **B52**, **B62** and **D2**. $[B42; B52; B62; D2]$ is full column rank and has rank **k**.

If **flag**='st' is given, a five blocks partition of the matrices is returned and **dims** has four components. If **flag**='pp' is given a four blocks partition is returned. In case **flag**='ge' one has **dims**=[**dimR**, **dimVg**, **dimV**, **dimV**+**nc2**, **dimV**+**ns2**] where **nc2** (resp. **ns2**) is the dimension of the controllable (resp. stabilizable) pair (**A44**, **B42**) (resp. $([A44, *; 0, A55], [B42; 0])$). In case **flag**='st' one has **dims**=[**dimR**, **dimVg**, **dimVg**+**nc**, **dimVg**+**ns**] and in case **flag**='pp' one has **dims**=[**dimR**, **dimR**+**nc**, **dimR**+**ns**]. **nc** (resp. **ns**) is here the dimension of the controllable (resp. stabilizable) subspace of the blocks 3 to 6 (resp. 2 to 6).

This function can be used for the (exact) disturbance decoupling problem.

DDPS:

Find $u = Fx + Rd = [F, R] * [x; d]$ which rejects $Q*d$ and stabilizes the plant:

$$\begin{aligned} \dot{x} &= Ax + Bu + Qd \\ y &= Cx + Du + Td \end{aligned}$$

DDPS has a solution if $\text{Im}(Q)$ is included in $V_g + \text{Im}(B)$ and stabilizability assumption is satisfied.

Let $G = (X(:, \text{dimVg}+1:\$))'$ = left annihilator of V_g i.e. $G*V_g=0$;

$B2=G*B$; $Q2=G*Q$; DDPS solvable iff $[B2; D]*R + [Q2; T] = 0$ has a solution.

The pair F, R is the solution (with F =output of **abinv**).

$\text{Im}(Q2)$ is in $\text{Im}(B2)$ means row-compression of $B2 \Rightarrow$ row-compression of $Q2$

Then $C*((sI - A - B*F)^{(-1)} + D)*(Q + B*R) = 0$ ($\Leftrightarrow G*(Q + B*R) = 0$)

Examples

```

nu=3;ny=4;nx=7;
nrt=2;ngt=3;ng0=3;nvt=5;rk=2;
flag=list('on',nrt,ngt,ng0,nvt,rk);
Sys=ssrand(ny,nu,nx,flag);my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(Sys,my_alpha,my_beta);
[A,B,C,D]=abcd(Sys);dimV=dims(3);dimR=dims(1);
V=X(:,1:dimV);X2=X(:,dimV+1:nx);
X2'*(A+B*F)*V
(C+D*F)*V
X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)
trzeros(Sys)
spec(A+B*F) //nr=2 evals at -1 and noc-dimV=2 evals at -2.
clean(ss2tf(Sys*Z))

// 2nd Example
nx=6;ny=3;nu=2;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B=[1,2;0,3;0,4;0,5;0,0;0,0];
C=[zeros(ny,ny),eye(ny,ny)];D=[0,1;0,2;0,3];
sl=syslin('c',A,B,C,D);//sl=ss2ss(sl,rand(6,6))*rand(2,2);
[A,B,C,D]=abcd(sl); //The matrices of sl.
my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(sl,my_alpha,my_beta);dimVg=dims(2);
clean(X'*(A+B*F)*X)
clean(X'*B*U)
clean((C+D*F)*X)
clean(D*U)
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2; //a valid [Q;T] since
[G*B;D]*R + [G*Q;T] // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed) // Closed loop is zero
spec(closed('A')) //The plant is not stabilizable!
[ns,nc,w,sl1]=st_ility(sl);
[A,B,C,D]=abcd(sl1);A=A(1:ns,1:ns);B=B(1:ns,:);C=C(:,1:ns);
slnew=syslin('c',A,B,C,D); //Now stabilizable
//Fnew=stabil(slnew('A'),slnew('B'),-11);
//slnew('A')=slnew('A')+slnew('B')*Fnew;
//slnew('C')=slnew('C')+slnew('D')*Fnew;
[X,dims,F,U,k,Z]=abinv(slnew,my_alpha,my_beta);dimVg=dims(2);
[A,B,C,D]=abcd(slnew);
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2; //a valid [Q;T] since
[G*B;D]*R + [G*Q;T] // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed) // Closed loop is zero
spec(closed('A'))

```

See Also

cainv , st_ility , ssrand , ss2ss , ddp

Authors

F.D.

Name

arhnk — Hankel norm approximant

```
[slm]=arhnk(sl,ord,[tol])
```

Parameters

sl
linear system (syslin list)

ord
integer, order of the approximant

tol
threshold for rank determination in `equil1`

Description

computes `slm`, the optimal Hankel norm approximant of the stable continuous-time linear system `sl` with matrices `[A,B,C,D]`.

Examples

```
A=diag([-1,-2,-3,-4,-5]);B=rand(5,1);C=rand(1,5);  
sl=syslin('c',A,B,C);  
slapprox=arhnk(sl,2);  
[nk,w]=hankelsv(sl);nk  
[nkred,wred]=hankelsv(slapprox);nkred
```

See Also

`equil` , `equil1` , `hankelsv`

Name

arl2 — SISO model realization by L2 transfer approximation

```
h=arl2(y,den0,n [,imp])
h=arl2(y,den0,n [,imp],'all')
[den,num,err]=arl2(y,den0,n [,imp])
[den,num,err]=arl2(y,den0,n [,imp],'all')
```

Parameters

- y**
real vector or polynomial in z^{-1} , it contains the coefficients of the Fourier's series of the rational system to approximate (the impulse response)
- den0**
a polynomial which gives an initial guess of the solution, it may be `poly(1, 'z', 'c')`
- n**
integer, the degree of approximating transfer function (degree of den)
- imp**
integer in $(0, 1, 2)$ (verbose mode)
- h**
transfer function num/den or transfer matrix (column vector) when flag 'all' is given.
- den**
polynomial or vector of polynomials, contains the denominator(s) of the solution(s)
- num**
polynomial or vector of polynomials, contains the numerator(s) of the solution(s)
- err**
real constant or vector, the l2-error achieved for each solutions

Description

`[den,num,err]=arl2(y,den0,n [,imp])` finds a pair of polynomials **num** and **den** such that the transfer function **num/den** is stable and its impulse response approximates (with a minimal l2 norm) the vector **y** assumed to be completed by an infinite number of zeros.

If $y(z) = y(1)(1/z) + y(2)(1/z^2) + \dots + y(ny)(1/z^{ny})$

then l2-norm of **num/den** - **y(z)** is **err**.

n is the degree of the polynomial **den**.

The **num/den** transfer function is a L2 approximant of the Fourier's series of the rational system.

Various intermediate results are printed according to **imp**.

`[den,num,err]=arl2(y,den0,n [,imp],'all')` returns in the vectors of polynomials **num** and **den** a set of local optimums for the problem. The solutions are sorted with increasing errors **err**. In this case **den0** is already assumed to be `poly(1, 'z', 'c')`

Examples

```
v=ones(1,20);
clf();
plot2d1('enn',0,[v';zeros(80,1)],2,'051',' ',[1,-0.5,100,1.5])

[d,n,e]=ar12(v,poly(1,'z','c'),1)
plot2d1('enn',0,ldiv(n,d,100),2,'000')
[d,n,e]=ar12(v,d,3)
plot2d1('enn',0,ldiv(n,d,100),3,'000')
[d,n,e]=ar12(v,d,8)
plot2d1('enn',0,ldiv(n,d,100),5,'000')

[d,n,e]=ar12(v,poly(1,'z','c'),4,'all')
plot2d1('enn',0,ldiv(n(1),d(1),100),10,'000')
```

See Also

ldiv , imrep2ss , time_id , armax , freq2tf

Name

arma — Scilab arma library

Description

Armax processes can be coded with Scilab tlist of type 'ar'. `armac` is used to build Armax scilab object. An 'ar' tlist contains the fields ['a', 'b', 'd', 'ny', 'nu', 'sig'].

`armac`

this function creates a Scilab tlist which code an Armax process $A(z^{-1})y = B(z^{-1})u + D(z^{-1})\text{sig}(t)$

```
-->ar=armac([1,2],[3,4],1,1,1,sig);  
  
-->ar('a')  
ans =  
  
! 1. 2. !  
-->ar('sig')  
ans =  
  
1.
```

`armap(ar [,out])`

Display the armax equation associated with `ar`

`armap_p(ar [,out])`

Display the armax equation associated with `ar` using polynomial matrix display.

`[A,B,D]=armap2p(ar)`

extract polynomial matrices from ar representation

`armax`

is used to identify the coefficients of a n-dimensional ARX process $A(z^{-1})y = B(z^{-1})u + \text{sig}(t)$

`armax1`

`armax1` is used to identify the coefficients of a 1-dimensional ARX process $A(z^{-1})y = B(z^{-1})u + D(z^{-1})\text{sig}(t)$

`arsimul`

armax trajectory simulation.

`narsimul`

armax simulation (using rtitr)

`odedi`

Simple tests of ode and arsimul. Tests the option 'discret' of ode

`prbs_a`

pseudo random binary sequences generation

`reglin`

Linear regression

Example

```

// Example extracted from the demo arma3.dem.sce in the cacsd module
// Spectral power estimation
// ( form Sawaragi et all)
m = 18;
a = [1, -1.3136, 1.4401, -1.0919, +0.83527];
b = [0.0, 0.13137, 0.023543, 0.10775, 0.03516];
u = rand(1,1000, 'n');
z = arsimul(a,b, [0],0,u);

//----Using macro mese
[sm,fr]=mese(z,m);

//----The theoretical result

function gx=gxx(z,a,b)
    w = exp(-%i*2*pi*z*(0:4))'
    gx = abs(b*w)^2/(abs(a*w)^2);
endfunction

res=[];
for x=fr
    res=[ res, gxx(x,a,b)];
end

//----using armax estimation of order (4,4)
// it's a bit tricky because we are not supposed to know the order

[arc,la,lb,sig,resid]=armax(4,4,z,u);
res1=[];
for x=fr
    res1=[ res1, gxx(x,la(1),lb(1))];
end

//-- visualization of the results
plot2d([fr;fr;fr]',[20*log10(sm/sm(1));20*log10(res/res(1));20*log10(res1/res1(1))]);
legend(["Using macro mese";"Theoretical value";"Arma identification"])
xtitle("Spectral power","frequency","spectral estimate")

```

Authors

J.P.C;;

Name

arma2p — extract polynomial matrices from ar representation

```
[A, B, D]=arma2p(ar)
```

Parameters

A,B,D

three polynomial matrices

ar

Scilab 'ar' tlist for arma storage (see `armac`).

Description

this function extract polynomial matrices (A, B, D) from an armax description.

Examples

```
a=[1, -2.851, 2.717, -0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

See Also

arma , armax , armax1 , arsimul , armac

Name

armac — Scilab description of an armax process

```
[ar]=armac(a,b,d,ny,nu,sig)
```

Parameters

`a`=[Id,a1,...,a_r]
is a matrix of size (ny,r*ny)

`b`=[b0,...,b_s]
is a matrix of size (ny,(s+1)*nu)

`d`=[Id,d1,...,d_p]
is a matrix of size (ny,p*ny);

`ny`
dimension of the output y

`nu`
dimension of the output u

`sig`
a matrix of size (ny,ny)

Description

This function creates a description as a tlist of an ARMAX process

`ar` is defined by

```
ar=tlist(['ar','a','b','d','ny','nu','sig'],a,b,d,ny,nu,sig);
```

and thus the coefficients of `ar` can be retrieved by e.g. `ar('a')` .

Examples

```
a=[1, -2.851, 2.717, -0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

See Also

`arma` , `armax` , `armax1` , `arsimul` , `arma2p` , `tlist`

Name

armax — armax identification

```
[arc, la, lb, sig, resid]=armax(r, s, y, u, [b0f, prf])
```

Parameters

y

output process y(ny,n); (ny: dimension of y , n : sample size)

u

input process u(nu,n); (nu: dimension of u , n : sample size)

r and s

auto-regression orders $r \geq 0$ et $s \geq -1$

b0f

optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if b0f=1 the b0 is supposed to be zero and is not identified

prf

optional parameter for display control. If prf =1, the default value, a display of the identified Arma is given.

arc

a Scilab arma object (see armac)

la

is the list(a,a+eta,a-eta) (la = a in dimension 1) ; where eta is the estimated standard deviation. , a=[Id,a1,a2,...,ar] where each ai is a matrix of size (ny,ny)

lb

is the list(b,b+etb,b-etb) (lb =b in dimension 1) ; where etb is the estimated standard deviation. b=[b0,...,b_s] where each bi is a matrix of size (nu,nu)

sig

is the estimated standard deviation of the noise and resid=[sig*e(t0),....] (

Description

armax is used to identify the coefficients of a n-dimensional ARX process

$$A(z^{-1})y = B(z^{-1})u + \text{sig} \cdot e(t)$$

where e(t) is a n-dimensional white noise with variance I. sig an nxn matrix and A(z) and B(z):

$$\begin{aligned} A(z) &= 1 + a_1 z + \dots + a_r z^r; \quad (r=0 \Rightarrow A(z)=1) \\ B(z) &= b_0 + b_1 z + \dots + b_s z^s \quad (s=-1 \Rightarrow B(z)=0) \end{aligned}$$

for the method see Eykhoff in trends and progress in system identification, page 96. with $z(t) = [y(t-1), \dots, y(t-r), u(t), \dots, u(t-s)]$ and $\text{coef} = [-a_1, \dots, -a_r, b_0, \dots, b_s]$ we can write $y(t) = \text{coef}' z(t) + \text{sig} \cdot e(t)$ and the algorithm minimises $\sum_{t=1}^N ([y(t) - \text{coef}' z(t)]^2)$ where $t_0 = \max(\max(r,s)+1, 1)$.

Examples

```
//-Ex1- Arma model :  $y(t) = 0.2*u(t-1)+0.01*e(t-1)$ 
ny=1,nu=1,sig=0.01;
Arma=armac(1,[0,0.2],[0,1],ny,nu,sig) //defining the above arma model
u=rand(1,1000,'normal'); //a random input sequence u
y=arsimul(Arma,u); //simulation of a y output sequence associated with u.
Armaest=armax(0,1,y,u); //Identified model given u and y.
Acoeff=Armaest('a'); //Coefficients of the polynomial A(x)
Bcoeff=Armaest('b') //Coefficients of the polynomial B(x)
Dcoeff=Armaest('d'); //Coefficients of the polynomial D(x)
[Ax,Bx,Dx]=arma2p(Armaest) //Results in polynomial form.

//-Ex2- Arma1:  $y_t - 0.8*y_{t-1} + 0.2*y_{t-2} = sig*e(t)$ 
ny=1,nu=1;sig=0.001;
// First step: simulation the Arma1 model, for that we define
// Arma2:  $y_t - 0.8*y_{t-1} + 0.2*y_{t-2} = sig*u(t)$ 
// with normal deviates for u(t).
Arma2=armac([1,-0.8,0.2],sig,0,ny,nu,0);
//Definition of the Arma2 arma model (a model with B=sig and without noise!)
u=rand(1,10000,'normal'); // An input sequence for Arma2
y=arsimul(Arma2,u); // y = output of Arma2 with input u
// can be seen as output of Arma1.
// Second step: identification. We look for an Arma model
//  $y(t) + a1*y(t-1) + a2*y(t-2) = sig*e(t)$ 
Arma1est=armax(2,-1,y,[]);
[A,B,D]=arma2p(Arma1est)
```

See Also

imrep2ss , time_id , arl2 , armax , freq2tf

Authors

J-Ph. Chancelier.

Name

armax1 — armax identification

```
[arc,resid]=armax1(r,s,q,y,u [,b0f])
```

Parameters

y

output signal

u

input signal

r,s,q

auto regression orders with $r \geq 0$, $s \geq -1$.

b0f

optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified.
if b0f=1 the b0 is supposed to be zero and is not identified

arc

is tlist with type "ar" and fields a, b, d, ny, nu, sig

a

is the vector $[1, a_1, \dots, a_r]$

b

is the vector $[b_0, \dots, b_s]$

d

is the vector $[1, d_1, \dots, d_q]$

sig

resid=[sig*echap(1),...];

Description

armax1 is used to identify the coefficients of a 1-dimensional ARX process:

```
A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)
e(t) is a 1-dimensional white noise with variance 1.
A(z)= 1+a1*z+...+a_r*z^r; ( r=0 => A(z)=1)
B(z)= b0+b1*z+...+b_s z^s ( s=-1 => B(z)=0)
D(z)= 1+d1*z+...+d_q*z^q ( q=0 => D(z)=1)
```

for the method, see Eykhoff in trends and progress in system identification) page 96. with

```
z(t)=[y(t-1),...,y(t-r),u(t),...,
      u(t-s),e(t-1),...,e(t-q)]
```

and

```
coef= [-a1,...,-a_r,b0,...,b_s,d1,...,d_q]'
```

```
y(t)= coef'* z(t) + sig*e(t).
```

a sequential version of the AR estimation where $e(t-i)$ is replaced by an estimated value is used (RLLS). With $q=0$ this method is exactly a sequential version of `armax`

Important notice

In Scilab versions up to 4.1.2 the returned value in `arc.sig` is the square of `sig` square. To be conform with the help, the display of arma models and the `armax` function, starting from Scilab-5.0 version the returned `arc.sig` is `sig`.

Authors

J.-Ph.C; ;

Name

arsimul — armax simulation

```
[z]=arsimul(a,b,d,sig,u,[up,yp,ep])  
[z]=arsimul(ar,u,[up,yp,ep])
```

Parameters

ar
an armax process. See armac.

a
is the matrix $[Id, a_1, \dots, a_r]$ of dimension $(n,(r+1)*n)$

b
is the matrix $[b_0, \dots, b_s]$ of dimension $(n,(s+1)*m)$

d
is the matrix $[Id, d_1, \dots, d_t]$ of dimension $(n,(t+1)*n)$

u
is a matrix (m,N) , which gives the entry $u(:,j)=u_j$

sig
is a (n,n) matrix $e_{\{k\}}$ is an n -dimensional Gaussian process with variance I

up, yp
optional parameter which describe the past. $up=[u_0, u_{-1}, \dots, u_{s-1}]$;
 $yp=[y_0, y_{-1}, \dots, y_{r-1}]$; $ep=[e_0, e_{-1}, \dots, e_{r-1}]$; if they are omitted, the past value are supposed to be zero

z
 $z=[y(1), \dots, y(N)]$

Description

simulation of an n -dimensional armax process $A(z^{-1}) \quad z(k)= \quad B(z^{-1})u(k) \quad +$
 $D(z^{-1}) * sig * e(k)$

```
A(z)= Id+a1*z+...+a_r*z^r; ( r=0 => A(z)=Id)  
B(z)= b0+b1*z+...+b_s z^s; ( s=-1 => B(z)=[])  
D(z)= Id+d1*z+...+d_t z^t; ( t=0 => D(z)=Id)
```

z et e are in R^n et u in R^m

Method

a state-space representation is constructed and ode with the option "discr" is used to compute z

Authors

J-Ph.C.

Name

augment — augmented plant

```
[P,r]=augment(G)
[P,r]=augment(G,flag1)
[P,r]=augment(G,flag1,flag2)
```

Parameters

- G**
linear system (syslin list), the nominal plant
- flag1**
one of the following (upper case) character string: 'S' , 'R' , 'T' , 'SR' , 'ST' , 'RT' , 'SRT'
- flag2**
one of the following character string: 'o' (stands for 'output', this is the default value) or 'i' (stands for 'input').
- P**
linear system (syslin list), the ``augmented" plant
- r**
1x2 row vector, dimension of $P_{22} = G$

Description

If `flag1='SRT'` (default value), returns the "full" augmented plant

```
P = [ I | -G]  -->'S'
     [ 0 |  I]  -->'R'
     [ 0 |  G]  -->'T'
     [-----]
     [ I | -G]
```

'S' , 'R' , 'T' refer to the first three (block) rows of **P** respectively.

If one of these letters is absent in `flag1`, the corresponding row in **P** is missing.

If **G** is given in state-space form, the returned **P** is minimal. **P** is calculated by: $[I, 0, 0; 0, I, 0; -I, 0, I; I, 0, 0] * [I, -G; 0, I; I, 0]$.

The augmented plant associated with input sensitivity functions, namely

```
P = [ I | -I]  -->'S' (input sensitivity)
     [ G | -G]  -->'R' (G*input sensitivity)
     [ 0 |  I]  -->'T' (K*G*input sensitivity)
     [-----]
     [ G | -G]
```

is obtained by the command `[P,r]=augment(G,flag,'i')`. For state-space **G**, this **P** is calculated by: $[I, -I; 0, 0; 0, I; 0, 0] + [0; I; 0; I] * G * [I, -I]$ and is thus generically minimal.

Note that weighting functions can be introduced by left-multiplying P by a diagonal system of appropriate dimension, e.g., $P = \text{sysdiag}(W1, W2, W3, \text{eye}(G)) * P$.

Sensitivity functions can be calculated by `lft`. One has:

For output sensitivity functions `[P,r]=augment(P,'SRT');`
`lft(P,r,K)=[inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];`

For input sensitivity functions `[P,r]=augment(P,'SRT','i');`
`lft(P,r,K)=[inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+G*K)];`

Examples

```
G=ssrand(2,3,2); //Plant
K=ssrand(3,2,2); //Compensator
[P,r]=augment(G,'T');
T=lft(P,r,K); //Complementary sensitivity function
Ktf=ss2tf(K);Gtf=ss2tf(G);
Ttf=ss2tf(T);T11=Ttf(1,1);
Oloop=Gtf*Ktf;
Tn=Oloop*inv(eye(Oloop)+Oloop);
clean(T11-Tn(1,1));
//
[Pi,r]=augment(G,'T','i');
T1=lft(Pi,r,K);T1tf=ss2tf(T1); //Input Complementary sensitivity function
Oloop=Ktf*Gtf;
T1n=Oloop*inv(eye(Oloop)+Oloop);
clean(T1tf(1,1)-T1n(1,1))
```

See Also

`lft`, `sensi`

Name

balreal — balanced realization

```
[slb [,U] ] = balreal(sl)
```

Parameters

sl,slb
linear systems (`syslin` lists)

Description

Balanced realization of linear system $sl=[A, B, C, D]$. sl can be a continuous-time or discrete-time state-space system. sl is assumed stable.

```
slb=[inv(U)*A*U ,inv(U)*B , C*U , D]
```

is the balanced realization.

slb is returned as a `syslin` list.

Examples

```
A=diag([-1, -2, -3, -4, -5]);B=rand(5,2);C=rand(1,5);  
sl=syslin('c',A,B,C);  
[slb,U]=balreal(sl);  
wc=clean(ctr_gram(slb))  
w0=clean(obs_gram(slb))
```

See Also

`ctr_gram` , `obs_gram` , `hankelsv` , `equil` , `equil1`

Name

bilin — general bilinear transform

```
[sl1]=bilin(sl,v)
```

Parameters

sl,sl1
linear systems (syslin lists)

v
real vector with 4 entries (v=[a, b, c, d])

Description

Given a linear system in state space form, sl=syslin(dom,A,B,C,D) (syslin list), sl1=bilin(sl,v) returns in sl1 a linear system with matrices [A1,B1,C1,D1] such that the transfer function $H_1(s)=C_1 \operatorname{inv}(s \operatorname{eye}() - A_1) B_1 + D_1$ is obtained from $H(z)=C \operatorname{inv}(z \operatorname{eye}() - A) B + D$ by replacing z by $z=(a*s+b)/(c*s+d)$. One has $w=\operatorname{bilin}(\operatorname{bilin}(w,[a,b,c,d]),[d,-b,-c,a])$

Examples

```
s=poly(0,'s');z=poly(0,'z');  
w=ssrand(1,1,3);  
wtf=ss2tf(w);v=[2,3,-1,4];a=v(1);b=v(2);c=v(3);d=v(4);  
[horner(wtf,(a*z+b)/(c*z+d)),ss2tf(bilin(w,[a,b,c,d]))]  
clean(ss2tf(bilin(bilin(w,[a,b,c,d]),[d,-b,-c,a]))-wtf)
```

See Also

horner , cls2dls

Name

black — Black's diagram (Nichols chart)

```
black( sl,[fmin,fmax] [,step] [,comments] )
black( sl,frq [,comments] )
black(frq,db,phi [,comments])
black(frq,repf [,comments])
```

Parameters

sl
list (linear system `syslin`)

fmin,fmax
real scalars (frequency bounds)

frq
row vector or matrix (frequencies)

db,phi
row vectors or matrices (modulus, phase)

repf
row vectors or matrices (complex frequency response)

step
real

comments
string

Description

Black's diagram (Nichols'chart) for a linear system `sl`. `sl` can be a continuous-time or discrete-time SIMO system (see `syslin`). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds `fmin,fmax` (in Hz) or by a row-vector (or a matrix for multi-output) `frq`.

`step` is the (logarithmic) discretization step. (see `calfrq` for the choice of default value).

`comments` is a vector of character strings (captions).

`db, phi` are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

`repf` matrix of complex numbers. One row for each response.

To plot the grid of iso-gain and iso-phase of $y/(1+y)$ use `chart()`.

Default values for `fmin` and `fmax` are `1.d-3, 1.d+3` if `sl` is continuous-time or `1.d-3, 0.5/sl.dt` (nyquist frequency) if `sl` is discrete-time.

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
```

```
clf();black(h,0.01,100);  
chart(list(1,0));  
  
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))  
clf()  
black([h1;h],0.01,100,['h1';'h'])  
chart(list(1,0));
```

See Also

bode, nyquist, chart, freq, repfreq, calfrq, phasemag

Name

bode — Bode plot

```
bode(sl,[fmin,fmax] [,step] [,comments] )
bode(sl,frq [,comments] )
bode(frq,db,phi [,comments])
bode(frq, repf [,comments])
```

Parameters

sl
syslin list (SISO or SIMO linear system) in continuous or discrete time.

fmin,fmax
real (frequency bounds (in Hz))

step
real (logarithmic step.)

comments
vector of character strings (captions).

frq
row vector or matrix (frequencies (in Hz)) (one row for each SISO subsystem).

db
row vector or matrix (magnitudes (in Db)). (one row for each SISO subsystem).

phi
row vector or matrix (phases (in degree)) (one row for each SISO subsystem).

repf
row vector or matrix of complex numbers (complex frequency response).

Description

Bode plot, i.e magnitude and phase of the frequency response of **sl**.

sl can be a continuous-time or discrete-time SIMO system (see **syslin**). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds **fmin**, **fmax** (in Hz) or by a row-vector (or a matrix for multi-output) **frq**.

step is the (logarithmic) discretization step. (see **calfrq** for the choice of default value).

comments is a vector of character strings (captions).

db, **phi** are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

repf matrix of complex numbers. One row for each response.

Default values for **fmin** and **fmax** are 10^{-3} , 10^3 if **sl** is continuous-time or 10^{-3} , $0.5/sl.dt$ (nyquist frequency) if **sl** is discrete-time. Automatic discretization of frequencies is made by **calfrq**.

Examples

```
s=poly(0, 's')
h=syslin('c', (s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
tit='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
bode(h,0.01,100,tit);
h1=h*syslin('c', (s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf()
bode([h1;h],0.01,100,['h1';'h'])
```

See Also

black, nyquist, gainplot, repfreq, g_margin, p_margin, calfrq, phasemag

Name

bstap — hankel approximant

```
[Q]=bstap(SL)
```

Parameters

- sl
linear system (`syslin` list) assumed continuous-time and anti-stable.
- Q
best stable approximation of SL (`syslin` list).

Description

Computes the best approximant Q of the linear system SL

where

$\|T\|$

is the H-infinity norm of the Hankel operator associated with SL.

See Also

syslin

Name

cainv — Dual of abinv

```
[X,dims,J,Y,k,Z]=cainv(Sl,alfa,beta,flag)
```

Parameters

- Sl**
syslin list containing the matrices $[A, B, C, D]$.
- alfa**
real number or vector (possibly complex, location of closed loop poles)
- beta**
real number or vector (possibly complex, location of closed loop poles)
- flag**
(optional) character string 'ge' (default) or 'st' or 'pp'
- X**
orthogonal matrix of size nx (dim of state space).
- dims**
integer row vector $\text{dims}=[\text{nd1}, \text{nu1}, \text{dimS}, \text{dimSg}, \text{dimN}]$ (5 entries, nondecreasing order). If $\text{flag}='st'$, (resp. 'pp'), dims has 4 (resp. 3) components.
- J**
real matrix (output injection)
- Y**
orthogonal matrix of size ny (dim of output space).
- k**
integer (normal rank of Sl)
- Z**
non-singular linear system (syslin list)

Description

cainv finds a bases (X, Y) (of state space and output space resp.) and output injection matrix J such that the matrices of Sl in bases (X,Y) are displayed as:

$$\begin{aligned} X' \cdot (A+J \cdot C) \cdot X &= \begin{bmatrix} A_{11} & * & * & * & * & * \\ 0 & A_{22} & * & * & * & * \\ 0 & 0 & A_{33} & * & * & * \\ 0 & 0 & 0 & A_{44} & * & * \\ 0 & 0 & 0 & 0 & A_{55} & * \\ 0 & 0 & 0 & 0 & 0 & A_{66} \end{bmatrix} & X' \cdot (B+J \cdot D) &= \begin{bmatrix} * \\ * \\ * \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ Y \cdot C \cdot X &= \begin{bmatrix} 0 & 0 & C_{13} & * & * & * \\ 0 & 0 & 0 & 0 & 0 & C_{26} \end{bmatrix} & Y \cdot D &= \begin{bmatrix} * \\ 0 \end{bmatrix} \end{aligned}$$

The partition of X is defined by the vector $\text{dims}=[\text{nd1}, \text{nu1}, \text{dimS}, \text{dimSg}, \text{dimN}]$ and the partition of Y is determined by k.

Eigenvalues of A_{11} ($nd1 \times nd1$) are unstable. Eigenvalues of A_{22} ($nu1-nd1 \times nu1-nd1$) are stable.

The pair (A_{33}, C_{13}) ($dimS-nu1 \times dimS-nu1, k \times dimS-nu1$) is observable, and eigenvalues of A_{33} are set to α .

Matrix A_{44} ($dimSg-dimS \times dimSg-dimS$) is unstable. Matrix A_{55} ($dimN-dimSg, dimN-dimSg$) is stable

The pair (A_{66}, C_{26}) ($nx-dimN \times nx-dimN$) is observable, and eigenvalues of A_{66} set to β .

The $dimS$ first columns of X span S = smallest (C,A) invariant subspace which contains $Im(B)$, $dimSg$ first columns of X span Sg the maximal "complementary detectability subspace" of S

The $dimN$ first columns of X span the maximal "complementary observability subspace" of S . ($dimS=0$ if $B(ker(D))=0$).

If `flag='st'` is given, a five blocks partition of the matrices is returned and `dims` has four components. If `flag='pp'` is given a four blocks partition is returned (see `abinv`).

This function can be used to calculate an unknown input observer:

```
// DDEP: dot(x)=A x + Bu + Gd
//          y= Cx    (observation)
//          z= Hx    (z=variable to be estimated, d=disturbance)
// Find: dot(w) = Fw + Ey + Ru such that
//          zhat = Mw + Ny
//          z-Hx goes to zero at infinity
// Solution exists iff Ker H contains Sg(A,C,G) inter KerC (assuming detectability)
// i.e. H is such that:
// For any W which makes a column compression of [Xp(1:dimSg,:);C]
// with Xp=X' and [X,dims,J,Y,k,Z]=cainv(syslin('c',A,G,C));
// [Xp(1:dimSg,:);C]*W = [0 | *] one has
// H*W = [0 | *] (with at least as many zero columns as above).
```

See Also

`abinv` , `dt_ility` , `ui_observer`

Name

calfrq — frequency response discretization

```
[frq,bnds,split]=calfrq(h,fmin,fmax)
```

Parameters

h

Linear system in state space or transfer representation (see `syslin`)

fmin,fmax

real scalars (min and max frequencies in Hz)

frq

row vector (discretization of the frequency interval)

bnds

vector `[Rmin Rmax Imin Imax]` where `Rmin` and `Rmax` are the lower and upper bounds of the frequency response real part, `Imin` and `Imax` are the lower and upper bounds of the frequency response imaginary part,

split

vector of frq splitting points indexes

Description

frequency response discretization; `frq` is the discretization of `[fmin,fmax]` such that the peaks in the frequency response are well represented.

Singularities are located between `frq(split(k)-1)` and `frq(split(k))` for `k>1`.

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
[f1,bnds,spl]=calfrq(h1,0.01,1000);
rf=repfreq(h1,f1);
plot2d(real(rf)',imag(rf)')
```

See Also

bode, black, nyquist, freq, repfreq, logspace

Name

canon — canonical controllable form

```
[Ac, Bc, U, ind]=canon(A, B)
```

Parameters

Ac, Bc

canonical form

U

current basis (square nonsingular matrix)

ind

vector of integers, controllability indices

Description

gives the canonical controllable form of the pair (A, B) .

$Ac = \text{inv}(U) * A * U$, $Bc = \text{inv}(U) * B$

The vector `ind` is made of the `epsilon_i`'s indices of the pencil $[sI - A, B]$ (decreasing order). For example with `ind=[3, 2]`, `Ac` and `Bc` are as follows:

```
Ac=      [* , * , * , * , * ]      [* ]
          [1, 0, 0, 0, 0]            [0]
          [0, 1, 0, 0, 0]            [0]
          [* , * , * , * , * ]      [* ]
          [0, 0, 0, 1, 0]            [0]

Bc=      [* ]
          [0]
          [0]
          [* ]
          [0]
```

If (A, B) is controllable, by an appropriate choice of F the $*$ entries of $Ac + Bc * F$ can be arbitrarily set to desired values (pole placement).

Examples

```
A=[1, 2, 3, 4, 5;
   1, 0, 0, 0, 0;
   0, 1, 0, 0, 0;
   6, 7, 8, 9, 0;
   0, 0, 0, 1, 0];
B=[1, 2;
   0, 0;
   0, 0;
   2, 1;
   0, 0];
X=rand(5,5); A=X*A*inv(X); B=X*B; //Controllable pair
[Ac, Bc, U, ind]=canon(A, B); //Two indices --> ind=[3,2];
index=1; for k=1:size(ind, '*')-1, index=[index, 1+sum(ind(1:k))]; end
Acstar=Ac(index, :); Bcstar=Bc(index, :);
s=poly(0, 's');
p1=s^3+2*s^2-5*s+3; p2=(s-5)*(s-3);
//p1 and p2 are desired closed-loop polynomials with degrees 3,2
c1=coeff(p1); c1=c1($-1:-1:1); c2=coeff(p2); c2=c2($-1:-1:1);
```

```
Acstardesired=[-c1,0,0;0,0,0,-c2];
//Acstardesired(index,:) is companion matrix with char. pol=p1*p2
F=Bcstar\ (Acstardesired-Acstar); //Feedbak gain
Ac+Bc*F // Companion form
spec(A+B*F/U) // F/U is the gain matrix in original basis.
```

See Also

obsv_mat , cont_mat , ctr_gram , contrss , ppol , contr , stabil

Authors

F.D.

Name

ccontrg — central H-infinity controller

```
[K]=ccontrg(P,r,gamma);
```

Parameters

P
syslin list (linear system in state-space representation)

r
1x2 row vector, dimension of the 2,2 part of P

gamma
real number

Description

returns a realization K of the central controller for the general standard problem in state-space form.

Note that gamma must be > gopt (ouput of gamitg)

P contains the parameters of plant realization (A,B,C,D) (syslin list) with

```
B = ( B1 , B2 ) ,      C= ( C1 ) ,      D = ( D11 D12 )
                      ( C2 )      ( D21 D22 )
```

r(1) and r(2) are the dimensions of D22 (rows x columns)

See Also

gamitg, h_inf

Authors

P. Gahinet (INRIA);

Name

chart — Nichols chart

```
chart([flags])
chart(gain [, flags])
chart(gain, phase [, flags])
```

Parameters

gain
real vector (gains (in DB))

phase
real vector (phases (in degree))

flags
a list of at most 4 flags list(sup [,leg [,cm [,cphi]]])

sup
1 indicates superposition on the previous plot 0 no superposition is done

leg
1 indicates that legends are drawn, 0: no legends

cm
color index for gain curves

cphi
color index for phase curves

Description

plot the Nichols'chart: iso-gain and iso-phase contour of $y/(1+y)$ in phase/gain plane

`chart` may be used in conjunction with `black`.

The default values for `gain` and `phase` are respectively :

```
[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]
```

```
[-(1:10) , -(20:10:160)]
```

Examples

```
s=poly(0, 's')
h=syslin('c', (s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
black(h,0.01,100)
chart(list(1,0,2,3));

clf()
h1=h*syslin('c', (s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
black([h1;h],0.01,100,['h1';'h'])
set(gca(),'data_bounds',[-180 -30;180 30]) //enlarge the frame
chart(list(1,0));
```

See Also

nyquist , black

Name

cls2dls — bilinear transform

```
[sl1]=cls2dls(sl,T [,fp])
```

Parameters

sl,sl1
linear systems (syslin lists)

T
real number, the sampling period

fp
prewarping frequency in hertz

Description

given $sl=[A, B, C, D]$ (syslin list), a continuous time system cls2dls returns the sampled system obtained by the bilinear transform $s=(2/T)*(z-1)/(z+1)$.

Examples

```
s=poly(0,'s');z=poly(0,'z');
sl=syslin('c',(s+1)/(s^2-5*s+2)); //Continuous-time system in transfer form
slss=tf2ss(sl); //Now in state-space form
sl1=cls2dls(slss,0.2); //sl1= output of cls2dls
sl1t=ss2tf(sl1) // Converts in transfer form
sl2=horner(sl,(2/0.2)*(z-1)/(z+1)) //Compare sl2 and sl1
```

See Also

horner

Name

colinout — inner-outer factorization

```
[Inn,X,Gbar]=colinout(G)
```

Parameters

G
linear system (syslin list) $[A, B, C, D]$

Inn
inner factor (syslin list)

Gbar
outer factor (syslin list)

X
row-compressor of G (syslin list)

Description

Inner-outer factorization (and column compression) of $(l \times p)$ $G = [A, B, C, D]$ with $l \leq p$.

G is assumed to be fat ($l \leq p$) without zero on the imaginary axis and with a D matrix which is full row rank.

G must also be stable for having Gbar stable.

Dual of rowinout.

See Also

syslin , rowinout

Name

colregul — removing poles and zeros at infinity

```
[Stmp,ws]=colregul(Sl,alfa,beta)
```

Parameters

Sl,Stmp
syslin lists

alfa,beta
reals (new pole and zero positions)

Description

computes a prefilter **WS** such that **Stmp=Sl*WS** is proper and with full rank **D** matrix.

Poles at infinity of **Sl** are moved to **alfa**;

Zeros at infinity of **Sl** are moved to **beta**;

Sl is assumed to be a left invertible linear system (**syslin** list) in state-space representation with possibly a polynomial **D** matrix.

See Also

invsyslin , inv , rowregul , rowshuff

Authors

F. D. , R. N. ;

Name

cont_frm — transfer to controllable state-space

```
[sl]=cont_frm(NUM,den)
```

Parameters

NUM

polynomial matrix

den

polynomial

sl

syslin list, sl=[A,B,C,D].

Description

controllable state-space form of the transfer NUM/den.

Examples

```
s=poly(0,'s');NUM=[1+s,s];den=s^2-5*s+1;
sl=cont_frm(NUM,den);
slss=ss2tf(sl);           //Compare with NUM/den
```

See Also

tf2ss , canon , contr

Name

`cont_mat` — controllability matrix

```
Cc=cont_mat(A,B)
Cc=cont_mat(sl)
```

Parameters

`a,b`
two real matrices of appropriate dimensions

`sl`
linear system (`syslin` list)

Description

`cont_mat` returns the controllability matrix of the pair A, B (resp. of the system $sl=[A, B, C, D]$).

```
Cc=[B, AB, A^2 B, ..., A^(n-1) B]
```

See Also

`ctr_gram` , `contr` , `canon` , `st_ility`

Name

contr — controllability, controllable subspace, staircase

```
n=contr(A,B [,tol])
[n,U]=contr(A,B [,tol])
[n,U,ind,V,Ac,Bc]=contr(A,B, [,tol])
```

Parameters

A, B
real matrices

tol
tolerance parameter

n
dimension of controllable subspace.

U
orthogonal change of basis which puts (A, B) in canonical form.

V
orthogonal matrix, change of basis in the control space.

Ac
block Hessenberg matrix $Ac=U' * A * U$

Bc
is $U' * B * V$.

ind
p integer vector associated with controllability indices (dimensions of subspaces $B, B + A*B, \dots = ind(1), ind(1)+ind(2), \dots$)

Description

$[n, [U]] = \text{contr}(A, B, [tol])$ gives the controllable form of an (A, B) pair. ($dx/dt = A x + B u$ or $x(n+1) = A x(n) + b u(n)$). The n first columns of U make a basis for the controllable subspace.

If $V=U(:, 1:n)$, then $V' * A * V$ and $V' * B$ give the controllable part of the (A, B) pair.

The pair (Bc, Ac) is in staircase controllable form.

$$[U' B V | sI - U' A U] = \begin{bmatrix} | & B & | & sI-A & & * & . & . & . & * & & * & | \\ | & 1 & | & & 11 & & . & & & . & & . & | \\ | & & | & A & & sI-A & . & & & . & & . & | \\ | & & | & & 21 & & 22 & & . & . & & . & | \\ | & & | & & & . & & . & & * & & * & | \\ | & 0 & | & & 0 & . & & . & & & & & | \\ | & & | & & & & A & & sI-A & & * & & | \\ | & & | & & & & & p, p-1 & & pp & & & | \\ | & 0 & | & & & 0 & & & 0 & sI-A & & & | \\ | & & | & & & & & & & & p+1, p+1 & & | \end{bmatrix}$$

Reference

Slicot library (see ab01od in SCI/modules/cacsd/src/slicot).

Examples

```
w=ssrand(2,3,5,list('co',3)); //cont. subspace has dim 3.  
A=w("A");B=w("B");  
[n,U]=contr(A,B);n  
A1=U'*A*U;  
spec(A1(n+1:$,n+1:$)) //uncontrollable modes  
spec(A+B*rand(3,5))
```

See Also

canon , cont_mat , unobs , stabil , st_ility

Name

contrss — controllable part

```
[slc]=contrss(sl [,tol])
```

Parameters

sl

linear system (syslin list)

tol

is a threshold for controllability (see `contr`). default value is `sqrt(%eps)`.

Description

returns the controllable part of the linear system $sl = (A, B, C, D)$ in state-space form.

Examples

```
A=[1,1;0,2];B=[1;0];C=[1,1];sl=syslin('c',A,B,C); //Non minimal  
slc=contrss(sl);  
sl1=ss2tf(sl);sl2=ss2tf(slc); //Compare sl1 and sl2
```

See Also

`cont_mat` , `ctr_gram` , `cont_frm` , `contr`

Name

`copfac` — right coprime factorization

```
[N,M,XT,YT]=copfac(G,[polf,polc,tol])
```

Parameters

`G`
syslin list (continuous-time linear system)

`polf, polc`
respectively the poles of `XT` and `YT` and the poles of `n` and `M` (default values `=-1`).

`tol`
real threshold for detecting stable poles (default value `100*%eps`)

`N,M,XT,YT`
linear systems represented by syslin lists

Description

`[N,M,XT,YT]=copfac(G,[polf,polc,[tol]])` returns a right coprime factorization of `G`.

$G = N \cdot M^{-1}$ where `N` and `M` are stable, proper and right coprime. (i.e. $\begin{bmatrix} N & M \end{bmatrix}$ left-invertible with stability)

`XT` and `YT` satisfy:

$\begin{bmatrix} XT & -YT \end{bmatrix} \cdot \begin{bmatrix} M & N \end{bmatrix}' = \text{eye}$ (Bezout identity)

`G` is assumed stabilizable and detectable.

See Also

`syslin` , `lcf`

Name

`csim` — simulation (time response) of linear system

```
[y [,x]]=csim(u,t,sl,[x0 [,tol]])
```

Parameters

- u**
function, list or string (control)
- t**
real vector specifying times with, $t(1)$ is the initial time ($x_0=x(t(1))$).
- sl**
list (`syslin`)
- y**
a matrix such that $y=[y(t(i))]$, $i=1,\dots,n$
- x**
a matrix such that $x=[x(t(i))]$, $i=1,\dots,n$
- tol**
a 2 vector $[atol \ rtol]$ defining absolute and relative tolerances for ode solver (see `ode`)

Description

simulation of the controlled linear system `sl`. `sl` is assumed to be a continuous-time system represented by a `syslin` list.

`u` is the control and `x0` the initial state.

`y` is the output and `x` the state.

The control can be:

1. a function : `[inputs]=u(t)`
2. a list : `list(ut,parameter1,...,parametern)` such that:
`inputs=ut(t,parameter1,...,parametern)` (`ut` is a function)
3. the string "impuls" for impulse response calculation (here `sl` is assumed SISO without direct feed through and $x_0=0$)
4. the string "step" for step response calculation (here `sl` is assumed SISO without direct feed-through and $x_0=0$)
5. a vector giving the values of `u` corresponding to each `t` value.

Examples

```
s=poly(0,'s');rand('seed',0);w=ssrand(1,1,3);w('A')=w('A')-2*eye();
t=0:0.05:5;
//impulse(w) = step(s * w)
clf(0);xset("window",0);xselect();
plot2d([t',t'],[(csim('step',t,tf2ss(s)*w))',0*t'])
clf(1);xset("window",1);xselect();
```

```
plot2d([t',t'],[(csim('impulse',t,w))',0*t'])
//step(w) = impulse (s^-1 * w)
clf(3);xset("window",3);xselect();
plot2d([t',t'],[(csim('step',t,w))',0*t'])
clf(4);xset("window",4);xselect();
plot2d([t',t'],[(csim('impulse',t,tf2ss(1/s)*w))',0*t'])

//input defined by a time function
deff('u=input(t)','u=abs(sin(t))')
clf();plot2d([t',t'],[(csim(input,t,w))',0*t'])
```

See Also

syslin , dsimul , flts , ltitr , rtitr , ode , impl

Name

ctr_gram — controllability gramian

```
[Gc]=ctr_gram(A,B [, dom])  
[Gc]=ctr_gram(sl)
```

Parameters

A,B
two real matrices of appropriate dimensions

dom
character string ('c' (default value) or 'd')

sl
linear system, syslin list

Description

Controllability gramian of (A, B) or sl (a syslin linear system).

dom character string giving the time domain : "d" for a discrete time system and "c" for continuous time (default case).

Examples

```
A=diag([-1, -2, -3]);B=rand(3,2);  
wc=ctr_gram(A,B)  
U=rand(3,3);A1=U*A/U;B1=U*B;  
wc1=ctr_gram(A1,B1) //Not invariant!
```

See Also

equil1 , obs_gram , contr , cont_mat , cont_frm , contrss

Authors

S. Steer INRIA 1988

Name

dbphi — frequency response to phase and magnitude representation

```
[db,phi] =dbphi(rep f)
```

Parameters

db,phi
vector of gains (db) and phases (degrees)

rep f
vector of complex frequency response

Description

$\text{db}(k)$ is the magnitude of $\text{rep f}(k)$ expressed in dB i.e. $\text{db}(k)=20*\log(\text{abs}(\text{rep f}(k)))/\log(10)$ and $\text{phi}(k)$ is the phase of $\text{rep f}(k)$ expressed in degrees.

See Also

repfreq , bode

Name

dcf — double coprime factorization

```
[N,M,X,Y,NT,MT,XT,YT]=dcf(G,[polf,polc,[tol]])
```

Parameters

G
syslin list (continuous-time linear system)

polf, polc
respectively the poles of XT and YT and the poles of N and M (default values = -1).

tol
real threshold for detecting stable poles (default value 100*%eps).

N,M,XT,YT,NT,MT,X,Y
linear systems represented by syslin lists

Description

returns eight stable systems (N, M, X, Y, NT, MT, XT, YT) for the doubly coprime factorization

G must be stabilizable and detectable.

See Also

copfac

Name

ddp — disturbance decoupling

```
[Closed, F, G]=ddp(Sys, zeroed, B1, D1)
[Closed, F, G]=ddp(Sys, zeroed, B1, D1, flag, alfa, beta)
```

Parameters

Sys

sys lin list containing the matrices (A, B2, C, D2).

zeroed

integer vector, indices of outputs of Sys which are zeroed.

B1

real matrix

D1

real matrix. B1 and D1 have the same number of columns.

flag

string 'ge' or 'st' (default) or 'pp'.

alpha

real or complex vector (loc. of closed loop poles)

beta

real or complex vector (loc. of closed loop poles)

Description

Exact disturbance decoupling (output nulling algorithm). Given a linear system, and a subset of outputs, z, which are to be zeroed, characterize the inputs w of Sys such that the transfer function from w to z is zero. Sys is a linear system {A,B2,C,D2} with one input and two outputs (i.e. Sys: u-->(z,y)), part the following system defined from Sys and B1, D1:

$$\begin{aligned} \dot{x} &= A x + B1 w + B2 u \\ z &= C1 x + D11 w + D12 u \\ y &= C2 x + D21 w + D22 u \end{aligned}$$

outputs of Sys are partitioned into (z,y) where z is to be zeroed, i.e. the matrices C and D2 are:

$$\begin{aligned} C &= [C1; C2] & D2 &= [D12; D22] \\ C1 &= C(\text{zeroed}, :) & D12 &= D2(\text{zeroed}, :) \end{aligned}$$

The matrix D1 is partitioned similarly as $D1 = [D11; D21]$ with $D11 = D1(\text{zeroed}, :)$. The control is $u = Fx + Gw$ and one looks for matrices F, G such that the closed loop system: $w \rightarrow z$ given by

$$\begin{aligned} \dot{x} &= (A + B2 * F) x + (B1 + B2 * G) w \\ z &= (C1 + D12 * F) x + (D11 + D12 * G) w \end{aligned}$$

has zero transfer function.

flag='ge' no stability constraints. flag='st' : look for stable closed loop system ($A+B_2^*F$ stable). flag='pp' : eigenvalues of $A+B_2^*F$ are assigned to `alfa` and `beta`.

Closed is a realization of the $w \rightarrow y$ closed loop system

$$\begin{aligned} \dot{x} &= (A+B_2^*F) x + (B_1 + B_2^*G) w \\ y &= (C_2+D_{22}^*F) x + (D_{21}+D_{22}^*G) w \end{aligned}$$

Stability (resp. pole placement) requires stabilizability (resp. controllability) of (A,B_2) .

Examples

```
rand('seed',0);nx=6;nz=3;nu=2;ny=1;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B2=[1,2;0,3;0,4;0,5;0,0;0,0];
C1=zeros(nz,nz),eye(nz,nz)];D12=[0,1;0,2;0,3];
Sys12=syslin('c',A,B2,C1,D12);
C=[C1;rand(ny,nx)];D2=[D12;rand(ny,size(D12,2))];
Sys=syslin('c',A,B2,C,D2);
[A,B2,C1,D12]=abcd(Sys12); //The matrices of Sys12.
my_alpha=-1;my_beta=-2;flag='ge';
[X,dims,F,U,k,Z]=abinv(Sys12,my_alpha,my_beta,flag);
clean(X'*(A+B2*F)*X)
clean(X'*B2*U)
clean((C1+D12*F)*X)
clean(D12*U);
//Calculating an ad-hoc B1,D1
G1=rand(size(B2,2),3);
B1=-B2*G1;
D11=-D12*G1;
D1=[D11;rand(ny,size(B1,2))];

[Closed,F,G]=ddp(Sys,1:nz,B1,D1,'st',my_alpha,my_beta);
closed=syslin('c',A+B2*F,B1+B2*G,C1+D12*F,D11+D12*G);
ss2tf(closed)
```

See Also

`abinv` , `ui_observer`

Authors

F.D.

Name

des2ss — descriptor to state-space

```
[Sl]=des2ss(A,B,C,D,E [,tol])  
[Sl]=des2ss(Des)
```

Parameters

A,B,C,D,E

real matrices of appropriate dimensions

Des

list

Sl

syslin list

tol

real parameter (threshold) (default value `100*%eps`).

Description

Descriptor to state-space transform.

`Sl=des2ss(A,B,C,D,E)` returns a linear system `Sl` equivalent to the descriptor system (E, A, B, C, D) .

For index one (E, A) pencil, explicit formula is used and for higher index pencils `rowshuff` is used.

`Sl=des2ss(Des)` with `Des=list('des',A,B,C,D,E)` returns a linear system `Sl` in state-space form with possibly a polynomial `D` matrix.

A generalized Leverrier algorithm is used.

Examples

```
s=poly(0,'s');G=[1/(s-1),s;1,2/s^3];  
S1=tf2des(G);S2=tf2des(G,"withD");  
W1=des2ss(S1);W2=des2ss(S2);  
clean(ss2tf(W1))  
clean(ss2tf(W2))
```

See Also

`des2tf` , `glever` , `rowshuff`

Name

des2tf — descriptor to transfer function conversion

```
[S]=des2tf(sl)
[Bfs,Bis,chis]=des2tf(sl)
```

Parameters

sl
list (linear system in descriptor form)

Bfs, Bis
two polynomial matrices

chis
polynomial

S
rational matrix

Description

Given the linear system in descriptor form i.e. `Sl=list('des',A,B,C,D,E)`, `des2tf` converts `sl` into its transfer function representation:

$$S=C*(s*E-A)^{-1}*B+D$$

Called with 3 outputs arguments `des2tf` returns `Bfs` and `Bis` two polynomial matrices, and `chis` polynomial such that:

$$S=Bfs/chis - Bis$$

`chis` is the determinant of $(s*E-A)$ (up to a xcatve constant);

Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];
Descrip=tf2des(G);Tf1=des2tf(Descrip)
Descrip2=tf2des(G,"withD");Tf2=des2tf(Descrip2)
[A,B,C,D,E]=Descrip2(2:6);Tf3=C*inv(s*E-A)*B+D
```

See Also

glever , pol2des , tf2des , ss2tf , des2ss , rowshuff

Authors

F. D.

Name

dhinf — H_infinity design of discrete-time systems

```
[AK, BK, CK, DK, (RCOND)] = dishin(A, B, C, D, ncon, nmeas, gamma)
```

Parameters

- A**
the n-by-n system state matrix A.
- B**
the n-by-m system input matrix B.
- C**
the p-by-n system output matrix C.
- D**
the p-by-m system matrix D.
- ncon**
the number of control inputs. $m \geq ncon \geq 0$, $p - nmeas \geq ncon$.
- nmeas**
the number of measurements. $p \geq nmeas \geq 0$, $m - ncon \geq nmeas$.
- gamma**
the parameter gamma used in H_infinity design. It is assumed that gamma is sufficiently large so that the controller is admissible. $gamma \geq 0$.
- AK**
the n-by-n controller state matrix AK.
- BK**
the n-by-nmeas controller input matrix BK.
- CK**
the ncon-by-n controller output matrix CK.
- DK**
the ncon-by-nmeas controller matrix DK.
- RCOND**
a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)
- RCOND**
(1) contains the reciprocal condition number of the matrix R3,
- RCOND**
(2) contains the reciprocal condition number of the matrix $R1 - R2 * \text{inv}(R3) * R2$
- RCOND**
(3) contains the reciprocal condition number of the matrix V21,
- RCOND**
(4) contains the reciprocal condition number of the matrix St3,
- RCOND**
(5) contains the reciprocal condition number of the matrix V12,

RCOND

(6) contains the reciprocal condition number of the matrix $I_{m2} + DKHAT * D22$,

RCOND

(7) contains the reciprocal condition number of the X-Riccati equation,

RCOND

(8) contains the reciprocal condition number of the Z-Riccati equation.

Description

$[AK, BK, CK, DK, (RCOND)] = dhinf(A, B, C, D, ncon, nmeas, \gamma)$ To compute the matrices of an H-infinity (sub)optimal n-state controller

$$K = \begin{bmatrix} | & AK & | & BK & | \\ | & --- & | & --- & | \\ | & CK & | & DK & | \end{bmatrix},$$

for the discrete-time system

$$P = \begin{bmatrix} | & A & | & B1 & B2 & | \\ | & --- & | & --- & --- & | \\ | & C1 & | & D11 & D12 & | \\ | & C2 & | & D21 & D22 & | \end{bmatrix} = \begin{bmatrix} | & A & | & B & | \\ | & --- & | & --- & | \\ | & C & | & D & | \end{bmatrix},$$

and for a given value of γ , where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of linear discrete-time control systems. Report99-8, Department of Engineering, Leicester University, April 1999.

Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[ -0.7  0    0.3  0    -0.5 -0.1
    -0.6  0.2 -0.4 -0.3  0    0
    -0.5  0.7 -0.1  0    0    -0.8
    -0.7  0    0    -0.5 -1    0
    0    0.3  0.6 -0.9  0.1 -0.4
    0.5 -0.8  0    0    0.2 -0.9];
B=[ -1 -2 -2  1  0
    1  0  1 -2  1
    -3 -4  0  2 -2
    1 -2  1  0 -1
    0  1 -2  0  3
    1  0  3 -1 -2];
C=[ 1 -1  2 -2  0 -3
    -3  0  1 -1  1  0
    0  2  0 -4  0 -2
```

```
      1 -3  0  0  3  1
      0  1 -2  1  0 -2];
D=[1 -1 -2  0  0
    0  1  0  1  0
    2 -1 -3  0  1
    0  1  0  1 -1
    0  0  1  2  1];

ncon=2
nmeas=2
gam=111.30;
[AK,BK,CK,DK] = dhinf(A,B,C,D,ncon,nmeas,gam)
```

See Also

hinf , h_inf

Name

dhnorm — discrete H-infinity norm

```
hinfnorm=dhnorm(sl,[tol],[normax])
```

Parameters

sl
the state space system (sys lin list) (discrete-time)

tol
tolerance in bisection step, default value 0.01

normax
upper bound for the norm , default value is 1000

hinfnorm
the discrete infinity norm of S1

Description

produces the discrete-time infinity norm of a state-space system (the maximum over all frequencies on the unit circle of the maximum singular value).

See Also

h_norm , linfo

Name

dscr — discretization of linear system

```
[sld [,r]]=dscr(sl,dt [,m])
```

Parameters

sl
syslin list containing $[A, B, C, D]$.

dt
real number, sampling period

m
covariance of the input noise (continuous time)(default value=0)

r
covariance of the output noise (discrete time) given if **m** is given as input

sld
sampled (discrete-time) linear system, syslin list

Description

Discretization of linear system. **sl** is a continuous-time system:

$$dx/dt = A \cdot x + B \cdot u \quad (+ \text{ noise}).$$

sld is the discrete-time system obtained by sampling **sl** with the sampling period **dt**.

Examples

```
s=poly(0,'s');  
Sys=syslin('c',[1,1/(s+1);2*s/(s^2+2),1/s])  
ss2tf(dscr(tf2ss(Sys),0.1))
```

See Also

syslin , flts , dsimul

Name

dsimul — state space discrete time simulation

```
y=dsimul(sl,u)
```

Parameters

sl
syslin list describing a discrete time linear system

u
real matrix of appropriate dimension

y
output of sl

Description

Utility function. If $[A,B,C,D]=abcd(sl)$ and $x0=sl('X0')$, dsimul returns $y=C*ltitr(A,B,u,x0)+D*u$ i.e. the time response of sl to the input u. sl is assumed to be in state space form (syslin list).

Examples

```
z=poly(0,'z');  
h=(1-2*z)/(z^2-0.2*z+1);  
sl=tf2ss(h);  
u=zeros(1,20);u(1)=1;  
x1=dsimul(sl,u) //Impulse response  
u=ones(1,20);  
x2=dsimul(sl,u); //Step response
```

See Also

syslin , flts , ltitr

Name

dt_ility — detectability test

```
[k, [n [U [Sld ] ] ]]=dt_ility(Sl [,tol])
```

Parameters

Sl
linear system (syslin list)

n
dimension of unobservable subspace

k
dimension of unstable, unobservable subspace ($k \leq n$).

U
orthogonal matrix

Sld
linear system (syslin list)

tol
threshold for controllability test.

Description

Detectability test for **Sl**, a linear system in state-space representation. **U** is a basis whose **k** first columns span the unstable, unobservable subspace of **Sl** (intersection of unobservable subspace of (A, C) and unstable subspace of **A**). Detectability means $k=0$.

Sld = (**U'*****A*****U**, **U'*****B**, **C*****U**, **D**) displays the "detectable part" of **Sl**=(**A**, **B**, **C**, **D**), i.e.

```
U'*A*U = [*,*,*]
          [0,*,*]
          [0,0,*]

C*U = [0,0,*]
```

with (**A33**, **C3**) observable (dimension $n \times n$), **A22** stable (dimension $n - k$) and **A11** unstable (dimension k).

Examples

```
A=[2,1,1;0,-2,1;0,0,3];
C=[0,0,1];
X=rand(3,3);A=inv(X)*A*X;C=C*X;
W=syslin('c',A,[],C);
[k,n,U,w1]=dt_ility(W);
w1("A")
w1("C")
```

See Also

contr , st_ility , unobs , stabil

Name

dtsi — stable anti-stable decomposition

```
[Ga, Gs, Gi]=dtsi(G, [tol])
```

Parameters

- G
linear system (syslin list)
- Ga
linear system (syslin list) antistable and strictly proper
- Gs
linear system (syslin list) stable and strictly proper
- Gi
real matrix (or polynomial matrix for improper systems)
- tol
optional parameter for detecting stables poles. Default value: `100*%eps`

Description

returns the stable-antistable decomposition of G:

$$G = G_a + G_s + G_i, (G_i = G(\infty))$$

G can be given in state-space form or in transfer form.

See Also

syslin , pbig , psmall , pfss

Name

equil — balancing of pair of symmetric matrices

```
T=equil(P,Q)
```

Parameters

P, Q
two positive definite symmetric matrices

T
nonsingular matrix

Description

equil returns t such that:

T^*P*T' and $\text{inv}(T)'*Q*\text{inv}(T)$ are both equal to a same diagonal and positive matrix.

Examples

```
P=rand(4,4);P=P*P';  
Q=rand(4,4);Q=Q*Q';  
T=equil(P,Q)  
clean(T*P*T')  
clean(inv(T)'*Q*inv(T))
```

See Also

equil1 , balanc , ctr_gram

Name

equil1 — balancing (nonnegative) pair of matrices

```
[T [,siz]]=equil1(P,Q [,tol])
```

Parameters

P, Q
two non-negative symmetric matrices

T
nonsingular matrix

siz
vector of three integers

tol
threshold

Description

equil1 computes t such that:

$P1 = T * P * T'$ and $Q1 = \text{inv}(T)' * Q * \text{inv}(T)$ are as follows:

$P1 = \text{diag}(S1, S2, 0, 0)$ and $Q1 = \text{diag}(S1, 0, S3, 0)$ with $S1, S2, S3$ positive and diagonal matrices with respective dimensions $\text{siz} = [n1, n2, n3]$

tol is a threshold for rank determination in SVD

Examples

```
S1=rand(2,2);S1=S1*S1';
S2=rand(2,2);S2=S2*S2';
S3=rand(2,2);S3=S3*S3';
P=sysdiag(S1,S2,zeros(4,4));
Q=sysdiag(S1,zeros(2,2),S3,zeros(2,2));
X=rand(8,8);
P=X*P*X';Q=inv(X)'*Q*inv(X);
[T,siz]=equil1(P,Q);
P1=clean(T*P*T')
Q1=clean(inv(T)'*Q*inv(T))
```

See Also

balreal , minreal , equil , hankelsv

Authors

S. Steer 1987

Name

evans — Evans root locus

```
evans(H [, kmax])
```

Parameters

H
list (linear system `syslin`)

kmax
real (maximum gain desired for the plot)

Description

Gives the Evans root locus for a linear system in state-space or transfer form $H(s)$ (`syslin` list). This is the locus of the roots of $1+k*H(s)=1+k*N(s)/D(s)$, in the complex plane. For a selected sample of gains $k \leq k_{max}$, the imaginary part of the roots of $D(s)+k*N(s)$ is plotted vs the real part.

To obtain the gain at a given point of the locus you can simply execute the following instruction : $k=-1/\text{real}(\text{horner}(h, [1, \%i]*\text{locate}(1)))$ and click the desired point on the root locus. If the coordinates of the selected point are in the real 2 x 1 vector $P=\text{locate}(1)$ this k solves the equation $k*N(w) + D(w) = 0$ with $w=P(1)+\%i*P(2)=[1, \%i]*P$.

Examples

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
P=3.0548543 - 8.8491842*%i;    //P=selected point
k=-1/real(horner(H,P));
Ns=H('num');Ds=H('den');
roots(Ds+k*Ns)    //contains P as particular root
// Another one
clf();s=poly(0,'s');n=1+s;
d=real(poly([-1 -2 -%i %i],'s'));
evans(n,d,100);
//
clf();n=real(poly([0.1-%i 0.1+%i,-10],'s'));
evans(n,d,80);
```

See Also

kpure , krac2 , locate

Name

feedback — feedback operation

```
S1=S11/.S12
```

Parameters

S11,S12

linear systems (`syslin` list) in state-space or transfer form, or ordinary gain matrices.

S1

linear system (`syslin` list) in state-space or transfer form

Description

The feedback operation is denoted by `/.` (slashdot). This command returns $S1 = S11 * (I + S12 * S11)^{-1}$, i.e the (negative) feedback of `S11` and `S12`. `S1` is the transfer $v \rightarrow y$ for $y = S11 u, u = v - S12 y$.

The result is the same as $S1 = \text{LFT}([0, I; I, -S12], S11)$.

Caution: do not use with decimal point (e.g. `1/.1` is ambiguous!)

Examples

```
S1=ssrand(2,2,3);S2=ssrand(2,2,2);
W=S1/.S2;
ss2tf(S1/.S2)
//Same operation by LFT:
ss2tf(lft([zeros(2,2),eye(2,2);eye(2,2),-S2],S1))
//Other approach: with constant feedback
BigS=sysdiag(S1,S2); F=[zeros(2,2),eye(2,2);-eye(2,2),zeros(2,2)];
Bigclosed=BigS/.F;
W1=Bigclosed(1:2,1:2); //W1=W (in state-space).
ss2tf(W1)
//Inverting
ss2tf(S1*inv(eye()+S2*S1))
```

See Also

`lft` , `sysdiag` , `augment` , `obscont`

Name

findABCD — discrete-time system subspace identification

```
[SYS,K] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
SYS = findABCD(S,N,L,R,METH)

[SYS,K,Q,Ry,S,RCND] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
[SYS,RCND] = findABCD(S,N,L,R,METH)
```

Parameters

S
integer, the number of block rows in the block-Hankel matrices

N
integer, the system order

L
integer, the number of output

R
matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.

METH
integer, an option for the method to use

= 1
MOESP method with past inputs and outputs;

= 2
N4SID method;

= 3
combined method: A and C via MOESP, B and D via N4SID.

Default: METH = 3.

NSMPL
integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.

TOL
the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
integer, switch for printing the warning messages.

PRINTW
= 1: print warning messages;

PRINTW
= 0: do not print warning messages.

Default: PRINTW = 0.

SYScomputes a state-space realization $SYS = (A,B,C,D)$ (an `syslin` object)**K**the Kalman predictor gain K (if $NSMPL > 0$)**Q**

state covariance

Ry

output covariance

S

state-output cross-covariance

RCND

vector, reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions

Description

Finds the system matrices and the Kalman gain of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP and/or N4SID).

- $[SYS,K] = \text{findABCD}(S,N,L,R,METH,NSMPL,TOL,PRINTW)$ computes a state- space realization $SYS = (A,B,C,D)$ (an `ss` object), and the Kalman predictor gain K (if $NSMPL > 0$). The model structure is:

$$\begin{aligned}x(k+1) &= Ax(k) + Bu(k) + Ke(k), & k \geq 1, \\y(k) &= Cx(k) + Du(k) + e(k),\end{aligned}$$

where $x(k)$ and $y(k)$ are vectors of length N and L , respectively.

- $[SYS,K,Q,Ry,S,RCND] = \text{findABCD}(S,N,L,R,METH,NSMPL,TOL,PRINTW)$ also returns the state, output, and state-output (cross-)covariance matrices Q , Ry , and S (used for computing the Kalman gain), as well as the vector $RCND$ of length lr containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions, where

$lr = 4$, if Kalman gain matrix K is not required, and
 $lr = 12$, if Kalman gain matrix K is required.

Matrix R , computed by `findR`, should be determined with suitable arguments `METH` and `JOB`. `METH = 1` and `JOB = 1` must be used in `findR`, for `METH = 1` in `findABCD`; `METH = 1` must be used in `findR`, for `METH = 3` in `findABCD`.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1, -0.1, 0.2;
      0.1, 0, -0.1, -0.1;
      -0.4, -0.6, -0.7, -0.1;
      0.8, 0, -0.6, -0.6];
B = [0.8;0.1;1;-1];
```

```
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;
[R,N1,SVAL] = findR(S,Y',U');
N=3;
SYS1 = findABCD(S,N,1,R) ;SYS1.dt=0.1;

SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findAC , findBD , findBDK , findR , sorder , sident

Name

findAC — discrete-time system subspace identification

```
[A,C] = findAC(S,N,L,R,METH,TOL,PRINTW)
[A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW)
```

Parameters

- S**
integer, the number of block rows in the block-Hankel matrices
- N**
integer
- L**
integer
- R**
matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.
- METH**
integer, an option for the method to use
- = 1
MOESP method with past inputs and outputs;
- = 2
N4SID method;
- Default: METH = 3.
- TOL**
the tolerance used for estimating the rank of matrices. If $TOL > 0$, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: $\text{prod}(\text{size}(\text{matrix})) \times \text{epsilon\_machine}$ where `epsilon_machine` is the relative machine precision.
- PRINTW**
integer, switch for printing the warning messages.
- PRINTW**
= 1: print warning messages;
- = 0
do not print warning messages.
- Default: PRINTW = 0.
- A**
matrix, state system matrix
- C**
matrix, output system matrix
- RCND**
vector of length 4, condition numbers of the matrices involved in rank decision

Description

finds the system matrices A and C of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- `[A,C] = findAC(S,N,L,R,METH,TOL,PRINTW)` computes the system matrices A and C. The model structure is: $x(k+1) = Ax(k) + Bu(k) + Ke(k)$, $k \geq 1$, $y(k) = Cx(k) + Du(k) + e(k)$, where $x(k)$ and $y(k)$ are vectors of length N and L, respectively.
- `[A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW)` also returns the vector RCND of length 4 containing the condition numbers of the matrices involved in rank decisions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1, -0.1, 0.2;
      0.1, 0, -0.1, -0.1;
      -0.4, -0.6, -0.7, -0.1;
      0.8, 0, -0.6, -0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
```

See Also

findABCD , findBD , findBDK , findR , sorder , sident

Name

findBD — initial state and system matrices B and D of a discrete-time system

```
[(x0) (,B (,D)) (,V) (,rcnd)] = findBD(jobx0,comuse (,job),A (,B),C (,D),Y  
(,U,tol,printw,ldwork))
```

Parameters

jobx0

integer option to specify whether or not the initial state should be computed:

=

1 : compute the initial state x0;

=

2 : do not compute the initial state (possibly, because x0 is known to be zero).

comuse

integer option to specify whether the system matrices B and D should be computed or used:

=

1 : compute the matrices B and D, as specified by job;

=

2 : use the matrices B and D, as specified by job;

=

3 : do not compute/use the matrices B and D.

job

integer option to determine which of the system matrices B and D should be computed or used:

=

1 : compute/use the matrix B only (D is known to be zero);

=

2 : compute/use the matrices B and D.

job must not be specified if jobx0 = 2 and comuse = 2, or if comuse = 3.

A

state matrix of the given system

B

optionnal, input matrix of the given system

C

output matrix of the given system

D

optionnal, direct feedthrough of the given system

Y

the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U

the t-by-m input-data sequence matrix (input when jobx0 = 1 and comuse = 2, or comuse = 1). Column j of U contains the t values of the j-th input component for consecutive time increments.

tol

optionnal, tolerance used for estimating the rank of matrices. If $\text{tol} > 0$, then the given value of tol is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than $1/\text{tol}$ is considered to be of full rank. Default: $m*n*\text{epsilon\_machine}$ where epsilon_machine is the relative machine precision.

printw

optionnal, switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: printw = 0.

ldwork

(optional) the workspace size. Default : computed by the formula $\text{LDWORK} = \text{MAX}(\text{minimum workspace size needed}, 2*CSIZE/3, CSIZE - (m + 1)*t - 2*n*(n + m + 1) - l*m)$ where CSIZE is the cache size in double precision words.

x0

initial state vector

Br

system input matrix

Dr

system direct feedthrough matrix

V

the n-by-n orthogonal matrix which reduces A to a real Schur form (output when jobx0 = 1 or comuse = 1).

rcnd

(optional) the reciprocal condition numbers of the matrices involved in rank decisions.

Description

findBD function for estimating the initial state and the system matrices B and D of a discrete-time system, using SLICOT routine IB01CD.

```
[x0, Br, V, rcnd] = findBD(1, 1, 1, A, C, Y, U)
[x0, Br, Dr, V, rcnd] = findBD(1, 1, 2, A, C, Y, U)
[Br, V, rcnd] = findBD(2, 1, 1, A, C, Y, U)
[B, Dr, V, rcnd] = findBD(2, 1, 2, A, C, Y, U)
[x0, V, rcnd] = findBD(1, 2, 1, A, B, C, Y, U)
[x0, V, rcnd] = findBD(1, 2, 2, A, B, C, D, Y, U)
[x0, rcnd] = findBD(2, 2) // (Set x0 = 0, rcnd = 1)
[x0, V, rcnd] = findBD(1, 3, A, C, Y)
```

Note: the example lines above may contain at the end the parameters tol, printw, ldwork.

FINDBD estimates the initial state and/or the system matrices Br and Dr of a discrete-time system, given the system matrices A, C, and possibly B, D, and the input and output trajectories of the system.

The model structure is :

$$\begin{aligned}x(k+1) &= Ax(k) + Bu(k), \quad k \geq 1, \\y(k) &= Cx(k) + Du(k),\end{aligned}$$

where $x(k)$ is the n -dimensional state vector (at time k),

$u(k)$ is the m -dimensional input vector,

$y(k)$ is the l -dimensional output vector,

and A , B , C , and D are real matrices of appropriate dimensions.

Comments

1. The n -by- m system input matrix B is an input parameter when $jobx0 = 1$ and $comuse = 2$, and it is an output parameter when $comuse = 1$.
2. The l -by- m system matrix D is an input parameter when $jobx0 = 1$, $comuse = 2$ and $job = 2$, and it is an output parameter when $comuse = 1$ and $job = 2$.
3. The n -vector of estimated initial state $x(0)$ is an output parameter when $jobx0 = 1$, but also when $jobx0 = 2$ and $comuse \leq 2$, in which case it is set to 0.
4. If $ldwork$ is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1, -0.1, 0.2;
      0.1, 0, -0.1, -0.1;
      -0.4, -0.6, -0.7, -0.1;
      0.8, 0, -0.6, -0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
[X0,B,D] = findBD(1,1,2,A,C,Y',U')
SYS1=syslin(1,A,B,C,D,X0);

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```


See Also

inistate , findx0BD , findABCD , findAC , findBD

Authors

V. Sima, Katholieke Univ. Leuven, Belgium, May 2000. (Revisions: V. Sima, July 2000)

Name

findBDK — Kalman gain and B D system matrices of a discrete-time system

```
[B,D,K] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
[B,D,RCND] = findBDK(S,N,L,R,A,C,METH,JOB)
[B,D,K,Q,Ry,S,RCND] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
```

Parameters

- S**
integer, the number of block rows in the block-Hankel matrices
- N**
integer
- L**
integer
- R**
matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findR.
- A**
square matrix
- C**
matrix
- METH**
integer, an option for the method to use
- = 1
MOESP method with past inputs and outputs;
- = 2
N4SID method;
- Default: METH = 2.
- JOB**
an option specifying which system matrices should be computed:
- = 1
compute the matrix B;
- = 2
compute the matrices B and D.
- Default: JOB = 2.
- NSMPL**
integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.
- TOL**
the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW

integer, switch for printing the warning messages.

PRINTW

= 1: print warning messages;

PRINTW

= 0: do not print warning messages.

Default: PRINTW = 0.

SYS

computes a state-space realization $SYS = (A,B,C,D)$ (an syslin object)

K

the Kalman predictor gain K (if NSMPL > 0)

Q

state covariance

Ry

output covariance

S

state-output cross-covariance

RCND

the vector of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

Description

finds the system matrices B and D and the Kalman gain of a discrete-time system, given the system order, the matrices A and C, and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- $[B,D,K] = \text{findBDK}(S,N,L,R,A,C,\text{METH},\text{JOB},\text{NSMPL},\text{TOL},\text{PRINTW})$ computes the system matrices B (if JOB = 1), B and D (if JOB = 2), and the Kalman predictor gain K (if NSMPL > 0). The model structure is:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + Ke(k), & k >= 1, \\ y(k) &= Cx(k) + Du(k) + e(k), \end{aligned}$$

where $x(k)$ and $y(k)$ are vectors of length N and L, respectively.

- $[B,D,RCND] = \text{findBDK}(S,N,L,R,A,C,\text{METH},\text{JOB})$ also returns the vector RCND of length 4 containing the reciprocal condition numbers of the matrices involved in rank decisions.
- $[B,D,K,Q,Ry,S,RCND] = \text{findBDK}(S,N,L,R,A,C,\text{METH},\text{JOB},\text{NSMPL},\text{TOL},\text{PRINTW})$ also returns the state, output, and state-output (cross-)covariance matrices Q, Ry, and S (used for computing the Kalman gain), as well as the vector RCND of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD. METH = 1 and JOBD = 1 must be used in findR, for METH = 1 in findBDK. Using METH = 1 in FINDR and METH = 2 in findBDK is allowed.

The number of output arguments may vary, but should correspond to the input arguments, e.g.,

```
B = findBDK(S,N,L,R,A,C,METH,1) or  
[B,D] = findBDK(S,N,L,R,A,C,METH,2) or  
[B,D,RCND] = findBDK(S,N,L,R,A,C,METH,2)
```

Examples

```
//generate data from a given linear system  
A = [ 0.5, 0.1, -0.1, 0.2;  
      0.1, 0, -0.1, -0.1;  
      -0.4, -0.6, -0.7, -0.1;  
      0.8, 0, -0.6, -0.6];  
B = [0.8;0.1;1;-1];  
C = [1 2 -1 0];  
SYS=syslin(0.1,A,B,C);  
nsmp=100;  
U=prbs_a(nsmpt,nsmp/5);  
Y=(flts(U,SYS)+0.3*rand(1,nsmp, 'normal'));  
  
// Compute R  
S=15;L=1;  
[R,N,SVAL] = findR(S,Y',U');  
  
N=3;  
METH=3;TOL=-1;  
[A,C] = findAC(S,N,L,R,METH,TOL);  
[B,D,K] = findBDK(S,N,L,R,A,C);  
SYS1=syslin(1,A,B,C,D);  
  
SYS1.X0 = inistate(SYS1,Y',U');  
  
Y1=flts(U,SYS1);  
clf();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findABCD , findAC , findBD , findR , sorder , sident

Name

findR — Preprocessor for estimating the matrices of a linear time-invariant dynamical system

```
[R,N [,SVAL,RCND]] = findR(S,Y,U,METH,ALG,JOB,TOL,PRINTW)
[R,N] = findR(S,Y)
```

Parameters

S

the number of block rows in the block-Hankel matrices.

Y

U

METH

an option for the method to use:

1

MOESP method with past inputs and outputs;

2

N4SI15 0 1 1 1000D method.

Default: METH = 1.

ALG

an option for the algorithm to compute the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

1

Cholesky algorithm on the correlation matrix;

2

fast QR algorithm;

3

standard QR algorithm.

Default: ALG = 1.

JOB

an option to specify if the matrices B and D should later be computed using the MOESP approach:

=

1 : the matrices B and D should later be computed using the MOESP approach;

=

2 : the matrices B and D should not be computed using the MOESP approach.

Default: JOB = 2. This parameter is not relevant for METH = 2.

TOL

a vector of length 2 containing tolerances:

TOL

(1) is the tolerance for estimating the rank of matrices. If TOL(1) > 0, the given value of TOL(1) is used as a lower bound for the reciprocal condition number.

Default: TOL(1) = prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

TOL

(2) is the tolerance for estimating the system order. If $\text{TOL}(2) \geq 0$, the estimate is indicated by the index of the last singular value greater than or equal to $\text{TOL}(2)$. (Singular values less than $\text{TOL}(2)$ are considered as zero.)

When $\text{TOL}(2) = 0$, then $S \cdot \text{epsilon\_machine} \cdot \text{sval}(1)$ is used instead $\text{TOL}(2)$, where $\text{sval}(1)$ is the maximal singular value. When $\text{TOL}(2) < 0$, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default: $\text{TOL}(2) = -1$.

PRINTW

a switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: $\text{PRINTW} = 0$.

R**N**

the order of the discrete-time realization

SVAL

singular values SVAL, used for estimating the order.

RCND

vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

Description

findR Preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOE-SP or N4SID), and estimates the system order.

$[R, N] = \text{findR}(S, Y, U, \text{METH}, \text{ALG}, \text{JOB}, \text{TOL}, \text{PRINTW})$ returns the processed upper triangular factor R of the concatenated block-Hankel matrices built from the input-output data, and the order N of a discrete-time realization. The model structure is:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + w(k), & k \geq 1, \\ y(k) &= Cx(k) + Du(k) + e(k). \end{aligned}$$

The vectors $y(k)$ and $u(k)$ are transposes of the k -th rows of Y and U, respectively.

$[R, N, \text{SVAL}, \text{RCND}] = \text{findR}(S, Y, U, \text{METH}, \text{ALG}, \text{JOB}, \text{TOL}, \text{PRINTW})$ also returns the singular values SVAL, used for estimating the order, as well as, if $\text{meth} = 2$, the vector RCND of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

$[R, N] = \text{findR}(S, Y)$ assumes $U = []$ and default values for the remaining input arguments.

Examples

```
//generate data from a given linear system
```

```
A = [ 0.5, 0.1, -0.1, 0.2;  
      0.1, 0, -0.1, -0.1;  
      -0.4, -0.6, -0.7, -0.1;  
      0.8, 0, -0.6, -0.6];  
B = [0.8;0.1;1;-1];  
C = [1 2 -1 0];  
SYS=syslin(0.1,A,B,C);  
U=(ones(1,1000)+rand(1,1000,'normal'));  
Y=(flts(U,SYS)+0.5*rand(1,1000,'normal'));  
// Compute R  
  
[R,N,SVAL] = findR(15,Y',U');  
SVAL  
N
```

See Also

findABCD , findAC , findBD , findBDK , sorder , sident

Name

findx0BD — Estimates state and B and D matrices of a discrete-time linear system

```
[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW)
[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U)
```

Parameters

A
state matrix of the system

C
C matrix of the system

Y
system output

U
system input

WITHX0
a switch for estimating the initial state x0.

=
1: estimate x0;
=
0: do not estimate x0.

Default: WITHX0 = 1.

WITHD
a switch for estimating the matrix D.

=
1: estimate the matrix D;
=
0: do not estimate the matrix D.

Default: WITHD = 1.

TOL
the tolerance used for estimating the rank of matrices. If $TOL > 0$, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: $\text{prod}(\text{size}(\text{matrix})) \times \text{epsilon\_machine}$ where `epsilon_machine` is the relative machine precision.

PRINTW
a switch for printing the warning messages.

=
1: print warning messages;
=
0: do not print warning messages.

Default: PRINTW = 0.

X0
initial state of the estimated linear system.

- B
B matrix of the estimated linear system.
- D
D matrix of the estimated linear system.
- V
orthogonal matrix which reduces the system state matrix A to a real Schur form
- rcnd
estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

Description

findx0BD Estimates the initial state and/or the matrices B and D of a discrete-time linear system, given the (estimated) system matrices A, C, and a set of input/output data.

[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW) estimates the initial state X0 and the matrices B and D of a discrete-time system using the system matrices A, C, output data Y and the input data U. The model structure is :

$$\begin{aligned}x(k+1) &= Ax(k) + Bu(k), & k >= 1, \\y(k) &= Cx(k) + Du(k),\end{aligned}$$

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U) also returns the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as some estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

```
B = findx0BD(A,C,Y,U,0,0) returns B only, and
[B,D] = findx0BD(A,C,Y,U,0) returns B and D only.
```

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1, -0.1, 0.2;
      0.1, 0, -0.1, -0.1;
      -0.4, -0.6, -0.7, -0.1;
      0.8, 0, -0.6, -0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
```

```
[X0,B,D,V,rcnd] = findx0BD(A,C,Y',U');  
SYS1=syslin(1,A,B,C,D,X0);  
Y1=flts(U,SYS1);  
clf();plot2d((1:nsmpl)',[Y',Y1'])
```

See Also

findBD , inistate

Name

flts — time response (discrete time, sampled system)

```
[y [,x]]=flts(u,sl [,x0])  
[y]=flts(u,sl [,past])
```

Parameters

u
matrix (input vector)

sl
list (linear system `syslin`)

x0
vector (initial state ; default value=0)

past
matrix (of the past ; default value=0)

x,y
matrices (state and output)

Description

- State-space form:

`sl` is a discrete linear system given by its state space representation (see `syslin`):

`sl=syslin('d',A,B,C,D):`

```
x[t+1] = A x[t] + B u[t]  
y[t] = C x[t] + D u[t]
```

or, more generally, if `D` is a polynomial matrix ($p = \text{degree}(D(z))$):

```
D(z) = D_0 + z D_1 + z^2 D_2 + .. + z^p D_p  
y[t] = C x[t] + D_0 u[t] + D_1 u[t+1] + .. + D_p u[t+p]
```

- Transfer form:

`y=flts(u,sl[,past])`. Here `sl` is a linear system in transfer matrix representation i.e

`sl=syslin('d',transfer_matrix)` (see `syslin`).

```
past = [u      , ..., u      ]  
        [ -nd      -1]  
        [y      , ..., y      ]  
        [ -nd      -1]
```

is the matrix of past values of `u` and `y`.

`nd` is the maximum of degrees of lcm's of each row of the denominator matrix of `sl`.

```

u=[u0 u1 ... un]   (input)
y=[y0 y1 ... yn]   (output)

```

p is the difference between maximum degree of numerator and maximum degree of denominator

Examples

```

sl=syslin('d',1,1,1);u=1:10;
y=flts(u,sl);
plot2d(y)
[y1,x1]=flts(u(1:5),sl);y2=flts(u(6:10),sl,x1);
y-[y1,y2]

//With polynomial D:
z=poly(0,'z');
D=1+z+z^2; p =degree(D);
sl=syslin('d',1,1,1,D);
y=flts(u,sl);[y1,x1]=flts(u(1:5),sl);
y2=flts(u(5-p+1:10),sl,x1); // (update)
y-[y1,y2]

//Delay (transfer form): flts(u,1/z)
// Usual responses
z=poly(0,'z');
h=syslin(0.1,(1-2*z)/(z^2+0.3*z+1))
imprep=flts(eye(1,20),tf2ss(h)); //Impulse response
clf();
plot(imprep,'b')
u=ones(1,20);
stprep=flts(ones(1,20),tf2ss(h)); //Step response
plot(stprep,'g')

// Other examples
A=[1 2 3;0 2 4;0 0 1];B=[1 0;0 0;0 1];C=eye(3,3);Sys=syslin('d',A,B,C);
H=ss2tf(Sys); u=[1;-1]*(1:10);
//
yh=flts(u,H); ys=flts(u,Sys);
norm(yh-ys,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-ys,1)
//
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4);yh(:,2:4)]);
norm([yh1,yh2]-yh,1)
//with D<>0
D=[-3 8;4 -0.5;2.2 0.9];
Sys=syslin('d',A,B,C,D);
H=ss2tf(Sys); u=[1;-1]*(1:10);
rh=flts(u,H); rs=flts(u,Sys);
norm(rh-rs,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-rs,1)
//With H:
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4); yh1(:,2:4)]);

```

```
norm([yh1,yh2]-rh)
```

See Also

ltitr, dsimul, rtitr

Name

fourplan — augmented plant to four plants

```
[P11,P12,P21,P22]=fourplan(P,r)
```

Parameters

P
syslin list (linear system)

r
1x2 row vector, dimension of P22

P11,P12,P21,P22
syslin lists.

Description

Utility function.

P being partitioned as follows:

```
P=[ P11 P12;  
    P21 P22]
```

with `size(P22)=r` this function returns the four linear systems P11,P12,P21,P22.

See Also

lqg , lqg2stan , lqr , lqe , lft

Name

frep2tf — transfer function realization from frequency response

```
[h [,err]]=frep2tf(frq,repf,dg [,dom,tols,weight])
```

Parameters

freq

vector of frequencies in Hz.

repf

vector of frequency response

dg

degree of linear system

dom

time domain ('c' or 'd' or dt)

tols

a vector of size 3 giving the relative and absolute tolerance and the maximum number of iterations (default values are `rtol=1.e-2; atol=1.e-4, N=10`).

weight

vector of weights on frequencies

h

SISO transfer function

err

error (for example if `dom='c'` `sum(abs(h(2i*pi*freq) - rep)^2)/size(freq,*)`)

Description

Frequency response to transfer function conversion. The order of `h` is a priori given in `dg` which must be provided. The following linear system is solved in the least square sense.

```
weight(k)*(n( phi_k) - d(phi_k)*rep_k)=0, k=1,...,n
```

where `phi_k= 2*i*pi*freq` when `dom='c'` and `phi_k=exp(2*i*pi*dom*freq)` if not. If the `weight` vector is not given a default penalization is used (when `dom='c'`).

A stable and minimum phase system can be obtained by using function `factors`.

Examples

```
s=poly(0,'s');
h=syslin('c',(s-1)/(s^3+5*s+20))
freq=0:0.05:3;repf=repfreq(h,freq);
clean(frep2tf(freq,repf,3))

Sys=ssrand(1,1,10);
freq=logspace(-3,2,200);
[frq,rep]=repfreq(Sys,freq); //Frequency response of Sys
```

```
[Sys2,err]=frep2tf(frq,rep,10);Sys2=clean(Sys2)//Sys2 obtained from freq. resp  
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2  
clf();bode(frq,[rep;rep2]) //Responses of Sys and Sys2  
[gsort(spec(Sys('A'))),gsort(roots(Sys2('den')))] //poles  
  
dom=1/1000; // Sampling time  
z=poly(0,'z');  
h=syslin(dom,(z^2+0.5)/(z^3+0.1*z^2-0.5*z+0.08))  
frq=(0:0.01:0.5)/dom;repf=repfreq(h,frq);  
[Sys2,err]=frep2tf(frq,repf,3,dom);  
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2  
clf();plot2d1("onn",frq',abs([repf;rep2]'));
```

See Also

imrep2ss , arl2 , time_id , armax , frfit

Name

freq — frequency response

```
[x]=freq(A,B,C [ ,D],f)
[x]=freq(NUM,DEN,f)
```

Parameters

A, B, C, D

real matrices of respective dimensions $n \times n$, $n \times p$, $m \times n$, $m \times p$.

NUM,DEN

polynomial matrices of dimension $m \times p$

x

real or complex matrix

Description

$x = \text{freq}(A, B, C [, D], f)$ returns a real or complex $m \times p \times t$ matrix such that:

$$x(:, k*p:(k+1)*p) = C \cdot \text{inv}(f(k) \cdot \text{eye}() - A) \cdot B + D.$$

Thus, for f taking values along the imaginary axis or on the unit circle x is the continuous or discrete time frequency response of (A, B, C, D) .

$x = \text{freq}(\text{NUM}, \text{DEN}, f)$ returns a real or complex matrix x such that columns $k*(p-1)+1$ to $k*p$ of x contain the matrix $\text{NUM}(f(k)) ./ \text{DEN}(f(k))$

Examples

```
s=poly(0, 's');
sys=(s+1)/(s^3-5*s+4)
rep=freq(sys("num"), sys("den"), [0, 0.9, 1.1, 2, 3, 10, 20])
[horner(sys, 0), horner(sys, 20)]
//
Sys=tf2ss(sys);
[A, B, C, D]=abcd(Sys);
freq(A, B, C, [0, 0.9, 1.1, 2, 3, 10, 20])
```

See Also

repfreq, horner

Name

freson — peak frequencies

```
fr=freson(h)
```

Parameters

h

syslin list

fr

vector of peak frequencies in Hz

Description

returns the vector of peak frequencies in Hz for the SISO plant h

Examples

```
h=syslin('c', -1+%s, (3+2*%s+%s^2)*(50+0.1*%s+%s^2))
fr=freson(h)
bode(h)
g=20*log(abs(repfreq(h, fr)))/log(10)
```

See Also

frep2tf , zgrid , h_norm

Name

fspecg — stable factorization

```
[gm]=fspecg(g) .
```

Parameters

g, gm
syslin lists (linear systems in state-space representation)

Description

returns gm with gm and gm^{-1} stable such that:

```
gtild(g)*g = gtild(gm)*gm
```

g and gm are continuous-time linear systems in state-space form.

Imaginary-axis poles are forbidden.

Name

fstabst — Youla's parametrization

```
[J]=fstabst(P,r)
```

Parameters

P

syslin list (linear system)

r

1x2 row vector, dimension of P22

J

syslin list (linear system in state-space representation)

Description

Parameterization of all stabilizing feedbacks.

P is partitioned as follows:

```
P=[ P11 P12;
    P21 P22]
```

(in state-space or transfer form: automatic conversion in state-space is done for the computations)

r = size of P22 subsystem, (2,2) block of P

```
J =[ J11 J12;
    J21 J22]
```

K is a stabilizing controller for P (i.e. P22) iff $K = \text{lft}(J, r, Q)$ with Q stable.

The central part of J , J11 is the lqg regulator for P

This J is such that defining T as the 2-port lft of P and J : $[T, rt] = \text{lft}(P, r, J, r)$ one has that T12 is inner and T21 is co-inner.

Examples

```
ny=2;nu=3;nx=4;
P22=ssrand(ny,nu,nx);
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';
[P,r]=lqg2stan(P22,bigQ,bigR);
J=fstabst(P,r);
Q=ssrand(nu,ny,1);Q('A')=-1; //Stable Q
K=lft(J,r,Q);
A=h_cl(P,r,K); spec(A)
```

See Also

obscont , lft , lqg , lqg2stan

Name

`g_margin` — gain margin and associated crossover frequency

```
gm=g_margin(h)
[gm,fr]=g_margin(h)
```

Parameters

`h`

a SISO linear system (see `:syslin`).

`gm`

a number, the gain margin (in dB) if any of `Inf`

`fr`

a number, the associated frequency in hertz, or an empty matrix if the gain margin does not exist.

Description

Given a SISO linear system in continuous or discrete time, `g_margin` returns `gm`, the gain margin in dB of `h` and `fr`, the achieved corresponding frequency in hz.

The gain margin, if it exists, is the minimal value of the system gain at points where the nyquist plot crosses the negative real axis. In other words the gain margin is $20 \cdot \log_{10}(1/g)$ where `g` is the open loop gain of `h` when the frequency response phase of `h` equals -180°

The algorithm uses polynomial root finder to solve the equations:

$h(s)=h(-s)$

for the continuous time case.

$h(z)=h(1/z)$

for the discrete time case.

Examples

```
h=syslin('c',-1+%s,3+2*%s+%s^2) //continuous time case
[g,fr]=g_margin(h)
[g,fr]=g_margin(h-10)
nyquist(h-10)

h = syslin(0.1,0.04798*%z+0.0464,%z^2-1.81*%z+0.9048); //discrete time case
[g ,fr]=g_margin(h);
show_margins(h)
```

See Also

`p_margin`, `show_margins`, `repfreq`, `black`, `bode`, `chart`, `nyquist`

Authors

Serge Steer, INRIA

Name

gainplot — magnitude plot

```
gainplot(sl,fmin,fmax [,step] [,comments] )  
gainplot(frq,db,phi [,comments])  
gainplot(frq, repf [,comments])
```

Parameters

sl
list (sys lin SIMO linear system).

fmin,fmax
real scalars (frequency interval).

step
real (discretization step (logarithmic scale))

comments
string

freq
matrix (row by row frequencies)

db,phi
matrices (magnitudes and phases corresponding to **freq**)

repf
complex matrix. One row for each frequency response.

Description

Same as Bode but plots only the magnitude.

Examples

```
s=poly(0,'s')  
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))  
gainplot(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')  
clf()  
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))  
gainplot([h1;h],0.01,100,['h1';'h'])
```

See Also

bode , black , nyquist , freq , repfreq , g_margin , p_margin

Name

gamitg — H-infinity gamma iterations

```
[gopt]=gamitg(G,r,prec [,options]);
```

Parameters

G
sys lin list (plant realization)

r
1x2 row vector (dimension of G22)

prec
desired relative accuracy on the norm

option
string 't'

gopt
real scalar, optimal H-infinity gain

Description

`gopt=gamitg(G,r,prec [,options])` returns the H-infinity optimal gain `gopt`.

G contains the state-space matrices $[A, B, C, D]$ of the plant with the usual partitions:

$$B = \begin{pmatrix} B1 & B2 \end{pmatrix}, \quad C = \begin{pmatrix} C1 \\ C2 \end{pmatrix}, \quad D = \begin{pmatrix} D11 & D12 \\ D21 & D22 \end{pmatrix}$$

These partitions are implicitly given in **r**: `r(1)` and `r(2)` are the dimensions of **D22** (rows x columns)

With `option='t'`, `gamitg` traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

See Also

`ccontrg`, `h_inf`

Authors

P. Gahinet

Name

gcare — control Riccati equation

```
[X,F]=gcare(Sl)
```

Parameters

Sl
linear system (syslin list)

X
symmetric matrix

F
real matrix

Description

Generalized Control Algebraic Riccati Equation (GCARE). X = solution , F = gain.

The GCARE for $Sl=[A,B,C,D]$ is:

$$(A-B*Si*D'*C)'*X+X*(A-B*Si*D'*C)-X*B*Si*B'*X+C'*Ri*C=0$$

where $S=(eye()+D'*D)$, $Si=inv(S)$, $R=(eye()+D*D')$, $Ri=inv(R)$ and $F=-Si*(D'*C+B'*X)$ is such that $A+B*F$ is stable.

See Also

gfare

Name

gfare — filter Riccati equation

```
[Z,H]=gfare(Sl)
```

Parameters

Sl
linear system (syslin list)

Z
symmetric matrix

H
real matrix

Description

Generalized Filter Algebraic Riccati Equation (GFARE). Z = solution, H = gain.

The GFARE for $Sl=[A, B, C, D]$ is:

$$(A-B*D'*R_i*C)*Z+Z*(A-B*D'*R_i*C)'-Z*C'*R_i*C*Z+B*S_i*B'=0$$

where $S=(eye()+D'*D)$, $S_i=inv(S)$, $R=(eye()+D*D')$, $R_i=inv(R)$ and $H=-(B*D'+Z*C')*R_i$ is such that $A+H*C$ is stable.

See Also

gcare

Name

gfrancis — Francis equations for tracking

```
[L,M,T]=gfrancis(Plant,Model)
```

Parameters

Plant
syslin list

Model
syslin list

L,M,T
real matrices

Description

Given the the linear plant:

$$\begin{aligned}x' &= F*x + G*u \\ y &= H*x + J*u\end{aligned}$$

and the linear model

$$\begin{aligned}x_m' &= A*x_m + B*u_m \\ y_m &= C*x_m + D*u_m\end{aligned}$$

the goal is for the plant to track the model i.e. $e = y - y_m \rightarrow 0$ while keeping stable the state $x(t)$ of the plant. u is given by feedforward and feedback

$$u = L*x_m + M*u_m + K*(x - T*x_m) = [K \quad L - K*T] * (x, x_m) + M*u_m$$

The matrices T,L,M satisfy generalized Francis equations

$$\begin{aligned}F*T + G*L &= T*A \\ H*T + J*L &= C \\ G*M &= T*B \\ J*M &= D\end{aligned}$$

The matrix K must be chosen as stabilizing the pair (F, G) See example of use in directory `demo/tracking`.

Examples

```
Plant=ssrand(1,3,5);  
[F,G,H,J]=abcd(Plant);  
nw=4;nuu=2;A=rand(nw,nw);
```

```
st=maxi(real(spec(A)));A=A-st*eye(A);  
B=rand(nw,nuu);C=2*rand(1,nw);D=0*rand(C*B);  
Model=syslin('c',A,B,C,D);  
[L,M,T]=gfrancis(Plant,Model);  
norm(F*T+G*L-T*A,1)  
norm(H*T+J*L-C,1)  
norm(G*M-T*B,1)  
norm(J*M-D,1)
```

See Also

lqg , ppol

Name

gtild — tilde operation

```
Gt=gtild(G)
Gt=gtild(G,flag)
```

Parameters

G
either a polynomial or a linear system (syslin list) or a rational matrix

Gt
same as G

flag
character string: either 'c' or 'd' (optional parameter).

Description

If G is a polynomial matrix (or a polynomial), $G_t = \text{gtild}(G, 'c')$ returns the polynomial matrix $G_t(s) = G(-s)'$.

If G is a polynomial matrix (or a polynomial), $G_t = \text{gtild}(G, 'd')$ returns the polynomial matrix $G_t = G(1/z) * z^n$ where n is the maximum degree of G.

For continuous-time systems represented in state-space by a syslin list, $G_t = \text{gtild}(G, 'c')$ returns a state-space representation of $G(-s)'$ i.e the ABCD matrices of G_t are $A', -C', B', D'$. If G is improper ($D = D(s)$) the D matrix of G_t is $D(-s)'$.

For discrete-time systems represented in state-space by a syslin list, $G_t = \text{gtild}(G, 'd')$ returns a state-space representation of $G(-1/z)'$ i.e the (possibly improper) state-space representation of $-z * C * \text{inv}(z * A - B) * C + D(1/z)$.

For rational matrices, $G_t = \text{gtild}(G, 'c')$ returns the rational matrix $G_t(s) = G(-s)$ and $G_t = \text{gtild}(G, 'd')$ returns the rational matrix $G_t(z) = G(1/z)'$.

The parameter flag is necessary when gtild is called with a polynomial argument.

Examples

```
//Continuous time
s=poly(0,'s');G=[s,s^3;2+s^3,s^2-5]
Gt=gtild(G,'c')
Gt-horner(G,-s)' //continuous-time interpretation
Gt=gtild(G,'d');
Gt-horner(G,1/s) '*s^3 //discrete-time interpretation
G=ssrand(2,2,3);Gt=gtild(G); //State-space (G is cont. time by default)
clean((horner(ss2tf(G),-s))'-ss2tf(Gt)) //Check

// Discrete-time
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss('dt')='d'; //discrete-time
Gss(5)=[1,2;0,1]; //With a constant D matrix
G=ss2tf(Gss);Gt1=horner(G,1/z)';
Gt=gtild(Gss);
Gt2=clean(ss2tf(Gt)); clean(Gt1-Gt2) //Check
```

```
//Improper systems
z=poly(0, 'z');
Gss=ssrand(2,2,3);Gss(7)='d'; //discrete-time
Gss(5)=[z,z^2;1+z,3]; //D(z) is polynomial
G=ss2tf(Gss);Gt1=horner(G,1/z)'; //Calculation in transfer form
Gt=gtild(Gss); //..in state-space
Gt2=clean(ss2tf(Gt));clean(Gt1-Gt2) //Check
```

See Also

syslin , horner , factors

Name

h2norm — H2 norm

```
[n]=h2norm(Sl [,tol])
```

Parameters

Sl
linear system (syslin list)

n
real scalar

Description

produces the H2 norm of a linear continuous time system Sl.

(For Sl in state-space form h2norm uses the observability gramian and for Sl in transfer form h2norm uses a residue method)

Name

`h_cl` — closed loop matrix

```
[Ac_l]=h_cl(P,r,K)
[Ac_l]=h_cl(P22,K)
```

Parameters

`P`, `P22`

linear system (`syslin` list), augmented plant or nominal plant respectively

`r`

1x2 row vector, dimensions of 2,2 part of `P` (`r=[rows,cols]=size(P22)`)

`K`

linear system (`syslin` list), controller

`Ac_l`

real square matrix

Description

Given the standard plant `P` (with `r=size(P22)`) and the controller `K`, this function returns the closed loop matrix `Ac_l`.

The poles of `Ac_l` must be stable for the internal stability of the closed loop system.

`Ac_l` is the A-matrix of the linear system $[I \ -P22; -K \ I]^{-1}$ i.e. the A-matrix of `lft(P,r,K)`

See Also

`lft`

Authors

F. D.

Name

`h_inf` — H-infinity (central) controller

```
[Sk,ro]=h_inf(P,r,romin,romax,nmax)
[Sk,rk,ro]=h_inf(P,r,romin,romax,nmax)
```

Parameters

`P`

`syslin` list : continuous-time linear system ("augmented" plant given in state-space form or in transfer form)

`r`

size of the P22 plant i.e. 2-vector `[#outputs,#inputs]`

`romin,romax`

a priori bounds on `ro` with `ro=1/gama^2`; (`romin=0` usually)

`nmax`

integer, maximum number of iterations in the gama-iteration.

Description

`h_inf` computes H-infinity optimal controller for the continuous-time plant `P`.

The partition of `P` into four sub-plants is given through the 2-vector `r` which is the size of the 22 part of `P`.

`P` is given in state-space e.g. `P=syslin('c',A,B,C,D)` with `A,B,C,D` = constant matrices or `P=syslin('c',H)` with `H` a transfer matrix.

`[Sk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` in `[romin,romax]` and the central controller `Sk` in the same representation as `P`.

(All calculations are made in state-space, i.e conversion to state-space is done by the function, if necessary).

Invoked with three LHS parameters,

`[Sk,rk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` and the Parameterization of all stabilizing controllers:

a stabilizing controller `K` is obtained by `K=lft(Sk,r,PHI)` where `PHI` is a linear system with dimensions `r'` and satisfy:

`H_norm(PHI) < gamma`. `rk` (`=r`) is the size of the `Sk22` block and `ro = 1/gama^2` after `nmax` iterations.

Algorithm is adapted from Safonov-Limebeer. Note that `P` is assumed to be a continuous-time plant.

See Also

`gamitg`, `ccontrg`, `leqr`

Authors

F.Delebecque INRIA (1990)

Name

`h_inf_st` — static H_infinity problem

```
[Kopt,gamaopt]=h_inf_stat(D,r)
```

Parameters

`D`
real matrix

`r`
1x2 vector

`Kopt`
matrix

Description

computes a matrix `Kopt` such that largest singular value of:

$\text{lft}(D, r, K) = D_{11} + D_{12} * K * \text{inv}(I - D_{22} * K) * D_{21}$ is minimal (Static H_infinity four blocks problem).

`D` is partitionned as `D=[D11 D12; D21 D22]` where `size(D22)=r=[r1 r2]`

Authors

F.D. ;

Name

h_norm — H-infinity norm

```
[hinfnorm [,frequency]]=h_norm(sl [,rerr])
```

Parameters

sl
the state space system (sys lin list)

rerr
max. relative error, default value `1e-8`

hinfnorm
the infinity norm of `Sl`

frequency
frequency at which maximum is achieved

Description

produces the infinity norm of a state-space system (the maximum over all frequencies of the maximum singular value).

See Also

linfn , linf , svplot

Name

hankelsv — Hankel singular values

```
[nk2,W]=hankelsv(sl [,tol])  
[nk2]=hankelsv(sl [,tol])
```

Parameters

sl
syslin list representing the linear system (state-space).

tol
tolerance parameter for detecting imaginary axis modes (default value is $1000*\%eps$).

Description

returns nk2, the squared Hankel singular values of sl and $W = P*Q$ = controllability gramian times observability gramian.

nk2 is the vector of eigenvalues of W.

Examples

```
A=diag([-1,-2,-3]);  
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)  
[Q,M]=pbig(W,nk2(2)-%eps,'c');  
slr=projsl(sl,Q,M);hankelsv(slr)
```

See Also

balreal , equil , equil1

Name

hinf — H_infinity design of continuous-time systems

```
[AK, BK, CK, DK, (RCOND)] = hinf(A, B, C, D, ncon, nmeas, gamma)
```

Parameters

A

the n-by-n system state matrix A.

B

the n-by-m system input matrix B.

C

the p-by-n system output matrix C.

D

the p-by-m system matrix D.

ncon

the number of control inputs. $m \geq ncon \geq 0$, $p - nmeas \geq ncon$.

nmeas

the number of measurements. $p \geq nmeas \geq 0$, $m - ncon \geq nmeas$.

gamma

the parameter gamma used in H_infinity design. It is assumed that gamma is sufficiently large so that the controller is admissible. $gamma \geq 0$.

AK

the n-by-n controller state matrix AK.

BK

the n-by-nmeas controller input matrix BK.

CK

the ncon-by-n controller output matrix CK.

DK

the ncon-by-nmeas controller matrix DK.

RCOND

a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)

RCOND

(1) contains the reciprocal condition number of the control transformation matrix TU,

RCOND

(2) contains the reciprocal condition number of the measurement transformation matrix TY,

RCOND

(3) contains an estimate of the reciprocal condition number of the X-Riccati equation,

RCOND

(4) contains an estimate of the reciprocal condition number of the Y-Riccati equation.

Description

$[AK, BK, CK, DK, (RCOND)] = \text{hinf}(A, B, C, D, ncon, nmeas, \gamma)$ To compute the matrices of an H-infinity (sub)optimal n-state controller

$$K = \begin{bmatrix} | & AK & | & BK & | \\ \hline & & & & \\ | & CK & | & DK & | \end{bmatrix},$$

for the continuous-time system

$$P = \begin{bmatrix} | & A & | & B1 & B2 & | \\ \hline & & & & & \\ | & C1 & | & D11 & D12 & | \\ | & C2 & | & D21 & D22 & | \end{bmatrix} = \begin{bmatrix} | & A & | & B & | \\ \hline & & & & \\ | & C & | & D & | \end{bmatrix},$$

and for a given value of gamma, where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of continuous-time linear control systems. Report98-14, Department of Engineering, Leicester University, August 1998.

Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[ -1  0  4  5 -3 -2
    -2  4 -7 -2  0  3
    -6  9 -5  0  2 -1
    -8  4  7 -1 -3  0
     2  5  8 -9  1 -4
     3 -5  8  0  2 -6];

B=[ -3 -4 -2  1  0
     2  0  1 -5  2
    -5 -7  0  7 -2
     4 -6  1  1 -2
    -3  9 -8  0  5
     1 -2  3 -6 -2];

C=[ 1 -1  2 -4  0 -3
    -3  0  5 -1  1  1
    -7  5  0 -8  2 -2
     9 -3  4  0  3  7
     0  1 -2  1 -6 -2];

D=[ 1 -2 -3  0  0
     0  4  0  1  0
     5 -3 -4  0  1
```

```
    0  1  0  1 -3  
    0  0  1  7  1];  
Gamma=10.18425636157899;  
[AK,BK,CK,DK] = hinf(A,B,C,D,2,2,Gamma)
```

See Also

dhinf

Name

imrep2ss — state-space realization of an impulse response

```
[sl]=imrep2ss(v [,deg])
```

Parameters

v
vector coefficients of impulse response, $v(:,k)$ is the k th sample

deg
integer (order required)

sl
syslin list

Description

Impulse response to linear system conversion (one input). **v** must have an even number of columns.

Examples

```
s=poly(0,'s');  
H=[1/(s+0.5);2/(s-0.4)] //strictly proper  
np=20;w=ldiv(H('num'),H('den'),np);  
rep=[w(1:np)';w(np+1:2*np)']; //The impulse response  
H1=ss2tf(imrep2ss(rep))  
z=poly(0,'z');  
H=(2*z^2-3.4*z+1.5)/(z^2-1.6*z+0.8) //Proper transfer function  
u=zeros(1,20);u(1)=1;  
rep=rtitr(H('num'),H('den'),u); //Impulse rep.  
// <=> rep=ldiv(H('num'),H('den'),20)  
w=z*imrep2ss(rep) //Realization with shifted impulse response  
// i.e strictly proper to proper  
H2=ss2tf(w);
```

See Also

freq2tf , arl2 , time_id , armax , markp2ss , ldiv

Name

inistate — Estimates the initial state of a discrete-time system

```
X0 = inistate(SYS,Y,U,TOL,PRINTW)
X0 = inistate(A,B,C,Y,U);
X0 = inistate(A,C,Y);

[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW)
```

Parameters

SYS

given system, syslin(dt,A,B,C,D)

Y

the output of the system

U

the input of the system

TOL

TOL is the tolerance used for estimating the rank of matrices. If $TOL > 0$, then the given value of TOL is used as a lower bound for the reciprocal condition number.

Default: $\text{prod}(\text{size}(\text{matrix})) * \text{epsilon\_machine}$ where `epsilon_machine` is the relative machine precision.

PRINTW

PRINTW is a switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: PRINTW = 0.

X0

the estimated initial state vector

V

orthogonal matrix which reduces the system state matrix A to a real Schur form

rcnd

estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

Description

inistate Estimates the initial state of a discrete-time system, given the (estimated) system matrices, and a set of input/output data.

`X0 = inistate(SYS,Y,U,TOL,PRINTW)` estimates the initial state X0 of the discrete-time system `SYS = (A,B,C,D)`, using the output data Y and the input data U. The model structure is :

$$x(k+1) = Ax(k) + Bu(k), \quad k \geq 1,$$

$$y(k) = Cx(k) + Du(k),$$

The vectors $y(k)$ and $u(k)$ are transposes of the k -th rows of Y and U , respectively.

Instead of the first input parameter `SYS` (an `syslin` object), equivalent information may be specified using matrix parameters, for instance, `X0 = inistate(A,B,C,Y,U)`; or `X0 = inistate(A,C,Y)`;

`[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW)` returns, besides `x0`, the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as an estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

See Also

`findBD` , `findx0BD`

Name

invsyslin — system inversion

```
[sl2]=invsyslin(sl1)
```

Parameters

sl1,sl2

syslin lists (linear systems in state space representation)

Description

Utility function. Computes the state form of the inverse `sl2` of the linear system `sl1` (which is also given in state form).

The D-matrix is supposed to be full rank. Old stuff used by `inv(S)` when `S` is a `syslin` list.

See Also

rowregul , inv

Name

kpure — continuous SISO system limit feedback gain

```
K=kpure(sys [,tol])  
[K,R]=kpure(sys [,tol])
```

Parameters

- sys**
SISO linear system (syslin)
- tol**
a positive scalar. tolerance used to determine if a root is imaginary or not. The default value is $1e-6$
- K**
Real vector, the vector of gains for which at least one closed loop pole is imaginary.
- R**
Complex vector, the imaginary closed loop poles associated with the values of K.

Description

`K=kpure(sys)` computes the gains K such that the system `sys` feedback by `K(i)` (`sys/.K(i)`) has poles on imaginary axis.

Examples

```
num=real(poly([-1+%i, -1-%i, -1+8*i -1-8*i], 's'));  
den=real(poly([0.5 0.5 -6+7*i -6-7*i -3 -7 -11], 's'));  
h=num/den;  
  
[K,Y]=kpure(h)  
clf();evans(h)  
plot(real(Y),imag(Y),'+r')  
  
num=real(poly([-1+%i*1, -1-%i*1, 2+%i*8 2-%i*8 -2.5+%i*13 -2.5-%i*13], 's'));  
den=real(poly([1 1 3+%i*3 3-%i*3 -15+%i*7 -15-%i*7 -3 -7 -11], 's'));  
h=num/den;  
  
[K,Y]=kpure(h)  
clf();evans(h,100000)  
plot(real(Y),imag(Y),'+r')
```

Author

Serge Steer, INRIA

See Also

evans , krac2

Name

krac2 — continuous SISO system limit feedback gain

```
g=krac2(sys)
```

Parameters

sys
SISO linear system (syslin)

g
constant

Description

krac2(sys) computes the gains g such that the system sys feedback by g (sys/.g) has 2 real equal poles.

Examples

```
h=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
clf();evans(h,100)
g=krac2(h)
hf1=h/.g(1);roots(denom(hf1))
hf2=h/.g(2);roots(denom(hf2))
```

See Also

evans , kpure

Name

lcf — normalized coprime factorization

```
[N,M]=lcf(sl)
```

Parameters

sl
linear system given in state space or transfer function (syslin list)

N,M
two linear systems (syslin list)

Description

Computes normalized coprime factorization of the linear dynamic system `sl`.

$$sl = M^{-1} N$$

Authors

F. D.; ;

Name

leqr — H-infinity LQ gain (full state)

```
[K,X,err]=leqr(P12,Vx)
```

Parameters

P12

syslin list

Vx

symmetric nonnegative matrix (should be small enough)

K,X

two real matrices

err

a real number (l1 norm of LHS of Riccati equation)

Description

leqr computes the linear suboptimal H-infinity LQ full-state gain for the plant $P12=[A, B2, C1, D12]$ in continuous or discrete time.

P12 is a syslin list (e.g. $P12=syslin('c', A, B2, C1, D12)$).

$$\begin{bmatrix} C1' \\ [\\ D12' \end{bmatrix} * \begin{bmatrix} C1 & D12 \end{bmatrix} = \begin{bmatrix} Q & S \\ [& \\ S' & R \end{bmatrix}$$

Vx is related to the variance matrix of the noise w perturbing x; (usually $Vx=gama^{-2}*B1*B1'$).

The gain K is such that $A + B2*K$ is stable.

X is the stabilizing solution of the Riccati equation.

For a continuous plant:

$$(A-B2*inv(R)*S')'*X+X*(A-B2*inv(R)*S')-X*(B2*inv(R)*B2'-Vx)*X+Q-S*inv(R)*S'=0$$

$$K=-inv(R)*(B2'*X+S)$$

For a discrete time plant:

$$X-(Abar'*inv((inv(X)+B2*inv(R)*B2'-Vx))*Abar+Qbar=0$$

$$K=-inv(R)*(B2'*inv(inv(X)+B2*inv(R)*B2'-Vx)*Abar+S')$$

with $Abar=A-B2*inv(R)*S'$ and $Qbar=Q-S*inv(R)*S'$

The 3-blocks matrix pencils associated with these Riccati equations are:

discrete							continuous								
z	I	-Vx	0	-	A	0	B2	s	I	0	0	-	A	Vx	B2
	0	A'	0	-	-Q	I	-S		0	I	0	-	-Q	-A'	-S
	0	B2'	0	-	S'	0	R		0	0	0	-	S'	-B2'	R

See Also

lqr

Authors

F.D.;

Name

lft — linear fractional transformation

```
[P1]=lft(P,K)
[P1]=lft(P,r,K)
[P1,r1]=lft(P,r,Ps,rs)
```

Parameters

- P**
linear system (`syslin` list), the ``augmented" plant, implicitly partitioned into four blocks (two input ports and two output ports).
- K**
linear system (`syslin` list), the controller (possibly an ordinary gain).
- r**
1x2 row vector, dimension of P22
- Ps**
linear system (`syslin` list), implicitly partitioned into four blocks (two input ports and two output ports).
- rs**
1x2 row vector, dimension of Ps22

Description

Linear fractional transform between two standard plants **P** and **PS** in state space form or in transfer form (`syslin` lists).

`r= size(P22) rs=size(P22s)`

`lft(P,r, K)` is the linear fractional transform between **P** and a controller **K** (**K** may be a gain or a controller in state space form or in transfer form);

`lft(P,K)` is `lft(P,r,K)` with `r=size of K transpose`;

$P1 = P11 + P12 * K * (I - P22 * K)^{-1} * P21$

`[P1,r1]=lft(P,r,Ps,rs)` returns the generalized (2 ports) lft of **P** and **PS**.

P1 is the pair two-port interconnected plant and the partition of **P1** into 4 blocks is given by **r1** which is the dimension of the 22 block of **P1**.

P and **R** can be PSSDs i.e. may admit a polynomial **D** matrix.

Examples

```
s=poly(0,'s');
P=[1/s, 1/(s+1); 1/(s+2),2/s]; K= 1/(s-1);
lft(P,K)
lft(P,[1,1],K)
P(1,1)+P(1,2)*K*inv(1-P(2,2)*K)*P(2,1) //Numerically dangerous!
ss2tf(lft(tf2ss(P),tf2ss(K)))
lft(P,-1)
```

```
f=[0,0;0,1];w=P/.f; w(1,1)
//Improper plant (PID control)
W=[1,1;1,1/(s^2+0.1*s)];K=1+1/s+s
lft(W,[1,1],K); ss2tf(lft(tf2ss(W),[1,1],tf2ss(K)))
```

See Also

sensi , augment , feedback , sysdiag

Name

lin — linearization

```
[A,B,C,D]=lin(sim,x0,u0)
[sl]=lin(sim,x0,u0)
```

Parameters

sim

function

x0, u0

vectors of compatible dimensions

A,B,C,D

real matrices

sl

syslin list

Description

linearization of the non-linear system $[y, \dot{x}] = \text{sim}(x, u)$ around x_0, u_0 .

`sim` is a function which computes y and $\dot{x}$.

The output is a linear system (syslin list) `sl` or the four matrices (A, B, C, D)

For example, if `ftz` is the function passed to ode e.g.

```
[zd]=ftz(t,z,u)
```

and if we assume that $y=x$

```
[z]=ode(x0,t0,tf,list(ftz,u) compute x(tf).
```

If `simula` is the following function:

```
deff(' [y,xdot]=simula(x,u)', 'xd=ftz(tf,x,u); y=x;');
```

the tangent linear system `sl` can be obtained by:

```
[A,B,C,D]=lin(simula,z,u)
sl = syslin('c',A,B,C,D,x0)
```

Examples

```
deff(' [y,xdot]=sim(x,u)', 'xdot=[u*sin(x); -u*x^2]; y=xdot(1)+xdot(2)')
sl=lin(sim,1,2);
```

See Also

external , derivat

Name

linf — infinity norm

```
linf(g [,eps],[tol])
```

Parameters

`g`
is a `syslin` linear system.

`eps`
is error tolerance on `n`.

`tol`
threshold for imaginary axis poles.

Description

returns the L_∞ norm of `g`.

```
n=sup_w [sigmax(g(jw))]
```

(sigmax largest singular value).

See Also

`h_norm` , `linfn`

Name

linfn — infinity norm

```
[x,freq]=linfn(G,PREC,RELTOL,options);
```

Parameters

G
is a `syslin` list

PREC
desired relative accuracy on the norm

RELTOL
relative threshold to decide when an eigenvalue can be considered on the imaginary axis.

options
available options are 'trace' or 'cond'

x
is the computed norm.

freq
vector

Description

Computes the Linf (or Hinf) norm of **G**. This norm is well-defined as soon as the realization $G=(A, B, C, D)$ has no imaginary eigenvalue which is both controllable and observable.

freq is a list of the frequencies for which $\|G\|$ is attained, i.e., such that $\|G(j\omega)\| = \|G\|$.

If -1 is in the list, the norm is attained at infinity.

If -2 is in the list, **G** is all-pass in some direction so that $\|G(j\omega)\| = \|G\|$ for all frequencies ω .

The algorithm follows the paper by G. Robel (AC-34 pp. 882-884, 1989). The case $D=0$ is not treated separately due to superior accuracy of the general method when (A, B, C) is nearly non minimal.

The 'trace' option traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

The 'cond' option estimates a confidence index on the computed value and issues a warning if computations are ill-conditioned

In the general case (**A** neither stable nor anti-stable), no upper bound is prespecified.

If by contrast **A** is stable or anti stable, lower and upper bounds are computed using the associated Lyapunov solutions.

See Also

`h_norm`

Authors

P. Gahinet

Name

linmeq — Sylvester and Lyapunov equations solver

```
[X(, sep)] = linmeq(task,A,(B,)C,flag,trans(,schur))
```

Parameters

task

integer option to determine the equation type:

=1

solve the Sylvester equation (1a) or (1b);

=2

solve the Lyapunov equation (2a) or (2b);

=3

solve for the Cholesky factor op(X) the Lyapunov equation (3a) or (3b).

A

real matrix

B

real matrix

C

real matrix

flag

(optional) integer vector of length 3 or 2 containing options.

task

= 1 : flag has length 3

flag(1)

= 0 : solve the continuous-time equation (1a); otherwise, solve the discrete-time equation (1b).

flag(2)

= 1 : A is (quasi) upper triangular;

flag(2)

= 2 : A is upper Hessenberg;

otherwise

A is in general form.

flag(3)

= 1 : B is (quasi) upper triangular;

flag(3)

= 2 : B is upper Hessenberg;

otherwise,

B is in general form.

task

= 2 : flag has length 2

flag(1)
if 0 solve continuous-time equation (2a), otherwise, solve discrete-time equation (2b).

flag(2)
= 1 : A is (quasi) upper triangular otherwise, A is in general form.

task
= 3 : flag has length 2

flag(1)
= 0 : solve continuous-time equation (3a); otherwise, solve discrete-time equation (3b).

flag(2)
= 1 : A is (quasi) upper triangular; otherwise, A is in general form.

Default: flag(1) = 0, flag(2) = 0 (, flag(3) = 0).

trans
(optional) integer specifying a transposition option.

=
0 : solve the equations (1) - (3) with $\text{op}(M) = M$.

=
1 : solve the equations (1) - (3) with $\text{op}(M) = M'$.

=
2 : solve the equations (1) with $\text{op}(A) = A'$; $\text{op}(B) = B$;

=
3 : solve the equations (1) with $\text{op}(A) = A$; $\text{op}(B) = B'$.

Default: trans = 0.

schur
(optional) integer specifying whether the Hessenberg-Schur or Schur method should be used.
Available for task = 1.

= 1 : Hessenberg-Schur method (one matrix is reduced to Schur form).

= 2 : Schur method (two matrices are reduced to Schur form).

Default: schur = 1.

X

sep
(optional) estimator of $\text{Sep}(\text{op}(A), -\text{op}(A)')$ for (2.a) or $\text{Sepd}(A, A')$ for (2.b).

Description

linmeq function for solving Sylvester and Lyapunov equations using SLICOT routines SB04MD, SB04ND, SB04PD, SB04QD, SB04RD, SB03MD, and SB03OD.

```
[X] = linmeq(1,A,B,C,flag,trans,schur)
[X,sep] = linmeq(2,A,C,flag,trans)
[X] = linmeq(2,A,C,flag,trans)
[X] = linmeq(3,A,C,flag,trans)
```

linmeq solves various Sylvester and Lyapunov matrix equations:

$$\text{op}(A)*X + X*\text{op}(B) = C, \quad (1a)$$

$$\text{op}(A)*X*\text{op}(B) + X = C, \quad (1b)$$

$$\text{op}(A)'*X + X*\text{op}(A) = C, \quad (2a)$$

$$\text{op}(A)'*X*\text{op}(A) - X = C, \quad (2b)$$

$$\begin{aligned} \text{op}(A)'*(\text{op}(X)'\text{op}(X)) + (\text{op}(X)'\text{op}(X))*\text{op}(A) = \\ - \text{op}(C)'\text{op}(C), \end{aligned} \quad (3a)$$

$$\begin{aligned} \text{op}(A)'*(\text{op}(X)'\text{op}(X))*\text{op}(A) - \text{op}(X)'\text{op}(X) = \\ - \text{op}(C)'\text{op}(C), \end{aligned} \quad (3b)$$

where $\text{op}(M) = M$, or M' .

Comments

1. For equation (1a) or (1b), when $\text{schur} = 1$, the Hessenberg-Schur method is used, reducing one matrix to Hessenberg form and the other one to a real Schur form. Otherwise, both matrices are reduced to real Schur forms. If one or both matrices are already reduced to Schur/Hessenberg forms, this could be specified by $\text{flag}(2)$ and $\text{flag}(3)$. For general matrices, the Hessenberg-Schur method could be significantly more efficient than the Schur method.
2. For equation (2a) or (2b), matrix C is assumed symmetric.
3. For equation (3a) or (3b), matrix A must be stable or convergent, respectively.
4. For equation (3a) or (3b), the computed matrix X is the Cholesky factor of the solution, i.e., the real solution is $\text{op}(X)'\text{op}(X)$, where X is an upper triangular matrix.

Revisions

V. Sima, Katholieke Univ. Leuven, Belgium, May 1999, May, Sep. 2000. V. Sima, University of Bucharest, Romania, May 2000.

Examples

```
//(1a)
n=40;m=30;
A=rand(n,n);C=rand(n,m);B=rand(m,m);
X = linmeq(1,A,B,C);
norm(A*X+X*B-C,1)
//(1b)
flag=[1,0,0]
X = linmeq(1,A,B,C,flag);
norm(A*X*B+X-C,1)
```

```
//(2a)
A=rand(n,n);C=rand(A);C=C+C';
X = linmeq(2,A,C);
norm(A'*X + X*A -C,1)
//(2b)
X = linmeq(2,A,C,[1 0]);
norm(A'*X*A -X-C,1)
//(3a)
A=rand(n,n);
A=A-(max(real(spec(A)))+1)*eye(); //shift eigenvalues
C=rand(A);
X=linmeq(3,A,C);
norm(A'*X'*X+X'*X*A +C'*C,1)
//(3b)
A = [-0.02, 0.02, -0.10, 0.02, -0.03, 0.12;
      0.02, 0.14, 0.12, -0.10, -0.02, -0.14;
      -0.10, 0.12, 0.05, 0.03, -0.04, -0.04;
      0.02, -0.10, 0.03, -0.06, 0.08, 0.11;
      -0.03, -0.02, -0.04, 0.08, 0.14, -0.07;
      0.12, -0.14, -0.04, 0.11, -0.07, 0.04]

C=rand(A);
X=linmeq(3,A,C,[1 0]);
norm(A'*X'*X*A - X'*X +C'*C,1)
```

See Also

sylv , lyap

Authors

H. Xu, TU Chemnitz, FR Germany, Dec. 1998.

Name

lqe — linear quadratic estimator (Kalman Filter)

```
[K,X]=lqe(P21)
```

Parameters

P21
syslin list

K, X
real matrices

Description

lqe returns the Kalman gain for the filtering problem in continuous or discrete time.

P21 is a syslin list representing the system $P21=[A, B1, C2, D21]$
 $P21=syslin('c',A,B1,C2,D21)$ or $P21=syslin('d',A,B1,C2,D21)$

The input to P21 is a white noise with variance:

$$\text{BigV} = \begin{bmatrix} B1 \\ [] \\ D21 \end{bmatrix} \begin{bmatrix} B1' & D21' \end{bmatrix} = \begin{bmatrix} Q & S \\ [] & [] \\ S' & R \end{bmatrix}$$

X is the solution of the stabilizing Riccati equation and $A+K*C2$ is stable.

In continuous time:

$$(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'=0$$

$$K=-(X*C2'+S)*inv(R)$$

In discrete time:

$$X=A*X*A'-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')*(C2*X*A'+D21*B1')+B1*B1'$$

$$K=-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')$$

$\text{xhat}(t+1) = E(x(t+1) | y(0), \dots, y(t))$ (one-step predicted x) satisfies the recursion:

$$\text{xhat}(t+1)=(A+K*C2)*\text{xhat}(t) - K*y(t).$$

Examples

```
//Assume the equations
```

```

//.
//x = Ax + Ge
//y = Cx + v
//with
//E ee' = Q_e,      Evv' = R,      Eev' = N
//
//This is equivalent to
//.
//x = Ax + B1 w
//y = C2x + D21 w
//with E { [Ge ]   [Ge v]' } = E { [B1w ] [B1w D21w]'} = bigR =
//          [ v ]               [D21w]
//
//[B1*B1'   B1*D21';
// D21*B1'  D21*D21']
//=
//[G*Q_e*G'  G*N;
// N*G'  R]

//To find (B1,D21) given (G,Q_e,R,N) form bigR =[G*Q_e*G'  G*N;N'*G'  R].
//Then [W,Wt]=fullrf(bigR);  B1=W(1:size(G,1),:);
//D21=W(($+1-size(C2,1)):$,:);
//
//P21=syslin('c',A,B1,C2,D21);
//[K,X]=lqe(P21);

//Example:
nx=5;ne=2;ny=3;
A=-diag(1:nx);G=ones(nx,ne);
C=ones(ny,nx); Q_e(ne,ne)=1; R=diag(1:ny); N=zeros(ne,ny);
bigR =[G*Q_e*G'  G*N;N'*G'  R];
[W,Wt]=fullrf(bigR);B1=W(1:size(G,1),:);
D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
[K,X]=lqe(P21);
//Riccati check:
S=G*N;Q=B1*B1';
(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'

//Stability check:
spec(A+K*C)

```

See Also

lqr , observer

Authors

F. D.

Name

`lqg` — LQG compensator

```
[K]=lqg(P,r)
```

Parameters

`P`
syslin list (augmented plant) in state-space form

`r`
1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of `P`)

`K`
syslin list (controller)

Description

`lqg` computes the linear optimal LQG (H2) controller for the "augmented" plant `P=syslin('c',A,B,C,D)` (continuous time) or `P=syslin('d',A,B,C,D)` (discrete time).

The function `lqg2stan` returns `P` and `r` given the nominal plant, weighting terms and variances of noises.

`K` is given by the following ABCD matrices: $[A+B*Kc+Kf*C+Kf*D*Kc, -Kf, Kc, 0]$ where `Kc=lqr(P12)` is the controller gain and `Kf=lqe(P21)` is the filter gain. See example in `lqg2stan`.

See Also

`lqg2stan`, `lqr`, `lqe`, `h_inf`, `obscont`

Authors

F.D.

Name

lqg2stan — LQG to standard problem

```
[P,r]=lqg2stan(P22,bigQ,bigR)
```

Parameters

P22

syslin list (nominal plant) in state-space form

bigQ

$[Q, S; S', N]$ (symmetric) weighting matrix

bigR

$[R, T; T', V]$ (symmetric) covariance matrix

r

1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of P)

P

syslin list (augmented plant)

Description

lqg2stan returns the augmented plant for linear LQG (H2) controller design.

P22=syslin(dom,A,B2,C2) is the nominal plant; it can be in continuous time (dom='c') or discrete time (dom='d').

```
.  
x = Ax + w1 + B2u  
y = C2x + w2
```

for continuous time plant.

```
x[n+1]= Ax[n] + w1 + B2u  
y = C2x + w2
```

for discrete time plant.

The (instantaneous) cost function is $[x' \ u'] \text{ bigQ } [x;u]$.

The covariance of $[w1;w2]$ is $E[w1;w2] [w1',w2'] = \text{bigR}$

If $[B1;D21]$ is a factor of bigQ, $[C1,D12]$ is a factor of bigR and $[A,B2,C2,D22]$ is a realization of P22, then P is a realization of $[A, [B1, B2], [C1, -C2], [0, D12; D21, D22]]$. The (negative) feedback computed by lqg stabilizes P22, i.e. the poles of $cl=P22/.K$ are stable.

Examples

```
ny=2;nu=3;nx=4;  
P22=ssrand(ny,nu,nx);  
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
```

```
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';  
[P,r]=lqg2stan(P22,bigQ,bigR);K=lqg(P,r); //K=LQG-controller  
spec(h_cl(P,r,K)) //Closed loop should be stable  
//Same as Cl=P22/.K; spec(Cl('A'))  
s=poly(0,'s')  
lqg2stan(1/(s+2),eye(2,2),eye(2,2))
```

See Also

[lqg](#) , [lqr](#) , [lqe](#) , [obscont](#) , [h_inf](#) , [augment](#) , [fstabst](#) , [feedback](#)

Authors

F.D.

Name

lqg_ltr — LQG with loop transform recovery

```
[kf, kc]=lqg_ltr(sl, mu, ro)
```

Parameters

sl
linear system in state-space form (syslin list)

mu, ro
real positive numbers chosen ``small enough"

kf, kc
controller and observer Kalman gains.

Description

returns the Kalman gains for:

```
(sl)      x = a*x + b*u + l*w1
          y = c*x + mu*I*w2
          z = h*x
```

Cost function:

$$J_{lqg} = E \left(\int_0^{+\infty} [z(t)'z(t) + ro^2 u(t)'u(t)] dt \right)$$

The lqg/ltr approach looks for L, μ, H, ro such that: $J(lqg) = J(freq)$ where

$$J_{freq} = \int_0^{+\infty} \text{tr} \begin{bmatrix} S & W & W^* & S^* \end{bmatrix} + \text{tr} \begin{bmatrix} T & T^* \end{bmatrix} dw$$

and

$$\begin{aligned} S &= (I + G^*K)^{-1} \\ T &= G^*K(I+G^*K)^{-1} \end{aligned}$$

See Also

syslin

Name

lqr — LQ compensator (full state)

```
[K,X]=lqr(P12)
```

Parameters

P12
syslin list (state-space linear system)

K,X
two real matrices

Description

lqr computes the linear optimal LQ full-state gain for the plant $P12=[A, B2, C1, D12]$ in continuous or discrete time.

P12 is a syslin list (e.g. $P12= \text{syslin}('c', A, B2, C1, D12)$).

The cost function is l2-norm of $z' * z$ with $z=C1 \ x + D12 \ u$ i.e. $[x,u]' * \text{BigQ} * [x;u]$ where

$$\text{BigQ} = \begin{bmatrix} C1' \\ D12' \end{bmatrix} * \begin{bmatrix} C1 & D12 \end{bmatrix} = \begin{bmatrix} Q & S \\ S' & R \end{bmatrix}$$

The gain K is such that $A + B2 * K$ is stable.

X is the stabilizing solution of the Riccati equation.

For a continuous plant:

$$(A - B2 * \text{inv}(R) * S')' * X + X * (A - B2 * \text{inv}(R) * S') - X * B2 * \text{inv}(R) * B2' * X + Q - S * \text{inv}(R) * S' = 0$$

$$K = -\text{inv}(R) * (B2' * X + S)$$

For a discrete plant:

$$X = A' * X * A - (A' * X * B2 + C1' * D12) * \text{pinv}(B2' * X * B2 + D12' * D12) * (B2' * X * A + D12' * C1) + C1' * C1;$$

$$K = -\text{pinv}(B2' * X * B2 + D12' * D12) * (B2' * X * A + D12' * C1)$$

An equivalent form for X is

$$X = \text{Abar}' * \text{inv}(\text{inv}(X) + B2 * \text{inv}(r) * B2') * \text{Abar} + Q\text{bar}$$

with $\bar{A} = A - B2 \cdot \text{inv}(R) \cdot S'$ and $\bar{Q} = Q - S \cdot \text{inv}(R) \cdot S'$

The 3-blocks matrix pencils associated with these Riccati equations are:

	discrete							continuous					
	I	0	0	- A	0	B2		I	0	0	- A	0	B2
z	0	A'	0	- -Q	I	-S	s	0	I	0	- -Q	-A'	-S
	0	B2'	0	S'	0	R		0	0	0	S'	-B2'	R

Caution: It is assumed that matrix R is non singular. In particular, the plant must be tall (number of outputs $\geq$ number of inputs).

Examples

```
A=rand(2,2);B=rand(2,1); //two states, one input
Q=diag([2,5]);R=2; //Usual notations x'Qx + u'Ru
Big=sysdiag(Q,R); //Now we calculate C1 and D12
[w,wp]=fullrf(Big);C1=wp(:,1:2);D12=wp(:,3:$); //[C1,D12]'*[C1,D12]=Big
P=syslin('c',A,B,C1,D12); //The plant (continuous-time)
[K,X]=lqr(P)
spec(A+B*K) //check stability
norm(A'*X+X*A-X*B*inv(R)*B'*X+Q,1) //Riccati check
P=syslin('d',A,B,C1,D12); // Discrete time plant
[K,X]=lqr(P)
spec(A+B*K) //check stability
norm(A'*X*A-(A'*X*B)*pinv(B'*X*B+R)*(B'*X*A)+Q-X,1) //Riccati check
```

See Also

lqe , gcare , leqr

Authors

F.D.;

Name

ltitr — discrete time response (state space)

```
[X]=ltitr(A,B,U,[x0])  
[xf,X]=ltitr(A,B,U,[x0])
```

Parameters

A,B
real matrices of appropriate dimensions

U,X
real matrices

x0,xf
real vectors (default value=0 for x0))

Description

calculates the time response of the discrete time system

```
x[t+1] = Ax[t] + Bu[t].
```

The inputs u_i 's are the columns of the U matrix

```
U=[u0,u1,...,un];
```

x0 is the vector of initial state (default value : 0) ;

X is the matrix of outputs (same number of columns as U).

```
X=[x0,x1,x2,...,xn]
```

xf is the vector of final state $xf=X[n+1]$

Examples

```
A=eye(2,2);B=[1;1];  
x0=[-1;-2];  
u=[1,2,3,4,5];  
x=ltitr(A,B,u,x0)  
x1=A*x0+B*u(1)  
x2=A*x1+B*u(2)  
x3=A*x2+B*u(3) //....
```

See Also

rtitr, flts

Name

`m_circle` — plots the complex plane iso-gain contours of $y/(1+y)$

```
m_circle()
m_circle(gain)
```

Parameters

`gain`

vector of gains (in DB). The default value is

`gain`

`=[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]`

Description

`m_circle` draws the iso-gain contours given by then `gain` argument in the complex plane (Re,Im).

The default value for `gain` is:

`[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]`

`m_circle` is used with `nyquist`.

Examples

```
//Example 1 :
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
nyquist(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
m_circle();
//Example 2:
clf();
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
nyquist([h1;h],0.01,100,['h1';'h'])
m_circle([-8 -6 -4]);
```

See Also

`nyquist` , `chart` , `black`

Authors

S.Steer.;

Name

macglov — Mac Farlane Glover problem

```
[P,r]=macglov(Sl)
```

Parameters

Sl
linear system (syslin list)

P
linear system (syslin list), ``augmented" plant

r
1x2 vector, dimension of P22

Description

`[P,r]=macglov(Sl)` returns the standard plant P for the Glover-McFarlane problem.

For this problem $ro\_optimal = 1 - \text{hankel\_norm}([N, M])$ with $[N, M] = \text{lcf}(Sl)$ (Normalized coprime factorization) i.e.

$gama\_optimal = 1/\text{sqrt}(ro\_optimal)$

Authors

F. Delebecque INRIA

Name

markp2ss — Markov parameters to state-space

```
[sl]=markp2ss(markpar,n,nout,nin)
```

Parameters

markpar
matrix

n,nout,nin
integers

sl
syslin list

Description

given a set of n Markov parameters stacked in the (row)-matrix `markpar` of size $n_{out} \times (n \times n_{in})$ `markp2ss` returns a state-space linear system `sl` (syslin list) such that with $[A, B, C, D] = \text{abcd}(sl)$:

```
C*B = markpar(1:nout,1:nin),  
C*A*B = markpar(1:nout,nin+1:2*nin), ....
```

Examples

```
W=ssrand(2,3,4); //random system with 2 outputs and 3 inputs  
[a,b,c,d]=abcd(W);  
markpar=[c*b,c*a*b,c*a^2*b,c*a^3*b,c*a^4*b];  
S=markp2ss(markpar,5,2,3);  
[A,B,C,D]=abcd(S);  
Markpar=[C*B,C*A*B,C*A^2*B,C*A^3*B,C*A^4*B];  
norm(markpar-Markpar,1)  
//Caution... c*a^5*b is not C*A^5*B !
```

See Also

frep2tf , tf2ss , imrep2ss

Name

minreal — minimal balanced realization

```
slb=minreal(sl [,tol])
```

Parameters

sl,slb
syslin lists

tol
real (threshold)

Description

[ae,be,ce]=minreal(a,b,c,domain [,tol]) returns the balanced realization of linear system sl (syslin list).

sl is assumed stable.

tol threshold used in equil1.

Examples

```
A=[-eye(2,2),rand(2,2);zeros(2,2),-2*eye(2,2)];  
B=[rand(2,2);zeros(2,2)];C=rand(2,4);  
sl=syslin('c',A,B,C);  
slb=minreal(sl);  
ss2tf(sl)  
ss2tf(slb)  
ctr_gram(sl)  
clean(ctr_gram(slb))  
clean(obs_gram(slb))
```

See Also

minss , balreal , arhnk , equil , equil1

Authors

S. Steer INRIA 1987

Name

minss — minimal realization

```
[slc]=minss( sl [,tol])
```

Parameters

sl,slc
syslin lists (linear systems in state-space form)

tol
real (threshold for rank determination (see `contr`))

Description

minss returns in slc a minimal realization of sl.

Examples

```
sl=syslin('c',[1 0;0 2],[1;0],[2 1]);  
ssprint(sl);  
ssprint(minss(sl))
```

See Also

`contr` , `minreal` , `arhnk` , `contrss` , `obsvss` , `balreal`

Name

mucomp — mu (structured singular value) calculation

```
[BOUND, D, G] = mucomp(Z, K, T)
```

Parameters

- Z**
the complex n-by-n matrix for which the structured singular value is to be computed
- K**
the vector of length m containing the block structure of the uncertainty.
- T**
the vector of length m indicating the type of each block. T(I) = 1 if the corresponding block is real T(I) = 2 if the corresponding block is complex.
- BOUND**
the upper bound on the structured singular value.
- D, G**
vectors of length n containing the diagonal entries of the diagonal matrices D and G, respectively, such that the matrix $Z' * D^2 * Z + \text{sqrt}(-1) * (G * Z - Z' * G) - \text{bound}^2 * D^2$ is negative semidefinite.

Description

To compute an upper bound on the structured singular value for a given square complex matrix and given block structure of the uncertainty.

Reference

Slicot routine AB13MD.

Name

narsimul — armax simulation (using rtitr)

```
[z]=narsimul(a,b,d,sig,u,[up,yp,ep])  
[z]=narsimul(ar,u,[up,yp,ep])
```

Description

ARMAX simulation. Same as arsimul but the method is different the simulation is made with rtitr

Authors

J-Ph. Chancelier ENPC Cergrene; ;

Name

nehari — Nehari approximant

```
[x]=nehari(R [,tol])
```

Parameters

R
linear system (syslin list)

x
linear system (syslin list)

tol
optional threshold

Description

`[x]=nehari(R [,tol])` returns the Nehari approximant of R.

R = linear system in state-space representation (syslin list).

R is strictly proper and $-R^*$ is stable (i.e. R is anti stable).

$$\|R - X\|_{\infty} = \min_{Y \text{ in } H_{\infty}} \|R - Y\|_{\infty}$$

Name

noisegen — noise generation

```
b=noisegen(pas,Tmax,sig)
```

Description

generates a Scilab function `[b]=Noise(t)` where `Noise(t)` is a piecewise constant function (constant on $[k*\text{pas}, (k+1)*\text{pas}]$). The value on each constant interval are random values from i.i.d Gaussian variables of standard deviation `sig`. The function is constant for $t \leq 0$ and $t \geq \text{Tmax}$.

Examples

```
noisegen(0.5,30,1.0);  
x=-5:0.01:35;  
y=feval(x,Noise);  
plot(x,y);
```

Name

nyquist — nyquist plot

```
nyquist( sl, [fmin, fmax] [, step] [, comments] )  
nyquist( sl, frq [, comments] )  
nyquist( frq, db, phi [, comments] )  
nyquist( frq, repf [, comments] )
```

Parameters

sl
syslin list (SIMO linear system in continuous or discrete time)

fmin, fmax
real scalars (frequency bounds (in Hz))

step
real (logarithmic discretization step)

comments
string vector (captions).

frq
vector or matrix of frequencies (in Hz) (one row for each output of **sl**).

db, phi
real matrices of modulus (in Db) and phases (in degree) (one row for each output of **sl**).

repf
matrix of complex numbers. Frequency response (one row for each output of **sl**)

Description

Nyquist plot i.e Imaginary part versus Real part of the frequency response of **sl**.

For continuous time systems $sl(2*i*\pi*w)$ is plotted. For discrete time system or discretized systems $sl(\exp(2*i*\pi*w*fd))$ is used ($fd=1$ for discrete time systems and $fd=sl('dt')$ for discretized systems)

sl can be a continuous-time or discrete-time SIMO system (see **syslin**). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds **fmin**, **fmax** (in Hz) or by a row-vector (or a matrix for multi-output) **frq**.

step is the (logarithmic) discretization step. (see **calfrq** for the choice of default value).

comments is a vector of character strings (captions).

db, **phi** are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

repf is a matrix of complex numbers. One row for each response.

Default values for **fmin** and **fmax** are $1.d-3$, $1.d+3$ if **sl** is continuous-time or $1.d-3$, $0.5/sl.dt$ (nyquist frequency) if **sl** is discrete-time.

Automatic discretization of frequencies is made by **calfrq**.

Examples

```
clf();
s=poly(0, 's');
h=syslin('c', (s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
comm='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
nyquist(h, 0.01, 100, comm);
h1=h*syslin('c', (s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225));
clf();
nyquist([h1;h], 0.01, 100, ['h1'; 'h'])
clf();nyquist([h1;h])
```

See Also

bode, black, calfrq, freq, repfreq, phasemag

Name

obs_gram — observability gramian

```
Go=obs_gram(A,C [, dom])  
Go=obs_gram(sl)
```

Parameters

A,C
real matrices (of appropriate dimensions)

dom
string ('d' or 'c' (default value))

sl
syslin list

Description

Observability gramian of the pair (A, C) or linear system sl (syslin list). dom is the domain which can be

"c"
continuous system (default)

"d"
discrete system

Examples

```
A=-diag(1:3);C=rand(2,3);  
Go=obs_gram(A,C,'c');      // <=> w=syslin('c',A,[],C); Go=obs_gram(w);  
norm(Go*A+A'*Go+C'*C,1)  
norm(lyap(A,-C'*C,'c')-Go,1)  
A=A/4; Go=obs_gram(A,C,'d');    //discrete time case  
norm(lyap(A,-C'*C,'d')-Go,1)
```

See Also

ctr_gram , obsvss , obsv_mat , lyap

Name

obscont — observer based controller

```
[K]=obscont(P,Kc,Kf)
[J,r]=obscont(P,Kc,Kf)
```

Parameters

P
syslin list (nominal plant) in state-space form, continuous or discrete time

Kc
real matrix, (full state) controller gain

Kf
real matrix, filter gain

K
syslin list (controller)

J
syslin list (extended controller)

r
1x2 row vector

Description

obscont returns the observer-based controller associated with a nominal plant **P** with matrices **[A, B, C, D]** (syslin list).

The full-state control gain is **Kc** and the filter gain is **Kf**. These gains can be computed, for example, by pole placement.

$A+B*Kc$ and $A+Kf*C$ are (usually) assumed stable.

K is a state-space representation of the compensator $y \rightarrow u$ in:

$$\dot{x} = A x + B u, \quad y = C x + D u, \quad \dot{z} = (A + Kf C) z - Kf y + B u, \quad u = Kc z$$

K is a linear system (syslin list) with matrices given by: $K=[A+B*Kc+Kf*C+Kf*D*Kc, Kf, -Kc]$.

The closed loop feedback system $Cl: v \rightarrow y$ with (negative) feedback **K** (i.e. $y = -P u$, $u = v - K y$), or

```
xdot = A x + B u,
y = C x + D u,
zdot = (A + Kf C) z - Kf y + B u,
u = v - F z
```

) is given by $Cl = P / (-K)$

The poles of Cl (`spec(cl('A'))`) are located at the eigenvalues of $A+B*Kc$ and $A+Kf*C$.

Invoked with two output arguments **obscont** returns a (square) linear system **K** which parametrizes all the stabilizing feedbacks via a LFT.

Let Q an arbitrary stable linear system of dimension $r(2) \times r(1)$ i.e. number of inputs $\times$ number of outputs in P . Then any stabilizing controller K for P can be expressed as $K = \text{lft}(J, r, Q)$. The controller which corresponds to $Q=0$ is $K=J(1:nu, 1:ny)$ (this K is returned by $K=\text{obscont}(P, Kc, Kf)$). r is $\text{size}(P)$ i.e the vector [number of outputs, number of inputs];

Examples

```
ny=2;nu=3;nx=4;P=ssrand(ny,nu,nx);[A,B,C,D]=abcd(P);
Kc=-ppol(A,B,[-1,-1,-1,-1]); //Controller gain
Kf=-ppol(A',C',[-2,-2,-2,-2]);Kf=Kf'; //Observer gain
cl=P/.( -obscont(P,Kc,Kf));spec(cl('A')) //closed loop system
[J,r]=obscont(P,Kc,Kf);
Q=ssrand(nu,ny,3);Q('A')=Q('A')-(maxi(real(spec(Q('A'))))+0.5)*eye(Q('A'))
//Q is a stable parameter
K=lft(J,r,Q);
spec(h_cl(P,K)) // closed-loop A matrix (should be stable);
```

See Also

ppol , lqg , lqr , lqe , h_inf , lft , syslin , feedback , observer

Authors

F.D.;;

Name

observer — observer design

```
Obs=observer(Sys,J)
[Obs,U,m]=observer(Sys [,flag,alfa])
```

Parameters

Sys
syslin list (linear system)

J
 $n_x \times n_y$ constant matrix (output injection matrix)

flag
character strings ('pp' or 'st' (default))

alfa
location of closed-loop poles (optional parameter, default=-1)

Obs
linear system (syslin list), the observer

U
orthogonal matrix (see dt_ility)

m
integer (dimension of unstable unobservable (st) or unobservable (pp) subspace)

Description

`Obs=observer(Sys,J)` returns the observer `Obs=syslin(td,A+J*C,[B+J*D,-J],eye(A))` obtained from `Sys` by a `J` output injection. (`td` is the time domain of `Sys`). More generally, `observer` returns in `Obs` an observer for the observable part of linear system `Sys`: $\dot{x}=Ax + Bu$, $y=Cx + Du$ represented by a `syslin` list. `Sys` has n_x state variables, n_u inputs and n_y outputs. `Obs` is a linear system with matrices `[Ao,Bo,Identity]`, where `Ao` is $n_o \times n_o$, `Bo` is $n_o \times (n_u+n_y)$, `Co` is $n_o \times n_o$ and $n_o=n_x-m$.

Input to `Obs` is `[u,y]` and output of `Obs` is:

$\hat{x}$ =estimate of x modulo unobservable subsp. (case `flag='pp'`) or

$\hat{x}$ =estimate of x modulo unstable unobservable subsp. (case `flag='st'`)

case `flag='st'`: $z=H*x$ can be estimated with stable observer iff $H*U(:,1:m)=0$ and assignable poles of the observer are set to `alfa(1),alfa(2),...`

case `flag='pp'`: $z=H*x$ can be estimated with given error spectrum iff $H*U(:,1:m)=0$ all poles of the observer are assigned and set to `alfa(1),alfa(2),...`

If H satisfies the constraint: $H*U(:,1:m)=0$ ($\ker(H)$ contains unobs-subsp. of `Sys`) one has $H*U=[0,H2]$ and the observer for $z=H*x$ is $H2*Obs$ with $H2=H*U(:,m+1:n_x)$ i.e. `Co`, the C -matrix of the observer for $H*x$, is `Co=H2`.

In the particular case where the pair (A,C) of `Sys` is observable, one has $m=0$ and the linear system `U*Obs` (resp. $H*U*Obs$) is an observer for x (resp. Hx). The error spectrum is `alfa(1),alfa(2),...,alfa(n_x)`.

Examples

```

nx=5;nu=1;ny=1;un=3;us=2;Sys=ssrand(ny,nu,nx,list('dt',us,us,un));
//nx=5 states, nu=1 input, ny=1 output,
//un=3 unobservable states, us=2 of them unstable.
[Obs,U,m]=observer(Sys); //Stable observer (default)
W=U';H=W(m+1:nx,:);[A,B,C,D]=abcd(Sys); //H*U=[0,eye(no,no)];
Sys2=ss2tf(syslin('c',A,B,H)) //Transfer u-->z
Idu=eye(nu,nu);Sys3=ss2tf(H*U(:,m+1:$)*Obs*[Idu;Sys])
//Transfer u-->[u;y=Sys*u]-->Obs-->xhat-->HUXhat=zhat i.e. u-->output of Obs
//this transfer must equal Sys2, the u-->z transfer (H2=eye).

//Assume a Kalman model
//dotx = A x + B u + G w
// y = C x + D u + H w + v
//with Eww' = QN, Evv' = RN, Ewv' = NN
//To build a Kalman observer:
//1-Form BigR = [G*QN*G'          G*QN*H'+G*NN;
//              H*QN*G'+NN*G'    H*QN*H'+RN];
//the covariance matrix of the noise vector [Gw;Hw+v]
//2-Build the plant P21 : dotx = A x + B1 e ; y = C2 x + D21 e
//with e a unit white noise.
// [W,Wt]=fullrf(BigR);
//B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
//C2=C;
//P21=syslin('c',A,B1,C2,D21);
//3-Compute the Kalman gain
//L = lqe(P21);
//4- Build an observer for the plant [A,B,C,D];
//Plant = syslin('c',A,B,C,D);
//Obs = observer(Plant,L);
//Test example:
A=-diag(1:4);
B=ones(4,1);
C=B'; D= 0; G=2*B; H=-3; QN=2;
RN=5; NN=0;
BigR = [G*QN*G'          G*QN*H'+G*NN;
        H*QN*G'+NN*G'    H*QN*H'+RN];
[W,Wt]=fullrf(BigR);
B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
L = lqe(P21);
Plant = syslin('c',A,B,C,D);
Obs = observer(Plant,L);
spec(Obs.A)

```

See Also

dt_ility , unobs , stabil

Authors

F.D.

Name

obsv_mat — observability matrix

```
[O]=obsv_mat(A,C)
[O]=obsv_mat(sl)
```

Parameters

A,C,O
real matrices

sl
syslin list

Description

obsv_mat returns the observability matrix:

```
O=[C; CA; CA^2;...; CA^(n-1) ]
```

See Also

contrss , obsvss , obs_gram

Name

obsvss — observable part

```
[Ao,Bo,Co]=obsvss(A,B,C [,tol])  
[slo]=obsvss(sl [,tol])
```

Parameters

A,B,C,Ao,Bo,Co
real matrices

sl,slo
syslin lists

tol
real (threshold) (default value 100*%eps)

Description

slo=(Ao,Bo,Co) is the observable part of linear system sl=(A,B,C) (syslin list)

tol threshold to test controllability (see contr); default value = 100*%eps

See Also

contr , contrss , obsv_mat , obs_gram

Name

p_margin — phase margin and associated crossover frequency

```
[phm,fr] = p_margin(h)
phm=p_margin(h)
```

Parameters

- h**
a SISO linear system (see :syslin).
- phm**
a number, the phase margin in degree if it exists or an empty matrix.
- fr**
a number, the corresponding frequency (in hz) or an empty matrix.

Description

Given a SISO linear system in continuous or discrete time, **p_margin** returns **phm**, the phase margin in degree of **h** and **fr**, the achieved corresponding frequency in hz.

The phase margin is the values of the phase at frequency points where the nyquist plot of **h** crosses the unit circle. In other words the phase margin is the difference between the phase of the frequency response of **h** and -180° when the gain of **h** is 1.

The algorithm uses polynomial root finder to solve the equations:

$h(s)*h(-s)=1$
for the continuous time case.

$h(z)*h(1/z)=1$
for the discrete time case.

Examples

```
//continuous case
h=syslin('c', -1+%s, 3+2*%s+%s^2)
[p,fr]=p_margin(h)
[p,fr]=p_margin(h+0.7)
show_margins(h+0.7, 'nyquist')

//discrete case
h = syslin(0.1, 0.04798*%z+0.0464, %z^2-1.81*%z+0.9048); //ok
[p ,f]=p_margin(h)
show_margins(h, 'nyquist')
```

See Also

p_margin, show_margins, repfreq, black, bode, chart, nyquist

Authors

Serge Steer, INRIA

Name

parrot — Parrot's problem

```
K=parrot(D,r)
```

Parameters

D,K

matrices

r

1X2 vector (dimension of the 2,2 part of D)

Description

Given a matrix D partitioned as $\begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}$ where $\text{size}(D_{22})=r=[r_1, r_2]$ compute a matrix K such that largest singular value of $\begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22}+K \end{bmatrix}$ is minimal (Parrot's problem)

See Also

h_inf_st

Name

pfss — partial fraction decomposition

```
elts=pfss(Sl)
elts=pfss(Sl,rmax)
elts=pfss(Sl,'cord')
elts=pfss(Sl,rmax,'cord')
```

Parameters

Sl

sys lin list (state-space or transfer linear system) **rmax** : real number controlling the conditioning of block diagonalization **cord** : character string 'c' or 'd'.

Description

Partial fraction decomposition of the linear system **Sl** (in state-space form, transfer matrices are automatically converted to state-space form by **tf2ss**):

elts is the list of linear systems which add up to **Sl** i.e. **elts=list(S1,S2,S3,...,Sn)** with:

$$Sl = S1 + S2 + \dots + Sn.$$

Each **Si** contains some poles of **S** according to the block-diagonalization of the **A** matrix of **S**.

For non proper systems the polynomial part of **Sl** is put in the last entry of **elts**.

If **Sl** is given in transfer form, it is first converted into state-space and each subsystem **Si** is then converted in transfer form.

The **A** matrix of the state-space is put into block diagonal form by function **bdiag**. The optional parameter **rmax** is sent to **bdiag**. If **rmax** should be set to a large number to enforce block-diagonalization.

If the optional flag **cord='c'** is given the elements in **elts** are sorted according to the real part (resp. magnitude if **cord='d'**) of the eigenvalues of **A** matrices.

Examples

```
w=ssrand(1,1,6);
elts=pfss(w);
w1=0;for k=1:size(elts), w1=w1+ss2tf(elts(k));end
clean(ss2tf(w)-w1)
```

See Also

pbig , bdiag , coffg , dtsi

Authors

F.D.;

Name

phasemag — phase and magnitude computation

```
[phi, db]=phasemag(z [, mod])
```

Parameters

z
matrix or row vector of complex numbers.

mod
character string

mod='c'
"continuous" representation between -infinity and +360 degrees (default)

mod='m'
representation between -360 and 0 degrees

phi
phases (in degree) of z.

db
magnitude (in Db)

Description

phasemag computes the phases and magnitudes of the entries of a complex matrix. For mod='c' phasemag computes `phi(:, i+1)` to minimize the distance with `phi(:, i)`, i.e. it tries to obtain a "continuous representation" of the phase.

To obtain the phase between $-\pi$ and π use `phi=atan(imag(z), real(z))`

Examples

```
s=poly(0, 's');  
h=syslin('c', 1/((s+5)*(s+10)*(100+6*s+s*s)*(s+.3)));  
[frq, rf]=repfreq(h, 0.1, 20, 0.005);  
scf();  
plot2d(frq', phasemag(rf, 'c'));  
scf();  
plot2d(frq', phasemag(rf, 'm'));
```

See Also

repfreq, gainplot, atan, bode

Name

ppol — pole placement

```
[K]=ppol(A,B,poles)
```

Parameters

A,B
real matrices of dimensions $n \times n$ and $n \times m$.

poles
real or complex vector of dimension n .

K
real matrix (negative feedback gain)

Description

`K=ppol(A,B,poles)` returns a $m \times n$ gain matrix K such that the eigenvalues of $A-B*K$ are `poles`. The pair (A,B) must be controllable. Complex number in `poles` must appear in conjugate pairs.

An output-injection gain F for (A,C) is obtained as follows:

```
Ft=ppol(A',C',poles); F=Ft'
```

The algorithm is by P.H. Petkov.

Examples

```
A=rand(3,3);B=rand(3,2);  
F=ppol(A,B,[-1,-2,-3]);  
spec(A-B*F)
```

See Also

canon , stabil

Name

prbs_a — pseudo random binary sequences generation

```
[u]=prbs_a(n,nc,[ids])
```

Description

generation of pseudo random binary sequences $u=[u_0, u_1, \dots, u_{(n-1)}]$ u takes values in $\{-1,1\}$ and changes at most nc times its sign. ids can be used to fix the date at which u must change its sign ids is then an integer vector with values in $[1:n]$.

Examples

```
u=prbs_a(50,10);  
plot2d2("onn",(1:50)',u',1,"151",' ',[0,-1.5,50,1.5]);
```

Name

projsl — linear system projection

```
[slp]=projsl(sl,Q,M)
```

Parameters

sl,slp

syslin lists

Q,M

matrices (projection factorization)

Description

slp= projected model of sl where Q^*M is the full rank factorization of the projection.

If (A, B, C, D) is the representation of sl, the projected model is given by (M^*A^*Q, M^*B, C^*Q, D) .

Usually, the projection Q^*M is obtained as the spectral projection of an appropriate auxiliary matrix W
e.g. W = product of (weighted) gramians or product of Riccati equations.

Examples

```
rand('seed',0);sl=ssrand(2,2,5);[A,B,C,D]=abcd(sl);poles=spec(A)
[Q,M]=pbig(A,0,'c'); //keeping unstable poles
slred=projsl(sl,Q,M);spec(slred('A'))
sl('D')=rand(2,2); //making proper system
trzeros(sl) //zeros of sl
wi=inv(sl); //wi=inverse in state-space
[q,m]=psmall(wi('A'),2,'d'); //keeping small zeros (poles of wi) i.e. abs(z)<2
slred2=projsl(sl,q,m);
trzeros(slred2) //zeros of slred2 = small zeros of sl
// Example keeping second order modes
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c'); //keeping 2 eigenvalues of W
slr=projsl(sl,Q,M); //reduced model
hankelsv(slr)
```

See Also

pbig

Authors

F. D.;

Name

reglin — Linear regression

```
[a,b,sig]=reglin(x,y)
```

Description

solve the regression problem $y=a*x+ b$ in the least square sense. `sig` is the standard deviation of the residual. `x` and `y` are two matrices of size $x(p,n)$ and $y(q,n)$, where n is the number of samples.

The estimator `a` is a matrix of size (q,p) and `b` is a vector of size $(q,1)$

```
// simulation of data for a(3,5) and b(3,1)
x=rand(5,100);
aa=testmatrix('magi',5);aa=aa(1:3,:);
bb=[9;10;11]
y=aa*x +bb*ones(1,100)+ 0.1*rand(3,100);
// identification
[a,b,sig]=reglin(x,y);
maxi(abs(aa-a))
maxi(abs(bb-b))
// an other example : fitting a polynom
f=1:100; x=[f.*f; f];
y= [ 2,3]*x+ 10*ones(f) + 0.1*rand(f);
[a,b]=reglin(x,y)
```

See Also

`pinv` , `leastsq` , `qr`

Name

repfreq — frequency response

```
[ [frq,] repf]=repfreq(sys,fmin,fmax [,step])  
[ [frq,] repf]=repfreq(sys [,frq])  
[ frq,repf,splitf]=repfreq(sys,fmin,fmax [,step])  
[ frq,repf,splitf]=repfreq(sys [,frq])
```

Parameters

sys

sys lin list : SIMO linear system

fmin,fmax

two real numbers (lower and upper frequency bounds)

frq

real vector of frequencies (Hz)

step

logarithmic discretization step

splitf

vector of indexes of critical frequencies.

repf

vector of the complex frequency response

Description

repfreq returns the frequency response calculation of a linear system. If $\text{sys}(s)$ is the transfer function of Sys, $\text{repf}(k)$ equals $\text{sys}(s)$ evaluated at $s = i * \text{frq}(k) * 2 * \pi$ for continuous time systems and at $\exp(2 * i * \pi * \text{dt} * \text{frq}(k))$ for discrete time systems (dt is the sampling period).

$\text{db}(k)$ is the magnitude of $\text{repf}(k)$ expressed in dB i.e. $\text{db}(k) = 20 * \log_{10}(\text{abs}(\text{repf}(k)))$ and $\text{phi}(k)$ is the phase of $\text{repf}(k)$ expressed in degrees.

If fmin, fmax, step are input parameters, the response is calculated for the vector of frequencies frq given by: $\text{frq} = [10.^{((\log_{10}(\text{fmin})): \text{step}: (\log_{10}(\text{fmax})))} \text{ fmax}]$;

If step is not given, the output parameter frq is calculated by $\text{frq} = \text{calfrq}(\text{sys}, \text{fmin}, \text{fmax})$.

Vector frq is splitted into regular parts with the split vector. $\text{frq}(\text{splitf}(k): \text{splitf}(k+1)-1)$ has no critical frequency. sys has a pole in the range $[\text{frq}(\text{splitf}(k)), \text{frq}(\text{splitf}(k)+1)]$ and no poles outside.

Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];  
Sys=syslin('c',A,B,C);  
frq=0:0.02:5;w=frq*2*pi; //frq=frequencies in Hz ;w=frequencies in rad/sec;  
[frq1,rep]=repfreq(Sys,frq);  
[db,phi]=dbphi(rep);  
Systf=ss2tf(Sys) //Transfer function of Sys  
x=horner(Systf,w(2)*sqrt(-1)) // x is Systf(s) evaluated at s = i w(2)
```

```
rep=20*log(abs(x))/log(10)    //magnitude of x in dB
db(2)    // same as rep
ang=atan(imag(x),real(x));    //in rad.
ang=ang*180/%pi              //in degrees
phi(2)
repf=repfreq(Sys,frq);
repf(2)-x
```

See Also

bode , freq , calfrq , horner , nyquist , dbphi

Authors

S. S.

Name

ric_desc — Riccati equation

```
X=ric_desc(H [,E])  
[X1,X2,zero]=ric_desc(H [,E])
```

Parameters

H,E
real square matrices

X1,X2
real square matrices

zero
real number

Description

Riccati solver with hamiltonian matrices as inputs.

In the continuous time case calling sequence is `ric_desc(H)` (one input):

Riccati equation is:

$$(Ec) \quad A'X + XA + XRX - Q = 0.$$

Defining the hamiltonian matrix H by:

$$H = \begin{bmatrix} A & R; \\ Q & -A' \end{bmatrix}$$

with the calling sequence `[X1,X2,zero]=ric_desc(H)`, the solution X is given by $X=X1/X2$.

`zero` = L1 norm of rhs of (Ec)

The solution X is also given by `X=riccati(A,Q,R,'c')`

In the discrete-time case calling sequence is `ric_desc(H,E)` (two inputs):

The Riccati equation is:

$$(Ed) \quad A'XA - (A'XB*(R+B'XB)^{-1})(B'XA)+C-X = 0.$$

Defining $G=B/R*B'$ and the hamiltonian pencil (E, H) by:

$$E = \begin{bmatrix} \text{eye}(n,n), G; \\ 0 * \text{ones}(n,n), A' \end{bmatrix} \quad H = \begin{bmatrix} A, 0 * \text{ones}(n,n); \\ -C, \text{eye}(n,n) \end{bmatrix};$$

with the calling sequence `[X1,X2,err]=ric_desc(H,E)`, the solution X is given by $X=X1/X2$.

zero= L1 norm of rhs of (Ed)

The solution X is also given by $X=\text{riccati}(A,G,C, 'd')$ with $G=B/R*B'$

See Also

[riccati](#)

Name

ricc — Riccati equation

```
[X,RCOND,FERR]=ricc(A,B,C,"cont","method")
[X,RCOND,FERR]=ricc(F,G,H,"disc","method")
```

Parameters

A,B,C

real matrices of appropriate dimensions

F,G,H

real matrices of appropriate dimensions

X

real matrix

"cont","disc"

imposed string (flag for continuous or discrete)

method

'schr' or 'sign' for continuous-time systems and 'schr' or 'invf' for discrete-time systems

Description

Riccati solver.

Continuous time:

```
X=ricc(A,B,C,'cont')
```

gives a solution to the continuous time ARE

$$A' * X + X * A - X * B * X + C = 0 \quad .$$

B and C are assumed to be nonnegative definite. (A, G) is assumed to be stabilizable with $G * G'$ a full rank factorization of B.

(A, H) is assumed to be detectable with $H * H'$ a full rank factorization of C.

Discrete time:

```
X=ricc(F,G,H,'disc')
```

gives a solution to the discrete time ARE

$$X = F' * X * F - F' * X * G1 * ((G2 + G1' * X * G1)^{-1}) * G1' * X * F + H$$

F is assumed invertible and $G = G1 * \text{inv}(G2) * G1'$.

One assumes (F, G1) stabilizable and (C, F) detectable with $C' * C$ full rank factorization of H. Use preferably `ric_desc`.

C, D are symmetric .It is assumed that the matrices A, C and D are such that the corresponding matrix pencil has N eigenvalues with moduli less than one.

Error bound on the solution and a condition estimate are also provided. It is assumed that the matrices A, C and D are such that the corresponding Hamiltonian matrix has N eigenvalues with negative real parts.

Examples

```
//Standard formulas to compute Riccati solutions
A=rand(3,3);B=rand(3,2);C=rand(3,3);C=C*C';R=rand(2,2);R=R*R'+eye();
B=B*inv(R)*B';
X=ricc(A,B,C,'cont');
norm(A'*X+X*A-X*B*X+C,1)
H=[A -B; -C -A'];
[T,d]=schur(eye(H),H,'cont');T=T(:,1:d);
X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
[T,d]=schur(H,'cont');T=T(:,1:d);
X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
//      Discrete time case
F=A;B=rand(3,2);G1=B;G2=R;G=G1/G2*G1';H=C;
X=ricc(F,G,H,'disc');
norm(F'*X*F-(F'*X*G1/(G2+G1'*X*G1))*(G1'*X*F)+H-X)
H1=[eye(3,3) G;zeros(3,3) F'];
H2=[F zeros(3,3);-H eye(3,3)];
[T,d]=schur(H2,H1,'disc');T=T(:,1:d);X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
Fi=inv(F);
Hami=[Fi Fi*G;H*Fi F'+H*Fi*G];
[T,d]=schur(Hami,'d');T=T(:,1:d);
Fit=inv(F');
Ham=[F+G*Fit*H -G*Fit;-Fit*H Fit];
[T,d]=schur(Ham,'d');T=T(:,1:d);X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
```

See Also

riccati , ric_desc , schur

Authors

P. Petkov

Used Functions

See SCI/modules/cacsd/src/slicot/riccpack.f

Name

riccati — Riccati equation

```
X=riccati(A,B,C,dom,[typ])  
[X1,X2]=riccati(A,B,C,dom,[typ])
```

Parameters

A,B,C

real matrices nxn, B and C symmetric.

dom

'c' or 'd' for the time domain (continuous or discrete)

typ

string: 'eigen' for block diagonalization or 'schur' for Schur method.

X1,X2,X

square real matrices (X2 invertible), X symmetric

Description

`X=riccati(A,B,C,dom,[typ])` solves the Riccati equation:

$$A' * X + X * A - X * B * X + C = 0$$

in continuous time case, or:

$$A' * X * A - (A' * X * B1 / (B2 + B1' * X * B1)) * (B1' * X * A) + C - X$$

with $B=B1/B2*B1'$ in the discrete time case. If called with two output arguments, `riccati` returns `X1`, `X2` such that $X=X1/X2$.

See Also

`ricc`, `ric_desc`

Name

routh_t — Routh's table

```
r=routh_t(p)
r=routh_t(h [,k])
```

Parameters

p
a real polynomial

h
a real SISO transfer system

k
a real polynomial or a scalar

r
a matrix

Description

`r=routh_t(p)` computes Routh's table of the polynomial `h`.

`r=routh_t(h,k)` computes Routh's table of denominator of the system described by transfer matrix SISO `h` with the feedback by the gain `k`.

If `k=poly(0,'k')` we will have a polynomial matrix with dummy variable `k`, formal expression of the Routh table.

Examples

```
s=%s;
P=5*s^3-10*s^2+7*s+20;
routh_t(P)

//transfer function with formal feedback
routh_t((1+s)/P,poly(0,'k'))

// One of the coefficients in the polynomial equals zero
P1=2*s^3-24*s+32;
routh_t(P1)

// A row full of zeros
P2=s^4-6*s^3+10*s^2-6*s+9;
routh_t(P2)
```

See Also

roots, kpure

Bibliography

<http://controls.engin.umich.edu/wiki/index.php/RouthStability>

http://www.jdotec.net/s3i/TD_Info/Routh/Routh.pdf

Comments on the Routh-Hurwitz criterion, Shamash, Y., Automatic Control, IEEE T.A.C Volume 25, Issue 1, Feb 1980 Page(s): 132 - 133

Name

rowinout — inner-outer factorization

```
[Inn,X,Gbar]=rowinout(G)
```

Parameters

G
linear system (syslin list) $[A, B, C, D]$

Inn
inner factor (syslin list)

Gbar
outer factor (syslin list)

X
row-compressor of **G** (syslin list)

Description

Inner-outer factorization (and row compression) of $(l \times p)$ $G = [A, B, C, D]$ with $l \geq p$.

G is assumed to be tall ($l \geq p$) without zero on the imaginary axis and with a **D** matrix which is full column rank.

G must also be stable for having **Gbar** stable.

G admits the following inner-outer factorization:

$$G = \begin{bmatrix} \text{Inn} & | & \text{Gbar} \\ & | & 0 \end{bmatrix}$$

where **Inn** is square and inner (all pass and stable) and **Gbar** square and outer i.e: **Gbar** is square bi-proper and bi-stable (**Gbar** inverse is also proper and stable);

Note that:

$$X^*G = \begin{bmatrix} \text{Gbar} \\ - \\ 0 \end{bmatrix}$$

is a row compression of **G** where $X = \text{Inn}$ inverse is all-pass i.e:

$$X^T(-s) X(s) = \text{Identity}$$

(for the continuous time case).

See Also

syslin , colinout

Name

rowregul — removing poles and zeros at infinity

```
[Stmp,ws]=rowregul(Sl,alfa,beta)
```

Parameters

Sl,Stmp
syslin lists

alfa,beta
real numbers (new pole and zero positions)

Description

computes a postfilter **WS** such that **Stmp=WS*Sl** is proper and with full rank **D** matrix.

Poles at infinity of **Sl** are moved to **alfa**;

Zeros at infinity of **Sl** are moved to **beta**;

Sl is assumed to be a right invertible linear system (syslin list) in state-space representation with possibly a polynomial **D** matrix.

This function is the dual of colregul (see function code).

Examples

```
s=%s;  
w=[1/s,0;s/(s^3+2),2/s];  
Sl=tf2ss(w);  
[Stmp,ws]=rowregul(Sl,-1,-2);  
Stmp('D')      // D matrix of Stmp  
clean(ss2tf(Stmp))
```

See Also

invsyslin , colregul

Authors

F. D. , R. N. ;

Name

rtitr — discrete time response (transfer matrix)

```
[y]=rtitr(Num,Den,u [,up,yp])
```

Parameters

Num,Den

polynomial matrices (resp. dimensions : $n \times m$ and $n \times n$)

u

real matrix (dimension $m \times (t+1)$)

up,yp

real matrices (up dimension $m \times (\max_i(\text{degree}(\text{Den})))$ (default values=0) , yp dimension $n \times (\max_i(\text{degree}(\text{Den})))$)

y

real matrix

Description

`y=rtitr(Num,Den,u [,up,yp])` returns the time response of the discrete time linear system with transfer matrix $\text{Den}^{-1} \text{Num}$ for the input u, i.e y and u are such that $\text{Den } y = \text{Num } u$ at $t=0,1,\dots$

If $d1=\max_i(\text{degree}(\text{Den}))$, and $d2=\max_i(\text{degree}(\text{Num}))$ the polynomial matrices $\text{Den}(z)$ and $\text{Num}(z)$ may be written respectively as:

$$\begin{aligned} D(z) &= D_0 + D_1 z + \dots + D_{d1} z^{d1} \\ N(z) &= N_0 + N_1 z + \dots + N_{d2} z^{d2} \end{aligned}$$

and $\text{Den } y = \text{Num } u$ is interpreted as the recursion:

$$D(0)y(t) + D(1)y(t+1) + \dots + D(d1)y(t+d1) = N(0)u(t) + \dots + N(d2)u(t+d2)$$

It is assumed that $D(d1)$ is non singular.

The columns of u are the inputs of the system at $t=0,1,\dots,T$:

$$u = [u(0), u(1), \dots, u(T)]$$

The outputs at $t=0,1,\dots,T+d1-d2$ are the columns of the matrix y:

$$y = [y(0), y(1), \dots, y(T+d1-d2)]$$

up and yp define the initial conditions for $t < 0$ i.e

$$up = [u(-d1), \dots, u(-1)]$$

```
yp = [y(-d1), ... y(-1) ]
```

Depending on the relative values of `d1` and `d2`, some of the leftmost components of `up`, `yp` are ignored. The default values of `up` and `yp` are zero: `up = 0*ones(m,d1)`, `yp=0*ones(n,d1)`

Examples

```
z=poly(0,'z');
Num=1+z;Den=1+z;u=[1,2,3,4,5];
rtitr(Num,Den,u)-u
//Other examples
//siso
//causal
n1=1;d1=poly([1 1],'z','coeff');          // y(j)=-y(j-1)+u(j-1)
r1=[0 1 0 1 0 1 0 1 0 1 0];
r=rtitr(n1,d1,ones(1,10));norm(r1-r,1)
//hot restart
r=rtitr(n1,d1,ones(1,9),1,0);norm(r1(2:11)-r)
//non causal
n2=poly([1 1 1],'z','coeff');d2=d1;      // y(j)=-y(j-1)+u(j-1)+u(j)+u(j+1)
r2=[2 1 2 1 2 1 2 1 2];
r=rtitr(n2,d2,ones(1,10));norm(r-r2,1)
//hot restart
r=rtitr(n2,d2,ones(1,9),1,2);norm(r2(2:9)-r,1)
//
//MIMO example
//causal
d1=d1*diag([1 0.5]);n1=[1 3 1;2 4 1];r1=[5;14]*r1;
r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//polynomial n1 (same ex.)
n1(1,1)=poly(1,'z','c');r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//non causal
d2=d1;n2=n2*n1;r2=[5;14]*r2;
r=rtitr(n2,d2,ones(3,10));norm(r2-r)
//
r=rtitr(n2,d2,ones(3,9),[1;1;1],[10;28]);
norm(r2(:,2:9)-r,1)
//
// State-space or transfer
a = [0.21 , 0.63 , 0.56 , 0.23 , 0.31
      0.76 , 0.85 , 0.66 , 0.23 , 0.93
      0 , 0.69 , 0.73 , 0.22 , 0.21
      0.33 , 0.88 , 0.2 , 0.88 , 0.31
      0.67 , 0.07 , 0.54 , 0.65 , 0.36];
b = [0.29 , 0.5 , 0.92
      0.57 , 0.44 , 0.04
      0.48 , 0.27 , 0.48
      0.33 , 0.63 , 0.26
      0.59 , 0.41 , 0.41];
c = [0.28 , 0.78 , 0.11 , 0.15 , 0.84
```

```
    0.13 , 0.21 , 0.69 , 0.7 , 0.41];  
d = [0.41 , 0.11 , 0.56  
     0.88 , 0.2 , 0.59];  
s=syslin('d',a,b,c,d);  
h=ss2tf(s);num=h('num');den=h('den');den=den(1,1)*eye(2,2);  
u=1;u(3,10)=0;r3=flts(u,s);  
r=rtitr(num,den,u);norm(r3-r,1)
```

See Also

ltitr , exp , flts

Name

sensi — sensitivity functions

```
[Se,Re,Te]=sensi(G,K)
[Si,Ri,Ti]=sensi(G,K,flag)
```

Parameters

G
standard plant (sys lin list)

K
compensator (sys lin list)

flag
character string 'o' (default value) or 'i'

Se
output sensitivity function $(I+G*K)^{-1}$

Re
 $K*Se$

Te
 $G*K*Se$ (output complementary sensitivity function)

Description

sensi computes sensitivity functions. If G and K are given in state-space form, the systems returned are generically minimal. Calculation is made by lft, e.g., Se can be given by the commands `P = augment(G, 'S')`, `Se=lft(P,K)`. If flag = 'i', `[Si,Ri,Ti]=sensi(G,K, 'i')` returns the input sensitivity functions.

```
[Se;Re;Te]= [inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];
[Si;Ri;Ti]= [inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+K*G)];
```

Examples

```
G=ssrand(1,1,3);K=ssrand(1,1,3);
[Se,Re,Te]=sensi(G,K);
Se1=inv(eye()+G*K); //Other way to compute
ss2tf(Se) //Se seen in transfer form
ss2tf(Se1)
ss2tf(Te)
ss2tf(G*K*Se1)
[Si,Ri,Ti]=sensi(G,K, 'i');
w1=[ss2tf(Si);ss2tf(Ri);ss2tf(Ti)]
w2=[ss2tf(inv(eye()+K*G));ss2tf(G*inv(eye()+K*G));ss2tf(K*G*inv(eye()+K*G))];
clean(w1-w2)
```

See Also

augment, lft, h_cl

Name

sgrid — s-plane grid lines.

```
sgrid()  
sgrid('new')  
sgrid(zeta,wn [,color])
```

Description

Used in conjunction with `evans`, plots lines of constant damping ratio (`zeta`) and natural frequency (`wn`).

`sgrid()`
add a grid over an existing continuous s-plane root with default values for `zeta` and `wn`.

`sgrid('new')`
clears the graphic screen and then plots a default s-plane grid

`sgrid(zeta,wn [,color])`
same as `sgrid()` but uses the provided damping ratio and natural frequency.

Examples

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));  
evans(H,100)  
sgrid()  
sgrid(0.6,2,7)
```

See Also

`evans`

Name

`show_margins` — display gain and phase margin and associated crossover frequencies

```
show_margins(h)
show_margins(h, 'bode')
show_margins(h, 'nyquist')
```

Parameters

`h`
a SISO linear system (see `:syslin`).

Description

Given a SISO linear system in continuous or discrete time, `show_margins` display gain and phase margin and associated crossover frequencies on a bode (the default) or nyquist representation of the frequency response of the system.

Examples

```
//continuous case
h=syslin('c',0.02909+0.11827*s+0.12823*s^2+0.35659*s^3+0.256*s^4+0.1*s^5, .
0.0409+0.1827*s+1.28225*s^2+3.1909*s^3+2.56*s^4+s^5);
show_margins(h)
show_margins(h, 'nyquist')

//discrete case
h = syslin(0.1,0.01547+0.01599*z ,%z^2-1.81*z+0.9048)
show_margins(h)
show_margins(h, 'nyquist')
```

See Also

`p_margin`, `g_margin`, `bode`, `nyquist`

Authors

Serge Steer, INRIA

Name

sident — discrete-time state-space realization and Kalman gain

```
[(A,C)(,B(,D))(,K,Q,Ry,S)(,rcnd)] = sident(meth,job,s,n,l,R(tol,t,Ai,Ci,printw))
```

Parameters

meth

integer option to determine the method to use:

=

1 : MOESP method with past inputs and outputs;

=

2 : N4SID method;

=

3 : combined method: A and C via MOESP, B and D via N4SID.

job

integer option to determine the calculation to be performed:

=

1 : compute all system matrices, A, B, C, D;

=

2 : compute the matrices A and C only;

=

3 : compute the matrix B only;

=

4 : compute the matrices B and D only.

s

the number of block rows in the processed input and output block Hankel matrices. $s > 0$.

n

integer, the order of the system

l

integer, the number of the system outputs

R

the $2*(m+1)*s$ -by- $2*(m+1)*s$ part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices.

tol

(optional) tolerance used for estimating the rank of matrices. If $\text{tol} > 0$, then the given value of tol is used as a lower bound for the reciprocal condition number; an m -by- n matrix whose estimated condition number is less than $1/\text{tol}$ is considered to be of full rank. Default: $m*n*\text{epsilon\_machine}$ where epsilon_machine is the relative machine precision.

t

(optional) the total number of samples used for calculating the covariance matrices. Either $t = 0$, or $t \geq 2*(m+1)*s$. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If $t = 0$, then K, Q, Ry, and S are not computed. Default: $t = 0$.

A_i

real matrix

C_i

real matrix

printw

(optional) switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: printw = 0.

A

real matrix

C

real matrix

B

real matrix

D

real matrix

K

real matrix, kalman gain

Q

(optional) the n -by- n positive semidefinite state covariance matrix used as state weighting matrix when computing the Kalman gain.

R_Y

(optional) the l -by- l positive (semi)definite output covariance matrix used as output weighting matrix when computing the Kalman gain.

S

(optional) the n -by- l state-output cross-covariance matrix used as cross-weighting matrix when computing the Kalman gain.

rcnd

(optional) vector of length l_r , containing estimates of the reciprocal condition numbers of the matrices involved in rank decisions, least squares, or Riccati equation solutions, where $l_r = 4$, if Kalman gain matrix K is not required, and $l_r = 12$, if Kalman gain matrix K is required.

Description

SIDENT function for computing a discrete-time state-space realization (A, B, C, D) and Kalman gain K using SLICOT routine IB01BD.

```
[A, C, B, D] = sident(meth, 1, s, n, l, R)
[A, C, B, D, K, Q, Ry, S, rcnd] = sident(meth, 1, s, n, l, R, tol, t)
[A, C] = sident(meth, 2, s, n, l, R)
B = sident(meth, 3, s, n, l, R, tol, 0, Ai, Ci)
[B, K, Q, Ry, S, rcnd] = sident(meth, 3, s, n, l, R, tol, t, Ai, Ci)
```

```
[B,D] = sident(meth,4,s,n,l,R,tol,0,Ai,Ci)
[B,D,K,Q,Ry,S,rcnd] = sident(meth,4,s,n,l,R,tol,t,Ai,Ci)
```

SIDENT computes a state-space realization (A,B,C,D) and the Kalman predictor gain K of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP, N4SID, or their combination).

The model structure is :

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + Ke(k), \quad k \geq 1, \\ y(k) &= Cx(k) + Du(k) + e(k), \end{aligned}$$

where $x(k)$ is the n -dimensional state vector (at time k),

$u(k)$ is the m -dimensional input vector,

$y(k)$ is the l -dimensional output vector,

$e(k)$ is the l -dimensional disturbance vector,

and A, B, C, D, and K are real matrices of appropriate dimensions.

Comments

1. The n -by- n system state matrix A, and the p -by- n system output matrix C are computed for job ≤ 2 .
2. The n -by- m system input matrix B is computed for job ≤ 2 .
3. The l -by- m system matrix D is computed for job = 1 or 4.
4. The n -by- l Kalman predictor gain matrix K and the covariance matrices Q, Ry, and S are computed for $t > 0$.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1, -0.1, 0.2;
      0.1, 0, -0.1, -0.1;
      -0.4, -0.6, -0.7, -0.1;
      0.8, 0, -0.6, -0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

S = 15;
N = 3;
METH=1;
[R,N1] = findR(S,Y',U',METH);
[A,C,B,D,K] = sident(METH,1,S,N,1,R);
SYS1=syslin(1,A,B,C,D);
SYS1.X0 = inistate(SYS1,Y',U');
```

```
Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])

METH = 2;
[R,N1,SVAL] = findR(S,Y',U',METH);
tol = 0;
t = size(U',1)-2*S+1;

[A,C,B,D,K] = sident(METH,1,S,N,1,R,tol,t)
SYS1=syslin(1,A,B,C,D)
SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findBD , sorder

Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999. Revisions: May 2000, July 2000.

Name

sm2des — system matrix to descriptor

```
[Des]=sm2des(Sm);
```

Parameters

Sm
polynomial matrix (pencil system matrix)

Des
descriptor system (`list('des',A,B,C,D,E)`)

Description

Utility function: converts the system matrix:

$$Sm = \begin{bmatrix} -sE + A & B \\ C & D \end{bmatrix}$$

to descriptor system `Des=list('des',A,B,C,D,E)`.

See Also

ss2des , sm2ss

Name

sm2ss — system matrix to state-space

```
[Sl]=sm2ss(Sm);
```

Parameters

Sm
polynomial matrix (pencil system matrix)

Sl
linear system (syslin list)

Description

Utility function: converts the system matrix:

$$S_m = \begin{bmatrix} -sI + A & B; \\ C & D \end{bmatrix}$$

to linear system in state-space representation (syslin) list.

See Also

ss2des

Name

sorder — computing the order of a discrete-time system

```
[Ro(,n,sval,rcnd)] = sorder(meth,alg,jobd,batch,conct,s,Y(,U,tol,
printw,ldwork,Ri))
```

Parameters

meth

integer option to determine the method to use:

=

1 : MOESP method with past inputs and outputs;

=

2 : N4SID method.

alg

integer option to determine the algorithm for computing the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

=

1 : Cholesky algorithm on the correlation matrix;

=

2 : fast QR algorithm;

=

3 : standard QR algorithm.

jobd

integer option to specify if the matrices B and D should later be computed using the MOESP approach:

=

1 : the matrices B and D should later be computed using the MOESP approach;

=

2 : the matrices B and D should not be computed using the MOESP approach.

This parameter is not relevant for meth = 2.

batch

integer option to specify whether or not sequential data processing is to be used, and, for sequential processing, whether or not the current data block is the first block, an intermediate block, or the last block, as follows:

=

1 : the first block in sequential data processing;

=

2 : an intermediate block in sequential data processing;

=

3 : the last block in sequential data processing;

=

4 : one block only (non-sequential data processing).

conct

integer option to specify whether or not the successive data blocks in sequential data processing belong to a single experiment, as follows:

=

1 : the current data block is a continuation of the previous data block and/or it will be continued by the next data block;

=

2 : there is no connection between the current data block and the previous and/or the next ones.

This parameter is not used if batch = 4.

s

the number of block rows in the input and output block Hankel matrices to be processed. $s > 0$

Y

the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U

(optional) the t-by-m input-data sequence matrix. Column j of U contains the t values of the j-th input component for consecutive time increments. Default: U = [].

tol

(optional) vector of length 2 containing tolerances: tol(1) - tolerance used for estimating the rank of matrices. If $\text{tol}(1) > 0$, then the given value of tol(1) is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than $1/\text{tol}(1)$ is considered to be of full rank. If $\text{tol}(1) \leq 0$, then a default value $m*n*\text{epsilon\_machine}$ is used, where epsilon_machine is the relative machine precision.

tol(2) - tolerance used for determining an estimate of the system order. If $\text{tol}(2) \geq 0$, the estimate is indicated by the index of the last singular value greater than or equal to tol(2). (Singular values less than tol(2) are considered as zero.) When $\text{tol}(2) = 0$, an internally computed default value, $\text{tol}(2) = s*\text{epsilon\_machine}*sval(1)$, is used, where sval(1) is the maximal singular value, and epsilon_machine the relative machine precision. When $\text{tol}(2) < 0$, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor.

Default: $\text{tol}(1:2) = [0, -1]$.

printw

(optional) switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: printw = 0.

ldwork

(optional) the workspace size. Default : computed by the formulas

```
nr = 2*( m + l )*s
LDWORK = ( t - 2*s + 3 + 64 )*nr
if ( CSIZE > MAX( nr*nr + t*( m + l ) + 16, 2*nr ) ) then
    LDWORK = MIN( LDWORK, CSIZE - nr*nr - t*( m + l ) - 16 )
else
    LDWORK = MIN( LDWORK, MAX( 2*nr, CSIZE/2 ) )
```

```
end if
```

LDWORK = MAX(minimum workspace size needed, LDWORK) where CSIZE is the cache size in double precision words.

If LDWORK is specified less than the minimum workspace size needed, that minimum value is used instead.

Ri

(optional) if batch = 2 or 3, the $2*(m+1)*s$ -by- $2*(m+1)*s$ (upper triangular, if alg $\neq$ 2) part of R must contain the (upper triangular) matrix R computed at the previous call of this mexfile in sequential data processing. If conct = 1, R has an additional column, also set at the previous call.

If alg = 2, R has $m+1+1$ additional columns, set at the previous call.

This parameter is not used for batch = 1 or batch = 4.

Ro

if batch = 3 or 4, the $2*(m+1)*s$ -by- $2*(m+1)*s$ part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices. If batch = 1 or 2, then R contains intermediate results needed at the next call of this mexfile. If batch = 1 or 2 and conct = 1, R has an additional column, also set before return. If batch = 1 or 2 and alg = 2, R has $m+1+1$ additional columns, set before return.

n

the order of the system.

sval

(optional) the singular values used for estimating the order of the system.

rcnd

(optional) if meth = 2, vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

Description

sorder - function for computing the order of a discrete-time system using SLICOT routine IB01AD.

For one block (data sequences Y, U): `[R,n,sval,rcnd] = sorder(meth,alg,jobd,4,conct,s,Y,U);`

For f blocks (data sequences Yj, Uj, j = 1 : f):

```
R = sorder(meth,alg,jobd,1,conct,s,Y1,U1);
for j = 2 : f - 1
    R = sorder(meth,alg,jobd,2,conct,s,Yj,Uj,tol,printw,ldwork,R)
end
[R,n,sval,rcnd] = sorder(meth,alg,jobd,3,conct,s,Yf,Uf,tol);
```

sorder preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP and N4SID), and then estimates the order of a discrete-time realization.

The model structure is :

$$x(k+1) = Ax(k) + Bu(k) + w(k), \quad k \geq 1,$$

$$y(k) = Cx(k) + Du(k) + e(k),$$

where $x(k)$ is the n -dimensional state vector (at time k),

$u(k)$ is the m -dimensional input vector,

$y(k)$ is the l -dimensional output vector,

$w(k)$ is the n -dimensional state disturbance vector,

$e(k)$ is the l -dimensional output disturbance vector,

and A , B , C , and D are real matrices of appropriate dimensions.

Comments

1. The Cholesy or fast QR algorithms can be much faster (for large data blocks) than QR algorithm, but they cannot be used if the correlation matrix, H^*H , is not positive definite. In such a case, the code automatically switches to the QR algorithm, if sufficient workspace is provided and $\text{batch} = 4$.

2. If ldwork is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

See Also

`findBD` , `sident`

Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999.; ; Revisions;; V. Sima, May 2000, July 2000.

Name

specfact — spectral factor

```
[W0,L]=specfact(A,B,C,D)
```

Description

Given a spectral density matrix $\phi(s)$:

$$R + C^*(s^*I - A)^{-1} B + B^*(-s^*I - A')^{-1} C \quad \text{with } R = D + D' > 0$$

specfact computes $W0$ and L such that $W(s) = W0 + L^*(s^*I - A)^{-1}B$ is a spectral factor of $\phi(s)$, i.e.

$$\phi(s) = W'(-s)W(s)$$

Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];D=1;s=poly(0,'s');
W1=syslin('c',A,B,C,D);
phi=gtild(W1,'c')+W1;
phis=clean(ss2tf(phi))
clean(phis-horner(phis,-s)); //check this is 0...
[A,B,C,D]=abcd(W1);
[W0,L]=specfact(A,B,C,D);
W=syslin('c',A,B,L,W0)
Ws=ss2tf(W);
horner(Ws,-s)*Ws
```

See Also

gtild, sfact, fspecg

Authors

F. D.

Name

ss2des — (polynomial) state-space to descriptor form

```
S=ss2des(Sl)
S=ss2des(Sl,flag)
```

Parameters

Sl
syslin list: proper or improper linear system.

flag
character string "withD"

S
list

Description

Given the linear system in state-space representation Sl (syslin list), with a D matrix which is either polynomial or constant, but not zero ss2des returns a descriptor system as list('des',A,B,C,0,E) such that:

```
Sl=C*(s*E-A)^(-1)*B
```

If the flag "withD" is given, S=list('des',A,B,C,D,E) with a D matrix of maximal rank.

Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];Sl=tf2ss(G);
S=ss2des(Sl)
S1=ss2des(Sl,"withD")
Des=des2ss(S);Des(5)=clean(Des(5))
Des1=des2ss(S1)
```

See Also

pol2des , tf2des , des2ss

Authors

F. D.; ;

Name

ss2ss — state-space to state-space conversion, feedback, injection

```
[Sl1, right, left]=ss2ss(Sl, T, [F, [G , [flag]]])
```

Parameters

Sl
linear system (syslin list) in state-space form

T
square (non-singular) matrix

Sl1, right, left
linear systems (syslin lists) in state-space form

F
real matrix (state feedback gain)

G
real matrix (output injection gain)

Description

Returns the linear system $Sl1=[A1, B1, C1, D1]$ where $A1=inv(T)*A*T$, $B1=inv(T)*B$, $C1=C*T$, $D1=D$.

Optional parameters F and G are state feedback and output injection respectively.

For example, $Sl1=ss2ss(Sl, T, F)$ returns Sl1 with:

$$sl1 = \begin{pmatrix} T^{-1} * (A + B * F) * T & T^{-1} * B \\ (C + D * F) * T & D \end{pmatrix}$$

and right is a non singular linear system such that $Sl1=Sl*right$.

$Sl1*inv(right)$ is a factorization of Sl.

$Sl1=ss2ss(Sl, T, 0*F, G)$ returns Sl1 with:

$$sl1 = \begin{pmatrix} T^{-1} * (A + G * C) * T & T^{-1} * (B + G * D) \\ C * T & D \end{pmatrix}$$

and left is a non singular linear system such that $Sl1=left*Sl$ (right=Id if F=0).

When both F and G are given, $Sl1=left*Sl*right$.

- When flag is used and flag=1 an output injection as follows is used

$$\begin{pmatrix} T^{-1} * (A + G * C) * T & T^{-1} * (B + G * D, -G) \\ C * T & (D, 0) \end{pmatrix}$$

and then a feedback is performed, F must be of size (m+p, n)

$$x \in \mathbb{R}^n, y \in \mathbb{R}^p, u \in \mathbb{R}^m$$

`right` and `left` have the following property:

```
S11 = left*sysdiag(sys,eye(p,p))*right
```

- When `flag` is used and `flag=2` a feedback (F must be of size (m, n)) is performed and then the above output injection is applied. `right` and `left` have the following property:

```
S11 = left*sysdiag(sys*right,eye(p,p)))
```

Examples

```
S1=ssrand(2,2,5); trzeros(S1)          // zeros are invariant:
S11=ss2ss(S1,rand(5,5),rand(2,5),rand(5,2));
trzeros(S11), trzeros(rand(2,2)*S11*rand(2,2))
// output injection [ A + GC, (B+GD, -G)]
//                  [   C   , (D   , 0)]
p=1,m=2,n=2; sys=ssrand(p,m,n);

// feedback (m,n) first and then output injection.

F1=rand(m,n);
G=rand(n,p);
[sys1,right,left]=ss2ss(sys,rand(n,n),F1,G,2);

// S11 equiv left*sysdiag(sys*right,eye(p,p)))

res=clean(ss2tf(sys1) - ss2tf(left*sysdiag(sys*right,eye(p,p))))

// output injection then feedback (m+p,n)
F2=rand(p,n); F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

// S11 equiv left*sysdiag(sys,eye(p,p))*right

res=clean(ss2tf(sys2)-ss2tf(left*sysdiag(sys,eye(p,p))*right))

// when F2= 0; sys1 and sys2 are the same
F2=0*rand(p,n);F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

res=clean(ss2tf(sys2)-ss2tf(sys1))
```

See Also

`projsl`, `feedback`

Name

ss2tf — conversion from state-space to transfer function

```
[h]=ss2tf(sl)
[Ds,NUM,chi]=ss2tf(sl)

[h]=ss2tf(sl,"b")
[Ds,NUM,chi]=ss2tf(sl,"b")

[h]=ss2tf(sl,rmax)
[Ds,NUM,chi]=ss2tf(sl,rmax)
```

Parameters

sl
linear system (syslin list)

h
transfer matrix

Description

Called with three outputs `[Ds,NUM,chi]=ss2tf(sl)` returns the numerator polynomial matrix NUM, the characteristic polynomial chi and the polynomial part DS separately i.e.:

```
h = NUM/chi + Ds
```

Method:

One uses the characteristic polynomial and $\det(A+E_{ij})=\det(A)+C(i,j)$ where C is the adjugate matrix of A.

With rmax or "b" argument uses a block diagonalization of sl.A matrix and applies "Leverrier" algorithm on blocks. If given, rmax controls the conditionning (see bdiag).

Examples

```
s=poly(0,'s');
h=[1,1/s;1/(s^2+1),s/(s^2-2)]
sl=tf2ss(h);
h=clean(ss2tf(sl))
[Ds,NUM,chi]=ss2tf(sl)
```

See Also

tf2ss , syslin , nlev , glever

Name

st_ility — stabilizability test

```
[ns, [nc, [,U [,Slo] ]]] = st_ility(Sl [,tol])
```

Parameters

Sl
syslin list (linear system)

ns
integer (dimension of stabilizable subspace)

nc
integer (dimension of controllable subspace $nc \leq ns$)

U
basis such that its *ns* (resp. *nc*) first components span the stabilizable (resp. controllable) subspace

Slo
a linear system (syslin list)

tol
threshold for controllability detection (see `contr`)

Description

`Slo = ( U' * A * U, U' * B, C * U, D, U' * x0 )` (syslin list) displays the stabilizable form of `Sl`. Stabilizability means $ns = nx$ (dim. of *A* matrix).

$$U' * A * U = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix} \quad U' * B = \begin{bmatrix} * \\ 0 \\ 0 \end{bmatrix}$$

where (A_{11}, B_1) ($\dim(A_{11}) = nc$) is controllable and A_{22} ($\dim(A_{22}) = ns - nc$) is stable. "Stable" means real part of eigenvalues negative for a continuous linear system, and magnitude of eigenvalues lower than one for a discrete-time system (as defined by `syslin`).

Examples

```
A=diag([0.9, -2, 3]); B=[0;0;1]; Sl=syslin('c', A, B, []);
[ns, nc, U]=st_ility(Sl);
U' * A * U
U' * B
[ns, nc, U]=st_ility(syslin('d', A, B, []));
U' * A * U
U' * B
```

See Also

`dt_ility`, `contr`, `stabil`, `ssrand`

Authors

S. Steer INRIA 1988

Name

stabil — stabilization

```
F=stabil(A,B,alfa)
K=stabil(Sys,alfa,beta)
```

Parameters

A
square real matrix ($n_x \times n_x$)

B
real matrix ($n_x \times n_u$)

alfa, beta
real or complex vector (in conjugate pairs) or real number.

F
real matrix ($n_x \times n_u$)

Sys
linear system (syslin list) (m inputs, p outputs).

K
linear system (p inputs, m outputs)

Description

F=stabil(A,B,alfa) returns a gain matrix **F** such that **A+B*F** is stable if pair (**A,B**) is stabilizable. Assignable poles are set to **alfa(1), alfa(2), ...**. If (**A,B**) is not stabilizable a warning is given and assignable poles are set to **alfa(1), alfa(2), ...**. If **alfa** is a number all eigenvalues are set to this **alfa** (default value is **alfa=-1**).

K=stabil(Sys,alfa,beta) returns **K**, a compensator for **Sys** such that (**A,B**)-controllable eigenvalues are set to **alfa** and (**C,A**)-observable eigenvalues are set to **beta**.

All assignable closed loop poles (which are given by the eigenvalues of **Aclosed=h_cl(Sys,K)**) are set to **alfa(i)**'s and **beta(j)**'s.

Examples

```
// Gain:
Sys=ssrand(0,2,5,list('st',2,3,3));
A=Sys('A');B=Sys('B');F=stabil(A,B);
spec(A) //2 controllable modes 2 unstable uncontrollable modes
//and one stable uncontrollable mode
spec(A+B*F) //the two controllable modes are set to -1.
// Compensator:
Sys=ssrand(3,2,5,list('st',2,3,3)); //3 outputs, 2 inputs, 5 states
//2 controllables modes, 3 controllable or stabilizable modes.
K=stabil(Sys,-2,-3); //Compensator for Sys.
spec(Sys('A'))
spec(h_cl(Sys,K)) //K Stabilizes what can be stabilized.
```

See Also

st_ility, contr, ppol

Name

svplot — singular-value sigma-plot

```
[SVM]=svplot(sl,[w])
```

Parameters

sl
syslin list (continuous, discrete or sampled system)

w
real vector (optional parameter)

Description

computes for the system $sl=(A, B, C, D)$ the singular values of its transfer function matrix:

```
or           $G(jw) = C(jw \cdot I - A)^{-1}B + D$   
or           $G(\exp(jw)) = C(\exp(jw) \cdot I - A)^{-1}B + D$   
or           $G(\exp(jwT)) = C(\exp(jw \cdot T) \cdot I - A)^{-1}B + D$ 
```

evaluated over the frequency range specified by **w**. (T is the sampling period, $T=sl('dt')$ for sampled systems).

sl is a **syslin** list representing the system $[A, B, C, D]$ in state-space form. **sl** can be continuous or discrete time or sampled system.

The i -th column of the output matrix **SVM** contains the singular values of G for the i -th frequency value $w(i)$.

```
SVM = svplot(sl)
```

is equivalent to

```
SVM = svplot(sl,logspace(-3,3)) (continuous)
```

```
SVM = svplot(sl,logspace(-3,%pi)) (discrete)
```

Examples

```
x=logspace(-3,3);  
y=svplot(ssrand(2,2,4),x);  
clf();plot2d1("oln",x',20*log(y')/log(10));  
xgrid(12)  
xtitle("Singular values plot","(Rd/sec)", "Db");
```

Authors

F.D; ;

Name

sysfact — system factorization

```
[S, Series]=sysfact(Sys, Gain, flag)
```

Parameters

Sys
syslin list containing the matrices $[A, B, C, D]$.

Gain
real matrix

flag
string 'post' or 'pre'

S
syslin list

Series
syslin list

Description

If flag equals 'post', sysfact returns in S the linear system with ABCD matrices $(A + B \cdot \text{Gain}, B, \text{Gain}, I)$, and Series, a minimal realization of the series system $\text{Sys} \cdot S$. If flag equals 'pre', sysfact returns the linear system $(A + \text{Gain} \cdot C, \text{Gain}, C, I)$ and Series, a minimal realization of the series system $S \cdot \text{Sys}$.

Examples

```
//Kalman filter
Sys=ssrand(3,2,4);Sys('D')=rand(3,2);
S=sysfact(Sys,lqr(Sys),'post');
ww=minss(Sys*S);
ss2tf(gtild(ww)*ww),Sys('D')'*Sys('D')
//Kernel
Sys=ssrand(2,3,4);
[X,d,F,U,k,Z]=abinv(Sys);
ss2tf(Sys*Z)
ss2tf(Sys*sysfact(Sys,F,'post')*U)
```

See Also

lqr , lqe

Authors

F.D.

Name

syssize — size of state-space system

```
[r,nx]=syssize(Sl)
```

Parameters

Sl
linear system (syslin list) in state-space

r
1 x 2 real vector

nx
integer

Description

returns in **r** the vector [number of outputs, number of inputs] of the linear system **Sl**. **nx** is the number of states of **Sl**.

See Also

size

Name

tf2des — transfer function to descriptor

```
S=tf2des(G)
S=tf2des(G,flag)
```

Parameters

G
linear system (sys lin list) with possibly polynomial D matrix

flag
character string "withD"

S
list

Description

Transfer function to descriptor form: $S=list('d',A,B,C,D,E)$

```
E*xdot = A*x+B*u
y = C*x + D*u
```

Note that $D=0$ if the optional parameter flag="withD" is not given. Otherwise a maximal rank D matrix is returned in the fifth entry of the list S

Examples

```
s=poly(0,'s');
G=[1/(s-1),s;1,2/s^3];
S1=tf2des(G);des2tf(S1)
S2=tf2des(G,"withD");des2tf(S2)
```

See Also

pol2des , tf2ss , ss2des , des2tf

Name

tf2ss — transfer to state-space

```
sl=tf2ss(h [,tol])
```

Parameters

h
rational matrix

tol
may be the constant rtol or the 2 vector [rtol atol]

rtol
tolerance used when evaluating observability.

atol
absolute tolerance used when evaluating observability.

sl
linear system (syslin list sl=[A,B,C,D(s)])

Description

transfer to state-space conversion:

$$h=C*(s*eye()-A)^{-1}*B+D(s)$$

Examples

```
s=poly(0,'s');  
H=[2/s,(s+1)/(s^2-5)];  
Sys=tf2ss(H)  
clean(ss2tf(Sys))
```

See Also

ss2tf , tf2des , des2tf

Name

time_id — SISO least square identification

```
[H [,err]]=time_id(n,u,y)
```

Parameters

n
order of transfer

u
one of the following

u1
a vector of inputs to the system

"impuls"
if y is an impulse response

"step"
if y is a step response.

y
vector of response.

H
rational function with degree n denominator and degree n-1 numerator if y(1)==0 or rational function with degree n denominator and numerator if y(1)<>0.

err
 $\|y - \text{impuls}(H, \text{npt})\|^2$, where `impuls(H, npt)` are the npt first coefficients of impulse response of H

Description

Identification of discrete time response. If y is strictly proper (y(1)=0) then `time_id` computes the least square solution of the linear equation: $\text{Den} \cdot y - \text{Num} \cdot u = 0$ with the constraint $\text{coeff}(\text{Den}, n) := 1$. if $y(1) \neq 0$ then the algorithm first computes the proper part solution and then add y(1) to the solution

Examples

```
z=poly(0,'z');  
h=(1-2*z)/(z^2-0.5*z+5)  
rep=[0;ldiv(h('num'),h('den'),20)]; //impulse response  
H=time_id(2,'impuls',rep)  
// Same example with flts and u  
u=zeros(1,20);u(1)=1;  
rep=flts(u,tf2ss(h)); //impulse response  
H=time_id(2,u,rep)  
// step response  
u=ones(1,20);  
rep=flts(u,tf2ss(h)); //step response.  
H=time_id(2,'step',rep)  
H=time_id(3,u,rep) //with u as input and too high order required
```


See Also

imrep2ss , arl2 , armax , frep2tf

Authors

Serge Steer INRIA

Name

trzeros — transmission zeros and normal rank

```
[tr]=trzeros(Sl)
[nt,dt,rk]=trzeros(Sl)
```

Parameters

Sl
linear system (syslin list)

nt
complex vectors

dt
real vector

rk
integer (normal rank of Sl)

Description

Called with one output argument, `trzeros(Sl)` returns the transmission zeros of the linear system `Sl`.

`Sl` may have a polynomial (but square) `D` matrix.

Called with 2 output arguments, `trzeros` returns the transmission zeros of the linear system `Sl` as `tr=nt./dt`;

(Note that some components of `dt` may be zeros)

Called with 3 output arguments, `rk` is the normal rank of `Sl`

Transfer matrices are converted to state-space.

If `Sl` is a (square) polynomial matrix `trzeros` returns the roots of its determinant.

For usual state-space system `trzeros` uses the state-space algorithm of Emami-Naeni and Van Dooren.

If `D` is invertible the transmission zeros are the eigenvalues of the "A matrix" of the inverse system :
 $A - B \cdot \text{inv}(D) \cdot C$;

If $C \cdot B$ is invertible the transmission zeros are the eigenvalues of $N \cdot A \cdot M$ where $M \cdot N$ is a full rank factorization of $\text{eye}(A) - B \cdot \text{inv}(C \cdot B) \cdot C$;

For systems with a polynomial `D` matrix zeros are calculated as the roots of the determinant of the system matrix.

Caution: the computed zeros are not always reliable, in particular in case of repeated zeros.

Examples

```
w1=ssrand(2,2,5);trzeros(w1) //call trzeros
roots(det(systmat(w1))) //roots of det(system matrix)
s=poly(0,'s');W=[1/(s+1);1/(s-2)];W2=(s-3)*W*W';[nt,dt,rk]=trzeros(W2);
```

```
St=systmat(tf2ss(w2));[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(St);  
St1=Q*St*Z;rowf=(Qd(1)+Qd(2)+1):(Qd(1)+Qd(2)+Qd(3));  
colf=(Zd(1)+Zd(2)+1):(Zd(1)+Zd(2)+Zd(3));  
roots(St1(rowf,colf), nt./dt //By Kronecker form
```

See Also

gspec , kroneck

Name

ui_observer — unknown input observer

```
[UIobs, J, N]=ui_observer(Sys, reject, C1, D1)
[UIobs, J, N]=ui_observer(Sys, reject, C1, D1, flag, alfa, beta)
```

Parameters

Sys
syslin list containing the matrices (A, B, C2, D2).

reject
integer vector, indices of inputs of Sys which are unknown.

C1
real matrix

D1
real matrix. C1 and D1 have the same number of rows.

flag
string 'ge' or 'st' (default) or 'pp'.

alfa
real or complex vector (loc. of closed loop poles)

beta
real or complex vector (loc. of closed loop poles)

Description

Unknown input observer.

Sys: $(w, u) \rightarrow y$ is a $(A, B, C2, D2)$ syslin linear system with two inputs w and u , w being the unknown input. The matrices B and $D2$ of Sys are (implicitly) partitioned as: $B = [B1, B2]$ and $D2 = [D21, D22]$ with $B1 = B(:, reject)$ and $D21 = D2(:, reject)$ where $reject$ = indices of unknown inputs. The matrices $C1$ and $D1$ define $z = C1 \cdot x + D1 \cdot (w, u)$, the to-be-estimated output.

The matrix $D1$ is (implicitly) partitioned as $D1 = [D11, D12]$ with $D11 = D(:, reject)$

The data (Sys, reject, C1, D1) define a 2-input 2-output system:

$$\begin{aligned} \dot{x} &= A x + B1 w + B2 u \\ z &= C1 x + D11 w + D12 u \\ y &= C2 x + D21 w + D22 u \end{aligned}$$

An observer $(u, y) \rightarrow \hat{z}$ is looked for the output z .

flag='ge' no stability constraints flag='st' stable observer (default) flag='pp' observer with pole placement alfa,beta = desired location of closed loop poles (default -1, -2) J=y-output to x-state injection. N=y-output to z-estimated output injection.

UIobs = linear system $(u, y) \rightarrow \hat{z}$ such that: The transfer function: $(w, u) \rightarrow z$ equals the composed transfer function: $[0, I; UIobs \text{ Sys}] (w, u) \rightarrow (u, y) \rightarrow \hat{z}$ i.e. transfer function of system $\{A, B, C1, D1\}$ equals transfer function $UIobs * [0, I; Sys]$

Stability (resp. pole placement) requires detectability (resp. observability) of (A,C2).

Examples

```
A=diag([3,-3,7,4,-4,8]);
B=[eye(3,3);zeros(3,3)];
C=[0,0,1,2,3,4;0,0,0,0,0,1];
D=[1,2,3;0,0,0];
rand('seed',0);w=ss2ss(syslin('c',A,B,C,D),rand(6,6));
[A,B,C,D]=abcd(w);
B=[B,matrix(1:18,6,3)];D=[D,matrix(-(1:6),2,3)];
reject=1:3;
Sys=syslin('c',A,B,C,D);
N1=[-2,-3];C1=-N1*C;D1=-N1*D;
nw=length(reject);nu=size(Sys('B'),2)-nw;
ny=size(Sys('C'),1);nz=size(C1,1);
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1);

W=[zeros(nu,nw),eye(nu,nu);Sys];UIobsW=UIobs*W;
//(w,u) --> z=UIobs*[0,I;Sys](w,u)
clean(ss2tf(UIobsW));
wu_to_z=syslin('c',A,B,C1,D1);clean(ss2tf(wu_to_z));
clean(ss2tf(wu_to_z)-ss2tf(UIobsW),1.d-7)
/////2nd example/////
nx=2;ny=3;nwu=2;Sys=ssrand(ny,nwu,nx);
C1=rand(1,nx);D1=[0,1];
UIobs=ui_observer(Sys,1,C1,D1);
```

See Also

cainv , ddp , abinv

Authors

F.D.

Name

unobs — unobservable subspace

```
[n, [U]]=unobs(A, C, [tol])
```

Parameters

A, C

real matrices

tol

tolerance used when evaluating ranks (QR factorizations).

n

dimension of unobservable subspace.

U

orthogonal change of basis which puts (A, B) in canonical form.

Description

`[n, [U]]=unobs(A, C, [tol])` gives the unobservable form of an (A, C) pair. The n first columns of U make a basis for the unobservable subspace.

The (2, 1) block (made of last $n \times -n$ rows and n first columns) of $U' * A * U$ is zero and the n first columns of $C * U$ are zero.

Examples

```
A=diag([1, 2, 3]); C=[1, 0, 0];  
unobs(A, C)
```

See Also

contr , contrss , canon , cont_mat , spantwo , dt_ility

Name

zeropen — zero pencil

```
[Z,U]=zeropen(Sl)
```

Parameters

Sl
a linear system (sys lin list in state-space form $[A, B, C, D]$)

Z
matrix pencil $Z=s^*E-A$

U
square orthogonal matrix

Description

$Z = sE - F$ is the zero pencil of the linear system Sl with matrices $[A, B, C, D]$. Utility function.

With U row compression of $[B; D]$ i.e, $U^*[B; D]=[0; *]$; one has:

$$U^* \begin{bmatrix} -sI+A & B \\ C & D \end{bmatrix} = \begin{bmatrix} Z & 0 \\ * & * \end{bmatrix}$$

The zeros of Z are the zeros of Sl .

See Also

systmat , kroneck

Name

zgrid — zgrid plot

```
zgrid()
```

Description

plots z-plane grid lines: lines of constant damping factor (zeta) and natural frequency (ω_n) are drawn in within the unit Z-plane circle.

Iso-frequency curves are shown in frequency*step on the interval [0,0.5]. Upper limit corresponds to Shannon frequency ($1/dt > 2*f$).

See Also

frep2tf , freson

Part X. Data Structures

Name

`cell` — Create a cell array of empty matrices.

```
c=cell()  
c=cell(m1)  
c=cell(m1, m2)  
c=cell(m1, m2, ..., mn)  
c=cell(x)
```

Parameters

`x`
Vector containing the dimensions of the cell to create.

`m1, m2,..`
Dimensions of the cell to create.

Description

Returns the create cell of empty matrices.

`cell()`
returns a $(0, 0)$ cell array of empty matrices.

`cell(m1)`
returns a $(m1, m1)$ cell array of empty matrices.

`cell(m1,m2)`
returns a $(m1, m2)$ cell array of empty matrices.

`cell(m1,m2,...,mn)`
creates a $(m1, m2, \dots, mn)$ cell array of empty matrices.

`cell(x)`
returns a cell array of empty matrices with: the first dimension of the cell array is $x(1)$, the second dimension is $x(2)$, ...

Remarks

`cell(x)` is not the same size that `x`.

`cell()` is equivalent to `cell(0)`.

If `A` is a cell array, you can access the contents of an element of `A` by using `A(m1, m2, ..., mn).entries`, the expression `A(1,1) = zeros(2,2)` is not valid, the right syntax is `A(1,1).entries = zeros(2,2)`.

If `A` is a cell array, you can get its dimensions by using `A.dims`.

Examples

```
a=cell(3)  
b=cell(3,1)  
c=cell([2,3,4])
```

```
// Assigning cell entries
b=cell(3,1);

// Assigning the first element of b using the 'entries' field
b(1).entries=1:3

// Assigning the second element of b using the 'entries' field
b(2).entries='Scilab'

// Assigning the third element of b using the 'entries' field
b(3).entries=poly(1:3,'s')

// Assigning sub-cells
X=cell(3,2);
X(:,1)=b

// Extracting a sub-cell: result is a cell
b(1)
b(1:2)

// Extracting a sub-cell value: result is an array
b(1).entries

// Dimensions of b
b.dims
```

See Also

eye, ones, zeros

Name

definedfields — return index of list's defined fields

```
k=definedfields(l)
```

Parameters

l
a list , tlist or mlist variable.

k
a vector of index.

Description

If **l** is a list tlist mlist **k=definedfields(l)** returns in **k** the indices of the defined list fields. This function is useful because indexing undefined fields produces an error.

Examples

```
l=list(1);l(3)=5
k=definedfields(l)

t=tlist('x');t(5)=4
definedfields(t)

m=mlist(['m','a','b']);m.b='sdfgfgd'
definedfields(m)
```

See Also

list , tlist , mlist , insertion , extraction

Name

fieldnames — returns the tlist, mlist or struct field names

```
f=fieldnames(x)
```

Parameters

x
a tlist or an mlist or a struct.

f
column vector of strings

Description

This function returns the tlist, mlist, cell or struct field names.

Examples

```
clear t;  
t.a=1;  
t.b=2;  
fieldnames(t)  
fieldnames(1/%s)  
fieldnames(tf2ss(1/%s))
```

See Also

extraction , getfield , tlist , mlist , struct

Name

getfield — list field extraction

```
[x,...]=getfield(i,l)
```

Parameters

- x
matrix of any possible types
- l
list, tlist or mlist variable
- i
field index, see extraction for more details.

Description

This function is an equivalent of `[x,...]=l(i)` syntax for field extraction with the only difference that it also applies to `mlist` objects.

Examples

```
l=list(1, 'qwerw', %s)
[a,b]=getfield([3 2], l)

a=hypermat([2,2,2], rand(1:2^3)); // hypermatrices are coded using mlists
a(1) // the a(1,1,1) entry
getfield(1,a) // the mlist first field
```

See Also

extraction

Name

hypermat — initialize an N dimensional matrices

```
M=hypermat(dims [,v])
```

Parameters

dims

vector of hypermatrix dimensions

v

vector of hypermatrix entries (default value `zeros(prod(dims),1)`)

Description

Initialize an hypermatrix whose dimensions are given in the vector `dims` and entries are given in optional argument `v`

M data structure contains the vector of matrix dimensions `M('dims')` and the vector of entries `M('entries')` such as the leftmost subscripts vary first
`[M(1,1,...);...;M(n1,1,...);...;M(1,n2,...);...;M(n1,n2,...);...]`

Examples

```
M=hypermat([2 3 2 2],1:24)
```

See Also

hypermatrices , zeros , ones , grand , matrix

Name

hypermatrices — Scilab object, N dimensional matrices in Scilab

Description

Hypermatrix type allows to manipulate multidimensional arrays

They can be defined by extension of 2D matrices as follows `a=[1 2;3 4];a(:,:,2)=rand(2,2)`

or directly using `hypermat` function

Entries can be real or complex numbers, polynomials, rationals, strings, booleans.

Hypermatrices are `mlists`: `mlist(['hm','dims','entries'],sz,v)` where `sz` is the row vector of dimensions and `v` the column vector of entries (first dimension are stored first)

NOTES: The number of dimension of hypermatrices with right-most sizes equal to 1 are automatically reduced. An hypermatrix with only two dimensions is automatically changed to a regular matrix (type 1).

Examples

```
a(1,1,1,1:2)=[1 2]
a=[1 2;3 4];a(:,:,2)=rand(2,2)
a(1,1,:)

size(a)

a(:,:,1) //dimensionnality reduction
type(a(:,:,1))

[a a]
```

See Also

`hypermat` , `zeros` , `ones` , `grand` , `matrix`

Name

iscell — Check if a variable is a cell array

```
bool = iscell(x)
```

Parameters

x
Scilab variable

bool
A boolean

Description

iscell(x) returns true if x is a cell array and false otherwise.

Examples

```
iscell(1)

iscell(cell())

c = cell(1,2);
c(1).entries="Scilab";
c(2).entries=datetime();
iscell(c)
```

See Also

cell, isstruct

Author

V.C.

Name

iscellstr — Check if a variable is a cell array of strings

```
bool = iscellstr(x)
```

Parameters

x
Scilab variable

bool
A boolean

Description

iscellstr(x) returns true if x is a cell array of strings (or an empty cell array) and false otherwise.

Examples

```
iscellstr(1)

iscellstr(cell())

iscellstr(cell(3))

strcell = cell(3,1);
strcell(1).entries="Scilab";
strcell(2).entries="iscellstr";
strcell(3).entries="help";
iscellstr(strcell)
```

See Also

cell, iscell, isstruct

Author

V.C.

Name

isfield — Checks if the given fieldname exists in the structure

```
bool = isfield(s,fieldname)
```

Parameters

s
A struct array

fieldname
A matrix of strings

bool
A matrix of boolean.

Description

This function returns true if the specified structure "s" includes the field "field", regardless of the corresponding value.

Examples

```
s = struct("field_1",123,"field_2",456,"field_4",789)

// Single Fieldname Syntax
isfield( s , "field_1" )

// Multiple Fieldname Syntax
isfield( s , [ "field_1" "field_2" ; "field_3" "field_4" ] )
```

See Also

struct , getfield , definedfields

Name

isstruct — Check if a variable is a structure array

```
bool = isstruct(x)
```

Parameters

x
Scilab variable

bool
A boolean

Description

isstruct(x) returns true if x is a struct array and false otherwise.

Examples

```
isstruct(1)

isstruct(cell())

isstruct(struct("name","Scilab", "version", getversion()))

info.name="Scilab";
info.function="isstruct";
info.module="help";
isstruct(info)
```

See Also

struct, iscell

Author

V.C.

Name

list — Scilab object and list function definition

```
list(a1, ..., an)
```

Description

Creates a `list` with elements `ai`'s which are arbitrary Scilab objects (`matrix`, `list`, ...). Type of `list` objects is 15. `list()` creates the empty `list` (0 element).

Operations on lists

extraction

`[x, y, z, ...] = L(v)` where `v` is a vector of indices; `[x, y, z] = L(:)` extracts all the elements.

insertion at index `i`

`L(i) = a` (note that it is not an error to use `L(i) = a` with $i > 1 + \text{size}(L)$ but some list entries are then undefined and their extraction gives raise to an error).

append an element in queue

`L($+1) = e`.

append an element in head

`L(0) = e`. (note that after this operation `e` is at index 1, the initial elements being shifted on the right).

deletion

`L(i) = null()` removes the `i`-th element of the list `L`.

concatenation of two lists

`L3 = lstcat(L1, L2)`.

number of elements of a list

you can use either `nb_elm = size(L)` or `nb_elm = length(L)`.

iterations with a list

it is possible to use a list `L` with a for loop: `for e=L, ..., end` is a loop with `length(L)` iterations, the loop variable `e` being equal to `L(i)` at the `i` th iteration.

Remarks

Scilab provides also other kinds of list, the `tlist` type (typed list) and the `mlist` type which are useful to define a new data type with operator overloading facilities (hypermatrices which are multi-dimensional arrays in scilab are in fact *mlist*).

Matlab *struct* are also available.

Examples

```
l = list(1, ["a" "b"])
l(0) = "foo"
l($+1) = "hello"
l(2) = "toto"
l(3) = rand(1,2)
l(3) = null()
lbis = list("gewurtz", "caipirina" , "debug")
```

```
lter = lstcat(l, lbis)  
size(lter) - size(lbis) - size(l)  // must be zero
```

See Also

[null](#) , [lstcat](#) , [tlist](#) , [insertion](#) , [extraction](#) , [size](#) , [length](#)

Name

lsslist — Scilab linear state space function definition

```
lsslist()  
lsslist(a1,...,an)
```

Description

`lsslist(a1,...,an)` is a shortcut to `tlist(['lss','A';'B';'C';'X0','dt'],a1,...,an)`

Creates a `tlist` with `['lss','A';'B';'C';'X0','dt']` as first entry and `ai`'s as next entries if any. No type nor size checking is done on `ai`'s.

See Also

`tlist` , `syslin`

Name

lstcat — list concatenation

```
lc=lstcat(l1,...ln)
```

Parameters

li
list or any other type of variable

lc
a list

Description

lc=lstcat(l1,...ln) catenates components of li lists in a single list. If the li are other type of variables they are simply added to the resulting list.

Examples

```
lstcat(list(1,2,3),33,list('foo',%s))  
lstcat(1,2,3)
```

See Also

list

Name

`mlist` — Scilab object, matrix oriented typed list definition.

```
mlist(typ,a1,...an )
```

Parameters

`typ`

vector of character strings

`ai`

any Scilab object (`matrix`, `list`, `string...`).

Description

`mlist` object are very similar to `tlist` objects. The only difference concerns the extraction and insertion syntax: if `M` is an `mlist`, for any index `i` which is not a field name, `M(i)` is no more the `i`th field of the list.

The semantic of the extraction and insertion syntax should be given by an overloading functions.

The overloading function for extraction syntax `b=a(i1,...,in)` has the following calling sequence: `b=%<type_of_a>_e_(i1,...,in,a)`

and the syntax `[x1,...,xm]=a(i1,...,in)` has the following calling sequence: `[x1,...,xm]=%<type_of_a>_e_(i1,...,in,a)`

The overloading function associated to the insertion syntax `a(i1,...,in)=b` has the following calling sequence: `a=%<type_of_b>_i_<type_of_a>(i1,...,in,b,a)`.

`mlist` fields must then be designed by their names. They can also be handled using the `getfield` and `setfield` functions.

Examples

```
M=mlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
//define display
function %V_p(M),disp(M.name+':'+string(M.value)),endfunction

//define extraction operation
function r=%V_e(varargin)
    M=varargin($);
    r=mlist(['V','name','value'],M.name(varargin(1:$-1)),M.value(varargin(1:$-1)));
endfunction
M(2,:) // the second row of M
M.value

//define insertion operations
function M=%V_i_V(varargin)
    M=varargin($);
    N=varargin($-1);
    M.value(varargin(1:$-2))=N.value;
    M.name(varargin(1:$-2))=N.name;
endfunction
```

```
M(1,1)=M(2,2)

function M=%s_i_V(varargin) //insertion of a regular matrix into a V matrix
    M=varargin($);
    N=varargin($-1);
    M.value(varargin(1:$-2))=N;
    M.name(varargin(1:$-2))=emptystr(N);
endfunction
M(1,1)=44

//tlist case
M=tlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
M(2)
M(2)='a'+string([1 2;3 4])

M('name')
```

See Also

tlist , list , overloading , getfield , setfield

Name

rlist — Scilab rational fraction function definition

```
rlist()  
rlist(a1,...an)
```

Description

`rlist(a1,...an)` is a shortcut to `tlist(['r','num';'den','dt'], a1,...an)`

Creates a `tlist` with `['r','num';'den','dt']` as first entry and `ai`'s as next entries if any.
No type nor size checking is done on `ai`'s.

See Also

`tlist` , `syslin`

Name

setfield — list field insertion

```
setfield(i,x,l)
```

Parameters

- x
matrix of any possible types
- l
list, tlist or mlist variable
- i
field index, see insertion for more details.

Description

This function is an equivalent of `l(i)=x` syntax for field extraction with the only difference that it also applies to `mlist` objects.

Examples

```
l=list(1, 'qwerw', %s)
l(1)='Changed'
l(0)='Added'
l(6)=[ 'one more'; 'added']

a=hypermat([2,2,2],rand(1:2^3)); // hypermatrices are coded using mlists
setfield(3,1:8,a);a // set the field value to 1:8
```

See Also

insertion

Name

struct — create a struct

```
st=struct(field1,value1,field2,value2...)
```

Parameters

field1, field2, ..
strings represents the fields names

value1, value2, ..
all data type (double, char, int, ...), represents the fields values

Description

This function returns a struct with the fields names `fields1`, `field2`, ..., and the fields values corresponding `value1`, `value2`, ...

Examples

```
// create a struct date
date=struct('day',25,'month' , 'DEC', 'year',2006)

//change the month
date.month='AUG';

// change the year
date.year=1973;

//change the day
date.day=19;

// add a new field
date.semaine=32
```

See Also

cell

Name

tlist — Scilab object and typed list definition.

```
tlist(typ,a1,...,an )
```

Parameters

typ

Character string or vector of character strings

ai

Any Scilab object (matrix, list, string...).

Description

Creates a **typed-list** with elements **ai**'s. The **typ** argument specifies the list type. Such **typed-list** allow the user to define new operations working on these object through scilab functions. The only difference between **typed-list** and **list** is the value of the type (16 instead of 15).

typ(1) specifies the list type (character string used to define soft coded operations)

if specified **typ(i)** may give the **i+1**th element formal name

Standard Operations on **list** work similarly for **typed-list**:

extraction : $[x, y, z, \dots] = l(v)$ where **v** is a vector of indices; $[x, y, z] = l(:)$ extracts all the elements.

insertion : $l(i) = a$

deletion : $l(i) = \text{null}()$ removes the **i**-th element of the **tlist l**.

display

Moreover if **typ(2:n+1)** are specified, user may point elements by their names

We give below examples where **tlist** are used.

Linear systems are represented by specific **typed-list** e.g.
a linear system $[A, B, C, D]$ is represented by the **tlist**
`Sys=tlist(['lss'; 'A'; 'B'; 'C'; 'D'; 'X0'; 'dt'], A, B, C, D, x0, 'c')` and this specific list may be created by the function `syslin`.

`Sys(2)`, `Sys('A')` or `Sys.A` is the state-matrix and `Sys('dt')` or `Sys.dt` is the time domain

A rational matrix **H** is represented by the **typed-list**
`H=tlist(['r'; 'num'; 'den'; 'dt'], Num, Den, [])` where **Num** and **Den** are two polynomial matrices and a (e.g. continuous time) linear system with transfer matrix **H** maybe created by `syslin('c', H)`.

`H(2)`, `H('num')` or `H.num` is the transfer matrix numerator

Examples

```
// tlist creation
t = tlist(["listtype", "field1", "field2"], [], []);
t.field1(1) = 10;
```

```
t.field1(2) = 20;
t.field2(1) = "Scilab";
t.field2(2) = "tlist";
t.field2(3) = "example";

// Fields contents display
disp(t.field1)
disp(t.field2)

// Generic tlist display
disp(t)

// Overloading display for this type of tlist
function %listtype_p(mytlist)
    f = fieldnames(mytlist);

    // typeof(mytlist) <=> f(1)
    mprintf("Displaying a tlist of type: %s\n", typeof(mytlist));

    mprintf("\n");

    mprintf("-- Field '%s' --\n", f(1));
    mprintf("Contents: %s\n", sci2exp(mytlist(f(1))));

    mprintf("\n");

    mprintf("-- Field '%s' --\n", f(2));
    mprintf("Contents: %s\n", sci2exp(mytlist(f(2))));
endfunction

// Display using overloading function
disp(t)
```

See Also

null , percent , syslin , list

Part XI. Shell

Name

`clc` — Clear Command Window

```
clc([nblines])
```

Parameters

`nblines`
a double value

Description

`clc()` clears all input and output from the Command Window.

After using `clc()`, you cannot use the scroll bar to see the history of functions, but still can use the up arrow to recall statements from the command history.

`clc(nblines)` clears `nblines` above cursor current line and move cursor up to this line.

Note that `clc([nblines])` cannot be used under Unix/Linux platforms when Scilab used in no window mode.

See Also

`tohome`

Authors

V.C.

Name

lines — rows and columns used for display

```
lines([nl [,nc]])  
ncl=lines()
```

Parameters

nl : an integer, the number of lines for vertical paging control. If 0
no vertical paging control is done.

nc
an integer, the number of column of output. Used for formatting output

ncl
a 1x2 vector [nc,nl]

Description

lines handles Scilab display paging.

lines() returns the vector [# columns, # rows] currently used by Scilab for displaying the results.

lines(nl) sets the number of displayed lines (before user is asked for more) to nl.

lines(0) disables vertical paging

lines(nl, nc) changes also the size of the output to nc columns.

When Scilab is launched without -nw option, the lines parameters are automatically set according to the output window size, these parameters are also automatically modified when the window is resized.

See Also

disp , print

Name

prompt — Get/Set current prompt

```
currentprompt = prompt()  
[currentprompt, pauselevel] = prompt()  
prompt(userprompt)
```

Parameters

currentprompt

String: current prompt returned as a character string.

pauselevel

integer: current pause level.

userprompt

String: prompt to display for next user input. Then current prompt will be used again.

Description

`currentprompt = prompt()` gets the current prompt.

`prompt(userprompt)` sets the prompt.

See Also

pause, input

Authors

A.C.

Name

tohome — Move the cursor to the upper left corner of the Command Window

```
tohome()
```

Description

`tohome()` moves the cursor to the upper-left corner of the Command Window and clears the screen.

You can use the scroll bar to see the history of previous functions.

Note that `tohome()` cannot be used under Windows platforms when Scilab used in no window mode.

See Also

`clc`

Authors

V.C.

Part XII. Console

Name

console — Keyboard Shortcuts in the Console Window

Description

UP or Ctrl+P	recall previous line.
DOWN or Ctrl+N	recall next line.
F1	call help.
F2	clear console.
F12	open console box only on Windows.
Ctrl+space or TAB	completion : scilab displays a list of all names that start with some characters.
Ctrl + A or HOME	move to beginning of current line.
Ctrl + B or LEFT	moves the cursor one character to the left.
Ctrl + C	interrupts Scilab if nothing selected in the console, else text selected is sent to clipboard.
Ctrl + D or DELETE	deletes the current character.
Ctrl + E or END	moves the cursor to the end of command line.
Ctrl + F or RIGHT	moves the cursor one character to the right.
Ctrl + H or BACKSPACE	deletes the previous character.
Ctrl + K	kills command line from cursor to the end.
Ctrl + S	select all.
Ctrl + U	delete the whole command line.
Ctrl + V	do a paste from clipboard.
Ctrl + W	delete the last word of the command line.
Ctrl + X	Interrupt Scilab
Ctrl + LEFT	move left one word.
Ctrl + RIGHT	move right one word.
Shift + HOME	select from cursor to beginning of statement.
Shift + END	select from cursor to end of statement.
Double-click	select current word.

Part XIII. Completion

Name

completion — returns words that start with the text you pass as parameter.

```
r = completion(beginning_of_a_word)
r = completion(beginning_of_a_word,dictionary)
[functions, commands, variables, macros, graphic_properties, files] = completion(beginning_of_a_word)
[functions, commands, variables, macros, graphic_properties] = completion(beginning_of_a_word)
[functions, commands, variables, macros] = completion(beginning_of_a_word)
[functions, commands, variables] = completion(beginning_of_a_word)
[functions, commands] = completion(beginning_of_a_word)
```

Parameters

r
a string matrix

beginning_of_a_word
a string

dictionary
a string ("functions", "commands", "variables", "macros", "graphic_properties", "files")

functions, commands, variables, macros, graphic_properties, files
a string matrix

Description

returns words that start with the text you pass as parameter.

functions: a string matrix of functions name (C gateways). see 'what'.

commands: a string matrix of command words (reserved). see 'what'.

variables: a string matrix of variables names. see 'who'.

macros: a string matrix of macros names. see 'who'.

graphic_properties: a string matrix

files: a string matrix

Examples

```
r = completion('w')
r = completion('w','functions')
r = completion('w','commands')
r = completion('w','variables')
r = completion('w','macros')
r = completion('w','graphic_properties')
r = completion('w','files')

[functions,commands,variables,macros,graphic_properties,files] = completion('w')
[functions,commands,variables,macros,graphic_properties] = completion('w')
[functions,commands,variables,macros] = completion('w')
[functions,commands,variables] = completion('w')
[functions,commands] = completion('w')
```




See Also

`getscilabkeywords`, `who`, `what`, `libraryinfo`, `librarieslist`

Part XIV. History manager

Name

addhistory — add lines to current history.

```
addhistory(string)
addhistory(string_matrix)
```

Parameters

string

a string

string_matrix

a string matrix

Description

add lines to current history.

Examples

```
addhistory('hello')
addhistory(['hello','Scilab'])
```

Authors

A.C

Name

displayhistory — displays current scilab history

```
displayhistory()
```

Description

displays current scilab history.

See Also

gethistory

Authors

A.C

Name

gethistory — returns current scilab history in a string matrix

```
matstr=gethistory()  
line=gethistory(N)
```

Parameters

matstr

a string matrix

N

Nth line in scilab's history

line

a string

Description

returns current scilab history in a string matrix.

See Also

savehistory , loadhistory , resethistory

Authors

A.C

Name

gethistoryfile — get filename used for scilab's history

```
filename = gethistoryfile()
```

Parameters

filename
file name used for history

Description

get filename for scilab's history

Examples

```
gethistoryfile()
```

Authors

A.C

Name

historymanager — enable or disable history manager

```
state1=historymanager(state2)
state1=historymanager()
```

Parameters

state1
returns history manager state 'on' or 'off'

state2
'on' or 'off' set history manager's state

Description

enable or disable history manager.

Examples

```
displayhistory()
backupstate=historymanager()
historymanager('off')
displayhistory()
historymanager('on')
loadhistory()
displayhistory()
historymanager(backupstate)
```

Authors

A.C

Name

historysize — get number of lines in history

```
nb=historysize()
```

Parameters

nb
number of lines in history.

Description

get number of lines in history.

Examples

```
historysize()
```

Authors

A.C

Name

loadhistory — load a history file

```
loadhistory()  
loadhistory(f)
```

Parameters

f
file pathname

Description

load a history file

by default, history filename is SCIHOME+'/.history.scilab'

Examples

```
loadhistory(SCI+'/.session.scilab')
```

See Also

savehistory , resethistory , gethistory

Authors

A.C

Name

removelinehistory — remove the Nth line in history.

```
removelinehistory(N)
```

Parameters

N
a line number

Description

remove the Nth line in history.

Examples

```
displayhistory()  
removelinehistory(historysize()-2)  
displayhistory()
```

Authors

A.C

Name

resethistory — Deletes all entries in the scilab history.

```
resethistory()
```

Description

Deletes all entries in the current scilab history.

See Also

savehistory , loadhistory

Authors

A.C

Name

saveafterncommands — Save the history file after n statements are added to the file.

```
saveafterncommands(n)
v = saveafterncommands()
```

Parameters

n
a integer, n statements

v
current value

Description

Save the history file after n statements are added to the file.

For example, when you select the option and set n to 5, after every 5 statements are added, the history file is automatically saved.

Use this option if you don't want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

saveafterncommands() returns current value.

0 is default value.

Examples

```
saveafterncommands(3)
```

Authors

A.C

Name

saveconsecutivecommands — Save consecutive duplicate commands.

```
saveconsecutivecommands(boolean_in)
boolean_out = saveconsecutivecommands()
```

Parameters

boolean_in
a boolean (%t or %f)

boolean_out
current value

Description

Save consecutive duplicate commands.

saveconsecutivecommands(%t) if you want consecutive executions of the same statement to be saved to the history file.

Examples

```
saveconsecutivecommands( )
saveconsecutivecommands(%t)
1
1
2
saveconsecutivecommands(%f)
1
1
2
```

Authors

A.C

Name

savehistory — save the current history in a file

```
savehistory()  
savehistory(f)
```

Parameters

f
file pathname

Description

save the current history in a file.

by default, history filename is SCIHOME+'/.history.scilab'

Examples

```
savehistory(SCI+'/session.scilab')
```

See Also

loadhistory , resethistory , gethistory

Authors

A.C

Name

sethistoryfile — set filename for scilab history

```
sethistoryfile(filename)  
sethistoryfile()
```

Parameters

filename
filename for history

Description

set filename for scilab history.

sethistoryfile() without parameters will use the default filename (SCIHOME/history.scilab)

Examples

```
gethistoryfile()  
sethistoryfile(gethistoryfile())
```

Authors

A.C

Part XV. GUI

Table of Contents

1. Tree	1188
uiConcatTree	1189
uiCreateNode	1190
uiCreateTree	1191
uiDeleteNode	1192
uiDisplayTree	1194
uiDumpTree	1195
uiEqualsTree	1196
uiFindNode	1197
uiGetChildrenNode	1199
uiGetNodePosition	1200
uiGetParentNode	1201
uiInsertNode	1202

Chapter 1. Tree

Name

uiConcatTree — Concatenation of Trees

```
concatenatedTree = uiConcatTree(tree1, tree2)
```

Input parameters

tree1, tree2
are of type Tree

Output parameters

concatenatedTree
a Tree, which is the concatenation of tree1 and tree2

Description

Concatenation will return a tree which is the concatenation of the first tree with the second one. The concatenation will took place at the parent level of the first tree.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf21 = uiCreateNode('leaf 2.1', 'iconLeaf2.1', 'callbackLeaf2.1')
leaf22 = uiCreateNode('leaf 2.2', 'iconLeaf2.2', 'callbackLeaf2.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

myTree1 = uiCreateTree(node1, leaf11, leaf12)
myTree2 = uiCreateTree(node2, leaf21, leaf22)

concatTree = uiConcatTree(myTree1, myTree2)

uiDisplayTree(concatTree)
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiEqual-
sTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiCreateNode — Creation of node (for Scilab Tree)

```
myNode = uiCreateNode(label[, icon[, callback]])
```

Input parameters

label

a string matrix which gives the label of the nodes.

icon (optional)

a string matrix which gives the icon image of the nodes.

callback (optional)

a string matrix which gives the callback instruction of the nodes.

Output parameters

myNode

a node of type Tree

Description

Creates a node(a node or a leaf) of type Tree.

Examples

```
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')
```

See Also

uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqual-
sTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiCreateTree — Creation of a Tree

```
myTree = uiCreateTree(myParentTree, mySubTree1, mySubTree2, ..., mySubTreeN)
```

Input parameters

myParentTree
a Tree.

mySubTree(s)
one or many trees

Output parameters

myTree
a Tree

Description

Creates a Tree in which myParentTree will have children(mySubTree1, mySubTree2,...,mySubTreeN).

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

uiDisplayTree(treeRoot)
```

See Also

uiCreateNode , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqual-
sTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiDeleteNode — Deletion in a Tree

```
delTree = uiDeleteNode(tree, node)
```

```
delTree = uiDeleteNode(tree, position)
```

Input parameters

tree

Tree where we do the deletion

node

node we want to delete

position

a string, which is the position of the node we want to delete

Output parameters

delTree

a Tree without the deleted node

Description

Deletion of a node (subTree) from a tree. If we have 3 nodes called 'Node1', 'Node2' and 'Node3' each one at position 1.1, 1.2, and 1.3. Deletion of node at position 1.2 ('Node2') will pull up the 'Node 3' to position 1.2.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Deletion of 'node2'
treeDel = uiDeleteNode(treeRoot, node2)
uiDisplayTree(treeDel)

// Deletion of node at position '3.2'
treeDel = uiDeleteNode(treeRoot, '3.2')
uiDisplayTree(treeDel)
```

See Also

`uiCreateNode` , `uiCreateTree` , `uiDisplayTree` , `uiDumpTree` , `uiInsertNode` , `uiConcatTree` , `uiEqualsTree` , `uiFindNode` , `uiGetParentNode` , `uiGetChildrenNode` , `uiGetNodePosition`

Name

uiDisplayTree — Printing a Tree in GUI mode

```
uiDisplayTree(tree)
```

Input parameters

tree
a Tree.

Description

Display a tree into the a graphic window.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1')
leaf12 = uiCreateNode('leaf 1.2')
leaf31 = uiCreateNode('leaf 3.1')
leaf32 = uiCreateNode('leaf 3.2')
node1 = uiCreateNode('Node 1')
node2 = uiCreateNode('Node 2')
node3 = uiCreateNode('Node 3')
root = uiCreateNode('Root')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)

treeRoot = uiCreateTree( root, treeNode1, node2, treeNode3)

uiDisplayTree(treeRoot)
```

See Also

uiCreateNode , uiCreateTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqual-
sTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiDumpTree — Printing a Tree in the console (text mode)

```
uiDumpTree(tree[,b])
```

Input parameters

tree

a Tree.

b(optional)

display features of each node of the tree. By default b is '%F'.

Description

Display a tree into the console(text mode).

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11,leaf12)
treeNode3 = uiCreateTree(node3, leaf31,leaf32)
treeRoot = uiCreateTree(root, node1,node2,node3)

uiDumpTree(treeRoot)
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqual-
sTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiEqualsTree — Comparing two trees

```
isEqual = uiEqualsTree(tree1, tree2)
```

Input parameters

tree1, tree2
are of type Tree

Output parameters

isEqual
a Boolean, which indicate if those trees are equal or not

Description

Compare two trees structures.

Examples

```
// Creation of trees
root = uiCreateNode('Root', 'path\rootImage.jpg', 'rootCallback')
node1 = uiCreateNode('Node 1', 'default', 'node1Callback')
node2 = uiCreateNode('Node 2', 'default', 'node2Callback')
myTree1 = uiCreateTree(root, node1, node2)
myTree2 = uiCreateTree(root, node1, node2)

// Compare myTree1 with myTree2
isEqual = uiEqualsTree(myTree1, myTree2)

// will return 'isEqual = T'
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiFindNode , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiFindNode — Find node in Tree

```
nodeList = uiFindNode(tree, node)
```

```
nodeList = uiFindNode(tree, position)
```

```
nodeList = uiFindNode(tree, property, value)
```

Input parameters

tree

Tree in which we find the node

node

the node we find

position

a string, which is the position of the node we find in the tree

property

a string, which finds node(s) by properties (label, icon or callback)

value

a string, which is the value of the property

Output parameters

nodeList

a list of matching node(s)

Description

Finds node(s) in a tree.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Creation of a node
myNode = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
```

```
// Find if treeRoot contains myNode
result = uiFindNode(treeRoot, myNode)
//will return 'result = list(node1)'

// Find node at position '1.1'
result = uiFindNode(treeRoot, '3.1')
//will return 'result = list(leaf31)'

// Find node where 'text' equals 'Node 2'
result = uiFindNode(treeRoot, 'label', 'Node 2')
//will return 'result = list(node2)'
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqualsTree , uiGetParentNode , uiGetChildrenNode , uiGetNodePosition

Name

uiGetChildrenNode — Get Children of a node

```
children = uiGetChildrenNode(tree, node)
```

```
children = uiGetChildrenNode(tree, position)
```

Input parameters

tree

Tree in which we look for children node(s)

node

the node we look for children

position

a string, which is the position of the node we look for children

Output parameters

children

a list of children nodes

Description

Finds the children of a specific node.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Search children node(s) of node 'node1'
children = uiGetChildrenNode(treeRoot, node1)
// will return 'children = list(leaf11, leaf12)'
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqualsTree , uiFindNode , uiGetParentNode , uiGetNodePosition

Name

uiGetPosition — Get the position(s) of a node

```
position = uiGetPosition(tree, node)
```

Input parameters

tree

Tree in which we look for position(s) of a node

node

the node we look for position(s)

Output parameters

position

a matrix of string which contains node position(s).

Description

Get the position of a given node.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Get the position of node 'node1'
position = uiGetPosition(treeRoot, node1)
// will return 'position = ['1']'

// Get the position of leaf 'leaf31'
position = uiGetPosition(treeRoot, leaf31)
// will return 'position = ['3.1']'
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqualsTree , uiFindNode , uiGetParentNode , uiGetChildrenNode

Name

uiGetParentNode — Get Parent of a node

```
parent = uiGetParentNode(tree, node)
```

```
parent = uiGetParentNode(tree, position)
```

Input parameters

tree

Tree in which we look for parent of a node

node

the node we look for parent

position

a string, which is the position of the node we look for parent

Output parameters

parent

the parent node

Description

Finds the parent of a specific node.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Search parent node of 'node1'
parentNode = uiGetParentNode(treeRoot, node1)
// will return 'parentNode = root'

// Search parent node of 'leaf31'
parentNode = uiGetParentNode(treeRoot, leaf31)
// will return 'parentNode = node3'
```

See Also

uiCreateNode , uiCreateTree , uiDisplayTree , uiDumpTree , uiInsertNode , uiDeleteNode , uiConcatTree , uiEqualsTree , uiFindNode , uiGetChildrenNode , uiGetNodePosition

Name

uiInsertNode — Insertion in a Tree

```
insertTree = uiInsertNode(tree, position, node)
```

```
insertTree = uiInsertNode(tree, parentNode, node)
```

Input parameters

tree

Tree were we do the insertion

position

a string, which is the position where we want to insert the node

parentNode

which is the parent node into we want to insert the node

node

node we want to insert

Output parameters

insertTree

a Tree with the node inserted

Description

Insertion of a node (subTree) into a tree. If we have 2 nodes called 'Node1' and 'Node2' each one at position 1.1 and 1.2. Insertion of a new node 'Node3' at position '1.2', will move the 'Node2' to position 1.3.

Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree(root, treeNode1, node2, treeNode3)

// Creation of a new nodes to insert
leaf13 = uiCreateNode('leaf 1.3', 'iconLeaf1.3', 'callbackLeaf1.3')
testNode = uiCreateNode('test', 'icon_test', 'callback_test')

// Insertion of 'leaf13' in 'node2'
treeInsert = uiInsertNode(treeRoot, node2, leaf13)
uiDisplayTree(treeInsert)
```



```
// Insertion of 'testNode' at position '1.1'  
treeInsert = uiInsertNode(treeRoot, '1.1', testNode)  
uiDisplayTree(treeInsert)
```

See Also

[uiCreateNode](#) , [uiCreateTree](#) , [uiDisplayTree](#) , [uiDumpTree](#) , [uiDeleteNode](#) , [uiConcatTree](#) , [uiEqualsTree](#) , [uiFindNode](#) , [uiGetParentNode](#) , [uiGetChildrenNode](#) , [uiGetNodePosition](#)

Name

about — show "about scilab" dialog box

```
about()
```

Description

show "about scilab" dialog box.

Examples

```
about()
```

Authors

Allan CORNET

Name

addmenu — interactive button or menu definition

```
addmenu(button [,submenus] [,action])  
addmenu(gwin,button [,submenus] [,action])
```

Parameters

button

a character string. The button name. An & can be placed before the character in the name to be used for keyboard shortcut; this character will be underlined on the GUI. Under MacOSX, a sub-menu with the same name is automatically added (no button can be added to the menu bar).

submenus

a vector of character string. The sub_menus items names

action

a list with 2 elements action=list(flag,proc_name)

flag

an integer (default value is 0)

flag==0

the action is defined by a scilab instruction

flag==1

the action is defined by a C or Fortran procedure

flag==2

the action is defined by a scilab function

proc_name

a character string which gives the name of scilab variable containing the instruction or the name of procedure to call.

gwin

integer. The number of graphic window where the button is required to be installed

Description

The function allows the user to add new buttons or menus in the main window or graphics windows command panels.

If

action argument is not given the action associated with a button must be defined by a scilab instruction given by the character string variable which name is

+ **button** for a main window command

+ **button_gwin** for a graphic window command

If

action argument is set to 0 **proc_name** should be the name of a Scilab string vector. Actions associated with the kth sub_menu must be defined by scilab instructions stored in the kth element of the character string variable.

If

action argument is set to 1 **proc_name** designates a C or Fortran procedure, this procedure may be interfaced in Fortran subroutine default/fbutn.f or dynamically linked with scilab using the **link** function. The C calling sequence is: (char* button_name, int* gwin,int *k)

If

action argument is set to 2 `proc_name` designates a Scilab function. This function calling sequence should be:

+ `proc_name(k)` for a main window command

+ `proc_name(k, gwin)` for a graphic window command or a main window command

Examples

```
if (getscilabmode() == "STD") then
    addmenu('foo');
    foo = 'disp(''hello'')';

    addmenu('Hello', ['Franck'; 'Peter'])
    Hello = ['disp(''hello Franck'')'; 'disp(''hello Peter'')'];

    addmenu('Bye', list(0, 'French_Bye'));
    French_Bye = 'disp(''Au revoir'')';
else
    mprintf('This example requires to use scilab with GUI mode.\n');
end

addmenu(0, 'Hello', ['Franck'; 'Peter']);
Hello_0 = ['disp(''hello Franck'')'; 'disp(''hello Peter'')'];

//C defined Callback
// creating Callback code
code=[ '#include ""machine.h""
        '#include ""sciprint.h""
        'void foo(char *name, int *win, int *entry)'
        '{'
        '    if (*win==-1) '
        '        sciprint("menu %s(%i) in Scilab window selected.\n", name, *entry)'
        '    else'
        '        sciprint("menu %s(%i) in window %i selected.\n", name, *entry+1,'
        '    }'];
//creating foo.c file
current_dir = pwd();
chdir(TMPDIR);
mputl(code, TMPDIR+'/foo.c');
//creating Makefile
ilib_for_link('foo', 'foo.c', [], 'c');
exec('loader.sce');
chdir(current_dir);
//add menu
addmenu(0, 'foo', ['a', 'b', 'c'], list(1, 'foo'));
```

See Also

[setmenu](#), [unsetmenu](#), [delmenu](#)

Name

clipboard — Copy and paste strings to and from the system clipboard.

```
clipboard("copy",data)
str=clipboard("paste")
clipboard("do","paste")
clipboard("do","copy")
clipboard("do","empty")
clipboard(winnum,"EMF")
clipboard(winnum,"DIB")
```

Parameters

data

Scilab variable or data to set as the clipboard contents.

str

The clipboard contents returned as a Scilab character string.

winnum

Number of the graphic window to set as the clipboard contents.

Description

`clipboard("copy",data)` sets the clipboard contents to data. If data is not a character array, the clipboard uses `sci2exp` to convert it to a string.

`str = clipboard("paste")` returns the current contents of the clipboard as a string or as an empty string (""), if the current clipboard contents cannot be converted to a string.

`clipboard("do","paste"),` `clipboard("do","copy"),`
`clipboard("do","empty")` performs a paste, copy or empty clipboard.

`clipboard(winnum,"EMF")` copy a graphic window identified by his window's number in the clipboard to EMF format.

`clipboard(winnum,"DIB")` copy a graphic window identified by his window's number in the clipboard to DIB format.

Note that `clipboard` function works only when Scilab used in window mode.

Authors

A.C.

Name

close — close a figure

Parameters

h
integer Handle of the window to close.

Description

This routine close a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is closed. Otherwise, the current (active) figure is closed.

Examples

```
h=figure();  
// creates figure number 1.  
uicontrol( h, 'style','text', ...  
           'string','scilab is great', ...  
           'position',[50 70 100 100], ...  
           'fontsize',15);  
// put a clever text in figure 1  
figure();  
// create figure 2  
uicontrol( 'style','text', ...  
           'string','Really great', 'position',[50 70 100 100], 'fontsize',15);  
// put a text in figure 2  
close();  
// close the current graphic window (ie fig. 2)  
close(h);  
// close figure 1
```

See Also

figure , gcf

Authors

Bertrand Guiheneuf

Name

delmenu — interactive button or menu deletion

```
delmenu(button)
delmenu(gwin,button)
```

Parameters

button

a character string. The button name. On Windows operating systems (not X_window), an & should be placed before the character in the name used for keyboard shortcut; this character is underlined on the GUI.

gwin

integer. The number of graphic window where the button is required to be installed

Description

The function allows the user to delete buttons or menus create by `addmenu` in the main or graphics windows command panels. Predefined buttons on Scilab graphic windows can also be deleted.

If possible, it is better to delete first the latest created button for a given window to avoid gaps in command panels.

Examples

```
addmenu( 'foo' )
delmenu( 'foo' )
```

See Also

setmenu , unsetmenu , addmenu

Name

`exportUI` — Call the file export graphical interface

```
exportUI(figId)
exportUI(fig)
```

Parameters

`figId`
integer, Id of the figure to export.

`fig`
Figure handle, handle of the figure to export.

Description

`exportUI` routine call the graphical interface dedicated in exporting a graphic window into an image file.

See Also

`xs2jpg` , `xs2eps` , `xs2png` , `xs2svg` , `xs2pdf`

Authors

Jean-Baptiste Silvy

Name

figure — create a figure

```
f = figure(num);  
f = figure("PropertyName1", Propertyvalue1, ..., ..., "PropertyNameN", Prop
```

Description

This routine creates a figure. If an ID is given, the figure corresponding to this ID is created. Otherwise, the window is created with the first free ID, that is the lowest integer not already used by a window.

Parameters

num

ID of the window to create. If not specified, the first free ID is used.

PropertyName{1, ..., N}

character string name of a property to set. One of the property names listed below.

PropertyValue{1, ..., N}

scilab object value to give to the corresponding property.

f

handle of the newly created window.

Properties

BackgroundColor

[1,3] real vector or string Background color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Figure_name

character string, allows to set the title of the figure.

ForegroundColor

[1,3] real vector or string Foreground color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Position

allows to control the geometrical aspect of the figure. It is a [1,4] real vector [x y width height] where the letters stand for the x location of the top left corner, the y location of the top left corner, the width and the height of the virtual graphics window (the part of the figure which contains uicontrols and graphics). See the **axes_size** property description in figure properties help page. One can also set this property by giving a string where the fields are separated by a "|", ie "x|y|width|height".

Tag

string this property is generally used to identify the figure. It allows to give it a "name". Mainly used in conjontion with findobj().

Userdata

this can be used to associate some Scilab objects to a fugure.

Examples

```
// Create figure having figure_id==3
h=figure(3);
// Add a text uicontrol in figure 3
uicontrol(h, "style", "text", ...
          "string", "This is a figure", ...
          "position", [50 70 100 100], ...
          "fontsize",15);

// Create figure having figure_id==1
figure();
// Add a text uicontrol in figure 1
uicontrol("style", "text", ...
          "string", "Another figure", ...
          "position", [50 70 100 100], ...
          "fontsize", 15);

// Close current figure (ie figure 1)
close();
// close figure 3
close(h);
```

See Also

`close` , `gcf`

Authors

Bertrand Guiheneuf

V.C.

Name

findobj — find an object with specified property

```
h = findobj(propertyName, propertyValue)
```

Parameters

propertyName

string character Name of the property to test (case unsensitive).

propertyValue

string character specify the value the tested propoerty should be equal to (case sensitive).

h

handle of the found object.

Description

This routine is currently used to find objects knowing their 'tag' property. It returns handle of the first found object which property *propertyName* is equal to *propertyValue*. If such an object does not exist, the function returns an empty matrix.

Examples

```
// Create a figure
h=figure();
// Put a text in the figure
uicontrol(h, "style","text", ...
          "string","This is a figure", ...
          "position",[50 70 100 100], ...
          "fontsize",15, ...
          "tag","Alabel");
// Find the object which "tag" value is "Alabel"
lab=findobj("tag","Alabel");
disp("The text of the label is ""+lab.string+""");
// Close the figure
close();
```

See Also

uicontrol , uimenu , set , get

Authors

Bertrand Guiheneuf

V.C.

Name

`gcbo` — Handle of the object whose callback is executing.

```
gcbo
```

Description

`gcbo` is a Scilab variable automatically created each time a callback is executed. This variable is initialised using `getcallbackobject`.

`gcbo` does not exist in Scilab environment if no callback is currently executed.

You can use `gcbo` in callback functions particularly if you write a single callback function for multiple objects, it helps you to know which object received a user action.

See Also

`getcallbackobject`

Authors

Vincent COUVERT

Name

`getcallbackobject` — Return the handle of the object whose callback is executing.

```
h = getcallbackobject()
```

Parameters

`h`
Handle: the handle of the object whose callback is executing.

Description

`getcallbackobject` is used to automatically create Scilab variable called `gcbo` each time a callback is executed.

`getcallbackobject` returns `[]` if no callback is currently executed.

See Also

`gcbo`

Authors

Vincent COUVERT

Name

getinstalledlookandfeels — returns a string matrix with all Look and Feels.

```
lnf=getinstalledlookandfeels()
```

Parameters

lnf
a string matrix.

Description

returns a string matrix with all Look and Feels that you can use.

Examples

```
getinstalledlookandfeels()
```

See Also

setlookandfeel , getlookandfeel

Authors

Allan CORNET

Name

getlookandfeel — gets the current default look and feel.

```
lnf=getlookandfeel()
```

Parameters

lnf
a string with current look and feel.

bok
a boolean.

Description

Gets the current default look and feel.

Examples

```
currentlnf = getlookandfeel();

// Look and feel CDE/Motif
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")

sleep(3000);

// Look and feel métal
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")

sleep(3000);

setlookandfeel(currentlnf)
```

See Also

getinstalledlookandfeels , setlookandfeel

Authors

Allan CORNET

Name

getvalue — xwindow dialog for data acquisition

```
[ok,x1,...,x14]=getvalue(desc,labels,typ,ini)
```

Parameters

desc

column vector of strings, dialog general comment

labels

n column vector of strings, `labels(i)` is the label of the ith required value

typ

`list(typ_1,dim_1,...,typ_n,dim_n)`

typ_i

defines the type of the ith value, may have the following values:

"mat"

for constant matrix

"col"

for constant column vector

"row"

for constant row vector

"vec"

for constant vector

"str"

for string

"lis"

for list

dim_i

defines the size of the ith value it must be a integer or a 2-vector of integer, -1 stands for undefined dimension

ini

n column vector of strings, `ini(i)` gives the suggested response for the ith required value

ok

boolean ,%t if ok button pressed, %f if cancel button pressed

xi

contains the ith value if `ok=%t`. If left hand side has one more `xi` than required values the last `xi` contains the vector of answered strings.

Description

This function encapsulate `x_mdialog` function with error checking, evaluation of numerical response, ...

Remarks

All valid expressions can be used as answers; for matrices and vectors `getvalues` automatically adds `[]` around the given answer before numeric evaluation.

Examples

```
labels=["magnitude";"frequency";"phase    "];  
[ok,mag,freq,ph]=getvalue("define sine signal",labels,...  
    list("vec",1,"vec",1,"vec",1),["0.85";"10^2";"%pi/3"])
```

See Also

[x_mdialog](#) , [x_matrix](#) , [x_dialog](#)

Authors

S. Steer ; ;

Name

messagebox — Open a message box.

```
[btn] = messagebox(msg)
[btn] = messagebox(msg, msgboxtitle)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons, ismodal)
```

Parameters

msg

Matrix of strings: the message box displays each entry of this matrix (one entry per line).

msgboxtitle

String: the title of the message box (default value is "Scilab Message").

msgboxicon

String: the name of the icon to be displayed in the message box, its possible values are:

- "error"
- "hourglass"
- "info"
- "passwd"
- "question"
- "warning"
- "scilab": default icon

buttons

1xn vector of strings: the names of the buttons to be displayed in the message box. By default, only one button is displayed with label "OK".

modal

String: "modal" to create a modal dialog, any other string to create a non-modal dialog. Please note that "modal" can replace any of the other input arguments except msg (See examples).

btn

Scalar: number of the button that the user pressed (1 is the leftmost button) for a modal dialog, 0 else.

Description

Creates a dialog window to display a message waiting or not for a user action.

Examples

```
// Simple example
messagebox("Single line message")
```

```
// Multi line message with title
messagebox(["Multi-line" "message"], "User defined title")

// Icon specified by the user
messagebox("An error message", "Error", "error")

// Buttons labels + "modal" replaces title
messagebox("Have you seen this beautiful message", "modal", "info", ["Yes" "No"]

// "modal" given as fifth input argument
messagebox("An error message", "Error", "error", ["Continue" "Stop"], "modal")
```

Authors

Vincent COUVERT

Name

printfigure — Opens a printing dialog and prints a figure.

```
printfigure(figid)
status = printfigure(figid)
```

Parameters

figid

Real: the id of the figure to be printed.

status

Boolean: %T if the printing succeeds, %F otherwise.

Description

This function opens a dialog to select a printer, printing options... and then prints the figure.

Examples

```
plot2d();
printfigure(get(gcf(), "figure_id"));
```

See Also

toprint , printsetupbox

Authors

V.C.

Name

printsetupbox — Display print dialog box.

```
printsetupbox()  
status=printsetupbox()
```

Parameters

status

Boolean: *%T* if the user clicked on the OK button, *%F* otherwise.

Description

Displays the built-in printing dialogbox and configure the printer.

See Also

toprint , printfigure

Authors

A.C

Name

progressionbar — Draw a progression bar

```
winId=progressionbar(mes)
progressionbar(winId[,mes])
```

Parameters

mes

string, message to display.

winId

integer greater than 0, window identifier.

Description

progressionbar(mes) create a new progression bar, return window identifier.

progressionbar(winId[,mes]) update the progression bar identified as winId.

Examples

```
winId=progressionbar('Do something');
realtimeinit(0.3);
for j=0:0.1:1,
    realtime(3*j);
    progressionbar(winId);
end
winclose(winId);
```

Authors

Jaime Urzua

Name

`root_properties` — description of the root object properties.

Description

The root object is a virtual object used to get the computer screen properties. Use `get` function with `0` as first argument to access its properties.

Root properties

`screen_size_px`:
The screen size in pixels.

`screen_size_pt`:
The screen size in points.

`screen_size_mm`:
The screen size in millimeters.

`screen_size_cm`:
The screen size in centimeters.

`screen_size_in`:
The screen size in inches.

`screen_size_norm`:
The normalized screen size.

`screen_depth`:
The number of bits used to encode colors.

Examples

```
get(0, "screen_size_px")
get(0, "screen_depth")
```

See Also

`get`

Author

Vincent COUVERT

Name

setlookandfeel — sets the current default look and feel.

```
bok=setlookandfeel()  
bok=setlookandfeel(lnf)
```

Parameters

lnf
a string with a look and feel.

bok
a boolean.

Description

Sets the current default Look and Feel.

setlookandfeel() without parameter set system default look and feel.

Examples

```
currentlnf = getlookandfeel();  
  
// Look and feel Windows Classic  
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel")  
  
// Look and feel Windows  
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel")  
  
sleep(3000);  
  
// Look and feel CDE/Motif  
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")  
  
sleep(3000);  
  
// Look and feel GTK+  
setlookandfeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel")  
  
sleep(3000);  
  
// Look and feel métal  
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")  
  
sleep(3000);  
  
// Look and feel Macintosh  
setlookandfeel("it.unitn.ing.swing.plaf.macos.MacOSLookAndFeel")  
  
// System default look and feel  
  
setlookandfeel()  
  
sleep(3000);
```



```
// restore previous look and feel  
setlookandfeel(currentInf)
```

See Also

[getinstalledlookandfeels](#) , [getlookandfeel](#)

Authors

Allan CORNET

Name

setmenu — interactive button or menu activation

```
setmenu(button [, nsub])  
setmenu(gwin, button [, nsub])
```

Parameters

button

a character string. The button name

gwin

integer. The number of graphic window where the button is installed

nsub

integer. The number of submenu to de-activate (if any). If button has no sub-menu, nsub is ignored

Description

The function allows the user to make active buttons or menus created by `addmenu` in the main or graphics windows command panels.

Examples

```
addmenu('foo')    // New button made in main scilab window  
unsetmenu('foo')  // button foo cannot be activated (grey string)  
setmenu('foo')    // button foo can be activated (black string)
```

See Also

`delmenu` , `unsetmenu` , `addmenu`

Name

toolbar — show or hide a toolbar

```
state1=toolbar(winnum, state2)
state1=toolbar(winnum)
```

Parameters

state1

returns toolbar's state 'on' or 'off'

winum

window's number (-1: Scilab console window)

state2

'on' or 'off' set toolbar's state

Description

show or hide a toolbar.

Examples

```
toolbar(-1, 'off')
state=toolbar(-1, 'on')

plot3d();
h=gcf();
toolbar(h.figure_id, 'off')
```

Authors

Allan CORNET
Vincent COUVERT

Name

toprint — Send text or figure to the printer.

```
toprint(filename)
toprint(linestoprint,pageheader)
toprint(figid)
toprint(figid,output)
status = toprint(filename)
status = toprint(linestoprint,pageheader)
status = toprint(figid)
status = toprint(figid,output)
```

Parameters

filename

String: path of the text file to be printed.

linestoprint

String matrix: text to be printed, each entry is a line in printed pages.

pageheader

String: header of printed pages.

figid

Real: the id of the figure to be printed.

output

String: printing output type, must be *"pos"* for PostScript or *"gdi"* for Bitmap format (*"gdi"* by default).

status

Boolean: *%T* if the printing succeeds, *%F* otherwise.

Description

Prints a text file, Scilab character strings or figure.

Examples

```
toprint(SCI+"/etc/scilab.start");
toprint(['Test','toprint primitive'],'Scilab page header');
scf(4);
plot();
toprint(4);
toprint(4,"pos");
```

See Also

printfigure , printsetupbox

Authors

A.C.

V.C.

Name

uicontrol — create a Graphic User Interface object

```
h = uicontrol(PropertyName,PropertyValue,...)
h = uicontrol(parent,PropertyName,PropertyValue,...)
h = uicontrol(uich)
```

Description

This routine creates an object in a figure.

If the handle of the figure is given (as the first parameter), the uicontrol is created in this figure. If no handle is given, the uicontrol is created in the current figure (which may be obtained with a call to `gcf()`). If there is no current figure, then one is created before the creation of the uicontrol.

Then when the control is created, the properties given as parameters are set with the corresponding values. It is equivalent to create the uicontrol, and then set its properties with the `set()` command. Nevertheless, it is generally more efficient to set the properties in the call to `uicontrol()`. This is particularly true concerning the "Style" property. Indeed, the default value for this property is "Pushbutton". So if you do not set it at creation time, a button will be created, and will be transformed to another uicontrol when you call the `set(h, "Style", ...)` instruction. Scilab and all the graphic objects communicate through the property mechanism. Thus, to create adapted uicontrol, one has to know the use of the property fields.

`h = uicontrol(PropertyName, PropertyValue, ...)` creates an uicontrol and assigns the specified properties and values to it. It assigns the default values to any properties you do not specify. The default uicontrol style is a "Pushbutton". The default parent is the current figure. See the Properties section for information about these and other properties.

`h = uicontrol(parent, PropertyName, PropertyValue, ...)` creates a uicontrol in the object specified by the handle, parent. If you also specify a different value for the Parent property, the value of the Parent property takes precedence. parent is the handle of a figure.

`h = uicontrol(uich)` gives focus to the uicontrol specified by uich.

Properties

BackgroundColor

[1,3] real vector or string

Background color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Callback

String

Instruction evaluated by the Scilab interpreter when an uicontrol is activated. (for example when you click on a button).

Enable

{on} | off

Enable or disable the uicontrol. If this property is set to "on" (default), the uicontrol is operational, but if this property is set to "off", the uicontrol will not respond to the mouse actions and will be grayed out.

FontAngle

{normal} | italic | oblique

For a control containing some text, this property sets the slant of the font.

FontSize

Scalar

For a control containing some text, this property sets the size of the font in FontUnits.

FontUnits

{points} | pixels | normalized

For a control containing some text, this property sets the units with which the FontSize is specified.

FontWeight

light | {normal} | demi | bold

For a control containing some text, this property sets the weight of the used font.

FontName

String

Used to choose the name of the font selected to display the text of the control.

ForegroundColor

[1,3] real vector or string

Foreground color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

HorizontalAlignment

left | {center} | right

Set text horizontal alignment in the uicontrol. This property has only effect with Text, Edit and Check Boxes.

ListboxTop

Scalar

For a ListBox, this property tells which item of the list appears on the first line of the visible area of the list.

Max

Scalar

Specifies the largest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Max is the value the "Value" property take when control is checked.
- Sliders: Maximum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Min

Scalar

Specifies the lowest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Min is the value the "Value" property take when control is unchecked.
- Sliders: Minimum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Parent

Handle

Handle of the uicontrol parent. Changing this property allows to move a control from a figure to another.

Path

This property is no more supported.

Position

[1,4] real vector or string.

This property is used to set or get the geometrical configuration of a control. It is a vector [x y w h] where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol or a character string where each value is separated by a "|", ie "x|y|w|h". The units are determined by the "Units" property.

The width and height values determine the orientation of sliders. If width is greater than height, then the slider is oriented horizontally, otherwise the slider is oriented vertically.

Relief

flat | groove | raised | ridge | solid | sunken

Appearance of the border of the uicontrol:

- PushButtons: the default value for "Relief" property is "raised".
- Edits: the default value for "Relief" property is "sunken".
- Other styles: the default value for "Relief" property is "flat".

SliderStep

[1,2] real vector

[small big], the small step represents the movement achieved when clicking on the slider trough or tapping on the keyboard arrows (when the slider has focus); the big step is the amount moved when using Ctrl-keyboard-arrows. If the big step is omitted, it is defaulted to 1/10 of the scale.

String

String.

This property represents the text appearing in a uicontrol (Except for Frame and Slider styles). For ListBoxes and PopupMenus, the value can be a vector of string or a string where the items are separated by a "|". For Text uicontrols, this string can contain HTML code to format the text.

Style

{pushbutton} | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu

Style of the uicontrol. Here is a short description of each one:

- Pushbutton: a rectangular button generally used to run a callback.
- Radiobutton: a button with to states. RadioButtons are intended to be mutually exclusive (Your code must implement mutually exclusive behavior).
- Checkbox: a button with to states (Used for multiple independent choices).

- Edit: an editable string zone.
- Text: a text control (generally static).
- Slider: a scale control, that is a scrollbar use to set values between in range with the mouse.
- Frame: a control representing a zone used to group related controls.
- Listbox: a control representing a list of items that can be scrolled. The items can be selected with the mouse.
- Popuptmenu: a button which make a menu appear when clicked.

Tag

String

This property is generally used to identify the control. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Units

{points} | pixels | normalized

Set the units used to specify the "Position" property.

Userdata

Scilab data

This can be used to associate some Scilab objects (string,string matrix, matrix mxn) to an uicontrol.

Value

Scalar or vector

Value of the uicontrol. The exact meaning depends on the style of the uicontrol:

- CheckBoxes, Radio buttons: value is set to Max (see above) when on and Min when off.
- ListBoxes, PopupMenus: value is a vector of indexes corresponding to the indexes of the selected entries in the list. 1 is the first item of the list.
- Sliders: value indicated by the slider bar.

Verticalalignment

top | {middle} | bottom

Set text vertical alignment in the uicontrol. This property has only effect with Text and CheckBoxes styles.

Visible

{on} | off

Set the visibility of the uicontrol. If this property is set to "on" (default), the uicontrol is visible, but if this property is set to "off", the uicontrol will not appear in its parent figure.

Examples

```
f=figure();
// create a figure
h=uicontrol(f,'style','listbox', ...
'position', [10 10 150 160]);
```

```
// create a listbox
set(h, 'string', "item 1|item 2|item3");
// fill the list
set(h, 'value', [1 3]);
// select item 1 and 3 in the list
close(f);
// close the figure
```

See Also

[figure](#), [set](#), [get](#), [uimenu](#)

Authors

Bertrand Guiheneuf

Vincent Couvert

Name

uigetcolor — Opens a dialog for selecting a color.

```
uigetcolor()
RGB = uigetcolor([title])
RGB = uigetcolor([title,] defaultRGB)
RGB = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
[R, G, B] = uigetcolor([title])
[R, G, B] = uigetcolor([title,] defaultRGB)
[R, G, B] = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
```

Parameters

title

String: Optional argument, the title to display in the dialog. Default value is "Color Chooser".

defaultRGB

1x3 vector: the default values for Red, Green and Blue values given as a vector [red, green, blue].

defaultRed

Scalar: the default value for red.

defaultGreen

Scalar: the default value for green.

defaultBlue

Scalar: the default value for blue.

RGB

1x3 vector: the values for Red, Green and Blue values given as a vector [red, green, blue] or [] if the user cancels.

R

Scalar: the value for red or [] if the user cancels.

G

Scalar: the value for green or [] if the user cancels.

B

Scalar: the value for blue or [] if the user cancels.

Description

Creates a dialog window for selecting a color. All (default and returned) values must be in the interval [0 255].

Examples

```
uigetcolor()
[R, G, B] = uigetcolor([255 128 0])
RBG = uigetcolor(0, 128, 255)
RBG = uigetcolor("My color chooser", 0, 128, 255)
```

See Also

[getcolor](#)

Authors

Vincent COUVERT

Name

uigetdir — dialog for selecting a directory

```
directory = uigetdir()  
directory = uigetdir(start_path [,title])
```

Parameters

start_path

a character string which gives the initial directory used for search. By default uigetdir uses current working directory.

title

the title for the uigetdir window.

directory

is the user selected directory if user answers "Ok" or the " " string if user cancels.

Description

Creates a dialog window for selecting a directory

Examples

```
uigetdir()  
uigetdir("SCI/modules/")  
uigetdir("SCI/modules/", "Choose a directory")
```

See Also

uigetfile , uiputfile

Name

uigetfile — dialog window to get a file(s) name(s), path and filter index

```
[FileName[,PathName[,FilterIndex]]]=uigetfile([file_mask[,dir[,boxTitle[,multiple]]]]  
PathFileName=uigetfile([file_mask[,dir[,boxTitle[,multiple]]]])
```

Input parameters

file_mask

a string matrix which gives the file masks to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

we can also add descriptions for masks, for example ['*.x*', 'X files'; '*.bin', 'BIN files'].

dir

a character string which gives the initial directory used for file search. By default uigetfile uses the previously selected directory.

boxTitle

a character string which gives the title of the uigetfile window. By default uigetfile's title is 'uiget-file'.

multipleSelection

a boolean which allows to load only one file if it is at '%f' (false) or multiple files if it is at '%t' (true). By default uigetfile's multiple file selection is not enable.

Output parameters

FileName

matrix of string which give the user selected file(s) (path + file(s) name(s)) if user answers "Ok" or the " " string if user answers "Cancel".

PathName

is the user selected file(s) path if user answers "Ok" or the " " string if user answers "Cancel".

FilterIndex

is the user selected filter index on the list box if user answers "Ok" or '0' string if user answers "Cancel"

Description

Creates a dialog window for file(s) selection.

Comments

On Windows, java component used by uigetfile browse also .zip archive then it is very slow with big .zip files.

To disable, this feature:

```
if MSDOS then unix("REGSVR32 /u %WINDIR%\System32\zipfldr.dll");end
```

To re-enable,

```
if MSDOS then unix("REGSVR32 %WINDIR%\System32\zipfldr.dll");end
```

Examples

```
uigetfile(['*.bin'; '*.sce'; '*.cos*'])
uigetfile(['*.sci'; '*.bin'], 'SCI/modules/gui/macros/')
uigetfile(['*.sc*'; '*.bin'], 'SCI/modules/gui/macros/')
uigetfile(['*.x*', 'X files'; '*.bin', 'BIN files'], 'SCI/modules/gui/macros/')
uigetfile(['*.sce'; '*.bin'], 'SCI/modules/gui/macros/', 'Choose a file name', %t
uigetfile(['*.sce'; '*.bin'], 'SCI/modules/gui/macros/', 'Choose a file name', %f
```

See Also

uiputfile, uigetdir, x_dialog, file, read, write, exec

Name

uigetfont — Opens a dialog for selecting a font.

```
uigetfont()  
  
[fontname [,fontsize [,bold [,italic]]]] = uigetfont([defaultfontname [,default  
[fontname ,fontsize ,bold ,italic] = uigetfont(defaultfontname ,defaultfontsize
```

Parameters

defaultfontname

String: the default font name to select in the dialog.

defaultfontsize

Scalar: the default font size to select in the dialog.

defaultbold

Boolean: the default bold attribute in the dialog (%T for bold font, %F otherwise).

defaultitalic

Boolean: the default italic attribute in the dialog (%T for bold font, %F otherwise).

fontname

The selected font name ("" if the user cancels).

fontsize

The selected font size ([] if the user cancels).

bold

%T if bold attribute has been selected, %F otherwise ([] if the user cancels).

italic

%T if italic attribute has been selected, %F otherwise ([] if the user cancels).

Description

Creates a dialog window for selecting a font.

Examples

```
uigetfont()  
uigetfont("arial")  
uigetfont("arial", 24)  
uigetfont("arial", 24, %T)  
uigetfont("arial", 24, %T, %F)
```

See Also

getfont

Authors

Vincent COUVERT

Name

uimenu — Create a menu or a submenu in a figure

```
h=uimenu([prop1, val1] [, prop2, val2] ...)  
h=uimenu(parent, [prop1, val1] [, prop2, val2] ...)
```

Parameters

parent

integer Handle of menu's parent

prop{1, 2 ...}

string character name of a property to set up

val{1, 2 ...}

scilab object value to affect to the corresponding property

h

integer handle of the corresponding menu

Description

This allows to create menus in a figure. If **parent** is a figure, then the menu item will be added to the menu bar of the figure. If **parent** is a menu item, then the new item will be added to the parent item, allowing to create cascaded submenu. To create a customized menu, you can use the properties listed below:

Properties

Callback

String

Instruction evaluated by the Scilab interpreter when the menu is activated. Under MacOSX, the callback will not be executed for a "button menu" (a menu without children), you must specify at least a child.

Enable

{on} | off

Enable or disable the menu. If this property is set to "on" (default), the menu is operational, but if this property is set to "off", the menu will not respond to the mouse actions and will be grayed out.

Checked

{on} | off

Menu check indicator. Setting this property to "on" (respectively "off") places (respectively removes) a check mark next to the corresponding menu item. This option can be used to create menus that indicate the state of a particular option.

Please note that a standard menu (without the option "Checked" set) has no mechanism to become a menu which will be checked by a user action and conversely.

This property is ignored for parent menus.

ForegroundColor

[1,3] real vector or string

Foreground color of the uimenu (font color). A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Label

String.

This property represents the text appearing in the menu.

Tag

String

This property is generally used to identify the menu. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Visible

{on} | off

Set the visibility of the uimenu. If this property is set to "on" (default), the uimenu is visible, but if this property is set to "off", the uimenu will not appear in its parent figure.

Examples

```
f=figure('position', [10 10 300 200]);
// create a figure
m=uimenu(f,'label', 'windows');
// create an item on the menu bar
m1=uimenu(m,'label', 'operations');
m2=uimenu(m,'label', 'quit scilab', 'callback', "exit");
//create two items in the menu "windows"
m11=uimenu(m1,'label', 'new window', 'callback',"xselect()");
m12=uimenu(m1,'label', 'clear window', 'callback',"clf()");
// create a submenu to the item "operations"
close(f);
// close the figure
```

See Also

`figure`, `uicontrol`, `set`, `get`

Authors

Bertrand Guiheneuf

Name

uinputfile — Open standard dialog box for selecting and saving file.

```
[FileName[,PathName[,FilterIndex]]]=uinputfile([file_mask[,dir[,boxTitle]]])  
PathFileName=uinputfile([file_mask[,dir[,boxTitle]]])
```

Input parameters

file_mask

a string matrix which gives the file masks to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

we can also add descriptions for masks, for example ["*.x*", "X files"; "*.bin", "BIN files"].

dir

a character string which gives the initial directory used for file search. By default uinputfile uses the previously selected directory.

boxTitle

a character string which gives the title of the uinputfile window. By default uinputfile's title is 'uinput-file'.

Output parameters

FileName

string which give the user selected file (path + file name) if user answers "Ok" or the " " string if user answers "Cancel".

PathName

is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel".

FilterIndex

is the user selected filter index on the list box if user answers "Ok" or '0' string if user answers "Cancel"

Description

Creates a dialog window for file saving.

Examples

```
uinputfile(["*.bin"; "*.sce"; "*.cos*"])  
uinputfile(["*.sci"; "*.bin"], "SCI/modules/gui/macros/")  
uinputfile(["*.sc*"; "*.bin"], "SCI/modules/gui/macros/")  
uinputfile(["*.x*", "X files"; "*.bin", "BIN files"], "SCI/modules/gui/macros/")  
uinputfile(["*.sce"; "*.bin"], "SCI/modules/gui/macros/", "Choose a file name");
```

See Also

uigetfile , uigetdir

Name

unsetmenu — interactive button or menu or submenu de-activation

```
unsetmenu(button, [nsub])  
unsetmenu(gwin, button, [nsub])
```

Parameters

button

a character string. The button name

gwin

integer. The number of graphic window where the button is installed

nsub

integer. The number of submenu to de-activate (if any). If button has no sub-menu, nsub is ignored

Description

The function allows the user to deactivate buttons or menus created by `addmenu` in the main or graphics windows command panels.

Examples

```
addmenu('foo')  
unsetmenu('foo')  
unsetmenu('File', 2)
```

See Also

`delmenu` , `setmenu` , `addmenu`

Name

usecanvas — Get/Set the main component used for Scilab graphics.

```
[canvasused] = usecanvas([usecanvasfordisplay]);
```

Parameters

canvasused

Boolean:

- %T if a "GLCanvas" is used for graphics display (Mixing uicontrols and graphics **not available**).
- %F if a "GLJPanel" is used for graphics display (Mixing uicontrols and graphics available).

usecanvasfordisplay

Boolean:

- %T to use a "GLCanvas" for graphics display (Mixing uicontrols and graphics **not available**).
- %F to use a "GLJPanel" for graphics display (Mixing uicontrols and graphics available).

Description

Scilab uses a "GLJPanel" (a Swing OpenGL component) to display graphics (plot3d, plot, ...). This component uses some high level OpenGL primitives which are not correctly supported on some platforms (depending on the operating system, video cards, drivers ...)

"GLCanvas" (AWT + OpenGL) is an alternative component provided by the Java Framework. Scilab can use it to render graphics. **However, using this component disables some capabilities such as mixing plots and uicontrols (see demo GUI/UIcontrol2). That is why it is not the default behavior.**

In some particular cases, the use of the "GLCanvas" component is forced when Scilab starts (a warning message is displayed when a graphics function is used for the first time), here is a list of these cases:

Operating System	Video Card	Details
64-bits Windows	All	When Scilab is used in a remote session.
Linux	NVIDIA Card	With free drivers.
ATI Card	With free drivers or ATI-drivers with version < 8.52.3 (Installer version < 8.8 / OpenGL version < 2.1.7873).	
INTEL Card	With Direct Rendering activated.	

You can also dynamically activate this component through Scilab using usecanvas:

- usecanvas(%T) will use "GLCanvas" for plot rendering.
- usecanvas(%F) will use "GLJPanel" for plot rendering. If your configuration is known as a one having problems with "GLJPanel" (See table above), a warning message will be displayed.

If you believe your configuration is able to use the "GLJPanel" and Scilab automatically forces the use of "GLCanvas", you can test your configuration by swithing to "GLJPanel" (usecanvas(%F)) and

try to plot something (`plot3d()` for example). If Scilab graphics work, please inform us about it by sending an email to scilab.support@scilab.org and giving us your Operating System/Video Card/Video Card driver version: this will help use to improve future versions of Scilab.

Technical Aspects

Since version 5.0, Scilab is doing an advanced use of JOGL (the Java Binding for the OpenGL), which is using the Java2D OpenGL Pipeline. For performance reasons, we use the Java2D OpenGL Pipeline. From a more technical aspect, it uses the internal buffer of the graphic cards called `pbuffer`.

Problems may occur when the driver of the graphic card does not support properly this approach. As far as we know, there is no free driver under Linux handling this feature. In the proprietary world, the situation is as follows:

- **NVIDIA:** Nvidia provides the appropriate proprietary drivers. Scilab's graphics work without any problem with most NVIDIA drivers.
- **ATI:** From the driver version 8.8, most ATI graphics supports the `pbuffer` under Linux.
- **Intel:** This is the big drawback of using the `pbuffer`. There is currently no support of `pbuffer` by any official Intel drivers under Linux.

There is a workaround for Linux to tackle this issue, but a solution is to use a software accelerated driver. To do it, in `/etc/X11/xorg.conf`, look for the *Section "Device"* and change the option *Driver* to *vesa*:

```
Section "Device"
    Identifier      "Your Graphic card"
    Driver          "vesa"
[...]
EndSection
```

Unfortunately, this solution makes Scilab pretty slow.

Under Windows, video cards manufacturers update regularly and `pbuffers` are managed. Please download recent drivers at:

- For **ATI** cards: <http://ati.amd.com/support/driver.html>
- For **Intel** cards: <http://www.intel.com/support/graphics/>
- For **Matrox** cards: <http://www.matrox.com/graphics/en/support/drivers/>
- For **NVIDIA** cards: <http://www.nvidia.com/content/drivers/drivers.asp>
- For **S3** cards: <http://www.s3graphics.com/en/resources/drivers/index.jsp>
- For **SiS** cards: <http://www.sis.com/download/>
- For **VIA** cards: <http://www.viaarena.com/default.aspx?PageID=2>

Some troubles can also occur when using Windows 2000 (video drivers are no more updated and no more supported).

In the cases where `pBuffer` create a problem, waiting for a working `pbuffer` is not a solution indeed: *The OpenGL community is moving away from pbuffers and toward the frame buffer object extension, which is a more portable and higher-performance solution for offscreen rendering than pbuffers.* [https://jogl.dev.java.net/issues/show_bug.cgi?id=163]. The JOGL team is working to fix this issue.

For more information about this problem, please refer to:

- JoGL bug database: Bug #366 [https://jogl.dev.java.net/issues/show_bug.cgi?id=366]
- Scilab bug database: Bug #3525 [http://bugzilla.scilab.org/show_bug.cgi?id=3525]
- Debian bug database: Bug #501799 [<http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=501799>]
- Freedesktop bug database: Bug #17603 [https://bugs.freedesktop.org/show_bug.cgi?id=17603]

Examples

```
// Example using GLJPanel (Mixing uicontrols and graphics is available)
usecanvas(%F);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can see the button on the figure.", "Usecanvas example", "info"

// Example using GLCanvas (Mixing uicontrols and graphics is not available, uic
usecanvas(%T);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can't see any button on the figure.", "Usecanvas example", "in
```

Authors

Vincent COUVERT

Name

waitbar — Draw a waitbar

```
winId=waitbar(x)
winId=waitbar(x,mes)
winId=waitbar(mes)
waitbar(x,winId)
waitbar(mes,winId)
waitbar(x,mes,winId)
```

Parameters

x
real, fraction to display.

mes
string, message to display.

winId
integer greater than 0, window identificator.

Description

`waitbar(x)` create a new waitbar displaying a fraction `x`, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction `x` and message `mes`, return window identificator.

`waitbar(mes)` create a new waitbar displaying a fraction 0 and message `mes`, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction 0 and message `mes`, return window identificator.

`waitbar(x,winId)`, `waitbar(mes,winId)` and `waitbar(x,mes,winId)` update waitbar with window identificator `winId`.

Examples

```
winId=waitbar('This is an example');
realtimeinit(0.3);
for j=0:0.1:1,
    realtime(3*j);
    waitbar(j,winId);
end
winclose(winId);
```

Authors

Jaime Urzua

Name

`x_choices` — interactive Xwindow choices through toggle buttons

```
rep=x_choices(title,items)
```

Parameters

`title`

vector of strings, title for the popup window.

`items`

a list of items `items=list(item1,...,itemn)`, where each `item` is also a list of the following type : `item=list('label',default_choice,choices)`. `default_choice` is an integer which gives the default toggle on entry and `choices` is a row vector of strings which gives the possible choices.

`rep`

an integer vector which gives for each item the number of the selected toggle. If user exits dialog with "cancel" button `rep` is set to `[]`.

Description

Select items through toggle lists and return in `rep` the selected items

Type `x_choices()` to see an example.

Examples

```
l1 = list('choice 1',1,['toggle c1','toggle c2','toggle c3']);
l2 = list('choice 2',2,['toggle d1','toggle d2','toggle d3']);
l3 = list('choice 3',3,['toggle e1','toggle e2']);
rep = x_choices('Toggle Menu',list(l1,l2,l3));
```

Name

`x_choose` — interactive window choice (modal dialog)

```
[num]=x_choose(items,title [,button])
```

Parameters

`items`

column vector of string, items to choose

`title`

column vector of string, comment for the dialog

`button`

string, text to appear in the button. Default value is 'Cancel'

`num`

integer, choosen item number or 0 if dialog resumed with "Cancel" button

Description

Returns in `num` the number of the chosen item.

WARNING: this dialog was not modal before Scilab 5.0, please use `x_choose_modeless` for ascendant compatibility.

Examples

```
n = x_choose(['item1';'item2';'item3'],['that is a comment';'for the dialog'])
n = x_choose(['item1';'item2';'item3'],['that is a comment'],'Return')
```

See Also

`x_choose_modeless` , `x_choices` , `x_mdialog` , `getvalue` , `unix_g`

Name

`x_choose_modeless` — interactive window choice (not modal dialog)

```
[num]=x_choose_modeless(items,title [,button])
```

Parameters

`items`

column vector of string, items to choose

`title`

column vector of string, comment for the dialog

`button`

string, text to appear in the button. Default value is 'Cancel'

`num`

integer, choosen item number or 0 if dialog resumed with "Cancel" button

Description

Returns in `num` the number of the chosen item.

Examples

```
n = x_choose_modeless(['item1';'item2';'item3'],['that is a comment';'for the d  
n = x_choose_modeless(['item1';'item2';'item3'],['that is a comment'],'Return')
```

See Also

`x_choose` , `x_choices` , `x_mdialog` , `getvalue` , `unix_g`

Name

`x_dialog` — Dialog for interactive multi-lines input.

```
result=x_dialog(labels,valueini)
```

Parameters

`labels`

column vector of strings, comment for dialog

`valueini`

n column vector of strings, initial value suggested

`result`

User answer: n column vector of strings if returned with "Ok" button or [] if returned with "Cancel" button

Description

Opens a dialog for interactive multi-lines input.

Examples

```
gain=evstr(x_dialog('value of gain ?','0.235'))
x_dialog(['Method';'enter sampling period'],'1')
m=evstr(x_dialog('enter a 3x3 matrix ',['[0 0 0';'0 0 0';'0 0 0]')))
```

See Also

`x_mdialog` , `x_matrix` , `evstr` , `execstr`

Name

x_matrix — Xwindow editing of matrix

```
[result]=x_matrix(label,matrix-init)
```

Parameters

label
character string (name of matrix)

matrix-init
real matrix

Description

For reading or editing a matrix .

Examples

```
m=evstr(x_matrix('enter a 3x3 matrix ',rand(3,3)))
```

See Also

x_mdialog , x_dialog

Name

`x_mdialog` — Dialog for interactive vector/matrix input.

```
result=x_mdialog(title, labels, default_inputs_vector)
result=x_mdialog(title, labelsv, labelsh, default_input_matrix)
```

Parameters

`title`

column vector of strings, dialog general comment

`labels`

n column vector of strings, `labels(i)` is the label of the ith required value

`default_input_vector`

n column vector of strings, `default_input_vector(i)` is the initial value of the ith required value

`labelsv`

n vector of strings, `labelsv(i)` is the label of the ith line of the required matrix

`labelsh`

m vector of strings, `labelsh(j)` is the label of the jth column of the required matrix

`default_input_matrix`

n x m matrix of strings, `default_input_matrix(i,j)` is the initial value of the (i,j) element of then required matrix

`result`

n x m matrix of string if returned with "Ok" button or [] if returned with "Cancel" button

Description

Opens a dialog for interactive vector/matrix input.

Examples

```
txt = ['magnitude'; 'frequency'; 'phase    '];
sig = x_mdialog('enter sine signal', txt, ['1'; '10'; '0'])
mag = evstr(sig(1))
frq = evstr(sig(2))
ph  = evstr(sig(3))

rep = x_mdialog(['System Simulation'; 'with PI regulator'], ...
                ['P gain'; 'I gain '], [' '; ' '])
```

See Also

`editvar` , `x_dialog` , `x_choose` , `messagebox` , `getvalue` , `evstr` , `execstr` , `editvar`

Name

x_message_modeless — X window modeless message

```
x_message_modeless(strings)
```

Parameters

strings
vector of characters strings to be displayed

Description

for displaying a message (information, user-guide ...). The function returns immediately. The message window is killed when "Ok" button is clicked.

Examples

```
x_message_modeless(['This is a modeless message'  
                   'Scilab may continue computation'  
                   ''  
                   'Click on ""Ok"" to close the message'])  
x_message_modeless('Now two message windows are opened')
```

See Also

x_dialog , x_mdialog , messagebox

Part XVI. Dynamic/incremental Link

Name

G_make — call make or nmake

```
Rfiles=G_make(files,dllname)
```

Parameters

files

a character string or a vector of character string.

dllname

a character string.

Rfiles

vector of character string. **Rfiles** can be used as a first argument when calling **addinter** function.

Description

On Unix like systems **G_make** calls the **make** utility for building target **files** and returns the value of **files** in the variable **Rfiles**. On windows platforms, **G_make** calls the **nmake** utility for building target **dllname** and it returns the value of **dllname** in the variable **Rfiles**. Of course **G_make** will work if appropriate Makefiles are provided in the current Scilab directory.

G_make can be used to provide OS independant call to **addinter**.

Examples

```
if MSDOS then
  txt = ['ex1c.dll:',
        ' @echo -----',
        ' @echo From Makefile.mak',
        ' @echo -----',
        ' '];
  mputl(txt,TMPDIR+'/makefile.mak')
  current_dir = pwd();
  cd TMPDIR
  files=G_make([TMPDIR+'/ex1cI.o',TMPDIR+'/ex1c.o'],'ex1c.dll');// compilation
  //
  //addinter(files,'foobar','foubare');// link
  cd(current_dir);
end
```

See Also

addinter , **api_scilab**

Name

VCtoLCCLib — converts Ms VC libs to LCC-Win32 libs.

```
VCtoLCCLib()
```

Description

converts Ms VC libs to LCC-Win32 libs.

Examples

```
b0K=chooselcccompiler();  
VCtoLCCLib()
```

Authors

Allan CORNET

Name

addinter — new functions interface dynamic link at run time.

```
addinter(files, spname, fcts)
```

Parameters

files

a character string or a vector of character string contain object files used to define the new Scilab interface routine (interface code, user routines or libraries, system libraries).

spname

a character string. Name of interface routine entry point

fcts

vector of character strings. The name of new Scilab function implemented in the new interface (in `fin` the order).

Description

`addinter` performs dynamic link of a compiled C or Fortran new Scilab interface routine and define corresponding scilab functions.

You can use the command `link('show')` to get the number of the shared libraries. And to reload a new version of an interface a call to `ulink` is necessary to get rid of the old version.

See `link` for more precision on use.

Number of 'addinter' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

Number of functions implemented in a interface is limited to 1000.

Examples

```
if haveacompile() then

chdir(TMPDIR);
mkdir('example_addinter');
chdir('example_addinter');

src = ['#include <math.h>'
'#include <stack-c.h>'
'#include <api_scilab.h>'
'#include <Scierror.h>'
'#include <localization.h>'
''
'static double fun2(double x);'
''
'void fun1(double *x,double *y) {'
'{'
' *y=fun2(*x)/(*x); '
'} '
''
'static double fun2(double x)'
'{'
' return( sin(x+1));'
```

```

    }'
    ,,'
    'int sci_fun1(char *fname)'
    '{'
    '    int iType1 = 0;'
    '    SciErr sciErr;'
    '    int m1 = 0, n1 = 0;'
    '    double *pdVarOne = NULL;'
    '    int *piAddressVarOne = NULL;'
    ,,'
    '    CheckRhs(1,1);'
    '    CheckLhs(1,1);'
    ,,'
    '    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
    '    if(sciErr.iErr)'
    '    {'
    '        printError(&sciErr, 0);'
    '        return 0;'
    '    }'
    ,,'
    '    sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
    '    if(sciErr.iErr)'
    '    {'
    '        printError(&sciErr, 0);'
    '        return 0;'
    '    }'
    ,,'
    '    if (iType1 != sci_matrix)'
    '    {'
    '        Scierror(999, _("%s: Wrong type for input argument #%d: A string expected.\n",
    '        return 0;'
    '    }'
    ,,'
    '    sciErr = getMatrixOfDouble(pvApiCtx, piAddressVarOne, &m1, &n1, &pdVarOne);'
    '    if(sciErr.iErr)'
    '    {'
    '        printError(&sciErr, 0);'
    '        return 0;'
    '    }'
    ,,'
    '    fun1(pdVarOne, pdVarOne);'
    '    LhsVar(1) = 1;'
    '    return 0;'
    '}''];

mputl(src, TMPDIR + '/example_addinter/example_addinter.c');
files=['example_addinter.c'];
ilib_build('addinter', ['fun1_in_scilab', 'sci_fun1'], files, []);
disp(mgetl('loader.sce'));
exec loader.sce;
fun1_in_scilab(%pi)

end // if haveacompiler()

```

See Also

[link](#), [intersci](#), [newfun](#), [clearfun](#), [api_scilab](#)

Authors

Allan CORNET

Name

`c_link` — check incremental/dynamic link

```
c_link(routine-name)
[test,ilib]=c_link(routine-name)
test=c_link(routine-name,num)
```

Parameters

`routine-name`

a character string

`num`

`test`

boolean, indicates if there is a shared library which contains `routine-name`.

`ilib`

a scalar, the number of the shared library which contains `routine-name`

Description

`c_link` is a boolean function which checks if the routine `routine-name` is currently linked. This function returns a boolean value true or false. When used with two return values, the function `c_link` returns a boolean value in `test` and the number of the shared library which contains `routine-name` in `ilib` (when `test` is true).

See Also

`link` , `fort` , `api_scilab`

Name

call — Fortran or C user routines call

```
// long form 'out' is present
[y1,...,yk]=call("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// short form : no 'out' parameter
[y1,...,yk]=call("ident",x1,...,xn)
```

Parameters

"ident"

string.

xi

real matrix or string

pxi, pyi

integers

txi, tyi

character string "d", "r", "i" or "c".

Description

Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with Scilab. This link may be done:

- with Scilab "link" command (dynamical linking) during the Scilab session.(see [link](#))

On Windows, C functions must use cdecl calling convention name (see options in your C compiler(default calling convention for x86 C compilers)).

There are two forms of calling syntax, a short one and a long one. The short one will give faster code and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident"

is the name of the called subroutine.

x1,...,xn

are input variables (real matrices or strings) sent to the routine,

px1,...,pxn

are the respective positions of these variables in the calling sequence of the routine "ident" and

tx1,...,txn

are their types ("r", "i", "d" and "c" for real (float) , integer, double precision and strings)

"out"

is a keyword used to separate input variables from output variables. when this key word is present it is assumes that the long form will be used and when it is not prsent, the short form is used.

[ny1, my1]

are the size (# of rows and columns. For 'c' arguments,m1*n1 is the number of charaters) of output variables and

py1, ...

are the positions of output variables (possibly equal to px_i) in the calling sequence of the routine. The py_i 's integers must be in increasing order.

"ty1", ...

are the Fortran types of output variables. The k first output variables are put in $y_1, \dots, y_k$.

If an output variable coincides with an input variable (i.e. $py_i=px_j$) one can pass only its position py_i . The size and type of y_i are then the same as those of x_i . If an output variable coincides with an input variable and one specify the dimensions of the output variable $[myl, nyl]$ must follow the compatibility condition $mxk \times nxk \geq myl \times nyl$.

Examples

```
//Example 1 with a simple C code
f1=['#include <math.h>'
    'void fooc(double c[],double a[],double *b,int *m,int *n)'
    '{'
    '    int i;'
    '    for ( i =0 ; i < (*m)*(*n) ; i++) '
    '        c[i] = sin(a[i]) + *b; '
    '}'];

mputl(f1, 'fooc.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('fooc', 'fooc.c', [], "c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);

// Inputs:
// a is in position 2 and double
// b          3      double
// n          4      integer
// m          5      integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");

//Example 2 with a simple Fortran code
f1=['      subroutine foof(c,a,b,n,m)'
    '      integer n,m'
    '      double precision a(*),b,c(*)'
    '      do 10 i=1,m*n '
    '          c(i) = sin(a(i))+b'
    '      10 continue'
    '      end'];
mputl(f1, 'foof.f')
```



```
//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('foof','foof.c',[],"f")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=call("foof",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

See Also

[link](#), [c_link](#), [intersci](#), [addinter](#), [api_scilab](#)

Name

chooselcccompiler — choose LCC-Win32 as the default C Compiler.

```
bOK=chooselcccompiler()
```

Parameters

bOK

returns %T if LCC-Win32 is the default C Compiler.

Description

choose LCC-Win32 as the default C Compiler.

Examples

```
bOK=chooselcccompiler()
```

Authors

Allan CORNET

Name

`configure_lcc` — set environments variables for LCC-Win32 C Compiler.

```
bOK=configure_lcc()
```

Parameters

`bOK`

returns %T if environments variables for LCC-Win32 C Compiler are OK.

Description

set environments variables for LCC-Win32 C Compiler.

Examples

```
bOK=configure_lcc()
```

Authors

Allan CORNET

Name

`configure_ifort` — set environments variables for Intel Fortran Compiler (Windows).

```
bOK=configure_msifort()
```

Parameters

`bOK`

returns %T if environments variables for Intel fortran (9 or 10) Compiler are OK.

Description

set environments variables for Intel fortran (9 or 10) Compiler.

Examples

```
bOK = configure_msifort()
```

Authors

Allan CORNET

Name

configure_msvc — set environments variables for Microsoft C Compiler.

```
bOK=configure_msvc()
```

Parameters

bOK

returns %T if environments variables for Ms C Compiler are OK.

Description

set environments variables for Microsoft C Compiler.

Examples

```
bOK=configure_msvc()
```

Authors

Allan CORNET

Name

dllinfo — provides information about the format and symbols provided in executable and DLL files (Windows).

```
infolist = dllinfo(filename,option)
```

Parameters

filename

a string : a filename .dll or .exe file

option

a string : 'machine' , 'exports' , 'imports'

infolist

a list :

infolist(1) : a string : name of dll or executable.

infolist(2) : a string matrix : symbols (imported or exported) or machine type (x86 or x64).

Description

This tool provides information about the format and symbols (imported or exported) provided in executable and DLL files.

This tool is based on dumpbin.exe. A tool provided with Visual studio SDK.

Examples

```
if MSDOS then
    filename = SCI+'\bin\libscilab.dll';

    dllinfolist = dllinfo(filename,'machine');
    printf('Machine destination of %s: %s\n',dllinfolist(1),dllinfolist(2));

    dllinfolist = dllinfo(filename,'imports');
    printf('Dlls dependencies of %s:\n',filename);
    for i=1:size(dllinfolist)
        printf('%s\n',dllinfolist(i)(1));
    end

    dllinfolist = dllinfo(filename,'exports');
    printf('Dll exports of %s:\n',filename);
    disp(dllinfolist);
end
```

See Also

[addinter](#), [link](#), [ilib_compile](#), [ilib_gen_Make](#), [ilib_gen_gateway](#), [ilib_gen_loader](#), [ilib_for_link](#)

Authors

Allan CORNET

Name

findlcccompiler — detects LCC-Win32 C Compiler

```
ret=findlcccompiler()
```

Parameters

ret
returns %T or %F

Description

detects LCC-Win32 C Compiler.

Examples

```
ret=findlcccompiler()
```

Authors

Allan CORNET

Name

findmsifortcompiler — detects Intel fortran Compiler

```
ifortv=findmsifortcompiler()
```

Parameters

ifortv
returns 'ifort90','ifort10','ifort11','unknown'

Description

detects Intel fortran Compiler (Windows).

Examples

```
ifortv = findmsifortcompiler()
```

Authors

Allan CORNET

Name

findmsvccompiler — detects Microsoft C Compiler

```
msvc=findmsvccompiler()
```

Parameters

msvc

returns

'msvc70','msvc71','msvc80express','msvc80std','msvc80pro','msvc90express','msvc90std','msvc90pro','unknown'

Description

detects Microsoft C Compiler.

Examples

```
msvc=findmsvccompiler()
```

Authors

Allan CORNET

Name

fort — Fortran or C user routines call

```
// long form 'out' is present
[y1,...,yk]=fort("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// short form : no 'out' parameter
[y1,...,yk]=fort("ident",x1,...,xn)
```

Parameters

"ident"
string.

xi
real matrix or string

pxi, pyi
integers

txi, tyi
character string "d", "r", "i" or "c".

Description

Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with Scilab. This link may be done:

- with Scilab "link" command (dynamic link) during the Scilab session.(see [link](#))

There are two forms of calling syntax, a short one and a long one. The short one will give faster code and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident"
is the name of the called subroutine.

x1,...,xn
are input variables (real matrices or strings) sent to the routine,

px1,...,pxn
are the respective positions of these variables in the calling sequence of the routine "ident" and

tx1,...,txn
are their types ("r", "i", "d" and "c" for real (float) , integer, double precision and strings)

"out"
is a keyword used to separate input variables from output variables. when this key word is present it is assumes that the long form will be used and when it is not prsent, the short form is used.

[ny1, my1]
are the size (number of rows and columns. For 'c' arguments,m1*n1 is the number of charaters) of output variables and

py1, ...
are the positions of output variables (possibly equal to pxi) in the calling sequence of the routine. The pyi's integers must be in increasing order.

"ty1", ...

are the Fortran types of output variables. The k first output variables are put in $y_1, \dots, y_k$.

If an output variable coincides with an input variable (i.e. $py_i = px_j$) one can pass only its position py_i . The size and type of y_i are then the same as those of x_i . If an output variable coincides with an input variable and one specifies the dimensions of the output variable $[myl, nyl]$ must follow the compatibility condition $mxk \times nxk \geq myl \times nyl$.

For example the following program:

```
subroutine foof(c,a,b,n,m)
  integer n,m
  double precision a(*),b,c(*)
  do 10 i=1,m*n
    c(i) = sin(a(i))+b
  10 continue
end
```

```
link("foof"+getdynlibext(),"foof")
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
// Inputs:
// a is in position 2 and double
// b          3      double
// n          4      integer
// m          5      integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=fort("foof",a,2,"d",b,3,"d",n,4,"i",m,5,"i","out",[m,n],1,"d");
```

returns the matrix $c=2*a+b$.

The same example coded in C:

```
void fooc(c,a,b,m,n)
double a[],*b,c[];
int *m,*n;
{
  double sin();
  int i;
  for ( i =0 ; i <= (*m)*(*n) ; i++)
    c[i] = sin(a[i]) + *b;
}
```

```
link("fooc"+getdynlibext(),"fooc","C") // note the third argument
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=fort("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

See Also

call, link, c_link, intersci, addinter, api_sciab

Name

getdynlibext — get the extension of dynamic libraries on your operating system.

```
ret=getdynlibext()
```

Description

get the extension of dynamic libraries on your operating system.

ret=getdynlibext() returns (.so on linux, .dylib on MacOS, .sl HP-UX,.dll on Windows, ...).

Examples

```
getdynlibext()
```

Authors

Allan CORNET

Name

haveacompile — detect if you have a C compiler.

```
bOK=haveacompile()
```

Parameters

bOK

returns %T if you have a C compiler.

Description

detect if you have a C compiler.

Examples

```
bOK = haveacompile();
```

See Also

findlccompiler , findmsvccompiler

Name

ilib_build — utility for shared library management

```
ilib_build(lib_name, table, files, libs [, makename, ldflags, cflags, fflags, ismex, cc
```

Parameters

lib_name

a character string, the generic name of the library without path and extension.

table

2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files

string matrix giving source (from Scilab 5.0) or object files needed for shared library creation

libs

string matrix giving extra libraries needed for shared library creation

makename

character string. The path of the Makefile file without extension.

This parameter is useless since Scilab 5.0. Default value to use: [].

A warning will be displayed in Scilab 5.3 if you use another value than the default.

ldflags, cflags, fflags

character strings to provide options for the loader, the C compiler and the Fortran compiler.

ismex

Internal variable to specify if we are working with mex or not.

cc

Provide the name of the C compiler.

Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with `addinter`

Many examples are provided in `SCI/modules/dynamic_link/examples` directory. They are all released into the public domain.

Note that a compiler must be available on the system to use this function.

Languages handled by this function are: C, C++, Fortran and Fortran 90.

Examples (C code)

```
//Here with give a complete example on adding new primitive to Scilab
//create the procedure files

cd TMPDIR;
mkdir('example_ilib_build_c');
cd('example_ilib_build_c');

f1=['extern double fun2();'
    'void fun1(double *x, double *y)'
    '{*y=fun2(*x)/(*x);}'];
```

```

mputl(f1,TMPDIR + '/example_ilib_build_c/fun1.c');

f2=['#include <math.h>'
    'double fun2(double x)'
    '{ return( sin(x+1.));}'];
mputl(f2,TMPDIR + '/example_ilib_build_c/fun2.c');

//creating the interface file
i=['#include <stdlib.h>'
  '#include <stack-c.h>'
  '#include <api_sciab.h>'
  '#include <Scierror.h>'
  '#include <localization.h>'
  ''
  'extern int fun1 ( double *x, double *y);'
  ''
  'int sci_fun1(char *fname)'
  '{'
  '  int iType1 = 0;'
  '  SciErr sciErr;'
  '  int m1 = 0, n1 = 0;'
  '  double *pdVarOne = NULL;'
  '  int *piAddressVarOne = NULL;'
  ''
  '  CheckRhs(1,1);'
  '  CheckLhs(1,1);'
  ''
  '  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
  '  if(sciErr.iErr)'
  '  {'
  '    printError(&sciErr, 0);'
  '    return 0;'
  '  }'
  ''
  '  sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
  '  if(sciErr.iErr)'
  '  {'
  '    printError(&sciErr, 0);'
  '    return 0;'
  '  }'
  ''
  '  if (iType1 != sci_matrix)'
  '  {'
  '    Scierror(999,_"("%s: Wrong type for input argument #%d: A string expected")",fname,iType1);'
  '    return 0;'
  '  }'
  ''
  '  sciErr = getMatrixOfDouble(pvApiCtx, piAddressVarOne, &m1, &n1, &pdVarOne);'
  '  if(sciErr.iErr)'
  '  {'
  '    printError(&sciErr, 0);'
  '    return 0;'
  '  }'
  ''
  '  fun1(pdVarOne, pdVarOne);'
  '  LhsVar(1) = 1;'
  '  return 0;'

```

```

    '}]];
mputl(i,TMPDIR + '/example_ilib_build_c/sci_fun1.c');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['fun1.c','fun2.c','sci_fun1.c'];
ilib_build('build_c',['fun1','sci_fun1'],files,[]);

// load the shared library

exec loader.sce;

//using the new primitive
fun1(33)

```

Examples (C code - previous Scilab API < 5.2)

```

cd TMPDIR;
mkdir('example_ilib_build_c_old');
cd('example_ilib_build_c_old');

//Here with give a complete example on adding new primitive to Scilab
//create the procedure files
f1=['extern double fun2();'
    'void fun1(double *x, double *y)'
    '{*y=fun2(*x)/(*x);}'];
mputl(f1,'fun1.c')

f2=['#include <math.h>'
    'double fun2(double x)'
    '{ return( sin(x+1.));}'];
mputl(f2,'fun2.c');

//creating the interface file
i=['#include ""stack-c.h""'
    '#include ""stackTypeVariable.h""'
    '#include ""version.h""'
    '#if SCI_VERSION_MAJOR <= 5'
    '#if SCI_VERSION_MINOR < 2'
    '    #error ""This example is obsolete see help ilib_buid""'
    '#endif'
    '#endif'
    ''
    'extern int fun1 ( double *x, double *y);'
    'int intfun1(char *fname)'
    '{'
    '    int m1,n1,l1;'
    '    CheckRhs(1,1);'
    '    CheckLhs(1,1);'
    '    GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m1, &n1, &l1);'
    '    fun1(stk(l1),stk(l1));'
    '    LhsVar(1) = 1;'
    '    return 0;'
    '}'];
mputl(i,'intfun1.c')

```



```
//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['fun1.c','fun2.c','intfun1.c'];
ilib_build('ilib_c_old',['scifun1','intfun1'],files,[]);

// load the shared library

exec loader.sce

//using the new primitive
scifun1(33)
```

Examples (C++ code)

```
cd TMPDIR;
mkdir('example_ilib_build_cpp');
cd('example_ilib_build_cpp');

i=['#include <string>'
'extern ""C"" {'
'#include <stdlib.h>'
'#include <stack-c.h>'
'#include <api_sciLab.h>'
'#include <localization.h>'
'#include <Scierror.h>'
''
'int sci_cppfind(char *fname)'
'{'
''
'  SciErr sciErr;'
'  int *piAddressVarOne = NULL;'
'  char *pStVarOne = NULL;'
'  int iType1 = 0;'
'  int lenStVarOne = 0;'
'  int m1 = 0, n1 = 0;'
''
'  int *piAddressVarTwo = NULL;'
'  char *pStVarTwo = NULL;'
'  int iType2 = 0;'
'  int lenStVarTwo = 0;'
'  int m2 = 0, n2 = 0;'
''
'  int m_out = 0;'
'  int n_out = 0;'
''
'  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
'  if(sciErr.iErr)'
'  {'
'    printError(&sciErr, 0);'
'    return 0;'
'  }'
''
'  sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
'  if(sciErr.iErr)'
```

```

    {
        printError(&sciErr, 0);
        return 0;
    }
    ,
    if (iType1 != sci_strings)
    {
        Scierror(999, _("%s: Wrong type for input argument #%d: A string expected"), fname);
        return 0;
    }
    ,
    sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddressVarTwo);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    ,
    sciErr = getVarType(pvApiCtx, piAddressVarTwo, &iType2);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    ,
    if (iType2 != sci_strings)
    {
        Scierror(999, _("%s: Wrong type for input argument #%d: A string expected"), fname);
        return 0;
    }
    ,
    sciErr = getMatrixOfString(pvApiCtx, piAddressVarOne, &m1, &n1, &lenStVarOne);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    ,
    pStVarOne = new char[lenStVarOne + 1];
    if (pStVarOne == NULL)
    {
        Scierror(999, _("%s : Memory allocation error.\n"), fname);
        return 0;
    }
    ,
    sciErr = getMatrixOfString(pvApiCtx, piAddressVarTwo, &m2, &n2, &lenStVarTwo);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    ,
    pStVarTwo = new char[lenStVarTwo + 1];
    if (pStVarTwo == NULL)
    {
        Scierror(999, _("%s : Memory allocation error.\n"), fname);
        return 0;
    }
    ,
    {

```

```

    '
    ' sciErr = getMatrixOfString(pvApiCtx, piAddressVarOne, &m1, &n1, &lenStVarOne);
    ' if(sciErr.iErr)
    ' {
    '     printError(&sciErr, 0);
    '     return 0;
    ' }
    '
    ' sciErr = getMatrixOfString(pvApiCtx, piAddressVarTwo, &m2, &n2, &lenStVarTwo);
    ' if(sciErr.iErr)
    ' {
    '     printError(&sciErr, 0);
    '     return 0;
    ' }
    '
    ' std::string myMessage(pStVarOne);
    ' std::string search(pStVarTwo);

    ' delete pStVarTwo;
    ' delete pStVarOne;

    ' double dOut = 0.0;
    '
    ' if (myMessage.find(search) != std::string::npos) {
    '     dOut = myMessage.find(search); /* The actual operation */
    ' } else {
    '     dOut = -1; /* Substring not found */
    ' }

    ' m_out = 1;
    ' n_out = 1;
    ' sciErr = createMatrixOfDouble(pvApiCtx, RhS + 1, m_out, n_out, &dOut);
    ' if(sciErr.iErr)
    ' {
    '     printError(&sciErr, 0);
    '     return 0;
    ' }
    '
    ' LhsVar(1) = RhS + 1;
    ' return 0;
    ' } /* extern ""C"" */
    ' }'];

mputl(i,TMPDIR + '/example_ilib_build_cpp/sci_cppfind.cxx');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files = ['sci_cppfind.cxx'];
ilib_build('ilib_build_cpp',['cppfind','sci_cppfind'],files,[]);

// load the shared library

exec loader.sce;

// Small test to see if the function is actually working.
if cppfind("my very long string","long") <> 8 pause, end
if cppfind("my very long string","very") <> 3 pause, end

```

```
if cppfind("my very long string","short") <> -1 pause, end
```

Examples (C++ code - previous Scilab API < 5.2)

```
cd TMPDIR;
mkdir('example_ilib_build_cpp_old');
cd('example_ilib_build_cpp_old');

i=['#include <string>'
'extern ""C"" {'
'#include ""stack-c.h""'
'#include ""version.h""'
'#if SCI_VERSION_MAJOR <= 5'
'#if SCI_VERSION_MINOR < 2'
'    #error ""This example is obsolete see help ilib_buid""'
'#endif'
'#endif'
''
'int sci_cppfind(char *fname) {'
'    int m1 = 0, n1 = 0, l1;'
'    char *inputString1, *inputString2;'
'    int m2 = 0, n2 = 0, l2;'
'    int m3 = 0, n3 = 0;'
'    double *position = NULL; /* Where we will store the position */'
'    CheckRhs(2,2); /* Check the number of input argument */'
'    CheckLhs(1,1); /* Check the number of output argument */'
'    GetRhsVar(1, ""c"", &m1, &n1, &l1); /* Retrieve the first input argument'
'    inputString1=cstk(l1);'
'    GetRhsVar(2, ""c"", &m2, &n2, &l2); /* Retrieve the second input argument'
'    inputString2=cstk(l2);'
'    std::string myMessage (inputString1);'
'    std::string search (inputString2);'
'    m3=1;n3=1;'
'    position = new double[1];'
'    if (myMessage.find(search) != std::string::npos) {'
'        position[0] = myMessage.find(search); /* The actual operation */'
'    } else {'
'        position[0] = -1; /* Substring not found */'
'    }'
'    CreateVarFromPtr(Rhs+1,""d"",&m3,&n3,&position); /* Create the output arg'
'    LhsVar(1) = Rhs+1;'
'    delete[] position;'
'    return 0;'
'}'
'}'];

mputl(i,'sci_cppfind.cxx');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['sci_cppfind.cxx'];
ilib_build('foo_old',['cppfind','sci_cppfind'],files,[]);

// load the shared library
```

```
exec loader.sce

// Small test to see if the function is actually working.
if cppfind("my very long string","long") <> 8 pause, end
if cppfind("my very long string","very") <> 3 pause, end
if cppfind("my very long string","short") <> -1 pause, end
```

Examples (Fortran 90 code)

```
cd TMPDIR;
mkdir('example_ilib_build_f90');
cd('example_ilib_build_f90');

sourcecode=['subroutine incrdoublef90(x,y)'
            '  implicit none'
            '  double precision, intent(in) :: x'
            '  double precision, intent(out) :: y'
            '  y=x+1'
            'end subroutine incrdoublef90'];
mputl(sourcecode,'incrdoublef90.f90');
libpath=ilib_for_link('incrdoublef90','incrdoublef90.f90',[],'f');
exec loader.sce
n=1.;
m=call("incrdoublef90",n,1,"d","out",[1,1],2,"d");
if abs(m-2.)>%eps then pause,end
n=2.;
m=call("incrdoublef90",n,1,"d","out",[1,1],2,"d");
if abs(m-3.)>%eps then pause,end
```

See Also

[addinter](#), [link](#), [ilib_compile](#), [ilib_gen_Make](#), [ilib_gen_gateway](#), [ilib_gen_loader](#), [ilib_for_link](#), [api_scilab](#)

Author

Allan CORNET

Name

`ilib_compile` — `ilib_build` utility: executes the Makefile produced by `ilib_gen_Make`

```
libn=ilib_compile(lib_name,makename [,files,ldflags,cflags,fflags,cc])
```

Parameters

`lib_name`

a character string, the generic name of the library without path and extension.

`makename`

character string. The path of the Makefile file without extension.

`files`

optionnal vector of character strings. If `files` is given the make is performed on each target contained in `files` then a whole make is performed

`libn`

character string. The path of the actual generated shared library file.

`ldflags,cflags,fflags,cc`

character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. `cc` provides the name of the compiler.

Description

Utility function used by `ilib_build`

This executes the Makefile produced by `ilib_gen_Make`, compiles the C and fortran files and generates the shared library.

Shared libraries can then be used with the `link` and `addinter` Scilab function for incremental/dynamic link.

Note that a compiler must be available on the system to use this function.

See Also

`addinter` , `link` , `ilib_build` , `ilib_gen_Make` , `ilib_gen_gateway` , `ilib_gen_loader` , `ilib_for_link` , `api_scilab`

Name

`ilib_for_link` — utility for shared library management with `link`

```
libn=ilib_for_link(names, files, libs, flag [, makename [, loadername [, libname [, ld
```

Parameters

`names`

a string matrix giving the entry names which are to be linked.

`files`

string matrix giving source (from Scilab 5.0) or object files needed for shared library creation

`libs`

string matrix giving extra libraries needed for shared library creation

`flag`

a string flag ("c" or "f") for C or Fortran entry points.

`makename`

character string. The pathname of the Makefile file without extension (default value `MakeLib`).

This parameter is useless since Scilab 5.0.

A warning will be displayed in Scilab 5.3 if you use another value than the default.

`loadername`

character string. The pathname of the loader file (default value is `loader.sce`).

`libname`

optional character string. The name of the generated shared library (default value is "", and in this case the name is derived from `names(1)`).

`ldflags`

optional character string. It can be used to add specific linker options in the generated Makefile. Default value is ""

`cflags`

optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is ""

`fflags`

optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is ""

`cc`

optional character string. It can be used to specify a C compiler. Default value is ""

`libn`

character string. The path of the really generated shared library file.

Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with the `link` function. New entry points given by `names` are then accessible through the `call` function or with non linear tools `ode`, `optim`,...

The file to compile are supposed to be located given by `makename`. If `makename` sets a path different to the current directory, `loader` script must be located in the same directory using the `loadername` variable.

Many examples are provided in `SCI/modules/dynamic_link/examples` directory. They are all released into the public domain.

Note that a compiler must be available on the system to use this function.

Languages handle by this function are: C, C++, Fortran and Fortran 90.

Examples (C code)

```
if haveacompile() then

chdir(TMPDIR)
f1=['int ext1c(int *n, double *a, double *b, double *c)'
   '{int k;'
   '  for (k = 0; k < *n; ++k) '
   '    c[k] = a[k] + b[k];'
   '  return(0);}'];

mputl(f1, 'fun1.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('ext1c', 'fun1.c', [], "c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3];b=[4,5,6];n=3;
c=call('ext1c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
if norm(c-(a+b)) > %eps then pause,end

end
```

See Also

`addinter` , `link` , `ilib_compile` , `ilib_gen_Make` , `ilib_gen_gateway` , `ilib_gen_loader` , `ilib_for_link` , `api_scilab`

Name

`ilib_gen_Make` — utility for `ilib_build`: produces a Makefile for building shared libraries

```
Makename=ilib_gen_Make(name, files, libs, makename [,with_gateway, ldflags, cflags, f
```

Parameters

`lib_name`

a character string, the generic name of the library without path and extension.

`files`

a vector of character string. The names of the C or Fortran files without the extension and the path part.

`libs`

a vector of character string, additionnal libraries paths or [].

`makename`

character string. The path of the Makefile file.

This parameter is useless since Scilab 5.0.

A warning will be displayed in Scilab 5.3 if you use another value than the default.

`with_gateway`

a boolean. If true a file with name `<lib_name>_gateway` is added. Default value is %t

`ldflags`

a string. It can be used to add specific linker options in the generated Makefile. Default value is "

`cflags`

a string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

`fflags`

a string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

`cc`

a string. The name of the C compiler. Default value is the C compiler detected on the host.

Description

Utility function used by `ilib_build`

This function generates a Makefile adapted to the Operating System for building shared libraries to be loaded in Scilab. Proper options and paths are set.

Shared libraries can then be used with the `link` and `addinter` scilab function for incremental/dynamic linking.

The shared library is build from a set of C or Fortran routines stored in a directory and if required from a set of external libraries.

Files are not required to exist, when Makefile is generated, but of course are required for executing the Makefile.

Only use this function if you know what you are doing (it is a semi-private function).

See Also

[addinter](#) , [link](#) , [ilib_build](#) , [ilib_compile](#) , [ilib_gen_gateway](#) , [ilib_gen_loader](#) , [ilib_for_link](#) , [api_scilab](#)

Name

`ilib_gen_cleaner` — utility for `ilib_build`: generates a cleaner file

```
ilib_gen_cleaner(makename,[loadername],[files])
```

Parameters

`makename`

character string. The pathname of the Makefile file without extension (default value `MakeLib`).

This parameter is useless since Scilab 5.0.

A warning will be displayed in Scilab 5.3 if you use another value than the default.

`loadername`

character string. The pathname of the loader file (default value is `loader.sce`).

`files`

matrix of character string. files to delete.

Description

This is internal function used by `ilib_build` and `ilib_for_link`. This function generates a cleaner file.

Only use this function if you know what you are doing (it is a semi-private function).

See Also

`ilib_gen_loader` , `ilib_build` , `ilib_for_link` , `api_scilab`

Name

`ilib_gen_gateway` — utility for `ilib_build`, generates a gateway file.

```
ilib_gen_gateway(name, table)
```

Parameters

`name`

a character string, the generic name of the library without path and extension.

`table`

2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

Description

Utility function used by `ilib_build` This function generates a gateway file used by `addinter`.

if `WITHOUT_AUTO_PUTLHSVAR` variable is defined and equals to `%t`, `PutLhsVar()`; will need to manage `PutLhsVar` in each interface as internal scilab functions.

In another case (default, for compatibility with previous version) , `PutLhsVar()`; is added in each interface.

You can also see `SCI/contrib/toolbox_skeleton/sci_gateway/c/builder_gateway_c.sce` (as example about `WITHOUT_AUTO_PUTLHSVAR`)

Example about `WITHOUT_AUTO_PUTLHSVAR = %t`

```
cd TMPDIR
WITHOUT_AUTO_PUTLHSVAR = %t;
name = 'gw_example1';
table = ['sci_func1', 'func1']
ilib_gen_gateway(name, table)

// generated gateway
mgetl(TMPDIR+'gw_example1.c')
```

```
int sci_func1(char *fname)
{
    // ... your C code
    // you need to add a explicit PutLhsVar();
    // as internal all gateways of scilab

    PutLhsVar();
    return 0;
}
```

Example about `WITHOUT_AUTO_PUTLHSVAR = %f` (default)

```
cd TMPDIR
WITHOUT_AUTO_PUTLHSVAR = %f;
```

```
name = 'gw_example2';  
table = ['sci_func2', 'func2']  
ilib_gen_gateway(name, table)
```

```
// generated gateway  
mgetl(TMPDIR+'gw_example2.c')
```

```
int sci_func2(char *fname)  
{  
    // ... your code  
    // you do NOT need to add a explicit PutLhsVar();  
    // added by scilab after call to sci_func2  
    // default mode in scilab 4  
  
    return 0;  
}
```

See Also

[addinter](#), [link](#), [ilib_build](#), [ilib_compile](#), [ilib_gen_Make](#), [ilib_gen_loader](#), [ilib_for_link](#), [api_scilab](#)

Name

`ilib_gen_loader` — utility for `ilib_build`: generates a loader file

```
ilib_gen_loader(name, table, [libs])
```

Parameters

`name`

a character string, the generic name of the library without path and extension.

`table`

2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

`libs`

a string matrix, externals dynamic libraries filenames to load by loader file (optional).

Description

Utility function used by `ilib_build` This function generates a loader file.

See Also

`addinter` , `link` , `ilib_build` , `ilib_compile` , `ilib_gen_Make` , `ilib_gen_loader` , `ilib_for_link` , `api_scilab`

Name

ilib_mex_build — utility for mex library management

```
ilib_mex_build(lib_name, table, files, libs [,makename, ldflags, cflags, fflags, cc])
```

Parameters

`lib_name`

a character string, the generic name of the library without path and extension.

`table`

3 column string matrix giving the table of 'scilab-name', 'interface name', 'cmex' or 'fmex'

`files`

string matrix giving objects files needed for shared library creation

`libs`

string matrix giving extra libraries needed for shared library creation

`makename`

character string. The path of the Makefile file without extension.

This parameter is useless since Scilab 5.0. Default value to use: [].

A warning will be displayed in Scilab 5.3 if you use another value than the default.

`ldflags, cflags, fflags, cc`

character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. `cc` provides the name of the compiler.

Description

This function is used to create mex libraries and to generate a loader file which can be used to dynamically load the mex shared library.

Note that the file name containing the mex code can be set in the third input argument (`files`) or the second value of the `table` input argument.

Note that a compiler must be available on the system to use this function.

Examples

```
cd(TMPDIR);

mputl(['#include "mex.h"'
      'void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])'
      '{'
      '    int *dims = mxGetDimensions(prhs[0]);'
      '    sciprint("%d %d %d\n", dims[0], dims[1], dims[2]);'
      '}'], 'mexfunction16.c');
ilib_mex_build('libmex', ['mexf16', 'mexfunction16', 'cmex'], [], [], 'Makelib', '', ''

exec(TMPDIR+'/loader.sce');
mexf16(rand(2,3,2));
```

See Also

[addinter](#) , [link](#) , [ilib_compile](#) , [ilib_gen_Make](#) , [ilib_gen_gateway](#) , [ilib_gen_loader](#) , [ilib_for_link](#) , [api_scilab](#)

Name

ilib_verbose — set level of display used by dynamic link functions.

```
level = ilib_verbose()
ilib_verbose(level)
```

Parameters

level :

level of verbose for dynamic link functions.

0 : no message

1 : default level (as previous version of scilab)

2 : maximum verbose level (configure , makefile, debug information, ...)

Description

"ilib_verbose" set level of display used by dynamic link functions.

All dynamic functions in dynamic link module check this value and display or not some informations.

Examples

```
if haveacompile() then
  cur_verbose = ilib_verbose();
  ilib_verbose(0);

  chdir(TMPDIR);
  f1=['int ext1c(int *n, double *a, double *b, double *c)'
      '{int k;'
      '  for (k = 0; k < *n; ++k) '
      '    c[k] = a[k] + b[k];'
      '  return(0);}'];

  mputl(f1, 'fun1.c');

  ilib_for_link('ext1c', 'fun1.c', [], "c");
  exec loader.sce;

  //using the new primitive
  a=[1,2,3];b=[4,5,6];n=3;
  c = call('ext1c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
  if norm(c-(a+b)) > %eps then pause,end

  ilib_verbose(1);

  f2=['int ext2c(int *n, double *a, double *b, double *c)'
      '{int k;'
      '  for (k = 0; k < *n; ++k) '
      '    c[k] = a[k] + b[k];'
      '  return(0);}'];
```

```
mputl(f2, 'fun2.c');

ilib_for_link('ext2c', 'fun2.c', [], "c")
exec loader.sce;

//using the new primitive
a = [1,2,3]; b = [4,5,6]; n = 3;
c = call('ext2c', n, 1, 'i', a, 2, 'd', b, 3, 'd', 'out', [1,3], 4, 'd');
if norm(c-(a+b)) > %eps then pause, end

ilib_verbose(cur_verbose);

end
```

See Also

mode, link, ilib_compile, ilib_build, ilib_for_link, api_scilab

Authors

Allan CORNET

Name

link — dynamic linker

```
x=link(files [, sub-names, flag]);  
link(x , sub-names [, flag]);  
ulink(x)  
lst=link('show')  
lst=link()
```

Parameters

files

a character string or a vector of character strings, the files names used to define the new entry point (compiled routines, user libraries, system libraries,...)

sub-names

a character string or a vector of character strings . Name of the entry points in `files` to be linked.

x

an integer which gives the id of a shared library linked into Scilab with a previous call to `link`.

flag

character string 'f' or 'c' for Fortran (default) or C code.

Description

`link` is a incremental/dynamic link facility: this command allows to add new compiled Fortran or C routines to Scilab executable code. Linked routines can be called interactively by the function `call`. Linked routines can also be used as "external" for e.g. non linear problem solvers (`ode`, `optim`, `intg`, `dassl`...).

`link()` returns a string matrix with linked functions.

a call to `link` returns an integer which gives the id of the shared library which is loaded into Scilab. This number can then be used as the first argument of the `link` function in order to link additional function from the linked shared library. The shared library is removed with the `ulink` command.

A routine can be unlinked with `ulink`. If the linked function has been modified between two links, it is required to `ulink` the previous instance before the new link.

`link('show')` returns the current linked routines.

To be able to link routines in a system independent way, it is convenient to use the `ilib_for_link` utility function instead of `link`.

(Experienced) users may also `link` a new Scilab interface routine to add a set of new functions. See `ilib_build` and `addinter` functions.

Number of 'link' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

Examples

```
//Example of the use of ilib_for_link with a simple C code  
cd TMPDIR  
f1=['#include <math.h>'
```

```
'void fooc(double c[],double a[],double *b,int *m,int *n)'
'{
'    int i;'
'    for ( i =0 ; i < (*m)*(*n) ; i++) '
'        c[i] = sin(a[i]) + *b; '
'}';

mputl(f1,'fooc.c');

//creating the shared library: a Makefile and a loader are
//generated, the code is compiled and a shared library built.
ilib_for_link('fooc','fooc.c',[],"c")

// display the loader.sce file which calls link
mprintf('%s\n',mgetl('loader.sce'))
// load the shared library
exec loader.sce;

link('show')
// call the new linked entry point
a=linspace(0,%pi,10);b=5;
y1=call('fooc',a,2,'d',b,3,'d',size(a,1),4,'i',size(a,2),5,'i','out',size(a),1,
// check
y1-(sin(a)+b)
exec cleaner.sce;
```

See Also

[call](#), [external](#), [c_link](#), [addinter](#), [ilib_for_link](#), [ilib_build](#), [api_scilab](#)

Name

ulink — unlink a dynamically linked shared object

```
ulink(x)
ulink()
```

Description

see `link`

If you plan to use `valgrind` to profile your toolbox, you must first set the environment variable `PROFILE_SCILAB_DYNAMIC_LINK` before starting `scilab`:

```
# under bash shell:
export PROFILE_SCILAB_DYNAMIC_LINK=1
export SCILAB_VALGRIND_OPT="--db-attach=no --show-below-main=yes --log-fd=2 --"
scilab -nwni -profile
```

This environment variable force Scilab not to release the loaded dynamic libraries. This will allow `valgrind` to perform its sum-up analysis.

See Also

`link`

Name

`with_lcc` — returns if LCC-Win32 is the default C Compiler.

```
bOK=with_lcc()
```

Parameters

`bOK`

returns %T if LCC-Win32 is the default C Compiler.

Description

checks if LCC-Win32 is the default C Compiler.

Examples

```
bOK=with_lcc()
```

Authors

Allan CORNET

Part XVII. Integers

Name

iconvert — conversion to 1 or 4 byte integer representation

```
y=iconvert(X, itype)
```

Parameters

X
matrix of floats or integers

y
matrix of integers coded on one, two or four bytes.

Description

converts and stores data two one, two or four bytes integers.

itype=0
return floating point numbers

itype=1
return int8 numbers in the range [-128,127]

itype=11
return uint8 numbers in the range [0,255]

itype=2
return int16 numbers in the range [-32768,32767]

itype=12
return uint16 numbers in the range [0, 65535]

itype=4
return int32 numbers in the range [-2147483648,2147483647]

itype=14
return uint32 numbers in the range [0, 4294967295]

Examples

```
b=int32([1 -120 127 312])  
y=iconvert(b,1)
```

See Also

double , inttype

Name

int8 — conversion to one byte integer representation
int16 — conversion to 2 bytes integer representation
int32 — conversion to 4 bytes integer representation
uint8 — conversion to one byte unsigned integer representation
uint16 — conversion to 2 bytes unsigned integer representation
uint32 — conversion to 4 bytes unsigned integer representation

```
y=int8(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

Parameters

X
matrix of floats or integers

y
matrix of integers coded on one, two or four bytes.

Description

converts and stores data two one, two or four bytes integers. These data types are specially useful to store big objects such as images, long signals,...

y=int8(X)
return numbers in the range [-128,127]

y=uint8(X)
return numbers in the range [0,255]

y=int16(X)
return numbers in the range [-32768,32767]

y=uint16(X)
return numbers in the range [0, 65535]

y=int32(X)
return numbers in the range [-2147483648,2147483647]

y=uint32(X)
return numbers in the range [0, 4294967295]

Examples

```
int8([1 -120 127 312])
uint8([1 -120 127 312])

x=int32(-200:100:400)
int8(x)
```

See Also

double , inttype , iconvert

Name

inttype — type integers used in integer data types

```
[i]=inttype(x)
```

Parameters

x
an matrix of integers (see int8,...)

i
integer

Description

inttype(x) returns an integer which is the type of the entries of x as following :

1 : one byte integer representation
2 : two bytes integer representation
4 : four bytes integer representation
11 : one byte unsigned integer representation
12 : two bytes unsigned integer representation
14 : four bytes unsigned integer representation

Examples

```
x=uint16(1:10);  
inttype(x)
```

See Also

int8

Part XVIII. Interpolation

Name

bsplin3val — 3d spline arbitrary derivative evaluation function

```
[dfp]=bsplin3val(xp,yp,zp,tl,der)
```

Parameters

xp, yp, zp

real vectors or matrices of same size

tl

tlst of type "splin3d", defining a 3d tensor spline (called *s* in the following)

der

vector with 3 components [*ox*, *oy*, *oz*] defining which derivative of *s* to compute.

dfp

vector or matrix of same format than *xp*, *yp* and *zp*, elementwise evaluation of the specified derivative of *s* on these points.

Description

While the function `interp3d` may compute only the spline *s* and its first derivatives, `bsplin3val` may compute any derivative of *s*. The derivative to compute is specified by the argument `der=[ox, oy, oz]` :

$$dfp(i) = \frac{\partial^{ox \times oy \times oz}}{\partial x^{ox} \partial y^{oy} \partial z^{oz}} s(xp(i), yp(i), zp(i))$$

So `der=[0 0 0]` corresponds to *s*, `der=[1 0 0]` to *ds/dx*, `der=[0 1 0]` to *ds/dy*, `der=[1 1 0]` to *d2s/dxdy*, etc...

For a point with coordinates (*xp(i)*,*yp(i)*,*zp(i)*) outside the grid, the function returns 0.

Examples

```
deff("v=f(x,y,z)", "v=cos(x).*sin(y).*cos(z)");
deff("v=fx(x,y,z)", "v=-sin(x).*sin(y).*cos(z)");
deff("v=fxy(x,y,z)", "v=-sin(x).*cos(y).*cos(z)");
deff("v=fxyz(x,y,z)", "v=sin(x).*cos(y).*sin(z)");
deff("v=fxyz(x,y,z)", "v=cos(x).*cos(y).*sin(z)");
n = 20; // n x n x n interpolation points
x = linspace(0,2*pi,n); y=x; z=x; // interpolation grid
[X,Y,Z] = ndgrid(x,y,z); V = f(X,Y,Z);
tl = splin3d(x,y,z,V,[5 5 5]);

// compute f and some derivates on a point
// and compare with the spline interpolant
xp = grand(1,1,"unf",0,2*pi);
yp = grand(1,1,"unf",0,2*pi);
zp = grand(1,1,"unf",0,2*pi);

f_e = f(xp,yp,zp)
f_i = bsplin3val(xp,yp,zp,tl,[0 0 0])
```

```
fx_e = fx(xp,yp,zp)
fx_i = bsplin3val(xp,yp,zp,tl,[1 0 0])

fxy_e = fxy(xp,yp,zp)
fxy_i = bsplin3val(xp,yp,zp,tl,[1 1 0])

fxyz_e = fxyz(xp,yp,zp)
fxyz_i = bsplin3val(xp,yp,zp,tl,[1 1 1])

fxyz_e = fxyz(xp,yp,zp)
fxyz_i = bsplin3val(xp,yp,zp,tl,[2 1 1])
```

See Also

splin3d, interp3d

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Name

cshep2d — bidimensional cubic shepard (scattered) interpolation

```
tl_coef = cshep2d(xyz)
```

Parameters

xyz

a $n \times 3$ matrix of the (no gridded) interpolation points (the i th row given the (x,y) coordinates then the altitude z of the i th interpolation point)

tl_coef

a tlist scilab structure (of type cshep2d)

Description

This function is useful to define a 2d interpolation function when the interpolation points are not on a grid (you may use it in this case but `splin2d` is better for that purpose). The interpolant is a cubic shepard one and is a C2 (twice continuously differentiable) bivariate function $s(x,y)$ such that : $s(x_i,y_i)=z_i$ for all $i=1,...,n$ ((x_i,y_i,z_i) being the i th row of `xyz`).

The evaluation of s at some points must be done by the `eval_cshep2d` function.

Remark

The function works if $n \geq 10$, if the nodes are not all colinears (i.e. the (x,y) coordinates of the interpolation points are not on the same straight line), and if there is no duplicate nodes (i.e. 2 or more interpolation points with the same (x,y) coordinates). An error is issued if these conditions are not respected.

Examples

```
// interpolation of cos(x)cos(y) with randomly choosen interpolation points
n = 150; // nb of interpolation points
xy = grand(n,2,"unf",0,2*pi);
z = cos(xy(:,1)).*cos(xy(:,2));
xyz = [xy z];
tl_coef = cshep2d(xyz);

// evaluation on a grid
m = 30;
xx = linspace(0,2*pi,m);
[X,Y] = ndgrid(xx,xx);
Z = eval_cshep2d(X,Y, tl_coef);
clf()
plot3d(xx,xx,Z,flag=[2 6 4])
param3d1(xy(:,1),xy(:,2),list(z,-9), flag=[0 0])
xtitle("Cubic Shepard Interpolation of cos(x)cos(y) with randomly choosen inter
legends("interpolation points",-9,1)
xselect()
```

See Also

`splin2d`, `eval_cshep2d`

Authors

Robert J. Renka

B. Pincon (scilab interface)

Name

eval_cshep2d — bidimensional cubic shepard interpolation evaluation

```
[zp [, dzpdx, dzpdy [, d2zpdxx, d2zpdxy, d2zpdyy]]] = eval_cshep2d(xp, yp, tl_coef)
```

Parameters

xp, yp

two real vectors (or matrices) of the same size

tl_coef

a tlist scilab structure (of type cshep2d) defining a cubic Shepard interpolation function (named S in the following)

zp

vector (or matrix) of the same size than xp and yp, evaluation of the interpolant S at these points

dzpdx, dzpdy

vectors (or matrices) of the same size than xp and yp, evaluation of the first derivatives of S at these points

d2zpdxx, d2zpdxy, d2zpdyy

vectors (or matrices) of the same size than xp and yp, evaluation of the second derivatives of S at these points

Description

This is the evaluation routine for cubic Shepard interpolation function computed with cshep2d, that is :

$$\begin{aligned}zp(i) &= S(xp(i), yp(i)) \\dzpdx(i) &= \frac{\partial S}{\partial x}(xp(i), yp(i)) \\dzpdy(i) &= \frac{\partial S}{\partial y}(xp(i), yp(i)) \\d2zpdxx(i) &= \frac{\partial^2 S}{\partial x^2}(xp(i), yp(i)) \\d2zpdxy(i) &= \frac{\partial^2 S}{\partial x \partial y}(xp(i), yp(i)) \\d2zpdyy(i) &= \frac{\partial^2 S}{\partial y^2}(xp(i), yp(i))\end{aligned}$$

Remark

The interpolant S is C2 (twice continuously differentiable) but is also extended by zero for (x,y) far enough the interpolation points. This leads to a discontinuity in a region far outside the interpolation points, and so, is not cumbersome in practice (in a general manner, evaluation outside interpolation points (i.e. extrapolation) leads to very inaccurate results).

Examples


```
// see example section of cshep2d

// this example shows the behavior far from the interpolation points ...
deff("z=f(x,y)", "z = 1+ 50*(x.*(1-x).*y.*(1-y)).^2")
x = linspace(0,1,10);
[X,Y] = ndgrid(x,x);
X = X(:); Y = Y(:); Z = f(X,Y);
S = cshep2d([X Y Z]);
// evaluation inside and outside the square [0,1]x[0,1]
m = 40;
xx = linspace(-1.5,0.5,m);
[xp,yp] = ndgrid(xx,xx);
zp = eval_cshep2d(xp,yp,S);
// compute facet (to draw one color for extrapolation region
// and another one for the interpolation region)
[xf,yf,zf] = genfac3d(xx,xx,zp);
color = 2*ones(1,size(zf,2));
// indices corresponding to facet in the interpolation region
ind=find( mean(xf,"r")>0 & mean(xf,"r")<1 & mean(yf,"r")>0 & mean(yf,"r")<1 );
color(ind)=3;
clf();
plot3d(xf,yf,list(zf,color), flag=[2 6 4])
legends(["extrapolation region","interpolation region"],[2 3],1)
xselect()
```

See Also

[cshep2d](#)

Authors

Robert J. Renka
B. Pincon (scilab interface)

Name

interp — cubic spline evaluation function

```
[yp [, yp1 [, yp2 [, yp3]]]] = interp(xp, x, y, d [, out_mode])
```

Parameters

- xp**
real vector or matrix
- x,y,d**
real vectors of the same size defining a cubic spline or sub-spline function (called s in the following)
- out_mode**
(optional) string defining the evaluation of s outside the $[x_1, x_n]$ interval
- yp**
vector or matrix of same size than xp , elementwise evaluation of s on xp ($yp(i) = s(xp(i))$ or $yp(i,j) = s(xp(i,j))$)
- yp1, yp2, yp3**
vectors (or matrices) of same size than xp , elementwise evaluation of the successive derivatives of s on xp

Description

Given three vectors (x, y, d) defining a spline or sub-spline function (see `splin`) with $y_i = s(x_i)$, $d_i = s'(x_i)$ this function evaluates s (and s' , s'' , s''' if needed) at $xp(i)$:

$$\begin{aligned} yp(i) &= s(xp(i)) \text{ or } yp(i,j) = s(xp(i,j)) \\ yp1(i) &= s'(xp(i)) \text{ or } yp1(i,j) = s'(xp(i,j)) \\ yp2(i) &= s''(xp(i)) \text{ or } yp2(i,j) = s''(xp(i,j)) \\ yp3(i) &= s'''(xp(i)) \text{ or } yp3(i,j) = s'''(xp(i,j)) \end{aligned}$$

The `out_mode` parameter set the evaluation rule for extrapolation, i.e. for $xp(i)$ not in $[x_1, x_n]$:

- "by_zero"
an extrapolation by zero is done
- "by_nan"
extrapolation by Nan
- "C0"
the extrapolation is defined as follows :

$$\begin{aligned} s(x) &= y_1 \text{ for } x < x_1 \\ s(x) &= y_n \text{ for } x > x_n \end{aligned}$$

- "natural"
the extrapolation is defined as follows (p_i being the polynomial defining s on $[x_i, x_{i+1}]$) :

$$s(x) = p_1(x) \text{ for } x < x_1$$

$$s(x) = p_{n-1}(x) \text{ for } x > x_n$$

"linear"

the extrapolation is defined as follows :

$$s(x) = y_1 + s'(x_1) \cdot (x - x_1) \text{ for } x < x_1$$

$$s(x) = y_n + s'(x_n) \cdot (x - x_n) \text{ for } x > x_n$$

"periodic"

s is extended by periodicity.

Examples

```
// see the examples of splin and lsq_splin

// an example showing C2 and C1 continuity of spline and subspline
a = -8; b = 8;
x = linspace(a,b,20)';
y = sinc(x);
dk = splin(x,y); // not_a_knot
df = splin(x,y, "fast");
xx = linspace(a,b,800)';
[yyk, yy1k, yy2k] = interp(xx, x, y, dk);
[yyf, yy1f, yy2f] = interp(xx, x, y, df);
clf()
subplot(3,1,1)
plot2d(xx, [yyk yyf])
plot2d(x, y, style=-9)
legends(["not_a_knot spline","fast sub-spline","interpolation points"],...
        [1 2 -9], "ur",%f)
xlabel("spline interpolation")
subplot(3,1,2)
plot2d(xx, [yy1k yy1f])
legends(["not_a_knot spline","fast sub-spline"], [1 2], "ur",%f)
xlabel("spline interpolation (derivatives)")
subplot(3,1,3)
plot2d(xx, [yy2k yy2f])
legends(["not_a_knot spline","fast sub-spline"], [1 2], "lr",%f)
xlabel("spline interpolation (second derivatives)")

// here is an example showing the different extrapolation possibilities
x = linspace(0,1,11)';
y = cosh(x-0.5);
d = splin(x,y);
xx = linspace(-0.5,1.5,401)';
yy0 = interp(xx,x,y,d,"C0");
yy1 = interp(xx,x,y,d,"linear");
yy2 = interp(xx,x,y,d,"natural");
yy3 = interp(xx,x,y,d,"periodic");
clf()
```

```
plot2d(xx,[yy0 yy1 yy2 yy3],style=2:5,frameflag=2,leg="C0@linear@natural@period.  
xtitle(" different way to evaluate a spline outside its domain")
```

See Also

splin, lsq_splin

Authors

B. Pincon

Name

interp1 — one_dimension interpolation function

```
[yp]=interp1(x, y,xp [, method,[extrapolation]])
```

Parameters

xp
reals scalar, vector or matrix (or hypermatrix)

x
reals vector

method
(optional) string defining the interpolation method

extrapolation
(optional) string, or real value defining the yp(j) components for xp(j) values outside [x1,xn] interval.

yp
vector, or matrix (or hypermatrix)

Description

Given (x,y,xp), this function performs the yp components corresponding to xp by the interpolation(linear by default) defined by x and y.

If yp is a vector then the length of xp is equal to the length of yp, if yp is a matrix then xp have the same length than the length of each columns of yp, if yp is a hypermatrix then the length of xp have the same length than the first dimension of yp.

If $\text{size}(y)=[C,M1,M2,M3,...,Mj]$ and $\text{size}(xp)=[N1,N2,N3,...,Nk]$ then $\text{size}(yp)=[N1,N2,...,Nk,M1,M2,...,Mj]$ and length of x must be equal to $\text{size}(y,1)$

The **method** parameter sets the evaluation rule for interpolation

"linear"
the interpolation is defined by linear method (see interp1n)

"spline"
the interpolation is defined by cubic spline interpolation (see splin , interp)

"nearest"
for each value xp(j), yp(j) takes the value or y(i) corresponding to x(i) the nearest neighbor of xp(j)

The **extrapolation** parameter sets the evaluation rule for extrapolation, i.e for xp(i) not in [x1,xn] interval

"extrap"
the extrapolation is performed by the defined method. $yp=\text{interp1}(x,y,xp,\text{method},\text{"extrap"})$

real value
you can choose a real value for extrapolation, in this way yp(i) takes this value for xp(i) not in [x1,xn] interval, for example 0 (but also nan or inf). $yi=\text{interp1}(x,y,xp,\text{method},0)$

by default
the extrapolation is performed by the defined method (for spline method), and by nan for linear and nearest methods. $yp=\text{interp1}(x,y,xp,\text{method})$

Examples

```
x=linspace(0,3,20);
y=x^2;
xx=linspace(0,3,100);
yy1=interp1(x,y,xx,'linear');
yy2=interp1(x,y,xx,'spline');
yy3=interp1(x,y,xx,'nearest');
plot(xx,[yy1;yy2;yy3],x,y,'*')
xlabel('interpolation of square function')
legend(['linear','spline','nearest'],a=2)
```

See Also

interp, interpln, splin

Authors

F.B

Name

interp2d — bicubic spline (2d) evaluation function

```
[zp[, dzpdx, dzpdy[, d2zpdxx, d2zpdxy, d2zpdyy]]]=interp2d(xp, yp, x, y, C [, out_mode])
```

Parameters

xp, yp

real vectors or matrices of same size

x,y,C

real vectors defining a bicubic spline or sub-spline function (called *s* in the following)

out_mode

(optional) string defining the evaluation of *s* outside $[x(1),x(nx)]x[y(1),y(ny)]$

zp

vector or matrix of same format than *xp* and *yp*, elementwise evaluation of *s* on these points.

dzpdx, dzpdy

vectors (or matrices) of same format than *xp* and *yp*, elementwise evaluation of the first derivatives of *s* on these points.

d2zpdxx, d2zpdxy, d2zpdyy

vectors (or matrices) of same format than *xp* and *yp*, elementwise evaluation of the second derivatives of *s* on these points.

Description

Given three vectors (*x*, *y*, *C*) defining a bicubic spline or sub-spline function (see `splin2d`) this function evaluates *s* (and ds/dx , ds/dy , d^2s/dxx , d^2s/dxy , d^2s/dyy if needed) at $(xp(i),yp(i))$:

$$zp(i) = s(xp(i),yp(i))$$

$$dzpdx(i) = \frac{ds}{dx}(xp(i),yp(i))$$

$$dzpdy(i) = \frac{ds}{dy}(xp(i),yp(i))$$

$$d2zpdxx(i) = \frac{d^2s}{dx^2}(xp(i),yp(i))$$

$$d2zpdxy(i) = \frac{d^2s}{dxdy}(xp(i),yp(i))$$

$$d2zpdyy(i) = \frac{d^2s}{dy^2}(xp(i),yp(i))$$

The `out_mode` parameter defines the evaluation rule for extrapolation, i.e. for $(xp(i),yp(i))$ not in $[x(1),x(nx)]x[y(1),y(ny)]$:

"by_zero"

an extrapolation by zero is done

"by_nan"

extrapolation by Nan

"C0"

the extrapolation is defined as follows :

```
s(x,y) = s(proj(x,y)) where proj(x,y) is nearest point
of [x(1),x(nx)]x[y(1),y(ny)] from (x,y)
```

"natural"

the extrapolation is done by using the nearest bicubic-patch from (x,y).

"periodic"

S is extended by periodicity.

Examples

```
// see the examples of splin2d

// this example shows some different extrapolation features
// interpolation of cos(x)cos(y)
n = 7; // a n x n interpolation grid
x = linspace(0,2*pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");

// now evaluate on a bigger domain than [0,2pi]x [0,2pi]
m = 80; // discretisation parameter of the evaluation grid
xx = linspace(-0.5*pi,2.5*pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz1 = interp2d(XX,YY, x, y, C, "C0");
zz2 = interp2d(XX,YY, x, y, C, "by_zero");
zz3 = interp2d(XX,YY, x, y, C, "periodic");
zz4 = interp2d(XX,YY, x, y, C, "natural");
clf()
subplot(2,2,1)
plot3d(xx, yy, zz1, flag=[2 6 4])
xlabel("extrapolation with the C0 outmode")
subplot(2,2,2)
plot3d(xx, yy, zz2, flag=[2 6 4])
xlabel("extrapolation with the by_zero outmode")
subplot(2,2,3)
plot3d(xx, yy, zz3, flag=[2 6 4])
xlabel("extrapolation with the periodic outmode")
subplot(2,2,4)
plot3d(xx, yy, zz4, flag=[2 6 4])
xlabel("extrapolation with the natural outmode")
xselect()
```

See Also

[splin2d](#)

Authors

B. Pincon

Name

interp3d — 3d spline evaluation function

```
[fp[,dfpdx,dfpdy,dfpdz]]=interp3d(xp,yp,zp,tl,out_mode)
```

Parameters

xp, yp, zp

real vectors or matrices of same size

tl

tlist of type "splin3d", defining a 3d tensor spline (called *s* in the following)

out_mode

(optional) string defining the evaluation of *s* outside the grid
([xmin,xmax]x[ymin,ymax]x[zmin,zmax])

fp

vector or matrix of same format than xp, yp and zp, elementwise evaluation of *s* on these points.

dfpdx, dfpdy, dfpdz

vectors (or matrices) of same format than xp, yp and zp, elementwise evaluation of the first derivatives of *s* on these points.

Description

Given a tlist *tl* defining a 3d spline function (see *splin3d*) this function evaluates *s* (and *ds/dx*, *ds/dy*, *ds/dz* if needed) at $(xp(i),yp(i),zp(i))$:

$$\begin{aligned}zp(i) &= s(xp(i),yp(i)) \\ dzpdx(i) &= \frac{ds}{dx}(xp(i),yp(i),zp(i)) \\ dzpdy(i) &= \frac{ds}{dy}(xp(i),yp(i),zp(i)) \\ dzpdz(i) &= \frac{ds}{dz}(xp(i),yp(i),zp(i))\end{aligned}$$

The *out_mode* parameter defines the evaluation rule for extrapolation, i.e. for $(xp(i),yp(i),zp(i))$ not in $[xmin,xmax] \times [ymin,ymax] \times [zmin,zmax]$:

"by_zero"

an extrapolation by zero is done

"by_nan"

extrapolation by Nan

"C0"

the extrapolation is defined as follows :

```
s(x,y) = s(proj(x,y)) where proj(x,y) is nearest point
of [x(1),x(nx)]x[y(1),y(ny)] from (x,y)
```

"periodic"
S is extended by periodicity.

Examples

```
// see the examples of the splin3d help page
```

See Also

splin3d, bsplin3val

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Name

interp1n — linear interpolation

```
[y]=interp1n(xyd,x)
```

Parameters

xyd
2 row matrix (xy coordinates of points)

x
vector (abscissae)

y
vector (y-axis values)

Description

given xyd a set of points in the xy-plane which increasing abscissae and x a set of abscissae, this function computes y the corresponding y-axis values by linear interpolation.

Examples

```
x=[1 10 20 30 40];  
y=[1 30 -10 20 40];  
plot2d(x',y',[-3],"011"," ",[-10,-40,50,50]);  
yi=interp1n([x;y],[-4:45]);  
plot2d((-4:45)',yi',[3],"000");
```

See Also

splin, interp, smooth

Name

intsplin — integration of experimental data by spline interpolation

```
v = intsplin([x,] s)
```

Parameters

x
vector of increasing x coordinate data. Default value is `1:size(y, 'r')`

s
vector of y coordinate data

v
value of the integral

Description

computes :

Where f is a function described by a set of experimental value:

$s(i)=f(x(i))$ and $x0=x(1)$, $x1=x(n)$

Between mesh points function is interpolated using spline's.

Examples

```
t=0:0.1:%pi  
intsplin(t, sin(t))
```

See Also

intg, integrate, inttrap, splin

Name

linear_interpn — n dimensional linear interpolation

```
vp = linear_interpn(xp1,xp2,..,xpn, x1, ..., xn, v [,out_mode])
```

Parameters

xp1, xp2, .., xpn

real vectors (or matrices) of same size

x1 ,x2, ..., xn

strictly increasing row vectors (with at least 2 components) defining the n dimensional interpolation grid

v

vector (case n=1), matrix (case n=2) or hypermatrix (case n > 2) with the values of the underlying interpolated function at the grid points.

out_mode

(optional) string defining the evaluation outside the grid (extrapolation)

vp

vector or matrix of same size than xp1, ..., xpn

Description

Given a n dimensional grid defined by the n vectors $x_1, x_2, \dots, x_n$ and the values v of a function (says f) at the grid points :

$$v(i_1, i_2, \dots, i_n) = f(x_1(i_1), x_2(i_2), \dots, x_n(i_n))$$

this function computes the linear interpolant of f from the grid (called s in the following) at the points which coordinates are defined by the vectors (or matrices) $xp_1, xp_2, \dots, xpn$:

$$vp(i) = s(xp_1(i), xp_2(i), \dots, xpn(i))$$

or $vp(i, j) = s(xp_1(i, j), xp_2(i, j), \dots, xpn(i, j))$ in case the xpk are matrices

The `out_mode` parameter set the evaluation rule for extrapolation: if we note $P_i = (xp_1(i), xp_2(i), \dots, xpn(i))$ then `out_mode` defines the evaluation rule when:

$$P(i) \notin [x_1(1), x_1(N)] \times [x_2(1), x_2(N)] \times \dots \times [x_n(1), x_n(N)]$$

The different choices are:

"by_zero"

an extrapolation by zero is done

"by_nan"

extrapolation by Nan

"C0"

the extrapolation is defined as follows:

```
s(P) = s(proj(P)) where proj(P) is nearest point from P
located on the grid boundary.
```

"natural"

the extrapolation is done by using the nearest n-linear patch from the point.

"periodic"

s is extended by periodicity.

Examples

```
// example 1 : 1d linear interpolation
x = linspace(0,2*pi,11);
y = sin(x);
xx = linspace(-2*pi,4*pi,400)';
yy = linear_interpn(xx, x, y, "periodic");
clf()
plot2d(xx,yy,style=2)
plot2d(x,y,style=-9, strf="000")
xtitle("linear interpolation of sin(x) with 11 interpolation points")

// example 2 : bilinear interpolation
n = 8;
x = linspace(0,2*pi,n); y = x;
z = 2*sin(x')*sin(y);
xx = linspace(0,2*pi, 40);
[xp,yp] = ndgrid(xx,xx);
zp = linear_interpn(xp,yp, x, y, z);
clf()
plot3d(xx, xx, zp, flag=[2 6 4])
[xg,yg] = ndgrid(x,x);
param3d1(xg,yg, list(z,-9*ones(1,n)), flag=[0 0])
xtitle("Bilinear interpolation of 2sin(x)sin(y)")
legends("interpolation points",-9,1)
xselect()

// example 3 : bilinear interpolation and experimentation
// with all the outmode features
nx = 20; ny = 30;
x = linspace(0,1,nx);
y = linspace(0,2, ny);
[X,Y] = ndgrid(x,y);
z = 0.4*cos(2*pi*X).*cos(pi*Y);
npx = 60 ; nyp = 120;
xp = linspace(-0.5,1.5, npx);
yp = linspace(-0.5,2.5, nyp);
[XP,YP] = ndgrid(xp,yp);
zp1 = linear_interpn(XP, YP, x, y, z, "natural");
zp2 = linear_interpn(XP, YP, x, y, z, "periodic");
zp3 = linear_interpn(XP, YP, x, y, z, "C0");
zp4 = linear_interpn(XP, YP, x, y, z, "by_zero");
zp5 = linear_interpn(XP, YP, x, y, z, "by_nan");
clf()
subplot(2,3,1)
plot3d(x, y, z, leg="x@y@z", flag = [2 4 4])
xtitle("initial function 0.4 cos(2 pi x) cos(pi y)")
subplot(2,3,2)
plot3d(xp, yp, zp1, leg="x@y@z", flag = [2 4 4])
xtitle("Natural")
subplot(2,3,3)
```

```

plot3d(xp, yp, zp2, leg="x@y@z", flag = [2 4 4])
xlabel("Periodic")
subplot(2,3,4)
plot3d(xp, yp, zp3, leg="x@y@z", flag = [2 4 4])
xlabel("C0")
subplot(2,3,5)
plot3d(xp, yp, zp4, leg="x@y@z", flag = [2 4 4])
xlabel("by_zero")
subplot(2,3,6)
plot3d(xp, yp, zp5, leg="x@y@z", flag = [2 4 4])
xlabel("by_nan")
xselect()

// example 4 : trilinear interpolation (see splin3d help
//           page which have the same example with
//           tricubic spline interpolation)
exec("SCI/demos/interp/interp_demo.sci")
func = "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)", func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
// compute (and display) the linear interpolant on some slices
m = 41;
dir = ["z=" "z=" "z=" "x=" "y="];
val = [ 0.1  0.5  0.9  0.5  0.5];
ebox = [0 1 0 1 0 1];

XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
    [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
    Vm = linear_interpn(Xm,Ym,Zm, x, y, z, V);
    [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
    Vp = linear_interpn(Xp,Yp,Zp, x, y, z, V);
    [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF, linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
clf()
xset("hidden3d",xget("background"))
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xlabel("tri-linear interpolation of "+func)
xselect()

```

See Also

interpIn, splin, splin2d, splin3d

Authors

B. Pincon

Name

lsq_splin — weighted least squares cubic spline fitting

```
[y, d] = lsq_splin(xd, yd [, wd], x)
```

Parameters

xd, yd

vectors of the same size, datas to be fitted by a cubic spline

wd

(optional) a vector of same format than xd and yd, weights of the least square fit.

x

a strictly increasing (row or column) vector, breakpoints of the cubic spline

y, d

vectors of same format than x, the triplet (x,y,d) defines the approximated cubic spline.

Description

This function computes an approximated cubic spline s for the datas xd , yd , wd (in the following m is supposed to be the length of these vectors) and from a choice of the spline breakpoints x (for instance if you want n breakpoints uniformly choosen you may use $x = \text{linspace}(\min(xd), \max(xd), n)$). If S is the space of all cubic splines functions with breakpoints $x_1 < x_2 < \dots < x_n$ then the resulting spline s is such that:

$$\sum_{k=1}^m wd(k)(s(xd(k)) - yd(k))^2 \leftarrow \sum_{k=1}^m wd(k)(f(xd(k)) - yd(k))^2$$

for all f in S , i.e. realizes the minimum of the sum of the squared errors over all functions of S .

The spline s is completely defined by the triplet (x, y, d) (y and d are the vectors of the spline ordinates and first derivatives at the x_i 's : $y_i = s(x_i)$ and $d_i = s'(x_i)$) and its evaluation at some points must be done by the `interp` function.

Remarks

When wd is not given, all the points have the same weight 1.

A point $(xd(k), yd(k))$ is considered in the fit if $xd(k)$ in $[x_1, x_n]$ and $wd(k) > 0$. In particular you can put a null (or even negative) weight to all data points you want to ignore in the fitting. When the total number of points taken into account in the fit procedure is (strictly) less than 4 an error is issued.

The vector xd do not need to be in increasing order.

Depending on the number and on the positions of the $xd(k)$'s and on the choice of the $x(i)$'s there may be several solutions but only one is selected. When this occurs a warning message is displayed in the Scilab command window. This function is intended to be used when m is much larger than n and in this case no such problem may occurred.

Examples

```
// this is an artificial example where the datas xd and yd  
// are build from a perturbed sin function
```



```
a = 0; b = 2*pi;
sigma = 0.1; // standard deviation of the gaussian noise
m = 200; // number of experimental points
xd = linspace(a,b,m)';
yd = sin(xd) + grand(xd,"nor",0,sigma);

n = 6; // number of breakpoints
x = linspace(a,b,n)';

// compute the spline
[y, d] = lsq_splin(xd, yd, x); // use equal weights

// plotting
ye = sin(xd);
ys = interp(xd, x, y, d);
clf()
plot2d(xd,[ye yd ys],style=[2 -2 3], ...
        leg="exact function@experimental measures (gaussian perturbation)@fitted")
xtitle("a least square spline")
xselect()
```

See Also

interp, splin

Authors

C. De Boor, A.H. Morris (code from the NSWC fortran lib)
B. Pincon (scilab interface and slight modifications)

Name

smooth — smoothing by spline functions

```
[pt]=smooth(ptd [,step])
```

Parameters

ptd
(2xn) real vector

step
real (discretization step of abscissae)

pt
(2xn) real vector

Description

this function computes interpolation by spline functions for a given set of points in the plane. The coordinates are $(ptd(1,i), ptd(2,i))$. The components $ptd(1,:)$ must be in ascending order. The default value for the step is $abs(maxi(ptd(1,:))-mini(ptd(1,:)))/100$

Examples

```
x=[1 10 20 30 40];  
y=[1 30 -10 20 40];  
plot2d(x',y',[3],"011"," ",[-10,-40,50,50]);  
yi=smooth([x;y],0.1);  
plot2d(yi(1,:)',yi(2,:)',[1],"000");
```

See Also

splin, interp, interpln

Name

splin — cubic spline interpolation

```
d = splin(x, y [, spline_type [, der]])
```

Parameters

x

a strictly increasing (row or column) vector (x must have at least 2 components)

y

a vector of same format than x

spline_type

(optional) a string selecting the kind of spline to compute

der

(optional) a vector with 2 components, with the end points derivatives (to provide when spline_type="clamped")

d

vector of the same format than x (d_i is the derivative of the spline at x_i)

Description

This function computes a cubic spline or sub-spline s which interpolates the (x_i, y_i) points, ie, we have $s(x_i) = y_i$ for all $i = 1, \dots, n$. The resulting spline s is completely defined by the triplet (x, y, d) where d is the vector with the derivatives at the x_i : $s'(x_i) = d_i$ (this is called the *Hermite* form). The evaluation of the spline at some points must be done by the interp function. Several kind of splines may be computed by selecting the appropriate spline_type parameter:

"not_a_knot"

this is the default case, the cubic spline is computed by using the following conditions (considering n points $x_1, \dots, x_n$):

$$\begin{aligned} s'''(x_2^-) &= s'''(x_2^+) \\ s'''(x_{n-1}^-) &= s'''(x_{n-1}^+) \end{aligned}$$

"clamped"

in this case the cubic spline is computed by using the end points derivatives which must be provided as the last argument der:

$$\begin{aligned} s'(x_1) &= \text{der}(1) \\ s'(x_n) &= \text{der}(2) \end{aligned}$$

"natural"

the cubic spline is computed by using the conditions:

$$\begin{aligned} s'(x_1) &= 0 \\ s'(x_n) &= 0 \end{aligned}$$

"periodic"

a periodic cubic spline is computed (y must verify $y_1=y_n$) by using the conditions:

$$\begin{aligned}s'(x_1) &= s'(x_n) \\ s''(x_1) &= s''(x_n)\end{aligned}$$

"monotone"

in this case a sub-spline (s is only one continuously differentiable) is computed by using a local scheme for the d_i such that s is monotone on each interval:

$$\begin{aligned}\text{if } y(i) \leq y(i+1) & \text{ } s \text{ is increasing on } [x(i), x(i+1)] \\ \text{if } y(i) \geq y(i+1) & \text{ } s \text{ is decreasing on } [x(i), x(i+1)]\end{aligned}$$

"fast"

in this case a sub-spline is also computed by using a simple local scheme for the d_i : $d(i)$ is the derivative at $x(i)$ of the interpolation polynomial of $(x(i-1), y(i-1)), (x(i), y(i)), (x(i+1), y(i+1))$, except for the end points (d_1 being computed from the 3 left most points and d_n from the 3 right most points).

"fast_periodic"

same as before but use also a centered formula for $d_1 = s'(x_1) = d_n = s'(x_n)$ by using the periodicity of the underlying function (y must verify $y_1=y_n$).

Remarks

From an accuracy point of view use essentially the **clamped** type if you know the end point derivatives, else use **not_a_knot**. But if the underlying approximated function is periodic use the **periodic** type. Under the good assumptions these kind of splines got an $O(h^4)$ asymptotic behavior of the error. Don't use the **natural** type unless the underlying function have zero second end points derivatives.

The **monotone**, **fast** (or **fast_periodic**) type may be useful in some cases, for instance to limit oscillations (these kind of sub-splines have an $O(h^3)$ asymptotic behavior of the error).

If $n=2$ (and `spline_type` is not **clamped**) linear interpolation is used. If $n=3$ and `spline_type` is **not_a_knot**, then a **fast** sub-spline type is in fact computed.

Examples

```
// example 1
deff("y=runge(x)", "y=1./(1 + x.^2)")
a = -5; b = 5; n = 11; m = 400;
x = linspace(a, b, n)';
y = runge(x);
d = spline(x, y);
xx = linspace(a, b, m)';
yyi = interp(xx, x, y, d);
yye = runge(xx);
clf()
plot2d(xx, [yyi yye], style=[2 5], leg="interpolation spline@exact function")
plot2d(x, y, -9)
xlabel("interpolation of the Runge function")

// example 2 : show behavior of different splines on random datas
```

```
a = 0; b = 1;           // interval of interpolation
n = 10;                 // nb of interpolation points
m = 800;                // discretisation for evaluation
x = linspace(a,b,n)'; // abscissae of interpolation points
y = rand(x);            // ordinates of interpolation points
xx = linspace(a,b,m)';
yk = interp(xx, x, y, splin(x,y,"not_a_knot"));
yf = interp(xx, x, y, splin(x,y,"fast"));
ym = interp(xx, x, y, splin(x,y,"monotone"));
clf()
plot2d(xx, [yf ym yk], style=[5 2 3], strf="121", ...
        leg="fast@monotone@not a knot spline")
plot2d(x,y,-9, strf="000") // to show interpolation points
xlabel("Various spline and sub-splines on random datas")
xselect()
```

See Also

[interp](#), [lsq_splin](#)

Authors

B. Pincon

F. N. Fritsch (pchim.f Slatec routine is used for monotone interpolation)

Name

splin2d — bicubic spline gridded 2d interpolation

```
C = splin2d(x, y, z, [, spline_type])
```

Parameters

- x,y**
strictly increasing row vectors (with at least 2 components) defining the interpolation grid
- z**
nx x ny matrix (nx being the length of x and ny the length of y)
- spline_type**
(optional) a string selecting the kind of bicubic spline to compute
- C**
a big vector with the coefficients of the bicubic patches (see details in Remarks)

Description

This function computes a bicubic spline or sub-spline s which interpolates the (x_i, y_j, z_{ij}) points, ie, we have $s(x_i, y_j) = z_{ij}$ for all $i = 1, \dots, nx$ and $j = 1, \dots, ny$. The resulting spline s is defined by the triplet (x, y, C) where C is the vector (of length $16(nx-1)(ny-1)$) with the coefficients of each of the $(nx-1)(ny-1)$ bicubic patches : on $[x(i) \ x(i+1)] \times [y(j) \ y(j+1)]$, s is defined by :

$$s(x, y) = \sum_{k=1}^4 \sum_{l=1}^4 c_{ij}(k, l) \cdot (x - x_i)^{k-1} \cdot (y - y_j)^{l-1}$$

The evaluation of s at some points must be done by the `interp2d` function. Several kind of splines may be computed by selecting the appropriate `spline_type` parameter. The method used to compute the bicubic spline (or sub-spline) is the old fashioned one 's, i.e. to compute on each grid point (x_i, y_j) an approximation of the first derivatives $ds/dx(x_i, y_j)$ and $ds/dy(x_i, y_j)$ and of the cross derivative $d^2s/dxdy(x_i, y_j)$. Those derivatives are computed by the mean of 1d spline schemes leading to a C2 function (s is twice continuously differentiable) or by the mean of a local approximation scheme leading to a C1 function only. This scheme is selected with the `spline_type` parameter (see `splin` for details) :

"not_a_knot"
this is the default case.

"periodic"
to use if the underlying function is periodic : you must have $z(1, j) = z(nx, j)$ for all j in $[1, ny]$ and $z(i, 1) = z(i, ny)$ for i in $[1, nx]$ but this is not verified by the interface.

Remarks

From an accuracy point of view use essentially the **not_a_knot** type or **periodic** type if the underlying interpolated function is periodic.

The **natural**, **monotone**, **fast** (or **fast_periodic**) type may be useful in some cases, for instance to limit oscillations (**monotone** being the most powerfull for that).

To get the coefficients of the bi-cubic patches in a more friendly way you can use `C = hypermat([4, 4, nx-1, ny-1], C)` then the coefficient (k, l) of the patch (i, j) (see equation here

before) is stored at $C(k, l, i, j)$. Nevertheless the `interp2d` function wait for the big vector C and not for the hypermatrix C (note that one can easily retrieve C from C with $C=C(:)$).

Examples

```
// example 1 : interpolation of cos(x)cos(y)
n = 7; // a regular grid with n x n interpolation points
      // will be used
x = linspace(0,2*pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");
m = 50; // discretisation parameter of the evaluation grid
xx = linspace(0,2*pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz = interp2d(XX,YY, x, y, C);
emax = max(abs(zz - cos(xx')*cos(yy)));
clf()
plot3d(xx, yy, zz, flag=[2 4 4])
[X,Y] = ndgrid(x,y);
param3d1(X,Y,list(z,-9*ones(1,n)), flag=[0 0])
str = sprintf(" with %d x %d interpolation points. ermax = %g",n,n,emax)
xlabel("spline interpolation of cos(x)cos(y)" + str)

// example 2 : different interpolation functions on random datas
n = 6;
x = linspace(0,1,n); y = x;
z = rand(n,n);
np = 50;
xp = linspace(0,1,np); yp = xp;
[XP, YP] = ndgrid(xp,yp);
ZP1 = interp2d(XP, YP, x, y, splin2d(x, y, z, "not_a_knot"));
ZP2 = linear_interpn(XP, YP, x, y, z);
ZP3 = interp2d(XP, YP, x, y, splin2d(x, y, z, "natural"));
ZP4 = interp2d(XP, YP, x, y, splin2d(x, y, z, "monotone"));
xset("colormap", jetcolormap(64))
clf()
subplot(2,2,1)
plot3d1(xp, yp, ZP1, flag=[2 2 4])
xlabel("not_a_knot")
subplot(2,2,2)
plot3d1(xp, yp, ZP2, flag=[2 2 4])
xlabel("bilinear interpolation")
subplot(2,2,3)
plot3d1(xp, yp, ZP3, flag=[2 2 4])
xlabel("natural")
subplot(2,2,4)
plot3d1(xp, yp, ZP4, flag=[2 2 4])
xlabel("monotone")
xselect()

// example 3 : not_a_knot spline and monotone sub-spline
//              on a step function
a = 0; b = 1; c = 0.25; d = 0.75;
// create interpolation grid
n = 11;
x = linspace(a,b,n);
ind = find(c <= x & x <= d);
```

```
z = zeros(n,n); z(ind,ind) = 1; // a step inside a square
// create evaluation grid
np = 220;
xp = linspace(a,b, np);
[XP, YP] = ndgrid(xp, xp);
zp1 = interp2d(XP, YP, x, x, splin2d(x,x,z));
zp2 = interp2d(XP, YP, x, x, splin2d(x,x,z,"monotone"));
// plot
clf()
xset("colormap",jetcolormap(128))
subplot(1,2,1)
plot3d1(xp, xp, zp1, flag=[-2 6 4])
xlabel("spline (not_a_knot)")
subplot(1,2,2)
plot3d1(xp, xp, zp2, flag=[-2 6 4])
xlabel("subspline (monotone)")
```

See Also

[cshep2d](#), [linear_interpn](#), [interp2d](#)

Authors

B. Pincon

Name

splin3d — spline gridded 3d interpolation

```
tl = splin3d(x, y, z, v, [order])
```

Parameters

x,y,z

strictly increasing row vectors (each with at least 3 components) defining the 3d interpolation grid

v

$n_x \times n_y \times n_z$ hypermatrix (n_x, n_y, n_z being the length of *x*, *y* and *z*)

order

(optional) a 1x3 vector [*kx*,*ky*,*kz*] given the order of the tensor spline in each direction (default [4,4,4], i.e. tricubic spline)

tl

a list of type splin3d defining the spline

Description

This function computes a 3d tensor spline *s* which interpolates the (x_i, y_j, z_k, v_{ijk}) points, ie, we have $s(x_i, y_j, z_k) = v_{ijk}$ for all $i=1, \dots, n_x, j=1, \dots, n_y$ and $k=1, \dots, n_z$. The resulting spline *s* is defined by *tl* which consists in a B-spline-tensor representation of *s*. The evaluation of *s* at some points must be done by the interp3d function (to compute *s* and its first derivatives) or by the bsplin3val function (to compute an arbitrary derivative of *s*). Several kind of splines may be computed by selecting the order of the spline in each direction *order*=[*kx*, *ky*, *kz*].

Remark

This function works under the conditions:

$$\begin{aligned}n_x, n_y, n_z &\geq 3 \\ 2 &\leq k_x < n_x \\ 2 &\leq k_y < n_y \\ 2 &\leq k_z < n_z\end{aligned}$$

an error being issued when they are not respected.

Examples

```
// example 1
// =====

func = "v=cos(2*pi*x).*sin(2*pi*y).*cos(2*pi*z)";
deff("v=f(x,y,z)", func);
n = 10; // n x n x n interpolation points
x = linspace(0,1,n); y=x; z=x; // interpolation grid
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
```

```

tl = splin3d(x,y,z,V,[5 5 5]);
m = 10000;
// compute an approximated error
xp = grand(m,1,"def"); yp = grand(m,1,"def"); zp = grand(m,1,"def");
vp_exact = f(xp,yp,zp);
vp_interp = interp3d(xp,yp,zp, tl);
er = max(abs(vp_exact - vp_interp))
// now retry with n=20 and see the error

// example 2 (see linear_interpn help page which have the
// same example with trilinear interpolation)
// =====

exec("SCI/modules/interpolation/demos/interp_demo.sci")
func = "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)",func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
tl = splin3d(x,y,z,V);
// compute (and display) the 3d spline interpolant on some slices
m = 41;
dir = ["z=" "z=" "z=" "x=" "y="];
val = [ 0.1 0.5 0.9 0.5 0.5];
ebox = [0 1 0 1 0 1];
XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
    [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
    Vm = interp3d(Xm,Ym,Zm, tl);
    [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
    Vp = interp3d(Xp,Yp,Zp, tl);
    [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
    XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF, linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
clf(); xset("hidden3d",xget("background"));
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xtitle("3d spline interpolation of "+func)
xselect()

```

See Also

linear_interpn, interp3d, bsplin3val

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Part XIX. Input/Output functions

Name

file — file management

```
[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])  
file(action,unit)  
[units [,typ [,nams [,mod [,swap]]]]] = file([unit])
```

Parameters

file-name

string, file name of the file to be opened

status

string, The status of the file to be opened

"new"

file must not exist new file (default)

"old"

file must already exists.

"unknown"

unknown status

"scratch"

file is to be deleted at end of session

access

string, The type of access to the file

"sequential"

sequential access (default)

"direct"

direct access.

format

string,

"formatted"

for a formatted file (default)

"unformatted"

binary record.

recl

integer, is the size of records in bytes when access="direct"

unit

integer, logical unit descriptor of the opened file

units

integer vector, logical unit descriptor of the opened files. Units 5 and 6 (%io) are reserved by the system for input and output devices.

typs

Character string vector, type (C or Fortran) of opened files.

nams

Character string vector, pathnames of opened files.

mod

file opening mode. Formed by three digits abc

Fortran files

a

0 stands for formatted and 1 for unformatted (binary)

b

0 stands for sequential access and 1 for direct access

c

0 stands for "new", 1 for "old", 2 for "scratch" and 3 for "unknown"

C files

b

is 1 if file has been opened with a "b" (binary) mode

swap

automatic swap switch. swap=1 if automatic swap is on. swap is always 0 for Fortran files.

err

integer, error message number (see error), if open fails. If err is omitted an error message is issued.

action

is one of the following strings:

"close"

closes the file(s) given by the logical unit descriptors given in `units`

"rewind"

puts the pointer at beginning of file

"backspace"

puts the pointer at beginning of last record.

"last"

puts the pointer after last record.

Description

selects a logical unit `unit` and manages the file `file-name`.

`[unit [,err]]=file('open', file-name [,status] [,access [,re-cl]] [,format])` allows to open a file with specified properties and to get the associated unit number `unit`. This unit number may be used for further actions on this file or as file descriptor in `read`, `write`, `readb`, `writb`, `save`, `load` function calls.

This function can not open a UTF filename. In this case, please uses `mopen`.

`file(action,unit)` allows to close the file , or move the current file pointer .

`file()` returns the logical unit descriptors of the opened files. So `file('close',file() )` closes all user opened files (C or Fortran type).

Examples

```
u=file('open',TMPDIR+'/foo','unknown')
```

```
for k=1:4
    a=rand(1,4)
    write(u,a)
end
file('rewind',u)
x=read(u,2,4)
file('close',u)
//
u1=file('open',TMPDIR+'/foo','unknown')
u2=mopen(TMPDIR+'/foo1','wb')
[units,typs,nams]=file()
file('close',u1);
mclose(u2);
```

See Also

save, load, write, read, writb, readb, uigetfile, mopen, mclose, file

Name

`getenv` — get the value of an environment variable

```
env=getenv(str [, rep] )
```

Parameters

`str`

character string specifying environment variable name `rep` : an optional character string. When this optional value is used, the function `getenv` returns the value `rep` when the environment variable `str` is not found.

`env`

character string which contain the environment variable value

Description

Return the value of an environment variable if it exists.

Examples

```
getenv('SCI')  
getenv('F00', 'foo')
```

Name

getio — get Scilab input/output logical units

```
ios=getio()
```

Parameters

ios

a vector [rio rte wio wte]

rio

current logical unit for reading instructions

rte

logical unit assigned for input in main scilab window

wio

logical unit relative to the diary file if any. wio=0 stands for no diary file opened

wte

logical unit assigned for output in main scilab window

Description

getio returns logical units assigned for main scilab input and output

See Also

file , exec

Name

getpid — get Scilab process identifier

```
id=getpid()
```

Description

Return an the scilab process identifier integer

Examples

```
d='SD_'+string(getpid())+'_'
```

Name

getscilabkeywords — returns a list with all scilab keywords.

```
list_keywords=getscilabkeywords()
```

Parameters

list_keywords
a list

Description

list_keywords(1) : primitives

list_keywords(2) : commands

list_keywords(3) : predef variables

list_keywords(4) :scilab functions

list_keywords(5) :scicos functions

Authors

A.C, adapted from Enrico Segre's code in Scipad

Name

halt — stop execution

```
halt()  
halt('a message')
```

Description

stops execution until something is entered in the keyboard.

Examples

```
halt('Press a key')  
  
halt()
```

See Also

pause , return , exec

Name

host — Unix or DOS command execution

```
stat=host(command-name)
```

Parameters

command-name

A character string containing Unix sh instruction

stat

An integer flag

Description

Sends a string `command-name` to Unix for execution by the command interpreter (sh under Unix, or `command.com` under DOS). Standard output and standard errors of the shell command are written in the calling shell. `stat` gives -1 if `host` can't be called (Not enough system memory available) or the command interpreter return code.

Examples

```
//create a getdir function based on host
function wd=getdir()
  if MSDOS then
    host('cd>'+TMPDIR+'\path');
  else
    host('pwd>'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)')
endfunction
//call it
wd=getdir()
```

See Also

`edit` , `manedit` , `unix_g` , `unix_s` , `unix_w` , `unix_x`

Name

`input` — prompt for user input

```
x = input(message [, "string"])
```

Parameters

`message`

character string

`"string"`

the character string `"string"` (may be abbreviated to `"s"`)

`x`

real number (or character string if `"string"` is in the calling sequence)

Description

`input(message)` gives the user the prompt in the text string and then waits for input from the keyboard. The input can be expression which is evaluated by `evstr`. If nothing but a carriage return is entered at the prompt `input(message)` returns an empty matrix

Invoked with two arguments, the output is a character string which is the expression entered at keyboard. If nothing but a carriage return is entered at the prompt `input(message)` returns a single white space " ".

Examples

```
//x=input("How many iterations?")
//x=input("What is your name?","string")
```

See Also

`evstr` , `x_dialog` , `x_mdialog`

Name

load — Load a saved variable or a serie of variables

```
load(filename [,x1,...,xn])  
load(fd [,x1,...,xn])
```

Parameters

filename

character string containing the path of the file

fd

a file descriptor given by a call to mopen

xi

arbitrary Scilab variable name(s) given as strings.

Description

The `load` command can be used to reload in the Scilab session variables previously saved in a file with the `save` command. If the file contains graphic handle variables, the corresponding `graphics_entities` are drawn.

Since Scilab 5.0, all `uimenu` or `uicontrol` handles are also drawn.

`load(filename)` loads the variables saved in file given by its path `filename`.

`load(fd)` loads the variables saved in file given by its descriptor `fd`.

`load(filename, 'x', 'y')` or `load(fd, 'x', 'y')` loads only variables `x, y`.

Even if the binary file format has changed with 2.5 version of Scilab, `load(filename, ...)` is able to read old format files.

Note that the written file is portable to other operating systems and architectures (little and big endian).

Examples

```
a=eye(2,2);b=ones(a);  
save('vals.dat',a,b);  
clear a  
clear b  
load('vals.dat','a','b');
```

See Also

`save`, `read`, `listvarinfile`, `save_format`, `exec`, `mopen`

Name

`oldload` — load saved variable in 2.4.1 and previous formats (OBSOLETE)

Description

The `oldload` function is obsolete and removed of scilab 5.2 and more.

Replaces in your code `oldload` by `load`.

See Also

`load`, `save`

Name

oldsave — saving variables in 2.4.1 and previous format (OBSOLETE)

```
oldsave(filename [,x1,x2,...,xn])
```

Description

The `oldsave` function is obsolete and removed of scilab 5.2 and more.

Replaces in your code `oldsave` by `save`.

See Also

load, save

Name

read — matrices read

```
[x]=read(file-desc,m,n,[format])  
[x]=read(file-desc,m,n,k,format)
```

Parameters

`file-desc`

character string specifying the file name or integer value specifying logical unit (see file).

`m, n`

integers (dimensions of the matrix `x`). Set `m=-1` if you do not know the numbers of rows, so the whole file is read.

`format` : character string, specifies a "Fortran" format. This

character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point or character edition modes.

`k`

integer or vector of integer

Description

reads row after row the `m` $\times$ `n` matrix `x` (`n=1` for character chain) in the file `file-desc` (string or integer). Each row of the matrix `x` begin in a new line of `file-desc` file. Depending on `format`, a given row of the `x` matrix may be read from more than one line of `file-desc` file.

The type of the result will depend on the specified format. If format contains only (`d`, `e`, `f`, `g`) descriptors the function tries to read numerical data (the result is matrix of real numbers).

If format contains only `a` descriptors the function tries to read character strings (the result is a character string column vector). In this case `n` must be equal to 1. Warning: The character strings are truncated when they are longer than 4093.

Examples for `format`:

```
(1x,e10.3,5x,3(f3.0))  
(10x,a20)
```

When format is omitted datas are read using numerical free format: blank, comma and slash may be used as data separators, `n*v` may be use to represent `n` occurrences of value `n`.

A direct access file can be used if using the parameter `k` which is the vector of record numbers to be read (one record per row), thus `m` must be `m=prod(size(k))`.

To read on the keyboard use `read(%io(1),...)`.

Remark

Last line of data files must be terminated by a newline to be taken into account.

Examples

```
if MSDOS then unix('del foo');  
else unix('rm -f foo'); end  
A=rand(3,5); write('foo',A);  
B=read('foo',3,5)  
B=read('foo',-1,5)  
read(%io(1),1,1,'(a)') // waits for user's input
```

See Also

[write](#) , [load](#) , [file](#) , [readb](#) , [x_dialog](#) , [mscanf](#) , [mfscanf](#) , [msscanf](#) , [fscanfMat](#)

Name

read4b — fortran file binary read

```
x=read4b(file-name,m,n [,rec])
```

Parameters

file-name

string or integer

m, n

integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the file is read

rec

vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of desired x.

Description

binary read of the matrix x in the file `file-name`. Matrix entries are supposed to have been stored on 4 byte words.

For direct record access, file must have been previously opened using `file` function to set the `record_length`. `file-name` must be the result of the `file` function.

See Also

`file` , `write` , `writb` , `mget` , `write4b`

Name

readb — fortran file binary read

```
x=readb(file-name,m,n [,rec])
```

Parameters

file-name

string or integer

m, n

integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the file is read

rec

vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of desired x.

Description

binary read of the matrix x in the file `file-name`. Matrix entries are supposed to have been stored on 8 byte words.

For direct record access, file must have been previously opened using `file` function to set the `record_length`. `file-name` must be the result of the `file` function.

See Also

`file` , `write` , `writb` , `mget` , `read4b`

Name

readc_ — read a character string (obsolete see input)

```
[c]=readc_(unit)
[c]=readc_()
```

Description

readc_ reads a character string. This function allows one to interrupt an exec file without pause; the exec file stops until carriage return is made.

See Also

read

Name

save — Save a variable or a serie of variables in a binary file

```
save(filename [,x1,x2,...,xn])
save(fd [,x1,x2,...,xn])
```

Parameters

filename

Character string containing the path of the file

fd

A file descriptor given by a call to `mopen`

xi

Arbitrary Scilab variable(s)

Description

The `save` command can be used to save Scilab current variables in a binary file. If a variable is a graphic handle, the `save` function saves all the corresponding `graphics_entities` definition.

Since Scilab 5.0, all `uimenu` or `uicontrol` handles are also saved by this function.

The file can be given either by its paths or by its descriptor previously given by `mopen`.

`save(filename)` saves all current variables in the file defined by `filename`.

`save(fd)` saves all current variables in the file defined by the descriptor `fd`.

`save(filename,x,y)` or `save(fd,x,y)` saves only named variables `x` and `y`.

Saved variables can be reloaded by the `load` command.

Note that the written file is portable to other operating systems and architectures (little and big endian).

Examples

```
a=eye(2,2);b=ones(a);
save('val.dat',a,b);
clear a
clear b
load('val.dat','a','b');

// sequential save into a file
fd=mopen('TMPDIR/foo','wb')
for k=1:4, x=k^2;save(fd,x,k),end
mclose(fd)
fd=mopen('TMPDIR/foo','rb')
for i=1:4, load(fd,'x','k');x,k,end
mclose(fd)

// appending variables to an old save file
fd=mopen('TMPDIR/foo','r+')
mseek(0,fd,'end')
lst=list(1,2,3)
```

```
save(fd, lst)
mclose(fd)
```

See Also

load, write, save_format, mopen

Name

setenv — set the value of an environment variable

```
rep=setenv(name, value )
```

Parameters

name

Points to the name of an environment variable . (name is a string)

value

Points to the value to be assigned to the environment variable. (value is a string)

rep

Returns %T if it is ok else %F.

Description

set the value of an environnment variable.

Examples

```
setenv('toto', 'example')
getenv('toto')
```

See Also

[getenv](#)

Authors

Allan CORNET

Name

unix — shell (sh) command execution

```
stat=unix(command-name)
```

Parameters

command-name

A character string containing Unix sh instruction

stat

An integer flag

Description

Sends a string `command-name` to Unix for execution by the sh shell. Standard output and standard errors of the shell command are written in the calling shell. `stat` gives -1 if unix can't be called (Not enough system memory available) or the sh return code.

Examples

```
if ~MSDOS then
  unix("ls $SCI/demos");
end

function wd=directory()
  if MSDOS then
    unix('cd>'+TMPDIR+'\path');
  else
    unix('pwd>'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)');
endfunction

wd=directory()
```

See Also

`edit` , `manedit` , `unix_g` , `unix_s` , `unix_w` , `unix_x` , `host`

Name

unix_g — shell (sh) command execution, output redirected to a variable

```
rep=unix_g(cmd)
[rep,stat]=unix_g(cmd)
[rep,stat,stderr]=unix_g(cmd)
```

Parameters

cmd
a character string

rep
a column vector of character strings (standard output)

stat
a integer, the error status. stat=0 if no error occurred

err
a column vector of character strings (standard error)

Description

Sends a string **cmd** to Unix for execution by the sh shell. The standard output is redirected to scilab variable **rep**. The standard error is redirected to scilab variable **err** or displays if you had only 2 output arguments. Unix execution errors are trapped; **NOTE** that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
function d=DIR(path)
    path=pathconvert(path,%t,%t)
    if MSDOS then
        d=unix_g('dir '+path)
    else
        d=unix_g('ls '+path)
    end
endfunction

DIR('SCI/etc')
```

See Also

unix_s, unix_w, unix_x, unix

Name

unix_s — shell (sh) command execution, no output

```
unix_s(cmd)
```

Parameters

cmd
a character string

Description

Sends a string cmd to Unix for execution by the sh shell. The standard output is redirected to /dev/null. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
    unix_s("del foo");
else
    unix_s("rm -f foo");
end
```

See Also

edit , manedit , unix_g , unix_w , unix_x , unix

Name

unix_w — shell (sh) command execution, output redirected to scilab window

```
unix_w(cmd)
```

Parameters

cmd
a character string

Description

Sends a string cmd to Unix for execution by the sh shell. The standard output is redirected to scilab window. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
  unix_w("dir "+'"'+WSCI+"\modules"+'"');
else
  unix_w("ls $SCI/modules");
end
```

See Also

edit , manedit , unix_g , unix_s , unix_x , unix

Name

unix_x — shell (sh) command execution, output redirected to a window

```
unix_x(cmd)
```

Parameters

cmd
a character string

Description

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to a window. Unix execution errors are trapped; **NOTE** that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
  unix_x("dir "+"++++"+WSCI+"modules\graphics\demos"+"++++");
else
  unix_x("ls $SCI/modules/graphics/demos");
end
```

See Also

edit , manedit , unix_g , unix_s , unix_w , unix

Name

writb — fortran file binary write

```
writb(file-name,a [,rec])
```

Parameters

file-name

string or integer

rec

vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of **a**

Description

writes in binary format the matrix **a** in the file '**filename**'.. Matrix entries are stored on 4 byte words

For direct record access, file must have been previously opened using **file** function to set the **record_length**. **file-name** must be the result of the **file** function.

See Also

file , **readb** , **write** , **mput** , **write4b**

Name

write — write in a formatted file

```
write(file-desc,a,[format])
write(file-desc,a,k,format)
```

Parameters

file-desc

character string specifying the file name or integer value specifying logical unit (see file).

This function can not open a UTF filename. In this case, please uses mopen.

a

real matrix or column vector of character strings.

format

character string, specifies a "Fortran" format. This character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point , integer or character edition modes

k

integer vector

Description

writes row-by-row a real matrix or a column vector of character strings in a formatted file. Each row of the **a** argument begin in a new line of **file-desc** file. Depending on **format** a given row of the **a** argument may be written in more than one line of **file-desc** file.

Format examples : (1x,e10.3,5x,3(f3.0)) , (10x,a20) ;

See a Fortran book for more precision.

Direct access files : **x=write(file_desc,a,k,format)**. Here **k** is the vector of records (one record by row, i.e. **m=prod(size(k))**)

write(%io(2),...) writes on Scilab's window. Note that in this case **format** should produce one output line per matrix row. If this constraint is not verified unpredictable behavior could happen.

Examples

```
if MSDOS then unix('del asave');
else unix('rm -f asave'); end
A=rand(5,3); write('asave',A); A=read('asave',5,3);
write(%io(2),A,('' | '',3(f10.3, '' | '')))
write(%io(2),string(1:10))
write(%io(2),strcat(string(1:10),','))
write(%io(2),1:10,'(10(i2,3x))')

if MSDOS then unix('del foo');
else unix('rm -f foo'); end
write('foo',A)
```

See Also

read , save , file , fileinfo , writb , print , string , mfprintf , mprintf , msprintf , fprintfMat

Name

write4b — fortran file binary write

```
write4b(file-name,a [,rec])
```

Parameters

file-name

string or integer

rec

vector of positive integers, the selected records for direct access. This vector size must be equal to the number of rows of **a**

Description

writes in binary format the matrix **a** in the file '**filename**'. Matrix entries are stored on 8 byte words

For direct record access, file must have been previously opened using **file** function to set the **record_length**. **file-name** must be the result of the **file** function.

See Also

file , **readb** , **write** , **mput** , **read4b**

Part XX. Output functions

Name

diary — diary of session

```
diary(filename)
[id,filename] = diary(filename, ['new'|'append'])

[ids, filenames] = diary()
[ids, filenames] = diary([], 'list')

diary([], 'close')
diary(0)
diary(filename, 'close')
diary(id, 'close')

diary([], 'pause'|'off')
diary(filename, 'pause'|'off')
diary(id, 'pause'|'off')

diary([], 'resume'|'on')
diary(filename, 'resume'|'on')
diary(id, 'resume'|'on')

diary(filenames, 'exists')
diary(ids, 'exists')

diary(filename, 'new'|'append', 'prefix=YYYY-MM-DD hh:mm:ss')
diary(filename, 'new'|'append', 'prefix=U')
diary(filename, 'new'|'append', [ 'prefix=YYYY-MM-DD hh:mm:ss' ; 'prefix-
```

Parameters

filename
a character string, give the full file name path.

id
a scalar to identify a diary.

Description

`diary(f)` function creates a log of keyboard input and the resulting text output.

Start a diary session

```
[id, filename] = diary(filename, ['new'|'append'])
```

returns :

- * **id** : a positive integer (≥ 1) which is the diary session identifier.
- * **filename** : A string, the absolute path of the effective written file.

The first input argument is a string that contain the path of the diary file. This can be a relative path or a absolute path.

The 2nd input argument controls if a new file is created ('new') or if `diary()` adds new content at the end of the file ('append'). If the 2nd input argument is not present, the default value is 'new'.

When `diary()` is called with 'new' mode : If 'filename' already exists and is not empty, an effective filename 'base(filename)+_#+extension(filename)' is built, used, and returned by `diary(filename)` as a second output argument (beside id). The rank # would be set as the smallest integer for which the resultant filename does not yet exists.

List diary sessions

```
[ids, filenames] = diary()
```

```
[ids, filenames] = diary([], 'list')
```

returns a column vector of integer : identifiers list of opened diary sessions. a column vector of strings : absolute paths of the files associated with opened diary sessions.

Close diary session(s)

```
diary([], 'close')
```

```
diary(0)
```

```
diary(filename, 'close')
```

```
diary(id, 'close')
```

The first and second syntaxes close all opened diary sessions.

The third syntax closes diary session(s) identified by 'filename'.

The fourth syntax closes the diary session identified by "id" which is a positive integer or a vector of positive integers.

Remark : `diary(0)` is retained as backwards compatibility.

Pause/Resume diary session(s)

```
diary([], 'pause'|'off')
```

```
diary(filename, 'pause'|'off')
```

```
diary(id, 'pause'|'off')
```

The first syntax suspends all opened diary sessions.

The second syntax suspend diary session(s) identified by 'filename'. 'filename' can be a single string or a character string array.

The third syntax suspend the diary session identified by 'id' which is a positive integer or a vector of positive integers.

```
diary([], 'resume'|'on')
```

```
diary(filename, 'resume'|'on')
```

```
diary(id, 'resume'|'on')
```

The first syntax resume all opened diary sessions.

The second syntax resume diary session(s) identified by 'filename'. 'filename' can be a single string or a character string array.

The third syntax resume the diary session identified by 'id' which is a positive integer or a vector of positive integers.

Does a diary session exists ?

```
diary(filename, 'exists')
```

```
diary(id, 'exists')
```

return true if a diary session is currently opened with the file 'filename', if not false.

Diary and time-stamp

```
diary(filename, 'new', 'prefix=YYYY-MM-DD hh:mm:ss')
```

```
diary(filename, 'new', 'prefix=U')
```

```
diary(filename, 'new', [ 'prefix=YYYY-MM-DD hh:mm:ss' ; 'prefix-only-commands' ] );
```

'prefix=YYYY-MM-DD hh:mm:ss' add date & hour

'prefix=U' add UNIX time epoch

'prefix-only-commands' add time-stamp only as prefix for commands

Filtering diary

```
diary(filename, "new", "filter=command")
```

 Log only the input commands.

```
diary(filename, "new", "filter=output")
```

 Log only the text output.

Examples

```
d1 = diary(TMPDIR + '/diary1.txt')
d2 = diary(TMPDIR + '/diary2.txt')

// some Scilab instructions ...
cd TMPDIR
dir

// returns infos about opened diary
[ids, filenames] = diary()

// close diary d1
diary(d1, 'close')
[ids, filenames] = diary()

// close diary d2
diary(TMPDIR + '/diary2.txt')
[ids, filenames] = diary()
// closes all diary
diary([], 'close')
[ids, filenames] = diary()
```

Name

disp — displays variables

```
disp(x1, [x2, ...xn])
```

Description

displays x_i with the current format. x_i 's are arbitrary objects (matrices of constants, strings, functions, lists, ...)

Display of objects defined by `tlist` may be overloaded by the definition of a function. This function must have no output argument a single input argument and its name is formed as follows: `%<tlist_type>_p` where `%<tlist_type>` stands for the first entry of the `tlist` type component.

The `lines` function may be used to control the output.

Examples

```
disp([1 2],3)
deff('[]=%t_p(l)', 'disp(l(3),l(2))')
disp(tlist('t',1,2))
```

See Also

`lines` , `write` , `read` , `print` , `string` , `tlist`

Name

`mprintf` — converts, formats, and writes data to the main scilab window

```
mprintf(format,a1,...,an);
```

Parameters

`format`

a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C `printf` format operand syntax.

`a1,...,an`

Specifies the data to be converted and printed according to the format parameter.

Description

The `mprintf` function is a interface for C-coded version of `printf` function.

The `mprintf` function writes formatted operands to the standard Scilab output (i.e the Scilab window). The argument operands are formatted under control of the format operand.

Examples

```
mprintf('At iteration %i, Result is:\nalpha=%f',33,0.535)
```

See Also

`disp`

Name

`msprintf` — converts, formats, and writes data in a string

```
str=msprintf(format,a1,...,an);
```

Parameters

`format`

a Scilab string describing the format to use to write the remaining operands.

`str`

a character string.

`a1,...,an`

Specifies the data to be converted and printed according to the format parameter.

Description

The `msprintf` writes formatted operands in its returned value (a Scilab string). The argument operands are formatted under control of the format operand.

Note that, in this case, the escape sequences ("`\n`") split string to a matrix of string (see example)

Examples

```
msprintf('%5.3f %5.3f',123,0.732)
msprintf('%5.3f\n%5.3f',123,0.732)
msprintf('--%s-\n-%d--',"hello",3)
```

See Also

`mprintf`, `printf_conversion`

Name

prettyprint — From any Scilab datatype and provide a representation to the TeX, LaTeX or MathML formats

```
str = prettyprint(a) // Show the variable a with the default format (LaTeX)
str = prettyprint(a,exportFormat) // Show the variable a with the specified format
str = prettyprint(a,exportFormat, delim) // As above but change the delimiter
str = prettyprint(a,exportFormat, delim, processByElement) // As above but process by element
str = prettyprint(a,exportFormat, delim, processByElement, isWrapped) // As above but wrap the result
```

Parameters

a:

is a Scilab variable

exportFormat:

is the format, if omitted 'latex' is used by default, it can be 'latex', 'tex' or 'mathml'.

delimiter:

is a string indicating the delimiter to use for the resulting matrix, it's only used if isWrapped is true. The delimiter can be '(', '{', '[', '|', '||' or ')'

processByElement:

is a boolean to indicate if the resulting matrix must be converted into a single string.

isWrapped:

is a boolean to indicate if the result must be wrapped inside delimiters ('\$' for latex and tex or nothing for mathml) to be used with xstring or xtitle

str:

the representation of the variable a

Description

Taking a variable, the prettyprint function will provide a formatted representation of it. Formats can be TeX, LaTeX or MathML. They can be used in third party applications but also within Scilab with the most of the Scilab graphic features. The following types are handled by this function:

- Real / Complex matrices
- Polynomial types
- Boolean
- Integer
- String
- Tlist
- Rational
- Cell

Examples


```
str = prettyprint(rand(3,3)) // Return the LaTeX representation of a 3,3 matrix
xstring(0.2,0.2,str) // Show the representation in a graphic Windows

prettyprint(rand(3,4),"mathml") // Return the MathML representation of a 3,4 ma
prettyprint(rand(3,4),"mathml","[") // Return the MathML representation of a 3,

s=poly(0,'s'); G=[1,s;1+s^2,3*s^3];
xstring(0.2,0.2,prettyprint(G*s-1)); // Show a polynom through a LaTeX represen
```

See also

[math_rendering_features_in_graphic](#), [xtitle](#), [axes_properties](#), [label_properties](#), [legend_properties](#),
[text_properties](#), [xstringb](#), [xstringl](#), [xstring](#)

Authors

Calixte Denizet

Name

print — prints variables in a file

```
print('file-name',x1,[x2,...xn])
```

Description

prints `xi` on file '`file-name`' with the current format, i.e. the format used by scilab to display the variables. All types of variables may be "print"ed

Note : `xi` must be a named variable, with expressions variable name part of the display is unpredictable.

`print(%io(2),...)` prints on Scilab's window. this syntax may be used to display variables within a macro.

Examples

```
a=rand(3,3);p=poly([1,2,3],'s');l=list(1,'asdf',[1 2 3]);  
print(%io(2),a,p,l)  
write(%io(2),a)
```

See Also

`write` , `read` , `format` , `printf` , `disp`

Name

`printf` — Emulator of C language `printf` function

```
printf(format,value_1,...,value_n)
```

Parameters

`format`

a Scilab string. Specifies a character string combining literal characters with conversion specifications.

`value_i`

Specifies the data to be converted according to the `format` parameter.

Description

The `printf` function converts, formats, and writes its `value` parameters, under control of the `format` parameter, to the standard output.

The `format` parameter is a character string that contains two types of objects:

Literal characters

which are copied to the output stream.

Conversion specifications

each of which causes zero or more items to be fetched from the `value` parameter list. see `printf_conversion` for details

If any `values` remain after the entire `format` has been processed, they are ignored.

Examples

```
printf('Result is:\nalpha=%f',0.535)
```

See Also

`string` , `print` , `write` , `format` , `disp` , `file` , `fprintf` , `sprintf` , `printf_conversion`

Name

printf_conversion — printf, sprintf, fprintf conversion specifications

Description

Each conversion specification in the `printf`, `sprintf`, `fprintf` format parameter has the following syntax:

- A `%` (percent) sign.
- Zero or more **options**, which modify the meaning of the conversion specification. The following list contains the **option** characters and their meanings:
 - `-` : Left align, within the field, the result of the conversion.
 - `+` : Begin the result of a signed conversion with a sign (+ or -).
 - `'space'` : Prefix a space character to the result if the first character of a signed conversion is not a sign. If both the (space) and `+` options appear, the (space) option is ignored
 - `#` : Convert the value to an alternate form. For `c`, `d`, `i`, `s`, and `u` conversions, the `#` option has no effect. For `o` conversion, `#` increases the precision to force the first digit of the result to be a 0 (zero). For `x` and `X` conversions, a nonzero result has `0x` or `0X` prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result always contains a decimal point, even if no digits follow it. For `g` and `G` conversions, trailing zeros are not removed from the result.
 - `0` : Pad to the field width, using leading zeros (following any indication of sign or base) for `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions; no space padding is performed. If the `0` and `\-` (dash) flags both appear, the `0` flag is ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag is also ignored.

An optional decimal digit string that specifies the minimum field width. If the converted value has fewer characters than the field width, the field is padded on the left to the length specified by the field width. If the left-adjustment option is specified, the field is padded on the right.

An optional precision. The precision is a `.` (dot) followed by a decimal digit string. If no precision is given, the parameter is treated as 0 (zero). The precision specifies:

- The minimum number of digits to appear for `d`, `u`, `o`, `x`, or `X` conversions
- The number of digits to appear after the decimal point for `e`, `E`, and `f` conversions
- The maximum number of significant digits for `g` and `G` conversions
- The maximum number of characters to be printed from a string in an `s` conversion
- A character that indicates the type of conversion to be applied:
 - `%` : Performs no conversion. Displays `%`.
 - `d,i` : Accepts an integer **value** and converts it to signed decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.
 - `u` : Accepts an integer **value** and converts it to unsigned decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting

a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.

- **o** : Accepts an integer **value** and converts it to unsigned octal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros. An octal value for field width is not implied.
- **x, X** : Accepts an integer **value** and converts it to unsigned hexadecimal notation. The letters ``abcdef" are used for the **x** conversion; the letters ``ABCDEF" are used for the **X** conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.
- **f** : Accepts a float or double **value** and converts it to decimal notation in the format `%[-]ddd .ddd`. The number of digits after the decimal point is equal to the precision specification.
 - If no precision is specified, six digits are output.
 - If the precision is zero, no decimal point appears and the system outputs a number rounded to the integer nearest to **value**.
 - If a decimal point is output, at least one digit is output before it.
- **e, E** : Accepts a real and converts it to the exponential form `%[-]d .ddde+/-dd`. There is one digit before the decimal point, and the number of digits after the decimal point is equal to the precision specification.
 - If no precision is specified, six digits are output.
 - If the precision is zero, , no decimal point appears.
 - The **E** conversion character produces a number with **E** instead of **e** before the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.
- **g, G** : Accepts a real and converts it in the style of the **e**, **E**, or **f** conversion characters, with the precision specifying the number of significant digits. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit. The style used depends on the value converted. Style **e** (**E**, if **G** is the flag used) results only if the exponent resulting from the conversion is less than -4, or if it is greater or equal to the precision.
- **c** : Accepts and displays an integer value converted to a character.
- **s** : Accepts a string **value** and displays characters from the string to the end or the number of characters indicated by the precision is reached. If no precision is specified, all characters up to the end are displayed.

A field width or precision can be indicated by an ***** (asterisk) instead of a digit string. In this case, an integer **value** parameter supplies the field width or precision. The **value** parameter converted for output is not fetched until the conversion letter is reached, so the parameters specifying field width or precision must appear before the value to be converted (if any).

If the result of a conversion is wider than the field width, the field is expanded to contain the converted result.

The representation of the plus sign depends on whether the **+** or (space) formatting option is specified.

Examples

```
printf('a string: %s\n', 'Scilab');
printf('an integer: %d\n', 10);
printf('an integer: %4d\n', 10);
printf('a left justified integer: %-4d\n', 10);
printf('an integer converted to float: %#fd\n', 10);
printf('an integer with a sign: %+4d\n', 10);
printf('an integer with a sign: %+4d\n', -10);
printf('an integer padded with zeros: %04d\n', 10);
printf('an unsigned integer: %u\n', 10);
printf('an unsigned integer: %4u\n', -10);
printf('an integer converted to hexadecimal: %x\n', 10);
printf('a float: %d\n', %pi);
printf('a float: %3.2d\n', %pi);
printf('a float (exponential form): %3.2e\n', %pi);
printf('a float (exponential form): %3.2g\n', %pi);
printf('a character: %c\n', 'a');
printf('a character: %c\n', 'aaa');
```

See Also

printf, fprintf, sprintf

Name

sprintf — Emulator of C language sprintf function

```
str=sprintf(format,value_1,...,value_n)
```

Parameters

`format`

a Scilab string. Specifies a character string combining literal characters with conversion specifications.

`value_i`

Specifies the data to be converted according to the `format` parameter.

`str`

column vector of character strings

Description

The `sprintf` function converts, formats, and stores its `value` parameters, under control of the `format` parameter.

The `format` parameter is a character string that contains two types of objects:

Literal characters

which are copied to the output stream.

Conversion specifications

each of which causes zero or more items to be fetched from the `value` parameter list. see `printf_conversion` for details

If there are not enough items for `format` in the `value` parameter list, `sprintf` generate an error. If any `values` remain after the entire `format` has been processed, they are ignored.

Note: `sprintf` is obsolete, use `msprintf` instead.

Examples

```
fahr=120
sprintf('%3d Fahrenheit = %6.1f Celsius',fahr,(5/9)*(fahr-32))
```

See Also

`string` , `print` , `write` , `format` , `disp` , `file` , `printf` , `fprintf` , `msprintf` , `printf_conversion`

Name

ssprint — pretty print for linear system

```
ssprint(sl [,out])
```

Parameters

sl
list (sys lin list)

out
output (default value out=%io(2))

Description

pretty print of a linear system in state-space form $sl=(A,B,C,D)$ sys lin list.

Examples

```
a=[1 1;0 1];b=[0 1;1 0];c=[1,1];d=[3,2];  
ssprint(syslin('c',a,b,c,d))  
ssprint(syslin('d',a,b,c,d))
```

See Also

texprint

Part XXI. Intersci

Name

intersci — scilab tool to interface C of Fortran functions with scilab

Description

All scilab primitive functions are defined in a set of interface routines. For each function the interfacing code checks first number of rhs and lhs arguments. Then it get pointers on input arguments in the Scilab data base and checks their types. After that it calls procedure associated with Scilab functions, checks returned errors flags and set the results in the data base.

`intersci\` is a program which permits to interface automatically FORTRAN subroutines or C functions to Scilab

With `intersci`, a user can group all his FORTRAN or C code into a same set, called an interface, and use them in Scilab as Scilab functions. The interfacing is made by creating a FORTRAN subroutine which has to be linked to Scilab together with the user code. This complex FORTRAN subroutine is automatically generated by `intersci\` from a description file of the interface.

Refer to intersci documentation for more details.

See Also

`fort` , `external` , `addinter`

Part XXII. JVM

Name

javaclasspath — set and get dynamic Java class path

```
res=javaclasspath()  
javaclasspath(path)
```

Parameters

res
a string matrix

Description

set and get the dynamic Java path to one or more directory or file specifications given in path.

Examples

```
res=javaclasspath();  
javaclasspath(SCI);  
javaclasspath([SCI,SCI+'/java']);
```

Authors

A.C

Name

javalibrarypath — set and get dynamic java.library.path

```
res=javalibrarypath()  
javlibrarypath(path)
```

Parameters

res
a string matrix

Description

set and get the dynamic Java Library path to one or more directory given in path.

When you use java classes with native methods, you need to define path where is dynamic library.

Examples

```
res=javalibrarypath();  
javlibrarypath(SCI);  
javlibrarypath([SCI,SCI+'/libs']);
```

See Also

javaclasspath

Authors

A.C

Name

jre_path — returns Java Runtime Environment used by Scilab

```
p=jre_path()
```

Parameters

p
a string path of JRE

Description

returns Java Runtime Environment used by Scilab.

See Also

system_getproperty

Authors

A.C

Name

system_getproperty — gets the system property indicated by a specified key.

```
res=system_getproperty(key)
```

Parameters

res
a string value

key
a string

Description

gets the system property indicated by a specified key.

java.version	Java Runtime Environment version
java.vendor	Java Runtime Environment vendor
java.vendor.url	Java vendor URL
java.home	Java installation directory
java.vm.specification.version	Java Virtual Machine specification version
java.vm.specification.vendor	Java Virtual Machine specification vendor
java.vm.specification.name	Java Virtual Machine specification name
java.vm.version	Java Virtual Machine implementation version
java.vm.vendor	Java Virtual Machine implementation vendor
java.vm.name	Java Virtual Machine implementation name
java.specification.version	Java Runtime Environment specification version
java.specification.vendor	Java Runtime Environment specification vendor
java.specification.name	Java Runtime Environment specification name
java.class.version	Java class format version number
java.class.path	Java class path
java.library.path	List of paths to search when loading libraries
java.io.tmpdir	Default temp file path
java.compiler	Name of JIT compiler to use
java.ext.dirs	Path of extension directory or directories
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
file.separator	File separator ("/" on UNIX)
path.separator	Path separator (":" on UNIX)
line.separator	Line separator ("n" on UNIX)
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

Examples

```
system_getproperty('awt.toolkit')
system_getproperty('file.encoding')
system_getproperty('file.encoding.pkg')
system_getproperty('java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment')
system_getproperty('java.awt.printerjob=sun.awt.windows.WPrinterJob')
system_getproperty('java.class.path')
system_getproperty('java.class.version')
system_getproperty('java.endorsed.dirs')
system_getproperty('java.ext.dirs')
system_getproperty('java.home')
system_getproperty('java.io.tmpdir')
system_getproperty('java.library.path')
system_getproperty('java.runtime.name')
system_getproperty('java.runtime.version')
system_getproperty('java.specification.name')
system_getproperty('java.specification.vendor')
system_getproperty('java.specification.version')
system_getproperty('java.vendor')
system_getproperty('java.vendor.url')
system_getproperty('java.vendor.url.bug')
system_getproperty('java.version')
system_getproperty('java.vm.info')
system_getproperty('java.vm.name')
system_getproperty('java.vm.specification.name')
system_getproperty('java.vm.specification.vendor')
system_getproperty('java.vm.specification.version')
system_getproperty('java.vm.vendor')
system_getproperty('java.vm.version')
system_getproperty('line.separator')
system_getproperty('os.arch')
system_getproperty('os.name')
system_getproperty('os.version')
system_getproperty('path.separator')
system_getproperty('sun.arch.data.model')
system_getproperty('sun.boot.class.path')
system_getproperty('sun.boot.library.path')
system_getproperty('sun.cpu.endian')
system_getproperty('sun.cpu.isalist')
system_getproperty('sun.desktop')
system_getproperty('sun.io.unicode.encoding')
system_getproperty('sun.jnu.encoding')
system_getproperty('sun.management.compiler')
system_getproperty('sun.os.patch.level')
system_getproperty('user.country')
system_getproperty('user.dir')
system_getproperty('user.home')
system_getproperty('user.language')
system_getproperty('user.name')
system_getproperty('user.timezone')
system_getproperty('user.variant')
```


Authors

A.C

Name

system_setproperty — set a system property indicated by a specified key and value.

```
prev = system_setproperty(key, value)
```

Parameters

prev
a string previous value or []

key
a string

value
a string

Description

Sets the system property indicated by the specified key.

Warning : change property with precaution.

Examples

```
system_getproperty('myproperty')  
system_setproperty('myproperty', 'hello')  
system_getproperty('myproperty')
```

Authors

A.C

Name

with_embedded_jre — checks if scilab uses a embedded JRE

```
res=with_embedded_jre()
```

Parameters

res

a boolean

Description

checks if scilab uses a embedded JRE.

Examples

```
res=with_embedded_jre();
```

Authors

A.C

Part XXIII. Linear Algebra

Name

aff2ab — linear (affine) function to A,b conversion

```
[A,b]=aff2ab(afunction,dimX,D [,flag])
```

Parameters

afunction

a scilab function $Y = fct(X,D)$ where X , D , Y are list of matrices

dimX

a $p \times 2$ integer matrix (p is the number of matrices in X)

D

a list of real matrices (or any other valid Scilab object).

flag

optional parameter (flag='f' or flag='sp')

A

a real matrix

b

a real vector having same row dimension as A

Description

aff2ab returns the matrix representation of an affine function (in the canonical basis).

afunction is a function with imposed syntax: $Y = afunction(X,D)$ where $X = list(X1, X2, \dots, Xp)$ is a list of p real matrices, and $Y = list(Y1, \dots, Yq)$ is a list of q real matrices which depend linearly of the X_i 's. The (optional) input D contains parameters needed to compute Y as a function of X . (It is generally a list of matrices).

dimX is a $p \times 2$ matrix: $dimX(i) = [nr_i, nci]$ is the actual number of rows and columns of matrix X_i . These dimensions determine na , the column dimension of the resulting matrix A: $na = nr_1 * nc_1 + \dots + nr_p * nc_p$.

If the optional parameter flag='sp' the resulting A matrix is returned as a sparse matrix.

This function is useful to solve a system of linear equations where the unknown variables are matrices.

Examples

```
// Lyapunov equation solver (one unknown variable, one constraint)
deff('Y=lyapunov(X,D)', '[A,Q]=D(:); Xm=X(:); Y=list(A'*Xm+Xm*A-Q)')
A=rand(3,3); Q=rand(3,3); Q=Q+Q'; D=list(A,Q); dimX=[3,3];
[Aly,bly]=aff2ab(lyapunov,dimX,D);
[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); lyapunov(Xv,D)
Xm=Xv(:); A'*Xm+Xm*A-Q

// Lyapunov equation solver with redundant constraint X=X'
// (one variable, two constraints) D is global variable
deff('Y=ly2(X,D)', '[A,Q]=D(:); Xm=X(:); Y=list(A'*Xm+Xm*A-Q, Xm'-Xm)')
A=rand(3,3); Q=rand(3,3); Q=Q+Q'; D=list(A,Q); dimX=[3,3];
[Aly,bly]=aff2ab(ly2,dimX,D);
```

```

[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); ly2(Xv,D)

// Francis equations
// Find matrices X1 and X2 such that:
// A1*X1 - X1*A2 + B*X2 -A3 = 0
// D1*X1 -D2 = 0
deff('Y=bruce(X,D)', '[A1,A2,A3,B,D1,D2]=D(:), ...
[X1,X2]=X(:);Y=list(A1*X1-X1*A2+B*X2-A3,D1*X1-D2)')
A1=[-4,10;-1,2];A3=[1;2];B=[0;1];A2=1;D1=[0,1];D2=1;
D=list(A1,A2,A3,B,D1,D2);
[n1,m1]=size(A1);[n2,m2]=size(A2);[n3,m3]=size(B);
dimX=[m1,n2];[m3,m2];
[Af,bf]=aff2ab(bruce,dimX,D);
[Xf,KerAf]=linsolve(Af,bf);Xsol=vec2list(Xf,dimX)
bruce(Xsol,D)

// Find all X which commute with A
deff('y=f(X,D)', 'y=list(D(:)*X(:)-X(:)*D(:))')
A=rand(3,3);dimX=[3,3];[Af,bf]=aff2ab(f,dimX,list(A));
[Xf,KerAf]=linsolve(Af,bf);[p,q]=size(KerAf);
Xsol=vec2list(Xf+KerAf*rand(q,1),dimX);
C=Xsol(:); A*C-C*A

```

See Also

[linsolve](#)

Name

balanc — matrix or pencil balancing

```
[Ab,X]=balanc(A)
[Eb,Ab,X,Y]=balanc(E,A)
```

Parameters

A:
a real square matrix

X:
a real square invertible matrix

E:
a real square matrix (same dimension as A)

Y:
a real square invertible matrix.

Description

Balance a square matrix to improve its condition number.

$[Ab, X] = \text{balanc}(A)$ finds a similarity transformation X such that

$Ab = \text{inv}(X) * A * X$ has approximately equal row and column norms.

For matrix pencils, balancing is done for improving the generalized eigenvalue problem.

$[Eb, Ab, X, Y] = \text{balanc}(E, A)$ returns left and right transformations X and Y such that $Eb = \text{inv}(X) * E * Y$, $Ab = \text{inv}(X) * A * Y$

Remark

Balancing is made in the functions `bdiag` and `spec`.

Examples

```
A=[1/2^10, 1/2^10; 2^10, 2^10];
[Ab,X]=balanc(A);
norm(A(1,:))/norm(A(2,:))
norm(Ab(1,:))/norm(Ab(2,:))
```

See Also

`bdiag` , `spec` , `schur`

Name

bdiag — block diagonalization, generalized eigenvectors

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

Parameters

A
real or complex square matrix

rmax
real number

Ab
real or complex square matrix

X
real or complex non-singular matrix

bs
vector of integers

Description

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

performs the block-diagonalization of matrix A. bs gives the structure of the blocks (respective sizes of the blocks). X is the change of basis i.e $Ab = \text{inv}(X) * A * X$ is block diagonal.

rmax controls the conditioning of X; the default value is the l1 norm of A.

To get a diagonal form (if it exists) choose a large value for rmax (rmax=1/%eps for example). Generically (for real random A) the blocks are (1x1) and (2x2) and X is the matrix of eigenvectors.

Examples

```
//Real case: 1x1 and 2x2 blocks  
a=rand(5,5);[ab,x,bs]=bdiag(a);ab  
  
//Complex case: complex 1x1 blocks  
[ab,x,bs]=bdiag(a+%i*0);ab
```

See Also

schur , sylv , spec

Name

chfact — sparse Cholesky factorization

```
spcho=chfact(A)
```

Parameters

A
square symmetric positive sparse matrix

spcho
list containing the Cholesky factors in coded form

Description

`spcho=chfact(A)` computes the sparse Cholesky factors of sparse matrix `A`, assumed symmetric positive definite. This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in `spcho`. This function is to be used with `chsolve`.

See Also

`chsolve` , `sparse` , `lufact` , `luget` , `spchol`

Name

chol — Cholesky factorization

```
[R]=chol(X)
```

Parameters

X
a symmetric positive definite real or complex matrix.

Description

If X is positive definite, then $R = \text{chol}(X)$ produces an upper triangular matrix R such that $R' * R = X$.

$\text{chol}(X)$ uses only the diagonal and upper triangle of X . The lower triangular is assumed to be the (complex conjugate) transpose of the upper.

References

Cholesky decomposition is based on the Lapack routines DPOTRF for real matrices and ZPOTRF for the complex case.

Examples

```
w=rand(5,5)+%i*rand(5,5);  
X=w*w';  
R=chol(X);  
norm(R'*R-X)
```

See Also

spchol , qr , svd , bdiag , fullrf

Name

chsolve — sparse Cholesky solver

```
sol=chsolve(spcho,rhs)
```

Parameters

spcho

list containing the Cholesky factors in coded form returned by chfact

rhs, sol

full column vectors

Description

`sol=chsolve(spcho,rhs)` computes the solution of $\text{rhs}=\text{A}*\text{sol}$, with A a symmetric sparse positive definite matrix. This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in `spcho`.

Examples

```
A=sprand(20,20,0.1);
A=A*A'+eye();
spcho=chfact(A);
sol=(1:20)';rhs=A*sol;
spcho=chfact(A);
chsolve(spcho,rhs)
```

See Also

chfact , sparse , lufact , luget , spchol

Name

classmarkov — recurrent and transient classes of Markov matrix

```
[perm,rec,tr,indsRec,indsT]=classmarkov(M)
```

Parameters

M

real N x N Markov matrix. Sum of entries in each row should add to one.

perm

integer permutation vector.

rec, tr

integer vector, number (number of states in each recurrent classes, number of transient states).

indsRec,indsT

integer vectors. (Indexes of recurrent and transient states).

Description

Returns a permutation vector **perm** such that

```
M(perm,perm) = [M11 0 0 0 0 0]
                 [0 M22 0 0 0 0]
                 [0 0 M33 0 0 0]
                 [  . . .  . . . ]
                 [0 0 0 0 Mrr 0]
                 [* * 0 0 * Q]
```

Each M_{ii} is a Markov matrix of dimension $rec(i)$ $i=1, \dots, r$. Q is sub-Markov matrix of dimension tr . States 1 to $sum(rec)$ are recurrent and states from $r+1$ to n are transient. One has $perm=[indsRec, indsT]$ where $indsRec$ is a vector of size $sum(rec)$ and $indsT$ is a vector of size tr .

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

See Also

genmarkov , eigenmarkov

Name

cmb_lin — symbolic linear combination

```
[x]=cmb_lin(alfa,x,beta,y)
```

Description

Evaluates $\text{alfa} \cdot x - \text{beta} \cdot y$. `alfa`, `beta`, `x`, `y` are character strings. (low-level routine)

See Also

`mul`, `add`

Name

`coff` — resolvent (cofactor method)

```
[N,d]=coff(M [,var])
```

Parameters

`M`
square real matrix

`var`
character string

`N`
polynomial matrix (same size as `M`)

`d`
polynomial (characteristic polynomial `poly(A, 's')`)

Description

`coff` computes $R=(s \cdot \text{eye}() - M)^{-1}$ for `M` a real matrix. `R` is given by `N/d`.

`N` = numerator polynomial matrix.

`d` = common denominator.

`var` character string ('s' if omitted)

Examples

```
M=[1,2;0,3];  
[N,d]=coff(M)  
N/d  
inv(%s*eye()-M)
```

See Also

`coffg`, `ss2tf`, `nlev`, `poly`

Name

colcomp — column compression, kernel, nullspace

```
[W,rk]=colcomp(A [,flag] [,tol])
```

Parameters

A
real or complex matrix

flag
character string

tol
real number

W
square non-singular matrix (change of basis)

rk
integer (rank of A)

Description

Column compression of A: $AC = A*W$ is column compressed i.e

$AC=[0, Af]$ with Af full column rank, $\text{rank}(Af) = \text{rank}(A) = rk$.

flag and tol are optional parameters: flag = 'qr' or 'svd' (default is 'svd').

tol = tolerance parameter (of order %eps as default value).

The ma-rk first columns of W span the kernel of A when $\text{size}(A)=(na,ma)$

Examples

```
A=rand(5,2)*rand(2,5);  
[X,r]=colcomp(A);  
norm(A*X(:,1:$-r),1)
```

See Also

rowcomp , fullrf , fullrfk , kernel

Authors

F.D.;

Name

companion — companion matrix

```
A=companion(p)
```

Parameters

p
polynomial or vector of polynomials

A
square matrix

Description

Returns a matrix **A** with characteristic polynomial equal to **p** if **p** is monic. If **p** is not monic the characteristic polynomial of **A** is equal to **p/c** where **c** is the coefficient of largest degree in **p**.

If **p** is a vector of monic polynomials, **A** is block diagonal, and the characteristic polynomial of the *i*th block is **p(i)**.

Examples

```
s=poly(0, 's');  
p=poly([1,2,3,4,1], 's', 'c')  
det(s*eye()-companion(p))  
roots(p)  
spec(companion(p))
```

See Also

spec , poly , randpencil

Authors

F.D.

Name

cond — condition number

```
cond(X)
```

Parameters

X
real or complex square matrix

Description

Condition number in 2-norm. `cond(X)` is the ratio of the largest singular value of X to the smallest.

Examples

```
A=testmatrix('hilb',6);  
cond(A)
```

See Also

rcond , svd

Name

`det` — determinant

```
det(X)
[e,m]=det(X)
```

Parameters

`X`
real or complex square matrix, polynomial or rational matrix.

`m`
real or complex number, the determinant base 10 mantissae

`e`
integer, the determinant base 10 exponent

Description

`det(X)` ($m \cdot 10^e$ is the determinant of the square matrix `X`).

For polynomial matrix `det(X)` is equivalent to `determ(X)`.

For rational matrices `det(X)` is equivalent to `detr(X)`.

References

`det` computations are based on the Lapack routines DGETRF for real matrices and ZGETRF for the complex case.

Examples

```
x=poly(0,'x');
det([x,1+x;2-x,x^2])
w=ssrand(2,2,4);roots(det(systmat(w))),trzeros(w) //zeros of linear system
A=rand(3,3);
det(A), prod(spec(A))
```

See Also

`detr` , `determ`

Name

eigenmarkov — normalized left and right Markov eigenvectors

```
[M,Q]=eigenmarkov(P)
```

Parameters

- P
real N x N Markov matrix. Sum of entries in each row should add to one.
- M
real matrix with N columns.
- Q
real matrix with N rows.

Description

Returns normalized left and right eigenvectors associated with the eigenvalue 1 of the Markov transition matrix P. If the multiplicity of this eigenvalue is m and P is N x N, M is a m x N matrix and Q a N x m matrix. M(k,:) is the probability distribution vector associated with the kth ergodic set (recurrent class). M(k,x) is zero if x is not in the k-th recurrent class. Q(x,k) is the probability to end in the k-th recurrent class starting from x. If P^k converges for large k (no eigenvalues on the unit circle except 1), then the limit is Q^*M (eigenprojection).

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2)
[M,Q]=eigenmarkov(P);
P*Q-Q
Q*M-P^20
```

See Also

genmarkov , classmarkov

Name

ereduc — computes matrix column echelon form by qz transformations

```
[E,Q,Z [,stair [,rk]]=ereduc(X,tol)
```

Parameters

X

m x n matrix with real entries.

tol

real positive scalar.

E

column echelon form matrix

Q

m x m unitary matrix

Z

n x n unitary matrix

stair

vector of indexes,

*

$ISTAIR(i) = + j$ if the boundary element $E(i, j)$ is a corner point.

*

$ISTAIR(i) = - j$ if the boundary element $E(i, j)$ is not a corner point.

($i=1, \dots, M$)

rk

integer, estimated rank of the matrix

Description

Given an $m \times n$ matrix X (not necessarily regular) the function `ereduc` computes a unitary transformed matrix $E=Q*X*Z$ which is in column echelon form (trapezoidal form). Furthermore the rank of matrix X is determined.

Examples

```
X=[1 2 3;4 5 6]
[E,Q,Z [,stair [,rk]]=ereduc(X,1.d-15)
```

See Also

`fstair`

Authors

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

Name

expm — square matrix exponential

```
expm(X)
```

Parameters

X
square matrix with real or complex entries.

Description

X is a square matrix `expm(X)` is the matrix

$$\expm(X) = I + X + X^2/2 + \dots$$

The computation is performed by first block-diagonalizing X and then applying a Pade approximation on each block.

Examples

```
X=[1 2;3 4]
expm(X)
logm(expm(X))
```

See Also

`logm` , `bdiag` , `coff` , `log` , `exp`

Name

fstair — computes pencil column echelon form by qz transformations

```
[AE, EE, QE, ZE, blcks, muk, nuk, muk0, nuk0, mnei]=fstair(A, E, Q, Z, stair, rk, tol)
```

Parameters

- A**
m x n matrix with real entries.
- tol**
real positive scalar.
- E**
column echelon form matrix
- Q**
m x m unitary matrix
- Z**
n x n unitary matrix
- stair**
vector of indexes (see `ereduc`)
- rk**
integer, estimated rank of the matrix
- AE**
m x n matrix with real entries.
- EE**
column echelon form matrix
- QE**
m x m unitary matrix
- ZE**
n x n unitary matrix
- nbcks**
is the number of submatrices having full row rank ≥ 0 detected in matrix **A**.
- muk:**
integer array of dimension (n). Contains the column dimensions $\mu(k)$ ($k=1,\dots,nbcks$) of the submatrices having full column rank in the pencil $sE(\epsilon)-A(\epsilon)$
- nuk:**
integer array of dimension (m+1). Contains the row dimensions $\nu(k)$ ($k=1,\dots,nbcks$) of the submatrices having full row rank in the pencil $sE(\epsilon)-A(\epsilon)$
- muk0:**
integer array of dimension (n). Contains the column dimensions $\mu(k)$ ($k=1,\dots,nbcks$) of the submatrices having full column rank in the pencil $sE(\epsilon,inf)-A(\epsilon,inf)$
- nuk:**
integer array of dimension (m+1). Contains the row dimensions $\nu(k)$ ($k=1,\dots,nbcks$) of the submatrices having full row rank in the pencil $sE(\epsilon,inf)-A(\epsilon,inf)$

mnei:

integer array of dimension (4). mnei(1) = row dimension of $sE(\text{eps})-A(\text{eps})$

Description

Given a pencil $SE-A$ where matrix E is in column echelon form the function `fstair` computes according to the wishes of the user a unitary transformed pencil $QE(SEE-AE)ZE$ which is more or less similar to the generalized Schur form of the pencil $SE-A$. The function yields also part of the Kronecker structure of the given pencil.

Q, Z are the unitary matrices used to compute the pencil where E is in column echelon form (see `ereduc`)

See Also

`quaskro` , `ereduc`

Authors

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

Name

fullrf — full rank factorization

```
[Q,M,rk]=fullrf(A,[tol])
```

Parameters

A
real or complex matrix

tol
real number (threshold for rank determination)

Q,M
real or complex matrix

rk
integer (rank of A)

Description

Full rank factorization : `fullrf` returns Q and M such that $A = Q \cdot M$ with $\text{range}(Q) = \text{range}(A)$ and $\text{ker}(M) = \text{ker}(A)$, Q full column rank , M full row rank, $\text{rk} = \text{rank}(A) = \text{#columns}(Q) = \text{#rows}(M)$.

`tol` is an optional real parameter (default value is `sqrt(%eps)`). The rank `rk` of A is defined as the number of singular values larger than $\text{norm}(A) \cdot \text{tol}$.

If A is symmetric, `fullrf` returns $M=Q'$.

Examples

```
A=rand(5,2)*rand(2,5);
[Q,M]=fullrf(A);
norm(Q*M-A,1)
[X,d]=rowcomp(A);Y=X';
svd([A,Y(:,1:d),Q])           //span(Q) = span(A) = span(Y(:,1:2))
```

See Also

`svd` , `qr` , `fullrfk` , `rowcomp` , `colcomp`

Authors

F.D.;

Name

fullrfk — full rank factorization of A^k

```
[Bk,Ck]=fullrfk(A,k)
```

Parameters

A
real or complex matrix

k
integer

Bk,Ck
real or complex matrices

Description

This function computes the full rank factorization of A^k i.e. $B_k * C_k = A^k$ where B_k is full column rank and C_k full row rank. One has $\text{range}(B_k) = \text{range}(A^k)$ and $\text{ker}(C_k) = \text{ker}(A^k)$.

For $k=1$, fullrfk is equivalent to fullrf.

Examples

```
A=rand(5,2)*rand(2,5); [Bk,Ck]=fullrfk(A,3);  
norm(Bk*Ck-A^3,1)
```

See Also

fullrf , range

Authors

F.D (1990);

Name

genmarkov — generates random markov matrix with recurrent and transient classes

```
M=genmarkov(rec,tr)
M=genmarkov(rec,tr,flag)
```

Parameters

rec
integer row vector (its dimension is the number of recurrent classes).

tr
integer (number of transient states)

M
real Markov matrix. Sum of entries in each row should add to one.

flag
string 'perm'. If given, a random permutation of the states is done.

Description

Returns in M a random Markov transition probability matrix with `size(rec,1)` recurrent classes with `rec(1), ... rec($)` entries respectively and tr transient states.

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

See Also

classmarkov , eigenmarkov

Name

givens — Givens transformation

```
U=givens(xy)
U=givens(x,y)
[U,c]=givens(xy)
[U,c]=givens(x,y)
```

Parameters

`x,y`
two real or complex numbers

`xy`
real or complex size 2 column vector

`U`
2x2 unitary matrix

`c`
real or complex size 2 column vector

Description

`U = givens(x, y)` or `U = givens(xy)` with `xy = [x;y]` returns a 2x2 unitary matrix `U` such that:

$U^*xy = [r; 0] = c.$

Note that `givens(x, y)` and `givens([x;y])` are equivalent.

Examples

```
A=[3,4;5,6];
U=givens(A(:,1));
U*A
```

See Also

`qr`

Name

glever — inverse of matrix pencil

```
[Bfs,Bis,chis]=glever(E,A [,s])
```

Parameters

E, A
two real square matrices of same dimensions

s
character string (default value 's')

Bfs,Bis
two polynomial matrices

chis
polynomial

Description

Computation of
 $(s^*E - A)^{-1}$
by generalized Leverrier's algorithm for a matrix pencil.

```
(s^*E-A)^-1 = (Bfs/chis) - Bis.
```

chis = characteristic polynomial (up to a multiplicative constant).

Bfs = numerator polynomial matrix.

Bis = polynomial matrix (- expansion of $(s^*E - A)^{-1}$ at infinity).

Note the - sign before Bis.

Caution

This function uses `cleanup` to simplify Bfs, Bis and chis.

Examples

```
s=%s;F=[-1,s,0,0;0,-1,0,0;0,0,s-2,0;0,0,0,s-1];  
[Bfs,Bis,chis]=glever(F)  
inv(F)-((Bfs/chis) - Bis)
```

See Also

rowshuff , det , invr , coffg , pengan , penlaur

Authors

F. D. (1988)

Name

gschur — generalized Schur form (obsolete).

```
[As,Es]=gschur(A,E)
[As,Es,Q,Z]=gschur(A,E)
[As,Es,Z,dim] = gschur(A,E,flag)
[As,Es,Z,dim]= gschur(A,E,extern)
```

Description

This function is obsolete and is now included in the `schur` function. In most cases the `gschur` function will still work as before, but it will be removed in the future release.

The first three syntaxes can be replaced by

```
[As,Es]=schur(A,E)
[As,Es,Q,Z]=schur(A,E);Q=Q' //NOTE THE TRANPOSITION HERE
[As,Es,Z,dim] = schur(A,E,flag)
```

The last syntax requires little more adaptations:

if
extern is a scilab function the new calling sequence should be `[As,Es,Z,dim]=schur(A,E,Nextern)` with `Nextern` defined as follow:

```
function t=Nextern(R)
if R(2)==0 then
    t=extern([1,R(1),R(3)])==1
else
    c=(R(1)+%i*R(2))/R(3)
    t=extern([2,real(c+c'),real(c*c')])==1
end
endfunction
```

if
extern is the name of an external function coded in Fortran or C the new calling sequence should be `[As,Es,Z,dim]= schur(A,E, 'nextern' )` with `nextern` defined as follow:

```
logical function nextern(ar,ai,beta)
double precision ar,ai,beta
integer r,extern
if (ai.eq.0.0d0) then
    r=extern(1,ar,beta,0.0d0,0.0d0)
else
    r=extern(2,0.0d0,0.0d0,2.0d0*ar,ar*ar+ai*ai)
endif
nextern=r.eq.1
end
```

See Also

external , schur

Name

gspec — eigenvalues of matrix pencil (obsolete)

```
[a1, be]=gspec(A, E)
[a1, be, Z]=gspec(A, E)
```

Description

This function is now included in the `spec` function. the calling syntax must be replaced by

```
[a1, be]=spec(A, E)
[a1, be, Z]=spec(A, E)
```

See Also

`spec`

Name

hess — Hessenberg form

```
H = hess(A)
[U, H] = hess(A)
```

Parameters

A
real or complex square matrix

H
real or complex square matrix

U
orthogonal or unitary square matrix

Description

`[U, H] = hess(A)` produces a unitary matrix `U` and a Hessenberg matrix `H` so that $A = U^* H U^T$ and $U^T U = \text{Identity}$. By itself, `hess(A)` returns `H`.

The Hessenberg form of a matrix is zero below the first subdiagonal. If the matrix is symmetric or Hermitian, the form is tridiagonal.

References

hess function is based on the Lapack routines DGEHRD, DORGHR for real matrices and ZGEHRD, ZORGHR for the complex case.

Examples

```
A=rand(3,3); [U,H]=hess(A);
and( abs(U'*U'-A)<1.d-10 )
```

See Also

qr , contr , schur

Used Functions

hess function is based on the Lapack routines DGEHRD, DORGHR for real matrices and ZGEHRD, ZORGHR for the complex case.

Name

householder — Householder orthogonal reflexion matrix

```
u=householder(v [,w])
```

Parameters

- v**
real or complex column vector
- w**
real or complex column vector with same size as **v**. Default value is `eye(v)`
- u**
real or complex column vector

Description

given 2 column vectors **v**, **w** of same size, `householder(v,w)` returns a unitary column vector **u**, such that $(\text{eye}() - 2*u*u')*v$ is proportional to **w**. $(\text{eye}() - 2*u*u')$ is the orthogonal Householder reflexion matrix .

w default value is `eye(v)`. In this case vector $(\text{eye}() - 2*u*u')*v$ is the vector `eye(v)*norm(v)`.

See Also

`qr` , `givens`

Name

im_inv — inverse image

```
[X,dim]=im_inv(A,B [,tol])  
[X,dim,Y]=im_inv(A,B, [,tol])
```

Parameters

A,B

two real or complex matrices with equal number of columns

X

orthogonal or unitary square matrix of order equal to the number of columns of A

dim

integer (dimension of subspace)

Y

orthogonal matrix of order equal to the number of rows of A and B.

Description

`[X,dim]=im_inv(A,B)` computes $(A^{-1})(B)$ i.e vectors whose image through A are in $\text{range}(B)$

The `dim` first columns of X span $(A^{-1})(B)$

`tol` is a threshold used to test if subspace inclusion; default value is `tol = 100*%eps`. If Y is returned, then $[Y^*A^*X, Y^*B]$ is partitioned as follows: $[A_{11}, A_{12}; 0, A_{22}], [B_1; 0]$

where B_1 has full row rank (equals $\text{rank}(B)$) and A_{22} has full column rank and has `dim` columns.

Examples

```
A=[rand(2,5);zeros(3,4),rand(3,1)];B=[[1,1;1,1];zeros(3,2)];  
W=rand(5,5);A=W*A;B=W*B;  
[X,dim]=im_inv(A,B)  
svd([A*X(:,1:dim),B]) //vectors A*X(:,1:dim) belong to range(B)  
[X,dim,Y]=im_inv(A,B);[Y*A*X,Y*B]
```

See Also

rowcomp , spaninter , spanplus , linsolve

Authors

F. Delebecque INRIA

Name

inv — matrix inverse

`inv(X)`

Parameters

X

real or complex square matrix, polynomial matrix, rational matrix in transfer or state-space representation.

Description

`inv(X)` is the inverse of the square matrix X. A warning message is printed if X is badly scaled or nearly singular.

For polynomial matrices or rational matrices in transfer representation, `inv(X)` is equivalent to `invr(X)`.

For linear systems in state-space representation (`syslin` list), `invr(X)` is equivalent to `invsyslin(X)`.

References

`inv` function for matrices of numbers is based on the Lapack routines DGETRF, DGETRI for real matrices and ZGETRF, ZGETRI for the complex case. For polynomial matrix and rational function matrix `inv` is based on the `invr` Scilab function.

Examples

```
A=rand(3,3);inv(A)*A

x=poly(0,'x');
A=[x,1,x;x^2,2,1+x;1,2,3];inv(A)*A

A=[1/x,2;2+x,2/(1+x)]
inv(A)*A

A=ssrand(2,2,3);
W=inv(A)*A
clean(ss2tf(W))
```

See Also

`slash` , `backslash` , `pinv` , `qr` , `lufact` , `lusolve` , `invr` , `coff` , `coffg`

Name

kernel — kernel, nullspace

```
w=kernel(A [,tol,[,flag]])
```

Parameters

A
full real or complex matrix or real sparse matrix

flag
character string 'svd' (default) or 'qr'

tol
real number

W
full column rank matrix

Description

`w=kernel(A)` returns the kernel (nullspace) of A. If A has full column rank then an empty matrix [] is returned.

`flag` and `tol` are optional parameters: `flag` = 'qr' or 'svd' (default is 'svd').

`tol` = tolerance parameter (of order %eps as default value).

Examples

```
A=rand(3,1)*rand(1,3);  
A*kernel(A)  
A=sparse(A);  
clean(A*kernel(A))
```

See Also

`colcomp` , `fullrf` , `fullrfk` , `linsolve`

Authors

F.D.;

Name

kroneck — Kronecker form of matrix pencil

```
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(E,A)
```

Parameters

F
real matrix pencil $F = sE - A$

E,A
two real matrices of same dimensions

Q,Z
two square orthogonal matrices

Qd,Zd
two vectors of integers

numbeps,numeta
two vectors of integers

Description

Kronecker form of matrix pencil: **kroneck** computes two orthogonal matrices **Q**, **Z** which put the pencil $F = sE - A$ into upper-triangular form:

$$Q(sE - A)Z = \begin{array}{c|c|c|c|c} & sE(\text{eps}) - A(\text{eps}) & \text{X} & \text{X} & \text{X} \\ \hline & 0 & sE(\text{inf}) - A(\text{inf}) & \text{X} & \text{X} \\ \hline & 0 & 0 & sE(\text{f}) - A(\text{f}) & \text{X} \\ \hline & 0 & 0 & 0 & sE(\text{eta}) - A(\text{eta}) \end{array}$$

The dimensions of the four blocks are given by:

$\text{eps} = Qd(1) \times Zd(1)$, $\text{inf} = Qd(2) \times Zd(2)$, $\text{f} = Qd(3) \times Zd(3)$, $\text{eta} = Qd(4) \times Zd(4)$

The **inf** block contains the infinite modes of the pencil.

The **f** block contains the finite modes of the pencil

The structure of epsilon and eta blocks are given by:

$\text{numbeps}(1) = \#$ of eps blocks of size 0×1

$\text{numbeps}(2) = \#$ of eps blocks of size 1×2

$\text{numbeps}(3) = \#$ of eps blocks of size 2×3 etc...

$\text{numbeta}(1) = \#$ of eta blocks of size 1×0

$\text{numbeta}(2) = \#$ of eta blocks of size 2×1

numbeta(3) = # of eta blocks of size 3 x 2 etc...

The code is taken from T. Beelen (Slicot-WGS group).

Examples

```
F=randpencil([1,1,2],[2,3],[-1,3,1],[0,3]);
Q=rand(17,17);Z=rand(18,18);F=Q*F*Z;
//random pencil with eps1=1,eps2=1,eps3=1; 2 J-blocks @ infty
//with dimensions 2 and 3
//3 finite eigenvalues at -1,3,1 and eta1=0,eta2=3
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
[Qd(1),Zd(1)] //eps. part is sum(epsi) x (sum(epsi) + number of epsi)
[Qd(2),Zd(2)] //infinity part
[Qd(3),Zd(3)] //finite part
[Qd(4),Zd(4)] //eta part is (sum(etai) + number(eta1)) x sum(etai)
numbeps
numbeta
```

See Also

gschur , gspec , systmat , pengan , randpencil , trzeros

Name

linsolve — linear equation solver

```
[x0, kerA]=linsolve(A,b [,x0])
```

Parameters

A
a $n_a \times m_a$ real matrix (possibly sparse)

b
a $n_a \times 1$ vector (same row dimension as *A*)

x0
a real vector

kerA
a $m_a \times k$ real matrix

Description

`linsolve` computes all the solutions to $A \cdot x + b = 0$.

x0 is a particular solution (if any) and `kerA` = nullspace of *A*. Any $x = x_0 + \text{kerA} \cdot w$ with arbitrary *w* satisfies $A \cdot x + b = 0$.

If compatible *x0* is given on entry, *x0* is returned. If not a compatible *x0*, if any, is returned.

Examples

```
A=rand(5,3)*rand(3,8);
b=A*ones(8,1);[x,kerA]=linsolve(A,b);A*x+b //compatible b
b=ones(5,1);[x,kerA]=linsolve(A,b);A*x+b //uncompatible b
A=rand(5,5);[x,kerA]=linsolve(A,b), -inv(A)*b //x is unique

// Benchmark with other linear sparse solver:
[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa")

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

See Also

`inv` , `pinv` , `colcomp` , `im_inv` , `umfpack` , `backslash`

Name

lsq — linear least square problems.

```
X=lsq(A,B [,tol])
```

Parameters

A

Real or complex (m x n) matrix

B

real or complex (m x p) matrix

tol

positive scalar, used to determine the effective rank of A (defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number $\leq 1/\text{tol}$. The tol default value is set to `sqrt(%eps)`).

X

real or complex (n x p) matrix

Description

`X=lsq(A,B)` computes the minimum norm least square solution of the equation $A^*X=B$, while `X=A\B` compute a least square solution with at at most `rank(A)` nonzero components per column.

References

lsq function is based on the LAPack functions DGELSY for real matrices and ZGELSY for complex matrices.

Examples

```
//Build the data
x=(1:10)';

y1=3*x+4.5+3*rand(x,'normal');
y2=1.8*x+0.5+2*rand(x,'normal');
plot2d(x,[y1,y2],[-2,-3])
//Find the linear regression
A=[x,ones(x)];B=[y1,y2];
X=lsq(A,B);

y1e=X(1,1)*x+X(2,1);
y2e=X(1,2)*x+X(2,2);
plot2d(x,[y1e,y2e],[2,3])

//Difference between lsq(A,b) and A\b
A=rand(4,2)*rand(2,3);//a rank 2 matrix
b=rand(4,1);
X1=lsq(A,b)
X2=A\b
[A*X1-b, A*X2-b] //the residuals are the same
```

See Also

backslash , inv , pinv , rank

Name

lu — LU factorization with pivoting

```
[L,U]=lu(A)
[L,U,E]=lu(A)
```

Parameters

A
real or complex matrix (m x n).

L
real or complex matrices (m x min(m,n)).

U
real or complex matrices (min(m,n) x n).

E
a (n x n) permutation matrix.

Description

`[L,U]=lu(A)` produces two matrices L and U such that $A = L*U$ with U upper triangular and L a general matrix without any particular structure. In fact, the matrix A is factored as $E*A=B*U$ where the matrix B is lower triangular and the matrix L is computed from $L=E'*B$.

If A has rank k, rows k+1 to n of U are zero.

`[L,U,E]=lu(A)` produces three matrices L, U and E such that $E*A = L*U$ with U upper triangular and $E*L$ lower triangular for a permutation matrix E.

If A is a real matrix, using the function `lufact` and `luget` it is possible to obtain the permutation matrices and also when A is not full rank the column compression of the matrix L.

Example #1

In the following example, we create the Hilbert matrix of size 4 and factor it with $A=LU$. Notice that the matrix L is not lower triangular. To get a lower triangular L matrix, we should have given the output argument E to Scilab.

```
a = testmatrix("hilb",4);
[l,u]=lu(a)
norm(l*u-a)
```

Example #2

In the following example, we create the Hilbert matrix of size 4 and factor it with $EA=LU$. Notice that the matrix L is lower triangular.

```
a = testmatrix("hilb",4);
[l,u,e]=lu(a)
norm(l*u-e*a)
```

Example #3

The following example shows how to use the `lu` and `luget` functions.

```
a=rand(4,4);
[l,u]=lu(a)
norm(l*u-a)

[h,rk]=lufact(sparse(a))
[P,L,U,Q]=luget(h);
ludel(h)
P=full(P);
L=full(L);
U=full(U);
Q=full(Q);
norm(P*L*U*Q-a)
```

See Also

`lu` , `luget` , `lusolve` , `qr` , `svd`

Used Functions

`lu` decompositions are based on the Lapack routines `DGETRF` for real matrices and `ZGETRF` for the complex case.

Name

lyap — Lyapunov equation

```
[X]=lyap(A,C,'c')  
[X]=lyap(A,C,'d')
```

Parameters

A, C
real square matrices, C must be symmetric

Description

`X = lyap(A,C,flag)` solves the continuous time or discrete time matrix Lyapunov matrix equation:

$$\begin{array}{ll} A' * X + X * A = C & (\text{flag} = 'c') \\ A' * X * A - X = C & (\text{flag} = 'd') \end{array}$$

Note that a unique solution exist if and only if an eigenvalue of A is not an eigenvalue of -A (flag='c') or 1 over an eigenvalue of A (flag='d').

Examples

```
A=rand(4,4);C=rand(A);C=C+C';  
X=lyap(A,C,'c');  
A'*X + X*A -C  
X=lyap(A,C,'d');  
A'*X*A - X -C
```

See Also

sylv , ctr_gram , obs_gram

Name

nlev — Leverrier's algorithm

```
[num,den]=nlev(A,z [,rmax])
```

Parameters

A
real square matrix

z
character string

rmax
optional parameter (see `bdiag`)

Description

`[num,den]=nlev(A,z [,rmax])` computes $(z \cdot \text{eye}() - A)^{-1}$

by block diagonalization of A followed by Leverrier's algorithm on each block.

This algorithm is better than the usual leverrier algorithm but still not perfect!

Examples

```
A=rand(3,3);x=poly(0,'x');  
[NUM,den]=nlev(A,'x')  
clean(den-poly(A,'x'))  
clean(NUM/den-inv(x*eye()-A))
```

See Also

`coff` , `coffg` , `glever` , `ss2tf`

Authors

F. Delebecque., S. Steer INRIA;

Name

orth — orthogonal basis

```
Q=orth(A)
```

Parameters

A
real or complex matrix

Q
real or complex matrix

Description

`Q=orth(A)` returns `Q`, an orthogonal basis for the span of `A`. $\text{Range}(Q) = \text{Range}(A)$ and $Q' * Q = \text{eye}$.

The number of columns of `Q` is the rank of `A` as determined by the QR algorithm.

Examples

```
A=rand(5,3)*rand(3,4);  
[X,dim]=rowcomp(A);X=X';  
svd([orth(A),X(:,1:dim)])
```

See Also

qr , rowcomp , colcomp , range

Name

pbig — eigen-projection

```
[Q,M]=pbig(A,thres,flag)
```

Parameters

A
real square matrix

thres
real number

flag
character string ('c' or 'd')

Q,M
real matrices

Description

Projection on eigen-subspace associated with eigenvalues with real part $\geq$ **thres** (**flag**='c') or with magnitude $\geq$ **thres** (**flag**='d').

The projection is defined by Q^*M , Q is full column rank, M is full row rank and $M^*Q=eye$.

If **flag**='c', the eigenvalues of M^*A^*Q = eigenvalues of A with real part $\geq$ **thres**.

If **flag**='d', the eigenvalues of M^*A^*Q = eigenvalues of A with magnitude $\geq$ **thres**.

If **flag**='c' and if $[Q1, M1]$ = full rank factorization (**fullrf**) of $eye() - Q^*M$ then eigenvalues of $M1^*A^*Q1$ = eigenvalues of A with real part $<$ **thres**.

If **flag**='d' and if $[Q1, M1]$ = full rank factorization (**fullrf**) of $eye() - Q^*M$ then eigenvalues of $M1^*A^*Q1$ = eigenvalues of A with magnitude $<$ **thres**.

Examples

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=pbig(A,1.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

See Also

psmall , projspec , fullrf , schur

Authors

F. D. (1988); ;

Used Functions

pbig is based on the ordered schur form (scilab function **schur**).

Name

pencan — canonical form of matrix pencil

```
[Q,M,i1]=pencan(Fs)
[Q,M,i1]=pencan(E,A)
```

Parameters

Fs
a regular pencil $s^*E - A$

E,A
two real square matrices

Q,M
two non-singular real matrices

i1
integer

Description

Given the regular pencil $Fs=s^*E-A$, `pencan` returns matrices Q and M such than $M^*(s^*E-A)*Q$ is in "canonical" form.

M^*E*Q is a block matrix

```
[I, 0;
 0, N]
```

with N nilpotent and $i1$ = size of the I matrix above.

M^*A*Q is a block matrix:

```
[Ar, 0;
 0, I]
```

Examples

```
F=randpencil([], [1,2], [1,2,3], []);
F=rand(6,6)*F*rand(6,6);
[Q,M,i1]=pencan(F);
W=clean(M*F*Q)
roots(det(W(1:i1,1:i1)))
det(W($-2:$,$-2:$))
```

See Also

`glever`, `penlaur`, `rowshuff`

Authors

F. D.; ;

Name

penlaur — Laurent coefficients of matrix pencil

```
[Si,Pi,Di,order]=penlaur(Fs)
[Si,Pi,Di,order]=penlaur(E,A)
```

Parameters

Fs
a regular pencil $S^*E - A$

E, A
two real square matrices

Si,Pi,Di
three real square matrices

order
integer

Description

penlaur computes the first Laurent coefficients of $(S^*E - A)^{-1}$ at infinity.

$(S^*E - A)^{-1} = \dots + Si/s - Pi - S^*Di + \dots$ at $s = \text{infinity}$.

order = order of the singularity (order=index-1).

The matrix pencil $FS = S^*E - A$ should be invertible.

For a index-zero pencil, $Pi, Di, \dots$ are zero and $Si = \text{inv}(E)$.

For a index-one pencil (order=0), $Di = 0$.

For higher-index pencils, the terms $-s^2 Di(2), -s^3 Di(3), \dots$ are given by:

$Di(2) = Di^*A^*Di, Di(3) = Di^*A^*Di^*A^*Di$ (up to $Di(\text{order})$).

Remark

Experimental version: troubles when bad conditioning of $S^*E - A$

Examples

```
F=randpencil([], [1,2], [1,2,3], []);
F=rand(6,6)*F*rand(6,6); [E,A]=pen2ea(F);
[Si,Pi,Di]=penlaur(F);
[Bfs,Bis,chis]=glever(F);
norm(coeff(Bis,1)-Di,1)
```

See Also

glever , pengan , rowshuff

Authors

F. Delebecque INRIA(1988,1990) ;

Name

`pinv` — pseudoinverse

```
pinv(A,[tol])
```

Parameters

`A`
real or complex matrix

`tol`
real number

Description

$X = \text{pinv}(A)$ produces a matrix X of the same dimensions as A' such that:

$A^*X^*A = A$, $X^*A^*X = X$ and both A^*X and X^*A are Hermitian .

The computation is based on SVD and any singular values lower than a tolerance are treated as zero: this tolerance is accessed by `X=pinv(A,tol)`.

Examples

```
A=rand(5,2)*rand(2,4);  
norm(A*pinv(A)*A-A,1)
```

See Also

`rank` , `svd` , `qr`

Used Functions

`pinv` function is based on the singular value decomposition (Scilab function `svd`).

Name

polar — polar form

```
[Ro, Theta]=polar(A)
```

Parameters

A
real or complex square matrix

Ro,
real matrix

Theta,
real or complex matrix

Description

`[Ro, Theta]=polar(A)` returns the polar form of A i.e. $A=Ro \cdot \exp(i \cdot Theta)$ Ro symmetric ≥ 0 and Theta hermitian ≥ 0 .

Examples

```
A=rand(5,5);  
[Ro, Theta]=polar(A);  
norm(A-Ro*expm(i*Theta),1)
```

See Also

`expm` , `svd`

Authors

F. Delebecque INRIA; ;

Name

proj — projection

```
P = proj(X1,X2)
```

Parameters

X1,X2
two real matrices with equal number of columns

P
real projection matrix ($P^2=P$)

Description

P is the projection on X2 parallel to X1.

Examples

```
X1=rand(5,2);X2=rand(5,3);  
P=proj(X1,X2);  
norm(P^2-P,1)  
trace(P)      // This is dim(X2)  
[Q,M]=fullrf(P);  
svd([Q,X2])   // span(Q) = span(X2)
```

See Also

projspec , orth , fullrf

Authors

F. D.; ;

Name

projspec — spectral operators

```
[S,P,D,i]=projspec(A)
```

Parameters

A
square matrix

S, P, D
square matrices

i
integer (index of the zero eigenvalue of A).

Description

Spectral characteristics of A at 0.

S = reduced resolvent at 0 (S = -Drazin_inverse(A)).

P = spectral projection at 0.

D = nilpotent operator at 0.

index = index of the 0 eigenvalue.

One has $(s \cdot \text{eye}() - A)^{-1} = D^{(i-1)}/s^i + \dots + D/s^2 + P/s - S - s \cdot S^2 - \dots$ around the singularity $s=0$.

Examples

```
deff('j=jdrn(n)', 'j=zeros(n,n);for k=1:n-1;j(k,k+1)=1;end')
A=sysdiag(jdrn(3),jdrn(2),rand(2,2));X=rand(7,7);
A=X*A*inv(X);
[S,P,D,index]=projspec(A);
index    //size of J-block
trace(P)  //sum of dimensions of J-blocks
A*S-(eye()-P)
norm(D^index,1)
```

See Also

coff

Authors

F. D.; ;

Name

psmall — spectral projection

```
[Q,M]=psmall(A,thres,flag)
```

Parameters

A
real square matrix

thres
real number

flag
character string ('c' or 'd')

Q,M
real matrices

Description

Projection on eigen-subspace associated with eigenvalues with real part < thres (flag='c') or with modulus < thres (flag='d').

The projection is defined by Q^*M , Q is full column rank, M is full row rank and $M^*Q=eye$.

If flag='c', the eigenvalues of M^*A^*Q = eigenvalues of A with real part < thres.

If flag='d', the eigenvalues of M^*A^*Q = eigenvalues of A with magnitude < thres.

If flag='c' and if $[Q1, M1]$ = full rank factorization (fullrf) of $eye() - Q^*M$ then eigenvalues of $M1^*A^*Q1$ = eigenvalues of A with real part $\geq$ thres.

If flag='d' and if $[Q1, M1]$ = full rank factorization (fullrf) of $eye() - Q^*M$ then eigenvalues of $M1^*A^*Q1$ = eigenvalues of A with magnitude $\geq$ thres.

Examples

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=psmall(A,2.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

See Also

pbig , proj , projspec

Authors

F. Delebecque INRIA. (1988);

Used Functions

This function is based on the ordered schur form (scilab function schur).

Name

qr — QR decomposition

```
[Q,R]=qr(X [, "e"])\n[Q,R,E]=qr(X [, "e"])\n[Q,R,rk,E]=qr(X [, tol])
```

Parameters

X
real or complex matrix

tol
nonnegative real number

Q
square orthogonal or unitary matrix

R
matrix with same dimensions as X

E
permutation matrix

rk
integer (QR-rank of X)

Description

`[Q,R] = qr(X)`
produces an upper triangular matrix R of the same dimension as X and an orthogonal (unitary in the complex case) matrix Q so that $X = Q \cdot R$. `[Q,R] = qr(X, "e")` produces an "economy size": If X is m-by-n with $m > n$, then only the first n columns of Q are computed as well as the first n rows of R.

From $Q \cdot R = X$, it follows that the kth column of the matrix X, is expressed as a linear combination of the k first columns of Q (with coefficients $R(1, k), \dots, R(k, k)$). The k first columns of Q make an orthogonal basis of the subspace spanned by the k first columns of X. If column k of X (i.e. $X(:, k)$) is a linear combination of the first p columns of X, then the entries $R(p+1, k), \dots, R(k, k)$ are zero. In this situation, R is upper trapezoidal. If X has rank rk, rows $R(rk+1, :), R(rk+2, :), \dots$ are zeros.

`[Q,R,E] = qr(X)`
produces a (column) permutation matrix E, an upper triangular R with decreasing diagonal elements and an orthogonal (or unitary) Q so that $X \cdot E = Q \cdot R$. If rk is the rank of X, the rk first entries along the main diagonal of R, i.e. $R(1, 1), R(2, 2), \dots, R(rk, rk)$ are all different from zero. `[Q,R,E] = qr(X, "e")` produces an "economy size": If X is m-by-n with $m > n$, then only the first n columns of Q are computed as well as the first n rows of R.

`[Q,R,rk,E] = qr(X, tol)`
returns rk = rank estimate of X i.e. rk is the number of diagonal elements in R which are larger than a given threshold tol.

`[Q,R,rk,E] = qr(X)`
returns rk = rank estimate of X i.e. rk is the number of diagonal elements in R which are larger than $\text{tol} = R(1, 1) * \%eps * \max(\text{size}(R))$. See `rankqr` for a rank revealing QR factorization, using the condition number of R.

Examples

```
// QR factorization, generic case
// X is tall (full rank)
X=rand(5,2);[Q,R]=qr(X); [Q'*X R]

//X is fat (full rank)
X=rand(2,3);[Q,R]=qr(X); [Q'*X R]

//Column 4 of X is a linear combination of columns 1 and 2:
X=rand(8,5);X(:,4)=X(:,1)+X(:,2); [Q,R]=qr(X); R, R(:,4)

//X has rank 2, rows 3 to $ of R are zero:
X=rand(8,2)*rand(2,5);[Q,R]=qr(X); R

//Evaluating the rank rk: column pivoting ==> rk first
//diagonal entries of R are non zero :
A=rand(5,2)*rand(2,5);
[Q,R,rk,E] = qr(A,1.d-10);
norm(Q'*A-R)
svd([A,Q(:,1:rk)]) //span(A) =span(Q(:,1:rk))
```

See Also

rankqr , rank , svd , rowcomp , colcomp

Used Functions

qr decomposition is based the Lapack routines DGEQRF, DGEQPF, DORGQR for the real matrices and ZGEQRF, ZGEQPF, ZORGQR for the complex case.

Name

quaskro — quasi-Kronecker form

```
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F,tol)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A,tol)
```

Parameters

F
real matrix pencil $F=s \ast E-A$ ($s=\text{poly}(0, 's')$)

E,A
two real matrices of same dimensions

tol
a real number (tolerance, default value=1.d-10)

Q,Z
two square orthogonal matrices

Qd,Zd
two vectors of integers

numbeps
vector of integers

Description

Quasi-Kronecker form of matrix pencil: **quaskro** computes two orthogonal matrices **Q**, **Z** which put the pencil $F=s \ast E -A$ into upper-triangular form:

$$Q(sE-A)Z = \begin{array}{c|c|c|c} & sE(\text{eps})-A(\text{eps}) & X & X \\ \hline & 0 & sE(\text{inf})-A(\text{inf}) & X \\ \hline & \text{=====} & \text{=====} & \text{=====} \\ & 0 & sE(r)-A(r) & \end{array}$$

The dimensions of the blocks are given by:

$\text{eps}=\text{Qd}(1) \times \text{Zd}(1)$, $\text{inf}=\text{Qd}(2) \times \text{Zd}(2)$, $r = \text{Qd}(3) \times \text{Zd}(3)$

The **inf** block contains the infinite modes of the pencil.

The **f** block contains the finite modes of the pencil

The structure of epsilon blocks are given by:

$\text{numbeps}(1) = \#$ of eps blocks of size 0 x 1

$\text{numbeps}(2) = \#$ of eps blocks of size 1 x 2

$\text{numbeps}(3) = \#$ of eps blocks of size 2 x 3 etc...

The complete (four blocks) Kronecker form is given by the function `kroneck` which calls `quaskro` on the (pertransposed) pencil $SE(r) - A(r)$.

The code is taken from T. Beelen

See Also

`kroneck` , `gschur` , `gspec`

Name

randpencil — random pencil

```
F=randpencil(eps,infi,fin,eta)
```

Parameters

eps

vector of integers

infi

vector of integers

fin

real vector, or monic polynomial, or vector of monic polynomial

eta

vector of integers

F

real matrix pencil $F=sE-A$ ($s=\text{poly}(0, 's')$)

Description

Utility function. `F=randpencil(eps,infi,fin,eta)` returns a random pencil F with given Kronecker structure. The structure is given by: `eps=[eps1,...,epsk]`: structure of epsilon blocks (size $\text{eps1} \times (\text{eps1}+1), \dots$) `fin=[l1,...,ln]` set of finite eigenvalues (assumed real) (possibly []) `infi=[k1,...,kp]` size of J-blocks at infinity $k_i \geq 1$ (`infi=[]` if no J blocks). `eta=[eta1,...,etap]`: structure of eta blocks (size $\text{eta1}+1 \times \text{eta1}, \dots$)

`epsi`'s should be ≥ 0 , `etai`'s should be ≥ 0 , `infi`'s should be ≥ 1 .

If `fin` is a (monic) polynomial, the finite block admits the roots of `fin` as eigenvalues.

If `fin` is a vector of polynomial, they are the finite elementary divisors of F i.e. the roots of $p(i)$ are finite eigenvalues of F .

Examples

```
F=randpencil([0,1],[2],[-1,0,1],[3]);
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
Qd, Zd
s=poly(0,'s');
F=randpencil([], [1,2], s^3-2, []); //regular pencil
det(F)
```

See Also

kroneck , pengan , penlaur

Name

range — range (span) of A^k

```
[X,dim]=range(A,k)
```

Parameters

A
real square matrix

k
integer

X
orthonormal real matrix

dim
integer (dimension of subspace)

Description

Computation of Range A^k ; the first dim rows of X span the range of A^k . The last rows of X span the orthogonal complement of the range. $X \cdot X'$ is the Identity matrix

Examples

```
A=rand(4,2)*rand(2,4); // 4 column vectors, 2 independent.
[X,dim]=range(A,1);dim // compute the range

y1=A*rand(4,1); //a vector which is in the range of A
y2=rand(4,1); //a vector which is not in the range of A
norm(X(dim+1:$,:)*y1) //the last entries are zeros, y1 is in the range of A
norm(X(dim+1:$,:)*y2) //the last entries are not zeros

I=X(1:dim,:)' //I is a basis of the range
coeffs=X(1:dim,:)*y1 // components of y1 relative to the I basis

norm(I*coeffs-y1) //check
```

See Also

fullrfk , rowcomp

Authors

F. D. INRIA ;

Used Functions

The range function is based on the rowcomp function which uses the svd decomposition.

Name

rank — rank

```
[i]=rank(X)
[i]=rank(X,tol)
```

Parameters

X
real or complex matrix

tol
nonnegative real number

Description

`rank(X)` is the numerical rank of X i.e. the number of singular values of X that are larger than `norm(size(X),'inf') * norm(X) * %eps`.

`rank(X,tol)` is the number of singular values of X that are larger than `tol`.

Note that the default value of `tol` is proportional to `norm(X)`. As a consequence `rank([1.d-80,0;0,1.d-80])` is 2 !.

Examples

```
rank([1.d-80,0;0,1.d-80])
rank([1,0;0,1.d-80])
```

See Also

`svd` , `qr` , `rowcomp` , `colcomp` , `lu`

Name

rankqr — rank revealing QR factorization

```
[Q, R, JPV, RANK, SVAL] = rankqr(A, [RCOND, JPV])
```

Parameters

A

real or complex matrix

RCOND

real number used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number $< 1/\text{RCOND}$.

JPV

integer vector on entry, if JPV(i) is not 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPV(i) = k, then the i-th column of A*P was the k-th column of A.

RANK

the effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T1 in the complete orthogonal factorization of A.

SVAL

real vector with 3 components; The estimates of some of the singular values of the triangular factor R.

SVAL(1) is the largest singular value of R(1:RANK, 1:RANK);

SVAL(2) is the smallest singular value of R(1:RANK, 1:RANK);

SVAL(3) is the smallest singular value of R(1:RANK+1, 1:RANK+1), if $\text{RANK} < \min(M, N)$, or of R(1:RANK, 1:RANK), otherwise.

Description

To compute (optionally) a rank-revealing QR factorization of a real general M-by-N real or complex matrix A, which may be rank-deficient, and estimate its effective rank using incremental condition estimation.

The routine uses a QR factorization with column pivoting:

$$A * P = Q * R, \quad \text{where} \quad R = \begin{bmatrix} R11 & R12 \\ 0 & R22 \end{bmatrix},$$

with R11 defined as the largest leading submatrix whose estimated condition number is less than $1/\text{RCOND}$. The order of R11, RANK, is the effective rank of A.

If the triangular factorization is a rank-revealing one (which will be the case if the leading columns were well-conditioned), then SVAL(1) will also be an estimate for the largest singular value of A, and SVAL(2) and SVAL(3) will be estimates for the RANK-th and (RANK+1)-st singular values of A, respectively.

By examining these values, one can confirm that the rank is well defined with respect to the chosen value of RCOND. The ratio $\text{SVAL}(1)/\text{SVAL}(2)$ is an estimate of the condition number of R(1:RANK, 1:RANK).

Examples

```
A=rand(5,3)*rand(3,7);  
[Q,R,JPVT,RANK,SVAL]=rankqr(A,%eps)
```

See Also

qr , rank

Used Functions

Slicot library routines MB03OD, ZB03OD.

Name

rcond — inverse condition number

```
rcond(X)
```

Parameters

X
real or complex square matrix

Description

`rcond(X)` is an estimate for the reciprocal of the condition of X in the 1-norm.

If X is well conditioned, `rcond(X)` is close to 1. If not, `rcond(X)` is close to 0.

`[r,z]=rcond(X)` sets r to `rcond(X)` and returns z such that `norm(X*z,1) = r*norm(X,1)*norm(z,1)`

Thus, if rcond is small z is a vector in the kernel.

Examples

```
A=diag([1:10]);  
rcond(A)  
A(1,1)=0.000001;  
rcond(A)
```

See Also

`svd`, `cond`, `inv`

Name

rowcomp — row compression, range

```
[W,rk]=rowcomp(A [,flag [,tol]])
```

Parameters

- A**
real or complex matrix
- flag**
optionnal character string, with possible values 'svd' or 'qr'. The default value is 'svd'.
- tol**
optionnal real non negative number. The default value is `sqrt(%eps)*norm(A,1)`.
- W**
square non-singular matrix (change of basis)
- rk**
integer (rank of A)

Description

Row compression of A. $AC = W^*A$ is a row compressed matrix: i.e. $AC=[Af;0]$ with Af full row rank.

flag and **tol** are optional parameters: **flag**='qr' or 'svd' (default 'svd').

tol is a tolerance parameter.

The **rk** first columns of **W'** span the range of A.

The **rk** first (top) rows of **W** span the row range of A.

A non zero vector **x** belongs to $\text{range}(A)$ iff W^*x is row compressed in accordance with **AC** i.e the norm of its last components is small w.r.t its first components.

Examples

```
A=rand(5,2)*rand(2,4);           // 4 col. vectors, 2 independent.
[X,dim]=rowcomp(A);Xp=X';
svd([Xp(:,1:dim),A])              //span(A) = span(Xp(:,1:dim))
x=A*rand(4,1);                    //x belongs to span(A)
y=X*x
norm(y(dim+1:$))/norm(y(1:dim))   // small
```

See Also

colcomp , fullrf , fullrfk

Authors

F. D.; INRIA

Used Functions

The `rowcomp` function is based on the `svd` or `qr` decompositions.

Name

rowshuff — shuffle algorithm

```
[Ws,Fs1]=rowshuff(Fs, [alfa])
```

Parameters

Fs

square real pencil $FS = s^*E - A$

Ws

polynomial matrix

Fs1

square real pencil $F1s = s^*E1 - A1$ with E1 non-singular

alfa

real number (alfa = 0 is the default value)

Description

Shuffle algorithm: Given the pencil $FS=s^*E-A$, returns $Ws=W(s)$ (square polynomial matrix) such that:

$Fs1 = s^*E1 - A1 = W(s) * (s^*E - A)$ is a pencil with non singular E1 matrix.

This is possible iff the pencil $FS = s^*E - A$ is regular (i.e. invertible). The degree of Ws is equal to the index of the pencil.

The poles at infinity of FS are put to alfa and the zeros of Ws are at alfa.

Note that $(s^*E - A)^{-1} = (s^*E1 - A1)^{-1} * W(s) = (W(s) * (s^*E - A))^{-1} * W(s)$

Examples

```
F=randpencil([], [2], [1,2,3], []);
F=rand(5,5)*F*rand(5,5); // 5 x 5 regular pencil with 3 evals at 1,2,3
[Ws,F1]=rowshuff(F, -1);
[E1,A1]=pen2ea(F1);
svd(E1) //E1 non singular
roots(det(Ws))
clean(inv(F)-inv(F1)*Ws,1.d-7)
```

See Also

pencan , glever , penlaur

Authors

F. D.; ; ; ;

Name

rref — computes matrix row echelon form by lu transformations

```
R=rref(A)
```

Parameters

A
m x n matrix with scalar entries

R
m x n matrix,row echelon form of a

Description

rref computes the row echelon form of the given matrix by left lu decomposition. If ones need the transformation used just call `X=rref([A, eye(m, m)])` the row echelon form R is `X(:, 1:n)` and the left transformation L is given by `X(:, n+1:n+m)` such as `L*A=R`

Examples

```
A=[1 2;3 4;5 6];  
X=rref([A, eye(3,3)]);  
R=X(:, 1:2)  
L=X(:, 3:5);L*A
```

See Also

lu , qr

Name

schur — [ordered] Schur decomposition of matrix and pencils

```
[U,T] = schur(A)
[U,dim [,T] ]=schur(A,flag)
[U,dim [,T] ]=schur(A,extern1)

[As,Es [,Q,Z]]=schur(A,E)
[As,Es [,Q],Z,dim] = schur(A,E,flag)
[Z,dim] = schur(A,E,flag)
[As,Es [,Q],Z,dim]= schur(A,E,extern2)
[Z,dim]= schur(A,E,extern2)
```

Parameters

A
real or complex square matrix.

E
real or complex square matrix with same dimensions as **A**.

flag
character string ('c' or 'd')

extern1
an ``external", see below

extern2
an ``external", see below

U
orthogonal or unitary square matrix

Q
orthogonal or unitary square matrix

Z
orthogonal or unitary square matrix

T
upper triangular or quasi-triangular square matrix

As
upper triangular or quasi-triangular square matrix

Es
upper triangular square matrix

dim
integer

Description

Schur forms, ordered Schur forms of matrices and pencils

MATRIX SCHUR FORM

Usual schur form:

$[U, T] = \text{schur}(A)$ produces a Schur matrix T and a unitary matrix U so that $A = U^* T U$ and $U^* U = \text{eye}(U)$. By itself, $\text{schur}(A)$ returns T . If A is complex, the Complex

Schur Form is returned in matrix T. The Complex Schur Form is upper triangular with the eigenvalues of A on the diagonal. If A is real, the Real Schur Form is returned. The Real Schur Form has the real eigenvalues on the diagonal and the complex eigenvalues in 2-by-2 blocks on the diagonal.

Ordered Schur forms

`[U, dim]=schur(A, 'c')` returns a unitary matrix U which transforms A into schur form. In addition, the dim first columns of U make a basis of the eigenspace of A associated with eigenvalues with negative real parts (stable "continuous time" eigenspace).

`[U, dim]=schur(A, 'd')` returns a unitary matrix U which transforms A into schur form. In addition, the dim first columns of U span a basis of the eigenspace of A associated with eigenvalues with magnitude lower than 1 (stable "discrete time" eigenspace).

`[U, dim]=schur(A, extern1)` returns a unitary matrix U which transforms A into schur form. In addition, the dim first columns of U span a basis of the eigenspace of A associated with the eigenvalues which are selected by the external function `extern1` (see external for details). This external can be described by a Scilab function or by C or Fortran procedure:

a Scilab function

If `extern1` is described by a Scilab function, it should have the following calling sequence: `s=extern1(Ev)`, where Ev is an eigenvalue and s a boolean.

a C or Fortran procedure

If `extern1` is described by a C or Fortran function it should have the following calling sequence: `int extern1(double *EvR, double *EvI)` where EvR and EvI are eigenvalue real and complex parts. a true or non zero returned value stands for selected eigenvalue.

PENCIL SCHUR FORMS

Usual Pencil Schur form

`[As, Es] = schur(A, E)` produces a quasi triangular AS matrix and a triangular ES matrix which are the generalized Schur form of the pair A, E.

`[As, Es, Q, Z] = schur(A, E)` returns in addition two unitary matrices Q and Z such that $As=Q' * A * Z$ and $Es=Q' * E * Z$.

Ordered Schur forms:

`[As, Es, Z, dim] = schur(A, E, 'c')` returns the real generalized Schur form of the pencil $s^*E - A$. In addition, the dim first columns of Z span a basis of the right eigenspace associated with eigenvalues with negative real parts (stable "continuous time" generalized eigenspace).

`[As, Es, Z, dim] = schur(A, E, 'd')`

returns the real generalized Schur form of the pencil $s^*E - A$. In addition, the dim first columns of Z make a basis of the right eigenspace associated with eigenvalues with magnitude lower than 1 (stable "discrete time" generalized eigenspace).

`[As, Es, Z, dim] = schur(A, E, extern2)`

returns the real generalized Schur form of the pencil $s^*E - A$. In addition, the dim first columns of Z make a basis of the right eigenspace associated with eigenvalues of the pencil which are selected according to a rule which is given by the function `extern2`. (see external for details). This external can be described by a Scilab function or by C or Fortran procedure:

A Scilab function

If `extern2` is described by a Scilab function, it should have the following calling sequence: `s=extern2(Alpha, Beta)`, where Alpha and Beta defines a generalized eigenvalue and s a boolean.

C or Fortran procedure

if external `extern2` is described by a C or a Fortran procedure, it should have the following calling sequence:

```
int extern2(double *AlphaR, double *AlphaI, double *Beta)
```

if A and E are real and

```
int extern2(double *AlphaR, double *AlphaI, double *BetaR,
double *BetaI)
```

if A or E are complex. Alpha, and Beta defines the generalized eigenvalue. a true or non zero returned value stands for selected generalized eigenvalue.

References

Matrix schur form computations are based on the Lapack routines DGEES and ZGEES.

Pencil schur form computations are based on the Lapack routines DGGES and ZGGES.

Examples

```
//SCHUR FORM OF A MATRIX
//-----
A=diag([-0.9, -2, 2, 0.9]);X=rand(A);A=inv(X)*A*X;
[U,T]=schur(A);T

[U,dim,T]=schur(A, 'c');
T(1:dim,1:dim) //stable cont. eigenvalues

function t=mytest(Ev),t=abs(Ev)<0.95,endfunction
[U,dim,T]=schur(A,mytest);
T(1:dim,1:dim)

// The same function in C (a Compiler is required)
cd TMPDIR;
C=['int mytest(double *EvR, double *EvI) {' //the C code
  'if (*EvR * *EvR + *EvI * *EvI < 0.9025) return 1;'
  'else return 0; }'];
mputl(C,TMPDIR+'/mytest.c')

//build and link
lp=ilib_for_link('mytest','mytest.c',[],'c');
link(lp,'mytest','c');

//run it
[U,dim,T]=schur(A,'mytest');
//SCHUR FORM OF A PENCIL
//-----
F=[-1,%s, 0, 1;
    0, -1,5-%s, 0;
    0, 0,2+%s, 0;
    1, 0, 0, -2+%s];
A=coeff(F,0);E=coeff(F,1);
[As,Es,Q,Z]=schur(A,E);
Q'*F*Z //It is As+%s*Es
```

```
[As,Es,Z,dim] = schur(A,E,'c')
function t=mytest(Alpha,Beta),t=real(Alpha)<0,endfunction
[As,Es,Z,dim] = schur(A,E,mytest)

//the same function in Fortran (a Compiler is required)
ftn=['integer function mytestf(ar,ai,b)' //the fortran code
     'double precision ar,ai,b'
     'mytestf=0'
     'if(ar.lt.0.0d0) mytestf=1'
     'end']
mputl('      '+ftn,TMPDIR+'/mytestf.f')

//build and link
lp=ilib_for_link('mytestf','mytestf.f',[],'F');
link(lp,'mytestf','f');

//run it

[As,Es,Z,dim] = schur(A,E,'mytestf')
```

See Also

spec , bdiag , ricc , pbig , psmall

Name

spaninter — subspace intersection

```
[X,dim]=spaninter(A,B [,tol])
```

Parameters

A, B
two real or complex matrices with equal number of rows

X
orthogonal or unitary square matrix

dim
integer, dimension of subspace range(A) inter range(B)

Description

computes the intersection of range(A) and range(B).

The first dim columns of X span this intersection i.e. $X(:, 1:dim)$ is an orthogonal basis for range(A) inter range(B)

In the X basis A and B are respectively represented by:

$X' * A$ and $X' * B$.

tol is a threshold ($\sqrt{\%eps}$ is the default value).

Examples

```
A=rand(5,3)*rand(3,4);    // A is 5 x 4, rank=3
B=[A(:,2),rand(5,1)]*rand(2,2);
[X,dim]=spaninter(A,B);
X1=X(:,1:dim);    //The intersection
svd(A,svd([X1,A]))    // X1 in span(A)
svd(B,svd([B,X1]))    // X1 in span(B)
```

See Also

spanplus , spantwo

Authors

F. D.; ;

Name

spanplus — sum of subspaces

```
[X, dim, dima]=spanplus(A,B[, tol])
```

Parameters

A, B
two real or complex matrices with equal number of rows

X
orthogonal or unitary square matrix

dim, dima
integers, dimension of subspaces

tol
nonnegative real number

Description

computes a basis X such that:

the first `dim` columns of X span `Range(A)` and the following `(dim-dima)` columns make a basis of `A+B` relative to A.

The `dim` first columns of X make a basis for `A+B`.

One has the following canonical form for `[A, B]`:

```
X' * [A, B] = [ *, * ]      (dima rows)
            [ 0, * ]      (dim-dima rows)
            [ 0, 0 ]
```

`tol` is an optional argument (see function code).

Examples

```
A=rand(6,2)*rand(2,5);      // rank(A)=2
B=[A(:,1), rand(6,2)]*rand(3,3); //two additional independent vectors
[X, dim, dimA]=spanplus(A, B);
dimA
dim
```

See Also

spaninter , im_inv , spantwo

Authors

F. D.; ;

Name

spantwo — sum and intersection of subspaces

```
[Xp,dima,dimb,dim]=spantwo(A,B, [tol])
```

Parameters

A, B
two real or complex matrices with equal number of rows

Xp
square non-singular matrix

dima, dimb, dim
integers, dimension of subspaces

tol
nonnegative real number

Description

Given two matrices A and B with same number of rows, returns a square matrix Xp (non singular but not necessarily orthogonal) such that :

```
      [A1, 0]      (dim-dimb rows)
Xp*[A,B]=[A2,B2]  (dima+dimb-dim rows)
      [0, B3]      (dim-dima rows)
      [0 , 0]
```

The first dima columns of `inv(Xp)` span `range(A)`.

Columns `dim-dimb+1` to `dima` of `inv(Xp)` span the intersection of `range(A)` and `range(B)`.

The `dim` first columns of `inv(Xp)` span `range(A)+range(B)`.

Columns `dim-dimb+1` to `dim` of `inv(Xp)` span `range(B)`.

Matrix `[A1;A2]` has full row rank ($=\text{rank}(A)$). Matrix `[B2;B3]` has full row rank ($=\text{rank}(B)$). Matrix `[A2,B2]` has full row rank ($=\text{rank}(A \text{ inter } B)$). Matrix `[A1,0;A2,B2;0,B3]` has full row rank ($=\text{rank}(A+B)$).

Examples

```
A=[1,0,0,4;
   5,6,7,8;
   0,0,11,12;
   0,0,0,16];
B=[1,2,0,0]';C=[4,0,0,1];
Sl=ss2ss(syslin('c',A,B,C),rand(A));
[no,X]=contr(Sl('A'),Sl('B'));CO=X(:,1:no); //Controllable part
[uo,Y]=unobs(Sl('A'),Sl('C'));UO=Y(:,1:uo); //Unobservable part
[Xp,dimc,dimu,dim]=spantwo(CO,UO); //Kalman decomposition
Slcan=ss2ss(Sl,inv(Xp));
```

See Also

spanplus , spaninter

Authors

F. D.

Name

spec — eigenvalues of matrices and pencils

```
evals=spec(A)
[R,diagevals]=spec(A)

evals=spec(A,B)
[alpha,beta]=spec(A,B)
[alpha,beta,Z]=spec(A,B)
[alpha,beta,Q,Z]=spec(A,B)
```

Parameters

- A
real or complex square matrix
- B
real or complex square matrix with same dimensions as A
- evals
real or complex vector, the eigenvalues
- diagevals
real or complex diagonal matrix (eigenvalues along the diagonal)
- alpha
real or complex vector, al./be gives the eigenvalues
- beta
real vector, al./be gives the eigenvalues
- R
real or complex invertible square matrix, matrix right eigenvectors.
- L
real or complex invertible square matrix, pencil left eigenvectors.
- R
real or complex invertible square matrix, pencil right eigenvectors.

Description

evals=spec(A)
returns in vector **evals** the eigenvalues.

[R,diagevals]=spec(A)
returns in the diagonal matrix **evals** the eigenvalues and in R the right eigenvectors.

evals=spec(A,B)
returns the spectrum of the matrix pencil $A - s B$, i.e. the roots of the polynomial matrix $s B - A$.

[alpha,beta]=spec(A,B)
returns the spectrum of the matrix pencil $A - s B$, i.e. the roots of the polynomial matrix $A - s B$. Generalized eigenvalues alpha and beta are so that the matrix $A - \alpha ./ \beta B$ is a singular matrix. The eigenvalues are given by **al./be** and if **beta(i) = 0** the ith eigenvalue is at infinity. (For $B = \text{eye}(A)$, **alpha./beta** is **spec(A)**). It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

[alpha,beta,R] = spec(A,B)

returns in addition the matrix R of generalized right eigenvectors of the pencil.

[al,be,L,R] = spec(A,B)

returns in addition the matrix L and R of generalized left and right eigenvectors of the pencil.

[al,be,Z] = spec(A,E)

returns the matrix Z of right generalized eigen vectors.

[al,be,Q,Z] = spec(A,E)

returns the matrices Q and Z of right and left generalized eigen vectors.

For big full / sparse matrix, you can use the Arnoldi module.

References

Matrix eigenvalues computations are based on the Lapack routines

- DGEEV and ZGEEV when the matrix are not symmetric,
- DSYEV and ZHEEV when the matrix are symmetric.

A complex symmetric matrix has conjugate offdiagonal terms and real diagonal terms.

Pencil eigenvalues computations are based on the Lapack routines DGGEV and ZGGEV.

Real and complex matrices

It must be noticed that the type of the output variables, such as evals or R for example, is not necessarily the same as the type of the input matrices A and B. In the following paragraph, we analyse the type of the output variables in the case where one computes the eigenvalues and eigenvectors of one single matrix A.

- Real A matrix

- Symetric

The eigenvalues and the eigenvectors are real.

- Not symmetric

The eigenvalues and eigenvectors are complex.

- Complex A matrix

- Symetric

The eigenvalues are real but the eigenvectors are complex.

- Not symmetric

The eigenvalues and the eigenvectors are complex.

Examples

```
// MATRIX EIGENVALUES
A=diag([1,2,3]);
X=rand(3,3);
A=inv(X)*A*X;
```

```
spec(A)

x=poly(0, 'x');
pol=det(x*eye()-A)
roots(pol)

[S,X]=bdiag(A);
clean(inv(X)*A*X)

// PENCIL EIGENVALUES
A=rand(3,3);
[al,be,R] = spec(A,eye(A));
al./be
clean(inv(R)*A*R) //displaying the eigenvalues (generic matrix)
A=A+%i*rand(A);
E=rand(A);
roots(det(A-%s*E)) //complex case
```

See Also

poly, det, schur, bdiag, colcomp, dsaupd, dnaupd

Name

sqroot — W^*W' hermitian factorization

```
sqroot(X)
```

Parameters

X
symmetric non negative definite real or complex matrix

Description

returns W such that $X=W^*W'$ (uses SVD).

Examples

```
X=rand(5,2)*rand(2,5);X=X*X';  
W=sqroot(X)  
norm(W*W'-X,1)  
  
X=rand(5,2)+%i*rand(5,2);X=X*X';  
W=sqroot(X)  
norm(W*W'-X,1)
```

See Also

chol , svd

Name

squeeze — squeeze

```
hypOut = squeeze(hypIn)
```

Parameters

hypIn

hypermatrix or matrix of constant type.

hypOut

hypermatrix or matrix of constant type.

Description

Remove singleton dimensions of a hypermatrix, that is any dimension for which the size is 1. If the input is a matrix, it is unaffected.

See Also

hypermat , hypermatrices

Authors

Eric Dubois, Jean-Baptiste Silvy

Name

sva — singular value approximation

```
[U, s, V]=sva(A, k)
[U, s, V]=sva(A, tol)
```

Parameters

A
real or complex matrix

k
integer

tol
nonnegative real number

Description

Singular value approximation.

`[U, S, V]=sva(A, k)` with k an integer ≥ 1 , returns U, S and V such that $B=U*S*V'$ is the best L2 approximation of A with $\text{rank}(B)=k$.

`[U, S, V]=sva(A, tol)` with tol a real number, returns U, S and V such that $B=U*S*V'$ such that L2-norm of $A-B$ is at most tol.

Examples

```
A=rand(5,4)*rand(4,5);
[U, s, V]=sva(A, 2);
B=U*s*V';
svd(A)
svd(B)
clean(svd(A-B))
```

See Also

svd

Name

svd — singular value decomposition

```
s=svd(X)
[U,S,V]=svd(X)
[U,S,V]=svd(X,0) (obsolete)
[U,S,V]=svd(X,"e")
[U,S,V,rk]=svd(X [,tol])
```

Parameters

X
a real or complex matrix

s
real vector (singular values)

S
real diagonal matrix (singular values)

U,V
orthogonal or unitary square matrices (singular vectors).

tol
real number

Description

`[U,S,V] = svd(X)` produces a diagonal matrix **S**, of the same dimension as **X** and with nonnegative diagonal elements in decreasing order, and unitary matrices **U** and **V** so that $X = U*S*V'$.

`[U,S,V] = svd(X,0)` produces the "economy size" decomposition. If **X** is *m*-by-*n* with *m* > *n*, then only the first *n* columns of **U** are computed and **S** is *n*-by-*n*.

`s = svd(X)` by itself, returns a vector **S** containing the singular values.

`[U,S,V,rk]=svd(X,tol)` gives in addition *rk*, the numerical rank of **X** i.e. the number of singular values larger than *tol*.

The default value of *tol* is the same as in `rank`.

Examples

```
X=rand(4,2)*rand(2,4)
svd(X)
sqrt(spec(X*X'))
```

See Also

`rank`, `qr`, `colcomp`, `rowcomp`, `sva`, `spec`

Used Functions

svd decompositions are based on the Lapack routines `DGESVD` for real matrices and `ZGESVD` for the complex case.

Name

sylv — Sylvester equation.

```
sylv(A,B,C,flag)
```

Parameters

A,B,C
three real matrices of appropriate dimensions.

flag
character string ('c' or 'd')

Description

$X = \text{sylv}(A, B, C, 'c')$ computes X , solution of the "continuous time" Sylvester equation

$$A^*X + X^*B = C$$

$X = \text{sylv}(A, B, C, 'd')$ computes X , solution of the "discrete time" Sylvester equation

$$A^*X^*B - X = C$$

Examples

```
A=rand(4,4);C=rand(4,3);B=rand(3,3);  
X = sylv(A,B,C,'c');  
norm(A*X+X*B-C)  
X=sylv(A,B,C,'d')  
norm(A*X*B-X-C)
```

See Also

lyap

Name

trace — trace

```
trace(X)
```

Parameters

X
real or complex square matrix, polynomial or rational matrix.

Description

`trace(X)` is the trace of the matrix `X`.

Same as `sum(diag(X))`.

Examples

```
A=rand(3,3);  
trace(A)-sum(spec(A))
```

See Also

`det`

Part XXIV. Localization

Name

`dgettext` — get text translated into the current locale and a specific domain domain.

```
msg=dgettext(domain, myString)
```

Parameters

`domain`

The name of the message domain

`string`

the message to be translated

Description

`dgettext` get the translation of a string to the current locale in a specified message domain.

Examples

```
dgettext('scilab','Startup execution:')
```

See Also

`gettext`

Authors

Sylvestre Ledru

Name

`getdefaultlanguage` — `getdefaultlanguage()` returns the default language used by Scilab.

```
getdefaultlanguage()
```

Description

`getdefaultlanguage()` returns the default language used by Scilab. By default, this function should return `en_US`.

Examples

```
getdefaultlanguage()
```

See Also

`setlanguage` `getlanguage`

Authors

Sylvestre Ledru

Name

getlanguage — getlanguage() returns current language used by Scilab.

```
get language ( )
```

Description

get language () returns current language used by Scilab.

Examples

```
set language ( 'en_US' )  
get language ( )
```

See Also

setlanguage

Authors

A.C.
Sylvestre Ledru

Name

`gettext` — get text translated into the current locale and domain.

```
msg=gettext(myString)
```

Parameters

`string`
the message to be translated

Description

`gettext` get the translation of a string to the current locale in the current domain.

Examples

```
gettext('Startup execution:')
```

See Also

`dgettext`

Authors

Sylvestre Ledru

Name

LANGUAGE — Variable defining the language (OBSOLETE)

Description

LANGUAGE is obsolete. If you need LANGUAGE, add LANGUAGE=getlanguage();

See Also

getlanguage

Name

setdefaultlanguage — sets and saves the internal LANGUAGE value.

```
setdefaultlanguage( language)
```

Parameters

language
with language='fr', 'en', 'ru_RU', 'zh_TW', ...

Description

setdefaultlanguage(language) changes current language and save this value in scilab.

You need to restart scilab, if you want to use menus.

setdefaultlanguage("") resets language to the system value.

setdefaultlanguage is used only Windows. On others operating systems , it returns always %f.

Examples

```
setdefaultlanguage('en_US')  
  
// restart scilab  
getlanguage()  
setdefaultlanguage('fr_FR')  
  
// restart scilab  
getlanguage()  
setdefaultlanguage('')  
  
// restart scilab
```

See Also

getlanguage, setlanguage

Authors

A.C.

Name

setlanguage — Sets the internal LANGUAGE value.

```
set language( language )
```

Parameters

language
with language='fr' or 'en', ...

Description

set language(language) changes current language in scilab.

Examples

```
set language( 'en_US' )  
get language()  
set language( 'en' )  
get language()  
set language( 'fr' )  
get language()  
set language( 'fr_FR' )  
get language()
```

See Also

getlanguage

Authors

A.C.

Part XXV. Optimization and Simulation

Table of Contents

2. Neldermead	1489
fminsearch	1490
neldermead	1499
overview	1520
nmplot	1524
optimget	1535
optimplotfunccount	1537
optimplotfval	1538
optimplotx	1539
optimset	1540
3. Optimization base	1544
optimbase	1545
4. Optimization simplex	1559
optimsimplex	1560

Chapter 2. Neldermead

Nom

`fminsearch` — Computes the unconstrained minimum of given function with the Nelder-Mead algorithm.

```
x = fminsearch ( costf , x0 )  
x = fminsearch ( costf , x0 , options )  
[x,fval,exitflag,output] = fminsearch ( costf , x0 , options )
```

Description

This function searches for the unconstrained minimum of a given cost function.

The provided algorithm is a direct search algorithm, i.e. an algorithm which does not use the derivative of the cost function. It is based on the update of a simplex, which is a set of $k \geq n+1$ vertices, where each vertex is associated with one point and one function value. This algorithm is the Nelder-Mead algorithm.

Design

This function is based on a specialized use of the more general `neldermead` component. Users which want to have a more flexible solution based on direct search algorithms should consider using the `neldermead` component instead of the `fminsearch` function.

Parameters

`costf`

The cost function.

`x0`

The initial guess.

`options`

A struct which contains configurable options of the algorithm (see below for details).

`x`

The minimum.

`fval`

The minimum function value.

`exitflag`

The flag associated with exist status of the algorithm.

The following values are available.

-1

The maximum number of iterations has been reached.

0

The maximum number of function evaluations has been reached.

1

The tolerance on the simplex size and function value delta has been reached. This signifies that the algorithm has converged, probably to a solution of the problem.

output

A struct which stores detailed information about the exit of the algorithm. This struct contains the following fields.

output.algorithm

A string containing the definition of the algorithm used, i.e. 'Nelder-Mead simplex direct search'.

output.funcCount

The number of function evaluations.

output.iterations

The number of iterations.

output.message

A string containing a termination message.

Options

In this section, we describe the options input argument which have an effect on the algorithm used by `fminsearch`.

The options input argument is a data structure which drives the behaviour of `fminsearch`. It allows to handle several options in a consistent and simple interface, without the problem of managing many input arguments.

These options must be set with the `optimset` function. See the `optimset` help for details of the options managed by this function.

The `fminsearch` function is sensitive to the following options.

options.MaxIter

The maximum number of iterations. The default is $200 * n$, where n is the number of variables.

options.MaxFunEvals

The maximum number of evaluations of the cost function. The default is $200 * n$, where n is the number of variables.

options.TolFun

The absolute tolerance on function value. The default value is $1.e-4$.

options.TolX

The absolute tolerance on simplex size. The default value is $1.e-4$.

options.Display

The verbose level.

options.OutputFcn

The output function, or a list of output functions. The default value is empty.

options.PlotFcns

The plot function, or a list of plotput functions. The default value is empty.

Termination criteria

In this section, we describe the termination criteria used by `fminsearch`.

The criteria is based on the following variables:

ssize

the current simplex size,

shiftfv

the absolute value of the difference of function value between the highest and lowest vertices.

If both the following conditions

```
ssize < options.TolX
```

and

```
shiftfv < options.TolFun
```

are true, then the iterations stop.

The size of the simplex is computed using the "sigmaplus" method of the `optimsimplex` component. The "sigmaplus" size is the maximum length of the vector from each vertex to the first vertex. It requires one loop over the vertices of the simplex.

The initial simplex

The `fminsearch` algorithm uses a special initial simplex, which is an heuristic depending on the initial guess. The strategy chosen by `fminsearch` corresponds to the `-simplex0method` flag of the `nelder-mead` component, with the "pfeffer" method. It is associated with the `-simplex0deltausual = 0.05` and `-simplex0deltazero = 0.0075` parameters. Pfeffer's method is an heuristic which is presented in "Global Optimization Of Lennard-Jones Atomic Clusters" by Ellen Fan. It is due to L. Pfeffer at Stanford. See in the help of `optimsimplex` for more details.

The number of iterations

In this section, we present the default values for the number of iterations in `fminsearch`.

The `options` input argument is an optionnal data structure which can contain the `options.MaxIter` field. It stores the maximum number of iterations. The default value is `200n`, where `n` is the number of variables. The factor 200 has not been chosen by chance, but is the result of experiments performed against quadratic functions with increasing space dimension.

This result is presented in "Effect of dimensionality on the nelder-mead simplex method" by Lixing Han and Michael Neumann. This paper is based on Lixing Han's PhD, "Algorithms in Unconstrained Optimization". The study is based on numerical experiment with a quadratic function where the number of terms depends on the dimension of the space (i.e. the number of variables). Their study shows that the number of iterations required to reach the tolerance criteria is roughly `100n`. Most iterations are based on inside contractions. Since each step of the Nelder-Mead algorithm only require one or two function evaluations, the number of required function evaluations in this experiment is also roughly `100n`.

Output and plot functions

The `optimset` function can be used to configure one or more output and plot functions.

The output or plot function is expected to have the following definition:

```
function myfun ( x , optimValues , state )
```

The input arguments `x`, `optimValues` and `state` are described in detail in the `optimset` help page. The `optimValues.procedure` field represent the type of step used during the current iteration and can be equal to one of the following strings

- `''` (the empty string),
- `"initial simplex"`,
- `"expand"`,
- `"reflect"`,
- `"contract inside"`,
- `"contract outside"`.

Example

In the following example, we use the `fminsearch` function to compute the minimum of the Rosenbrock function. We first define the function `"banana"`, and then use the `fminsearch` function to search the minimum, starting with the initial guess `[-1.2 1.0]`. In this particular case, 85 iterations are performed and 159 function evaluations are

```
function y = banana (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
[x, fval, exitflag, output] = fminsearch ( banana , [-1.2 1] );
x
fval
exitflag
output
```

Example with customized options

In the following example, we configure the absolute tolerance on the size of the simplex to a larger value, so that the algorithm performs less iterations. Since the default value of `"TolX"` for the `fminsearch` function is `1.e-4`, we decide to use `1.e-2`. The `optimset` function is used to create an optimization data structure and the field associated with the string `"TolX"` is set to `1.e-2`. The `opt` data structure is then passed to the `fminsearch` function as the third input argument. In this particular case, the number of iterations is 70 with 130 function evaluations.

```
function y = banana (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "TolX" , 1.e-2 );
[x, fval, exitflag, output] = fminsearch ( banana , [-1.2 1] , opt );
x
fval
exitflag
output
```

Example with a pre-defined plot function

In the following example, we want to produce a graphic of the progression of the algorithm, so that we can include that graphic into a report without defining a customized plot function. The `fminsearch` function comes with the following 3 pre-defined functions :

- `optimplotfval`, which plots the function value,
- `optimplotx`, which plots the current point `x`,
- `optimplotfunccount`, which plots the number of function evaluations.

In the following example, we use the three pre-defined functions in order to create one graphic, representing the function value depending on the number of iterations.

```
function y = banana (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfval );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

In the previous example, we could replace the `"optimplotfval"` function with `"optimplotx"` or `"optimplotfunccount"` and obtain different results. In fact, we can get all the figures at the same time, by setting the `"PlotFcns"` to a list of functions, as in the following example.

```
function y = banana (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
myfunctions = list ( optimplotfval , optimplotx , optimplotfunccount );
opt = optimset ( "PlotFcns" , myfunctions );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

Example with an customized output function

In the following example, we want to produce intermediate outputs of the algorithm. We define the `outfun` function, which takes the current point `x` as input argument. The function plots the current point into the current graphic window with the `plot` function. We use the `"OutputFcn"` feature of the `optimset` function and set it to the output function. Then the option data structure is passed to the `fminsearch` function. At each iteration, the output function is called back, which creates and update an interactive plot. While this example creates a 2D plot, the user may customized the output function so that it writes a message in the console, write some data into a data file, etc... The user can distinguish between the output function (associated with the `"OutputFcn"` option) and the plot function (associated with the `"PlotFcns"` option). See the `optimset` for more details on this feature.

```
function y = banana (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
function outfun ( x , optimValues , state )
```

```
plot( x(1),x(2),'.');  
endfunction  
opt = optimset ( "OutputFcn" , outfun);  
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

Example with customized "Display" option

The "Display" option allows to get some input about the intermediate steps of the algorithm as well as to be warned in case of a convergence problem.

In the following example, we present what happens in case of a convergence problem. We set the number of iterations to 10, instead of the default 400 iterations. We know that 85 iterations are required to reach the convergence criteria. Therefore, the convergence criteria is not met and the maximum number of iterations is reached.

```
function y = banana (x)  
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;  
endfunction  
opt = optimset ( "MaxIter" , 10 );  
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

Since the default value of the "Display" option is "notify", a message is generated, which warns the user about a possible convergence problem. The previous script produces the following output.

```
Exiting: Maximum number of iterations has been exceeded  
        - increase MaxIter option.  
        Current function value: 4.1355598
```

Notice that if the "Display" option is now set to "off", no message is displayed at all. Therefore, the user should be warned that turning the Display "off" should be used at your own risk...

In the following example, we present how to display intermediate steps used by the algorithm. We simply set the "Display" option to the "iter" value.

```
function y = banana (x)  
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;  
endfunction  
opt = optimset ( "Display" , "iter" );  
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

The previous script produces the following output. It allows to see the number of function evaluations, the minimum function value and which type of simplex step is used for the iteration.

Iteration	Func-count	min f(x)	Procedure
0	3	24.2	
1	3	20.05	initial simplex

2	5	5.161796	expand
3	7	4.497796	reflect
4	9	4.497796	contract outside
5	11	4.3813601	contract inside
etc...			

Example with customized output option

In this section, we present an example where all the fields from the `optimValues` data structure are used to print a message at each iteration.

```
function outfun ( x , optimValues , state )
    fc = optimValues.funccount;
    fv = optimValues.fval;
    it = optimValues.iteration;
    pr = optimValues.procedure;
    mprintf ( "%d %e %d -%s-\n" , fc , fv , it , pr )
endfunction
opt = optimset ( "OutputFcn" , outfun );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

The previous script produces the following output.

```
3 2.420000e+001 0 --
3 2.005000e+001 1 -initial simplex-
5 5.161796e+000 2 -expand-
7 4.497796e+000 3 -reflect-
9 4.497796e+000 4 -contract outside-
11 4.381360e+000 5 -contract inside-
13 4.245273e+000 6 -contract inside-
[...]
157 1.107549e-009 84 -contract outside-
159 8.177661e-010 85 -contract inside-
159 8.177661e-010 85 --
```

Some advices

In this section, we present some practical advices with respect to the Nelder-Mead method. A deeper analysis is provided in the bibliography at the end of this help page, as well as in the "Nelder-Mead User's Manual" provided on Scilab's Wiki. The following is a quick list of tips to overcome problems that may happen with this algorithm.

- We should consider the `optim` function before considering the `fminsearch` function. Because `optim` uses the gradient of the function and uses this information to guess the local curvature of the cost function, the number of iterations and function evaluations is (much) lower with `optim`, when the function is sufficiently smooth. If the derivatives of the function are not available, it is still possible to use numerical derivatives combined with the `optim` function (this feature is provided by the `derivative` and `numdiff` functions). If the function has discontinuous derivatives, the `optim` function provides the `nd` solver which is very efficient. Still, there are situations where the cost function is discontinuous or "noisy". In these situations, the `fminsearch` function can perform well.

- We should not expect a fast convergence with many parameters, i.e. more than 10 to 20 parameters. It is known that the efficiency of this algorithm decreases rapidly when the number of parameters increases.
- The default tolerances are set to pretty loose values. We should not reduce the tolerances in the goal of getting very accurate results. Because the convergence rate of Nelder-Mead's algorithm is low (at most linear), getting a very accurate solution will require a large number of iterations. Instead, we can most of the time expect a "good reduction" of the cost function with this algorithm.
- Although the algorithm practically converges in many situations, the Nelder-Mead algorithm is not a provably convergent algorithm. There are several known counter-examples where the algorithm fails to converge on a stationary point and, instead, converge to a non-stationary point. This situation is often indicated by a repeated application of the contraction step. In that situation, we simply restart the algorithm with the final point as the new initial guess. If the algorithm converges to the same point, there is a good chance that this point is a "good" solution.
- Taking into account for bounds constraints or non-linear inequality constraints can be done by penalization methods, i.e. setting the function value to a high value when the constraints are not satisfied. While this approach works in some situations, it may also fail. In this case, users might be interested in Box's complex algorithm, provided by Scilab in the `neldermead` component. If the problem is really serious, Box's complex algorithm will also fail and a more powerful solver is necessary.

Bibliography

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R., The Computer Journal, 1965

"Iterative Methods for Optimization", C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

"Algorithm AS47 - Function minimization using a simplex procedure", O'Neill, R., Applied Statistics, 1971

"Effect of dimensionality on the nelder-mead simplex method", Lixing Han and Michael Neumann, Optimization Methods and Software, 21, 1, 1--16, 2006.

"Algorithms in Unconstrained Optimization", Lixing Han, Ph.D., The University of Connecticut, 2000.

"Global Optimization Of Lennard-Jones Atomic Clusters" Ellen Fan, Thesis, February 26, 2002, McMaster University

TODO

- implement the 'interrupt' state of the algorithm
- add a demo with an interactive output function, which draws the plot during the optimization.
- implement the stop of an optimization via the stop output argument of the output function

Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

Acknowledgements

Michael Baudin would like to thank Lixing Han, who kindly sent his PhD thesis.

See Also

`optim`, `neldermead`, `optimset`, `optimget`, `optimplotfval`, `optimplotx`, `optimplotfunccount`

Nom

neldermead — Provides several direct search optimization algorithms based on the simplex method.

```
newobj = neldermead_new ()
this = neldermead_destroy (this)
this = neldermead_configure (this, key, value)
value = neldermead_cget (this, key)
neldermead_display ( this )
value = neldermead_get ( this , key )
this = neldermead_search ( this )
this = neldermead_restart ( this )
[ this , result ] = neldermead_function ( this , x )
```

Description

This class provides several direct search optimization algorithms based on the simplex method.

The optimization problem to solve is the minimization of a cost function, with bounds and nonlinear constraints

```
min f(x)
l_i <= x_i <= h_i, i = 1,n
g_i(x) <= 0, i = 1,nbineq
```

where

n
number of variables

nbineq
number of inequality constraints

The provided algorithms are direct search algorithms, i.e. algorithms which do not use the derivative of the cost function. They are based on the update of a simplex, which is a set of $k \geq n+1$ vertices, where each vertex is associated with one point and one function value.

The following algorithms are available :

Spendley, Hext and Himsworth fixed size simplex method

This algorithm solves an unconstrained optimization problem with a fixed sized simplex made of $k=n+1$ vertices.

Nelder and Mead variable size simplex method

This algorithm solves an unconstrained optimization problem with a variable sized simplex made of $k=n+1$ vertices.

Box complex method

This algorithm solves a constrained optimization problem with a variable sized simplex made of an arbitrary k number of vertices ($k=2n$ is recommended by Box).

Design

The neldermead component is built on top of the optimbase and optimsimplex components.

Functions

The following functions are available.

`newobj = neldermead_new ()`

Creates a new neldermead object.

`newobj`

The new object.

`this = neldermead_destroy (this)`

Destroy the given object.

`this`

The current object.

`this = neldermead_configure (this,key,value)`

Configure the current object with the given value for the given key.

`this`

The current object.

`key`

the key to configure. The following keys are available.

`-verbose`

set to 1 to enable verbose logging. (default is 0)

`-verbosetermination`

set to 1 to enable verbose termination logging. (default is 0)

`-x0`

the initial guess, as a $n \times 1$ column vector, where n is the number of variables.

`-maxfunevals`

the maximum number of function evaluations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

`-maxiter`

the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

`-tolfunabsolute`

the absolute tolerance for the function value (default is 0.0).

`-tolfunrelative`

the relative tolerance for the function value (default is %eps).

`-tolfunmethod`

the method used for the tolerance on function value in the termination criteria.

The following values are available : %t, %f (default is %f). If this criteria is triggered, the status of the optimization is set to "tolf".

`-tolxabsolute`

the absolute tolerance on x (default is 0.0).

`-tolxrelative`

the relative tolerance on x (default is %eps).

`-tolxmethod`

the method used for the tolerance on x in the termination criteria.

The following values are available : %t, %f (default is %t). If this criteria is triggered, the status of the optimization is set to "tolx".

-function

the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument

an additionnal argument, passed to the cost function.

-outputcommand

a command which is called back for output.

See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg

an additionnal argument, passed to the output command.

-numberofvariables

the number of variables to optimize (default is 0).

-storehistory

set to 1 to enable the history storing (default is 0).

-boundsmin

the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax

the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst

the number of inequality constraints (default is 0)

-method

the name of the algorithm to use. The following methods are available :

"fixed"

the Spendley et al. fixed simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"variable"

the Nelder-Mead variable simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"box"

the Box complex algorithm. This algorithm takes into account bounds and nonlinear inequality constraints.

"mine"

the user-defined algorithm, associated with the `-mymethodoption`. See below for details.

-simplex0method

the method to use to compute the initial simplex. The first vertex in the simplex is always the initial guess associated with the `-x0` option. The following methods are available :

"given"

the coordinates associated with the `-coords0` option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"

the simplex is computed from the coordinate axes and the length associated with the `-simplex0length` option.

"spendley"

the simplex is computed so that it is regular with the length associated with the `-simplex0length` option (i.e. all the edges have the same length).

"pfeffer"

the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the `-simplex0deltausual` and `-simplex0deltazero`.

"randbounds"

the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the `-boxnbpoints` option.

`-coords0`

the coordinates of the vertices of the initial simplex. If the `-simplex0method` option is set to "given", these coordinates are used to compute the initial simplex. This matrix is expected to have shape $nbve \times n$ where $nbve$ is the number of vertices and n is the number of variables.

`-simplex0length`

the length to use when the initial simplex is computed with the "axes" or "spendley" methods. If the initial simplex is computed from "spendley" method, the length is expected to be a scalar value. If the initial simplex is computed from "axes" method, it may be either a scalar value or a vector of values, with rank n , where n is the number of variables.

`-simplex0deltausual`

the relative delta for non-zero parameters in "pfeffer" method. The default value is 0.05.

`-simplex0deltazero`

the absolute delta for non-zero parameters in "pfeffer" method. The default value is 0.0075.

`-rho`

the reflection coefficient. This parameter is used when the `-method` option is set to "fixed" or "variable". The default value is 1.0.

`-chi`

the expansion coefficient. This parameter is used when the `-method` option is set to "variable". The default value is 2.0.

`-gamma`

the contraction coefficient. This parameter is used when the `-method` option is set to "variable". The default value is 0.5.

-sigma

the shrinkage coefficient. This parameter is used when the `-method` option is set to "fixed" or "variable". The default value is 0.5.

-tolsimplexizemethod

set to %f to disable the tolerance on the simplex size. The default value is %t. If this criteria is triggered, the status of the optimization is set to "tolsize".

When this criteria is enabled, the values of the options `-tolsimplexizeabsolute` and `-tolsimplexizerelative` are used in the termination criteria. The method to compute the size is the "sigmaplus" method.

-tolsimplexizeabsolute

the absolute tolerance on the simplex size. The default value is 0.0.

-tolsimplexizerelative

the relative tolerance on the simplex size. The default value is %eps.

-tolssizedeltafvmethod

set to %t to enable the termination criteria based on the size of the simplex and the difference of function value in the simplex. The default value is %f. If this criteria is triggered, the status of the optimization is set to "tolssizedeltafv".

This termination criteria uses the values of the options `-tolsimplexizeabsolute` and `-toldeltafv`. This criteria is identical to Matlab's `fminsearch`.

-toldeltafv

the absolute tolerance on the difference between the highest and the lowest function values.

-tolvarianceflag

set to %t to enable the termination criteria based on the variance of the function value. If this criteria is triggered, the status of the optimization is set to "tolvariance".

This criteria is suggested by Nelder and Mead.

-tolabsolutevariance

the absolute tolerance on the variance of the function values of the simplex.

-tolrelativevariance

the relative tolerance on the variance of the function values of the simplex.

-kelleystagnationflag

set to %t to enable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. The default value is %f. If this criteria is triggered, the status of the optimization is set to "kelleystagnation".

-kelleynormalizationflag

set to %f to disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the option `-kelleystagnationalpha0` as is. Default value is %t, i.e. the simplex gradient of the initial simplex is taken into account in the stagnation detection.

-kelleystagnationalpha0

the parameter used in Kelley's stagnation detection. The default value is 1.e-4.

-restartflag

set to %t to enable the automatic restart of the algorithm. Default value is %f.

-restartdetection

the method to detect if the automatic restart must be performed. The following methods are available :

"oneill"

the factorial local optimality test by O'Neill is used. If the test finds a local point which is better than the computed optimum, a restart is performed.

"kelley"

the sufficient decrease condition by O'Neill is used. If the test finds that the status of the optimization is "kelleystagnation", a restart is performed. This status may be generated if the -kelleystagnationflag option is set to %t.

The default method is "oneill".

-restartmax

the maximum number of restarts, when automatic restart is enabled via the -restartflag option. Default value is 3.

-restarteps

the absolute epsilon value used to check for optimality in the factorial O'Neill restart detection. The default value is %eps.

-restartstep

the absolute step length used to check for optimality in the factorial O'Neill restart detection. The default value is 1.0.

-restartsimplexmethod

the method to compute the initial simplex after a restart. The following methods are available.

"given"

the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mc Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"

the simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

"spendley"

the simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

"pfeffer"

the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

"randbounds"

the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

"oriented"

the simplex is computed so that it is oriented, as suggested by C.T. Kelley.

The default method is "oriented".

-scalingsimplex0

the algorithm used to scale the initial simplex into the nonlinear constraints. The following two algorithms are provided :

- "tox0": scales the vertices toward the initial guess.
- "tocentroid": scales the vertices toward the centroid, as recommended by Box.

If the centroid happens to be unfeasible, because the constraints are not convex, the scaling of the initial simplex toward the centroid may fail. Since the initial guess is always feasible, scaling toward the initial guess cannot fail. The default value is "tox0".

-boxnbpoints

the number of points in the initial simplex, when the -simplex0method is set to "rand-bounds". The value of this option is also use to update the simplex when a restart is performed and the -restartsimplexmethod option is set to "randbounds". The default value is so that the number of points is twice the number of variables of the problem.

-boxineqscaling

the scaling coefficient used to scale the trial point for function improvement or into the constraints of Box's algorithm. Default value is 0.5.

-guinalphamin

the minimum value of alpha when scaling the vertices of the simplex into nonlinear constraints in Box's algorithm. Default value is 1.e-5.

-boxreflect

the reflection factor in Box's algorithm. Default value is 1.3.

-boxtermination

set to %t to enable Box termination criteria. Default value is %f.

-boxtolf

the absolute tolerance on difference of function values in the simplex, suggested by Box. This tolerance is used if the -boxtermination option is set to %t. Default value is 1.e-5.

-boxnbmatch

the number of consecutive match of Box termination criteria. Default value is 5.

-boxboundsalpha

the parameter used to project the vertices into the bounds in Box's algorithm. Default value is 0.000001.

-mymethod

a user-derined simplex algorithm. See below for details (default is empty).

-myterminate

a user-defined terminate function. See below for details (default is empty).

-myterminateflag

set to %t to enable the user-defined terminate function (default is %f).

value

the value.

value = neldermead_cget (this,key)

Get the value for the given key. If the key is unknown, generates an error.

this

The current object.

key

the name of the key to query. The list of available keys is the same as for the `neldermead_configure` function.

`value = neldermead_get ( this , key )`

Get the value for the given key. If the key is unknown, generates an error.

this

The current object.

key

the key to get.

The following keys are available :

-funevals

the number of function evaluations

-iterations

the number of iterations

-xopt

the x optimum, as a $n \times 1$ column vector, where n is the number of variables.

-fopt

the optimum cost function value

-historyxopt

an array, with `nbiter` values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the `-storehistory` option.

-historyfopt

an array, with `nbiter` values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the `-storehistory` option.

-fx0

the function value for the initial guess

-status

a string containing the status of the optimization. See below for details about the optimization status.

-historysimplex

a matrix containing the history of the simplex during the iterations. This matrix has rank $\text{nbiter} \times \text{nbve} \times n$, where `nbiter` is the number of iterations, `nbve` is the number of vertices in the simplex and n is the number of variables.

-simplex0

the initial simplex. This is a simplex object, which is suitable for processing with the `optimsimplex` component.

-simplexopt

the optimum simplex. This is a simplex object, which is suitable for processing with the `optimsimplex` component.

-restartnb

the number of actual restarts performed.

Most fields are available only after an optimization has been performed with one call to the `neldermead_search` method.

`neldermead_display ( this )`

Display the current settings in the console.

`this`

The current object.

`this = neldermead_search ( this )`

Performs the optimization associated with the method associated with the `-method` option and find the optimum.

`this`

The current object.

If the `-restartflag` option is enabled, automatic restarts are performed, based on the `-restartdetection` option.

`this = neldermead_restart ( this )`

Restarts the optimization by updating the simplex and performing a new search.

`this`

The current object.

`[ this , result ] = neldermead_function ( this , x )`

Call the cost function and return the value.

`this`

The current object.

`x`

the point where the function is to be evaluated

`index`

optionnal, a flag to pass to the cost function (default = 1). See the section "The cost function" for available values of `index`.

The cost function

The option `-function` allows to configure the cost function. The cost function is used to compute the objective function value `f`. If the `-nbineqconst` option is configured to a non-zero value, the cost function must also compute the value of the nonlinear, positive, inequality constraints `c`. The cost function can also take as input/output an additional argument, if the `-costfargument` option is configured. Depending of these options, the cost function can have one of the following headers :

```
// Without additionnal data
[ f , index ] = costf ( x , index )
[ f , c , index ] = costf ( x , index )
// With -costfargument
[ f , index , data ] = costf ( x , index , data )
[ f , c , index , data ] = costf ( x , index , data )
```

where

`x`

the current point, as a column vector

`index`

optional, an integer representing the value to compute

- f
the value of the cost function
- c
the value of the non-linear, positive, inequality constraints
- data
an user-provided input/output argument

The index input parameter tells to the cost function what is expected in the output arguments. It has the following meaning

index = 2
compute f

index = 5
compute c

index = 6
compute f and c

In the most simplex case, there is no additionnal cost function argument and no nonlinear constraints. In this case, the cost function is expected to have the following header

```
function [ f , index ]= myfunction ( x , index )
```

The `data` argument is both input and output and the following paragraph explains for what reason and how it works.

This feature may be used in the situation where the cost function has to update its environment from call to call. Its simplest use is to count the number of calls to the cost function, but this feature is already available directly. Consider the more practical situation where the optimization requires the execution of an underlying Newton method (a chemical solver for example). This Newton method requires an initial guess `x0`. If the initial guess for this underlying Newton method is kept constant, the Newton method may have problems to converge when the current optimization point get far away from the its initial point. If the `-costfargument` option is provided, the initial guess for the Newton method can be updated so that it gets the value of the previous call. This way, the Newton method will have less problems to converge and the cost function evaluation may be faster.

We now present how the feature works. Everytime the cost function is called back, the `-costfargument` option is passed to the cost function as an input argument. If the cost function modifies its content in the output argument, the content of the `-costfargument` option is updated accordingly. Once the optimization is performed, the user may call the `neldermead_cget` function and get back an updated `-costfargument` content.

The output function

The option `-outputcommand` allows to configure a command which is called back at the start of the optimization, at each iteration and at the end of the optimization.

The output function must have the following header

```
function outputcmd(state, data, myobj)
```

where

state
a string representing the current state of the algorithm. Available values are "init", "iter", "done".

data

a tlist containing at least the following entries

x

the current optimum

fval

the current function value

iteration

the current iteration index

funccount

the number of function evaluations

simplex

the current simplex

step

the previous step in the algorithm. The following values are available : "init", "done", "reflection", "expansion", "insidecontraction", "outsidecontraction", "reflectionnext", "shrink".

myobj

a user-defined parameter.

This input parameter is defined with the `-outputcommandarg` option.

The output function may be used when debugging the specialized optimization algorithm, so that a verbose logging is produced. It may also be used to write one or several report files in a specialized format (ASCII, LaTeX, Excel, Hdf5, etc...). The user-defined parameter may be used in that case to store file names or logging options.

The data tlist argument may contain more fields than the current presented ones. These additionnal fields may contain values which are specific to the specialized algorithm, such as the simplex in a Nelder-Mead method, the gradient of the cost function in a BFGS method, etc...

Termination

The current component takes into account for several generic termination criterias.

The following termination criterias are enabled by default :

- `-maxiter`,
- `-maxfunevals`,
- `-tolxmethod`.
- `-tolsimplexizemethod`.

The `optimization_terminate` function uses a set of rules to compute if the termination occurs, which leads to an optimization status which is equal to one of the following : "continue", "maxiter", "maxfunevals", "tolf", "tolx", "tolsize", "tolsizedeltafv", "kelleystagnation", "tolboxf", "tolvariance". The value of the status may also be a user-defined string, in the case where a user-defined termination function has been set.

The following set of rules is examined in this order.

- By default, the status is "continue" and the `terminate` flag is %f.
- The number of iterations is examined and compared to the `-maxiter` option : if the following condition

```
iterations >= maxiter
```

is true, then the status is set to "maxiter" and terminate is set to %t.

- The number of function evaluations and compared to the -maxfunevals option is examined : if the following condition

```
funevals >= maxfunevals
```

is true, then the status is set to "maxfuneval" and terminate is set to %t.

- The tolerance on function value is examined depending on the value of the -tolfunmethod.

%f
then the criteria is just ignored.

%t
if the following condition

```
abs(currentfopt) < tolfunrelative * abs(previousfopt) + tolfunabsolute
```

is true, then the status is set to "tolf" and terminate is set to %t.

The relative termination criteria on the function value works well if the function value at optimum is near zero. In that case, the function value at initial guess fx0 may be used as previousfopt.

This criteria is sensitive to the -tolfunrelative and -tolfunabsolute options.

The absolute termination criteria on the function value works if the user has an accurate idea of the optimum function value.

- The tolerance on x is examined depending on the value of the -tolxmethod.

%f
then the criteria is just ignored.

%t
if the following condition

```
norm(currentxopt - previousxopt) < tolxrelative * norm(currentxopt) + tolxabsolute
```

is true, then the status is set to "tolx" and terminate is set to %t.

This criteria is sensitive to the -tolxrelative and -tolxabsolute options.

The relative termination criteria on x works well if x at optimum is different from zero. In that case, the condition measures the distance between two iterates.

The absolute termination criteria on x works if the user has an accurate idea of the scale of the optimum x. If the optimum x is near 0, the relative tolerance will not work and the absolute tolerance is more appropriate.

- The tolerance on simplex size is examined depending on the value of the -tolsimplex-izemethod option.

%f
then the criteria is just ignored.

%t
if the following condition

```
ssize < tolsimplexizerelative * simplexsize0 + tolsimplexizeabsolute
```

is true where `simplexsize0` is the size of the simplex at iteration 0, then the `status` is set to "tolsize" and `terminate` is set to %t.

The size of the simplex is computed from the "sigmaplus" method of the `optimsimplex` component. This criteria is sensitive to the `-tolsimplexizeabsolute` and the `-tol-simplexizerelative` options.

- The absolute tolerance on simplex size and absolute difference of function value is examined depending on the value of the `-tolssizedeltafvmethod` option.

%f
then the criteria is just ignored.

%t
if both the following conditions

```
ssize < tolsimplexizeabsolute
```

```
shiftfv < toldeltafv
```

is true where `ssize` is the current simplex size and `shiftfv` is the absolute value of the difference of function value between the highest and lowest vertices, then the `status` is set to "tolssizedeltafv" and `terminate` is set to %t.

- The stagnation condition based on Kelley sufficient decrease condition is examined depending on the value of the `-kelleystagnationflag` option.

%f
then the criteria is just ignored.

%t
if the following condition

```
newfvmean <= oldfvmean - alpha * sg' * sg
```

is true where `newfvmean` (resp. `oldfvmean`) is the function value average in the current iteration (resp. in the previous iteration), then the `status` is set to "kelleystagnation" and `terminate` is set to %t. Here, `alpha` is a non-dimensional coefficient and `sg` is the simplex gradient.

- The termination condition suggested by Box is examined depending on the value of the `-boxtermination` option.

%f
then the criteria is just ignored.

%t

if both the following conditions

```
shiftfv < boxtolf
```

```
boxkount == boxnbmatch
```

is true where `shiftfv` is the difference of function value between the best and worst vertices, and `boxkount` is the number of consecutive iterations where this criteria is met, then the status is set to "tolboxf" and terminate is set to %t. Here, the `boxtolf` parameter is the value associated with the `-boxtolf` option and is a user-defined absolute tolerance on the function value. The `boxnbmatch` parameter is the value associated with the `-boxnbmatch` option and is the user-defined number of consecutive match.

- The termination condition based on the variance of the function values in the simplex is examined depending on the value of the `-tolvarianceflag` option.

%f

then the criteria is just ignored.

%t

if the following condition

```
var < tolrelativevariance * variancesimplex0 + tolabsolutevariance
```

is true where `var` is the variance of the function values in the simplex, then the status is set to "tolvariance" and terminate is set to %t. Here, the `tolrelativevariance` parameter is the value associated with the `-tolrelativevariance` option and is a user-defined relative tolerance on the variance of the function values. The `tolabsolutevariance` parameter is the value associated with the `-tolabsolutevariance` option and is the user-defined absolute tolerance of the variance of the function values.

- The user-defined termination condition is examined depending on the value of the `-myterminateflag` option.

%f

then the criteria is just ignored.

%t

if the `term` output argument boolean returned by the termination function is true, then the status is set to the user-defined status and `terminate` is set to %t.

Kelley's stagnation detection

The stagnation detection criteria suggested by Kelley is based on a sufficient decrease condition, which requires a parameter $\alpha > 0$ to be defined. The `-kelleynormalizationflag` option allows to configure the method to use to compute this α parameter : two methods are available, where each method corresponds to a different paper by Kelley :

constant

In "Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition", Kelley uses a constant α , with the suggested value $1.e-4$, which is is typical choice for line search method.

normalized

in "Iterative Methods for Optimization", Kelley uses a normalized alpha, computed from the following formula

$$\alpha = \alpha_0 * \sigma_0 / nsg$$

where σ_0 is the size of the initial simplex and nsg is the norm of the simplex gradient for the initial guess point.

O'Neil factorial optimality test

In "Algorithm AS47 - Function minimization using a simplex procedure", R. O'Neil presents a fortran 77 implementation of the simplex method. A factorial test is used to check if the computed optimum point is a local minimum. If the -restartdetection option is set to "oneill", that factorial test is used to see if a restart should be performed.

Spendley et al. implementation notes

The original paper may be implemented with several variations, which might lead to different results. This section defines what algorithmic choices have been used.

The paper states the following rules.

- "Rule 1. Ascertain the lowest reading y , of $y_1 \dots y_{k+1}$ Complete a new simplex S_p by excluding the point V_p corresponding to y , and replacing it by V^* defined as above."
- "Rule 2. If a result has occurred in $(k + 1)$ successive simplexes, and is not then eliminated by application of Rule 1, do not move in the direction indicated by Rule 1, or at all, but discard the result and replace it by a new observation at the same point."
- "Rule 3. If y is the lowest reading in S_o , and if the next observation made, y^* , is the lowest reading in the new simplex S , do not apply Rule 1 and return to S_o from S_p . Move out of S , by rejecting the second lowest reading (which is also the second lowest reading in S_o)."

We implement the following "rules" of the Spendley et al. method.

- Rule 1 is strictly applied, but the reflection is done by reflection the high point, since we minimize a function instead of maximizing it, like Spendley.
- Rule 2 is NOT implemented, as we expect that the function evaluation is not subject to errors.
- Rule 3 is applied, ie reflection with respect to next to high point.

The original paper does not mention any shrink step. When the original algorithm cannot improve the function value with reflection steps, the basic algorithm stops. In order to make the current implementation of practical value, a shrink step is included, with shrinkage factor σ . This perfectly fits into to the spirit of the original paper. Notice that the shrink step make the rule #3 (reflection with respect to next-to-worst vertex) unnecessary. Indeed, the minimum required steps are the reflection and shrinkage. Never the less, the rule #3 has been kept in order to make the algorithm as close as it can be to the original.

Nelder-Mead implementation notes

The purpose of this section is to analyse the current implementation of Nelder-Mead's algorithm.

The algorithm that we use is described in "Iterative Methods for Optimization" by C. T. Kelley.

The original paper uses a "greedy" expansion, in which the expansion point is accepted whatever its function value. The current implementation, as most implementations, uses the expansion point only if it improves over the reflection point, that is,

- if $f_e < f_r$, then the expansion point is accepted,
- if not, the reflection point is accepted.

The termination criteria suggested by Nelder and Mead is based on an absolute tolerance on the standard deviation of the function values in the simplex. We provide this original termination criteria with the `-tolvarianceflag` option, which is disabled by default.

Box's complex algorithm implementation notes

In this section, we analyse the current implementation of Box's complex method.

The initial simplex can be computed as in Box's paper, but this may not be safe. In his paper, Box suggest that if a vertex of the initial simplex does not satisfy the non linear constraints, then it should be "moved halfway toward the centroid of those points already selected". This behaviour is available when the `-scalingsimplex0` option is set to `"tcenter"`. It may happen, as suggested by Guin, that the centroid is not feasible. This may happen if the constraints are not convex. In this case, the initial simplex cannot be computed. This is why we provide the `"tox0"` option, which allows to compute the initial simplex by scaling toward the initial guess, which is always feasible.

In Box's paper, the scaling into the non linear constraints is performed "toward" the centroid, that is, by using a scaling factor equal to 0.5. This default scaling factor might be sub-optimal in certain situations. This is why we provide the `-boxineqscaling` option, which allows to configure the scaling factor.

In Box's paper, whether we are concerned with the initial simplex or with the simplex at a given iteration, the scaling for the non linear constraints is performed without end. This is because Box's hypothesis is that "ultimately, a satisfactory point will be found". As suggested by Guin, if the process fails, the algorithm goes into an infinite loop. In order to avoid this, we perform the scaling until a minimum scaling value is reached, as defined by the `-guinalphamin` option.

We have taken into account for the comments by Guin, but it should be emphasized that the current implementation is still as close as possible to Box's algorithm and is not Guin's algorithm. More precisely, during the iterations, the scaling for the non linear constraints is still performed toward the centroid, be it feasible or not.

User-defined algorithm

The `-mymethod` option allows to configure a user-defined simplex-based algorithm. The reason for this option is that many simplex-based variants of Nelder-Mead's algorithm have been developped over the years, with specific goals. While it is not possible to provide them all, it is very convenient to use the current structure without being forced to make many developments.

The value of the `-mymethod` option is expected to be a Scilab function with the following header

```
function this = myalgorithm ( this )
endfunction
```

where `-this` is the current object.

In order to use the user-defined algorithm, the `-method` option must be set to "mine". In this case, the component performs the optimization exactly as if the user-defined algorithm was provided by the component.

The user interested in that feature may use the internal scripts provided in the distribution as templates and tune his own algorithm from that point. There is of course no warranty that the user-defined algorithm improves on the standard algorithm, so that users use this feature at their own risks.

User-defined termination

Many termination criteria are found in the bibliography. Users which aim at reproducing the results exhibited in a particular paper may find that none of the provided termination criteria match the one which is used in the paper. It may also happen that the provided termination criteria are not suitable for the specific test case. In those situation the `-myterminate` option allows to configure a user-defined termination function.

The value of the `-myterminate` option is expected to be a Scilab function with the following header

```
function [ this , terminate , status ] = mystoppingrule ( this , simplex )
endfunction
```

where `-this` is the current object and `-simplex` is the current simplex. The `terminate` output argument is a boolean which is false if the algorithm must continue and true if the algorithm must stop. The `status` output argument is a string which is associated with the current termination criteria.

In order to enable the use of the user-defined termination function, the value of the `-myterminate-flag` must be set to true. At each iteration, if the `-myterminateflag` option has been set to true, the user-defined termination is called. If the `terminate` output argument is true, then the algorithm is stopped. In that case, the value of the `-status` option of the `neldermead_get` is the value of the `status` output argument of the user-defined termination function.

Example #1 : basic use

In the following example, we solve a simple quadratic test case. We begin by defining the cost function, which takes 2 input arguments and returns the objective. The classical starting point `[-1.2 1.0]` is used. The `neldermead_new` creates a new `neldermead` object. Then we use the `neldermead_configure` method to configure the parameters of the problem. We use all default settings and perform the search for the optimum. The `neldermead_display` function is used to display the state of the optimization and the `neldermead_get` is used to retrieve the optimum parameters.

```
function [ f , index ] = quadratic ( x , index )
    f = x(1)^2 + x(2)^2;
endfunction
x0 = [1.0 1.0].';
nm = neldermead_new ();
nm = neldermead_configure(nm, "-numberofvariables",2);
nm = neldermead_configure(nm, "-function",quadratic);
nm = neldermead_configure(nm, "-x0",x0);
nm = neldermead_search(nm);
neldermead_display(nm);
xopt = neldermead_get(nm, "-xopt");
nm = neldermead_destroy(nm);
```

Example #2 : customized use

In the following example, we solve the Rosenbrock test case. We begin by defining the Rosenbrock function, which takes 2 input arguments and returns the objective. The classical starting point [-1.2 1.0] is used. The `neldermead_new` creates a new `neldermead` object. Then we use the `neldermead_configure` method to configure the parameters of the problem. The initial simplex is computed from the axes and the single length 1.0 (this is the default, but is explicitly written here as an example). The variable simplex algorithm by Nelder and Mead is used, which corresponds to the `-method "variable"` option. The `neldermead_search` function performs the search for the minimum. Once the minimum is found, the `neldermead_contour` allows to compute the data required by the contour function. This is possible since our problem involves only 2 parameters. This function uses the cost function previously configured to compute the required data. The contour plot is directly drawn from the data provided by `neldermead_contour`. Then we plot the initial guess on the contour plot as a blue dot. The `neldermead_get` function is used to get the optimum, which is associated with the `-xopt` option. The optimum is plot on the contour plot as a red dot.

```
mprintf("Defining Rosenbrock function...\n");
function [ f , index ] = rosenbrock ( x , index )
    y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
x0 = [-1.2 1.0]';
mprintf("x0=%s\n",strcat(string(x0)," "));
mprintf("Creating object...\n");
nm = neldermead_new ();
mprintf("Configuring object...\n");
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_configure(nm,"-x0",x0);
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",1.0);
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",0);
nm = neldermead_configure(nm,"-verbosetermination",0);
mprintf("Searching for minimum...\n");
nm = neldermead_search(nm);
mprintf("Plot contour...\n");
xmin = -2.0 ; xmax = 2.0 ; ymin = -2.0 ; ymax = 2.0 ; nx = 100 ; ny = 100;
stepx = (xmax - xmin)/nx
xdata = xmin:stepx:xmax;
stepy = (ymax - ymin)/ny
ydata = ymin:stepy:ymax;
for ix = 1:length(xdata)
    for iy = 1:length(ydata)
        experiment = [xdata(ix) ydata(iy)];
        [ nm , fiexp ] = neldermead_function ( nm , experiment );
        zdata ( ix , iy ) = fiexp;
    end
end
end
wnum = 100001;
my_handle = scf(wnum);
contour ( xdata , ydata , zdata , [1 10 100 500 1000 2000] )
// Plot starting point
```

```
mprintf("x0 : blue dot\n");
plot(x0(1),x0(2));
my_handle.children.children(1).children.mark_mode="on";
my_handle.children.children(1).children.mark_size = 5;
my_handle.children.children(1).children.mark_foreground = 2;
mprintf("xopt : red dot\n");
xopt = neldermead_get(nm, "-xopt");
plot(xopt(1),xopt(2));
my_handle.children.children(1).children.mark_mode="on";
my_handle.children.children(1).children.mark_size = 5;
my_handle.children.children(1).children.mark_foreground = 5;

nm = neldermead_destroy(nm);
```

The `-verbose` option allows to get detailed informations about the current optimization process. The following is a sample output for an optimization based on the Nelder and Mead variable-shape simplex algorithm. Only the output corresponding to the iteration #156 is displayed. In order to display specific outputs (or to create specific output files and graphics), the `-outputcommand` option should be used.

```
=====
Iteration #156 (total = 156)
Function Eval #297
Xopt : 1 1
Fopt : 6.871176e-027
DeltaFv : 2.880999e-026
Center : 1 1
Size : 2.548515e-013
Vertex #1/3 : fv=0.000000, x=1.000000 1.000000
Vertex #2/3 : fv=0.000000, x=1.000000 1.000000
Vertex #3/3 : fv=0.000000, x=1.000000 1.000000
nmplot_outputcmd (1)
Reflect
xbar=1 1
Function Evaluation #298 is [1.155D-25] at [1 1]
xr=[1 1], f(xr)=0.000000
Contract - inside
Function Evaluation #299 is [6.023D-27] at [1 1]
xc=1 1, f(xc)=0.000000
  > Perform Inside Contraction
Sort
```

Example #3 : use output function

In the following example, we show how to use the output command to create specialized outputs. These outputs may be used to create specific data files or make interactive graphic outputs.

We define the function "myoutputcmd", which takes the current state as its first argument. The state is a string which can contain "init", "iter" or "done", depending on the status of the optimization. The data input argument is a tlist, which contains the data associated with the current iteration. In this case, we use the fields to print a message in the console. As another example of use, we could format the message so that it uses LaTeX formatting rules, which may allow the user to directly copy and paste the output into a LaTeX report.

```

function [ f , index ] = rosenbrock ( x , index )
    f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

//
// myoutputcmd --
// This command is called back by the Nelder-Mead
// algorithm.
// Arguments
// state : the current state of the algorithm
// "init", "iter", "done"
// data : the data at the current state
// This is a tlist with the following entries:
// * x : the optimal vector of parameters
// * fval : the minimum function value
// * simplex : the simplex, as a simplex object
// * iteration : the number of iterations performed
// * funccount : the number of function evaluations
// * step : the type of step in the previous iteration
//
function myoutputcmd ( state , data )
    iter = data.iteration
    if ( state == "init" ) then
        mprintf ( "=====\n");
        mprintf ( "Initialization\n");
    elseif ( state == "done" ) then
        mprintf ( "=====\n");
        mprintf ( "End of Optimization\n");
    end
    fc = data.funccount
    fval = data.fval
    x = data.x
    simplex = data.simplex
    // Simplex is a data structure, which can be managed
    // by the optimsimplex class.
    ssize = optimsimplex_size ( simplex )
    mprintf ( "Iteration #%d, Feval #%d, Fval = %e, x = %s, Size = %e\n", iter, f
endfunction

nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_configure(nm,"-x0",[-1.2 1.0]');
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunvals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",1.0);
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",0);
nm = neldermead_configure(nm,"-verbosetermination",0);
nm = neldermead_configure(nm,"-outputcommand",myoutputcmd);
nm = neldermead_search(nm);
nm = neldermead_destroy(nm);

```

The previous script produces the following output.

```
=====
Initialization
Iteration #0, Feval #4, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #1, Feval #4, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #2, Feval #6, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #3, Feval #8, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #4, Feval #10, Fval = 9.999182e+000, x = -1.0125 0.78125, Size = 5.97
...
Iteration #155, Feval #296, Fval = 2.024754e-026, x = 1 1, Size = 4.601219e-013
Iteration #156, Feval #298, Fval = 6.871176e-027, x = 1 1, Size = 2.548515e-013
Iteration #157, Feval #300, Fval = 6.023002e-027, x = 1 1, Size = 2.814328e-013
=====
End of Optimization
Iteration #157, Feval #300, Fval = 6.023002e-027, x = 1 1, Size = 2.814328e-013
```

Bibliography

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R., The Computer Journal, 1965

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

"Discussion and correspondence: modification of the complex method of constrained optimization", J. A. Guin, The Computer Journal, 1968

"Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition", Kelley C. T., SIAM J. on Optimization, 1999

"Iterative Methods for Optimization", C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

"Algorithm AS47 - Function minimization using a simplex procedure", O'Neill, R., Applied Statistics, 1971

Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

See Also

optimbase, optimsimplex, nmplot

Nom

overview — An overview of the Nelder-Mead toolbox.

Purpose

The goal of this toolbox is to provide a Nelder-Mead direct search optimization method. That Nelder-Mead algorithm may be used in the following optimization context :

- there is no need to provide the derivatives of the objective function,
- the number of parameters is small (up to 10-20),
- there are bounds and/or non linear constraints.

Design

This package provides the following components :

- `neldermead` provides various Nelder-Mead variants and manages for Nelder-Mead specific settings, such as the method to compute the initial simplex, the specific termination criteria,
- `fminsearch` provides a simplified Nelder-Mead algorithm. Specific terminations criteria, initial simplex and auxiliary settings are automatically configured.
- `optimset`, `optimget` provide Scilab commands to emulate their Matlab counterparts.
- `optimplotfunccount`, `optimplotx` and `optimplotfval` provide plotting features for the `fminsearch` function.
- `nmpplot` provides a high-level component which provides directly output pictures for Nelder-Mead algorithm.

The current component is based on the following components

- `optimbase` provides an abstract class for a general optimization component, including the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the logging system, various termination criteria, the cost function, etc...
- `optimsimplex` provides a class to manage a simplex made of an arbitrary number of vertices, including the computation of a simplex by various methods (axes, regular, Pfeffer's, randomized bounds), the computation of the size by various methods (diameter, sigma +, sigma-, etc...),

Features

The following is a list of features the Nelder-Mead prototype algorithm currently provides :

- Provides 3 algorithms, including
 - Spendley et al. fixed shaped algorithm,
 - Nelder-Mead variable shape algorithm,
 - Box "complex" algorithm managing bounds and nonlinear inequality constraints based on arbitrary number of vertices in the simplex.
- Manage various simplex initializations
 - initial simplex given by user,
 - initial simplex computed with a length and along the coordinate axes,

- initial regular simplex computed with Spendley et al. formula
- initial simplex computed by a small perturbation around the initial guess point
- Manage cost function
 - optionnal additionnal argument
 - direct communication of the task to perform : cost function or inequality constraints
- Manage various termination criteria, including maximum number of iterations, tolerance on function value (relative or absolute),
 - tolerance on x (relative or absolute),
 - tolerance on standard deviation of function value (original termination criteria in [3]),
 - maximum number of evaluations of cost function,
 - absolute or relative simplex size,
- Manage the history of the convergence, including
 - history of function values,
 - history of optimum point,
 - history of simplices,
 - history of termination criterias,
- Provide a plot command which allows to graphically see the history of the simplices toward the optimum,
- Provide query features for the status of the optimization process number of iterations, number of function evaluations, status of execution, function value at initial point, function value at optimal point, etc...
- Kelley restart based on simplex gradient,
- O'Neill restart based on factorial search around optimum,

Example : Optimizing the Rosenbrock function

In the following example, one searches the minimum of the 2D Rosenbrock function. One begins by defining the function "rosenbrock" which computes the Rosenbrock function. The traditionnal initial guess [-1.2 1.0] is used. The initial simplex is computed along the axes with a length equal to 0.1. The Nelder-Mead algorithm with variable simplex size is used. The verbose mode is enabled so that messages are generated during the algorithm. After the optimization is performed, the optimum is retrieved with quiry features.

```
function y = rosenbrock (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

nm = neldermead_new ();
nm = neldermead_configure(nm,"-x0",[-1.2 1.0]');
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",0.1);
```

```
nm = neldermead_configure(nm, "-method", "variable");
nm = neldermead_configure(nm, "-verbose", 1);
nm = neldermead_configure(nm, "-function", rosenbrock);
nm = neldermead_search(nm);
xopt = neldermead_get(nm, "-xopt");
fopt = neldermead_get(nm, "-fopt");
historyfopt = neldermead_get(nm, "-historyfopt");
iterations = neldermead_get(nm, "-iterations");
historyxopt = neldermead_get(nm, "-historyxopt");
historysimplex = neldermead_get(nm, "-historysimplex");
fx0 = neldermead_get(nm, "-fx0");
status = neldermead_get(nm, "-status");
nm = neldermead_destroy(nm);
```

Authors

Michael Baudin, 2008-2009

Bibliography

“Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation”, Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

“A Simplex Method for Function Minimization”, Nelder, J. A. and Mead, R., The Computer Journal, 1965

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

“Convergence Properties of the Nelder--Mead Simplex Method in Low Dimensions”, Jeffrey C. Lagarias and James A. Reeds and Margaret H. Wright and Paul E. Wright, SIAM Journal on Optimization, 1998

“Compact numerical methods for computers : linear algebra and function minimisation”, Nash, J. C., Hilger, Bristol, 1979

“Iterative Methods for Optimization”, C. T. Kelley, 1999

“Iterative Methods for Optimization: Matlab Codes”, http://www4.ncsu.edu/~ctk/matlab_darts.html

“Sequential Simplex Optimization: A Technique for Improving Quality and Productivity in Research, Development, and Manufacturing”, Walters, Fred H. and Jr, Lloyd R. and Morgan, Stephen L. and Deming, Stanley N., 1991

“Numerical Recipes in C, Second Edition”, W. H. Press and Saul A. Teukolsky and William T. Vetterling and Brian P. Flannery, 1992

“Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition”, SIAM J. on Optimization, Kelley,, C. T., 1999

Matlab – fminsearch , <http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?access/helpdesk/help/techdoc/ref/fminsearch.html>

GAMS, A19A20 - description, <http://gams.nist.gov/serve.cgi/Module/NASHLIB/A19A20/11238/>

asa047.f, http://people.sc.fsu.edu/~burkardt/f77_src/asa047/asa047.f

optim1.f, <http://www.stat.uconn.edu/~mhchen/survbook/example51/optim1.f>

as47,f, <http://lib.stat.cmu.edu/apstat/47>

“Algorithm AS47 - Function minimization using a simplex procedure, O'Neill, R., 1971, Applied Statistics

TODO

- include Multi-Directionnal Search algorithm in the release
- include parabolic interpolation in the release
- provide a `neldermead_outputfunction` to make nice and easy outputs
- provide a stopping feature to the cost function and manage the `stop` output argument to the output functions of `fminsearch`
- provide a (quadratic) penalized version of `neldermead` for bound constrained problems

Nom

nmplot — Provides several direct search optimization algorithms based on the simplex method.

```
newobj = nmplot_new ()
this = nmplot_destroy (this)
this = nmplot_configure (this, key, value)
value = nmplot_cget (this, key)
this = nmplot_display ( this )
value = nmplot_get ( this , key )
this = nmplot_search ( this )
this = nmplot_restart ( this )
[ this , xdata , ydata , zdata ] = nmplot_contour ( this , xmin , xmax , ymin ,
this = nmplot_historyplot ( this , datafile , mytitle , myxlabel , myylabel )
this = nmplot_simplexhistory ( this , colorforeground , markforeground , markst
```

Description

This class provides several direct search optimization algorithms based on the simplex method.

The goal of this class is to provide a neldermead component with plotting features. It enables to make fast plots of the algorithm progress through the iterations.

It is a specialized neldermead class, with a specific output command. This output function allows to store the history of several datas through the iterations of the algorithm. These datas are :

- the history of the coordinates of the simplex ,
- the history of the function value (averaged on the vertices),
- the history of the minimum function value in the simplex,
- the history of the size of the simplex (as computed with the sigma+ method).

These data are stored into several data files during the optimization process. Several methods allows to plot the data stored into these data files.

Design

The nmplot component is built on top of the neldermead component. The -outputcommand option (of the neldermead class) is not available since the nmplot class uses its own output function. Additionnal options -simplexfn, -fbarfn, -foptfn and -sigmafz are provided, which allows to configure the file names where the data is stored.

The nmplot class can be considered as a sample test case of the -outputcommand option of the neldermead class. It gives an example of the situation where the user wants to get specialized outputs out of the neldermead class.

Functions

The following functions are available.

newobj = nmplot_new ()
Creates a new nmplot object.

newobj
The new object.

`this = nmpplot_destroy (this)`
 Destroy the given object.

`this`
 The current object.

`this = nmpplot_configure (this,key,value)`
 Configure the current object with the given value for the given key.

`this`
 The current object.

`key`
 the key to configure. The following keys are available.

`-verbose`
 set to 1 to enable verbose logging. (default is 0)

`-verbosetermination`
 set to 1 to enable verbose termination logging. (default is 0)

`-x0`
 the initial guess, as a $n \times 1$ column vector, where n is the number of variables.

`-maxfunevals`
 the maximum number of function evaluations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

`-maxiter`
 the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

`-tolfunabsolute`
 the absolute tolerance for the function value (default is 0.0).

`-tolfunrelative`
 the relative tolerance for the function value (default is %eps).

`-tolfunmethod`
 the method used for the tolerance on function value in the termination criteria.

The following values are available : "absolute+relative", "relative", "absolute", "disabled" (default is "disabled"). If this criteria is triggered, the status of the optimization is set to "tolf".

`-tolxabsolute`
 the absolute tolerance on x (default is 0.0).

`-tolxrelative`
 the relative tolerance on x (default is %eps).

`-tolxmethod`
 the method used for the tolerance on x in the termination criteria.

The following values are available : "relative", "absolute", "disabled" (default is "relative"). If this criteria is triggered, the status of the optimization is set to "tolx".

`-function`
 the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument
an additionnal argument, passed to the cost function.

-outputcommand
a command which is called back for output.

See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg
an additionnal argument, passed to the output command.

-numberofvariables
the number of variables to optimize (default is 0).

-storehistory
set to 1 to enable the history storing (default is 0).

-boundsmin
the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax
the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst
the number of inequality constraints (default is 0)

-method
the name of the algorithm to use. The following methods are available :

"fixed"
the Spendley et al. fixed simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"variable"
the Nelder-Mead variable simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"box"
the Box complex algorithm. This algorithm takes into account bounds and nonlinear inequality constraints.

-simplex0method
the method to use to compute the initial simplex. The first vertex in the simplex is always the initial guess associated with the -x0 option. The following methods are available :

"given"
the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"

the simplex is computed from the coordinate axes and the length associated with the `-simplex0length` option.

"spendley"

the simplex is computed so that it is regular with the length associated with the `-simplex0length` option (i.e. all the edges have the same length).

"pfeffer"

the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the `-simplex0deltausual` and `-simplex0deltazero`.

"randbounds"

the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the `-boxnbpoints` option.

`-coords0`

the coordinates of the vertices of the initial simplex. If the `-simplex0method` option is set to "given", these coordinates are used to compute the initial simplex. This matrix is expected to have shape $nbve \times n$ where $nbve$ is the number of vertices and n is the number of variables.

`-simplex0length`

the length to use when the initial simplex is computed with the "axes" or "spendley" methods. If the initial simplex is computed from "spendley" method, the length is expected to be a scalar value. If the initial simplex is computed from "axes" method, it may be either a scalar value or a vector of values, with rank n , where n is the number of variables.

`-simplex0deltausual`

the relative delta for non-zero parameters in "pfeffer" method. The default value is 0.05.

`-simplex0deltazero`

the absolute delta for non-zero parameters in "pfeffer" method. The default value is 0.0075.

`-rho`

the reflection coefficient. This parameter is used when the `-method` option is set to "fixed" or "variable". The default value is 1.0.

`-chi`

the expansion coefficient. This parameter is used when the `-method` option is set to "variable". The default value is 2.0.

`-gamma`

the contraction coefficient. This parameter is used when the `-method` option is set to "variable". The default value is 0.5.

`-sigma`

the shrinkage coefficient. This parameter is used when the `-method` option is set to "fixed" or "variable". The default value is 0.5.

`-tolfstdeviationmethod`

set to "enabled" to enable the termination criteria based on the standard deviation of the function values in the simplex. The default value is "disabled". If this criteria is triggered, the status of the optimization is set to "tolfstdev".

This criteria is suggested by Nelder and Mead.

-tolfstdeviation

the absolute tolerance on standard deviation. The default value is 0.0.

-tolsimplexizemethod

set to "disabled" to disable the tolerance on the simplex size. The default value is "enabled". If this criteria is triggered, the status of the optimization is set to "tolsize".

When this criteria is enabled, the values of the options -tolsimplexizeabsolute and -tolsimplexizerelative are used in the termination criteria. The method to compute the size is the "sigmaplus" method.

-tolsimplexizeabsolute

the absolute tolerance on the simplex size. The default value is 0.0.

-tolsimplexizerelative

the relative tolerance on the simplex size. The default value is %eps.

-tolssizedeltafvmethod

set to "enabled" to enable the termination criteria based on the size of the simplex and the difference of function value in the simplex. The default value is "disabled". If this criteria is triggered, the status of the optimization is set to "tolssizedeltafv".

This termination criteria uses the values of the options -tolsimplexizeabsolute and -toldeltafv. This criteria is identical to Matlab's fminsearch.

-toldeltafv

the absolute tolerance on the difference between the highest and the lowest function values.

-kelleystagnationflag

set to 1 to enable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. The default value is 0. If this criteria is triggered, the status of the optimization is set to "kelleystagnation".

-kelleynormalizationflag

set to 0 to disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the option -kelleystagnationalpha0 as is. Default value is 1, i.e. the simplex gradient of the initial simplex is taken into account in the stagnation detection.

-kelleystagnationalpha0

the parameter used in Kelley's stagnation detection. The default value is 1.e-4.

-restartflag

set to 1 to enable the automatic restart of the algorithm. Default value is 0.

-restartdetection

the method to detect if the automatic restart must be performed. The following methods are available :

"oneill"

the factorial local optimality test by O'Neill is used. If the test finds a local point which is better than the computed optimum, a restart is performed.

"kelley"

the sufficient decrease condition by O'Neill is used. If the test finds that the status of the optimization is "kelleystagnation", a restart is performed. This status may be generated if the -kelleystagnationflag option is set to 1.

The default method is "oneill".

-restartmax

the maximum number of restarts, when automatic restart is enabled via the `-restartflag` option. Default value is 3.

-restarteps

the absolute epsilon value used to check for optimality in the factorial O'Neill restart detection. The default value is %eps.

-restartstep

the absolute step length used to check for optimality in the factorial O'Neill restart detection. The default value is 1.0.

-restartsimplexmethod

the method to compute the initial simplex after a restart. The following methods are available.

"given"

the coordinates associated with the `-coords0` option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"

the simplex is computed from the coordinate axes and the length associated with the `-simplex0length` option.

"spendley"

the simplex is computed so that it is regular with the length associated with the `-simplex0length` option (i.e. all the edges have the same length).

"pfeffer"

the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the `-simplex0deltausual` and `-simplex0deltazero`.

"randbounds"

the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the `-boxnbpoints` option.

"oriented"

the simplex is computed so that it is oriented, as suggested by C.T. Kelley.

The default method is "oriented".

-boxnbpoints

the number of points in the initial simplex, when the `-restartsimplexmethod` option is set to "randbounds". The default value is so that the number of points is twice the number of variables of the problem.

-nbineqloops

the number of loops to perform in Box and Box-Guin algorithms to scale the trial point for function improvement or into the constraints. Default value is 10.

-ineqscaling
the scaling coefficient used to scale the trial point for function improvement or into the constraints. Default value is 0.5

-simplexfn
the name of the file containing the history of the simplex. Default value is the empty string.

-fbarfn
the name of the file containing the history of the function value, averaged on the vertices of the simplex. Default value is the empty string.

-foptfn
the name of the file containing the history of the minimum function value in the simplex. Default value is the empty string.

-sigmagn
the name of the file containing the history of the size of the simplex. Default value is the empty string.

value
the value.

value = nmpplot_cget (this,key)
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the name of the key to query. The list of available keys is the same as for the nmpplot_configure function.

value = nmpplot_get (this , key)
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the key to get.

The following keys are available :

-funevals
the number of function evaluations

-iterations
the number of iterations

-xopt
the x optimum, as a n x 1 column vector, where n is the number of variables.

-fopt
the optimum cost function value

-historyxopt
an array, with nbiter values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-historyfopt

an array, with nbiter values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-fx0

the function value for the initial guess

-status

a string containing the status of the optimization. See below for details about the optimization status.

-historysimplex

a matrix containing the history of the simplex during the iterations. This matrix has rank nbiter x nbve x n, where nbiter is the number of iterations, nbve is the number of vertices in the simplex and n is the number of variables.

-simplexopt

the optimum simplex. This is a simplex object, which is suitable for processing with the simplex interface.

-restartnb

the number of actual restarts performed.

Most fields are available only after an optimization has been performed with one call to the neldermead_search method.

`this = nmplot_display ( this )`

Display the current settings in the console.

`this`

The current object.

`this = nmplot_search ( this )`

Performs the optimization associated with the method associated with the -method option and find the optimum.

`this`

The current object.

If the -restartflag option is enabled, automatic restarts are performed, based on the -restartdetection option.

`this = nmplot_restart ( this )`

Restarts the optimization by updating the simplex and performing a new search.

`this`

The current object.

`[ this , xdata , ydata , zdata ] = nmplot_contour ( this , xmin , xmax , ymin , ymax , nx , ny )`

Plot the contours of the cost function. The cost function must be a function with two parameters.

`this`

The current object.

`xmin , xmax , ymin , ymax`

the bounds for the contour plot

`nx , ny`

the number of points in the directions x, y

xdata , ydata , zdata
vectors of data, as required by the contour function

nmplot_simplexhistory (this , colorforeground , markforeground , markstyle)
Plots the simplex history on the current graphic window. The colorforeground , markforeground , markstyle options are provided to produce fast plots. Specific settings can still be applied with the usual graphic features.

this
The current object.

colorforeground
the color of the foreground for the simplices. Default value is 5.

markforeground
the foreground mark for the simplices. Default value is 3.

markstyle
the mark style for the simplices. Default value is 9.

nmplot_historyplot (this , datafile , mytitle , myxlabel , myylabel)
Plots the history from the given data file on the current graphic window. The mytitle, myxlabel, myylabel options are provided as a way to produce plots faster. Specific settings can still be applied with the usual graphic features.

this
The current object.

datafile
the data file which contains the history. The file is expected to be formatted in a way similar to the files associated with the -fbarfn, -foptfn and -sigmafzn files. The default value is the value of the -foptfn option.

mytitle
the title of the plot. Default value is the empty string.

myxlabel
the x label for the plot. Default value is the empty string.

myylabel
the y label for the plot. Default value is the empty string.

[this , result] = nmplot_function (this , x , index)
Call the cost function and return the value.

this
The current object.

x
the point where the function is to be evaluated

index
optionnal, a flag to pass to the cost function (default = 1). See the section "The cost function" of the neldermead component for available values of index.

Example

In the following example, we use the fixed shape Spendley et al. simplex algorithm and find the minimum of a quadratic function. We begin by defining a quadratic function associated with 2 input variables. We then define an nmplot object and configure the object so that

the "fixed" shape simplex algorithm is used with the regular initial simplex associated with the "spendley" key. Four files are configured, which will contain the history of the simplex, the history of fbar, fopt and sigma through the iterations. The search is performed by the nmplot_search function, which writes the 4 data files during the iterations. The nmplot_contour function is called in order to compute the arrays xdata, ydata and zdata which are required as input to the contour function. The nmplot_simplexhistory then uses the history of the simplex, as stored in the rosenbrock.fixed.history.simplex.txt data file, and plot the various simplices on the contour plot. The nmplot_historyplot is used with the files rosenbrock.fixed.history.fbar.txt, rosenbrock.fixed.history.fopt.txt and rosenbrock.fixed.history.sigma.txt, which produces 3 plots of the history of the optimization algorithm through the iterations.

```
mprintf("Defining quadratic function...\n");
function y = quadratic (x)
    y = x(1)^2 + x(2)^2 - x(1) * x(2);
endfunction

mprintf("Creating nmplot object...\n");
nm = nmplot_new ();
nm = nmplot_configure(nm, "-numberofvariables", 2);
nm = nmplot_configure(nm, "-function", quadratic);
nm = nmplot_configure(nm, "-x0", [2.0 2.0]');
nm = nmplot_configure(nm, "-maxiter", 100);
nm = nmplot_configure(nm, "-maxfunevals", 300);
nm = nmplot_configure(nm, "-tolxrelative", 1.e-8);
nm = nmplot_configure(nm, "-simplex0method", "spendley");
nm = nmplot_configure(nm, "-method", "fixed");
//
// Setup output files
//
nm = nmplot_configure(nm, "-simplexfn", "rosenbrock.fixed.history.simplex.txt");
nm = nmplot_configure(nm, "-fbarfn", "rosenbrock.fixed.history.fbar.txt");
nm = nmplot_configure(nm, "-foptfn", "rosenbrock.fixed.history.fopt.txt");
nm = nmplot_configure(nm, "-sigmafn", "rosenbrock.fixed.history.sigma.txt");
//
// Perform optimization
//
mprintf("Searching for minimum...\n");
nm = nmplot_search(nm);
nmplot_display(nm);
// Plot the contours of the cost function and the simplex history
mprintf("Plotting contour...\n");
[nm , xdata , ydata , zdata ] = nmplot_contour ( nm , xmin = -2.0 , xmax = 4.0
f = scf());
contour ( xdata , ydata , zdata , [0.1 1.0 2.0 5.0 10.0 15.0 20.0] )
nmplot_simplexhistory ( nm );
mprintf("Plotting history of fbar...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.fbar.txt" , ...
    mytitle = "Function Value Average" , myxlabel = "Iterations" );
mprintf("Plotting history of fopt...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.fopt.txt" , ...
    mytitle = "Minimum Function Value" , myxlabel = "Iterations" );
mprintf("Plotting history of sigma...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.sigma.txt" , ...
    mytitle = "Maximum Oriented length" , myxlabel = "Iterations" );
```

```
deletefile("rosenbrock.fixed.history.simplex.txt");
deletefile("rosenbrock.fixed.history.fbar.txt");
deletefile("rosenbrock.fixed.history.fopt.txt");
deletefile("rosenbrock.fixed.history.sigma.txt");
nm = nmplot_destroy(nm);
```

TODO

- add an example

Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

See Also

neldermead

Nom

`optimget` — Queries an optimization data structure.

```
val = optimget ( options , key )  
val = optimget ( options , key , value )
```

Description

This function allows to make queries on an existing optimization data structure. This data structure must have been created and updated by the `optimset` function. The `optimget` allows to retrieve the value associated with a given key.

In the following, we analyse the various ways to call the `optimget` function.

The following calling sequence

```
val = optimget(options , key)
```

returns the value associated with the given key. The key is expected to be a string. We search the key among the list of all possible fields in the `options` data structure. In this search, the case is ignored. The key which matches a possible field is returned. Some letters of the field may be dropped, provided that the matching field is unique. For example, the key "MaxF" corresponds to the field "MaxFunEval". But the key "Tol" corresponds both to the "TolX" and "TolFun" fields and therefore will generate an error.

The following calling sequence

```
val = optimget(options, key, value)
```

allows to manage default value for optimization parameters. Indeed, if the field corresponding to the key is empty (i.e. has not been set by the user), the input argument `value` is returned. Instead, if the field corresponding the the key is not empty (i.e. has been set by the user), the parameter stored in the `options` argument is returned.

Parameters

`options`

A struct which contains the optimization fields.

`key`

A string corresponding to a field of the optimization structure.

`value`

A real default value.

`val`

The real value corresponding to the key.

Example #1

In the following example, we create an optimization structure and set the "TolX" field to 1.e-12. Then we use `optimget` to get back the value.

```
op = optimset();
op = optimset(op, 'TolX', 1.e-12);
val = optimget(op, 'TolX'); // val is 1.e-12
```

Example #2

In the following example, we create an empty optimization structure. Then we use `optimget` to get back the value corresponding to the "TolX" field, with 1.e-5 as the default value. Since the field is empty, we retrieve 1.e-5.

```
op = optimset();
val = optimget(op, 'TolX' , 1.e-5); // val = 1.e-5
```

Example #3

In the following example, we create an optimization structure and set the "TolX" field to 1.e-12. Then we use `optimget` to get back the value corresponding to the "TolX" field, with 1.e-5 as the default value. Since the field is not empty, we retrieve 1.e-12.

```
op = optimset();
op = optimset(op, 'TolX', 1.e-12);
val = optimget(op, 'TolX' , 1.e-5); // val = 1.e-12
```

Example #4

In the following example, we create an optimization structure and configure the maximum number of function evaluations to 1000. Then we query the data structure, giving only the "MaxF" key to the `optimget` function. Since that corresponds only to the "MaxFunEvals" field, there is only one match and the function returns 10000.

```
op = optimset();
op = optimset(op, 'MaxFunEvals' , 1000);
val = optimget(op, 'MaxF'); // val = 1000
```

Authors

Michael Baudin - Digiteo - 2009

See Also

[optimset](#)

Nom

optimplotfunccount — Plot the number of function evaluations of an optimization algorithm

```
optimplotfunccount ( x , optimValues , state )
```

Description

This function creates and updates a plot of the number of function evaluations, depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

Example

In the following example, we use the `optimplotfunccount` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfunccount );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

Authors

Michael Baudin - Digiteo - 2009

See Also

`optimset` , `fminsearch` , `optimplotx` , `optimplotfval`

Nom

optimplotfval — Plot the function value of an optimization algorithm

```
optimplotfval ( x , optimValues , state )
```

Description

This function creates and updates a plot of the function value, depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

Example

In the following example, we use the `optimplotfval` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfval );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

Authors

Michael Baudin - Digiteo - 2009

See Also

`optimset` , `fminsearch` , `optimplotx` , `optimplotfunccount`

Nom

optimplotx — Plot the value of the parameters of an optimization algorithm

```
optimplotx ( x , optimValues , state )
```

Description

This function creates and updates a plot of the value of the parameters depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

Example

In the following example, we use the `optimplotx` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
    y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotx );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

Authors

Michael Baudin - Digiteo - 2009

See Also

`optimset` , `fminsearch` , `optimplotfval` , `optimplotfunccount`

Nom

optimset — Configures and returns an optimization data structure.

```
options = optimset ()
options = optimset ( funname )
options = optimset ( key , value )
options = optimset ( key1 , value1 , key2 , value2, ... )
options = optimset ( oldoptions , key , value )
```

Description

This function creates or updates a data structure which can be used on modify the behaviour of optimization methods. The goal of this function is to manage the "options" data structure, which is a struct with a set of fields (for example, "MaxFunEvals", "MaxIter", etc...). The user can create a new structure with empty fields or create a new structure with default fields which correspond to a particular algorithm. The user can also configure each field and set it to a particular value. Finally, the user pass the structure to an optimization function so that the algorithm uses the options configured by the user.

In the following, we analyse the various ways to call the optimset function.

The following calling sequence

```
options = optimset ()
```

creates a new data structure where the fields have been set to the empty matrix (i.e. []).

The following calling sequence

```
options = optimset ( funname )
```

creates a new data structure where the default parameters which correspond to the "funname" function have been set. For example,

```
options = optimset ( "fminsearch" )
```

returns a new data structure where the default parameters which correspond to the "fminsearch" function have been set.

The following calling sequence

```
options = optimset ( oldoptions , key , value )
```

creates a new data structure where all fields from the "oldoptions" structure have been copied, except the field corresponding to the "key", which has been set to "value".

Parameters

key

a string. The following keys are available :

- "Display"
- "FunValCheck"
- "MaxFunEvals"
- "MaxIter"
- "OutputFcn"
- "PlotFcns"

value

the value of the parameter

options

A struct which contains the following fields. By default, all fields are empty. Specific settings are associated with a particular function name.

options.Display

The verbose level. The default value is "notify". The following is a list of available verbose levels.

- options.Display = "off": the algorithm displays no message at all.
- options.Display = "notify": the algorithm displays message if the termination criteria is not reached at the end of the optimization. This may happen if the maximum number of iterations of the maximum number of function evaluations is reached and warns the user of a convergence problem.
- options.Display = "final": the algorithm displays a message at the end of the optimization, showing the number of iterations, the number of function evaluations and the status of the optimization. This option includes the messages generated by the "notify" option i.e. warns in case of a convergence problem.
- options.Display = "iter": the algorithm displays a one-line message at each iteration. This option includes the messages generated by the "notify" option i.e. warns in case of a convergence problem. It also includes the message generated by the "final" option.

options.FunValCheck

A boolean to enable the checking of function values.

options.MaxFunEvals

The maximum number of evaluations of the cost function.

options.MaxIter

The maximum number of iterations.

options.OutputFcn

A function which is called at each iteration to print out intermediate state of the optimization algorithm (for example into a log file).

options.PlotFcns

A function which is called at each iteration to plot the intermediate state of the optimization algorithm (for example into a 2D graphic).

options.TolFun

The absolute tolerance on function value.

options.TolX

The absolute tolerance on the variable x.

funname

A string containing the name of an optimization function which takes the options structure as input argument. Currently, the only possible value is "fminsearch".

Design

Most optimization algorithms require many algorithmic parameters such as the number of iterations or the number of function evaluations. If these parameters are given to the optimization function as input parameters, this forces both the user and the developer to manage many input parameters. For example, the "optim" function provides more than 20 input arguments. The goal of the optimset function is to simplify the management of input arguments, by gathering all the parameters into a single data structure.

While the current implementation of the "optimset" function only supports the fminsearch function, it is designed to be extended to as many optimization function as required. Because all optimization algorithms do not require the same parameters, the data structure aims at remaining flexible. But, most of the time, most parameters are the same from algorithm to algorithm, for example, the tolerance parameters which drive the termination criteria are often the same, even if the termination criteria itself is not the same.

Output and plot functions

The "OutputFcn" and "PlotFcns" options accept as argument a function (or a list of functions). In the client optimization algorithm, this output or plot function is called back once per iteration. It can be used by the user to display a message in the console, write into a file, etc...

The output or plot function is expected to have the following definition:

```
function myfun ( x , optimValues , state )
```

where the input parameters are:

- `x`: the current point
- `optimValues`: a struct which contains the following fields
 - `optimValues.funccount`: the number of function evaluations
 - `optimValues.fval`: the best function value
 - `optimValues.iteration`: the current iteration number
 - `optimValues.procedure`: the type of step performed. This string depends on the specific algorithm see fminsearch for details.
- `state`: the state of the algorithm. The following states are available : "init", "iter" and "done".
 - "init", when the algorithm is initializing,
 - "iter", when the algorithm is performing iterations,
 - "done", when the algorithm is terminated.

Example #1

In the following example, we create an empty optimization structure.

```
op = optimset ( )
```

Example #2

In the following example, we create an optimization structure with all fields set to specific settings.

```
op = optimset ( "Display", "iter", ...  
               "FunValCheck", "on", ...  
               "MaxFunEvals", 100, ...  
               "MaxIter", 110, ...  
               "OutputFcn", myoutputfun, ...  
               "PlotFcns", myplotfun, ...  
               "TolFun", 1.e-12, ...  
               "TolX", 1.e-13)
```

Example #3

In the following example, we create an optimization structure with all fields set to the default settings for the "fminsearch" optimization function.

```
op = optimset ( "fminsearch")
```

Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

See Also

[optimget](#)

Chapter 3. Optimization base

Nom

optimbase — Provides an abstract class for a general optimization component.

```
newobj = optimbase_new ()
this = optimbase_destroy (this)
this = optimbase_configure (this,key,value)
value = optimbase_cget (this,key)
value = optimbase_get (this,key)
this = optimbase_set ( this , key , value )
[ this , isok ] = optimbase_checkbounds ( this )
[ opt , isok ] = optimbase_checkx0 ( this )
this = optimbase_display ( this )
[ this , f , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] )
[ this , f , c , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] )
[ this , f , g , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] )
[ this , f , g , c , gc , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] )
[ this , hasbounds ] = optimbase_hasbounds ( this )
[ this , hascons ] = optimbase_hasconstraints ( this )
[ this , hasnlcons ] = optimbase_hasnlcons ( this )
value = optimbase_histget ( this , iter , key )
this = optimbase_histset ( this , iter , key , value )
this = optimbase_incriter ( this )
[ this , isfeasible ] = optimbase_isfeasible ( this , x )
this = optimbase_log (this,msg)
optimbase_outputcmd ( this , state , data )
data = optimbase_outstruct ( this )
[ this , p ] = optimbase_proj2bnds ( this , x )
this = optimbase_stoplog ( this , msg )
[this , terminate , status] = optimbase_terminate (this , previousfopt , currentfopt , currentcost )
this = optimbase_checkcostfun ( this )
[ this , isfeasible ] = optimbase_isinbounds ( this , x )
[ this , isfeasible ] = optimbase_isinnonlinconst ( this , x )
```

Purpose

The goal of this component is to provide a building block for optimization methods. The goal is to provide a building block for a large class of specialized optimization methods. This component manages

- the number of variables,
- the minimum and maximum bounds,
- the number of non linear inequality constraints,
- the cost function,
- the logging system,
- various termination criteria,
- etc...

Design

This toolbox is designed with Oriented Object ideas in mind.

Features

The following is a list of features the Optimbase toolbox currently provides :

- Manage cost function
 - optionnal additionnal argument
 - direct communication of the task to perform : cost function or inequality constraints
- Manage various termination criteria, including
 - maximum number of iterations,
 - tolerance on function value (relative or absolute),
 - tolerance on x (relative or absolute),
 - maximum number of evaluations of cost function,
- Manage the history of the convergence, including
 - history of function values,
 - history of optimum point.
- Provide query features for
 - the status of the optimization process,
 - the number of iterations,
 - the number of function evaluations,
 - function value at initial point,
 - function value at optimal point,
 - the optimum parameters,
 - etc...

Description

This set of commands allows to manage an abstract optimization method. The goal is to provide a building block for a large class of specialized optimization methods. This component manages the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the logging system, various termination criteria, the cost function, etc...

The optimization problem to solve is the following

```
min f(x)
l_i <= x_i <= h_i, i = 1,n
g_i(x) <= 0, i = 1,nbineq
```

where

n
number of variables

nbineq
number of inequality constraints

Functions

The following functions are available.

newobj = optimbase_new ()
Creates a new optimization object.

newobj
The new object.

this = optimbase_destroy (this)
Destroy the given object.

this
The current object.

this = optimbase_configure (this,key,value)
Configure the current object with the given value for the given key.

this
The current object.

key
the key to configure. The following keys are available.

-verbose
set to 1 to enable verbose logging. (default is 0)

-verbosetermination
set to 1 to enable verbose termination logging. (default is 0)

-x0
the initial guess.

-maxfunevals
the maximum number of function evaluations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

-maxiter
the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

-tolfunabsolute
the absolute tolerance for the function value (default is 0.0).

-tolfunrelative
the relative tolerance for the function value (default is %eps).

-tolfunmethod
the method used for the tolerance on function value in the termination criteria.

The following values are available : %t, %f (default is %f). If this criteria is triggered, the status of the optimization is set to "tolf".

-tolxabsolute
the absolute tolerance on x (default is 0.0).

-tolxrelative
the relative tolerance on x (default is %eps).

-tolxmethod

the method used for the tolerance on x in the termination criteria.

The following values are available : %t, %f (default is %t). If this criteria is triggered, the status of the optimization is set to "tolx".

-function

the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument

an additionnal argument, passed to the cost function.

-outputcommand

a command which is called back for output.

See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg

an additionnal argument, passed to the output command.

-numberofvariables

the number of variables to optimize (default is 0).

-storehistory

set to %t to enable the history storing (default is %f).

-boundsmin

the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax

the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst

the number of inequality constraints (default is 0)

-logfile

the name of the log file

-withderivatives

set to %t if the algorithm uses derivatives. Default is %f.

value

the value.

`value = optimbase_cget (this,key)`

Get the value for the given key. If the key is unknown, generates an error.

this

The current object.

key

the name of the key to query. The list of available keys is the same as for the `optimbase_configure` function.

`[ this , isok ] = optimbase_checkbounds ( this )`

Check if the bounds are consistent and puts an error message if not. One could generate an error, but errors are not testable with the current system.

`this`

The current object.

`[ opt , isok ] = optimbase_checkx0 ( this )`

Returns %T if the initial guess is consistent with the bounds and the non linear inequality constraints, if any.

`this`

The current object.

`this = optimbase_display ( this )`

Display the current settings in the console.

`this`

The current object.

`[this , f , index [, data]] = optimbase_function (this , x , index [, data]), [this , f , c , index [, data]] = optimbase_function (this , x , index [, data]), [this , f , g , index [, data]] = optimbase_function (this , x , index [, data]), [this , f , g , c , gc , index [, data]] = optimbase_function (this , x , index [, data])`

Call the cost function and return the required results.

If a cost function additionnal argument is defined in current object, pass it to the function as the last argument. See below for details.

`this`

The current object.

`x`

the current point, as a column vector

`index`

what value to compute.

See below in the section "Cost function" for details on this index.

`f`

the value of the cost function

`g`

the gradient of the cost function

`c`

the nonlinear, positive, inequality constraints

`gc`

the gradient of the nonlinear, positive, inequality constraints

`data`

an optionnal user-provided argument

`this = optimbase_set ( this , key , value )`

Set the value for the given key. If the key is unknown, generates an error.

`this`

The current object.

key

the key to set

The following keys are available :

-funevals

the number of function evaluations

-iterations

the number of iterations

-xopt

the x optimum

-fopt

the optimum cost function value

-historyxopt

an array, with nbiter values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-historyfopt

an array, with nbiter values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-fx0

the function value for the initial guess

-status

a string containing the status of the optimization

value

the value to set

value = optimbase_get (this,key)

Get the value for the given key. If the key is unknown, generates an error. This command corresponds with options which are not available directly to the optimbase_configure function, but are computed internally.

this

The current object.

key

the name of the key to query.

The list of available keys is the same as the optimbase_set function.

[this , hasbounds] = optimbase_hasbounds (this)

Returns %T if current problem has bounds.

this

The current object.

[this , hascons] = optimbase_hasconstraints (this)

Returns %T if current problem has bounds constraints, linear constraints or non linear constraints.

this

The current object.

[this , hasnlcons] = optimbase_hasnlcons (this)

Returns %T if current problem has non linear constraints.

this

The current object.

this = optimbase_histset (this , iter , key , value)

Set the history value at given iteration for the given key. If the key is unknown, generates an error.

this

The current object.

iter

the iteration number to get

key

the name of the key to query.

The list of available keys is the following : "-xopt", "-fopt".

value

the value to set

value = optimbase_histget (this , iter , key)

Returns the history value at the given iteration number for the given key. If the key is unknown, generates an error.

this

The current object.

iter

the iteration number to get

key

the name of the key to query.

The list of available keys is the same as the optimbase_histset function.

this = optimbase_incrater (this)

Increments the number of iterations.

this

The current object.

[this , isfeasible] = optimbase_isfeasible (this , x)

Returns 1 if the given point satisfies bounds constraints and inequality constraints. Returns 0 if the given point is not in the bounds. Returns -1 if the given point does not satisfies inequality constraints.

this

The current object.

x

the current point

this = optimbase_log (this,msg)

If verbose logging is enabled, prints the given message in the console. If verbose logging is disabled, does nothing. If the -lofgile option has been set, writes the message into the file instead of writing to the console. If the console is too slow, writing into a file can be a solution, since it is very fast.

this
The current object.

msg
the message to print

optimbase_outputcmd (this , state , data)
Calls back user's output command.

this
The current object.

data = optimbase_outstruct (this)
Returns a tlist with basic optimization fields. This tlist is suitable for use as an input argument of the output function. This tlist may be enriched by children (specialize) optimization methods.

this
The current object.

[this , p] = optimbase_proj2bnds (this , x)
Returns a point, which is the projection of the given point into the bounds.

this
The current object.

x
the current point

this = optimbase_stoplog (this , msg)
Prints the given stopping rule message if verbose termination is enabled. If verbose termination is disabled, does nothing.

this
The current object.

msg
the message to print

[this , terminate , status] = optimbase_terminate (this , previousfopt , currentfopt , previousxopt , currentxopt)
Returns 1 if the algorithm terminates. Returns 0 if the algorithm must continue. If the -verbosetermination option is enabled, messages are printed detailing the termination intermediate steps. The optimbase_terminate function takes into account the number of iterations, the number of evaluations of the cost function, the tolerance on x and the tolerance on f. See below in the section "Termination" for more details.

this
The current object.

previousfopt
the previous value of the cost function

currentfopt
the current value of the cost function

previousxopt
the previous x optimum

currentxopt
the current x optimum

terminate

%t if the algorithm must terminate, %f if the algorithm must continue

status

if terminate = %t, the detailed status of the termination, as a string. If terminate = %f, the status is "continue".

The following status are available :

"maxiter"

the maximum number of iterations, provided by the -maxiter option, is reached.

"maxfuneval"

the maximum number of function evaluations, provided by the -maxfunevals option, is reached

"tolf"

the tolerance on the function value is reached. This status is associated with the -tolfun-method, -tolfunabsolute and -tolfunrelative options.

"tolx"

the tolerance on x is reached. This status is associated with the -tolxmethod, -tolxabsolute and -tolxrelative options.

this = optimbase_checkcostfun (this)

Check that the cost function is correctly connected. Generate an error if there is one. Takes into account for the cost function at the initial guess x0 only. Checks that all values of the index argument are valid. If there are nonlinear constraints, check that the matrix has the correct shape.

This function requires at least one call to the cost function to make the necessary checks.

this

The current object.

[this , isfeasible] = optimbase_isinbounds (this , x)

Returns isfeasible = %t if the given point satisfies bounds constraints. Returns isfeasible = %f if the given point is not in the bounds.

this

The current object.

isfeasible

a boolean

[this , isfeasible] = optimbase_isinnonlinconst (this , x)

Returns isfeasible = %t if the given point satisfies the nonlinear constraints. Returns isfeasible = %f if the given point does not satisfy the nonlinear constraints.

this

The current object.

isfeasible

a boolean

The cost function

The -function option allows to configure the cost function. The cost function is used, depending on the context, to compute the cost, the nonlinear inequality positive constraints, the gradient of the function and the gradient of the nonlinear inequality constraints.

The cost function can also be used to produce outputs and to terminate an optimization algorithm.

In the following, the variables are

- **f**: scalar, the objective,
- **g**: row matrix, the gradient of the objective,
- **c**: row matrix, the constraints,
- **gc**: matrix, the gradient of the constraints.

Each calling sequence of the `optimbase_function` function corresponds to a specific calling sequence of the user-provided cost function.

- If the `-withderivatives` is false and there is no nonlinear constraint, the calling sequence is

```
[ this , f , index ] = optimbase_function ( this , x , index )
```

which corresponds to the cost functions

```
function [ f , index ] = costf ( x , index )
function [ f , index , data ] = costf ( x , index , data ) // if there is a -c
```

- If the `-withderivatives` is false and there are nonlinear constraints, the calling sequence is

```
[ this , f , c , index ] = optimbase_function ( this , x , index )
```

which corresponds to the cost functions

```
function [ f , c , index ] = costf ( x , index )
function [ f , c , index , data ] = costf ( x , index , data ) // if there is
```

- If the `-withderivatives` is true and there is no nonlinear constraint, the calling sequence is

```
[ this , f , g , index ] = optimbase_function ( this , x , index )
```

which corresponds to the cost functions

```
function [ f , g , index ] = costf ( x , index )
function [ f , g , index , data ] = costf ( x , index , data ) // if there is
```

- If the `-withderivatives` is true and there is are nonlinear constraints, the calling sequence is

```
[ this , f , g , c , gc , index ] = optimbase_function ( this , x , index )
```

which corresponds to the cost functions

```
function [ f , g , c , gc , index ] = costf ( x , index )
```



```
function [ f , g , c , gc , index , data ] = costf ( x , index , data ) // if
```

Each calling sequence corresponds to a particular class of algorithms, including for example

- unconstrained, derivative-free algorithms,
- nonlinearly constrained, derivative-free algorithms,
- unconstrained, derivative-based algorithms,
- nonlinearlyconstrained, derivative-based algorithms,
- etc...

The current component is designed in order to be able to handle many situations.

The index input parameter has the following meaning.

- index = 1: nothing is to be computed, the user may display messages, for example
- index = 2: compute f
- index = 3: compute g
- index = 4: compute f and g
- index = 5: compute c
- index = 6: compute f and c
- index = 7: compute f, g, c and gc

The index output parameter has the following meaning.

- index > 0: everything is fine,
- index = 0: the optimization must stop,
- index < 0: one function could not be avaluated.

The output function

The option -outputcommand allows to configure a command which is called back at the start of the optimization, at each iteration and at the end of the optimization.

The output function must have the following header

```
function outputcmd(state, data, myobj)
```

where

state

a string representing the current state of the algorithm. Available values are "init", "iter", "done".

data

a tlist containing at least the following entries

x

the current optimum

fval
the current function value

iteration
the current iteration index

funccount
the number of function evaluations

myobj
a user-defined parameter.

This input parameter is defined with the `-outputcommandarg` option.

The output function may be used when debugging the specialized optimization algorithm, so that a verbose logging is produced. It may also be used to write one or several report files in a specialized format (ASCII, LaTeX, Excel, Hdf5, etc...). The user-defined parameter may be used in that case to store file names or logging options.

The data `tlist` argument may contain more fields than the current presented ones. These additional fields may contain values which are specific to the specialized algorithm, such as the simplex in a Nelder-Mead method, the gradient of the cost function in a BFGS method, etc...

Termination

The current component takes into account for several generic termination criterias. Specialized termination criterias should be implemented in specialized optimization algorithms, by calling the `optimbase_termination` function and adding external criterias, rather than by modification of this function.

The `optimbase_terminate` function uses a set of rules to compute if the termination occurs, which leads to an optimization status which is equal to one of the following : "continue", "maxiter", "maxfunevals", "tolf", "tolx". The set of rules is the following.

- By default, the status is "continue" and the terminate flag is 0.
- The number of iterations is examined and compared to the `-maxiter` option : if the following condition

```
iterations >= maxiter
```

is true, then the status is set to "maxiter" and terminate is set to %t.

- The number of function evaluations and compared to the `-maxfunevals` option is examined : if the following condition

```
funevals >= maxfunevals
```

is true, then the status is set to "maxfuneval" and terminate is set to %t.

- The tolerance on function value is examined depending on the value of the `-tolfunmethod`.

"disabled"
then the tolerance on f is just skipped.

"enabled"
if the following condition

```
abs(currentfopt) < tolfunrelative * abs(previousfopt) + tolfunabsolute
```

is true, then the status is set to "tolf" and terminate is set to %t.

The relative termination criteria on the function value works well if the function value at optimum is near zero. In that case, the function value at initial guess `fx0` may be used as `previousfopt`.

The absolute termination criteria on the function value works if the user has an accurate idea of the optimum function value.

- The tolerance on `x` is examined depending on the value of the `-tolxmethod`.

%f
then the tolerance on `x` is just skipped.

%t
if the following condition

```
norm(currentxopt - previousxopt) < tolxrelative * norm(currentxopt) + tolxabsolute
```

is true, then the status is set to "tolx" and terminate is set to %t.

The relative termination criteria on `x` works well if `x` at optimum is different from zero. In that case, the condition measures the distance between two iterates.

The absolute termination criteria on `x` works if the user has an accurate idea of the scale of the optimum `x`. If the optimum `x` is near 0, the relative tolerance will not work and the absolute tolerance is more appropriate.

Example : Setting up an optimization

In the following example, one searches the minimum of the 2D Rosenbrock function. One begins by defining the function "rosenbrock" which computes the Rosenbrock function. The traditional initial guess `[-1.2 1.0]` is used. The initial simplex is computed along the axes with a length equal to 0.1. The Nelder-Mead algorithm with variable simplex size is used. The verbose mode is enabled so that messages are generated during the algorithm. After the optimization is performed, the optimum is retrieved with query features.

```
function [ f , index ] = rosenbrock ( x , index )
    f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

opt = optimbase_new();
opt = optimbase_configure(opt,"-numberofvariables",2);
nbvar = optimbase_cget(opt,"-numberofvariables");
opt = optimbase_configure(opt,"-function",rosenbrock);
[ this , f , index ] = optimbase_function ( opt , [0.0 0.0] , index );
opt = optimbase_destroy(opt);
```

Authors

Michael Baudin, 2008-2009

TODO

- manage equality constraints
- manage linear constraints
- manage quadratic objective
- manage linear objective
- manage linear inequality constraints
- manage non linear equality constraints
- manage linear equality constraints
- rename -outputcommand to -outputfunction

Chapter 4. Optimization simplex

Nom

optimsimplex — Manage a simplex with arbitrary number of points.

```
[ newobj [, data] ] = optimsimplex_new ( coords , fun [, data] )
[ newobj [, data] ] = optimsimplex_new ( "axes" , x0 , fun , len [, data] )
[ newobj [, data] ] = optimsimplex_new ( "spendley" , x0 , fun , len [, data] )
[ newobj [, data] ] = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , de
[ newobj [, data] ] = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , l
[ newobj [, data] ] = optimsimplex_new ( "oriented" , simplex0 , fun [, data] )
this = optimsimplex_destroy (this)
this = optimsimplex_setall ( this , simplex )
this = optimsimplex_setallfv ( this , fv )
this = optimsimplex_setallx ( this , x )
this = optimsimplex_setfv ( this , ive , fv )
this = optimsimplex_setn ( this , n )
this = optimsimplex_setnbve ( this , nbve )
this = optimsimplex_setve ( this , ive , fv , x )
this = optimsimplex_setx ( this , ive , x )
simplex = optimsimplex_getall ( this )
fv = optimsimplex_getallfv ( this )
x = optimsimplex_getallx ( this )
fv = optimsimplex_getfv ( this , ive )
n = optimsimplex_getn ( this )
nbve = optimsimplex_getnbve ( this )
vertex = optimsimplex_getve ( this , ive )
x = optimsimplex_getx ( this , ive )
sicenter = optimsimplex_center ( this )
optimsimplex_check ( this )
[ this , data ] = optimsimplex_computefv ( this , fun , data )
df = optimsimplex_deltafv ( this )
dfm = optimsimplex_deltafvmax ( this )
m = optimsimplex_dirmat ( this )
m = optimsimplex_fvmean ( this )
sd = optimsimplex_fvstdev ( this )
v = optimsimplex_fvvariance ( this )
[ g , data ] = optimsimplex_gradientfv ( this , fun , method , data )
optimsimplex_print ( this )
[ r , data ] = optimsimplex_reflect ( this , fun , data )
[ this , data ] = optimsimplex_shrink ( this , fun , sigma , data )
ssize = optimsimplex_size ( this , method )
this = optimsimplex_sort ( this )
str = optimsimplex_tostring ( this )
cen = optimsimplex_xbar ( this , iexcl )
```

Purpose

The goal of this component is to provide a building block for optimization algorithms based on a simplex. The optimsimplex package may be used in the following optimization methods :

- the Spendley et al. simplex method,
- the Nelder-Mead method,
- the Box algorithm for constrained optimization,
- the multi-dimensional search by Virginia Torczon,
- etc ...

Design

This toolbox is designed with Oriented Object ideas in mind.

Features

The following is a list of features the Nelder-Mead prototype algorithm currently provides :

- Manage various simplex initializations
 - initial simplex given by user,
 - initial simplex computed with a length and along the coordinate axes,
 - initial regular simplex computed with Spendley et al. formula,
 - initial simplex computed by a small perturbation around the initial guess point,
 - initial simplex computed from randomized bounds.
- sort the vertices by increasing function values,
- compute the standard deviation of the function values in the simplex,
- compute the simplex gradient with forward or centered differences,
- shrink the simplex toward the best vertex,
- etc...

Description

This set of commands allows to manage a simplex made of $k \geq n+1$ points in a n -dimensional space. This component is the building block for a class of direct search optimization methods such as the Nelder-Mead algorithm or Torczon's Multi-Dimensionnal Search.

A simplex is designed as a collection of $k \geq n+1$ vertices. Each vertex is made of a point and a function value at that point.

The simplex can be created with various shapes. It can be configured and queried at will. The simplex can also be reflected or shrunk. The simplex gradient can be computed with a order 1 forward formula and with a order 2 centered formula.

The `optimsimplex_new` function allows to create a simplex. If vertices coordinates are given, there are registered in the simplex. If a function is provided, it is evaluated at each vertex. The `optimsimplex_destroy` function destroys the object and frees internal memory. Several functions allow to create a simplex with special shapes, including axes-by-axes (`optimsimplex_axes`), regular (`optimsimplex_spendley`), randomized bounds simplex with arbitrary nbve vertices (`optimsimplex_randbounds`) and an heuristical small variation around a given point (`optimsimplex_pfeffer`).

In the following functions, simplices and vertices are, depending on the functions either input or output arguments. The following general principle have been used to manage the storing of the coordinates of the points.

- The vertices are stored row by row, while the coordinates are stored column by column. This implies the following rules.
- The coordinates of a vertex are stored in a row vector, i.e. a $1 \times n$ matrix where n is the dimension of the space.

- The function values are stored in a column vector, i.e. a nbve x 1 matrix where nbve is the number of vertices.

Functions

The following functions are available.

`newobj = optimsimplex_new ( )`

Creates a new simplex object. All input arguments are optional. If no input argument is provided, returns an empty simplex object.

The following is a complete list of available calling sequences.

```
newobj = optimsimplex_new ( coords )
newobj = optimsimplex_new ( coords , fun )
[ newobj , data ] = optimsimplex_new ( coords , fun , data )
```

`newobj`

The new object.

`coords`

optional, matrix of point coordinates in the simplex.

The coords matrix is expected to be a nbve x n matrix, where n is the dimension of the space and nbve is the number of vertices in the simplex, with $\text{nbve} \geq n+1$.

`fun`

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

where `x` is a row vector.

`data`

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`this = optimsimplex_new ( "axes" , x0 )`

Creates a new simplex object so that it is computed axis by axis, with the given length.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "axes" , x0 , fun )
this = optimsimplex_new ( "axes" , x0 , fun , len )
```



```
[ this , data ] = optimsimplex_new ( "axes" , x0 , fun , len , data )
```

this

The current simplex object.

x0

the initial point, as a row vector.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

len

optional, the length of the simplex. The default length is 1.0. If length is a value, that unique length is used in all directions. If length is a vector with n values, each length is used with the corresponding direction.

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

```
this = optimsimplex_new ( "pfeffer" , x0 )
```

Creates a new simplex so that it is computed from Pfeffer's method, i.e. a relative delta for non-zero values and an absolute delta for zero values.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "pfeffer" , x0 , fun )
this = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual )
this = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , deltazero )
[ this , data ] = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , del
```

this

The current simplex object.

x0

the initial point, as a row vector.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

deltusual

optional, the absolute delta for non-zero values. The default value is 0.05.

deltazero

optional, the absolute delta for zero values. The default value is 0.0075.

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

```
this = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , boundsmax , nbpoints )
```

Creates a new simplex so that it is computed by taking the bounds into account with random scaling. The number of vertices in the simplex is arbitrary.

The following is a complete list of available calling sequences.

```
[ this , data ] = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , b
```

this

The current simplex object.

x0

the initial point, as a row vector. It is the first vertex in the simplex.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

boundsmin

array of minimum bounds

boundsmax

array of maximum bounds

Each component $ix = 1, n$ of the vertex $\#k = 2, nbve$ is computed from the formula :

```
x ( k , ix ) = boundsmin( ix ) + rand() * ( boundsmax( ix ) - boundsmin( i
```

nbpoints
total number of points in the simplex

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ("spendley" , x0)

Creates a new simplex so that it is computed from Spendley's et al. method, i.e. a regular simplex made of $nbve = n+1$ vertices.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "spendley" , x0 , fun )
this = optimsimplex_new ( "spendley" , x0 , fun , len )
[ this , data ] = optimsimplex_new ( "spendley" , x0 , fun , len , data )
```

this
The current simplex object.

x0
the initial point, as a row vector.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

len
optional, the length of the simplex. The default length is 1.0.

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`this = optimsimplex_new ( "oriented" , simplex0 , fun )`

Returns a new oriented simplex, in sorted order. The new simplex has the same sigma- length of the base simplex, but is "oriented" depending on the function value. The created simplex may be used, as Kelley suggests, for a restart of Nelder-Mead algorithm.

The following is a complete list of available calling sequences.

```
[ this , data ] = optimsimplex_new ( "oriented" , simplex0 , fun, data )
```

`this`

The current simplex object.

`simplex0`

the base simplex

`fun`

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

`data`

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`this = optimsimplex_destroy (this)`

Destroy the given object.

`this`

The current simplex object.

`this = optimsimplex_setall ( this , simplex )`

Set all the coordinates and the function values of all the vertices.

`this`

The current simplex object.

`simplex`

the simplex to set.

The given matrix is expected to be a `nbve x n+1` matrix where `n` is the dimension of the space, `nbve` is the number of vertices and with the following content (where the data is organized by row with function value first, and `x` coordinates)

- `simplex(k,1)` is the function value of the vertex `#k`, with `k = 1 , nbve`

- `simplex(k,2:n+1)` is the coordinates of the vertex #k, with $k = 1, \text{nbve}$

`this = optimsimplex_setallfv ( this , fv )`

Set all the function values of all the vertices. The vertex #k is expected to be stored in `fv(k)` with $k = 1, \text{nbve}$. The `fv` input argument is expected to be a row vector.

`this`

The current simplex object.

`fv`

the array of function values

`this = optimsimplex_setallx ( this , x )`

Set all the coordinates of all the vertices. The vertex #k is expected to be stored in `x(k,1:n)` with $k = 1, \text{nbve}$

`this`

The current simplex object.

`x`

the coordinates of the vertices.

`this = optimsimplex_setfv ( this , ive , fv )`

Set the function value at given index and // returns an updated simplex.

`this`

The current simplex object.

`ive`

vertex index

`fv`

the function value

`this = optimsimplex_setn ( this , n )`

Set the dimension of the space of the simplex.

`this`

The current simplex object.

`n`

the dimension

`this = optimsimplex_setnbve ( this , nbve )`

Set the number of vertices of the simplex.

`this`

The current simplex object.

`nbve`

the number of vertices

`this = optimsimplex_setve ( this , ive , fv , x )`

Sets the coordinates of the vertex and the function value at given index in the current simplex.

`this`

The current simplex object.

`ive`

the vertex index

`fv`

the function value

x
the coordinates of the point, as a row vector.

this = optimsimplex_setx (this , ive , x)
Set the coordinates of the vertex at given index, as a row vector, into the current simplex.

this
The current simplex object.

ive
the vertex index

x
the coordinates of the point, as a row vector

simplex = optimsimplex_getall (this)
Returns all the coordinates of all the vertices and the function values in the same matrix.

this
The current simplex object.

simplex
the simplex data.

The simplex matrix has size nbve x n+1, and is organized by row by row as follows :

- simplex(k,1) is the function value of the vertex #k, with k = 1 , nbve
- simplex(k,2:n+1) is the coordinates of the vertex #k, with k = 1 , nbve

fv = optimsimplex_getallfv (this)
Returns all the function values of all the vertices, as a row vector.

this
The current simplex object.

fv
The array of function values. The function value of vertex #k is stored in fv(k) with k = 1 , nbve.

x = optimsimplex_getallx (this)
Returns all the coordinates of all the vertices.

this
The current simplex object.

x
the coordinates.

The vertex #k is stored in x(k,1:n) with k = 1 , nbve.

fv = optimsimplex_getfv (this , ive)
Returns the function value at given index

this
The current simplex object.

ive
the vertex index

n = optimsimplex_getn (this)
Returns the dimension of the space of the simplex

this
The current simplex object.

nbve = optimsimplex_getnbve (this)
Returns the number of vertices in the simplex.

this
The current simplex object.

vertex = optimsimplex_getve (this , ive)
Returns the vertex at given index as a tlist, with fields n, x and fv

this
The current simplex object.

ive
the vertex index

x = optimsimplex_getx (this , ive)
Returns the coordinates of the vertex at given index, as a row vector.

this
The current simplex object.

ive
the vertex index

sicenter = optimsimplex_center (this)
Returns the center of the given simplex

this
The current simplex object.

optimsimplex_check (this)
Check the consistency of the internal data. Generates an error if necessary.

this
The current simplex object.

[this , data] = optimsimplex_computefv (this , fun , data)
Set the values of the function at vertices points.

this
The current simplex object.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`df = optimsimplex_deltafv ( this )`

Returns the vector of difference of function values with respect to the function value at vertex #1.

`this`

The current simplex object.

`dfm = optimsimplex_deltafvmax ( this )`

Returns the difference of function value between the high and the low vertices. It is expected that the vertex #1 is associated with the smallest function value and that the vertex #nbve is associated with the highest function value. Since vertices are ordered, the high is greater than the low.

`this`

The current simplex object.

`m = optimsimplex_dirmat ( this )`

Returns the $n \times n$ matrix of simplex directions i.e. the matrix of differences of vertices coordinates with respect to the vertex #1.

`this`

The current simplex object.

`m = optimsimplex_fvmean ( this )`

Returns the mean of the function value on the simplex.

`this`

The current simplex object.

`sd = optimsimplex_fvstdev ( this )`

Returns the standard deviation of the function value on the simplex.

`this`

The current simplex object.

`v = optimsimplex_fvvariance ( this )`

Returns the variance of the function value on the simplex.

`this`

The current simplex object.

`g = optimsimplex_gradientfv ( this , fun , method )`

Returns the simplex gradient of the function.

The following is a complete list of available calling sequences.

```
g = optimsimplex_gradientfv ( this , fun , method )
[ g , data ] = optimsimplex_gradientfv ( this , fun , method , data )
```

`this`

The current simplex object.

`fun`

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :


```
function y = myfunction (x)
```

method

optional, the method to use to compute the simplex gradient. Two methods are available : "forward" or "centered". The forward method uses the current simplex to compute the simplex gradient. The centered method creates an intermediate reflected simplex and computes the average.

If not provided, the default method is "forward".

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

optimsimplex_print (this)

Display the current simplex, with coordinates and function values.

this

The current simplex object.

[r , data] = optimsimplex_reflect (this , fun , data)

Returns a new simplex by reflexion of current simplex, by reflection with respect to the first vertex in the simplex. This move is used in the centered simplex gradient.

this

The current simplex object.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing

the function value. This feature may be used, for example, to count the number of times that the function has been called.

`[ this , data ] = optimsimplex_shrink ( this , fun , sigma , data )`

Shrink the simplex with given coefficient sigma and returns an updated simplex. The shrink is performed with respect to the first point in the simplex.

`this`

The current simplex object.

`fun`

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

`sigma`

optional, the shrinkage coefficient. The default value is 0.5.

`data`

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additional parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`ssize = optimsimplex_size ( this , method )`

Returns the size of the simplex.

`this`

The current simplex object.

`method`

optional, the method to use to compute the size.

The available methods are the following :

- "sigmaplus" (this is the default)

The sigmaplus size is the maximum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.

- "sigmaminus"

The sigmaminus size is the minimum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.

- "Nash"

The "Nash" size is the sum of the norm of the norm-1 length of the vector from the given vertex to the first vertex. It requires one loop over the vertices.

- "diameter"

The diameter is the maximum norm-2 length of all the edges of the simplex. It requires 2 nested loops over the vertices.

`this = optimsimplex_sort ( this )`

Sorts the simplex with increasing function value order so that the smallest function value is at vertex #1

`this`

The current simplex object.

`str = optimsimplex_tostring ( this )`

Returns the current simplex as a string.

`this`

The current simplex object.

`cen = optimsimplex_xbar ( this , iexcl )`

Returns the center of n vertices, by excluding the vertex with index iexcl. Returns a row vector.

`this`

The current simplex object.

`iexcl`

the index of the vertex to exclude in center computation. The default value of iexcl is the number of vertices : in that case, if the simplex is sorted in increasing function value order, the worst vertex is excluded.

Example : Creating a simplex with given vertices coordinates

In the following example, one creates a simplex with known vertices coordinates. The function values at the vertices are unset.

```
coords = [
    0.    0.
    1.    0.
    0.    1.
];
s1 = optimsimplex_new ( coords );
computed = optimsimplex_getallx ( s1 );
computed = optimsimplex_getn(s1);
computed = optimsimplex_getnbve (s1);
s1 = optimsimplex_destroy(s1);
```

Example : Creating a simplex with randomized bounds

In the following example, one creates a simplex with in the 2D domain $[-5 \ 5]^2$, with $[-1.2 \ 1.0]$ as the first vertex. One uses the randomized bounds method to generate a simplex with 5 vertices. The function takes an additionnal argument mystuff, which is counts the number of times the function is called. After the creation of the simplex, the value of mystuff.nb is 5, which is the expected result because there is one function call by vertex.

```
function y = rosenbrock (x)
    y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
function [ y , mystuff ] = mycostf ( x , mystuff )
    y = rosenbrock(x);
    mystuff.nb = mystuff.nb + 1
endfunction

mystuff = tlist(["T_MYSTUFF","nb"]);
mystuff.nb = 0;
s1 = optimsimplex_new ();
[ s1 , mystuff ] = optimsimplex_randbounds ( s1 , x0 = [-1.2 1.0], fun = mycostf ,
    boundsmin = [-5.0 -5.0] , boundsmax = [5.0 5.0], nbve=5 , data = mystuff );
mprintf("Function evaluations: %d\n",mystuff.nb)
s1 = optimsimplex_destroy ( s1 );
```

Initial simplex strategies

In this section, we analyse the various initial simplex which are provided in this component.

It is known that direct search methods based on simplex designs are very sensitive to the initial simplex. This is why the current component provides various ways to create such an initial simplex.

The first historical simplex-based algorithm is the one presented in "Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation" by W. Spendley, G. R. Hext and F. R. Himsworth. The "spendley" simplex creates the regular simplex which is presented in the paper.

The "randbounds" simplex is due to M.J. Box in "A New Method of Constrained Optimization and a Comparison With Other Methods".

Pfeffer's method is an heuristic which is presented in "Global Optimization Of Lennard-Jones Atomic Clusters" by Ellen Fan. It is due to L. Pfeffer at Stanford and it is used in fminsearch.

TODO

- implement reflection and expansion as in multidimensional search by Torczon
- turn `optimsimplex_reflect` into a proper constructor, i.e. an option of the `optimsimplex_new` function. Another possibility is to reflect "in place" as in the `optimsimplex_shrink` function (but in this case we must provide a "copy" constructor from current simplex before reflecting it).

Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

Bibliography

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R. The Computer Journal, January, 1965, 308--313

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

"Iterative Methods for Optimization", C.T. Kelley, 1999, Chapter 6., section 6.2

"Compact Numerical Methods For Computers - Linear Algebra and Function Minimization", J.C. Nash, 1990, Chapter 14. Direct Search Methods

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", W. Spendley, G. R. Hext, F. R. Himsforth, Technometrics, Vol. 4, No. 4 (Nov., 1962), pp. 441-461, Section 3.1

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

"Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition", SIAM J. on Optimization, Kelley,, C. T., 1999

" Multi-Directional Search: A Direct Search Algorithm for Parallel Machines", by E. Boyd, Kenneth W. Kennedy, Richard A. Tapia, Virginia Joanne Torczon,, Virginia Joanne Torczon, 1989, Phd Thesis, Rice University

"Grid Restrained Nelder-Mead Algorithm", Árpád B#rmen, Janez Puhon, Tadej Tuma, Computational Optimization and Applications, Volume 34 , Issue 3 (July 2006), Pages: 359 - 375

"A convergent variant of the Nelder-Mead algorithm", C. J. Price, I. D. Coope, D. Byatt, Journal of Optimization Theory and Applications, Volume 113 , Issue 1 (April 2002), Pages: 5 - 19,

"Global Optimization Of Lennard-Jones Atomic Clusters", Ellen Fan, Thesis, February 26, 2002, McMaster University

Name

NDcost — generic external for optim computing gradient using finite differences

```
[f,g,ind]=NDcost(x,ind,fun,varargin)
```

Parameters

x
real vector or matrix

ind
integer parameter (see optim)

fun
Scilab function with calling sequence `F=fun(x,varargin)` `varargin` may be use to pass parameters `p1, ... pn`

f
criterion value at point `x` (see optim)

g
gradient value at point `x` (see optim)

Description

This function can be used as an external for `optim` to minimize problem where gradient is too complicated to be programmed. only the function `fun` which computes the criterion is required.

This function should be used as follow:
`[f,xopt,gopt]=optim(list(NDcost,fun,p1,...pn),x0,...)`

Examples

```
// example #1 (a simple one)
//function to minimize
function f=rosenbrock(x,varargin)
    p=varargin(1)
    f=1+sum( p*(x(2:$)-x(1:$-1)^2)^2 + (1-x(2:$))^2)
endfunction

x0=[1;2;3;4];
[f,xopt,gopt]=optim(list(NDcost,rosenbrock,200),x0)

// example #2: This example (by Rainer von Seggern) shows a quick (*) way to
//             identify the parameters of a linear differential equation with
//             the help of scilab.
//             The model is a simple damped (linear) oscillator:
//
//              $x''(t) + c x'(t) + k x(t) = 0$  ,
//
// and we write it as a system of two differential equations of first
// order with  $y(1) = x$ , and  $y(2) = x'$ :
//
//     dy1/dt = y(2)
//     dy2/dt = -c*y(2) -k*y(1).
//
// We suppose to have m measurements of x (that is y(1)) at different times
```

```

// t_obs(1), ..., t_obs(m) called x_obs(1), ..., x_obs(m) (in this example
// these measurements will be simulated), and we want to identify the paramete
// c and k by minimizing the sum of squared errors between x_obs and y1(t_obs,p
//
// (*) This method is not the most efficient but it is easy to implement.
//
function dy = DEQ(t,y,p)
    // The rhs of our first order differential equation system.
    c = p(1); k = p(2)
    dy = [y(2); -c*y(2) - k*y(1)]
endfunction

function y = uN(p, t, t0, y0)
    // Numerical solution obtained with ode. (In this linear case an exact analyt
    // solution can easily be found, but ode would also work for "any" system.)
    // Note: the ode output must be an approximation of the solution at
    // times given in the vector t=[t(1),...,t($)]
    y = ode(y0, t0, t, list(DEQ, p))
endfunction

function r = cost_func(p, t_obs, x_obs, t0, y0)
    // This is the function to be minimized, that is the sum of the squared
    // errors between what gives the model and the measurements.
    sol = uN(p, t_obs, t0, y0)
    e = sol(1,:) - x_obs
    r = sum(e.*e)
endfunction

// Data
y0 = [10; 0]; t0 = 0; // Initial conditions y0 for initial time t0.
T = 30; // Final time for the measurements.

// Here we simulate experimental data, (from which the parameters
// should be identified).
pe = [0.2; 3]; // Exact parameters
m = 80; t_obs = linspace(t0+2, T, m); // Observation times
// Noise: each measurement is supposed to have a (gaussian) random error
// of mean 0 and std deviation proportional to the magnitude
// of the value (sigma*|x_exact(t_obs(i))|).
sigma = 0.1;
y_exact = uN(pe, t_obs, t0, y0);
x_obs = y_exact(1,:) + grand(1, m, "nor", 0, sigma).*abs(y_exact(1,:));

// Initial guess parameters
p0 = [0.5 ; 5];

// The value of the cost function before optimization:
cost0 = cost_func(p0, t_obs, x_obs, t0, y0);
mprintf("\n\nr The value of the cost function before optimization = %g \n\nr", ...

// Solution with optim
[costopt, popt] = optim(list(NDcost, cost_func, t_obs, x_obs, t0, y0), p0, ...
                        'ar', 40, 40, 1e-3);

mprintf("\n\nr The value of the cost function after optimization = %g", costopt)
mprintf("\n\nr The identified values of the parameters: c = %g, k = %g \n\nr", ...
        popt(1), popt(2))

```

```
// A small plot:
t = linspace(0,T,400);
y = uN(popt, t, t0, y0);
clf();
plot2d(t',y(1,:)',style=5)
plot2d(t_obs',x_obs(1,:)',style=-5)
legend(["model","measurements"]);
xlabel("Least square fit to identify ode parameters")
```

See Also

optim, external, derivative

Name

aplat — Flattens a list.

```
[lf, ind] = aplat(l, r)
```

Parameters

l

a list

r

an optional flat list

lf

a flat list (a single hierachical level)

ind

a list, each entry give the path of the corresponding *lf* entry in the original list

Description

Creates a flat list, built with the initial *l* list leaves and if given prepended by the *r* list entries

Examples

```
[lf, ind]=aplat(list(1,2, list([3,1], 'xxx', list([3,2,1]))))
```

See Also

recons

Authors

F.D. and S.S., INRIA

Name

datafit — Parameter identification based on measured data

```
[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0],[mem]),  
[work],[stop],[in'])
```

Parameters

imp

scalar argument used to set the trace mode. `imp=0` nothing (except errors) is reported, `imp=1` initial and final reports, `imp=2` adds a report per iteration, `imp>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

G

function descriptor ($e=G(p,z)$, e : $n_e \times 1$, p : $n_p \times 1$, z : $n_z \times 1$)

DG

partial of G wrt p function descriptor (optional; $S=DG(p,z)$, S : $n_e \times n_p$)

Z

matrix $[z_1, z_2, \dots, z_n]$ where z_i ($n_z \times 1$) is the i th measurement

W

weighting matrix of size $n_e \times n_e$ (optional; default no ponderation)

contr

'b', binf, bsup with binf and bsup real vectors with same dimension as p0. binf and bsup are lower and upper bounds on p.

p0

initial guess (size $n_p \times 1$)

algo

'qn' or 'gc' or 'nd'. This string stands for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that 'nd' does not accept bounds on x).

df0

real scalar. Guessed decreasing of f at first iteration. (`df0=1` is the default value).

mem :

integer, number of variables used to approximate the Hessian, (`algo='gc'` or '`nd`'). Default value is around 6.

stop

sequence of optional parameters controlling the convergence of the algorithm. `stop='ar', nap, [iter [, epsg [, epsf [, epsx]]]]`

"ar"

reserved keyword for stopping rule selection defined as follows:

nap

maximum number of calls to fun allowed.

iter

maximum number of iterations allowed.

epsg

threshold on gradient norm.

epsf
threshold controlling decreasing of f

epsx
threshold controlling variation of x . This vector (possibly matrix) of same size as x_0 can be used to scale x .

"in"
reserved keyword for initialization of parameters used when `fun` is given as a Fortran routine (see below).

p
Column vector, optimal solution found

err
scalar, least square error.

Description

`datafit` is used for fitting data to a model. For a given function $G(p, z)$, this function finds the best vector of parameters p for approximating $G(p, z_i) = 0$ for a set of measurement vectors z_i . Vector p is found by minimizing $G(p, z_1)'WG(p, z_1) + G(p, z_2)'WG(p, z_2) + \dots + G(p, z_n)'WG(p, z_n)$

`datafit` is an improved version of `fit_dat`.

Examples

```
//generate the data
function y=FF(x,p),y=p(1)*(x-p(2))+p(3)*x.*x,endfunction
X=[];Y=[];
pg=[34;12;14] //parameter used to generate data
for x=0:.1:3, Y=[Y,FF(x,pg)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];

//The criterion function
function e=G(p,z),
    y=z(1),x=z(2);
    e=y-FF(x,p),
endfunction

//Solve the problem
p0=[3;5;10]
[p,err]=datafit(G,Z,p0);

scf(0);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1) // the noisy data
plot2d(X,FF(X,p),12) //the solution

//the gradient of the criterion function
function s=DG(p,z),
    a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),
    s=-[x-b,-a,x*x]
endfunction

[p,err]=datafit(G,DG,Z,p0);
```

```
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1) // the noisy data
plot2d(X,FF(X,p),12) //the solution

// Add some bounds on the estimate of the parameters
// We want positive estimation (the result will not change)
[p,err]=datafit(G,DG,Z,'b',[0;0;0],[%inf;%inf;%inf],p0,algo='gc');
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1) // the noisy data
plot2d(X,FF(X,p),12) //the solution
```

See Also

lsqrsolve, optim, leastsq

Name

derivative — approximate derivatives of a function

```
derivative(F,x)
[J [,H]] = derivative(F,x [,h ,order ,H_form ,Q])
```

Parameters

F
a Scilab function $F: \mathbb{R}^n \rightarrow \mathbb{R}^m$ or a `list(F, p1, ..., pk)`, where F is a scilab function in the form $y=F(x, p1, ..., pk)$, p1, ..., pk being any scilab objects (matrices, lists,...).

x
real column vector of dimension n.

h
(optional) real, the stepsize used in the finite difference approximations.

order
(optional) integer, the order of the finite difference formula used to approximate the derivatives (order = 1,2 or 4, default is order=2).

H_form
(optional) string, the form in which the Hessian will be returned. Possible forms are:

H_form='default'
H is a $m \times (n^2)$ matrix ; in this form, the k-th row of H corresponds to the Hessian of the k-th component of F, given as the following row vector :

$$\left(\frac{d(\text{grad}(F_k))}{dx_1}, \dots, \frac{d(\text{grad}(F_k))}{dx_n} \right)$$

((grad(F_k) being a row vector).

H_form='blockmat' :
H is a $(mxn) \times n$ block matrix : the classic Hessian matrices (of each component of F) are stacked by row ($H = [H1 ; H2 ; \dots ; Hm]$ in scilab syntax).

H_form='hypermat' :
H is a $n \times n$ matrix for $m=1$, and a $n \times n \times m$ hypermatrix otherwise. $H(:, :, k)$ is the classic Hessian matrix of the k-th component of F.

Q
(optional) real matrix, orthogonal (default is $\text{eye}(n,n)$). Q is added to have the possibility to remove the arbitrariness of using the canonical basis to approximate the derivatives of a function and it should be an orthogonal matrix. It is not mandatory but better to recover the derivative as you need the inverse matrix (and so simply Q' instead of $\text{inv}(Q)$).

J
approximated Jacobian

H
approximated Hessian

Description

Numerical approximation of the first and second derivatives of a function $F: \mathbb{R}^n \rightarrow \mathbb{R}^m$ at the point x. The Jacobian is computed by approximating the directional derivatives of the components of

F in the direction of the columns of Q. (For m=1, v=Q(:,k) : grad(F(x))*v = Dv(F(x)).) The second derivatives are computed by composition of first order derivatives. If H is given in its default form the Taylor series of F(x) up to terms of second order is given by :

$$F(x+dx) = F(x) + J(x) \cdot dx + \frac{1}{2} \cdot dx^T \cdot H(x) \cdot dx + \dots$$

(([J,H]=derivative(F,x,H_form='default'), J=J(x), H=H(x).))

Performances

If the problem is correctly scaled, increasing the accuracy reduces the total error but requires more function evaluations. The following list presents the number of function evaluations required to compute the Jacobian depending on the order of the formula and the dimension of x, denoted by n:

- order=1, the number of function evaluations is n+1,
- order=2, the number of function evaluations is 2n,
- order=4, the number of function evaluations is 4n.

Computing the Hessian matrix requires square the number of function evaluations, as detailed in the following list.

- order=1, the number of function evaluations is (n+1)^2,
- order=2, the number of function evaluations is 4n^2,
- order=4, the number of function evaluations is 16n^2.

Remarks

The step size h must be small to get a low error but if it is too small floating point errors will dominate by cancellation. As a rule of thumb, do not change the default step size. To work around numerical difficulties one may also change the order and/or choose different orthogonal matrices Q (the default is eye(n,n)), especially if the approximate derivatives are used in optimization routines. All the optional arguments may also be passed as named arguments, so that one can use calls in the form :

```
derivative(F, x, H_form = "hypermat")
derivative(F, x, order = 4) etc.
```

Examples

```
function y=F(x)
    y=[sin(x(1)*x(2))+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

function y=G(x,p)
    y=[sin(x(1)*x(2)*p)+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

x=[1;2;3];
[J,H]=derivative(F,x,H_form='blockmat')

n=3;
```

```

// form an orthogonal matrix :
Q = qr(rand(n,n))
// Test order 1, 2 and 4 formulas.
for i=[1,2,4]
    [J,H]=derivative(F,x,order=i,H_form='blockmat',Q=Q);
    mprintf("order= %d \n",i);
    H,
end

p=1;
h=1e-3;
[J,H]=derivative(list(G,p),x,h,2,H_form='hypermat');
H
[J,H]=derivative(list(G,p),x,h,4,Q=Q);
H

// Taylor series example:
dx=1e-3*[1;1;-1];
[J,H]=derivative(F,x);
F(x+dx)
F(x+dx)-F(x)
F(x+dx)-F(x)-J*dx
F(x+dx)-F(x)-J*dx-1/2*H*(dx .* dx)

// A trivial example
function y=f(x,A,p,w)
    y=x'*A*x+p'*x+w;
endfunction
// with Jacobian and Hessian given by J(x)=x'*(A+A')+p', and H(x)=A+A'.
A = rand(3,3);
p = rand(3,1);
w = 1;
x = rand(3,1);
[J,H]=derivative(list(f,A,p,w),x,h=1,H_form='blockmat')

// Since f(x) is quadratic in x, approximate derivatives of order=2 or 4 by fin
// differences should be exact for all h~0. The apparent errors are caused by
// cancellation in the floating point operations, so a "big" h is choosen.
// Comparison with the exact matrices:
Je = x'*(A+A')+p'
He = A+A'
clean(Je - J)
clean(He - H)

```

Accuracy issues

The `derivative` function uses the same step h whatever the direction and whatever the norm of x . This may lead to a poor scaling with respect to x . An accurate scaling of the step is not possible without many evaluations of the function. Still, the user has the possibility to compare the results produced by the `derivative` and the `numdiff` functions. Indeed, the `numdiff` function scales the step depending on the absolute value of x . This scaling may produce more accurate results, especially if the magnitude of x is large.

In the following Scilab script, we compute the derivative of an univariate quadratic function. The exact derivative can be computed analytically and the relative error is computed. In this rather extreme case, the `derivative` function produces no significant digits, while the `numdiff` function produces 6 significant digits.

```
// Difference between derivative and numdiff when x is large
function y = myfunction (x)
    y = x*x;
endfunction
x = 1.e100;
fe = 2.0 * x;
fp = derivative(myfunction,x);
e = abs(fp-fe)/fe;
mprintf("Relative error with derivative: %e\n",e)
fp = numdiff(myfunction,x);
e = abs(fp-fe)/fe;
mprintf("Relative error with numdiff: %e\n",e)
```

The previous script produces the following output.

```
Relative error with derivative: 1.000000e+000
Relative error with numdiff: 7.140672e-006
```

In a practical situation, we may not know what is the correct numerical derivative. Still, we are warned that the numerical derivatives should be used with caution in this specific case.

See Also

[numdiff](#), [derivat](#)

Authors

Rainer von Seggern, Bruno Pincon

Name

fit_dat — Parameter identification based on measured data

```
[p,err]=fit_dat(G,p0,Z [,W] [,pmin,pmax] [,DG])
```

Parameters

G
Scilab function (e=G(p,z), e: nex1, p: npx1, z: nzx1)

p0
initial guess (size npx1)

Z
matrix [z_1,z_2,...z_n] where z_i (npx1) is the ith measurement

W
weighting matrix of size nexne (optional; default 1)

pmin
lower bound on p (optional; size npx1)

pmax
upper bound on p (optional; size npx1)

DG
partial of G wrt p (optional; S=DG(p,z), S: nexnp)

Description

fit_dat is used for fitting data to a model. For a given function G(p,z), this function finds the best vector of parameters p for approximating $G(p,z_i)=0$ for a set of measurement vectors z_i. Vector p is found by minimizing $G(p,z_1)'WG(p,z_1)+G(p,z_2)'WG(p,z_2)+\dots+G(p,z_n)'WG(p,z_n)$

Examples

```
deff('y=FF(x)', 'y=a*(x-b)+c*x.*x')
X=[];Y=[];
a=34;b=12;c=14;for x=0:.1:3, Y=[Y,FF(x)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];
deff('e=G(p,z)', 'a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),e=y-FF(x)')

[p,err]=fit_dat(G,[3;5;10],Z)

xset('window',0)
clf();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')

a=34;b=12;c=14;
deff('s=DG(p,z)', 'y=z(1),x=z(2),s=-[x-p(2),-p(1),x*x]')

[p,err]=fit_dat(G,[3;5;10],Z,DG)
```

```
xset('window',1)
clf();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')
```

See Also

optim, datafit

Name

`fsolve` — find a zero of a system of n nonlinear functions

```
[x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
```

Parameters

- `x0`
real vector (initial value of function argument).
- `fct`
external (i.e function or list or string).
- `fjac`
external (i.e function or list or string).
- `tol`
real scalar. precision tolerance: termination occurs when the algorithm estimates that the relative error between x and the solution is at most `tol`. (`tol=1.d-10` is the default value).
- `x :`
real vector (final value of function argument, estimated zero).
- `v :`
real vector (value of function at x).
- `info`
termination indicator
- | | |
|---|--|
| 0 | improper input parameters. |
| 1 | algorithm estimates that the relative error between x and the solution is at most <code>tol</code> . |
| 2 | number of calls to <code>fcn</code> reached |
| 3 | <code>tol</code> is too small. No further improvement in the approximate solution x is possible. |
| 4 | iteration is not making good progress. |

Description

find a zero of a system of n nonlinear functions in n variables by a modification of the powell hybrid method. Jacobian may be provided.

```
0 = fct(x) w.r.t x.
```

`fct` is an "external". This external returns $v=fct(x)$ given x .

The simplest calling sequence for `fct` is:

```
[v]=fct(x).
```

If `fct` is a character string, it refers to a C or Fortran routine which must be linked to Scilab. Fortran calling sequence must be

```
fct(n,x,v,iflag)
integer n,iflag
double precision x(n),v(n)
```

and C Calling sequence must be

```
fct(int *n, double x[],double v[],int *iflag)
```

Incremental link is possible ([help link](#)).

`jac` is an "external". This external returns $v = d(fct)/dx(x)$ given x .

The simplest calling sequence for `jac` is:

```
[v]=jac(x).
```

If `jac` is a character string, it refers to a C or Fortran routine which must be linked to Scilab. calling sequences are the same as those for `fct`. Note however that `v` must be a nxn array.

Examples

```
// A simple example with fsolve
a=[1,7;2,8];b=[10;11];

deff(' [y]=fsol1(x)', 'y=a*x+b');
deff(' [y]=fsolj1(x)', 'y=a');

[xres]=fsolve([100;100],fsol1);
a*xres+b

[xres]=fsolve([100;100],fsol1,fsolj1);
a*xres+b

// See SCI/modules/optimization/sci_gateway/fortran/Ex-fsolve.f
[xres]=fsolve([100;100], 'fsol1', 'fsolj1', 1.e-7);
a*xres+b
```

For some starting points and some equations system, the `fsolve` method can fail. The `fsolve` method is a local search method. So, to have a good chance to find a solution to your equations system, you must ship, a good starting point to `fsolve`.

Here is an example on which `fsolve` can fail:

```
// Another example with fsolve
function F=feuler(x,r)
```

```
F=x-r-dt*(x^2-x^3);
endfunction
function J=dFdx(x) //definition de la derivee de F
    J=1-dt*(2*x-3*x^2);
endfunction

r = 0.04257794928862307 ;
dt = 10;

[x,v,info]=fsolve(r,list(feuler,r),dFdx); // fsolve don't find the solution
disp(v); // The residual
disp(info); // The termination indicator

[x,v,info]=fsolve(1,list(feuler,r),dFdx); // fsolve find the solution
disp(v); // The residual
disp(info); // The termination indicator

clf();x=linspace(0,1,1000);plot(x,feuler(x))
a=gca();a.grid=[5 5];
```

So, each time you use fsolve, be sure to check the termination indicator and the residual value to see if fsolve has converged.

See Also

external, qpsolve, optim

Name

karmarkar — karmarkar algorithm

```
[x1]=karmarkar(a,b,c,x0)
```

Parameters

a
matrix (n,p)

b
n - vector

c
p - vector

x0
initial vector

eps
threshold (default value : 1.d-5)

gamma
descent step $0 < \text{gamma} < 1$, default value : 1/4

x1
solution

crit
value of $c' \cdot x1$

Description

Computes x which minimizes

$$\begin{aligned} &\min c^t \cdot x \\ &\text{with } a \cdot x = b \\ &\text{and } x \geq 0 \end{aligned}$$

Examples

```
n=10;p=20;  
a=rand(n,p);  
c=rand(p,1);  
x0=abs(rand(p,1));  
b=a*x0;  
x1=karmarkar(a,b,c,x0);
```

Name

leastsq — Solves non-linear least squares problems

```
[fopt,[xopt],[grdopt]]=leastsq(fun, x0)
[fopt,[xopt],[grdopt]]=leastsq(fun, dfun, x0)
[fopt,[xopt],[grdopt]]=leastsq(fun, cstr, x0)
[fopt,[xopt],[grdopt]]=leastsq(fun, dfun, cstr, x0)
[fopt,[xopt],[grdopt]]=leastsq(fun, dfun, cstr, x0, algo)
[fopt,[xopt],[grdopt]]=leastsq([imp], fun [,dfun] [,cstr],x0 [,algo],[df0],[mem])
```

Parameters

fopt

value of the function $f(x)=||fun(x)||^2$ at xopt

xopt

best value of x found to minimize $||fun(x)||^2$

grdopt

gradient of f at xopt

fun

a scilab function or a list defining a function from R^n to R^m (see more details in DESCRIPTION).

x0

real vector (initial guess of the variable to be minimized).

dfun

a scilab function or a string defining the Jacobian matrix of fun (see more details in DESCRIPTION).

cstr

bound constraints on x. They must be introduced by the string keyword 'b' followed by the lower bound binf then by the upper bound bsup (so cstr appears as 'b', binf, bsup in the calling sequence). Those bounds are real vectors with same dimension than x0 (-%inf and +%inf may be used for dimension which are unrestricted).

algo

a string with possible values: 'qn' or 'gc' or 'nd'. These strings stand for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that 'nd' does not accept bounds on x.

imp

scalar argument used to set the trace mode. imp=0 nothing (except errors) is reported, imp=1 initial and final reports, imp=2 adds a report per iteration, imp>2 add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

df0

real scalar. Guessed decreasing of $||fun||^2$ at first iteration. (df0=1 is the default value).

mem

integer, number of variables used to approximate the Hessian (second derivatives) of f when algo='qn'. Default value is around 6.

stop

sequence of optional parameters controlling the convergence of the algorithm. They are introduced by the keyword 'ar', the sequence being of the form 'ar', nap, [iter [, epsg [, epsf [, epsx]]]]

nap
 maximum number of calls to `fun` allowed.

iter
 maximum number of iterations allowed.

epsg
 threshold on gradient norm.

epsf
 threshold controlling decreasing of `f`

epsx
 threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

Description

`fun` being a function from R^n to R^m this routine tries to minimize w.r.t. `x`, the function:

$$f(x) = \| \text{fun}(x) \|^2 = \sum_{i=1}^m \text{fun}_i^2(x)$$

which is the sum of the squares of the components of `fun`. Bound constraints may be imposed on `x`.

How to provide fun and dfun

`fun` can be either a usual scilab function (case 1) or a fortran or a C routine linked to scilab (case 2). For most problems the definition of `fun` will need supplementary parameters and this can be done in both cases.

case 1:

when `fun` is a Scilab function, its calling sequence must be: `y=fun(x [, opt_par1, opt_par2, ... ])`. When `fun` needs optional parameters it must appear as `list(fun, opt_par1, opt_par2, ...)` in the calling sequence of `leastsq`.

case 2:

when `fun` is defined by a Fortran or C routine it must appear as `list(fun_name, m [, opt_par1, opt_par2, ... ])` in the calling sequence of `leastsq`, `fun_name` (a string) being the name of the routine which must be linked to Scilab (see link). The generic calling sequences for this routine are:

```
In Fortran:      subroutine fun(m, n, x, params, y)
                  integer m,n
                  double precision x(n), params(*), y(m)
```

```
In C:            void fun(int *m, int *n, double *x, double *params, double *y
```

where `n` is the dimension of vector `x`, `m` the dimension of vector `y` (which must store the evaluation of `fun` at `x`) and `params` is a vector which contains the optional parameters `opt_par1`, `opt_par2`, ... (each parameter may be a vector, for instance if `opt_par1` has 3 components, the description of `opt_par2` begin from `params(4)` (fortran case), and from `params[3]` (C case), etc... Note that even if `fun` doesn't need supplementary parameters you must anyway write the fortran code with a `params` argument (which is then unused in the subroutine core).

In many cases it is advised to provide the Jacobian matrix `dfun` ($dfun(i,j)=df_i/dx_j$) to the optimizer (which uses a finite difference approximation otherwise) and as for `fun` it may be given as a usual scilab function or as a fortran or a C routine linked to scilab.

case 1:

when `dfun` is a scilab function, its calling sequence must be: `y=dfun(x [, optional parameters])` (notes that even if `dfun` needs optional parameters it must appear simply as `dfun` in the calling sequence of `leastsq`).

case 2:

when `dfun` is defined by a Fortran or C routine it must appear as `dfun_name` (a string) in the calling sequence of `leastsq` (`dfun_name` being the name of the routine which must be linked to Scilab). The calling sequences for this routine are nearly the same than for `fun`:

```
In Fortran:      subroutine dfun(m, n, x, params, y)
                  integer m,n
                  double precision x(n), params(*), y(m,n)
```

```
In C:            void fun(int *m, int *n, double *x, double *params, double *y
```

in the C case $dfun(i,j)=dfi/dxj$ must be stored in `y[m*(j-1)+i-1]`.

Remarks

Like `datafit`, `leastsq` is a front end onto the `optim` function. If you want to try the Levenberg-Marquard method instead, use `lsqrsolve`.

A least squares problem may be solved directly with the `optim` function ; in this case the function `NDcost` may be useful to compute the derivatives (see the `NDcost` help page which provides a simple example for parameters identification of a differential equation).

Examples

```
// We will show different calling possibilities of leastsq on one (trivial) exam
// which is non linear but doesn't really need to be solved with leastsq (apply
// log linearizes the model and the problem may be solved with linear algebra).
// In this example we look for the 2 parameters x(1) and x(2) of a simple
// exponential decay model (x(1) being the unknow initial value and x(2) the
// decay constant):

function y = yth(t, x)
    y = x(1)*exp(-x(2)*t)
endfunction

// we have the m measures (ti, yi):
m = 10;
tm = [0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5]';
ym = [0.79, 0.59, 0.47, 0.36, 0.29, 0.23, 0.17, 0.15, 0.12, 0.08]';
wm = ones(m,1); // measure weights (here all equal to 1...)

// and we want to find the parameters x such that the model fits the given
// datas in the least square sense:
//
// minimize f(x) = sum_i wm(i)^2 ( yth(tm(i),x) - ym(i) )^2

// initial parameters guess
x0 = [1.5 ; 0.8];

// in the first examples, we define the function fun and dfun
// in scilab language
```

```

function e = myfun(x, tm, ym, wm)
    e = wm.*( yth(tm, x) - ym )
endfunction

function g = mydfun(x, tm, ym, wm)
    v = wm.*exp(-x(2)*tm)
    g = [v , -x(1)*tm.*v]
endfunction

// now we could call leastsq:

// 1- the simplest call
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),x0)

// 2- we provide the Jacobian
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,x0)

// a small graphic (before showing other calling features)
tt = linspace(0,1.1*max(tm),100)';
yy = yth(tt, xopt);
clf()
plot2d(tm, ym, style=-2)
plot2d(tt, yy, style = 2)
legend(["measure points", "fitted curve"]);
xlabel("a simple fit with leastsq")

// 3- how to get some information (we use imp=1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0)

// 4- using the conjugate gradient (instead of quasi Newton)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"gc")

// 5- how to provide bound constraints (not useful here !)
xinf = [-%inf, -%inf]; xsup = [%inf, %inf];
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),"b",xinf,xsup,x0) // without Jac
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,"b",xinf,xsup,x0) // with

// 6- playing with some stopping parameters of the algorithm
// (allows only 40 function calls, 8 iterations and set epsf=0.01, epsf=0.1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"ar",40,8,0.01,0.1)

// 7 and 8: now we want to define fun and dfun in fortran then in C
// Note that the "compile and link to scilab" method used here
// is believed to be OS independant (but there are some requirements,
// in particular you need a C and a fortran compiler, and they must
// be compatible with the ones used to build your scilab binary).

// 7- fun and dfun in fortran

// 7-1/ Let 's Scilab write the fortran code (in the TMPDIR directory):
f_code = [
    "      subroutine myfun(m,n,x,param,f)"
    "      param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
    "      implicit none"
    "      integer n,m"
    "      double precision x(n), param(*), f(m)"
    "      integer i"
    "      do i = 1,m"

```

```

"      f(i) = param(2*m+i)*( x(1)*exp(-x(2)*param(i)) - param(m+i)
"      enddo"
"      end ! subroutine fun"
""
"      subroutine mydfun(m,n,x,param,df)"
"*      param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
"      implicit none"
"      integer n,m"
"      double precision x(n), param(*), df(m,n)"
"      integer i"
"      do i = 1,m"
"          df(i,1) = param(2*m+i)*exp(-x(2)*param(i))"
"          df(i,2) = -x(1)*param(i)*df(i,1)"
"      enddo"
"      end ! subroutine dfun"];

cd TMPDIR;
mputl(f_code,TMPDIR+'/myfun.f')

// 7-2/ compiles it. You need a fortran compiler !
names = ["myfun" "mydfun"]
flibname = ilib_for_link(names,"myfun.o",[],"f");

// 7-3/ link it to scilab (see link help page)
link(flibname,names,"f")

// 7-4/ ready for the leastsq call: be carreful don't forget to
//      give the dimension m after the routine name !
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),x0) // without Jacobian
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),"mydfun",x0) // with Jacobian

// 8- last example: fun and dfun in C

// 8-1/ Let 's Scilab write the C code (in the TMPDIR directory):
c_code = ["#include <math.h>"
"void myfunc(int *m,int *n, double *x, double *param, double *f)"
"{"
"    /* param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
"    int i;"
"    for ( i = 0 ; i < *m ; i++ )"
"        f[i] = param[2*(*m)+i]*( x[0]*exp(-x[1]*param[i]) - param[(*m)+i]"
"    return;"
"}"
""
"void mydfunc(int *m,int *n, double *x, double *param, double *df)"
"{"
"    /* param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
"    int i;"
"    for ( i = 0 ; i < *m ; i++ )"
"    {"
"        df[i] = param[2*(*m)+i]*exp(-x[1]*param[i]);"
"        df[i+(*m)] = -x[0]*param[i]*df[i];"
"    }"
"    return;"
"}"];

mputl(c_code,TMPDIR+'/myfunc.c')

// 8-2/ compiles it. You need a C compiler !

```

```
names = ["myfunc" "mydfunc"]
clibname = ilib_for_link(names, "myfunc.o", [], "c");

// 8-3/ link it to scilab (see link help page)
link(clibname, names, "c")

// 8-4/ ready for the leastsq call
[f, xopt, gropt] = leastsq(list("myfunc", m, tm, ym, wm), "mydfunc", x0)
```

See Also

lsqrsolve, optim, NDcost, datafit, external, qpsolve

Name

list2vec — Catenates list entries in a matrix.

```
[bigVector,varsizes] = list2vec(li)
```

Parameters

li

a list with n entries. The list entries must be 2D matrices with comptible types.

bigVector

A column vector. Formed by the elements of the corresponding list entry.

varsizes

An n by 3 matrix. Each row contains the dimensions of the corresponding list entry.

Description

Catenates list entries in a column vector. The list entries are supposed to be matrices with compatible types with respect to catenation.

This function is a subsidiary for lmisolver

Examples

```
list2vec(list(1,2,3))  
list2vec(list([1 2 3],[4;5],[%s %s+1]))
```

See Also

vec2list

Authors

F.D. INRIA

Name

lmsolver — linear matrix inequation solver

```
[XLISTF[,OPT]] = lmsolver(XLIST0,evalfunc [,options])
```

Parameters

XLIST0

a list of containing initial guess (e.g. $XLIST0=list(X1,X2,\dots,Xn)$)

evalfunc

a Scilab function ("external" function with specific syntax)

The syntax the function `evalfunc` must be as follows:

$[LME, LMI, OBJ]=evalfunc(X)$ where X is a list of matrices, LME , LMI are lists and OBJ a real scalar.

XLISTF

a list of matrices (e.g. $XLIST0=list(X1,X2,\dots,Xn)$)

options

optional parameter. If given, `options` is a real row vector with 5 components $[Mbound, abstol, nu, maxiters, reltol]$

Description

`lmsolver` solves the following problem:

minimize $f(X1,X2,\dots,Xn)$ a linear function of Xi 's

under the linear constraints: $G_i(X1,X2,\dots,Xn)=0$ for $i=1,\dots,p$ and LMI (linear matrix inequalities) constraints:

$H_j(X1,X2,\dots,Xn) > 0$ for $j=1,\dots,q$

The functions f , G , H are coded in the Scilab function `evalfunc` and the set of matrices Xi 's in the list X (i.e. $X=list(X1,\dots,Xn)$).

The function `evalfun` must return in the list LME the matrices $G1(X),\dots,Gp(X)$ (i.e. $LME(i)=Gi(X1,\dots,Xn)$, $i=1,\dots,p$). `evalfun` must return in the list LMI the matrices $H1(X),\dots,Hq(X)$ (i.e. $LMI(j)=Hj(X1,\dots,Xn)$, $j=1,\dots,q$). `evalfun` must return in OBJ the value of $f(X)$ (i.e. $OBJ=f(X1,\dots,Xn)$).

`lmsolver` returns in $XLISTF$, a list of real matrices, i. e. $XLIST=list(X1,X2,\dots,Xn)$ where the Xi 's solve the LMI problem:

Defining Y , Z and $cost$ by:

$[Y,Z, cost]=evalfunc(XLIST)$, Y is a list of zero matrices, $Y=list(Y1,\dots,Yp)$, $Y1=0$, $Y2=0$, $\dots$, $Yp=0$.

Z is a list of square symmetric matrices, $Z=list(Z1,\dots,Zq)$, which are semi positive definite $Z1>0$, $Z2>0$, $\dots$, $Zq>0$ (i.e. $spec(Z(j)) > 0$),

$cost$ is minimized.

`lmsolver` can also solve LMI problems in which the Xi 's are not matrices but lists of matrices. More details are given in the documentation of `LMITOOL`.

Examples

```
//Find diagonal matrix X (i.e. X=diag(diag(X), p=1) such that
//A1'*X+X*A1+Q1 < 0, A2'*X+X*A2+Q2 < 0 (q=2) and trace(X) is maximized
n = 2;
A1 = rand(n,n);
A2 = rand(n,n);
Xs = diag(1:n);
Q1 = -(A1'*Xs+Xs*A1+0.1*eye());
Q2 = -(A2'*Xs+Xs*A2+0.2*eye());

deff(' [LME,LMI,OBJ]=evalf(Xlist)', 'X    = Xlist(1); ...
                                     LME = X-diag(diag(X));...
                                     LMI = list(-(A1'*X+X*A1+Q1), -(A2'*X+X*A2+Q2));
                                     OBJ = -sum(diag(X)) ');

X=lmsolver(list(zeros(A1)),evalf);

X=X(1)
[Y,Z,c]=evalf(X)
```

See Also

lmitool

Name

lmitool — tool for solving linear matrix inequations

```
lmitool()  
lmitool(filename)  
txt=lmitool(probname,varlist,datalist)
```

Parameters

filename

a string referring to a `.sci` function

probname

a string containing the name of the problem

varlist

a string containing the names of the unknown matrices (separated by commas if there are more than one)

datalist

a string containing the names of data matrices (separated by commas if there are more than one)

txt

a string providing information on what the user should do next

Description

`lmitool()` or `lmitool(filename)` is used to define interactively a LMI problem. In the non interactive mode, `txt=lmitool(probname,varlist,datalist)` generates a file in the current directory. The name of this file is obtained by adding `.sci` to the end of `probname`. This file is the skeleton of a solver function and the corresponding evaluation function needed by `lmsolver`.

See Also

`lmsolver`

Name

lsqrsolve — minimize the sum of the squares of nonlinear functions, levenberg-marquardt algorithm

```
[x [,v [,info]]]=lsqrsolve(x0,fct,m [,stop [,diag]])  
[x [,v [,info]]]=lsqrsolve(x0,fct,m ,fjac [,stop [,diag]])
```

Parameters

- x0**
real vector of size **n** (initial estimate of the solution vector).
- fct**
external (i.e function or list or string).
- m**
integer, the number of functions. **m** must be greater than or equal to **n**.
- fjac**
external (i.e function or list or string).
- stop**
optional vector [**ftol**,**xtol**,**gtol**,**maxfev**,**epsfcn**,**factor**] the default value is [1.d-8,1.d-8,1.d-5,1000,0,100]
- ftol**
A positive real number, termination occurs when both the actual and predicted relative reductions in the sum of squares are at most **ftol**. therefore, **ftol** measures the relative error desired in the sum of squares.
- xtol**
A positive real number, termination occurs when the relative error between two consecutive iterates is at most **xtol**. therefore, **xtol** measures the relative error desired in the approximate solution.
- gtol**
A nonnegative input variable. termination occurs when the cosine of the angle between **fct(x)** and any column of the jacobian is at most **gtol** in absolute value. therefore, **gtol** measures the orthogonality desired between the function vector and the columns of the jacobian.
- maxfev**
A positive integer, termination occurs when the number of calls to **fct** is at least **maxfev** by the end of an iteration.
- epsfcn**
A positive real number, used in determining a suitable step length for the forward-difference approximation. this approximation assumes that the relative errors in the functions are of the order of **epsfcn**. if **epsfcn** is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.
- factor**
A positive real number, used in determining the initial step bound. this bound is set to the product of **factor** and the euclidean norm of **diag*****x** if nonzero, or else to **factor** itself. in most cases **factor** should lie in the interval (0.1, 100). 100 is a generally recommended value.
- diag**
is an array of length **n**. **diag** must contain positive entries that serve as multiplicative scale factors for the variables.

x :
real vector (final estimate of the solution vector).

v :
real vector (value of `fct(x)`).

info
termination indicator

0	improper input parameters.
1	algorithm estimates that the relative error between <code>x</code> and the solution is at most <code>tol</code> .
2	number of calls to <code>fcn</code> reached
3	<code>tol</code> is too small. No further improvement in the approximate solution <code>x</code> is possible.
4	iteration is not making good progress.
5	number of calls to <code>fcn</code> has reached or exceeded <code>maxfev</code>
6	<code>ftol</code> is too small. no further reduction in the sum of squares is possible.
7	<code>xtol</code> is too small. no further improvement in the approximate solution <code>x</code> is possible.
8	<code>gtol</code> is too small. <code>fvec</code> is orthogonal to the columns of the jacobian to machine precision.

Description

minimize the sum of the squares of `m` nonlinear functions in `n` variables by a modification of the levenberg-marquardt algorithm. the user must provide a subroutine which calculates the functions. the jacobian is then calculated by a forward-difference approximation.

minimize $\text{sum}(\text{fct}(x, m) .^2)$ where `fct` is function from R^n to R^m

`fct` should be :

- a Scilab function whose calling sequence is `v=fct(x, m)` given `x` and `m`.
- a character string which refers to a C or Fortran routine which must be linked to Scilab.

Fortran calling sequence should be `fct(m, n, x, v, iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `v` a double precision vector of size `m`.

C calling sequence should be `fct(int *m, int *n, double x[], double v[], int *iflag)`

`fjac` is an external which returns $v=d(\text{fct})/dx(x)$. it should be :

a Scilab function

whose calling sequence is `J=fjac(x, m)` given `x` and `m`.

a character string

it refers to a C or Fortran routine which must be linked to Scilab.

Fortran calling sequence should be `fjac(m,n,x,jac,iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `jac` a double precision vector of size `m*n`.

C calling sequence should be `fjac(int *m, int *n, double x[],double v[],int *iflag)`

return -1 in `iflag` to stop the algorithm if the function or jacobian could not be evaluated.

Examples

```
// A simple example with lsqrsolve
a=[1,7;
   2,8
   4 3];
b=[10;11;-1];

function y=f1(x,m)
    y=a*x+b;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3)
xsol+a\b

function y=fj1(x,m)
    y=a;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3,fj1)
xsol+a\b

// Data fitting problem
// 1 build the data
a=34;b=12;c=14;

deff('y=FF(x)','y=a*(x-b)+c*x.*x');
X=(0:.1:3)';Y=FF(X)+100*(rand()-.5);

//solve
function e=f1(abc,m)
    a=abc(1);b=abc(2),c=abc(3),
    e=Y-(a*(X-b)+c*X.*X);
endfunction

[abc,v]=lsqrsolve([10;10;10],f1,size(X,1));
abc
norm(v)
```

See Also

external, qpsolve, optim, fsolve

Used Functions

lmdif, lmdr from minpack, Argonne National Laboratory.

Name

numdiff — numerical gradient estimation

```
g=numdiff(fun,x [,dx])
```

Parameters

fun

an external, Scilab function or list. See below for calling sequence, see also external for details about external functions.

x

vector, the argument of the function fun

dx

vector, the finite difference step. Default value is $dx = \sqrt{\%eps} * (1 + 1d - 3 * \text{abs}(x))$

g

vector, the estimated gradient

Description

given a function $\text{fun}(x)$ from $\mathbb{R}^n$ to $\mathbb{R}^p$ computes the matrix g such as

```
g(i,j) = (df_i)/(dx_j)
```

using finite difference methods.

Without parameters, the function fun calling sequence is $y = \text{fun}(x)$, and numdiff can be called as $g = \text{numdiff}(\text{fun}, x)$. Else the function fun calling sequence must be $y = \text{fun}(x, \text{param}_1, \text{param}_2, \dots, \text{param}_q)$. If parameters $\text{param}_1, \text{param}_2, \dots, \text{param}_q$ exist then numdiff can be called as follow $g = \text{numdiff}(\text{list}(\text{fun}, \text{param}_1, \text{param}_2, \dots, \text{param}_q), x)$.

See the derivative with respect to numerical accuracy issues and comparison between the two algorithms.

Examples

```
// example 1 (without parameters)
// myfun is a function from R^2 to R : (x(1),x(2)) |--> myfun(x)
function f=myfun(x)
    f=x(1)*x(1)+x(1)*x(2)
endfunction
```

```
x=[5 8]
g=numdiff(myfun,x)
```

```
// The exact gradient (i.e derivate belong x(1) :first component and derivate b
exact=[2*x(1)+x(2) x(1)]
```

```
//example 2 (with parameters)
// myfun is a function from R to R: x(1) |--> myfun(x)
// myfun contains 3 parameters, a, b, c
```

```
function f=myfun(x,a,b,c)
    f=(x+a)^c+b
endfunction

a=3; b=4; c=2;
x=1
g2=numdiff(list(myfun,a,b,c),x)

// The exact gradient, i.e derivate belong x(1), is :
exact2=c*(x+a)^(c-1)
```

See Also

optim, derivative, external

Name

optim — non-linear optimization routine

```
[f,xopt]=optim(costf,x0)
[f [,xopt [,gradopt [,work]]]]=optim(costf [,<contr>],x0 [,algo] [,df0 [,mem]])
```

Parameters

costf

external, i.e Scilab function list or string (**costf** is the cost function, that is, a Scilab script, a Fortran 77 routine or a C function).

x0

real vector (initial value of variable to be minimized).

f

value of optimal cost (**f=costf(xopt)**)

xopt

best value of **x** found.

<contr>

keyword representing the following sequence of arguments: 'b', binf, bsup with binf and bsup are real vectors with same dimension as x0. binf and bsup are lower and upper bounds on **x**.

algo

- 'qn' : quasi-Newton (this is the default solver)
- 'gc' : conjugate gradient
- 'nd' : non-differentiable.

Note that the conjugate gradient solver does not accept bounds on **x**.

df0

real scalar. Guessed decreasing of **f** at first iteration. (**df0=1** is the default value).

mem :

integer, number of variables used to approximate the Hessian. Default value is 10. This feature is available for the Gradient-Conjugate algorithm "gc" without constraints and the non-smooth algorithm "nd" without constraints.

<stop>

keyword representing the sequence of optional parameters controlling the convergence of the algorithm. 'ar', nap [,iter [,epsg [,epsf [,epsx]]]]

"ar"

reserved keyword for stopping rule selection defined as follows:

nap

maximum number of calls to **costf** allowed (default is 100).

iter

maximum number of iterations allowed (default is 100).

epsg

threshold on gradient norm.

epsf
threshold controlling decreasing of f

epsx
threshold controlling variation of x . This vector (possibly matrix) of same size as x_0 can be used to scale x .

<params>

keyword representing the method to initialize the arguments t_i , t_d passed to the objective function, provided as a C or Fortran routine. This option has no meaning when the cost function is a Scilab script. <params> can be set to only one of the following values.

- "in"

That mode allows to allocate memory in the internal Scilab workspace so that the objective function can get arrays with the required size, but without directly allocating the memory. "in" stands for "initialization". In that mode, before the value and derivative of the objective function is to be computed, there is a dialog between the optim Scilab primitive and the objective function. In this dialog, the objective function is called two times, with particular values of the "ind" parameter. The first time, ind is set to 10 and the objective function is expected to set the nzs, nrzs and ndzs integer parameters of the "nird" common.

```
common /nird/ nzs,nrzs,ndzs
```

This allows Scilab to allocate memory inside its internal workspace. The second time the objective function is called, ind is set to 11 and the objective function is expected to set the t_i , t_r and t_z arrays. After this initialization phase, each time it is called, the objective function is ensured that the t_i , t_r and t_z arrays which are passed to it have the values that have been previously initialized.

- "ti",valti

In this mode, valti is expected to be a Scilab vector variable containing integers. Whenever the objective function is called, the t_i array it receives contains the values of the Scilab variable.

- "td", valtd

In this mode, valtd is expected to be a Scilab vector variable containing double values. Whenever the objective function is called, the t_d array it receives contains the values of the Scilab variable.

- "ti",valti,"td",valtd

This mode combines the two previous.

The t_i , t_d arrays may be used so that the objective function can be computed. For example, if the objective function is a polynomial, the t_i array may be used to store the coefficients of that polynomial.

Users should choose carefully between the "in" mode and the "ti" and "td" mode, depending on the fact that the arrays are Scilab variables or not. If the data is available as Scilab variables, then the "ti", valti, "td", valtd mode should be chosen. If the data is available directly from the objective function, the "in" mode should be chosen. Notice that there is no "tr" mode, since, in Scilab, all real values are of "double" type.

If neither the "in" mode, nor the "ti", "td" mode is chosen, that is, if <params> is not present as an option of the optim primitive, the user may should not assume that the t_i , t_r and t_d arrays can be used : reading or writing the arrays may generate unpredictable results.

"imp=iflag"

named argument used to set the trace mode. The possible values for iflag are 0,1,2 and >2. Use this option with caution : most of these reports are written on the Scilab standard output.

- iflag=0: nothing (except errors) is reported (this is the default),
- iflag=1: initial and final reports,
- iflag=2: adds a report per iteration,
- iflag>2: add reports on linear search.
- iflag<0: calls the cost function with ind=1 every -imp iterations.

gradopt

gradient of `costf` at `xopt`

work

working array for hot restart for quasi-Newton method. This array is automatically initialized by `optim` when `optim` is invoked. It can be used as input parameter to speed-up the calculations.

Description

Non-linear optimization routine for programs without constraints or with bound constraints:

```
min costf(x) w.r.t x.
```

`costf` is an "external" i.e a Scilab function, a list or a string giving the name of a C or Fortran routine (see "external"). This external must return the value `f` of the cost function at the point `x` and the gradient `g` of the cost function at the point `x`.

- Scilab function case

If `costf` is a Scilab function, the calling sequence for `costf` must be:

```
[f,g,ind]=costf(x,ind)
```

Here, `costf` is a function which returns `f`, value (real number) of cost function at `x`, and `g`, gradient vector of cost function at `x`. The variable `ind` is described below.

- List case

If `costf` is a list, it should be of the form: `list(real_costf, arg1,...,argn)` with `real_costf` a Scilab function with calling sequence : `[f,g,ind]=costf(x,ind,arg1,... argn)`. The `x`, `f`, `g`, `ind` arguments have the same meaning that above. `argi` arguments can be used to pass function parameters.

- String case

If `costf` is a character string, it refers to the name of a C or Fortran routine which must be linked to Scilab

* Fortran case

The interface of the Fortran subroutine computing the objective must be :

```
subroutine costf(ind,n,x,f,g,ti,tr,td)
```


with the following declarations:

```
integer ind,n ti(*)
double precision x(n),f,g(n),td(*)
real tr(*)
```

The argument `ind` is described below.

If `ind = 2, 3` or `4`, the inputs of the routine are : `x, ind, n, ti, tr, td`.

If `ind = 2, 3` or `4`, the outputs of the routine are : `f` and `g`.

* C case

The interface of the C function computing the objective must be :

```
void costf(int *ind, int *n, double *x, double *f, double *g, int *ti, fl
```

The argument `ind` is described below.

The inputs and outputs of the function are the same as in the fortran case.

The `ind` input argument is a message sent from the solver to the cost function so that the cost function knows what to compute.

- If `ind=2`, `costf` must provide `f`.
- If `ind=3`, `costf` must provide `g`.
- If `ind=4`, `costf` must provide `f` and `g`.
- If `ind=1`, `costf` can print messages.

When the `imp` option is set to a negative integer value, say `m` for example, the cost function is called back every `-m` iterations, with the `ind=1` parameter. This can be used to print messages, fill a log file or creating an interactive plot. See below for sample uses of this feature.

On output, `ind<0` means that `f` cannot be evaluated at `x` and `ind=0` interrupts the optimization.

Example: Scilab function

The following is an example with a Scilab function. Notice, for simplifications reasons, the Scilab function "cost" of the following example computes the objective function `f` and its derivative no matter of the value of `ind`. This allows to keep the example simple. In practical situations though, the computation of "f" and "g" may raise performances issues so that a direct optimization may be to use the value of "ind" to compute "f" and "g" only when needed.

```
// External function written in Scilab
function [f,g,ind] = cost(x,ind)
    xref=[1;2;3];
    f=0.5*norm(x-xref)^2;
    g=x-xref;
endfunction

// Simplest call
x0=[1;-1;1];
```

```

[f,xopt]=optim(cost,x0)

// By conjugate gradient - you can use 'qn', 'gc' or 'nd'
[f,xopt,gopt]=optim(cost,x0,'gc')

//Seen as non differentiable
[f,xopt,gopt]=optim(cost,x0,'nd')

// Upper and lower bounds on x
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0)

// Upper and lower bounds on x and setting up the algorithm to 'gc'
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc')

// Bound on the number of call to the objective function
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3)

// Set max number of call to the objective function (3)
// Set max number of iterations (100)
// Set stopping threshold on the value of f (1e-6),
// on the value of the norm of the gradient of the objective function (1e-6)
// on the improvement on the parameters x_opt (1e-6;1e-6;1e-6)
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3,100,1e-6,1e-6,[1e-6;1e-6;1e-6])

// Additionnal messages are printed in the console.
[f,xopt]=optim(cost,x0,imp=3)

```

Example: C function

The following is an example with a C function, where a C source code is written into a file, dynamically compiled and loaded into Scilab, and then used by the "optim" solver. The interface of the "rosenc" function is fixed, even if the arguments are not really used in the cost function. This is because the underlying optimization solvers must assume that the objective function has a known, constant interface. In the following example, the arrays ti and tr are not used, only the array "td" is used, as a parameter of the Rosenbrock function. Notice that the content of the arrays ti and td are the same that the content of the Scilab variable, as expected.

```

// External function written in C (C compiler required)
// write down the C code (Rosenbrock problem)
C=['#include <math.h>'
'double sq(double x)'
'{ return x*x;}'
'void rosenc(int *ind, int *n, double *x, double *f, double *g, '
'                                     int *ti, float *tr, double *td)'
'{
'  double p;'
'  int i;'
'  p=td[0];'
'  if (*ind==2||*ind==4) {'
'    *f=1.0;'
'    for (i=1;i<*n;i++)'
'      *f+=p*sq(x[i]-sq(x[i-1]))+sq(1.0-x[i]);'
'  }'
'  if (*ind==3||*ind==4) {'
'    g[0]=-4.0*p*(x[1]-sq(x[0]))*x[0];'

```

```

        for (i=1;i<*n-1;i++)'
            g[i]=2.0*p*(x[i]-sq(x[i-1]))-4.0*p*(x[i+1]-sq(x[i]))*x[i]-2.0*(1.0-x[
            g[*n-1]=2.0*p*(x[*n-1]-sq(x[*n-2]))-2.0*(1.0-x[*n-1]);'
        '}'
    '}]';
cd TMPDIR;
mputl(C,TMPDIR+'/rosenc.c')

// compile the C code
l=ilib_for_link('rosenc','rosenc.c',[],'c');

// incremental linking
link(l,'rosenc','c')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenc',x0,'td',p)

```

Example: Fortran function

The following is an example with a Fortran function.

```

// External function written in Fortran (Fortran compiler required)
// write down the Fortran code (Rosenbrock problem)
F=[ '      subroutine rosenf(ind, n, x, f, g, ti, tr, td)'
    '      integer ind,n,ti(*)'
    '      double precision x(n),f,g(n),td(*)'
    '      real tr(*)'
    'c'
    '      double precision y,p'
    '      p=td(1)'
    '      if (ind.eq.2.or.ind.eq.4) then'
    '          f=1.0d0'
    '          do i=2,n'
    '              f=f+p*(x(i)-x(i-1)**2)**2+(1.0d0-x(i))**2'
    '          enddo'
    '      endif'
    '      if (ind.eq.3.or.ind.eq.4) then'
    '          g(1)=-4.0d0*p*(x(2)-x(1)**2)*x(1)'
    '          if(n.gt.2) then'
    '              do i=2,n-1'
    '                  g(i)=2.0d0*p*(x(i)-x(i-1)**2)-4.0d0*p*(x(i+1)-x(i)**2)*x(i)'
    '                  & -2.0d0*(1.0d0-x(i))'
    '              enddo'
    '          endif'
    '          g(n)=2.0d0*p*(x(n)-x(n-1)**2)-2.0d0*(1.0d0-x(n))'
    '      endif'
    '      return'
    '      end']';
cd TMPDIR;
mputl(F,TMPDIR+'/rosenf.f')

// compile the Fortran code
l=ilib_for_link('rosenf','rosenf.f',[],'f');

```

```
// incremental linking
link(l,'rosenf','f')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenf',x0,'td',p)
```

Example: Fortran function with initialization

The following is an example with a Fortran function in which the "in" option is used to allocate memory inside the Scilab environment. In this mode, there is a dialog between Scilab and the objective function. The goal of this dialog is to initialize the parameters of the objective function. Each part of this dialog is based on a specific value of the "ind" parameter.

At the beginning, Scilab calls the objective function, with the ind parameter equals to 10. This tells the objective function to initialize the sizes of the arrays it needs by setting the nizs, nrzs and ndzs integer parameters of the "nird" common. Then the objective function returns. At this point, Scilab creates internal variables and allocate memory for the variable izs, rzs and dzs. Scilab calls the objective function back again, this time with ind equals to 11. This tells the objective function to initialize the arrays izs, rzs and dzs. When the objective function has done so, it returns. Then Scilab enters in the real optimization mode and calls the optimization solver the user requested. Whenever the objective function is called, the izs, rzs and dzs arrays have the values that have been previously initialized.

```
//
// Define a fortran source code and compile it (fortran compiler required)
//
fortransource=['      subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
'c      -----'
'c      Example of cost function given by a subroutine'
'c      if n<=2 returns ind=0'
'c      f.bonnans, oct 86'
'      implicit double precision (a-h,o-z)'
'      real rzs(1)'
'      double precision dzs(*)'
'      dimension x(n),g(n),izs(*)'
'      common/nird/nizs,nrzs,ndzs'
'      if (n.lt.3) then'
'          ind=0'
'          return'
'      endif'
'      if(ind.eq.10) then'
'          nizs=2'
'          nrzs=1'
'          ndzs=2'
'          return'
'      endif'
'      if(ind.eq.11) then'
'          izs(1)=5'
'          izs(2)=10'
'          dzs(2)=100.0d+0'
'          return'
'      endif'
'      if(ind.eq.2)go to 5'
'      if(ind.eq.3)go to 20'
'      if(ind.eq.4)go to 5'
```

```

'      ind=-1'
'      return'
'5      f=1.0d+0'
'      do 10 i=2,n'
'          im1=i-1'
'10      f=f + dzs(2)*(x(i)-x(im1)**2)**2 + (1.0d+0-x(i))**2'
'      if(ind.eq.2)return'
'20      g(1)=-4.0d+0*dzs(2)*(x(2)-x(1)**2)*x(1)'
'      nm1=n-1'
'      do 30 i=2,nm1'
'          im1=i-1'
'          ip1=i+1'
'          g(i)=2.0d+0*dzs(2)*(x(i)-x(im1)**2)'
'30      g(i)=g(i) -4.0d+0*dzs(2)*(x(ip1)-x(i)**2)*x(i) - '
'          &      2.0d+0*(1.0d+0-x(i))'
'      g(n)=2.0d+0*dzs(2)*(x(n)-x(nm1)**2) - 2.0d+0*(1.0d+0-x(n))
'      return'
'      end'];

cd TMPDIR;
mputl(fortransource,TMPDIR+'/rosenf.f')

// compile the C code
libpath=ilib_for_link('rosenf','rosenf.f',[],'f');

// incremental linking
linkid=link(libpath,'rosenf','f');

x0=1.2*ones(1,5);
//
// Solve the problem
//
[f,x,g]=optim('rosenf',x0,'in');
```

Example: Fortran function with initialization on Windows with Intel Fortran Compiler

Under the Windows operating system with Intel Fortran Compiler, one must carefully design the fortran source code so that the dynamic link works properly. On Scilab's side, the optimization component is dynamically linked and the symbol "nird" is exported out of the optimization dll. On the cost function's side, which is also dynamically linked, the "nird" common must be imported in the cost function dll.

The following example is a re-writing of the previous example, with special attention for the Windows operating system with Intel Fortran compiler as example. In that case, we introduce additionnal compiling instructions, which allows the compiler to import the "nird" symbol.

```

fortransource=['subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
'cDEC$ IF DEFINED (FORDLL)'
'cDEC$ ATTRIBUTES DLLIMPORT:: /nird/'
'cDEC$ ENDIF'
'C
'c -----'
'c Example of cost function given by a subroutine'
'c if n<=2 returns ind=0'
'c f.bonnans, oct 86'
'c implicit double precision (a-h,o-z)'
```

[etc...]

Example: Logging features

The `imp` flag may take negative integer values, say `k`. In that case, the cost function is called once every `-k` iterations. This allows to draw the function value or write a log file.

In the following example, we solve the Rosenbrock test case. For each iteration of the algorithm, we print the value of `x`, `f` and `g`.

```
function [f,g,ind] = cost(x,ind)
    xref=[1;2;3];
    if ( ind == 1 | ind == 4 ) then
        f=0.5*norm(x-xref)^2;
    end
    if ( ind == 1 | ind == 4 ) then
        g=x-xref;
    end
    if ( ind == 1 ) then
        mprintf("=====\n")
        mprintf("x = %s\n", strcat(string(x), " "))
        mprintf("f = %e\n", f)
        g=x-xref;
        mprintf("g = %s\n", strcat(string(g), " "))
    end
endfunction
x0=[1;-1;1];
[f,xopt]=optim(cost,x0,imp=-1)
```

The previous script produces the following output.

```
=====
x = 1 -1 1
f = 6.500000e+000
g = 0 -3 -2
=====
x = 1 0 1.666667
f = 2.888889e+000
g = 0 -2 -1.333333
=====
x = 1 2 3
f = 9.860761e-031
g = 0 -4.441D-16 1.332D-15
=====
x = 1 2 3
f = 0.000000e+000
g = 0 0 0
```

In the following example, we solve the Rosenbrock test case. For each iteration of the algorithm, we plot the current value of `x` into a 2D graph containing the contours of Rosenbrock's function. This allows to see the progress of the algorithm while the algorithm is performing. We could as well write the value of `x`, `f` and `g` into a log file if needed.

```
// 1. Define rosenbrock for contouring
function f = rosenbrockC ( x1 , x2 )
    x = [x1 x2];
    f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
x0 = [-1.2 1.0];
xopt = [1.0 1.0];
// 2. Draw the contour of Rosenbrock's function
xdata = linspace(-2,2,100);
ydata = linspace(-2,2,100);
contour ( xdata , ydata , rosenbrockC , [1 10 100 500 1000])
plot(x0(1) , x0(2) , "b.")
plot(xopt(1) , xopt(2) , "r*")
// 3. Define Rosenbrock for optimization
function [ f , g , ind ] = rosenbrock ( x , ind )
    if ((ind == 1) | (ind == 4)) then
        f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
    end
    if ((ind == 1) | (ind == 4)) then
        g(1) = - 400. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
        g(2) = 200. * ( x(2) - x(1)**2 )
    end
    if (ind == 1) then
        plot ( x(1) , x(2) , "g." )
    end
endfunction
// 4. Plot the optimization process, during optimization
[ fopt , xopt ] = optim ( rosenbrock , x0 , imp = -1)
```

Example: Optimizing with numerical derivatives

It is possible to optimize a problem without an explicit knowledge of the derivative of the cost function. For this purpose, we can use the numdiff or derivative function to compute a numerical derivative of the cost function.

In the following example, we use the numdiff function to solve Rosenbrock's problem.

```
function f = rosenbrock ( x )
    f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
function [ f , g , ind ] = rosenbrockCost ( x , ind )
    if ((ind == 1) | (ind == 4)) then
        f = rosenbrock ( x );
    end
    if ((ind == 1) | (ind == 4)) then
        g= numdiff ( rosenbrock , x );
    end
endfunction
x0 = [-1.2 1.0];
[ fopt , xopt ] = optim ( rosenbrockCost , x0 )
```

In the following example, we use the derivative function to solve Rosenbrock's problem. Given that the step computation strategy is not the same in numdiff and derivative, this might lead to improved results.

```
function f = rosenbrock ( x )
    f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
function [ f , g , ind ] = rosenbrockCost2 ( x , ind )
    if ((ind == 1) | (ind == 4)) then
        f = rosenbrock ( x );
    end
    if ((ind == 1) | (ind == 4)) then
        g= derivative ( rosenbrock , x.' , order = 4 );
    end
endfunction
x0 = [-1.2 1.0];
[ fopt , xopt ] = optim ( rosenbrockCost2 , x0 )
```

Example: Counting function evaluations and number of iterations

The `imp` option can take negative values. If the `imp` is equal to `m` where `m` is a negative integer, then the cost function is evaluated every `-m` iterations, with the `ind` input argument equal to 1. The following example uses this feature to compute the number of iterations. The global variable `mydata` is used to store the number of function evaluations as well as the number of iterations.

```
xref=[1;2;3];
x0=[1;-1;1];
global _MYDATA_
_MYDATA_ = tlist ( ["T_MYDATA","niter","nfevals"])
_MYDATA_.niter = 0
_MYDATA_.nfevals = 0
function [f,g,ind] = cost(x,ind)
    global _MYDATA_
    disp(ind)
    if ( ind == 1 )
        _MYDATA_.niter = _MYDATA_.niter + 1
    end
    _MYDATA_.nfevals = _MYDATA_.nfevals + 1
    f=0.5*norm(x-xref)^2;
    g=x-xref;
endfunction
[f,xopt]=optim(cost,x0,imp=-1)
mprintf ( "Number of function evaluations:%d\n",_MYDATA_.nfevals)
mprintf ( "Number of iterations:%d\n",_MYDATA_.niter)
```

While the previous example perfectly works, there is a risk that the same variable `_MYDATA_` is used by some internal function used by `optim`. In this case, the value may be wrong. This is why a sufficiently weird variable name has been used.

Example : Passing extra parameters

In most practical situations, the cost function depends on extra parameters which are required to evaluate the cost function. There are several methods to achieve this goal.

In the following example, the cost function uses 4 parameters a , b , c and d . We define the cost function with additional input arguments, which are declared after the index argument. Then we pass a list as the first input argument of the `optim` solver. The first element of the list is the cost function. The additional variables are directly passed to the cost function.

```
function [ f , g , ind ] = costfunction ( x , ind , a , b , c , d )
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
costf = list ( costfunction , a , b , c , d );
[ fopt , xopt ] = optim ( costf , x0 , imp = 2 )
```

These coefficients are defined before the optimizer is called. There are directly used in the cost function.

```
// The example NOT to follow
function [ f , g , ind ] = costfunction ( x , ind )
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
[ fopt , xopt ] = optim ( costfunction , x0 , imp = 2 )
```

While the previous example perfectly works, there is a risk that the same variables are used by some internal function used by `optim`. In this case, the value of the parameters are not what is expected and the optimization can fail or, worse, give a wrong result.

In the following example, we define the cost function with the classical header. Inside the function definition, we declare that the parameters a , b , c and d are global variables. Then we declare and set the global variables.

```
// Another example NOT to follow
function [ f , g , ind ] = costfunction ( x , ind )
```

```

    global a b c d
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
global a b c d
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
[ fopt , xopt ] = optim ( costfunction , x0 , imp = 2 )

```

While the previous example perfectly works, there is a risk that the same variables are used by some internal function used by `optim`. In this case, the value of the parameters are not what is expected and the optimization can fail or, worse, give a wrong result.

Example : Checking that derivatives are correct

Many optimization problem can be avoided if the derivatives are computed correctly. One common reason for failure in the step-length procedure is an error in the calculation of the cost function and its gradient. Incorrect calculation of derivatives is by far the most common user error.

In the following example, we give a false implementation of Rosenbrock's gradient. In order to check the computation of the derivatives, we use the `derivative` function. We define the `simplified` function, which delegates the computation of `f` to the `rosenbrock` function. The `simplified` function is passed as an input argument of the `derivative` function.

```

function [ f , g , index ] = rosenbrock ( x , index )
    f = 100.0 * (x(2)-x(1)^2)^2 + (1-x(1))^2;
    // Exact :
    g(1) = - 400. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
    // Wrong :
    g(1) = - 1200. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
    g(2) = 200. * ( x(2) - x(1)**2 )
endfunction
function f = simplified ( x )
    index = 1;
    [ f , g , index ] = rosenbrock ( x , index )
endfunction
x0 = [-1.2 1];
index = 1;
[ f , g , index ] = rosenbrock ( x0 , index );
gnd = derivative ( simplified , x0.' );
mprintf ( "Exact derivative:[%s]\n" , strcat ( string(g) , " " ));
mprintf ( "Numerical derivative:[%s]\n" , strcat ( string(gnd) , " " ));

```

The previous script produces the following output. Obviously, the difference between the two gradient is enormous, which shows that the wrong formula has been used in the gradient.

```

Exact derivative:[-638 -88]
Numerical derivative:[-215.6 -88]

```

See Also

external, qpsolve, datafit, leastsq, numdiff, derivative, NDcost

References

The following is a map from the various options to the underlying solvers, with some comments about the algorithm, when available.

"qn" without constraints

n1qn1 : a quasi-Newton method with a Wolfe-type line search

"qn" with bounds constraints

qnbd : a quasi-Newton method with projection

RR-0242 - A variant of a projected variable metric method for bound constrained optimization problems, Bonnans Frederic, Rapport de recherche de l'INRIA - Rocquencourt, Octobre 1983

"gc" without constraints

n1qn3 : a conjugate gradient method with BFGS.

"gc" with bounds constraints

gcbb : a BFGS-type method with limited memory and projection

"nd" without constraints

n1fc1 : a bundle method

"nd" with bounds constraints

not available

Author

The Modulopt library : J.Frederic Bonnans, Jean-Charles Gilbert, Claude Lemarechal

The interfaces to the Modulopt library : J.Frederic Bonnans

This help : Michael Baudin

Name

qld — linear quadratic programming solver

```
[x, lagr]=qld(Q,p,C,b,ci,cs,me [,tol])  
[x, lagr, info]=qld(Q,p,C,b,ci,cs,me [,tol])
```

Parameters

- Q**
real positive definite symmetric matrix (dimension $n \times n$).
- p**
real (column) vector (dimension n)
- C**
real matrix (dimension $(me + md) \times n$)
- b**
RHS column vector (dimension $(me + md)$)
- ci**
column vector of lower-bounds (dimension n). If there are no lower bound constraints, put **ci** = []. If some components of **x** are bounded from below, set the other (unconstrained) values of **ci** to a very large negative number (e.g. **ci(j)** = `-number_properties('huge')`).
- cs**
column vector of upper-bounds. (Same remarks as above).
- me**
number of equality constraints (i.e. **C(1:me, :)*x = b(1:me)**)
- tol**
Floating point number, required precision.
- x**
optimal solution found.
- lagr**
vector of Lagrange multipliers. If lower and upper-bounds **ci**, **cs** are provided, **lagr** has $n + me + md$ components and **lagr(1:n)** is the Lagrange vector associated with the bound constraints and **lagr (n+1 : n + me + md)** is the Lagrange vector associated with the linear constraints. (If an upper-bound (resp. lower-bound) constraint **i** is active **lagr(i)** is > 0 (resp. < 0). If no bounds are provided, **lagr** has only $me + md$ components.
- info**
integer, return the execution status instead of sending errors.
- info==1** : Too many iterations needed
- info==2** : Accuracy insufficient to satisfy convergence criterion
- info==5** : Length of working array is too short
- info==10**: The constraints are inconsistent

Description

$$\begin{aligned} \min & \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x \\ \text{with} & C(j,:) \cdot x = b(j), j = 1, \dots, me \\ & C(j,:) \cdot x \leq b(j), j = me + 1, \dots, me + md \\ & ci \leq x \leq cs \end{aligned}$$

This function requires Q to be positive definite, if it is not the case, one may use the The contributed toolbox "**quapro**".

Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1, -1, 1, 0, 3, 1;
      -1, 0, -3, -4, 5, 6;
      2, 5, 3, 0, 1, 0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0, 1, 0, 1, 2, -1;
     -1, 0, 2, 1, 1, 0];
b2=[-1;2.5];

//with x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x, lagr]=qld(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active (lagr(1:6)=0):
```

See Also

qpsolve, optim

The contributed toolbox "quapro" may also be of interest, in particular for singular Q.

Authors

K.Schittkowski
, University of Bayreuth, Germany

A.L. Tits and J.L. Zhou
, University of Maryland

Used Functions

ql0001.f in modules/optimization/src/fortran/ql0001.f

Name

qp_solve — linear quadratic programming solver builtin

```
[x [, iact [, iter [, f]]]] = qp_solve(Q, p1, C1, b, me)
```

Parameters

- Q**
real positive definite symmetric matrix (dimension $n \times n$).
- p**
real (column) vector (dimension n)
- C**
real matrix (dimension $(me + md) \times n$). This matrix may be dense or sparse.
- b**
RHS column vector (dimension $m = (me + md)$)
- me**
number of equality constraints (i.e. $x' \cdot C(:, 1:me) = b(1:me)'$)
- x**
optimal solution found.
- iact**
vector, indicator of active constraints. The first non zero entries give the index of the active constraints
- iter**
2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

Description

$$\begin{aligned} \min & \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x \\ \text{with } & x^t \cdot C(:, j) = b(j), j = 1, \dots, me \\ & x^t \cdot C(:, j) \geq b(j), j = me + 1, \dots, me + md \end{aligned}$$

This function requires **Q** to be symmetric positive definite. If this hypothesis is not satisfied, one may use the contributed **quapro toolbox**.

Examples

```
// Find x in R^6 such that:  
// x'*C1 = b1 (3 equality constraints i.e me=3)  
C1= [ 1, -1, 2;  
      -1, 0, 5;  
       1, -3, 3;  
       0, -4, 0;  
       3, 5, 1;  
       1, 6, 0];
```

```

b1=[1;2;3];

// x'*C2 >= b2 (2 inequality constraints)
C2= [ 0 ,1;
      -1, 0;
       0,-2;
      -1,-1;
      -2,-1;
       1, 0];
b2=[ 1;-2.5];

// and minimize 0.5*x'*Q*x - p'*x with
p=[-1;-2;-3;-4;-5;-6]; Q=eye(6,6);

me=3;
[x,iact,iter,f]=qp_solve(Q,p,[C1 C2],[b1;b2],me)
// Only linear constraints (1 to 4) are active

```

See Also

optim, qld, qpsolve

The contributed toolbox "quapro" may also be of interest, in particular for singular Q.

Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qp_solve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

Authors

S. Steer

INRIA (Scilab interface)

Berwin A. Turlach

School of Mathematics and Statistics (M019), The University of Western Australia, Crawley,
AUSTRALIA (solver code)

References

- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach, <http://www.maths.uwa.edu.au/~berwin/software/quadprog.html>

Used Functions

qpgen2.f and >qpgen1.f (also named QP.solve.f) developed by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

Name

qpsolve — linear quadratic programming solver

```
[x [, iact [, iter [, f]]]] = qpsolve(Q, p, C, b, ci, cs, me)
```

Parameters

- Q**
real positive definite symmetric matrix (dimension $n \times n$).
- p**
real (column) vector (dimension n)
- C**
real matrix (dimension $(me + md) \times n$). This matrix may be dense or sparse.
- b**
RHS column vector (dimension $m = (me + md)$)
- ci**
column vector of lower-bounds (dimension n). If there are no lower bound constraints, put **ci** = []. If some components of **x** are bounded from below, set the other (unconstrained) values of **ci** to a very large negative number (e.g. **ci(j) = -number_properties('huge')**).
- cs**
column vector of upper-bounds. (Same remarks as above).
- me**
number of equality constraints (i.e. $C(1:me, :) \cdot x = b(1:me)$)
- x**
optimal solution found.
- iact**
vector, indicator of active constraints. The first non zero entries give the index of the active constraints
- iter**
. 2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

Description

$$\begin{aligned} \min & \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x \\ \text{with } & C(j,:) \cdot x = b(j), j = 1, \dots, me \\ & C(j,:) \cdot x \leq b(j), j = me + 1, \dots, me + md \\ & ci \leq x \leq cs \end{aligned}$$

This function requires **Q** to be symmetric positive definite. If that hypothesis is not satisfied, one may use the **quapro** function, which is provided in the Scilab **quapro** toolbox.

The **qpsolve** solver is implemented as a Scilab script, which calls the compiled **qp_solve** primitive. It is provided as a facility, in order to be a direct replacement for the former **quapro** solver : indeed, the

qpsolve solver has been designed so that it provides the same interface, that is, the same input/output arguments. But the x0 and imp input arguments are available in quapro, but not in qpsolve.

Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1, -1, 1, 0, 3, 1;
     -1, 0, -3, -4, 5, 6;
     2, 5, 3, 0, 1, 0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0, 1, 0, 1, 2, -1;
    -1, 0, 2, 1, 1, 0];
b2=[-1;2.5];

//with x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];
cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x,iact,iter,f]=qpsolve(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active
```

See Also

optim, qp_solve, qld

The contributed toolbox "quapro" may also be of interest, in particular for singular Q.

Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qpsolve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

Authors

S. Steer
INRIA (Scilab interface)

Berwin A. Turlach

School of Mathematics and Statistics (M019), The University of Western Australia, Crawley,
AUSTRALIA (solver code)

References

- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach, <http://www.maths.uwa.edu.au/~berwin/software/quadprog.html>

Used Functions

qpge1.f (also named QP.solve.f) developed by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

Name

quapro — linear quadratic programming solver (obsolete)

Description

This function is superseded by qpsolve.

Users who are still interested by quapro may consider the Scilab quapro toolbox which provide the same features as in older Scilab releases.

See at http://www.scilab.org/contrib/index_contrib.php?page=download.php.

See Also

qpsolve

Name

readmps — reads a file in MPS format

```
mps= readmps (file-name,bounds [,maxsizes]);
```

Parameters

file-name

a string, the name of the mps file

bounds

2-vector [lowbound, upbound] , default lower and upper bounds

maxsizes

3-vector [maxm, maxn, maxnza] Maximum number of constraints and variables, maximum number of nonzeros entries in the LP constraint matrix. If omitted readmps reads the file once just to compute these numbers.

mps

tlist with following fields

irobj

integer (index of the objective row).

namec

character string (Name of the objective).

nameb

character string (Name of the right hand side).

namran

character string (Name of the ranges section).

nambnd

character string (Name of the bounds section).

name

character string (Name of the LP problem).

rownames

character string column vector (Name of the rows).

colnames

character string row vector (Name of the columns).

rowstat

integer vector, row types:

1

row type is "="

2

row type is ">="

3

row type is "<="

4

objective row

5
other free row

rowcode

real matrix [hdrowcd, lnkrow] with

hdrowcd

real vector (Header to the linked list of rows with the same codes).

lnkrow

integer vector (Linked list of rows with the same codes).

colcode

real matrix [hdcolcd, lnkcol] with

hdcolcd

integer vector (Header to the linked list of columns with the same codes).

lnkcol

integer vector (Linked list of columns with the same codes).

rownmbs

integer vector (Row numbers of nonzeros in columns of matrix A.)

colpnts

integer vector (Pointers to the beginning of columns of matrix A).

acoeff

real vector (Array of nonzero elements for each column).

rhs

real vector (Right hand side of the linear program).

ranges

real vector of constraint ranges.

bounds

real matrix [lbounds, ubounds] with

ubounds

full column vector of upper bounds

lbounds

full column vector of lower bounds

stavar

full column vector of variable status

0
standard (non negative) variable

1
upper bounded variable

2
lower bounded variable

3
lower and upper bounded variable

4
minus infinity type variable i.e $-\text{inf} < x \leq u$

- 5
plus infinity type variable i.e $l \leq x < \text{inf}$
- 6
fixed type variable i.e $l = x = u$
- k
free variable

Description

Reads a file containing description of an LP problem given in MPS format and returns a `tList` which describes the problem. It is an interface with the program `rdmps1.f` of hopdm (J. Gondzio). For a description of the variables, see the file `rdmps1.f`.

MPS format is a standard ASCII medium for LP codes. MPS format is described in more detail in Murtagh's book:

Murtagh B. (1981). Advanced Linear Programming, McGraw-Hill, New York, 1981.

Examples

```
//Let the LP problem:
//objective:
//  min      XONE + 4 YTWO + 9 ZTHREE
//constraints:
//  LIM1:    XONE +   YTWO                < = 5
//  LIM2:    XONE +                ZTHREE > = 10
//  MYEQN:           -   YTWO  +  ZTHREE  = 7
//Bounds
//  0 <= XONE <= 4
// -1 <= YTWO <= 1

//Generate MPS file
txt=['NAME          TESTPROB'
    'ROWS'
    ' N  COST'
    ' L  LIM1'
    ' G  LIM2'
    ' E  MYEQN'
    'COLUMNS'
    '   XONE      COST          1   LIM1          1'
    '   XONE      LIM2          1'
    '   YTWO      COST          4   LIM1          1'
    '   YTWO      MYEQN        -1'
    '   ZTHREE     COST          9   LIM2          1'
    '   ZTHREE     MYEQN          1'
    'RHS'
    '   RHS1      LIM1          5   LIM2          10'
    '   RHS1      MYEQN          7'
    'BOUNDS'
    ' UP BND1     XONE          4'
    ' LO BND1     YTWO        -1'
    ' UP BND1     YTWO          1'
    'ENDATA'];
mputl(txt,TMPDIR+'/test.mps')
```



```
//Read the MPS file  
P=readmps(TMPDIR+'/test.mps',[0 10^30])  
disp(P)
```

Nom

recons — reciprocal function for aplat.

```
r = recons(fl, ind)
```

Parameters

- fl
a "flat" list.
- ind
a list of paths
- r
a hierachical list build with the leaves of fl

Description

Reciprocal function for aplat. Creates a hierachical list given a flat one and a list of paths.

utility function for vec2list and lmisolver.

Examples

```
[lf, ind]=aplat(list(1,2, list([3,1], 'xxx', list([3,2,1]))));  
recons(lf, ind)
```

See Also

aplat

Authors

F.D and S.S. INRIA

Name

semidef — semidefinite programming

```
[x,Z,ul,info]=semidef(x0,Z0,F,blk_szs,c,options)
```

Parameters

x0

m x 1 real column vector (must be strictly primal feasible, see below)

Z0

L x 1 real vector (compressed form of a strictly feasible dual matrix, see below)

F

L x (m+1) real matrix

blk_szs

p x 2 integer matrix (sizes of the blocks) defining the dimensions of the (square) diagonal blocks
size(Fi(j))=blk_szs(j) j=1, ..., m+1.

c

m x 1 real vector

options

row vector with five entries [nu, abstol, reltol, 0, maxiters]

ul

row vector with two entries

Description

[x,Z,ul,info]=semidef(x0,Z0,F,blk_szs,c,options) solves semidefinite program:

$$\begin{aligned} & \min c^t \cdot x \\ & \text{with } F_0 + x_1 \cdot F_1 + \dots + x_m \cdot F_m \geq 0 \end{aligned}$$

and its dual:

$$\begin{aligned} & \max - \text{trace}(F_0 \cdot Z) \\ & \text{with } \text{trace}(F_i \cdot Z) = c_i, i = 1, \dots, m \\ & Z \geq 0 \end{aligned}$$

exploiting block structure in the matrices F_i.

It interfaces L. Vandenberghe and S. Boyd sp.c program.

The Fi's matrices are stored columnwise in F in compressed format: if Fi^j, i=0,...,m, j=1,...,L denote the jth (symmetric) diagonal block of Fi, then

$$F = \begin{pmatrix} \text{pack}(F_0^1) & \text{pack}(F_1^1) & \dots & \text{pack}(F_m^1) \\ \text{pack}(F_0^2) & \text{pack}(F_1^2) & \dots & \text{pack}(F_m^2) \\ \dots & \dots & \dots & \dots \\ \text{pack}(F_0^L) & \text{pack}(F_1^L) & \dots & \text{pack}(F_m^L) \end{pmatrix}$$

where $\text{pack}(M)$, for symmetric M , is the vector $[M(1,1); M(1,2); \dots; M(1,n); M(2,2); M(2,3); \dots; M(2,n); \dots; M(n,n)]$ (obtained by scanning columnwise the lower triangular part of M).

`blk_szs` gives the size of block j , ie, $\text{size}(F_{i^j}) = \text{blk\_szs}(j)$.

Z is a block diagonal matrix with L blocks $Z^0, \dots, Z^{L-1}$. Z^j has size `blk_szs[j]` times `blk_szs[j]`. Every block is stored using packed storage of the lower triangular part.

The 2 vector `ul` contains the primal objective value $c'x$ and the dual objective value $-\text{trace}(F_0^*Z)$.

The entries of `options` are respectively: `nu` = a real parameter which ntrols the rate of convergence. `abstol` = absolute tolerance. `reltol` = relative tolerance (has a special meaning when negative). `tv` target value, only referenced if `reltol` < 0. `iters` = on entry: maximum number of iterations >= 0, on exit: the number of iterations taken. Notice that the absolute tolerance cannot be lower than $1.0e-8$, that is, the absolute tolerance used in the algorithm is the maximum of the user-defined tolerance and the constant tolerance $1.0e-8$.

`info` returns 1 if maxiters exceeded, 2 if absolute accuracy is reached, 3 if relative accuracy is reached, 4 if target value is reached, 5 if target value is not achievable; negative values indicate errors.

Convergence criterion:

- (1) maxiters is exceeded
- (2) duality gap is less than `abstol`
- (3) primal and dual objective are both positive and duality gap is less than (`reltol` * dual objective) or primal and dual objective are both negative and duality gap is less than (`reltol` * minus the primal objective)
- (4) `reltol` is negative and primal objective is less than `tv` or dual objective is greater than `tv`

Examples

```
F0=[2,1,0,0;
    1,2,0,0;
    0,0,3,1
    0,0,1,3];

F1=[1,2,0,0;
    2,1,0,0;
    0,0,1,3;
    0,0,3,1]
```

```
F2=[2,2,0,0;
    2,2,0,0;
    0,0,3,4;
    0,0,4,4];

blk_szs=[2,2];

F01=F0(1:2,1:2);F02=F0(3:4,3:4);
F11=F1(1:2,1:2);F12=F1(3:4,3:4);
F21=F2(1:2,1:2);F22=F2(3:4,3:4);

x0=[0;0]
Z0=2*F0;
Z01=Z0(1:2,1:2);Z02=Z0(3:4,3:4);
FF=[ [F01(:);F02(:)], [F11(:);F12(:)], [F21(:);F22(:)]]
ZZ0=[ [Z01(:);Z02(:)]];

c=[trace(F1*Z0);trace(F2*Z0)];
options=[10,1.d-10,1.d-10,0,50];

[x,Z,ul,info]=semidef(x0,pack(ZZ0),pack(FF),blk_szs,c,options)

w=vec2list(unpack(Z,blk_szs),[blk_szs;blk_szs]);Z=sysdiag(w(1),w(2))

c'*x+trace(F0*Z)
spec(F0+F1*x(1)+F2*x(2))
trace(F1*Z)-c(1)
trace(F2*Z)-c(2)
```

References

L. Vandenberghe and S. Boyd, "Semidefinite Programming," Informations Systems Laboratory, Stanford University, 1994.

Ju. E. Nesterov and M. J. Todd, "Self-Scaled Cones and Interior-Point Methods in Nonlinear Programming," Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, April 1994.

SP: Software for Semidefinite Programming, <http://www.ee.ucla.edu/~vandenbe/sp.html>

Name

vec2list — list2vec reciprocal function

```
li=vec2list(bigVector,varsizes,ind)
```

Parameters

bigVector

An m by n matrix. Each column is used to generate the corresponding list entry.

varsizes

An n by 2 matrix. Each row give the dimensions of the matrix to be built with the corresponding column of bigVector.

ind

a list with n entries. Each entry is a vector of positive integers which gives the hierchical path where the corresponding matrix has to be put.

li

a list or a recursive list with n leaves. The list entries (or leaves) are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes.

Description

If the ind argument is not given, this function creates a list. The list entries (or leaves) are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes.

If the ind argument is given, this function creates a hierachical list with n leaves. The leaves are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes. The hierachical path for each leaf if given by the corresponding entry of ind.

This function is a subsidiary for lmisolver

Examples

```
vec2list(1:4,ones(4,2))
vec2list(1:4,[2 1;1 2])
vec2list(1:4,ones(4,2),list(1,2,[3,1],[3,2,1]))
```

See Also

list2vec

Authors

F.D., INRIA

S.S., INRIA

Part XXVI. Overloading

Name

overloading — display, functions and operators overloading capabilities

Description

In scilab, variable display, functions and operators may be defined for new objects using functions (scilab coded or primitives).

Display

The display of new objects defined by `tlist` structure may be overloaded (the default display is similar to `list`'s one). The overloading function must have no output argument a single input argument. It's name is formed as follow `%<tlist_type>_p` where `%<tlist_type>` stands for the first entry of the `tlist` type component truncated to the first 9 characters.

Operators

Each operator which is not defined for given operands type may be defined. The overloading function must have a single output argument and one or two inputs according to the number of operands. The function name is formed as follow:

for binary operators: `%<first_operand_type>_<op_code>_<second_operand_type>`

for unary operators: `%<operand_type>_<op_code>`

extraction and insertion operators which are n-nary operators are described below.

Be careful, only the types registered by the `typename` function can be used in an overloading macros

`<operand_type>`, `<first_operand_type>`, `<second_operand_type>` are sequence of characters associated with each data type as described in the following table:

data type	char code	typeof	comments
double matrix	s	constant	
polynomial matrix	p	polynomial	
boolean matrix	b	boolean	
sparse matrix	sp	sparse	
boolean sparse matrix	spb	boolean sparse	
Matlab sparse matrix	msh	Matlab sparse	
integer matrix	i	int8, int16,int32, uint8, uint16, uint32	
string matrix	s	string	
handle	h	handle	
compiled function	fptr	fptr	
script function	function	mc	
library	library	f	
list	list	l	
tlist	tlist type	tlist type	the first string in the first tlist entry
mlist	mlist type	mlist type	the first string in the first mlist entry
hypermatrix	hypermat	hm	

pointer	pointer	ptr	

<op_code> is a single character associated with each operator as described in the following table:

op	char code	op	char code
'	t	+	a
-	s	*	m
/	r	\	l
^	p	.*	x
./	d	.\	q
.*.	k	./.	y
.\.	z	:	b
*.	u	./.	v
\.	w	[a,b]	c
[a;b]	f	() extraction	e
() insertion	i	==	o
<>	n		g
&	h	.^	j
~	5	.'	0
<	1	>	2
<=	3	>=	4
		iext	6

The overloading function for extraction syntax `b=a(i1, ..., in)` has the following calling sequence: `b=%<type_of_a>_e_(i1, ..., in, a)`

and the syntax `[x1, ..., xm]=a(i1, ..., in)` has the following calling sequence: `[x1, ..., xm]=%<type_of_a>_e_(i1, ..., in, a)`

The overloading function associated to the insertion syntax `a(i1, ..., in)=b` has the following calling sequence: `a=%<type_of_b>_i_<type_of_a>(i1, ..., in, b, a)`.

The 6 char code may be used for some complex insertion algorithm like `x.b(2)=33` where `b` field is not defined in the structure `x`. The insertion is automatically decomposed into `temp=x.b; temp(2)=33; x.b=temp`. The 6 char code is used for the first step of this algorithm. The 6 overloading function is very similar to the `e`'s one.

Functions :

Some basic primitive function

may also be overloaded for new data type. When such a function is undefined for a particular data types the function `%<type_of_an_argument>_<function_name>` is called. User may add in this called function the definition associated with the input data types.

Examples

```
//DISPLAY
deff('[]=%tab_p(l)', 'disp([[ ' ' ';l(3)] [l(2);string(l(4))]])')
tlist('tab', ['a', 'b'], ['x'; 'y'], rand(2,2))
```

```
//OPERATOR
deff('x=%c_a_s(a,b)', 'x=a+string(b)')
's'+1

//FUNCTION
deff('x=%c_sin(a)', 'x=''sin(''+a+'')''')
sin('2*x')
```

See Also

[tlist](#), [disp](#), [symbols](#), [typeof](#), [type](#), [type](#)

Part XXVII. Polynomials

Name

bezout — Bezout equation for polynomials or integers

```
[thegcd,U]=bezout(p1,p2)
```

Parameters

p1, p2

two real polynomials or two integer scalars (type equal to 8)

Description

`[thegcd,U]=bezout(p1,p2)` computes GCD `thegcd` of `p1` and `p2` and in addition a (2x2) unimodular matrix `U` such that:

$$\begin{bmatrix} p1 & p2 \end{bmatrix} * U = \begin{bmatrix} thegcd & 0 \end{bmatrix}$$

The lcm of `p1` and `p2` is given by:

$$p1 * U(1, 2) \text{ (or } -p2 * U(2, 2))$$

Examples

```
// polynomial case
x=poly(0,'x');
p1=(x+1)*(x-3)^5;p2=(x-2)*(x-3)^3;
[thegcd,U]=bezout(p1,p2)
det(U)
clean([p1,p2]*U)
thelcm=p1*U(1,2)
lcm([p1,p2])

// integer case
i1=int32(2^3^5); i2=int32(2^3^3^2);
[thegcd,U]=bezout(i1,i2)
V=int32([2^2^3^5, 2^3^3^2, 2^2^3^4^5]);
[thegcd,U]=gcd(V)
V*U
lcm(V)
```

See Also

`poly` , `roots` , `simp` , `clean` , `lcm`

Authors

S. Steer INRIA

Name

`clean` — cleans matrices (round to zero small entries)

```
B=clean(A [, epsa [, epsr]])
```

Parameters

`A`
a numerical matrix (scalar, polynomial, sparse...)

`epsa, epsr`
real numbers. Cleaning tolerances (default values resp. 10^{-10} and 10^{-10})

Description

This function eliminates (i.e. set to zero) all the coefficients with absolute value $< \text{epsa}$ or relative value $< \text{epsr}$ (relative means relative w.r.t. 1-norm of coefficients) in a polynomial (possibly matrix polynomial or rational matrix).

Default values are `epsa=1.d-10` and `epsr=1.d-10`;

For a constant (non polynomial) matrix `clean(A, epsa)` sets to zero all entries of `A` smaller than `epsa`.

Examples

```
x=poly(0, 'x');  
w=[x, 1, 2+x; 3+x, 2-x, x^2; 1, 2, 3+x]/3;  
w*inv(w)  
clean(w*inv(w))
```

Name

cmndred — common denominator form

```
[n, d]=cmndred(num, den)
```

Parameters

num, den
two polynomial matrices of same dimensions

Description

`[n, d]=cmndred(num, den)` computes a polynomial matrix `n` and a common denominator polynomial `d` such that:

$n/d = \text{num} ./ \text{den}$

The rational matrix defined by `num ./ den` is `n/d`

See Also

`simp` , `clean`

Name

coeff — coefficients of matrix polynomial

```
[C]=coeff(Mp [,v])
```

Parameters

Mp
polynomial matrix

v
integer (row or column) vector of selected degrees

C
big matrix of the coefficients

Description

`C=coeff(Mp)` returns in a big matrix **C** the coefficients of the polynomial matrix **Mp**. **C** is partitioned as `C=[C0,C1,...,Ck]` where the **Ci** are arranged in increasing order $k = \max_i(\text{degree}(\text{Mp}))$

`C=coeff(Mp,v)` returns the matrix of coefficients with degree in **v**. (**v** is a row or column vector).

See Also

poly , degree , inv_coeff

Authors

S. Steer INRIA

Name

coffg — inverse of polynomial matrix

```
[Ns, d]=coffg(Fs)
```

Parameters

Fs
square polynomial matrix

Description

coffg computes FS^{-1} where FS is a polynomial matrix by co-factors method.

FS inverse = NS/d

d = common denominator; NS = numerator (a polynomial matrix)

(For large matrices, be patient...results are generally reliable)

Examples

```
s=poly(0, 's')
a=[ s, s^2+1; s s^2-1];
[a1,d]=coffg(a);
(a1/d)-inv(a)
```

See Also

determ , detr , invr , penlaur , glever

Authors

F. D.; ;

Name

colcompr — column compression of polynomial matrix

```
[Y,rk,ac]=colcompr(A);
```

Parameters

A
polynomial matrix

Y
square polynomial matrix (right unimodular basis)

rk
normal rank of A

Ac
Ac=A*Y, polynomial matrix

Description

column compression of polynomial matrix A (compression to the left)

Examples

```
s=poly(0,'s');  
p=[s;s*(s+1)^2;2*s^2+s^3];  
[Y,rk,ac]=colcompr(p*p');  
p*p'*Y
```

See Also

rowcompr

Name

degree — degree of polynomial matrix

```
[D]=degree(M)
```

Parameters

M
polynomial matrix

D
integer matrix

Description

returns the matrix of highest degrees of M.

See Also

poly , coeff , clean

Name

denom — denominator

```
den=denom(r)
```

Parameters

`r`
rational or polynomial or constant matrix.

`den`
polynomial matrix

Description

`den=denom(r)` returns the denominator of a rational matrix.

Since rationals are internally represented as `r=list(['r','num','den','dt'],num,den,[ ])`, `denom(r)` is the same as `r(3)`, `r('den')` or `r.den`

See Also

`numer`

Name

derivat — rational matrix derivative

```
pd=derivat(p)
```

Parameters

p
polynomial or rational matrix

Description

computes the derivative of the polynomial or rational function matrix w.r.t the dummy variable.

Examples

```
s=poly(0, 's');  
derivat(1/s)  // -1/s^2;
```

Name

determ — determinant of polynomial matrix

```
res=determ(w [, k])
```

Parameters

w

real square polynomial matrix

k

integer (upper bound for the degree of the determinant of W)

Description

returns the determinant of a real polynomial matrix (computation made by FFT if W size is greater than 2*2).

`res=determ(w [, k])` k is an integer larger than the actual degree of the determinant of W.

The default value of k is the smallest power of 2 which is larger than `n*maxi(degree(w))`.

Method (Only if W size is greater than 2*2) : evaluate the determinant of W for the Fourier frequencies and apply inverse FFT to the coefficients of the determinant.

Examples

```
s=poly(0, 's');  
w=s*rand(10,10);  
determ(w)  
det(coeff(w,1))*s^10
```

See Also

det , detr , coffg

Authors

F.D.

Name

`detr` — polynomial determinant

```
d=detr(h)
```

Parameters

`h`
polynomial or rational square matrix

Description

`d=detr(h)` returns the determinant `d` of the polynomial or rational function matrix `h`. Based on Leverrier's algorithm.

See Also

`det` , `determ`

Name

diophant — diophantine (Bezout) equation

```
[x,err]=diophant(p1p2,b)
```

Parameters

p1p2
polynomial vector $p1p2 = [p1 \ p2]$

b
polynomial

x
polynomial vector $[x1;x2]$

Description

diophant solves the bezout equation:

$p1*x1+p2*x2=b$ with p1p2 a polynomial vector. If the equation is not solvable

else err=0

Examples

```
s=poly(0,'s');p1=(s+3)^2;p2=(1+s);  
x1=s;x2=(2+s);  
[x,err]=diophant([p1,p2],p1*x1+p2*x2);  
p1*x1+p2*x2-p1*x(1)-p2*x(2)
```

Name

factors — numeric real factorization

```
[lnum,g]=factors(pol [, 'flag'])  
[lnum,lden,g]=factors(rat [, 'flag'])  
rat=factors(rat, 'flag')
```

Parameters

pol
real polynomial

rat
real rational polynomial (rat=pol1/pol2)

lnum
list of polynomials (of degrees 1 or 2)

lden
list of polynomials (of degrees 1 or 2)

g
real number

flag
character string 'c' or 'd'

Description

returns the factors of polynomial **pol** in the list **lnum** and the "gain" **g**.

One has $\text{pol} = g$ times product of entries of the list **lnum** (if **flag** is not given). If **flag**='c' is given, then one has $|\text{pol}(i \text{ omega})| = |g \cdot \text{prod}(\text{lnum}_j(i \text{ omega})|$. If **flag**='d' is given, then one has $|\text{pol}(\exp(i \text{ omega}))| = |g \cdot \text{prod}(\text{lnum}_i(\exp(i \text{ omega}))|$. If argument of **factors** is a 1x1 rational rat=pol1/pol2, the factors of the numerator pol1 and the denominator pol2 are returned in the lists **lnum** and **lden** respectively.

The "gain" is returned as **g**, i.e. one has: rat= g times (product entries in lnum) / (product entries in lden).

If **flag** is 'c' (resp. 'd'), the roots of **pol** are reflected wrt the imaginary axis (resp. the unit circle), i.e. the factors in **lnum** are stable polynomials.

Same thing if **factors** is invoked with a rational arguments: the entries in **lnum** and **lden** are stable polynomials if **flag** is given. **R2=factors(R1, 'c')** or **R2=factors(R1, 'd')** with **R1** a rational function or SISO **syslin** list then the output **R2** is a transfer with stable numerator and denominator and with same magnitude as **R1** along the imaginary axis ('c') or unit circle ('d').

Examples

```
n=poly([0.2,2,5], 'z');  
d=poly([0.1,0.3,7], 'z');  
R=syslin('d', n, d);  
R1=factors(R, 'd')  
roots(R1('num'))  
roots(R1('den'))  
w=exp(2*i*pi*[0:0.1:1]);
```



```
norm(abs(horner(R1,w))-abs(horner(R,w)))
```

See Also

[simp](#)

Name

gcd — gcd calculation

```
[pgcd, U]=gcd(p)
```

Parameters

p
polynomial row vector $p=[p_1, \dots, p_n]$ or integer row vector (type equal to 8)

Description

computes the gcd of components of **p** and a unimodular matrix (with polynomial inverse) **U**, with minimal degree such that

$$p*U = [0 \quad \dots \quad 0 \quad \text{pgcd}]$$

Examples

```
//polynomial case
s=poly(0,'s');
p=[s,s*(s+1)^2,2*s^2+s^3];
[pgcd,u]=gcd(p);
p*u

//integer case
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
[thegcd,U]=gcd(V)
V*U
```

See Also

bezout , lcm , hermit

Name

hermit — Hermite form

```
[Ar,U]=hermit(A)
```

Parameters

A
polynomial matrix

Ar
triangular polynomial matrix

U
unimodular polynomial matrix

Description

Hermite form: U is an unimodular matrix such that $A*U$ is in Hermite triangular form:

The output variable is $Ar=A*U$.

Warning: Experimental version

Examples

```
s=poly(0,'s');  
p=[s, s*(s+1)^2, 2*s^2+s^3];  
[Ar,U]=hermit(p'*p);  
clean(p'*p*U), det(U)
```

See Also

hrmt , htrianr

Name

horner — polynomial/rational evaluation

```
horner(P,x)
```

Parameters

P
polynomial or rational matrix

x
array of numbers or polynomials or rationals

Description

evaluates the polynomial or rational matrix $P = P(s)$ when the variable s of the polynomial is replaced by x :

$\text{horner}(P, x) = P(x)$

Example (Bilinear transform): Assume $P = P(s)$ is a rational matrix then the rational matrix $P((1+s)/(1-s))$ is obtained by $\text{horner}(P, (1+s)/(1-s))$.

To evaluate a rational matrix at given frequencies use preferably the `freq` primitive.

Examples

```
//evaluation of a polynomial for a vector of numbers
P=poly(1:3,'x')
horner(P,[1 2 5])
horner(P,[1 2 5]+%i)

//evaluation of a rational
s=poly(0,'s');M=[s,1/s];
horner(M,1)
horner(M,%i)
horner(M,1/s)

//evaluation of a polynomial for a matrix of numbers
X= [1 2;3 4]
p=poly(1:3,'x','c')
m=horner(p, X)
1*X.^0+2*X.^1+3*X.^2
```

See Also

`freq` , `repfreq` , `evstr`

Name

hrmt — gcd of polynomials

```
[pg,U]=hrmt(v)
```

Parameters

v
row of polynomials i.e. 1xk polynomial matrix

pg
polynomial

U
unimodular matrix polynomial

Description

`[pg,U]=hrmt(v)` returns a unimodular matrix **U** and **pg** = gcd of row of polynomials **v** such that $v*U = [pg, \theta]$.

Examples

```
x=poly(0,'x');  
v=[x*(x+1),x^2*(x+1),(x-2)*(x+1),(3*x^2+2)*(x+1)];  
[pg,U]=hrmt(v);U=clean(U)  
det(U)
```

See Also

gcd, htianr

Authors

S. Steer INRIA

Name

htrianr — triangularization of polynomial matrix

```
[Ar,U,rk]=htrianr(A)
```

Parameters

A
polynomial matrix

Ar
polynomial matrix

U
unimodular polynomial matrix

rk
integer, normal rank of A

Description

triangularization of polynomial matrix A.

A is $[m,n]$, $m \leq n$.

$Ar=A*U$

Warning: there is an elimination of "small" terms (see function code).

Examples

```
x=poly(0,'x');  
M=[x;x^2;2+x^3]*[1,x-2,x^4];  
[Mu,U,rk]=htrianr(M)  
det(U)  
M*U(:,1:2)
```

See Also

hrmt , colcompr

Name

`invr` — inversion of (rational) matrix

```
F = invr(H)
```

Parameters

H
polynomial or rational matrix

F
polynomial or rational matrix

Description

If H is a polynomial or rational function matrix, `invr` computes H^{-1} using Leverrier's algorithm (see function code)

Examples

```
s=poly(0,'s')
H=[s,s*s+2;1-s,1+s]; invr(H)
[Num,den]=coffg(H);Num/den
H=[1/s,(s+1);1/(s+2),(s+3)/s];invr(H)
```

See Also

`glever` , `coffg` , `inv`

Name

lcm — least common multiple

```
[pp, fact]=lcm(p)
```

Parameters

p

fact

polynomial vector or integer vector (type equal to 8)

pp

polynomial or integer

Description

`pp=lcm(p)` computes the lcm pp of polynomial vector p.

`[pp, fact]=lcm(p)` computes in addition the vector fact such that:

`p.*fact=pp*ones(p)`

Examples

```
//polynomial case
s=poly(0, 's');
p=[s, s*(s+1)^2, s^2*(s+2)];
[pp, fact]=lcm(p);
p.*fact, pp

//integer case
V=int32([2^2*3^5, 2^3*3^2, 2^2*3^4*5]);
lcm(V)
```

See Also

`gcd`, `bezout`

Name

lcmdiag — least common multiple diagonal factorization

```
[N,D]=lcmdiag(H)
[N,D]=lcmdiag(H,flag)
```

Parameters

H
rational matrix

N
polynomial matrix

D
diagonal polynomial matrix

flag
character string: 'row' or 'col' (default)

Description

`[N,D]=lcmdiag(H, 'row')` computes a factorization $D^*H=N$, i.e. $H=D^{(-1)}*N$ where D is a diagonal matrix with $D(k,k)=\text{lcm}$ of kth row of H('den').

`[N,D]=lcmdiag(H)` or `[N,D]=lcmdiag(H, 'col')` returns $H=N*D^{(-1)}$ with diagonal D and $D(k,k)=\text{lcm}$ of kth col of H('den')

Examples

```
s=poly(0,'s');
H=[1/s,(s+2)/s/(s+1)^2;1/(s^2*(s+2)),2/(s+2)];
[N,D]=lcmdiag(H);
N/D-H
```

See Also

lcm , gcd , bezout

Name

ldiv — polynomial matrix long division

```
[x]=ldiv(n,d,k)
```

Parameters

n,d
two real polynomial matrices

k
integer

Description

$x=ldiv(n,d,k)$ gives the k first coefficients of the long division of n by d i.e. the Taylor expansion of the rational matrix $[n_{ij}(z)/d_{ij}(z)]$ near infinity.

Coefficients of expansion of n_{ij}/d_{ij} are stored in $x((i-1)*n+k,j)$ $k=1:n$

Examples

```
wss=ssrand(1,1,3);[a,b,c,d]=abcd(wss);  
wtf=ss2tf(wss);  
x1=ldiv(numden(wtf),denom(wtf),5)  
x2=[c*b;c*a*b;c*a^2*b;c*a^3*b;c*a^4*b]  
wssbis=markp2ss(x1',5,1,1);  
wtfbis=clean(ss2tf(wssbis))  
x3=ldiv(numden(wtfbis),denom(wtfbis),5)
```

See Also

arl2 , markp2ss , pdiv

Name

numer — numerator

```
num=numer(R)
```

Parameters

R
rational or polynomial or constant matrix.

num
polynomial matrix

Description

Utility fonction. `num=numer(R)` returns the numerator `num` of a rational function matrix `R` (`R` may be also a constant or polynomial matrix). `numer(R)` is equivalent to `R(2)`, `R('num')` or `R.num`

See Also

`denom`

Name

pdiv — polynomial division

```
[R,Q]=pdiv(P1,P2)
[Q]=pdiv(P1,P2)
```

Parameters

P1
polynomial matrix

P2
polynomial or polynomial matrix

R,Q
two polynomial matrices

Description

Element-wise euclidan division of the polynomial matrix P1 by the polynomial P2 or by the polynomial matrix P2. R_{ij} is the matrix of remainders, Q_{ij} is the matrix of quotients and $P1_{ij} = Q_{ij} * P2 + R_{ij}$ or $P1_{ij} = Q_{ij} * P2_{ij} + R_{ij}$.

Examples

```
x=poly(0,'x');
p1=(1+x^2)*(1-x);p2=1-x;
[r,q]=pdiv(p1,p2)
p2*q-p1
p2=1+x;
[r,q]=pdiv(p1,p2)
p2*q+r-p1
```

See Also

ldiv , gcd

Name

pol2des — polynomial matrix to descriptor form

```
[N,B,C]=pol2des(Ds)
```

Parameters

Ds
polynomial matrix

N, B, C
three real matrices

Description

Given the polynomial matrix $DS=D_0 +D_1 s +D_2 s^2 +\dots +D_k s^k$, `pol2des` returns three matrices N , B , C , with N nilpotent such that:

$$Ds = C (s^N - \text{eye}())^{-1} B$$

Examples

```
s=poly(0,'s');  
G=[1,s;1+s^2,3*s^3];[N,B,C]=pol2des(G);  
G1=clean(C*inv(s*N-eye())*B),G2=numer(G1)
```

See Also

ss2des , tf2des

Authors

F.D.;

Name

pol2str — polynomial to string conversion

```
[str]=pol2str(p)
```

Parameters

p
real polynomial

str
character string

Description

converts polynomial to character string (utility function).

See Also

string , pol2tex

Name

polfact — minimal factors

```
[f]=polfact(p)
```

Parameters

p
polynomial

f
vector $[f_0 \ f_1 \ \dots \ f_n]$ such that $p=\text{prod}(f)$

f_0
constant

f_i
polynomial

Description

$f=\text{polfact}(p)$ returns the minimal factors of p i.e. $f=[f_0 \ f_1 \ \dots \ f_n]$ such that $p=\text{prod}(f)$

See Also

lcm , cmndred , factors

Authors

S. Steer INRIA

Name

residu — residue

```
[V]=residu(P,Q1,Q2)
```

Parameters

P, Q1, Q2

polynomials or matrix polynomials with real or complex coefficients.

Description

`V=residu(P,Q1,Q2)` returns the matrix `V` such that $V(i,j)$ is the sum of the residues of the rational fraction $P(i,j)/(Q1(i,j)*Q2(i,j))$ calculated at the zeros of $Q1(i,j)$.

$Q1(i,j)$ and $Q2(i,j)$ must not have any common root.

Examples

```
s=poly(0,'s');
H=[s/(s+1)^2,1/(s+2)];N=numer(H);D=denom(H);
w=residu(N.*horner(N,-s),D,horner(D,-s)); //N(s) N(-s) / D(s) D(-s)
sqrt(sum(w)) //This is H2 norm
h2norm(tf2ss(H))

p=(s-1)*(s+1)*(s+2)*(s+10);a=(s-5)*(s-1)*(s*s)*((s+1/2)**2);
b=(s-3)*(s+2/5)*(s+3);
residu(p,a,b)+531863/4410 //Exact
z=poly(0,'z');a=z^3+0.7*z^2+0.5*z-0.3;b=z^3+0.3*z^2+0.2*z+0.1;
atild=gtild(a,'d');btild=gtild(b,'d');
residu(b*btild,z*a,atild)-2.9488038 //Exact
a=a+0%i;b=b+0%i;
real(residu(b*btild,z*a,atild)-2.9488038) //Complex case
```

See Also

pfss , bdiag , roots , poly , gtild

Authors

F.Delebecque INRIA

Name

roots — roots of polynomials

```
[x]=roots(p)
[x]=roots(p, 'e')
```

Parameters

p
polynomial with real or complex coefficients or vector of the polynomial coefficients in decreasing degree order (Matlab compatibility).

Description

`x=roots(p)` returns in the complex vector `x` the roots of the polynomial `p`. For real polynomials of degree ≤ 100 the fast RPOLY algorithm (based on Jenkins-Traub method) is used. In the other cases the roots are computed as the eigenvalues of the associated companion matrix. Use `x=roots(p, 'e')` to force this algorithm in any cases.

Examples

```
p=poly([0,10,1+%i,1-%i], 'x');
roots(p)
A=rand(3,3);roots(poly(A, 'x'))    // Evals by characteristic polynomial
spec(A)
```

See Also

`poly`, `spec`, `companion`

Authors

Serge Steer (INRIA)

References

The RPOLY algorithm is described in "Algorithm 493: Zeros of a Real Polynomial", ACM TOMS Volume 1, Issue 2 (June 1975), pp. 178-189

Jenkins, M. A. and Traub, J. F. (1970), A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration, SIAM J. Numer. Anal., 7(1970), 545-566.

Jenkins, M. A. and Traub, J. F. (1970), Principles for Testing Polynomial Zerofinding Programs. ACM TOMS 1, 1 (March 1975), pp. 26-34

Used Functions

The `rpoly.f` source codes can be found in the directory `SCI/modules/polynomials/src/fortran` of a Scilab source distribution. In the case where the companion matrix is used, the eigenvalue computation is performed using `DGEEV` and `ZGEEV` LAPACK codes.

Name

rowcompr — row compression of polynomial matrix

```
[X, rk, Ac]=rowcompr(A)
```

Parameters

A
polynomial matrix

Y
square polynomial matrix (left unimodular basis)

rk
normal rank of A

Ac
Ac=X*A, polynomial matrix

Description

row compression of polynomial matrix A.

X is a left polynomial unimodular basis which row compressed thee rows of A. rk is the normal rank of A.

Warning: elimination of "small" terms (use with care!).

See Also

colcompr

Name

sfact — discrete time spectral factorization

```
F=sfact(P)
```

Parameters

P
real polynomial matrix

Description

Finds F, a spectral factor of P. P is a polynomial matrix such that each root of P has a mirror image w.r.t the unit circle. Problem is singular if a root is on the unit circle.

sfact(P) returns a polynomial matrix $F(z)$ which is antistable and such that

$$P = F(z) * F(1/z) * z^n$$

For scalar polynomials a specific algorithm is implemented. Algorithms are adapted from Kucera's book.

Examples

```
//Simple polynomial example
z=poly(0,'z');
p=(z-1/2)*(2-z)
w=sfact(p);
w*numer(horner(w,1/z))

//matrix example
F1=[z-1/2,z+1/2,z^2+2;1,z,-z;z^3+2*z,z,1/2-z];
P=F1*gtild(F1,'d'); //P is symmetric
F=sfact(P)
roots(det(P))
roots(det(gtild(F,'d'))) //The stable roots
roots(det(F)) //The antistable roots
clean(P-F*gtild(F,'d'))

//Example of continuous time use
s=poly(0,'s');
p=-3*(s+(1+i))*(s+(1-i))*(s+0.5)*(s-0.5)*(s-(1+i))*(s-(1-i));p=real(p);
//p(s) = polynomial in s^2, looks for stable f such that p=f(s)*f(-s)
w=horner(p,(1-s)/(1+s)); // bilinear transform w=p((1-s)/(1+s))
wn=numer(w); //take the numerator
fn=sfact(wn);f=numer(horner(fn,(1-s)/(s+1))); //Factor and back transform
f=f/sqrt(horner(f*gtild(f,'c'),0));f=f*sqrt(horner(p,0)); //normalization
roots(f) //f is stable
clean(f*gtild(f,'c')-p) //f(s)*f(-s) is p(s)
```

See Also

gtild, fspecg

Name

`simp` — rational simplification

```
[N1,D1]=simp(N,D)
H1=simp(H)
```

Parameters

N,D

real polynomials or real matrix polynomials

H

rational matrix (i.e matrix with entries n/d , n and d real polynomials)

Description

`[n1,d1]=simp(n,d)` calculates two polynomials $n1$ and $d1$ such that $n1/d1 = n/d$.

If N and D are polynomial matrices the calculation is performed element-wise.

`H1=simp(H)` is also valid (each entry of H is simplified in $H1$).

Caution:

-no threshold is given i.e. `simp` cannot forces a simplification.

-For linear dynamic systems which include integrator(s) simplification changes the static gain. ($H(0)$ for continuous systems or $H(1)$ for discrete systems)

-for complex data, `simp` returns its input(s).

-rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using `simp_mode` function.

Examples

```
s=poly(0,'s');
[n,d]=simp((s+1)*(s+2),(s+1)*(s-2))

simp_mode(%F);hns=s/s
simp_mode(%T);hns=s/s
```

See Also

`roots`, `trfmod`, `poly`, `clean`, `simp_mode`

Name

`simp_mode` — toggle rational simplification

```
mod=simp_mode()  
simp_mode(mod)
```

Parameters

`mod`
a boolean

Description

rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using `simp_mode` function.

`simp_mod(%t)` set rational simplification mode on

`simp_mod(%f)` set rational simplification mode off

`mod=simp_mod()` returns in `mod` the current rational simplification mode

Examples

```
s=poly(0, 's');  
mod=simp_mode()  
simp_mode(%f);hns=s/s  
simp_mode(%t);hns=s/s  
simp_mode(mod);
```

See Also

`simp`

Name

`sylv` — Sylvester matrix

```
[S]=sylv(a,b)
```

Parameters

`a,b`
two polynomials

`S`
matrix

Description

`sylv(a,b)` gives the Sylvester matrix associated to polynomials `a` and `b`, i.e. the matrix `S` such that:

$\text{coeff}(a*x + b*y)' = S * [\text{coeff}(x)'; \text{coeff}(y)']$.

Dimension of `S` is equal to `degree(a)+degree(b)`.

If `a` and `b` are coprime polynomials then

`rank(sylv(a,b))=degree(a)+degree(b)` and the instructions

```
u = sylv(a,b) \ eye(na+nb,1)
x = poly(u(1:nb), 'z', 'coeff')
y = poly(u(nb+1:na+nb), 'z', 'coeff')
```

compute Bezout factors `x` and `y` of minimal degree such that $a*x+b*y = 1$

Name

systmat — system matrix

```
[Sm]=systmat(Sl);
```

Parameters

Sl
linear system (syslin list) or descriptor system

Sm
matrix pencil

Description

System matrix of the linear system Sl (syslin list) in state-space form (utility function).

$$Sm = \begin{bmatrix} -sI + A & B \\ C & D \end{bmatrix}$$

For a descriptor system (Sl=list('des',A,B,C,D,E)), systmat returns:

$$Sm = \begin{bmatrix} -sE + A & B \\ C & D \end{bmatrix}$$

See Also

ss2des , sm2des , sm2ss

Part XXVIII. Signal Processing

Table of Contents

5. How to	1684
How to design an elliptic filter	1685

Chapter 5. How to

Name

How to design an elliptic filter — How to design an elliptic filter (analog and digital)

Description

The goal is to design a simple analog and digital elliptic filter.

Designing an analog elliptic filter

There are several possibilities to design an elliptic lowpass filter. We can use `analfp` or `zpell`. We will use `zpell` to produce the poles and zeros of the filter. Once we have got these poles and zeros, we will have to translate this representation into a `syslin` one.

And then, the filter can be represented in bode plot.

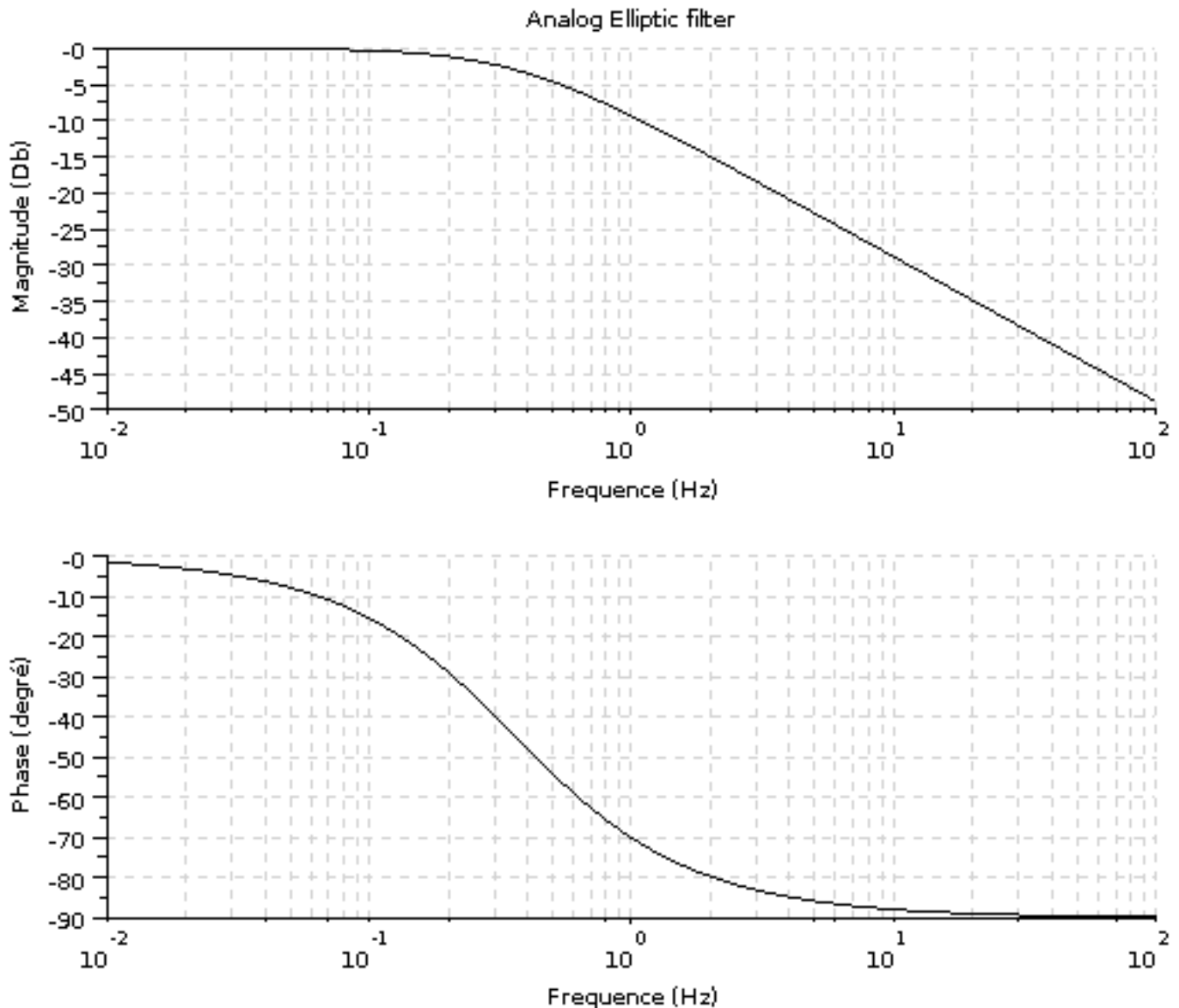
```
// analog elliptic (Bessel), order 2, cutoff 1 Hz
Epsilon = 3; // ripple of filter in pass band (0<epsilon<1)
A        = 60; // attenuation of filter in stop band (A<1)
OmegaC   = 10; // pass band cut-off frequency in Hertz
OmegaR   = 50; // stop band cut-off frequency in Hertz

// Generate the filter
[_zeros,pols,gain] = zpell(3,60,10,50);

// Generate the equivalent linear system of the filter
num  = gain * real(poly(_zeros,'s'));;
den  = real(poly(pols,'s'));
elatf = syslin('c',num,den);

// Plot the resulting filter
bode(elatf,0.01,100);
title('Analog Elliptic filter');
```

Bode plot is only suited for analog filters.



If you want to design a highpass, bandpass or bandstop filter, you can first design a lowpass and then transform this lowpass filter using the transform function.

Designing a digital elliptic filter

Now, let's focus on how to produce a digital lowpass elliptic filter.

We can produce two kinds of digital filters:

- an IIR (Infinite Impulse Response).

To compute such a filter, we can use the following functions:

- `iir`
- `eqiir`
- a FIR (Finite Impulse Response).

To compute such a filter, we can use the following functions:

- eqfir
- ffilt
- wfir
- fsfirlin

For our demonstration, we will use the iir function.

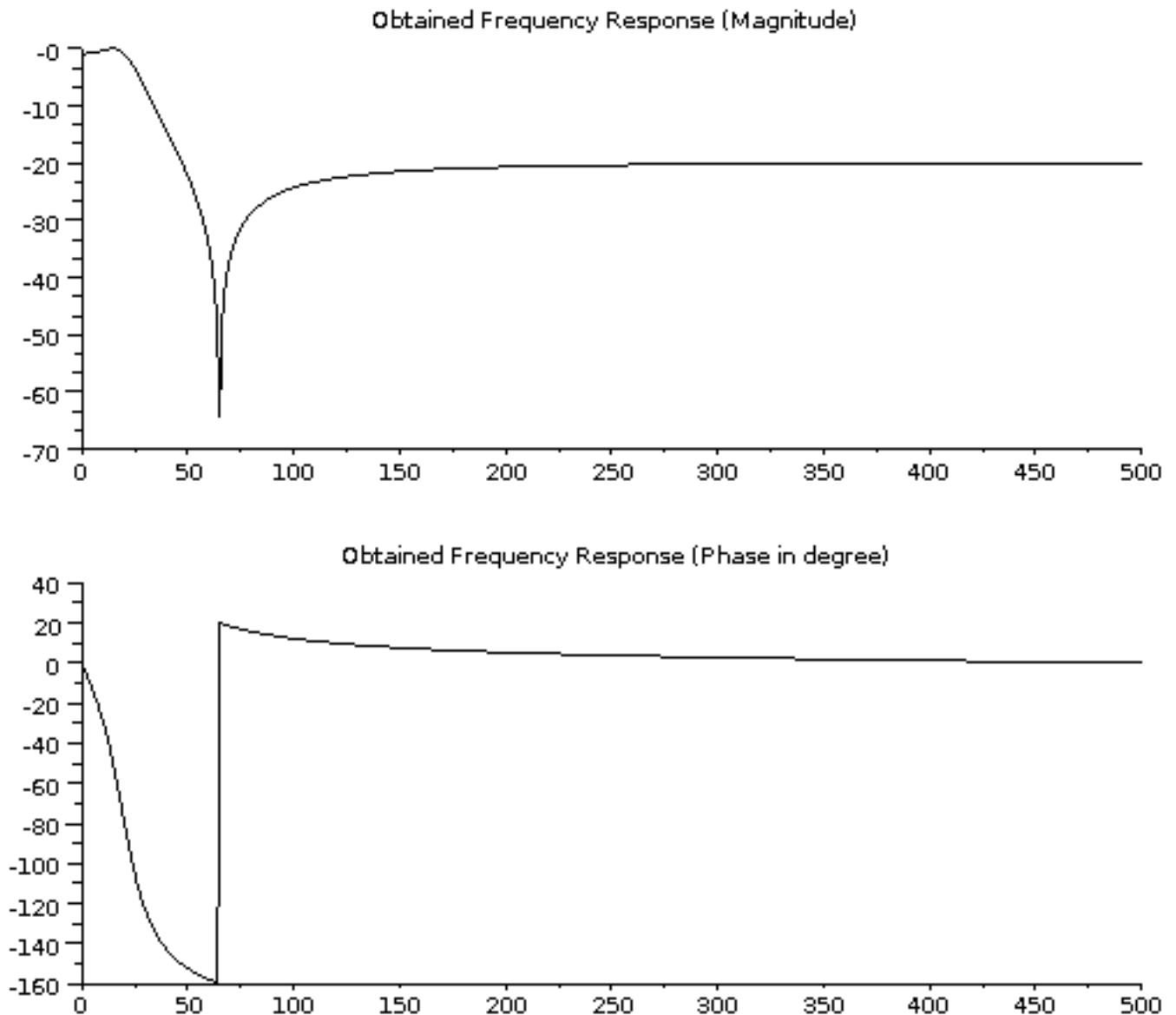
```
Order    = 2; // The order of the filter
Fs       = 1000; // The sampling frequency
Fcutoff  = 40; // The cutoff frequency

// We design a low pass elliptic filter
hz = iir(Order, 'lp', 'ellip', [Fcutoff/Fs/2 0], [0.1 0.1]);

// We compute the frequency response of the filter
[frq, repf] = repfreq(hz, 0:0.001:0.5);
[db_repf, phi_repf] = dbphi(repf);

// And plot the bode like representation of the digital filter
subplot(2,1,1);
plot2d(Fs*frq, db_repf);
xtitle('Obtained Frequency Response (Magnitude)');
subplot(2,1,2);
plot2d(Fs*frq, phi_repf);
xtitle('Obtained Frequency Response (Phase in degree)');
```

Here is the representation of the digital elliptic filter.



To represent the filter in phase and magnitude, we need first to convert the discrete impulse response into magnitude and phase using the `dbphi` function. This conversion is done using a set of normalized frequencies.

Filtering a signal using the digital filter

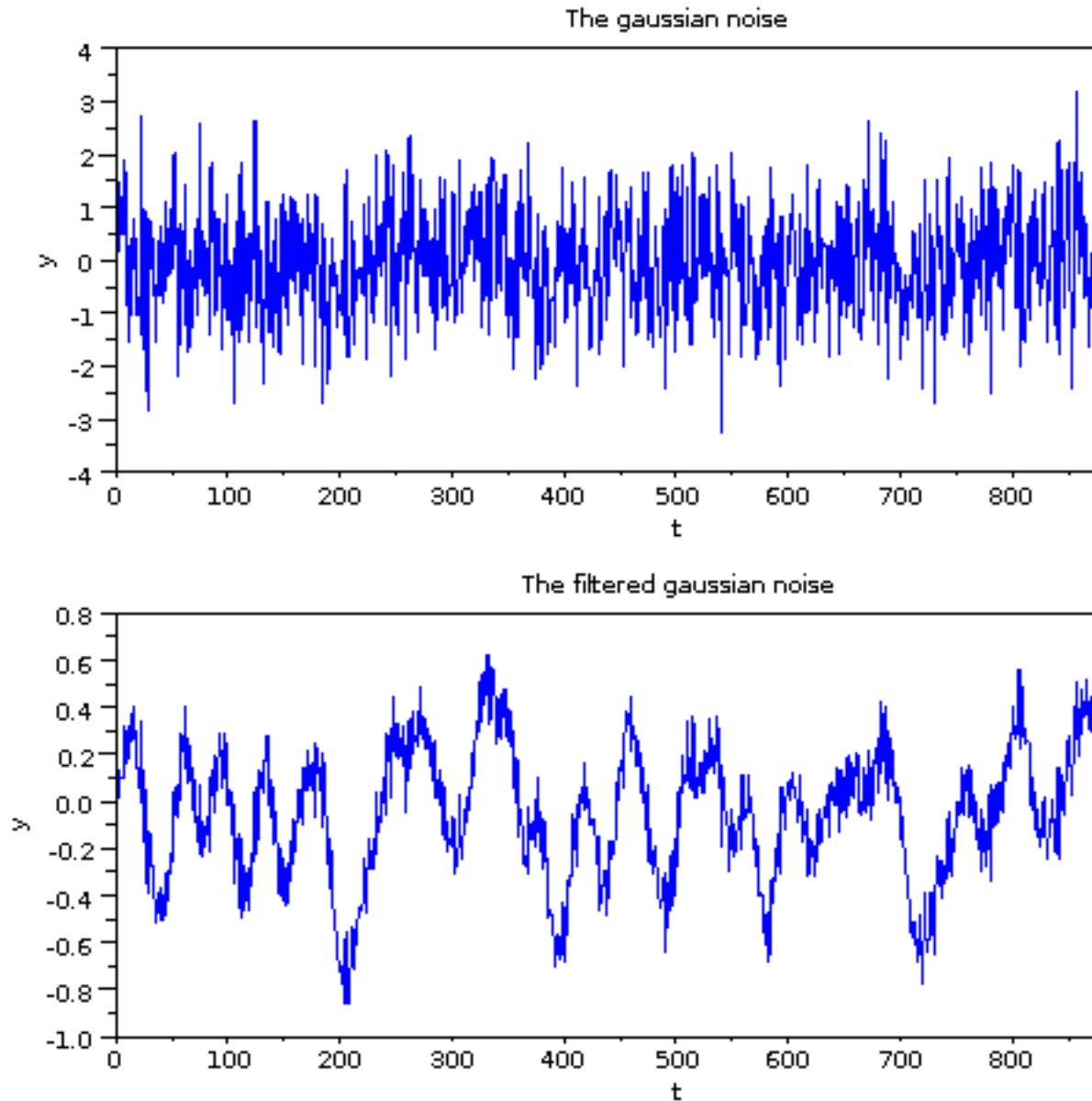
Designing a filter is a first step. Once done, this filter will be used to transform a signal. To get rid of some noise for example.

In the following examples, we will filter a gaussian noise.

```
rand('normal');  
Input = rand(1,1000); // Produce a random gaussian noise  
t      = 1:1000;  
  
sl= tf2ss(hz); // From transfert function to syslin representation  
y = flts(Input,sl); // Filter the signal
```

```
subplot(2,1,1);  
plot(t,Input);  
xtitle('The gaussian noise','t','y');  
subplot(2,1,2);  
plot(t,y);  
xtitle('The filtered gaussian noise','t','y');
```

Here is the representation of the signal before and after filtering.



As we can see in the result, the high frequencies of the noise have been removed and it remains only the low frequencies. The signal is still noisy, but it contains mainly low frequencies.

Filtering a signal using the analog filter

To filter a signal using an analog filter, we have two strategies:

- transform the analog filter into a discrete one using the `dscr` function
- apply the `csim` function to filter the signal

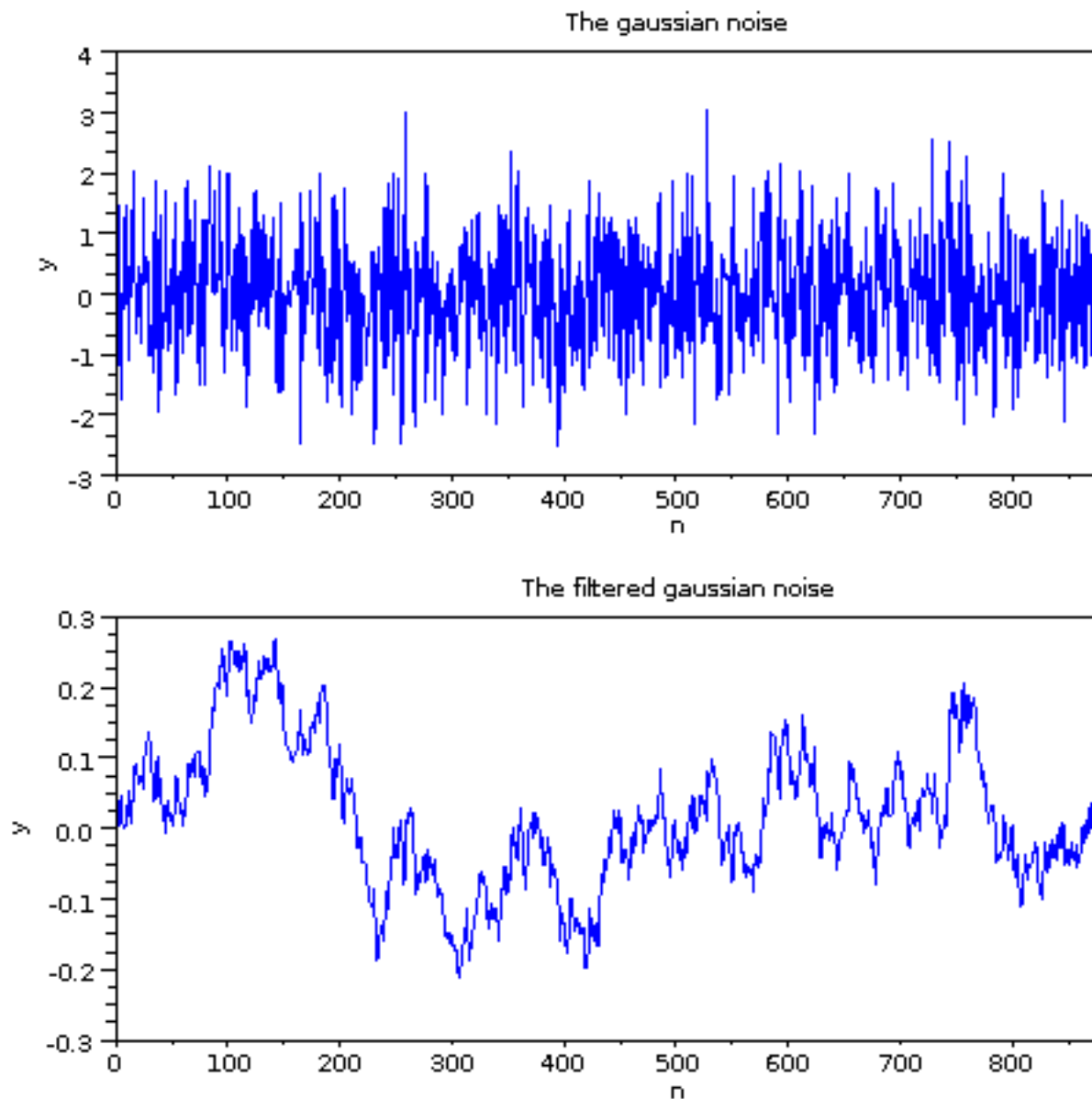
First, we try using the `dscr` + `flts` functions.

```
rand('normal');
Input = rand(1,1000); // Produce a random gaussian noise
n      = 1:1000; // The sample index

eldtf = dscr(elatf,1/100); // Discretization of the linear filter
y = flts(Input,eldtf); // Filter the signal

subplot(2,1,1);
plot(n,Input);
xtitle('The gaussian noise','n','y');
subplot(2,1,2);
plot(n,y);
xtitle('The filtered gaussian noise','n','y');
```

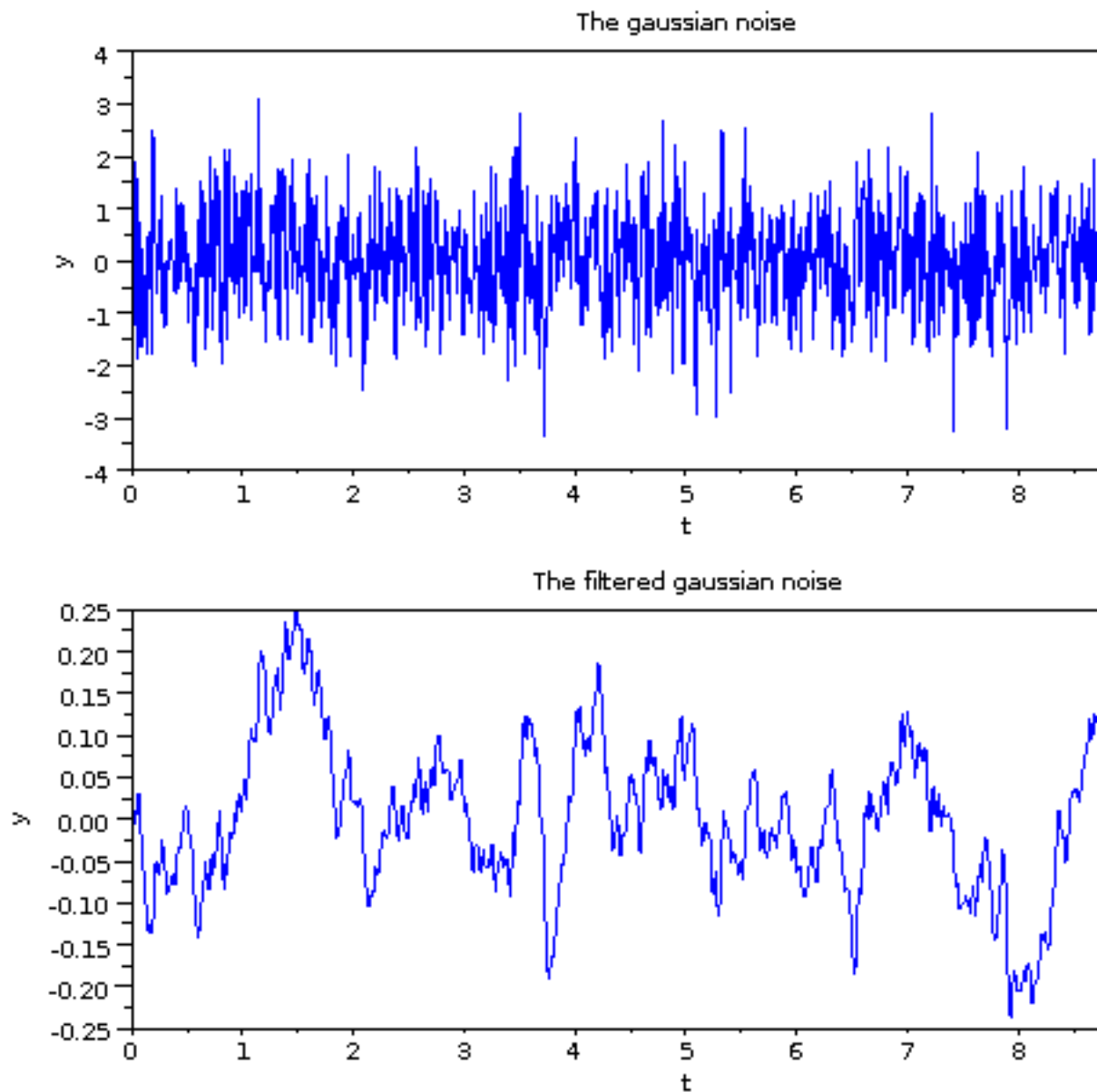
Here is the representation of the signal before and after filtering using the `dscr` + `flts` approach.



Next, we use the csim function.

```
rand('normal');  
Input = rand(1,1000); // Produce a random gaussian noise  
t      = 1:1000;  
t      = t*0.01; // Convert sample index into time steps  
  
y = csim(Input,t,elarf); // Filter the signal  
  
subplot(2,1,1);  
plot(t,Input);  
xtitle('The gaussian noise','t','y');  
subplot(2,1,2);  
plot(t,y);  
xtitle('The filtered gaussian noise','t','y');
```

Here is the representation of the signal before and after filtering using the csim approach.



The main difference between the dscr + flts approach and the csim approach: the dscr + flts uses samples whereas the csim functions uses time steps.

See Also

bode, iir, poly, syslin, zpell, flts, tf2ss, dscr, csim, trans, analpf

Name

Signal — Signal manual description

Filters

- analf
analog low-pass filter
- buttmag
squared magnitude response of a Butterworth filter
- casc
creates cascade realization of filter
- cheb1mag
square magnitude response of a type 1 Chebyshev filter
- cheb2mag
square magnitude response of a type 1 Chebyshev filter
- chepol
recursive implementation of Chebychev polynomial
- convol
convolution of 2 discrete series
- ell1 mag
squared magnitude of an elliptic filter
- eqfir
minimax multi-band, linear phase, FIR filter
- eqiir
design of iir filter
- faurre
optimal lqg filter.
- lindquis
optimal lqg filter lindquist algorithm
- ffilt
FIR low-pass,high-pass, band-pass, or stop-band filter
- filter
compute the filter model
- find_freq
parameter compatibility for elliptic filter design
- findm
for elliptic filter design
- frmag
magnitude of the frequency responses of FIR and IIR filters.
- fsfirlin
design of FIR, linear phase (frequency sampling technique)
- fwiiir
optimum design of IIR filters in cascade realization,

- iir
 - designs an iir digital filter using analog filter designs.
- iirgroup
 - group delay of iir filter
- iirlp
 - Lp IIR filters optimization
- group
 - calculate the group delay of a digital filter
- remezb
 - minimax approximation of a frequency domain magnitude response.
- kalm
 - Kalman update and error variance
- lev
 - resolve the Yule-Walker equations :
- levin
 - solve recursively Toeplitz system (normal equations)
- srfaur
 - square-root algorithm for the algebraic Riccati equation.
- srkf
 - square-root Kalman filter algorithm
- sskf
 - steady-state Kalman filter
- system
 - generates the next observation given the old state
- trans
 - transformation of standardized low-pass filter into low-pass, high-pass, band-pass, stop-band.
- wfir
 - linear-phase windowed FIR low-pass, band-pass, high-pass, stop-band
- wiener
 - Wiener estimate (forward-backward Kalman filter formulation)
- wigner
 - time-frequency wigner spectrum of a signal.
- window
 - calculate symmetric window
- zpbutt
 - Butterworth analog filter
- zpch1
 - poles of a type 1 Chebyshev analog filter
- zpch2
 - poles and zeros of a type 2 Chebyshev analog filter
- zpell
 - poles and zeros of prototype lowpass elliptic filter

Spectral estimation

- corr
 - correlation coefficients
- cspect
 - spectral estimation using the modified periodogram method.
- czf
 - chirp z-transform algorithm
- intdec
 - change the sampling rate of a 1D or 2D signal
- mese
 - calculate the maximum entropy spectral estimate
- pspect
 - auto and cross-spectral estimate
- wigner
 - Wigner-Ville time/frequency spectral estimation

Transforms

- dft
 - discrete Fourier transform
- fft
 - fast flourier transform
- hilb
 - Hilbert transform centred around the origin.
- hank
 - hankel matrix of the covariance sequence of a vector process
- mfft
 - fft for a multi-dimensional signal

Identification

- latn, lattp
 - recursive solution of normal equations
- phc
 - State space realisation by the principal hankel component approximation method,
- rpem
 - identification by the recursive prediction error method

Miscellaneous

- sinc
 - calculate the function $\sin(2\pi f t)/(\pi t)$
- sincd
 - calculates the function $\text{Sin}(N x)/\text{Sin}(x)$

%k
Jacobi's complete elliptic integral

%asn
.TP the elliptic integral :

%sn
Jacobi 's elliptic function with parameter m

bilt
bilinear transform or biquadratic transform.

jmat
permutes block rows or block columns of a matrix

Name

analpf — create analog low-pass filter

```
[hs,pols,zers,gain]=analpf(n,fdesign,rp,omega)
```

Parameters

n
positive integer : filter order

fdesign
string : filter design method : 'butt' or 'cheb1' or 'cheb2' or 'ellip'

rp
2-vector of error values for cheb1, cheb2 and ellip filters where only `rp(1)` is used for cheb1 case, only `rp(2)` is used for cheb2 case, and `rp(1)` and `rp(2)` are both used for ellip case.
 $0 < rp(1), rp(2) < 1$

- for cheb1 filters $1 - rp(1) < ripple < 1$ in passband
- for cheb2 filters $0 < ripple < rp(2)$ in stopband
- for ellip filters $1 - rp(1) < ripple < 1$ in passband $0 < ripple < rp(2)$ in stopband

omega
cut-off frequency of low-pass filter in Hertz

hs
rational polynomial transfer function

pols
poles of transfer function

zers
zeros of transfer function

gain
gain of transfer function

Description

Creates analog low-pass filter with cut-off frequency at omega.

```
hs=gain*poly(zers,'s')/poly(pols,'s')
```

Examples

```
//Evaluate magnitude response of continuous-time system
hs=analpf(4,'cheb1',[.1 0],5)
fr=0:.1:15;
hf=freq(hs(2),hs(3),%i*fr);
hm=abs(hf);
plot(fr,hm)
```



Authors

C. B.

Name

bilt — bilinear or biquadratic transform SISO system given by a zero/poles representation

```
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)
```

Parameters

- pl**
a vector, the poles of the given system.
- zr**
a vector, the zeros of the given system.
- num**
a polynomial with degree equal to the degree of **den**, the numerator of the transform.
- den**
a polynomial with degree 1 or 2, the denominator of the transform.
- npl**
a vector, the poles of the transformed system.
- nzr**
a vector, the zeros of the transformed system.
- ngn**
a scalar, the gain of the transformed system.

Description

function for calculating the gain poles and zeros which result from a bilinear transform or from a biquadratic transform. Used by the functions iir and trans.

Examples

```
Hlp=iir(3,'lp','ellip',[0.1 0],[.08 .03]);
pl=roots(Hlp.den);
zr=roots(Hlp.num);
gn=coeff(Hlp.num,degree(Hlp.num))/coeff(Hlp.den,degree(Hlp.den));
z=poly(0,'z');
a=0.3;
num=z-a;
den=1-a*z;
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)

Hlpt=ngn*poly(nzr,'z','r')/poly(npl,'z','r')

//comparison with horner
horner(Hlp,num/den)
```

Authors

Carey Bunks ;

See Also

iir, trans, horner

Name

buttmag — response of Butterworth filter

```
[h]=buttmag(order,omegac,sample)
```

Parameters

order

integer : filter order

omegac

real : cut-off frequency in Hertz

sample

vector of frequency where `buttmag` is evaluated

h

Butterworth filter values at sample points

Description

squared magnitude response of a Butterworth filter `omegac` = cutoff frequency ; `sample` = sample of frequencies

Examples

```
//squared magnitude response of Butterworth filter
h=buttmag(13,300,1:1000);
mag=20*log(h)/log(10);
plot2d((1:1000)',mag,[2],"011"," ",[0,-180,1000,20])
```

Authors

F. D.

Name

casc — cascade realization of filter from coefficients

```
[cels]=casc(x,z)
```

Parameters

x
(4xN)-matrix where each column is a cascade element, the first two column entries being the numerator coefficients and the second two column entries being the denominator coefficients

z
string representing the cascade variable

cels
resulting cascade representation

Description

Creates cascade realization of filter from a matrix of coefficients (utility function).

Examples

```
x=[1,2,3;4,5,6;7,8,9;10,11,12]  
cels=casc(x,'z')
```

Name

cepstrum — cepstrum calculation

```
fresp = cepstrum(w,mag)
```

Parameters

w
positive real vector of frequencies (rad/sec)

mag
real vector of magnitudes (same size as **w**)

fresp
complex vector

Description

`fresp = cepstrum(w,mag)` returns a frequency response `fresp(i)` whose magnitude at frequency `w(i)` equals `mag(i)` and such that the phase of `fresp` corresponds to a stable and minimum phase system. `w` needs not to be sorted, but minimal entry should not be close to zero and all the entries of `w` should be different.

Examples

```
w=0.1:0.1:5;mag=1+abs(sin(w));  
fresp=cepstrum(w,mag);  
plot2d([w',w'],[mag(:),abs(fresp)])
```

See Also

`frfit`

Name

cheb1mag — response of Chebyshev type 1 filter

```
[h2]=cheb1mag(n,omegac,epsilon,sample)
```

Parameters

n
integer : filter order

omegac
real : cut-off frequency

epsilon
real : ripple in pass band

sample
vector of frequencies where `cheb1mag` is evaluated

h2
Chebyshev I filter values at sample points

Description

Square magnitude response of a type 1 Chebyshev filter.

omegac=passband edge.

epsilon such that $1/(1+\epsilon^2)$ =passband ripple.

sample vector of frequencies where the square magnitude is desired.

Examples

```
//Chebyshev; ripple in the passband
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;
h=cheb1mag(n,omegac,epsilon,sample);
plot2d(sample,h)
xtitle('','frequencies','magnitude')
```

See Also

buttmag

Name

cheb2mag — response of type 2 Chebyshev filter

```
[h2]=cheb2mag(n,omegar,A,sample)
```

Parameters

n
integer ; filter order

omegar
real scalar : cut-off frequency

A
attenuation in stop band

sample
vector of frequencies where cheb2mag is evaluated

h2
vector of Chebyshev II filter values at sample points

Description

Square magnitude response of a type 2 Chebyshev filter.

omegar = stopband edge, sample = vector of frequencies where the square magnitude h2 is desired.

Examples

```
//Chebyshev; ripple in the stopband
n=10;omegar=6;A=1/0.2;sample=0.0001:0.05:10;
h2=cheb2mag(n,omegar,A,sample);
plot(sample,log(h2)/log(10),'frequencies','magnitude in dB')

//Plotting of frequency edges
minval=(-maxi(-log(h2)))/log(10);
plot2d([omegar;omegar],[minval;0],[2],"000");

//Computation of the attenuation in dB at the stopband edge
attenuation=-log(A*A)/log(10);
plot2d(sample',attenuation*ones(sample)',[5],"000")
```

See Also

cheb1mag

Name

chepol — Chebychev polynomial

```
[Tn]=chepol(n,var)
```

Parameters

n
integer : polynomial order

var
string : polynomial variable

Tn
polynomial in the variable **var**

Description

Recursive implementation of Chebychev polynomial.
 $T_n = 2 \cdot \text{poly}(0, \text{var}) \cdot \text{chepol}(n-1, \text{var}) - \text{chepol}(n-2, \text{var})$ with $T_0 = 1$ and $T_1 = \text{poly}(0, \text{var})$.

Examples

```
chepol(4, 'x')
```

Authors

F. D.

Name

convol — convolution

```
[y]=convol(h,x)
[y,e1]=convol(h,x,e0)
```

Parameters

- h**
a vector, first input sequence ("short" one)
- x**
a vector, second input sequence ("long" one)
- e0**
a vector,old tail to overlap add (not used in first call)
- y**
a vector, the convolution.
- e1**
new tail to overlap add (not used in last call)

Description

Calculates the convolution $y = h * x$ of two discrete sequences by using the fft. The convolution is defined as follow:

$$y_k = \sum_j h_j * x_{k+1-j}$$

Overlap add method can be used.

USE OF OVERLAP ADD METHOD: For $x=[x_1, x_2, \dots, x_{Nm1}, x_N]$ First call is $[y_1, e_1]=\text{convol}(h, x_1)$; Subsequent calls : $[y_k, e_k]=\text{convol}(h, x_k, e_{k-1})$; Final call : $[y_N]=\text{convol}(h, x_N, e_{Nm1})$; Finally $y=[y_1, y_2, \dots, y_{Nm1}, y_N]$.

The algorithm based on the convolution definition is implemented for polynomial product: $y=\text{convol}(h, x)$ is equivalent to $y=\text{coeff}(\text{poly}(h, 'z', 'c')*\text{poly}(x, 'z', 'c'))$ but much more efficient if x is a "long" array.

Examples

```
x=1:3;
h1=[1,0,0,0,0];h2=[0,1,0,0,0];h3=[0,0,1,0,0];
x1=convol(h1,x),x2=convol(h2,x),x3=convol(h3,x),
convol(h1+h2+h3,x)
p1=poly(x,'x','coeff')
p2=poly(h1+h2+h3,'x','coeff')
p1*p2
```

See Also

corr, fft, pspect

Authors

F. D , C. Bunks Date 3 Oct. 1988; ;

Name

corr — correlation, covariance

```
[cov,Mean]=corr(x,[y],nlags)
[cov,Mean]=corr('fft',xmacro,[ymacro],n,sect)

[w,xu]=corr('updt',x1,[y1],w0)
[w,xu]=corr('updt',x2,[y2],w,xu)
...
[wk]=corr('updt',xk,[yk],w,xu)
```

Parameters

x
a real vector

y
a real vector, default value x.

nlags
integer, number of correlation coefficients desired.

xmacro
a scilab external (see below).

ymacro
a scilab external (see below), default value xmacro

n
an integer, total size of the sequence (see below).

sect
size of sections of the sequence (see below).

xi
a real vector

yi
a real vector,default value xi.

cov
real vector, the correlation coefficients

Mean
real number or vector, the mean of x and if given y

Description

Computes

```
      n - m
      ====
      \
cov(m) = > (x(k) - xmean) (y(m+k) - ymean) * ---
      /
      ====
      k = 1
                                     1
                                     ---
                                     n
```

for $m=0,\dots,n\text{lag}-1$ and two vectors $x=[x(1), \dots, x(n)]$ $y=[y(1), \dots, y(n)]$

Note that if x and y sequences are different $\text{corr}(x,y,\dots)$ is different with $\text{corr}(y,x,\dots)$

Short sequences

`[cov, Mean]=corr(x, [y], nlags)` returns the first `nlags` correlation coefficients and `Mean = mean(x)` (mean of $[x, y]$ if y is an argument). The sequence x (resp. y) is assumed real, and x and y are of same dimension n .

Long sequences

`[cov, Mean]=corr('fft', xmacro, [ymacro], n, sect)` Here `xmacro` is either

- a function of type `[xx]=xmacro(sect, istart)` which returns a vector `xx` of dimension `nsect` containing the part of the sequence with indices from `istart` to `istart+sect-1`.
- a fortran subroutine or C procedure which performs the same calculation. (See the source code of `dgetx` for an example). `n` = total size of the sequence. `sect` = size of sections of the sequence. `sect` must be a power of 2. `cov` has dimension `sect`. Calculation is performed by FFT.

Updating method

```
w0,xu]=corr('updt', x1, [y1], w0)
[w, xu]=corr('updt', x2, [y2], w, xu)
...
wk=corr('updt', xk, [yk], w, xu)
```

With this calling sequence the calculation is updated at each call to `corr`.

```
w0 = 0*ones(1, 2*nlags);
nlags = power of 2.
```

$x_1, x_2, \dots$ are parts of x such that $x=[x_1, x_2, \dots]$ and sizes of x_i a power of 2. To get `nlags` coefficients a final fft must be performed `c=fft(w, 1)/n`; `cov=c(1:nlags)` (n is the size of x (y)). Caution: this calling sequence assumes that `xmean = ymean = 0`.

Examples

```
x=%pi/10:%pi/10:102.4*pi;
rand('seed'); rand('normal');
y=[.8*sin(x)+.8*sin(2*x)+rand(x); .8*sin(x)+.8*sin(1.99*x)+rand(x)];
c=[];
for j=1:2, for k=1:2, c=[c; corr(y(k,:), y(j,:), 64)]; end; end;
c=matrix(c, 2, 128); cov=[];
for j=1:64, cov=[cov; c(:, (j-1)*2+1:2*j)]; end;
rand('unif')

rand('normal'); x=rand(1, 256); y=-x;
deff(' [z]=xx(inc, is)', ' z=x(is:is+inc-1) ');
deff(' [z]=yy(inc, is)', ' z=y(is:is+inc-1) ');
[c, mxy]=corr(x, y, 32);
x=x-mxy(1)*ones(x); y=y-mxy(2)*ones(y); //centring
c1=corr(x, y, 32); c2=corr(x, 32);
```

```

norm(c1+c2,1)
[c3,m3]=corr('fft',xx,yy,256,32);
norm(c1-c3,1)
[c4,m4]=corr('fft',xx,256,32);
norm(m3,1),norm(m4,1)
norm(c3-c1,1),norm(c4-c2,1)
x1=x(1:128);x2=x(129:256);
y1=y(1:128);y2=y(129:256);
w0=0*ones(1:64); //32 coeffs
[w1,xu]=corr('u',x1,y1,w0);w2=corr('u',x2,y2,w1,xu);
zz=real(fft(w2,1))/256;c5=zz(1:32);
norm(c5-c1,1)
[w1,xu]=corr('u',x1,w0);w2=corr('u',x2,w1,xu);
zz=real(fft(w2,1))/256;c6=zz(1:32);
norm(c6-c2,1)
rand('unif')

// test for Fortran or C external
//
deff('[y]=xmacro(sec,ist)','y=sin(ist:(ist+sec-1))');
x=xmacro(100,1);
[cc1,mm1]=corr(x,2^3);
[cc,mm]=corr('fft',xmacro,100,2^3);
[cc2,mm2]=corr('fft','corexx',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2)))]

deff('[y]=ymacro(sec,ist)','y=cos(ist:(ist+sec-1))');
y=ymacro(100,1);
[cc1,mm1]=corr(x,y,2^3);
[cc,mm]=corr('fft',xmacro,ymacro,100,2^3);
[cc2,mm2]=corr('fft','corexx','corexy',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2)))]

```

See Also

fft

Name

cspect — two sided cross-spectral estimate between 2 discrete time signals using the correlation method

```
[sm [,cwp]]=cspect(nlags,npoints,wtype,x [,y] [,wpar])  
[sm [,cwp]]=cspect(nlags,npoints,wtype,nx [,ny] [,wpar])
```

Parameters

- x**
vector, the data of the first signal.
- y**
vector, the data of the second signal. If y is omitted it is supposed to be equal to x (auto-correlation). If it is present, it must have the same number of element than x .
- nx**
a scalar : the number of points in the x signal. In this case the segments of the x signal are loaded by a user defined function named `getx` (see below).
- ny**
a scalar : the number of points in the y signal. In this case the segments of the y signal are loaded by a user defined function named `gety` (see below). If present ny must be equal to nx.
- nlags**
number of correlation lags (positive integer)
- npoints**
number of transform points (positive integer)
- wtype**
The window type
- 're': rectangular
 - 'tr': triangular
 - 'hm': Hamming
 - 'hn': Hanning
 - 'kr': Kaiser, in this case the wpar argument must be given
 - 'ch': Chebyshev, in this case the wpar argument must be given
- wpar**
optional parameters for Kaiser and Chebyshev windows:
- 'kr': wpar must be a strictly positive number
 - 'ch': wpar must be a 2 element vector [main_lobe_width, side_lobe_height] with $0 < \text{main\_lobe\_width} < .5$, and $\text{side\_lobe\_height} > 0$
- sm**
The power spectral estimate in the interval $[0, 1]$ of the normalized frequencies. It is a row array of size npoints. The array is real in case of auto-correlation and complex in case of cross-correlation.

cwp

the unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

Description

Computes the cross-spectrum estimate of two signals x and y if both are given and the auto-spectral estimate of x otherwise. Spectral estimate obtained using the correlation method.

The cross spectrum of two signal x and y is defined to be

$$S_{xy}(\omega) = \frac{1}{N} \left(\sum_{n=0}^{N-1} x(n) e^{-i\omega n} \right) \left(\sum_{n=0}^{N-1} \bar{y}(n) e^{i\omega n} \right)$$

The correlation method calculates the spectral estimate as the Fourier transform of a modified estimate of the auto/cross correlation function. This auto/cross correlation modified estimate consist of repeatedly calculating estimates of the autocorrelation function from overlapping sub-segments if the data, and then averaging these estimates to obtain the result.

The number of points of the window is $2 * n_{lags} - 1$.

For batch processing, the x and y data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

$x_k = \text{getx}(ns, \text{offset})$ and $y_k = \text{gety}(ns, \text{offset})$ where ns is the segment size and offset is the index of the first element of the segment in the full signal.

Warning

For Scilab version up to 5.0.2 the returned value was the modulus of the current one.

Reference

Oppenheim, A.V., and R.W. Schaffer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);
//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);
//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1); xf=fft(x1,-1);yf=hf.*xf;y=real(fft(yf,1));
sm=cspect(100,200,'tr',y);
smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm))
```

See Also

pspect, mese, corr

Authors

C. Bunks INRIA

Name

czt — chirp z-transform algorithm

```
[czx]=czt(x,m,w,phi,a,theta)
```

Parameters

x
input data sequence

m
czt is evaluated at m points in z -plane

w
magnitude multiplier

phi
phase increment

a
initial magnitude

theta
initial phase

czx
chirp z-transform output

Description

chirp z-transform algorithm which calculates the z -transform on a spiral in the z -plane at the points $[a \cdot \exp(j \cdot \theta)] [w^k \exp(j \cdot k \cdot \phi)]$ for $k=0, 1, \dots, m-1$.

Examples

```
a=.7*exp(%i*%pi/6);
[ffr,bds]=xgetech(); //preserve current context
rect=[-1.2,-1.2*sqrt(2),1.2,1.2*sqrt(2)];
t=2*%pi*(0:179)/179;xsetech([0,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[3],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=.93*exp(-%i*%pi/15);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech([0.5,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[-1],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=w0/(.93*.93);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech(ffr,bds); //restore context
```

Authors

C. Bunks

Name

detrend — remove constant, linear or piecewise linear trend from a vector

```
y = detrend(x)
y = detrend(x, flag)
y = detrend(x, flag, bp)
```

Parameters

x
vector or matrix of real or complex numbers (the signal to treat)

flag
a string equal to "linear" (or "l") for linear or piecewise linear treatment or "constant" (or "c") for constant treatment.

bp
the breakpoints to provide if you want a piecewise linear treatment.

y
output, the signal x with the trend removed from it.

Description

This function removes the constant or linear or piecewise linear trend from a vector **x**. In general this can be useful before a fourier analysis. If **x** is matrix this function removes the trend of each column of **x**.

When **flag** = "constant" or "c" **detrend** removes the constant trend (simply the mean of the signal) and when **flag** = "linear" or "l" the function removes the linear trend. By adding a third argument **bp** it is possible to remove a continuous *piecewise linear* trend. Note that the "instants" of the signal **x** goes from 0 to m-1 (m = length(x) if **x** is a vector and m = size(x,1) in case **x** is a matrix). So the breakpoints **bp(i)** must be reals in $[0\ m-1]$ (breakpoints outside are simply removed from **bp** vector).

The trend is got by a least square fit of **x** on the appropriate function space.

Examples

```
// example #1
t = linspace(0,16*pi,1000)';
x = -20 + t + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x);
clf()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()

// example #2
t = linspace(0,32*pi,2000)';
x = abs(t-16*pi) + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x,"linear",1000);
clf()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()
```



Authors

Bruno Pincon

Name

dft — discrete Fourier transform

```
[xf]=dft(x,flag);
```

Parameters

x
input vector

flag
indicates dft (flag=-1) or idft (flag=1)

xf
output vector

Description

Function which computes dft of vector x.

Examples

```
n=8;omega = exp(-2*pi*i/n);  
j=0:n-1;F=omega.^(j'*j); //Fourier matrix  
x=1:8;x=x(:);  
F*x  
fft(x,-1)  
dft(x,-1)  
inv(F)*x  
fft(x,1)  
dft(x,1)
```

See Also

fft

Authors

C. B.

Name

ell1mag — magnitude of elliptic filter

```
[v]=ell1mag(eps,m1,z)
```

Parameters

eps
passband ripple= $1/(1+\text{eps}^2)$

m1
stopband ripple= $1/(1+(\text{eps}^2)/m1)$

z
sample vector of values in the complex plane

v
elliptic filter values at sample points

Description

Function used for squared magnitude of an elliptic filter. Usually $m1=\text{eps}*\text{eps}/(a*a-1)$. Returns $v=\text{real}(\text{ones}(z)./(\text{ones}(z)+\text{eps}*\text{eps}*s.*s))$ for $s=\text{sn}(z,m1)$.

Examples

```
deff('[alpha,Beta]=alpha_beta(n,m,m1)',...
'if 2*int(n/2)==n then, BeTa=K1; else, BeTa=0;end;...
alpha=%k(1-m1)/%k(1-m);')
epsilon=0.1;A=10; //ripple parameters
m1=(epsilon*epsilon)/(A*A-1);n=5;omegac=6;
m=find_freq(epsilon,A,n);omegar = omegac/sqrt(m)
%k(1-m1)*%k(m)/(%k(m1)*%k(1-m))-n //Check...
[alpha,Beta]=alpha_beta(n,m,m1)
alpha*%asn(1,m)-n*%k(m1) //Check
sample=0:0.01:20;
//Now we map the positive real axis into the contour...
z=alpha*%asn(sample/omegac,m)+Beta*ones(sample);
plot(sample,ell1mag(epsilon,m1,z))
```

See Also

buttmag

Name

eqfir — minimax approximation of FIR filter

```
[hn]=eqfir(nf,bedge,des,wate)
```

Parameters

nf
number of output filter points desired

bedge
Mx2 matrix giving a pair of edges for each band

des
M-vector giving desired magnitude for each band

wate
M-vector giving relative weight of error in each band

hn
output of linear-phase FIR filter coefficients

Description

Minimax approximation of multi-band, linear phase, FIR filter

Examples

```
hn=eqfir(33,[0 .2;.25 .35;.4 .5],[0 1 0],[1 1 1]);  
[hm,fr]=frmag(hn,256);  
plot(fr,hm),
```

Authors

C. B.

Name

eqiir — Design of iir filters

```
[cells, fact, zzeros, zpoles]=eqiir(ftype, approx, om, deltap, deltas)
```

Parameters

ftype

filter type ('lp', 'hp', 'sb', 'bp')

approx

design approximation ('butt', 'cheb1', 'cheb2', 'ellip')

om

4-vector of cutoff frequencies (in radians) $om=[om1, om2, om3, om4]$, $0 \leq om1 \leq om2 \leq om3 \leq om4 \leq \pi$. When ftype='lp' or 'hp', om3 and om4 are not used and may be set to 0.

deltap

ripple in the passband. $0 \leq \text{deltap} \leq 1$

deltas

ripple in the stopband. $0 \leq \text{deltas} \leq 1$

cells

realization of the filter as second order cells

fact

normalization constant

zzeros

zeros in the z-domain

zpoles

poles in the z-domain

Description

Design of iir filter based on syredi.

The filter obtained is $h(z)=\text{fact}*\text{product of the elements of cells}$.

That is $hz=\text{fact}*\text{prod}(\text{cells.num})./\text{prod}(\text{cells.den})$.

Examples

```
[cells, fact, zzeros, zpoles]=eqiir('lp', 'ellip', [2*pi/10, 4*pi/10], 0.02, 0.001)
h=fact*poly(zzeros, 'z')/poly(zpoles, 'z')
```

See Also

eqfir, iir, syredi

Name

faurre — filter computation by simple Faurre algorithm

```
[P,R,T]=faurre(n,H,F,G,R0)
```

Parameters

n
number of iterations.

H, F, G
estimated triple from the covariance sequence of y.

R0
 $E(y_k * y_k')$

P
solution of the Riccati equation after n iterations.

R, T
gain matrix of the filter.

Description

This function computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices R and T of the filter model. The algorithm tries to compute the solution P as the growing limit of a sequence of matrices P_n such that

$$P_{n+1} = F * P_n * F' + (G - F * P_n * h') * (R_0 - H * P_n * H')^{-1} * (G' - H * P_n * F') \\ P_0 = G * R_0^{-1} * G'$$

Note that this method may not converge, especially when F has poles near the unit circle. Use preferably the srfaur function.

See Also

srfaur , lindquist , phc

Authors

G. Le V.

Name

ffilt — coefficients of FIR low-pass

```
[x]=ffilt(ft,n,fl,fh)
```

Parameters

ft
filter type where **ft** can take the values

"lp"
for low-pass filter

"hp"
for high-pass filter

"bp"
for band-pass filter

"sb"
for stop-band filter

n
integer (number of filter samples desired)

fl
real (low frequency cut-off)

fh
real (high frequency cut-off)

x
vector of filter coefficients

Description

Get **n** coefficients of a FIR low-pass, high-pass, band-pass, or stop-band filter. For low and high-pass filters one cut-off frequency must be specified whose value is given in **fl**. For band-pass and stop-band filters two cut-off frequencies must be specified for which the lower value is in **fl** and the higher value is in **fh**

Authors

C. B.

Name

`fft` — fast Fourier transform.

`ifft` — fast Fourier transform.

```
x=fft(a, -1) or x=fft(a)
x=fft(a, 1) or x=ifft(a)
x=fft(a, -1, dim, incr)
x=fft(a, 1, dim, incr)
```

Parameters

`x`
real or complex vector. Real or complex matrix (2-dim `fft`)

`a`
real or complex vector, matrix or multidimensionnal array.

`dim`
integer

`incr`
integer

Description

Short syntax

direct
`x=fft(a, -1)` or `x=fft(a)` gives a direct transform.

single variate
If `a` is a vector a single variate direct FFT is computed that is:
$$x(k) = \sum_{m=1}^n a(m) \exp(-2i\pi(m-1)(k-1)/n)$$
for `k` varying from 1 to `n` (`n`=size of vector `a`).

(the `-1` argument refers to the sign of the exponent..., NOT to "inverse"),

multivariate
If `a` is a matrix or a multidimensionnal array a multivariate direct FFT is performed.

inverse
`a=fft(x, 1)` or `a=ifft(x)` performs the inverse transform normalized by `1/n`.

single variate
If `a` is a vector a single variate inverse FFT is computed

multivariate
If `a` is a matrix or a multidimensionnal array a multivariate inverse FFT is performed.

Long syntax for multidimensional FFT

`x=fft(a, -1, dim, incr)` allows to perform an multidimensional `fft`.

If `a` is a real or complex vector implicitly indexed by `j1, j2, ..., jp` i.e. `a(j1, j2, ..., jp)` where `j1` lies in `1:dim(1)`, `j2` in `1:dim(2)`, ... one gets a `p`-variate FFT by calling `p` times `fft` as follows

```

incrk=1; x=a;
for k=1:p
    x=fft(x , -1,dim(k),incrk)
    incrk=incrk*dim(k)
end

```

where `dimk` is the dimension of the current variable w.r.t which one is integrating and `incrk` is the increment which separates two successive `j`k elements in `a`.

In particular, if `a` is an `m`x`n` matrix, `x=fft(a, -1)` is equivalent to the two instructions:

`a1=fft(a, -1,m,1)` and `x=fft(a1, -1,n,m)`.

Examples

```

//Comparison with explicit formula
//-----
a=[1;2;3];n=size(a, ' ');
norm(1/n*exp(2*i*pi*(0:n-1)'.*(0:n-1)/n)*a -fft(a,1))
norm(exp(-2*i*pi*(0:n-1)'.*(0:n-1)/n)*a -fft(a, -1))

//Frequency components of a signal
//-----
// build a noides signal sampled at 1000hz containing to pure frequencies
// at 50 and 70 Hz
sample_rate=1000;
t = 0:1/sample_rate:0.6;
N=size(t, ' '); //number of samples
s=sin(2*pi*50*t)+sin(2*pi*70*t+pi/4)+grand(1,N, 'nor', 0,1);

y=fft(s);
//the fft response is symmetric we retain only the first N/2 points
f=sample_rate*(0:(N/2))/N; //associated frequency vector
n=size(f, ' ')
clf()
plot2d(f,abs(y(1:n)))

```

See Also

[corr](#)

Name

fft2 — two-dimension fast Fourier transform

```
y=fft2(x)
y=fft2(x,n,m)
```

Parameters

x
a vector/matrix/array (Real or Complex)

y
a vector/matrix/array (Real or Complex)

m
integer, number of rows.

n
integer, number of columns.

Description

This functions performs the two-dimension discrete Fourier transform.

`y=fft2(x)` y and x have the same size

`y=fft2(x,m,n)`: If m (respectively n) is less than the rows number (respectively columns) of x then the x rows number (resp. columns) is truncated, else if m (resp. n) is more than the rows number (resp. columns) of x then x rows are completed by zero (resp. columns) .

if x is a matrix then y is a matrix, if x is a hypermatrix then y is a hypermatrix, with the size of the first dimension of y is equal to m, the size of the second dimension of y is equal to n, the size of the ith dimension of y (for i>2, case hypermatrix) equal to the size of the ith dimension of x. (i.e size(y,1)=m, size(y,2)=n and size(y,i)=size(x,i) for i>2)

Examples

```
//Comparison with explicit formula
a=[1 2 3 ;4 5 6 ;7 8 9 ;10 11 12]
m=size(a,1)
n=size(a,2)

// fourier transform along the rows
for i=1:n
a1(:,i)=exp(-2*i*pi*(0:m-1)'.*(0:m-1)/m)*a(:,i)
end

// fourier transform along the columns
for j=1:m
a2temp=exp(-2*i*pi*(0:n-1)'.*(0:n-1)/n)*(a1(j,:)).'
a2(j,:)=a2temp.'
end
norm(a2-fft2(a))
```

See Also

fft

Name

fftshift — rearranges the fft output, moving the zero frequency to the center of the spectrum

```
y=fftshift(x [, job])
```

Parameters

x
real or complex vector or matrix.

y
real or complex vector or matrix.

job
integer, dimension selection, or string 'all'

Description

if **x** results of an fft computation **y= fftshift(x)** or **y= fftshift(x, "all")** moves the zero frequency component to the center of the spectrum, which is sometimes a more convenient form.

If **x** is a vector of size **n**, **y** is the vector **x([n/2+1:n, 1:n/2])**

If **x** is an **m** by **n** matrix **y** is the matrix **x([m/2+1:n, 1:m/2], [n/2+1:n, 1:n/2])**.

```
  [x11 x12]           [x22 x21]
x=[          ]   gives  y=[          ]
  [x21 x22]           [x12 x11]
```

y= fftshift(x, n) make the swap only along the **n**th dimension

Examples

```
//make a signal
t=0:0.1:1000;
x=3*sin(t)+8*sin(3*t)+0.5*sin(5*t)+3*rand(t);
//compute the fft
y=fft(x, -1);

//display
clf();
subplot(2,1,1);plot2d(abs(y))
subplot(2,1,2);plot2d(fftshift(abs(y)))

//make a 2D image
t=0:0.1:30;
x=3*sin(t')*cos(2*t)+8*sin(3*t')*sin(5*t)+..
    0.5*sin(5*t')*sin(5*t)+3*rand(t')*rand(t);
//compute the fft
y=fft(x, -1);

//display
clf();
xset('colormap',hotcolormap(256))
```

```
subplot(2,1,1);Matplot(abs(y))  
subplot(2,1,2);Matplot(fftshift(abs(y)))
```

See Also

[fft](#)

Name

`filt_sinc` — samples of sinc function

```
[x]=filt_sinc(n,fl)
```

Parameters

`n`
number of samples

`fl`
cut-off frequency of the associated low-pass filter in Hertz.

`x`
samples of the sinc function

Description

Calculate `n` samples of the function $\sin(2\pi fl t)/(\pi t)$ for $t=-(n-1)/2:(n-1)/2$ (i.e. centred around the origin).

Examples

```
plot(filt_sinc(100,0.1))
```

See Also

`sincd`

Authors

C. B.;

Name

filter — filters a data sequence using a digital filter

```
[y,zf] = filter(num,den,x [,zi])
```

Parameters

num

real vector : the coefficients of the filter numerator in decreasing power order, or a polynomial.

den

real vector : the coefficients of the filter denominator in decreasing power order, or a polynomial.

x

real row vector : the input signal

zi

real row vector of length $\max(\text{length}(a), \text{length}(b)) - 1$: the initial condition relative to a "direct form II transposed" state space representation. The default value is a vector filled with zeros.

y

real row vector : the filtered signal.

zf

real row vector : the final state. It can be used to filter a next batch of the input signal.

Description

This function filters a data sequence using a digital filter using a "direct form II transposed" implementation

References

Oppenheim, A. V. and R.W. Schaffer. Discrete-Time Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1989, pp. 311-312.

See Also

flts , rtitr , ltitr

Authors

Serge Steer, INRIA

Name

find_freq — parameter compatibility for elliptic filter design

```
[m]=find_freq(epsilon,A,n)
```

Parameters

epsilon

passband ripple

A

stopband attenuation

n

filter order

m

frequency needed for construction of elliptic filter

Description

Search for m such that $n=K(1-m)K(m)/(K(m)K(1-m))$ with

$m1=(\epsilon*\epsilon)/(A*A-1);$

If $m = \omega_r^2/\omega_{ac}^2$, the parameters `epsilon`, `A`, `omegac`, `omegar` and `n` are then compatible for defining a prototype elliptic filter. Here, $K=k(m)$ is the complete elliptic integral with parameter m .

See Also

`%k`

Authors

F. D.

Name

findm — for elliptic filter design

```
[m]=findm(chi)
```

Description

Search for m such that $\text{chi} = \%k(1-m)/\%k(m)$ (For use with `find_freq`).

See Also

`%k`

Authors

F. D.;

Name

frfit — frequency response fit

```
sys=frfit(w,fresp,order)
[num,den]=frfit(w,fresp,order)
sys=frfit(w,fresp,order,weight)
[num,den]=frfit(w,fresp,order,weight)
```

Parameters

w
positive real vector of frequencies (Hz)

fresp
complex vector of frequency responses (same size as **w**)

order
integer (required order, degree of **den**)

weight
positive real vector (default value `ones(w)`).

num,den
stable polynomials

Description

`sys=frfit(w,fresp,order,weight)` returns a bi-stable transfer function $G(s)=\text{sys}=\text{num}/\text{den}$, of of given order such that its frequency response $G(w(i))$ matches `fresp(i)`, i.e. `freq(num,den,%i*w)` should be close to `fresp`. `weight(i)` is the weight given to `w(i)`.

Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2), (s^2+s+1)*(s^2+0.3*s+1));
fresp=repfreq(G,w);
Gid=frfit(w,fresp,4);
frespfit=repfreq(Gid,w);
bode(w,[fresp;frespfit])
```

See Also

`frep2tf` , `factors` , `cepstrum` , `mrfit` , `freq` , `calfrq`

Name

frmag — magnitude of FIR and IIR filters

```
[xm, fr]=frmag(sys, npts)
[xm, fr]=frmag(num, den, npts)
```

Parameters

sys

a single input, single output discrete transfer function, or a polynomial or the vector of polynomial coefficients, the filter.

num

a polynomial or the vector of polynomial coefficients, the numerator of the filter

den

a polynomial or the vector of polynomial coefficients, the denominator of the filter (the default value is 1).

npts

integer, the number of points in frequency response.

xm

vector of magnitude of frequency response at the points `fr`.

fr

points in the normalized frequency domain where magnitude is evaluated.

Description

calculates the magnitude of the frequency responses of FIR and IIR filters. The filter description can be one or two vectors of coefficients, one or two polynomials, or a single output discrete transfer function.

the frequency discretisation is given by `fr=linspace(0,1/2,npts)`.

Authors

Carey Bunks.

Examples

```
hz=iir(3,'bp','cheb1',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm)
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm,'r')
```

See Also

iir, eqfir, repfreq, calfrq, phasemag

Name

fsfirlin — design of FIR, linear phase filters, frequency sampling technique

```
[hst]=fsfirlin(hd,flag)
```

Parameters

hd

vector of desired frequency response samples

flag

is equal to 1 or 2, according to the choice of type 1 or type 2 design

hst

vector giving the approximated continuous response on a dense grid of frequencies

Description

function for the design of FIR, linear phase filters using the frequency sampling technique

Examples

```
//
//Example of how to use the fsfirlin macro for the design
//of an FIR filter by a frequency sampling technique.
//
//Two filters are designed : the first (response hst1) with
//abrupt transitions from 0 to 1 between passbands and stop
//bands; the second (response hst2) with one sample in each
//transition band (amplitude 0.5) for smoothing.
//
hd=zeros(1,15) ones(1,10) zeros(1,39); //desired samples
hst1=fsfirlin(hd,1); //filter with no sample in the transition
hd(15)=.5;hd(26)=.5; //samples in the transition bands
hst2=fsfirlin(hd,1); //corresponding filter
pas=1/prod(size(hst1))*0.5;
fg=0:pas:.5; //normalized frequencies grid
plot2d([1 1].*.fg(1:257)',[hst1' hst2']);

// 2nd example
hd=[0*ones(1,15) ones(1,10) 0*ones(1,39)]; //desired samples
hst1=fsfirlin(hd,1); //filter with no sample in the transition
hd(15)=.5;hd(26)=.5; //samples in the transition bands
hst2=fsfirlin(hd,1); //corresponding filter
pas=1/prod(size(hst1))*0.5;
fg=0:pas:.5; //normalized frequencies grid
n=prod(size(hst1))
plot(fg(1:n),hst1);
plot2d(fg(1:n)',hst2',[3],"000");
```

See Also

ffilt , wfir

Authors

G. Le Vey

Name

group — group delay for digital filter

```
[tg, fr]=group(npts, a1i, a2i, b1i, b2i)
```

Parameters

npts

integer : number of points desired in calculation of group delay

a1i

in coefficient, polynomial, rational polynomial, or cascade polynomial form this variable is the transfer function of the filter. In coefficient polynomial form this is a vector of coefficients (see below).

a2i

in coeff poly form this is a vector of coeffs

b1i

in coeff poly form this is a vector of coeffs

b2i

in coeff poly form this is a vector of coeffs

tg

values of group delay evaluated on the grid fr

fr

grid of frequency values where group delay is evaluated

Description

Calculate the group delay of a digital filter with transfer function $h(z)$.

The filter specification can be in coefficient form, polynomial form, rational polynomial form, cascade polynomial form, or in coefficient polynomial form.

In the coefficient polynomial form the transfer function is formulated by the following expression

$$h(z)=\text{prod}(a1i+a2i*z+z**2)/\text{prod}(b1i+b2i*z+z^2)$$

Examples

```
z=poly(0, 'z');  
h=z/(z-.5);  
[tg, fr]=group(100, h);  
plot(fr, tg)
```

Authors

C. B.

Name

hank — covariance to hankel matrix

```
[hk]=hank(m,n,cov)
```

Parameters

m
number of bloc-rows

n
number of bloc-columns

cov
sequence of covariances; it must be given as :[R0 R1 R2...Rk]

hk
computed hankel matrix

Description

this function builds the hankel matrix of size $(m*d, n*d)$ from the covariance sequence of a vector process

Examples

```
//Example of how to use the hank macro for
//building a Hankel matrix from multidimensional
//data (covariance or Markov parameters e.g.)
//
//This is used e.g. in the solution of normal equations
//by classical identification methods (Instrumental Variables e.g.)
//
//1)let's generate the multidimensional data under the form :
// C=[c_0 c_1 c_2 .... c_n]
//where each bloc c_k is a d-dimensional matrix (e.g. the k-th correlation
//of a d-dimensional stochastic process X(t) [c_k = E(X(t) X'(t+k)], '
//being the transposition in scilab)
//
//we take here d=2 and n=64

c=rand(2,2*64)

//generate the hankel matrix H (with 4 bloc-rows and 5 bloc-columns)
//from the data in c

H=hank(4,5,c);
```

See Also

toeplitz

Authors

G. Le Vey

Name

hilb — FIR approximation to a Hilbert transform filter

```
xh=hilb(n [,wtype [,par]])
```

Parameters

n

odd integer : number of points in filter

wtype

string : window type ('re', 'tr', 'hn', 'hm', 'kr', 'ch') (default = 're')

par

window parameter for wtype='kr' or 'ch' default par=[0 0] see the function window for more help

xh

Hilbert transform

Description

Returns the first n points of an FIR approximation to a Hilbert transform filter centred around the origin.

The FIR filter is designed by appropriately windowing the ideal impulse response $h(n) = (2/(n\pi)) * (\sin(n\pi/2))^2$ for n not equal 0 and $h(0) = 0$.

An approximation to an analytic signal generator can be built by designing an FIR (Finite Impulse Response) filter approximation to the Hilbert transform operator. The analytic signal can then be computed by adding the appropriately time-shifted real signal to the imaginary part generated by the Hilbert filter.

References

<http://ieeexplore.ieee.org/iel4/78/7823/00330385.pdf?tp=&arnumber=330385&isnumber=7823>

A. Reilly, G. Frazer, and B. Boashash, "Analytic signal generation Tips and traps", IEEE Trans. Signal Processing, vol. 42, pp.3241-3245, Nov. 1994.

See Also

window , hilbert

Examples

```
plot(hilb(51))
```

Authors

C. B.

Name

hilbert — Discrete-time analytic signal computation of a real signal using Hilbert transform

```
x=hilbert(xr)
```

Parameters

`xr`

real vector : the real signal samples

`x`

Complex vector: the discrete-time analytic signal.

Description

Returns the analytic signal, from a real data sequence.

The analytic signal $x = x_r + i \cdot x_i$ has a real part, x_r , which is the original data, and an imaginary part, x_i , which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90° phase shift.

References

<http://ieeexplore.ieee.org/iel5/78/16975/00782222.pdf?arnumber=782222>

Marple, S.L., "Computing the discrete-time analytic signal via FFT," IEEE Transactions on Signal Processing, Vol. 47, No.9 (September 1999), pp.2600-2603

See Also

`window` , `hil`

Examples

```
//compare the discrete-time analytic signal imaginary part of the impulse real  
// with the FIR approximation of the Hilbert transform filter  
m=25;  
n=2*m+1;  
y=hilbert(eye(n,1));  
h=hilb(n)';  
h=[h((m+1):$);h(1:m)];  
plot([imag(y) h])
```

Authors

C. B.

Name

iir — iir digital filter

```
[hz]=iir(n,ftype,fdesign,frq,delta)
```

Parameters

n

positive number with integer value, the filter order.

ftype

string specifying the filter type, the possible values are: 'lp' for low-pass, 'hp' for high pass, 'bp' for band pass and 'sb' for stop band.

fdesign

string specifying the analog filter design, the possible values are: 'butt', 'cheb1', 'cheb2' and 'ellip'

frq

2-vector of discrete cut-off frequencies (i.e., $0 < \text{frq} < 0.5$). For 'lp' and 'hp' filters only `frq(1)` is used. For 'bp' and 'sb' filters `frq(1)` is the lower cut-off frequency and `frq(2)` is the upper cut-off frequency

delta

2-vector of error values for cheb1, cheb2, and ellip filters where only `delta(1)` is used for cheb1 case, only `delta(2)` is used for cheb2 case, and `delta(1)` and `delta(2)` are both used for ellip case. $0 < \text{delta}(1), \text{delta}(2) < 1$

- for cheb1 filters $1 - \text{delta}(1) < \text{ripple} < 1$ in passband
- for cheb2 filters $0 < \text{ripple} < \text{delta}(2)$ in stopband
- for ellip filters $1 - \text{delta}(1) < \text{ripple} < 1$ in passband and $0 < \text{ripple} < \text{delta}(2)$ in stopband

Description

function which designs an iir digital filter using analog filter designs and bilinear transformation .

Examples

```
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot2d(fr',hzm')
xlabel('Discrete IIR filter band pass 0.15<fr<0.25 ',' ',' ');
q=poly(0,'q'); //to express the result in terms of the delay operator q=z^-1
hzd=horner(hz,1/q)
```

See Also

eqfir, eqiir, analpf, bilt

Authors

Carey Bunks

Name

iirgroup — group delay Lp IIR filter optimization

```
[lt,grad]=iirgroup(p,r,theta,omega,wt,td)
[cout,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

Parameters

r
vector of the module of the poles and the zeros of the filters

theta
vector of the argument of the poles and the zeros of the filters

omega
frequencies where the filter specifications are given

wt
weighting function for and the group delay

td
desired group delay

lt, grad
criterium and gradient values

Description

optimization of IIR filters for the Lp criterium for the the group delay. (Rabiner & Gold pp270-273).

Name

iirlp — Lp IIR filter optimization

```
[cost,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

Parameters

x

1X2 vector of the module and argument of the poles and the zeros of the filters

flag

string : 'a' for amplitude, 'gd' for group delay; default case for amplitude and group delay.

omega

frequencies where the filter specifications are given

wa,wt

weighting functions for the amplitude and the group delay

lambda

weighting (with 1 - lambda) of the costs ('a' and 'gd' for getting the global cost.

ad, td

desired amplitude and group delay

cost, grad

criterion and gradient values

Description

optimization of IIR filters for the Lp criterium for the amplitude and/or the group delay. (Rabiner & Gold pp270-273).

Name

intdec — Changes sampling rate of a signal

```
[y]=intdec(x,lom)
```

Parameters

x

input sampled signal

lom

For a 1D signal this is a scalar which gives the rate change. For a 2D signal this is a 2-Vector of sampling rate changes **lom**=(col rate change,row rate change)

y

Output sampled signal

Description

Changes the sampling rate of a 1D or 2D signal by the rates in lom

Authors

C. B.

Name

jmat — row or column block permutation

```
[j]=jmat(n,m)
```

Parameters

n
number of block rows or block columns of the matrix

m
size of the (square) blocks

Description

This function permutes block rows or block columns of a matrix

Name

kalm — Kalman update

```
[x1, p1, x, p]=kalm(y, x0, p0, f, g, h, q, r)
```

Parameters

f,g,h

current system matrices

q, r

covariance matrices of dynamics and observation noise

x0,p0

state estimate and error variance at t=0 based on data up to t=-1

y

current observation Output from the function is:

x1,p1

updated estimate and error covariance at t=1 based on data up to t=0

x

updated estimate and error covariance at t=0 based on data up to t=0

Description

function which gives the Kalman update and error variance

Authors

C. B.

Name

lattn — recursive solution of normal equations

```
[la, lb]=lattn(n,p,cov)
```

Parameters

- n
maximum order of the filter
- p
fixed dimension of the MA part. If $p = -1$, the algorithm reduces to the classical Levinson recursions.
- cov
matrix containing the R_k 's ($d \times d$ matrices for a d -dimensional process). It must be given the following way
- la
list-type variable, giving the successively calculated polynomials (degree 1 to degree n), with coefficients A_k

Description

solves recursively on n (p being fixed) the following system (normal equations), i.e. identifies the AR part (poles) of a vector ARMA(n, p) process

where $\{R_k; k=1, n \text{ lag}\}$ is the sequence of empirical covariances

Authors

G. Le V.

Name

lattp — lattp

```
[la, lb]=lattp(n,p,cov)
```

Description

see lattn

Authors

G.Levey

Name

lev — Yule-Walker equations (Levinson's algorithm)

```
[ar, sigma2, rc]=lev(r)
```

Parameters

r
correlation coefficients

ar
auto-Regressive model parameters

sigma2
scale constant

rc
reflection coefficients

Description

resolve the Yule-Walker equations
using Levinson's algorithm.

Authors

C. B.

Name

levin — Toeplitz system solver by Levinson algorithm (multidimensional)

```
[la, sig]=levin(n, cov)
```

Parameters

n

A scalar with integer value: the maximum order of the filter

cov

A (nlag*d) × d matrix. It contains the Rk (d × d matrices for a d-dimensional process) stored in the following way :

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{nlag} \end{pmatrix}$$

la

A list, the successively calculated Levinson polynomials (degree 1 to n), with coefficients Ak

sig

A list, the successive mean-square errors.

Description

function which solves recursively on n the following Toeplitz system (normal equations)

$$(I \quad -A_1 \quad \dots \quad -A_n) * \begin{pmatrix} R_1 & R_2 & \dots & R_n \\ R_0 & R_1 & \dots & R_{n-1} \\ R_{-1} & R_0 & \dots & R_{n-2} \\ \vdots & \vdots & \dots & \vdots \\ R_{2-n} & R_{3-n} & \dots & R_1 \\ R_{1-n} & R_{2-n} & \dots & R_0 \end{pmatrix} = 0$$

where {Rk; k=1:nlag} is the sequence of nlag empirical covariances

Examples

```
//We use the 'levin' macro for solving the normal equations
//on two examples: a one-dimensional and a two-dimensional process.
//We need the covariance sequence of the stochastic process.
//This example may usefully be compared with the results from
```

```

//the 'phc' macro (see the corresponding help and example in it)
//
//
//1) A one-dimensional process
// -----
//
//We generate the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observed process);
//the simulated process is sampled at 10 Hz (step 0.1 in t, underafter).

t1=0:.1:100;rand('normal');
y1=sin(2*pi*t1)+sin(2*pi*2*t1);y1=y1+rand(y1);plot(t1,y1);

//covariance of y1

nlag=128;
c1=corr(y1,nlag);
c1=c1';//c1 needs to be given columnwise (see the section PARAMETERS of this he

//compute the filter for a maximum order of n=10
//la is a list-type variable each element of which
//containing the filters of order ranging from 1 to n; (try varying n)
//in the d-dimensional case this is a matrix polynomial (square, d X d)
//sig gives, the same way, the mean-square error

n=15;
[la1,sig1]=levin(n,c1);

//verify that the roots of 'la' contain the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need to retrieve
//the original frequencies (1Hz and 2 Hz) through
//the log and correct scaling by the frequency sampling)
//we verify this for each filter order

for i=1:n, s1=roots(la1(i));s1=log(s1)/2/pi/.1;

//now we get the estimated poles (sorted, positive ones only !)

s1=gsort(imag(s1));s1=s1(1:i/2);end;

//the last two frequencies are the ones really present in the observed
//process ---> the others are "artifacts" coming from the used model size.
//This is related to the rather difficult problem of order estimation.
//
//2) A 2-dimensional process
// -----
//(4 frequencies 1, 2, 3, and 4 Hz, sampled at 0.1 Hz :
// |y_1|          y_1=sin(2*Pi*t)+sin(2*Pi*2*t)+Gaussian noise
// y=|   | with :
// |y_2|          y_2=sin(2*Pi*3*t)+sin(2*Pi*4*t)+Gaussian noise

d=2;dt=0.1;
nlag=64;
t2=0:2*pi*dt:100;
y2=[sin(t2)+sin(2*t2)+rand(t2);sin(3*t2)+sin(4*t2)+rand(t2)];
c2=[];

```

```
for j=1:2, for k=1:2, c2=[c2;corr(y2(k,:),y2(j,:),nlag)];end;end;
c2=matrix(c2,2,128);cov=[];
for j=1:64,cov=[cov;c2(:,(j-1)*d+1:j*d)];end;//covar. columnwise
c2=cov;

//in the multidimensional case, we have to compute the
//roots of the determinant of the matrix polynomial
//(easy in the 2-dimensional case but tricky if d>=3 !).
//We just do that here for the maximum desired
//filter order (n); mp is the matrix polynomial of degree n

[la2,sig2]=levin(n,c2);
mp=la2(n);determinant=mp(1,1)*mp(2,2)-mp(1,2)*mp(2,1);
s2=roots(determinant);s2=log(s2)/2/%pi/0.1;//same trick as above for 1D process
s2=gsort(imag(s2));s2=s2(1:d*n/2);//just the positive ones !

//There the order estimation problem is seen to be much more difficult !
//many artifacts ! The 4 frequencies are in the estimated spectrum
//but beneath many non relevant others.
```

See Also

phc

Authors

G. Le Vey

Name

lindquist — Lindquist's algorithm

```
[P,R,T]=lindquist(n,H,F,G,R0)
```

Parameters

n
number of iterations.

H, F, G
estimated triple from the covariance sequence of y.

R0
 $E(y_k*y_k')$

P
solution of the Riccati equation after n iterations.

R, T
gain matrices of the filter.

Description

computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices R and T of the filter model, by the Lindquist's algorithm.

See Also

srfaur , faurre , phc

Authors

G. Le V.

Name

mese — maximum entropy spectral estimation

```
[sm,fr]=mese(x [,npts]);
```

Parameters

- x
Input sampled data sequence
- npts
Optional parameter giving number of points of fr and sm (default is 256)
- sm
Samples of spectral estimate on the frequency grid fr
- fr
npts equally spaced frequency samples in $[0, .5)$

Description

Calculate the maximum entropy spectral estimate of x

Authors

C. B.

Name

mfft — multi-dimensional fft

```
[xk]=mfft(x,flag,dim)
```

Parameters

x
 $x(i, j, k, \dots)$ input signal in the form of a row vector whose values are arranged so that the i index runs the quickest, followed by the j index, etc.

flag
(-1) FFT or (1) inverse FFT

dim
dimension vector which gives the number of values of x for each of its indices

xk
output of multidimensional fft in same format as for x

Description

FFT for a multi-dimensional signal

For example for a three dimensional vector which has three points along its first dimension, two points along its second dimension and three points along its third dimension the row vector is arranged as follows

```
x=[x(1,1,1),x(2,1,1),x(3,1,1),  
   x(1,2,1),x(2,2,1),x(3,2,1),  
   x(1,1,2),x(2,1,2),x(3,1,2),  
   x(1,2,2),x(2,2,2),x(3,2,2),  
   x(1,1,3),x(2,1,3),x(3,1,3),  
   x(1,2,3),x(2,2,3),x(3,2,3)]
```

and the `dim` vector is: `dim=[3,2,3]`

Authors

C. B.

Name

mrfit — frequency response fit

```
sys=mrfit(w,mag,order)
[num,den]=mrfit(w,mag,order)
sys=mrfit(w,mag,order,weight)
[num,den]=mrfit(w,mag,order,weight)
```

Parameters

w
positive real vector of frequencies (Hz)

mag
real vector of frequency responses magnitude (same size as *w*)

order
integer (required order, degree of *den*)

weight
positive real vector (default value `ones(w)`).

num,den
stable polynomials

Description

`sys=mrfit(w,mag,order,weight)` returns a bi-stable transfer function $G(s)=sys=num/den$, of of given order such that its frequency response magnitude `abs(G(w(i)))` matches `mag(i)` i.e. `abs(freq(num,den,%i*w))` should be close to `mag`. `weight(i)` is the weighth given to `w(i)`.

Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2),(s^2+s+1)*(s^2+0.3*s+1)); // syslin('c',Num,Den);
fresp=repfreq(G,w);
mag=abs(fresp);
Gid=mrfit(w,mag,4);
frespfit=repfreq(Gid,w);
plot2d([w',w'],[mag(:),abs(frespfit(:))])
```

See Also

`cepstrum` , `frfit` , `freq` , `calfrq`

Name

%asn — elliptic integral

```
[y]=%asn(x,m)
```

Parameters

x
upper limit of integral ($x > 0$) (can be a vector)

m
parameter of integral ($0 < m < 1$)

y
value of the integral

Description

Calculates the elliptic integral

If x is a vector, y is a vector of same dimension as x.

Examples

```
m=0.8;z=%asn(1/sqrt(m),m);K=real(z);Ktilde=imag(z);
x2max=1/sqrt(m);
x1=0:0.05:1;x2=1:((x2max-1)/20):x2max;x3=x2max:0.05:10;
x=[x1,x2,x3];
y=%asn(x,m);
rect=[0,-Ktilde,1.1*K,2*Ktilde];
plot2d(real(y)',imag(y)',1,'011',' ',rect)

deff('y=f(t)','y=1/sqrt((1-t^2)*(1-m*t^2))');
intg(0,0.9,f)-%asn(0.9,m) //Works for real case only!
```

Authors

F. D.

Name

`%k` — Jacobi's complete elliptic integral

```
[K]=%k(m)
```

Parameters

`m`
parameter of the elliptic integral $0 < m < 1$ (`m` can be a vector)

`K`
value of the elliptic integral from 0 to 1 on the real axis

Description

Calculates Jacobi's complete elliptic integral of the first kind :

References

Abramowitz and Stegun page 598

Examples

```
m=0.4;  
%asn(1,m)  
%k(m)
```

See Also

`%asn`

Authors

F.D.

Name

%sn — Jacobi 's elliptic function

```
[y]=%sn(x,m)
```

Parameters

x
a point inside the fundamental rectangle defined by the elliptic integral; x is a vector of complex numbers

m
parameter of the elliptic integral ($0 < m < 1$)

y
result

Description

Jacobi 's sn elliptic function with parameter m: the inverse of the elliptic integral for the parameter m.

The amplitude am is computed in fortran and the addition formulas for elliptic functions are applied

Examples

```
m=0.36;  
K=%k(m);  
P=4*K; //Real period  
real_val=0:(P/50):P;  
plot(real_val,real(%sn(real_val,m)))  
clf();  
KK=%k(1-m);  
Ip=2*KK;  
ima_val1=0:(Ip/50):KK-0.001;  
ima_val2=(KK+0.05):(Ip/25):(Ip+KK);  
z1=%sn(%i*ima_val1,m);z2=%sn(%i*ima_val2,m);  
plot2d([ima_val1',ima_val2'],[imag(z1)',imag(z2)']]);  
xgrid(3)
```

See Also

%asn , %k

Authors

F. D.

Name

phc — Markovian representation

```
[H, F, G]=phc(hk, d, r)
```

Parameters

hk
hankel matrix

d
dimension of the observation

r
desired dimension of the state vector for the approximated model

H, F, G
relevant matrices of the Markovian model

Description

Function which computes the matrices H , F , G of a Markovian representation by the principal hankel component approximation method, from the hankel matrix built from the covariance sequence of a stochastic process.

Examples

```
//This example may usefully be compared with the results from
//the 'levin' macro (see the corresponding help and example)
//
//We consider the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observation);
//the simulated process is sampled at 10 Hz.

t=0:.1:100;rand('normal');
y=sin(2*pi*t)+sin(2*pi*2*t);y=y+rand(y);plot(t,y)

//covariance of y

nlag=128;
c=corr(y,nlag);

//hankel matrix from the covariance sequence
//(we can choose to take more information from covariance
//by taking greater n and m; try it to compare the results !

n=20;m=20;
h=hank(n,m,c);

//compute the Markov representation (mh,mf,mg)
//We just take here a state dimension equal to 4 :
//this is the rather difficult problem of estimating the order !
//Try varying ns !
//(the observation dimension is here equal to one)
```



```
ns=4;
[mh,mf,mg]=phc(h,1,ns);

//verify that the spectrum of mf contains the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need
//to retrieve the original frequencies through the log
//and correct scaling by the frequency sampling)

s=spec(mf);s=log(s);
s=s/2/%pi/.1;

//now we get the estimated spectrum
imag(s),
```

See Also

levin

Name

pspect — two sided cross-spectral estimate between 2 discrete time signals using the Welch's average periodogram method.

```
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,x [,y] [,wpar])
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,nx [,ny] [,wpar])
```

Parameters

- x**
vector, the time-domain samples of the first signal.
- y**
vector, the time-domain samples of the second signal. If **y** is omitted it is supposed to be equal to **x** (auto-correlation). If it is present, it must have the same number of element than **x**.
- nx**
a scalar : the number of samples in the **x** signal. In this case the segments of the **x** signal are loaded by a user defined function named **getx** (see below).
- ny**
a scalar : the number of samples in the **y** signal. In this case the segments of the **y** signal are loaded by a user defined function named **gety** (see below). If present **ny** must be equal to **nx**.
- sec_step**
offset of each data window. The overlap **D** is given by **sec_leng -sec_step**. if **sec_step==sec_leng/2** 50% overlap is made. The overlap
- sec_leng**
Number of points of the window.
- wtype**
The window type
- 're': rectangular
 - 'tr': triangular
 - 'hm': Hamming
 - 'hn' : Hanning
 - 'kr': Kaiser,in this case the **wpar** argument must be given
 - 'ch': Chebyshev, in this case the **wpar** argument must be given
- wpar**
optional parameters for Kaiser and Chebyshev windows:
- 'kr': **wpar** must be a strictly positive number
 - 'ch': **wpar** must be a 2 element vector [main_lobe_width,side_lobe_height]with $0 < \text{main\_lobe\_width} < .5$, and $\text{side\_lobe\_height} > 0$
- sm**
Two sided power spectral estimate in the interval $[0, 1]$ of the normalized frequencies. It is a row array with **sec_len** elements . The array is real in case of auto-correlation and complex in case of cross-correlation.

The associated normalized frequencies array is `linspace(0,1,sec_len)`.

`cwp`

unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

Description

Computes the cross-spectrum estimate of two signals `x` and `y` if both are given and the auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the modified periodogram method.

The cross spectrum of two signal `x` and `y` is defined to be

$$S_{xy}(\omega) = \frac{1}{N} \left(\sum_{n=0}^{N-1} x(n) e^{-i\omega n} \right) \left(\sum_{n=0}^{N-1} \bar{y}(n) e^{i\omega n} \right)$$

The modified periodogram method of spectral estimation repeatedly calculates the periodogram of windowed sub-sections of the data contained in `x` and `y`. These periodograms are then averaged together and normalized by an appropriate constant to obtain the final spectral estimate. It is the averaging process which reduces the variance in the estimate.

For batch processing, the `x` and `y` data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

`xk=getx(ns,offset)` and `yk=gety(ns,offset)` where `ns` is the segment size and `offset` is the index of the first element of the segment in the full signal.

Reference

Oppenheim, A.V., and R.W. Schaffer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);

//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);

//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1); xf=fft(x1,-1);y=real(fft(hf.*xf,1));

//plot magnitude of filter
h2=[h 0*ones(1:968)];hf2=fft(h2,-1);hf2=real(hf2.*conj(hf2));
hsize=maxi(size(hf2));fr=(1:hsize)/hsize;plot(fr,log(hf2));

//pspect example
sm=pspect(100,200,'tr',y);smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm));
rand('unif');
```

See Also

`cpspect`, `pspect`, `mese`, `window`

Authors

C. Bunks INRIA

Name

`remez` — Remez exchange algorithm for the weighted chebyshev approximation of a continuous function with a sum of cosines.

```
an=remez(guess,mag,fgrid,weight)
```

Parameters

`guess`

real array of size `n+2` the initial guess

`fgrid`

real array of size `ng`: the grid of normalized frequency points in `[0,.5[`

`mag`

real array of size `ng`: the desired magnitude on grid `fg`

`weight`

real array of size `ng`: weighting function on error on grid `fg`

`an`

real array of size `n`: cosine coefficients

Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form

```
h = sum[a(i)*cos(weight)], i=1:n
```

An FIR, linear-phase filter can be obtained from the the output of `remez` by using the following commands:

```
hn(1:nc-1)=an(nc:-1:2)/2;  
hn(nc)=an(1);  
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

This function is mainly intended to be called by the `remezb` function.

Bibliography

E.W. Cheney, Introduction to Approximation Theory, McGraw-Hill, 1966

http://en.wikipedia.org/wiki/Remez_algorithm

References

This function is based on the fortran code `remez.f` written by:

- James H. McClellan, department of electrical engineering and computer science, Massachusetts Institute of Technology, Cambridge, Massachussets. 02139
- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001

- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001

Examples

```
nc=21;
ngrid=nc*250;
fgrid=.5*(0:(ngrid-1))/(ngrid-1);
mag(1:ngrid/2)=ones(1:ngrid/2);
mag(ngrid/2+1:ngrid)=0*ones(1:ngrid/2);
weight=ones(fgrid);
guess=round(1:ngrid/nc:ngrid);
guess(nc+1)=ngrid;
guess(nc+2)=ngrid;
an=remez(guess,mag,fgrid,weight);
```

See Also

remezb, eqfir

Name

remezb — Minimax approximation of magnitude response

```
[an]=remezb(nc,fg,ds,wt)
```

Parameters

nc
Number of cosine functions

fg
Grid of frequency points in [0,.5)

ds
Desired magnitude on grid fg

wt
Weighting function on error on grid fg

an
Cosine filter coefficients

Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form $h = \sum [a(n) * \cos(wn)]$ for $n=0,1,...,nc$. An FIR, linear-phase filter can be obtained from the the output of the function by using the following commands

```
hn(1:nc-1)=an(nc:-1:2)/2;  
hn(nc)=an(1);  
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

Examples

```
// Choose the number of cosine functions and create a dense grid  
// in [0,.24) and [.26,.5)  
nc=21;ngrid=nc*16;  
fg=.24*(0:ngrid/2-1)/(ngrid/2-1);  
fg(ngrid/2+1:ngrid)=fg(1:ngrid/2)+.26*ones(1:ngrid/2);  
  
// Specify a low pass filter magnitude for the desired response  
ds(1:ngrid/2)=ones(1:ngrid/2);  
ds(ngrid/2+1:ngrid)=zeros(1:ngrid/2);  
  
// Specify a uniform weighting function  
wt=ones(fg);  
  
// Run remezb  
an=remezb(nc,fg,ds,wt)  
  
// Make a linear phase FIR filter  
hn(1:nc-1)=an(nc:-1:2)/2;  
hn(nc)=an(1);
```

```
hn(nc+1:2*nc-1)=an(2:nc)/2;

// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));

// Choose the number of cosine functions and create a dense grid in [0,.5)
nc=21; ngrid=nc*16;
fg=.5*(0:(ngrid-1))/ngrid;

// Specify a triangular shaped magnitude for the desired response
ds(1:ngrid/2)=(0:ngrid/2-1)/(ngrid/2-1);
ds(ngrid/2+1:ngrid)=ds(ngrid/2:-1:1);

// Specify a uniform weighting function
wt=ones(fg);

// Run remezb
an=remezb(nc,fg,ds,wt)

// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;

// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
```

See Also

[eqfir](#)

Authors

C. B.

Name

rpem — Recursive Prediction-Error Minimization estimation

```
[w1, [v]] = rpem(w0, u0, y0, [lambda, [k, [c]]])
```

Arguments

w0

list(theta, p, l, phi, psi) where:

theta

[a,b,c] is a real vector of order 3*n

a,b,c

$a=[a(1), \dots, a(n)]$, $b=[b(1), \dots, b(n)]$, $c=[c(1), \dots, c(n)]$

p

(3*n x 3*n) real matrix.

phi,psi,l

real vector of dimension 3*n

Applicable values for the first call:

```
theta=phi=psi=l=0*ones(1, 3*n). p=eye(3*n, 3*n)
```

u0

real vector of inputs (arbitrary size). (u0(\$)) is not used by rpem)

y0

vector of outputs (same dimension as u0). (y0(1)) is not used by rpem).

If the time domain is (t0, t0+k-1) the u0 vector contains the inputs

$u(t_0), u(t_0+1), \dots, u(t_0+k-1)$ and y0 the outputs

$y(t_0), y(t_0+1), \dots, y(t_0+k-1)$

Optional arguments

lambda

optional argument (forgetting constant) choosed close to 1 as convergence occur:

lambda=[lambda0, alfa, beta] evolves according to :

```
lambda(t)=alfa*lambda(t-1)+beta
```

with $\lambda(0)=\lambda_0$

k

contraction factor to be chosen close to 1 as convergence occurs.

k=[k0, mu, nu] evolves according to:

$$k(t) = \mu * k(t-1) + \nu u$$

with $k(0) = k_0$.

c
Large argument. ($c=1000$ is the default value).

Outputs:

w1
Update for w_0 .

v
sum of squared prediction errors on u_0, y_0 . (optional).

In particular $w1(1)$ is the new estimate of θ . If a new sample $u1, y1$ is available the update is obtained by:

$[w2, [v]] = \text{rpem}(w1, u1, y1, [\text{lambda}, [k, [c]]])$. Arbitrary large series can thus be treated.

Description

Recursive estimation of arguments in an ARMAX model. Uses Ljung-Soderstrom recursive prediction error method. Model considered is the following:

$$y(t) + a(1)*y(t-1) + \dots + a(n)*y(t-n) = b(1)*u(t-1) + \dots + b(n)*u(t-n) + e(t) + c(1)*e(t-1) + \dots + c(n)*e(t-n)$$

The effect of this command is to update the estimation of unknown argument $\theta = [a, b, c]$ with $a = [a(1), \dots, a(n)]$, $b = [b(1), \dots, b(n)]$, $c = [c(1), \dots, c(n)]$.

Name

sincd — digital sinc function or Dirichlet kernel

```
[s]=sincd(n,flag)
```

Parameters

n

integer

flag

if `flag = 1` the function is centred around the origin; if `flag = 2` the function is delayed by $\pi/(2 \cdot n)$

s

vector of values of the function on a dense grid of frequencies

Description

function which calculates the function $\text{Sin}(N \cdot x)/N \cdot \text{Sin}(x)$

Examples

```
plot(sincd(10,1))
```

Authors

G. Le V.

Name

srfaur — square-root algorithm

```
[p,s,t,l,rt,tt]=srfaur(h,f,g,r0,n,p,s,t,l)
```

Parameters

h, f, g

convenient matrices of the state-space model.

r0

$E(y_k y_k')$.

n

number of iterations.

p

estimate of the solution after n iterations.

s, t, l

intermediate matrices for successive iterations;

rt, tt

gain matrices of the filter model after n iterations.

p, s, t, l

may be given as input if more than one recursion is desired (evaluation of intermediate values of p).

Description

square-root algorithm for the algebraic Riccati equation.

Examples

```
//GENERATE SIGNAL
x=%pi/10:%pi/10:102.4*pi;
rand('seed',0);rand('normal');
y=[1;1]*sin(x)+[sin(2*x);sin(1.9*x)]+rand(2,1024);

//COMPUTE CORRELATIONS
c=[];for j=1:2,for k=1:2,c=[c;corr(y(k,:),y(j,:),64)];end;end
c=matrix(c,2,128);

//FINDING H,F,G with 6 states
hk=hank(20,20,c);
[H,F,G]=phc(hk,2,6);

//SOLVING RICCATI EQN
r0=c(1:2,1:2);
[P,s,t,l,Rt,Tt]=srfaur(H,F,G,r0,200);

//Make covariance matrix exactly symmetric
Rt=(Rt+Rt')/2
```

See Also

phc , faurre , lindquist

Name

srkf — square root Kalman filter

```
[x1, p1]=srkf(y, x0, p0, f, h, q, r)
```

Parameters

f, h

current system matrices

q, r

covariance matrices of dynamics and observation noise

x0, p0

state estimate and error variance at t=0 based on data up to t=-1

y

current observation Output from the function is

x1, p1

updated estimate and error covariance at t=1 based on data up to t=0

Description

square root Kalman filter algorithm

Authors

C. B.

Name

sskf — steady-state Kalman filter

```
[xe,pe]=sskf(y,f,h,q,r,x0)
```

Parameters

y
data in form $[y_0, y_1, \dots, y_n]$, y_k a column vector

f
system matrix $\text{dim}(N \times N)$

h
observations matrix $\text{dim}(M \times N)$

q
dynamics noise matrix $\text{dim}(N \times N)$

r
observations noise matrix $\text{dim}(M \times M)$

x0
initial state estimate

xe
estimated state

pe
steady-state error covariance

Description

steady-state Kalman filter

Authors

C. B.

Name

syredi — Design of iir filters, syredi code interface

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(ityp,iapro,om,deltap,deltas)
```

Parameters

itype

integer, the filter type: 1 stands for low-pass, 2 for high-pass, 3 for band-pass, 4 for stop-band.

iapro

integer, the design approximation type: 1 stands for butterworth, 2 for elliptic, 3 for Chebychev1, 4 for Chebychev2.

om

4-vector of cutoff frequencies (in radians) $om=[om1, om2, om3, om4]$,

$0 \leq om1 \leq om2 \leq om3 \leq om4 \leq \pi$.

When `ftype='lp'` or `'hp'`, `om3` and `om4` are not used and may be set to 0.

deltap

a real scalar, the ripple in the passband. $0 < \text{deltap} < 1$

deltas

a real scalar, the ripple in the stopband. $0 < \text{deltas} < 1$

gain

scalar, the filter gain

b2

real row vector, degree 2 coefficients of numerators.

b1

real row vector, degree 1 coefficients of numerators.

b0

real row vector, degree 0 coefficients of numerators.

c1

real row vector, degree 1 coefficients of denominators.

c0

real row vector, degree 0 coefficients of denominators.

zzeros

complex row vector, filter zeros in the z-domain

zpoles

complex row vector, filter poles in the z-domain

Description

Computes iir filter approximation. The result is given as a set of second order transfer functions $H_i=(b0(i)+b1(i)*z+b2(i)*z^2)/(c0(i)+c1(i)*z+z^2)$ and also as a poles, zeros, gain representation.

The filter obtained is $h=fact*H1*...*Hn$.

Remark

This built-in function is mainly intended to be used by the `eqiir` function.

References

The `syredi` code is derived from `doredi` package written by Guenter F. Dehner, Institut fuer Nachrichtentechnik Universitaet Erlangen-Nuernberg, Germany.

Dehner, G.F. 1979, DOREDI: Program for Design and Optimization of REcursive DIgital filters-Programs for Digital Signal Processing, ed: Digital Signal Processing committee of IEEE Acoustics, Speech and Signal Processing Society.

For DOREDI.f source code see <http://michaelgellis.tripod.com/dsp/pgm25.html>

Examples

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(1,4,[2*pi/10,4*pi/10,0,0],0.02,0.0)
h=fact*(b0+b1*z+b2*z^2)./(c0+c1*z+z^2)
```

See Also

`eqiir`

Name

system — observation update

```
[x1, y]=system(x0, f, g, h, q, r)
```

Parameters

x0
input state vector

f
system matrix

g
input matrix

h
Output matrix

q
input noise covariance matrix

r
output noise covariance matrix

x1
output state vector

y
output observation

Description

define system function which generates the next observation given the old state. System recursively calculated

```
x1=f*x0+g*u  
y=h*x0+v
```

where u is distributed $N(0, q)$ and v is distribute $N(0, r)$.

Authors

C. B.

Name

trans — low-pass to other filter transform

```
hzt=trans(hz, tr_type, frq)
hzt=trans(pd, zd, gd, tr_type, frq)
```

Parameters

hz
a single input single output discrete transfer function, the low pass filter

pd
Vector of given filter poles

zd
Vector of given filter zeros

gd
scalar: the given filter gain

tr_type
string, the type of transformation, see description for possible values

frq
2-vector of discrete cut-off frequencies (i.e., $0 < frq < .5$). see description for details.

hzt
transformed filter transfert function.

Description

function for transforming standardized low-pass filter given its poles-zeros_gain representation into one of the following filters:

tr_type='lp'
low pass filter, the cutoff frequency is given by the first entry of **frq**, the second one is ignored.

tr_type='hp'
high pass filter, the cutoff frequency is given by the first entry of **frq**, the second one is ignored.

tr_type='bp'
band pass filter, the frequency range is given by **frq(1)** and **frq(2)**.

tr_type='sb'
stop band filter, the frequency range is given by **frq(1)** and **frq(2)**.

Used functions

bilt

Examples

```
clf()
Hlp=iir(3, 'lp', 'ellip', [0.1 0], [.08 .03]);
```

```
Hbp=trans(Hlp, 'bp', [0.01 0.1]);  
Hsb=trans(Hlp, 'sb', [0.01 0.1])  
  
clf();gainplot([Hlp;Hbp;Hsb],1d-3,0.48);  
l=legend(['original low pass';'band pass';'stop band']);  
l.legend_location="in_lower_left";
```

Authors

Carey Bunks ;

See Also

iir, bilt

Name

wfir — linear-phase FIR filters

```
[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

Parameters

ftype

string : 'lp', 'hp', 'bp', 'sb' (filter type)

forder

Filter order (pos integer)(odd for ftype='hp' or 'sb')

cfreq

2-vector of cutoff frequencies ($0 < \text{cfreq}(1), \text{cfreq}(2) < .5$) only $\text{cfreq}(1)$ is used when ftype='lp' or 'hp'

wtype

Window type ('re', 'tr', 'hm', 'hn', 'kr', 'ch')

fpar

2-vector of window parameters. Kaiser window $\text{fpar}(1) > 0$ $\text{fpar}(2) = 0$. Chebyshev window $\text{fpar}(1) > 0$, $\text{fpar}(2) < 0$ or $\text{fpar}(1) < 0$, $0 < \text{fpar}(2) < .5$

wft

time domain filter coefficients

wfm

frequency domain filter response on the grid fr

fr

Frequency grid

Description

Function which makes linear-phase, FIR low-pass, band-pass, high-pass, and stop-band filters using the windowing technique. Works interactively if called with no arguments.

Authors

C. Bunks

Name

wiener — Wiener estimate

```
[xs,ps,xf,pf]=wiener(y,x0,p0,f,g,h,q,r)
```

Parameters

f, g, h

system matrices in the interval $[t_0, t_f]$

f

$= [f_0, f_1, \dots, f_f]$, and f_k is a $n \times n$ matrix

g

$= [g_0, g_1, \dots, g_f]$, and g_k is a $n \times n$ matrix

h

$= [h_0, h_1, \dots, h_f]$, and h_k is a $m \times n$ matrix

q, r

covariance matrices of dynamics and observation noise

q

$= [q_0, q_1, \dots, q_f]$, and q_k is a $n \times n$ matrix

r

$= [r_0, r_1, \dots, r_f]$, and r_k is a $m \times m$ matrix

x0, p0

initial state estimate and error variance

y

observations in the interval $[t_0, t_f]$. $y = [y_0, y_1, \dots, y_f]$, and y_k is a column m -vector

xs

Smoothed state estimate $xs = [xs_0, xs_1, \dots, xs_f]$, and xs_k is a column n -vector

ps

Error covariance of smoothed estimate $ps = [p_0, p_1, \dots, p_f]$, and p_k is a $n \times n$ matrix

xf

Filtered state estimate $xf = [xf_0, xf_1, \dots, xf_f]$, and xf_k is a column n -vector

pf

Error covariance of filtered estimate $pf = [p_0, p_1, \dots, p_f]$, and p_k is a $n \times n$ matrix

Description

function which gives the Wiener estimate using the forward-backward Kalman filter formulation

Authors

C. B.

Name

wigner — 'time-frequency' wigner spectrum

```
[tab]=wigner(x,h,deltat,zp)
```

Parameters

tab
wigner spectrum (lines correspond to the time variable)

x
analyzed signal

h
data window

deltat
analysis time increment (in samples)

zp
length of FFT's. π/zp gives the frequency increment.

Description

function which computes the 'time-frequency' wigner spectrum of a signal.

Name

window — compute symmetric window of various type

```
win_l=window('re',n)
win_l=window('tr',n)
win_l=window('hn',n)
win_l=window('hm',n)
win_l=window('kr',n,alpha)
[win_l,cwp]=window('ch',n,par)
```

Parameters

n

window length

par

parameter 2-vector $\text{par}=[dp, df]$, where dp ($0 < dp < .5$) rules the main lobe width and df rules the side lobe height ($df > 0$).

Only one of these two value should be specified the other one should set equal to -1.

alpha

kaiser window parameter α ($\alpha > 0$).

win

window

cwp

unspecified Chebyshev window parameter

Description

function which calculates various symmetric window for Disgital signal processing

The Kaiser window is a nearly optimal window function. α is an arbitrary positive real number that determines the shape of the window, and the integer n is the length of the window.

By construction, this function peaks at unity for $k = n/2$, i.e. at the center of the window, and decays exponentially towards the window edges. The larger the value of α , the narrower the window becomes; $\alpha = 0$ corresponds to a rectangular window. Conversely, for larger α the width of the main lobe increases in the Fourier transform, while the side lobes decrease in amplitude. Thus, this parameter controls the tradeoff between main-lobe width and side-lobe area.

alpha	window shape
0	Rectangular shape
5	Similar to the Hamming window
6	Similar to the Hanning window
8.6	Similar to the Blackman window

The Chebyshev window minimizes the mainlobe width, given a particular sidelobe height. It is characterized by an equiripple behavior, that is, its sidelobes all have the same height.

The Hanning and Hamming windows are quite similar, they only differ in the choice of one parameter α : $w = \alpha + (1 - \alpha) \cos(2\pi x / (n-1))$ α is equal to 1/2 in Hanning window and to 0.54 in Hamming window.

Examples

```
// Hamming window
clf()
N=64;
w=window('hm',N);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Kaiser window
clf()
N=64;
w=window('kr',N,6);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Chebyshev window
clf()
N=64;
[w,df]=window('ch',N,[0.005,-1]);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
```

See Also

wfir , frmag , ffilt

Authors

Carey Bunks

Bibliography

IEEE. Programs for Digital Signal Processing. IEEE Press. New York: John Wiley and Sons, 1979. Program 5.2.

Name

yulewalk — least-square filter design

```
Hz = yulewalk(N, frq, mag)
```

Parameters

N
integer (order of desired filter)

frq
real row vector (non-decreasing order), frequencies.

mag
non negative real row vector (same size as frq), desired magnitudes.

Hz
filter $B(z)/A(z)$

Description

Hz = yulewalk(N,frq,mag) finds the N-th order iir filter

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1)z^{n-1} + b(2)z^{n-2} + \dots + b(n)}{z^{n-1} + a(2)z^{n-2} + \dots + a(n)}$$

which matches the magnitude frequency response given by vectors frq and mag. Vectors frq and mag specify the frequency and magnitude of the desired frequency response. The frequencies in frq must be between 0.0 and 1.0, with 1.0 corresponding to half the sample rate. They must be in increasing order and start with 0.0 and end with 1.0.

Examples

```
f=[0,0.4,0.4,0.6,0.6,1];H=[0,0,1,1,0,0];Hz=yulewalk(8,f,H);
fs=1000;fhz = f*fs/2;
clf(0);xset('window',0);plot2d(fhz',H');
xlabel('Desired Frequency Response (Magnitude)')
[frq,repf]=repfreq(Hz,0:0.001:0.5);
clf(1);xset('window',1);plot2d(fs*frq',abs(repf));
xlabel('Obtained Frequency Response (Magnitude)')
```

Name

zpbutt — Butterworth analog filter

```
[pols, gain]=zpbutt(n, omegac)
```

Parameters

n
integer (filter order)

omegac
real (cut-off frequency in Hertz)

pols
resulting poles of filter

gain
resulting gain of filter

Description

computes the poles of a Butterworth analog filter of order n and cutoff frequency omegac transfer function $H(s)$ is calculated by $H(s)=\text{gain}/\text{real}(\text{poly}(\text{pols}, 's'))$

Authors

F. Delebecque INRIA

Name

zpch1 — Chebyshev analog filter

```
[poles, gain]=zpch1(n, epsilon, omegac)
```

Parameters

n
integer (filter order)

epsilon
real : ripple in the pass band ($0 < \text{epsilon} < 1$)

omegac
real : cut-off frequency in Hertz

poles
resulting filter poles

gain
resulting filter gain

Description

Poles of a Type 1 Chebyshev analog filter. The transfer function is given by :

```
H(s)=gain/poly(poles, 's')
```

Authors

F.D.

Name

zpch2 — Chebyshev analog filter

```
[zeros,poles,gain]=zpch2(n,A,omegar)
```

Parameters

n
integer : filter order

A
real : attenuation in stop band ($A>1$)

omegar
real : cut-off frequency in Hertz

zeros
resulting filter zeros

poles
resulting filter poles

gain
Resulting filter gain

Description

Poles and zeros of a type 2 Chebyshev analog filter gain is the gain of the filter

```
H(s)=gain*poly(zeros,'s')/poly(poles,'s')
```

Authors

F.D.

Name

zpell — lowpass elliptic filter

```
[zeros,poles,gain]=zpell(epsilon,A,omegac,omegar)
```

Parameters

epsilon

real : ripple of filter in pass band ($0 < \text{epsilon} < 1$)

A

real : attenuation of filter in stop band ($A > 1$)

omegac

real : pass band cut-off frequency in Hertz

omegar

real : stop band cut-off frequency in Hertz

zeros

resulting zeros of filter

poles

resulting poles of filter

gain

resulting gain of filter

Description

Poles and zeros of prototype lowpass elliptic filter. `gain` is the gain of the filter

See Also

`ell1mag` , `eqiir`

Authors

F.D.

Part XXIX. Sparses Matrix

Name

full — sparse to full matrix conversion

```
X=full(sp)
```

Parameters

sp
real or complex sparse (or full) matrix

X
full matrix

Description

`X=full(sp)` converts the sparse matrix `sp` into its full representation. (If `sp` is already full then `X` equals `sp`).

Examples

```
sp=sparse([1,2;5,4;3,1],[1,2,3]);  
A=full(sp)
```

See Also

`sparse` , `sprand` , `speye`

Name

gmres — Generalized Minimum RESidual method

```
[x,flag,err,iter,res] = gmres(A,b,rstr,tol,maxi,M,x0)
```

Parameters

A
n-by-n matrix or function returning $A \cdot x$

b
right hand side vector

x0
initial guess vector (default: zeros(n,1))

M
preconditioner: matrix or function returning $M \cdot x$ (In the first case, default: eye(n,n))

rstr
number of iterations between restarts (default: 10)

maxi
maximum number of iterations (default: n)

tol
error tolerance (default: 1e-6)

x
solution vector

err
final residual norm

iter
number of iterations performed

flag
0 =
 gmres converged to the desired tolerance within maxi iterations
1 =
 no convergence given maxi

res
residual vector

Description

GMRES
solves the linear system $Ax=b$ using the Generalized Minimal residual method with restarts.

Details
of this algorithm are described in :

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993 (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).

"Iterative Methods for Sparse Linear Systems, Second Edition" Saad, SIAM Publications, 2003
(ftp ftp.cs.umn.edu; cd dept/users/saad/PS; get all_ps.zip).

Examples

```
// GMRES call x=gmres(A,b);
```

See Also

pcg , qmr

Authors

Sage Group (IRISA, 2005)

Name

ludel — utility function used with lufact

```
ludel(hand)
```

Parameters

hand
handle to sparse lu factors (output of lufact)

Description

This function is used in conjunction with `lufact`. It clears the internal memory space used to store the result of `lufact`.

The sequence of commands `[p,r]=lufact(A);x=lusolve(p,b);ludel(p);` solves the sparse linear system $A \cdot x = b$ and clears `p`.

See Also

sparse , lufact , luget

Name

lufact — sparse lu factorization

```
[hand, rk]=lufact(A, prec)
```

Parameters

- A
square sparse matrix
- hand
handle to sparse lu factors
- rk
integer (rank of A)
- prec
a vector of size two `prec=[eps, reps]` giving the absolute and relative thresholds.

Description

`[hand, rk]=lufact(A)` performs the lu factorization of sparse matrix A. `hand` (no display) is used by `lusolve` (for solving linear system) and `luget` (for retrieving the factors). `hand` should be cleared by the command: `ludel(hand);`

The A matrix needs not be full rank but must be square (since A is assumed sparse one may add zeros if necessary to squaring down A).

eps :

The absolute magnitude an element must have to be considered as a pivot candidate, except as a last resort. This number should be set significantly smaller than the smallest diagonal element that is expected to be placed in the matrix. the default value is `%eps`.

reps :

This number determines what the pivot relative threshold will be. It should be between zero and one. If it is one then the pivoting method becomes complete pivoting, which is very slow and tends to fill up the matrix. If it is set close to zero the pivoting method becomes strict Markowitz with no threshold. The pivot threshold is used to eliminate pivot candidates that would cause excessive element growth if they were used. Element growth is the cause of roundoff error. Element growth occurs even in well-conditioned matrices. Setting the `reps` large will reduce element growth and roundoff error, but setting it too large will cause execution time to be excessive and will result in a large number of fill-ins. If this occurs, accuracy can actually be degraded because of the large number of operations required on the matrix due to the large number of fill-ins. A good value seems to be 0.001 which is the default value. The default is chosen by giving a value larger than one or less than or equal to zero. This value should be increased and the matrix resolved if growth is found to be excessive. Changing the pivot threshold does not improve performance on matrices where growth is low, as is often the case with ill-conditioned matrices. `reps` was chosen for use with nearly diagonally dominant matrices such as node- and modified-node admittance matrices. For these matrices it is usually best to use diagonal pivoting. For matrices without a strong diagonal, it is usually best to use a larger threshold, such as 0.01 or 0.1.

Examples

```
a=rand(5,5);b=rand(5,1);A=sparse(a);  
[h,rk]=lufact(A);  
x=lusolve(h,b);a*x-b
```

`ludel(h)`

See Also

`sparse` , `lusolve` , `luget`

Name

luget — extraction of sparse LU factors

```
[P, L, U, Q]=luget(hand)
```

Parameters

hand

handle, output of `lufact`

P

sparse permutation matrix

L

sparse matrix, lower triangular if `hand` is obtained from a non singular matrix

U

square non singular upper triangular sparse matrix with ones along the main diagonal

Q

sparse permutation matrix

Description

`[P, L, U, Q]=luget(hand)` with `hand` obtained by the command `[hand, rk]=lufact(A)` with `A` a sparse matrix returns four sparse matrices such that $P^*L^*U^*Q=A$.

The `A` matrix needs not be full rank but must be square (since `A` is assumed sparse one may add zeros if necessary to squaring down `A`).

If `A` is singular, the `L` matrix is column compressed (with `rk` independent nonzero columns): the nonsingular sparse matrix `Q'*inv(U)` column compresses `A`.

Examples

```
a=rand(5,2)*rand(2,5);A=sparse(a);
[hand,rk]=lufact(A);[P,L,U,Q]=luget(hand);
full(L), P*L*U*Q-A
clean(P*L*U*Q-A)
ludel(hand)
```

See Also

`sparse` , `lusolve` , `luget` , `clean`

Name

lusolve — sparse linear system solver

```
lusolve(hand,b)
lusolve(A,b)
```

Parameters

b
full real matrix

A
real square sparse invertible matrix

hand
handle to a previously computed sparse lu factors (output of lufact)

Description

`x=lusolve(hand,b)` solves the sparse linear system $A*x = b$.

`[hand,rk]=lufact(A)` is the output of lufact.

`x=lusolve(A,b)` solves the sparse linear system $A*x = b$

Examples

```
non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
[h,rk]=lufact(sp);x=lusolve(h,[1;1;1;1]);ludel(h)
rk,sp*x

non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
x=lusolve(sp,-ones(4,1));
sp*x
```

See Also

sparse , lufact , slash , backslash

Name

mtlb_sparse — convert sparse matrix

```
Y=mtlb_sparse(X)
```

Parameters

X
sparse matrix

Y
sparse matrix in Matlab format

Description

`Y=mtlb_sparse(X)` is used to convert `X`, a Scilab sparse matrix, to Matlab format. `Y` is the a variable with type 7, i.e. `type(Y)` is equal to 7. This function should be used in mexfiles (a Matlab mexfile containing sparse matrices can be used only if the Scilab sparse matrices are converted to that format). The functions `full` and `spget` work with this format.

Other operations and functions using this format can be overloaded with Scilab functions using the prefix "%msp". For instance the function `%msp_p(x)` (see SCIDIR/macros/percent directory) is used to display such "type 7" objects.

Examples

```
X=sparse(rand(2,2)); Y=mtlb_sparse(X);  
Y, full(Y), [ij,v,mn]=spget(Y)
```

See Also

`full`, `spget`

Name

nnz — number of non zero entries in a matrix

```
n=nnz(X)
```

Parameters

X
real or complex sparse (or full) matrix

n
integer, the number of non zero elements in X

Description

nnz counts the number of non zero entries in a sparse or full matrix

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3]);  
nnz(sp)  
a=[1 0 0 0 2];  
nnz(a)
```

See Also

spget

Name

pcg — preconditioned conjugate gradient

```
[x, flag, err, iter, res] = pcg(A, b [, tol [, maxIter [, M [, M2 [, x0 [, verbose]]]]]]
[x, flag, err, iter, res] = pcg(A, b [key=value,...])
```

Parameters

A,

a matrix, or a function, or a list computing $A \cdot x$ for each given x . The following is a description of the computation of $A \cdot x$ depending on the type of A.

- **matrix**. If A is a matrix, it can be dense or sparse
- **function**. If A is a function, it must have the following header :

```
function y = A ( x )
```

- **list**. If A is a list, the first element of the list is expected to be a function and the other elements in the list are the arguments of the function, from index 2 to the end. When the function is called, the current value of x is passed to the function as the first argument. The other arguments passed are the one given in the list.

b

right hand side vector (size: $n \times 1$)

tol

error relative tolerance (default: $1e-8$). The termination criteria is based on the 2-norm of the residual $r=b-Ax$, divided by the 2-norm of the right hand side b.

maxIter

maximum number of iterations (default: n)

M

preconditioner: full or sparse matrix or function returning $M \backslash x$ (default: none)

M2

preconditioner: full or sparse matrix or function returning $M2 \backslash x$ for each x (default: none)

x0

initial guess vector (default: `zeros(n,1)`)

verbose

set to 1 to enable verbose logging (default 0)

x

solution vector

flag

0 if pcg converged to the desired tolerance within `maxi` iterations, 1 else

err

final relative norm of the residual (the 2-norm of the right-hand side b is used)

iter

number of iterations performed

res

vector of the residual relative norms

Description

Solves the linear system $Ax=b$ using the conjugate gradient method with or without preconditioning. The preconditioning should be defined by a symmetric positive definite matrix M , or two matrices $M1$ and $M2$ such that $M=M1*M2$, in the case the function solves $\text{inv}(M)*A*x = \text{inv}(M)*b$ for x . M , $M1$ and $M2$ can be Scilab functions with calling sequence $y=M1*x(x)$ which computes the corresponding left division $y=M1\backslash x$.

The A matrix must be a symmetric positive definite matrix (full or sparse) or a function with calling sequence $y=A(x)$ which computes $y=A*x$

Example with well conditioned and ill conditioned problems

In the following example, two linear systems are solved. The first matrix has a condition number equals to ~ 0.02 , which makes the algorithm converge in exactly 10 iterations. Since this is the size of the matrix, it is an expected behaviour for a gradient conjugate method. The second one has a low condition number equals to $1.d-6$, which makes the algorithm converge in a larger 22 iterations. This is why the parameter `maxIter` is set to 30. See below for other examples of the "key=value" syntax.

```
//Well conditioned problem
A=[ 94  0  0  0  0  28  0  0  32  0
    0  59 13  5  0  0  0 10  0  0
    0  13 72 34  2  0  0  0  0 65
    0  5 34 114 0  0  0  0  0 55
    0  0 2  0 70  0 28 32 12  0
   28  0 0  0  0 87 20  0 33  0
    0  0 0  0 28 20 71 39  0  0
    0 10 0  0 32  0 39 46  8  0
   32  0 0  0 12 33  0  8 82 11
    0  0 65 55  0  0  0  0 11 100];

b=ones(10,1);
[x, fail, err, iter, res]=pcg(A,b,1d-12,15);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

//Ill conditioned one
A=[ 894  0  0  0  0  28  0  0 1000 70000
    0  5 13  5  0  0  0  0  0  0
    0  13 72 34  0  0  0  0  0 6500
    0  5 34  1  0  0  0  0  0 55
    0  0 0  0 70  0 28 32 12  0
   28  0 0  0  0 87 20  0 33  0
    0  0 0  0 28 20 71 39  0  0
    0  0 0  0 32  0 39 46  8  0
  1000  0 0  0 12 33  0  8 82 11
 70000 0 6500 55  0  0  0  0 11 100];

[x, fail, err, iter, res]=pcg(A,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

Examples with A given as a sparse matrix, or function, or list

The following example shows that the method can handle sparse matrices as well. It also shows the case where a function, computing the right-hand side, is given to the "pcg" primitive. The final case shown by this example, is when a list is passed to the primitive.

```
//Well conditioned problem
A=[ 94  0  0  0  0  28  0  0  32  0
    0  59 13  5  0  0  0  10  0  0
    0  13 72 34  2  0  0  0  0  65
    0  5  34 114 0  0  0  0  0  55
    0  0  2  0  70 0  28 32 12  0
   28  0  0  0  0  87 20  0  33  0
    0  0  0  0  28 20 71 39  0  0
    0 10  0  0  32  0 39 46  8  0
   32  0  0  0  12 33  0  8  82 11
    0  0 65 55  0  0  0  0  11 100];
b=ones(10,1);

// Convert A into a sparse matrix
Asparse=sparse(A);
[x, fail, err, iter, res]=pcg(Asparse,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesx(x)
    A=[ 94  0  0  0  0  28  0  0  32  0
        0  59 13  5  0  0  0  10  0  0
        0  13 72 34  2  0  0  0  0  65
        0  5  34 114 0  0  0  0  0  55
        0  0  2  0  70 0  28 32 12  0
       28  0  0  0  0  87 20  0  33  0
        0  0  0  0  28 20 71 39  0  0
        0 10  0  0  32  0 39 46  8  0
       32  0  0  0  12 33  0  8  82 11
        0  0 65 55  0  0  0  0  11 100];
    y=A*x
endfunction

// Pass the script Atimesx to the primitive
[x, fail, err, iter, res]=pcg(Atimesx,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesxbis(x,A)
    y=A*x
endfunction

// Pass a list to the primitive
Alist = list(Atimesxbis,Asparse);
[x, fail, err, iter, res]=pcg(Alist,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

Examples with key=value syntax

The following example shows how to pass arguments with the "key=value" syntax. This allows to set non-positionnal arguments, that is, to set arguments which are not depending on their order in the list of arguments. The available keys are the names of the optional arguments, that is : tol, maxIter, %M, %M2, x0, verbose. Notice that, in the following example, the verbose option is given before the maxIter option. Without the "key=value" syntax, the positionnal arguments would require that maxIter come first and verbose after.

```
// Example of an argument passed with key=value syntax
A=[100,1;1,10];
b=[101;11];
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1);

// With key=value syntax, the order does not matter
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1,maxIter=0);
```

See Also

backslash, qmr, gmres

Authors

Sage Group, IRISA, 2004

Serge Steer, INRIA, 2006

Michael Baudin, INRIA, 2008-2009

References

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993, <ftp://netlib2.cs.utk.edu/linalg/templates.ps>

"Iterative Methods for Sparse Linear Systems, Second Edition", Saad, SIAM Publications, 2003, ftp://cs.umn.edu/dept/users/saad/PS/all_ps.zip

Name

qmr — quasi minimal residual method with preconditioning

```
[x,flag,err,iter,res] = qmr(A,b,x0,M1,M1p,M2,M2p,maxi,tol)
```

Parameters

A
matrix of size n-by-n or function returning $A \cdot x$

b
right hand side vector

x0
initial guess vector (default: zeros(n,1))

M1
left preconditioner: matrix or function returning $M1 \cdot x$ (In the first case, default: eye(n,n))

M1p
must only be provided when **M1** is a function. In this case **M1p** is the function which returns $M1' \cdot x$

M2
right preconditioner: matrix or function returning $M2 \cdot x$ (In the first case, default: eye(n,n))

M2p
must only be provided when **M2** is a function. In this case **M2p** is the function which returns $M2' \cdot x$

maxi
maximum number of iterations (default: n)

tol
error tolerance (default: 1000*%eps)

x
solution vector

flag
0 =
 gmres converged to the desired tolerance within maxi iterations
1 =
 no convergence given maxi

res
residual vector

err
final residual norm

iter
number of iterations performed

Description

Solves the linear system $Ax=b$ using the Quasi Minimal Residual Method with preconditioning.

See Also

gmres

Authors

SAGE Group, IRISA 2005

Name

sparse — sparse matrix definition

```
sp=sparse(X)
sp=sparse(ij,v [,mn])
```

Parameters

X
real or complex full (or sparse) matrix

ij
two columns integer matrix (indices of non-zeros entries)

v
vector

mn
integer vector with two entries (row-dimension, column-dimension)

sp
sparse matrix

Description

`sparse` is used to build a sparse matrix. Only non-zero entries are stored.

`sp = sparse(X)` converts a full matrix to sparse form by squeezing out any zero elements. (If `X` is already sparse `sp` is `X`).

`sp=sparse(ij,v [,mn])` builds an `mn(1)`-by-`mn(2)` sparse matrix with `sp(ij(k,1),ij(k,2))=v(k)`. `ij` and `v` must have the same column dimension. If optional `mn` parameter is not given the `sp` matrix dimensions are the max value of `ij(:,1)` and `ij(:,2)` respectively.

Operations (concatenation, addition, etc.) with sparse matrices are made using the same syntax as for full matrices.

Elementary functions are also available (`abs`, `maxi`, `sum`, `diag`, ...) for sparse matrices.

Mixed operations (full-sparse) are allowed. Results are full or sparse depending on the operations.

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
size(sp)
x=rand(2,2);abs(x)-full(abs(sparse(x)))
```

See Also

`full` , `spget` , `sprand` , `speye` , `lufact`

Name

spchol — sparse cholesky factorization

```
[R,P] = spchol(X)
```

Parameters

X
symmetric positive definite real sparse matrix

P
permutation matrix

R
cholesky factor

Description

$[R,P] = \text{spchol}(X)$ produces a lower triangular matrix R such that $P^*R^*R'^*P' = X$.

Examples

```
X=[
3., 0., 0., 2., 0., 0., 2., 0., 2., 0., 0. ;
0., 5., 4., 0., 0., 0., 0., 0., 0., 0., 0. ;
0., 4., 5., 0., 0., 0., 0., 0., 0., 0., 0. ;
2., 0., 0., 3., 0., 0., 2., 0., 2., 0., 0. ;
0., 0., 0., 0., 5., 0., 0., 0., 0., 0., 4. ;
0., 0., 0., 0., 0., 4., 0., 3., 0., 3., 0. ;
2., 0., 0., 2., 0., 0., 3., 0., 2., 0., 0. ;
0., 0., 0., 0., 0., 3., 0., 4., 0., 3., 0. ;
2., 0., 0., 2., 0., 0., 2., 0., 3., 0., 0. ;
0., 0., 0., 0., 0., 3., 0., 3., 0., 4., 0. ;
0., 0., 0., 0., 4., 0., 0., 0., 0., 0., 5.];
X=sparse(X);[R,P] = spchol(X);
max(P*R*R'*P'-X)
```

See Also

sparse , lusolve , luget , chol

Name

spcompact — converts a compressed adjacency representation

Parameters

xadj
integer vector of length (n+1).

xlindx
integer vector of length n+1 (pointers).

lindx
integer vector

adjncy
integer vector

Description

Utility fonction spcompact is used to convert a compressed adjacency representation into standard adjacency representation.

Examples

```
// A is the sparse matrix:
A=[1,0,0,0,0,0,0;
   0,1,0,0,0,0,0;
   0,0,1,0,0,0,0;
   0,0,1,1,0,0,0;
   0,0,1,1,1,0,0;
   0,0,1,1,0,1,0;
   0,0,1,1,0,1,1];
A=sparse(A);
//For this matrix, the standard adjacency representation is given by:
xadj=[1,2,3,8,12,13,15,16];
adjncy=[1, 2, 3,4,5,6,7, 4,5,6,7, 5, 6,7, 7];
//(see sp2adj).
// increments in vector xadj give the number of non zero entries in each column
// ie there is 2-1=1 entry in the column 1
//      there is 3-2=1 entry in the column 2
//      there are 8-3=5 entries in the column 3
//              12-8=4                      4
//etc
//The row index of these entries is given by the adjncy vector
// for instance,
// adjncy (3:7)=adjncy(xadj(3):xadj(4)-1)=[3,4,5,6,7]
// says that the 5=xadj(4)-xadj(3) entries in column 3 have row
// indices 3,4,5,6,7.
//In the compact representation, the repeated sequences in adjncy
//are eliminated.
//Here in adjncy the sequences 4,5,6,7 and 7 are eliminated.
//The standard structure (xadj,adjncy) takes the compressed form (lindx,xlindx)
lindx=[1, 2, 3,4,5,6,7, 5, 6,7];
xlindx=[1,2,3,8,9,11];
```

```
//(Columns 4 and 7 of A are eliminated).  
//A can be reconstructed from (xadj,xlindx,lindx).  
[xadj,adjncy,anz]= sp2adj(A);  
adjncy-spcompact(xadj,xlindx,lindx)
```

See Also

sp2adj , adj2sp , spget

Name

`spget` — retrieves entries of sparse matrix

```
[ij, v, mn]=spget( sp)
```

Parameters

`sp`
real or complex sparse matrix

`ij`
two columns integer matrix (indices of non-zeros entries)

`mn`
integer vector with two entries (row-dimension, column-dimension)

`v`
column vector

Description

`spget` is used to convert the internal representation of sparse matrices into the standard `ij`, `v` representation.

Non zero entries of `sp` are located in rows and columns with indices in `ij`.

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3])  
[ij, v, mn]=spget( sp);
```

See Also

`sparse` , `sprand` , `speye` , `lufact`

Part XXX. Special Functions

Name

besseli — Modified Bessel functions of the first kind (I sub alpha).
besselj — Bessel functions of the first kind (J sub alpha).
besselk — Modified Bessel functions of the second kind (K sub alpha).
bessely — Bessel functions of the second kind (Y sub alpha).
besselh — Bessel functions of the third kind (aka Hankel functions)

```
y = besseli(alpha,x [,ice])
y = besselj(alpha,x [,ice])
y = besselk(alpha,x [,ice])
y = bessely(alpha,x [,ice])
y = besselh(alpha,x)
y = besselh(alpha,K,x [,ice])
```

Parameters

x
real or complex vector.

alpha
real vector

ice
integer flag, with default value 0

K
integer, with possible values 1 or 2, the Hankel function type.

Description

Warning: the semantics of these functions changes between Scilab-3.0 and Scilab-3.1. The old semantics is available for compatibility using the `oldbesseli`, `oldbesselj`, `oldbesselk`, `oldbessely` functions.

- `besseli(alpha,x)` computes modified Bessel functions of the first kind (I sub alpha), for real order `alpha` and argument `x`. `besseli(alpha,x,1)` computes `besseli(alpha,x) .* exp(-abs(real(x)))`.
- `besselj(alpha,x)` computes Bessel functions of the first kind (J sub alpha), for real order `alpha` and argument `x`. `besselj(alpha,x,1)` computes `besselj(alpha,x) .* exp(-abs(imag(x)))`.
- `besselk(alpha,x)` computes modified Bessel functions of the second kind (K sub alpha), for real order `alpha` and argument `x`. `besselk(alpha,x,1)` computes `besselk(alpha,x) .* exp(x)`.
- `bessely(alpha,x)` computes Bessel functions of the second kind (Y sub alpha), for real order `alpha` and argument `x`. `bessely(alpha,x,1)` computes `bessely(alpha,x) .* exp(-abs(imag(x)))`.
- `besselh(alpha [,K] ,x)` computes Bessel functions of the third kind (Hankel function H1 or H2 depending on K), for real order `alpha` and argument `x`. If omitted K is supposed to be equal to 1. `besselh(alpha,1,x,1)` computes `besselh(alpha,1,x) .* exp(-%i*x)` and `besselh(alpha,2,x,1)` computes `besselh(alpha,2,x) .* exp(%i*x)`

Remarks

If `alpha` and `x` are arrays of the same size, the result `y` is also that size. If either input is a scalar, it is expanded to the other input's size. If one input is a row vector and the other is a column vector, the result is a two-dimensional table of function values.

Y_α and J_α Bessel functions are 2 independant solutions of the Bessel 's differential equation :

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} + (x^2 - \alpha^2) \cdot y = 0, \alpha \geq 0$$

K_α and I_α modified Bessel functions are 2 independant solutions of the modified Bessel 's differential equation :

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} - (x^2 + \alpha^2) \cdot y = 0, \alpha \geq 0$$

H^1_α and H^2_α , the Hankel functions of first and second kind, are linear linear combinations of Bessel functions of the first and second kinds:

$$H^1_\alpha(z) = J_\alpha(z) + i \cdot Y_\alpha(z)$$

$$H^2_\alpha(z) = J_\alpha(z) - i \cdot Y_\alpha(z)$$

Examples

```
// besseli functions
// =====
x = linspace(0.01,10,5000)';
clf()
subplot(2,1,1)
plot2d(x,besseli(0:4,x), style=2:6)
legend('I'+string(0:4),2);
xlabel("Some modified Bessel functions of the first kind")
subplot(2,1,2)
plot2d(x,besseli(0:4,x,1), style=2:6)
legend('I'+string(0:4),1);
xlabel("Some modified scaled Bessel functions of the first kind")

// besselj functions
// =====
x = linspace(0,40,5000)';
clf()
plot2d(x,besselj(0:4,x), style=2:6, leg="J0@J1@J2@J3@J4")
legend('I'+string(0:4),1);
xlabel("Some Bessel functions of the first kind")

// use the fact that J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
// to compare the algorithm of besselj(0.5,x) with a more direct formula
x = linspace(0.1,40,5000)';
y1 = besselj(0.5, x);
y2 = sqrt(2 ./(%pi*x)).*sin(x);
er = abs((y1-y2)./y2);
ind = find(er > 0 & y2 ~= 0);
clf()
subplot(2,1,1)
plot2d(x,y1,style=2)
xlabel("besselj(0.5,x)")
```

```

subplot(2,1,2)
plot2d(x(ind), er(ind), style=2, logflag="nl")
xtitle("relative error between 2 formulae for besselj(0.5,x)")

// bessell functions
// =====
x = linspace(0.01,10,5000)';
clf()
subplot(2,1,1)
plot2d(x,bessell(0:4,x), style=0:4, rect=[0,0,6,10])
legend('K'+string(0:4),1);
xtitle("Some modified Bessel functions of the second kind")
subplot(2,1,2)
plot2d(x,bessell(0:4,x,1), style=0:4, rect=[0,0,6,10])
legend('K'+string(0:4),1);
xtitle("Some modified scaled Bessel functions of the second kind")

// bessely functions
// =====
x = linspace(0.1,40,5000)'; // Y Bessel functions are unbounded for x ->
clf()
plot2d(x,bessely(0:4,x), style=0:4, rect=[0,-1.5,40,0.6])
legend('Y'+string(0:4),4);
xtitle("Some Bessel functions of the second kind")

// besselh functions
// =====
x=-4:0.025:2; y=-1.5:0.025:1.5;
[X,Y] = ndgrid(x,y);
H = besselh(0,1,X+%i*Y);
clf();f=gcf();
xset("fpf"," ")
f.color_map=jetcolormap(16);
contour2d(x,y,abs(H),0.2:0.2:3.2,strf="034",rect=[-4,-1.5,3,1.5])
legends(string(0.2:0.2:3.2),1:16,"ur")
xtitle("Level curves of |H1(0,z)|")

```

Authors

Amos, D. E., (SNLA)
 Daniel, S. L., (SNLA)
 Weston, M. K., (SNLA)

Used Functions

The source codes can be found in SCI/modules/special_functions/src/fortran/slatec and SCI/modules/special_functions/src/fortran

Slatec : dbesi.f, zbesi.f, dbesj.f, zbesj.f, dbesk.f, zbesk.f, dbesy.f, zbesy.f, zbesh.f

Drivers to extend definition area (Serge Steer INRIA): dbesig.f, zbesig.f, dbesjg.f, zbesjg.f, dbeskg.f, zbeskg.f, dbesyg.f, zbesyg.f, zbeshg.f

Name

beta — beta function

```
z = beta(x,y)
```

Parameters

x, y

2 positive reals or 2 matrices (or vectors) of positive reals of same size.

z

a real or a matrix of the same size than x with $z(i,j) = \text{beta}(x(i,j), y(i,j))$.

Description

Computes the complete beta function :

$$B(x,y) = \int_0^1 t^{x-1} \cdot (1-t)^{y-1} \cdot dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

For small x and y the algorithm uses the expression in function of the gamma function, else it applies the exponential function onto the result of the `beta ln` function provided with the DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameter (see `cdfbet` for more information about DCDFLIB).

Examples

```
// example 1 :
beta(5,2) - beta(2,5)    // symetry (must be exactly 0)
beta(0.5,0.5)            // exact value is pi

// example 2 : an error study based on the relation B(1,x) = 1/x
// (computing 1/x must lead to only a relative error of eps_m, so
// it may be used near as a reference to evaluate the error in B(1,x))
x = logspace(-8,8,20000)';
e = beta(ones(x),x) - (1)./x;
er = abs(e) .* x;
ind = find(er ~= 0);
eps = ones(x(ind))*number_properties("eps");
clf()
plot2d(x(ind),[er(ind) eps 2*eps],style=[1 2 3],logflag="ll",leg="er@eps_m@2 eps")
xlabel("approximate relative error in computing beta(1,x)")
xselect()

// example 3 : plotting the beta function
t = linspace(0.2,10,60);
X = t'*ones(t); Y = ones(t')*t;
Z = beta(X,Y);
clf()
plot3d(t, t, Z, flag=[2 4 4], leg="x@y@z", alpha=75, theta=30)
xlabel("The beta function on [0.2,10]x[0.2,10]")
xselect()
```

See Also

gamma, cdfbet

Name

`calerf` — computes error functions.

Parameters

`x`
real vector or matrix

`flag`
integer indicator

`y`
real vector or matrix (of same size than `x`)

Description

`calerf(x, 0)` computes the error function `erf(x)`

`calerf(x, 1)` computes the complementary error function `erfc(x)`

`calerf(x, 2)` computes the scaled complementary error function `erfcx(x)`

Examples

```
deff('y=f(t)', 'y=exp(-t^2)');  
calerf(1, 0)  
2/sqrt(%pi)*intg(0, 1, f)
```

See Also

`erf`, `erfc`, `erfcx`

Authors

W. J. Cody (code from Netlib (specfun))

Name

dlgamma — derivative of gammaln function, psi function

```
y = dlgamma(x)
```

Parameters

x

real vector

y

real vector with same size.

Description

dlgamma(x) evaluates, at all the elements of x the logarithmic derivative of the gamma function which corresponds also to the derivative of the gammaln function :

$$\frac{1}{\Gamma(x)} \frac{d\Gamma(x)}{dx} = \frac{d}{dx}(\ln(\Gamma(x)))$$

x must be real. Also known as the psi function.

Examples

```
dlgamma(0.5)
```

See Also

gamma, gammaln

Authors

W. J. Cody (code from Netlib (specfun))

Name

erf — The error function.

```
y = erf(x)
```

Parameters

x
real vector or matrix

y
real vector or matrix (of same size than x)

Description

erf computes the error function:

$$y = \frac{2}{\sqrt{\pi}} \cdot \int_0^x e^{-t^2} \cdot dt$$

Examples

```
deff('y=f(t)', 'y=exp(-t^2)');  
erf(0.5)-2/sqrt(%pi)*intg(0,0.5,f)
```

See Also

erfc, erfcx, calerf, cdfnor

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfc — The complementary error function.

```
y = erfc(x)
```

Parameters

x
real vector or matrix

y
real vector or matrix (of same size than x)

Description

erfc computes the complementary error function:

$$y = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} \cdot dt$$
$$y = 1 - \text{erf}(x)$$

Examples

```
erf([0.5,0.2])+erfc([0.5,0.2])
```

See Also

erf, erfcx, calerf

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfcx — scaled complementary error function.

```
y = erfcx(x)
```

Parameters

x
real vector or matrix

y
real vector or matrix (of same size than x)

Description

erfcx computes the scaled complementary error function:

$$y = \exp(x^2) \cdot \operatorname{erfc}(x)$$
$$y \longrightarrow \frac{1}{x\sqrt{\pi}} \text{ when } x \rightarrow +\infty$$

See Also

erf, erfc, calerf

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfinv — The inverse of the error function.

```
y = erfinv(x)
```

Parameters

x
real vector or matrix

y
real vector or matrix (of same size than x)

Description

erfinv computes the inverse of the error function erf. $x = \text{erfinv}(y)$ satisfies $y = \text{erf}(x)$, $-1 \leq y < 1$, $-\infty \leq x \leq \infty$.

Examples

```
x=linspace(-0.99,0.99,100);  
y=erfinv(x);  
plot2d(x,y)  
norm(x-erf(y),'inf')
```

See Also

erfc , cdfnor

References

Milton Abramowitz and Irene A. Stegun, eds. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover, 1972.

Name

gamma — The gamma function.

```
y = gamma(x)
```

Parameters

x
real vector or matrix

y
real vector or matrix with same size than x.

Description

gamma(x) evaluates the gamma function at all the elements of x. The gamma function is defined by :

$$\Gamma(x) = \int_0^{+\infty} t^{x-1} \cdot e^{-t} \cdot dt$$

and generalizes the factorial function for real numbers (gamma(n+1) = n!).

Examples

```
// simple examples
gamma(0.5)
gamma(6)-prod(1:5)

// the graph of the Gamma function on [a,b]
a = -3; b = 5;
x = linspace(a,b,40000)';
y = gamma(x);
clf()
c=xget("color")
xset("color",2)
plot2d(x, y, style=0, axesflag=5, rect=[a, -10, b, 10])
xset("color",c)
xtitle("The gamma function on [" + string(a) + ", " + string(b) + "]")
xselect()
```

See Also

gammaln, dlgamma

Authors

W. J. Cody and L. Stoltz (code from Netlib (specfun))

Name

`gammaln` — The logarithm of gamma function.

```
y = gammaln(x)
```

Parameters

`x`
real vector

`y`
real vector with same size.

Description

`gammaln(x)` evaluates the logarithm of gamma function at all the elements of `x`, avoiding underflow and overflow. `x` must be real.

Examples

```
gammaln(0.5)
```

See Also

`gamma`, `dlgamma`

Authors

W. J. Cody and L. Stoltz (code from Netlib (`specfun`))

Name

legendre — associated Legendre functions

```
y = legendre(n,m,x [,normflag])
```

Parameters

n
non negative integer or vector of non negative integers regularly spaced with increment equal to 1

m
non negative integer or vector of non negative integers regularly spaced with increment equal to 1

x
real (row) vector (elements of x must be in the $(-1, 1)$ interval)

normflag
(optional) scalar string

Description

When n and m are scalars, `legendre(n,m,x)` evaluates the associated Legendre function $P_n^m(x)$ at all the elements of x. The definition used is :

$$P_n^m(x) = (-1)^m \cdot (1-x^2)^{m/2} \cdot \frac{d^m}{dx^m} P_n(x)$$

where P_n is the Legendre polynomial of degree n. So `legendre(n,0,x)` evaluates the Legendre polynomial $P_n(x)$ at all the elements of x.

When the normflag is equal to "norm" you get a normalized version (without the $(-1)^m$ factor), precisely :

$$P_n^m(x, \text{norm}) = \sqrt{\frac{2n+1}{2} \cdot \frac{(n-m)!}{(n+m)!}} \cdot (1-x^2)^{m/2} \cdot \frac{d^m}{dx^m} P_n(x)$$

which is useful to compute spherical harmonic functions (see Example 3):

For efficiency, one of the two first arguments may be a vector, for instance `legendre(n1:n2,0,x)` evaluates all the Legendre polynomials of degree $n1, n1+1, \dots, n2$ at the elements of x and `legendre(n,m1:m2,x)` evaluates all the Legendre associated functions P_n^m for $m=m1, m1+1, \dots, m2$ at x.

Output format

In any case, the format of y is :

```
max(length(n),length(m)) x length(x)
```

and :

```
y(i,j) = P(n(i),m;x(j))    if n is a vector  
y(i,j) = P(n,m(i);x(j))    if m is a vector
```

```
y(1,j) = P(n,m;x(j))      if both n and m are scalars
```

so that x is preferably a row vector but any $m \times x \times n$ matrix is accepted and considered as an $1 \times (m \times n)$ matrix, reshaped following the column order.

Examples

```
// example 1 : plot of the 6 first Legendre polynomials on (-1,1)
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(0:5, 0, x);
clf()
plot2d(x,y', leg="p0@p1@p2@p3@p4@p5@p6")
xlabel("the 6 th first Legendre polynomials")

// example 2 : plot of the associated Legendre functions of degree 5
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(5, 0:5, x, "norm");
clf()
plot2d(x,y', leg="p5,0@p5,1@p5,2@p5,3@p5,4@p5,5")
xlabel("the (normalised) associated Legendre functions of degree 5")

// example 3 : define then plot a spherical harmonic
// 3-1 : define the function Ylm
function [y] = Y(l,m,theta,phi)
    // theta may be a scalar or a row vector
    // phi may be a scalar or a column vector
    if m >= 0 then
        y = (-1)^m/(sqrt(2*pi))*exp(i*m*phi)*legendre(l, m, cos(theta), "norm")
    else
        y = 1/(sqrt(2*pi))*exp(i*m*phi)*legendre(l, -m, cos(theta), "norm")
    end
endfunction

// 3.2 : define another useful function
function [x,y,z] = sph2cart(theta,phi,r)
    // theta row vector      1 x nt
    // phi   column vector   np x 1
    // r      scalar or np x nt matrix (r(i,j) the length at phi(i) theta(j))
    x = r.*(cos(phi)*sin(theta));
    y = r.*(sin(phi)*sin(theta));
    z = r.*(ones(phi)*cos(theta));
endfunction

// 3-3 plot Y31(theta,phi)
l = 3; m = 1;
theta = linspace(0.1,%pi-0.1,60);
phi = linspace(0,2*pi,120)';
f = Y(l,m,theta,phi);
[x1,y1,z1] = sph2cart(theta,phi,abs(f));      [xf1,yf1,zf1] = nf3d(x1,y1,z1);
[x2,y2,z2] = sph2cart(theta,phi,abs(real(f))); [xf2,yf2,zf2] = nf3d(x2,y2,z2);
[x3,y3,z3] = sph2cart(theta,phi,abs(imag(f))); [xf3,yf3,zf3] = nf3d(x3,y3,z3);

clf()
subplot(1,3,1)
```

```
plot3d(xf1,yf1,zf1,flag=[2 4 4]); xtitle("|Y31(theta,phi)|")
subplot(1,3,2)
plot3d(xf2,yf2,zf2,flag=[2 4 4]); xtitle("|Real(Y31(theta,phi))|")
subplot(1,3,3)
plot3d(xf3,yf3,zf3,flag=[2 4 4]); xtitle("|Imag(Y31(theta,phi))|")
```

Authors

Smith, John M. (code dxlegf.f from Slatec)
 B. Pincon (scilab interface)

Name

oldbesseli — Modified Bessel functions of the first kind (I sub alpha).
oldbesselj — Bessel functions of the first kind (J sub alpha).
oldbesselk — Modified Bessel functions of the second kind (K sub alpha).
oldbessely — Bessel functions of the second kind (Y sub alpha).

```
y = oldbesseli(alpha,x)
y = oldbesseli(alpha,x,ice)
y = oldbesselj(alpha,x)
y = oldbesselk(alpha,x)
y = oldbesselk(alpha,x,ice)
y = oldbessely(alpha,x)
```

Parameters

x
real vector with non negative entries

alpha
real vector with non negative entries regularly spaced with increment equal to one
alpha=alpha0+(n1:n2)

ice
integer flag, with default value 1

Description

These functions are obsolete, use besseli, besselj, besselk, bessely instead. Note however that the semantics of these two sets of functions are different.

oldbesseli(alpha,x) computes modified Bessel functions of the first kind (I sub alpha), for real, non-negative order alpha and real non negative argument x. besseli(alpha,x,2) computes besseli(alpha,x).*exp(-x).

oldbesselj(alpha,x) computes Bessel functions of the first kind (J sub alpha), for real, non-negative order alpha and real non negative argument x.

oldbesselk(alpha,x) computes modified Bessel functions of the second kind (K sub alpha), for real, non-negative order alpha and real non negative argument x. besselk(alpha,x,2) computes besselk(alpha,x).*exp(x).

oldbessely(alpha,x) computes Bessel functions of the second kind (Y sub alpha), for real, non-negative order alpha and real non negative argument x.

alpha and x may be vectors. The output is m-by-n with m = size(x,''), n = size(alpha,'') whose (i,j) entry is oldbessel?(alpha(j),x(i)).

Remarks

Y_alpha and J_alpha Bessel functions are 2 independant solutions of the Bessel 's differential equation :

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} + (x^2 - \alpha^2) \cdot y = 0, \alpha \geq 0$$

K_α and I_α modified Bessel functions are 2 independant solutions of the modified Bessel's differential equation :

$$x^2 \cdot \frac{d^2 y}{dx^2} + x \cdot \frac{dy}{dx} - (x^2 + \alpha^2) \cdot y = 0, \alpha \geq 0$$

Examples

```
// example #1: display some I Bessel functions
x = linspace(0.01,10,5000)';
y = oldbesseli(0:4,x);
ys = oldbesseli(0:4,x,2);
clf()
subplot(2,1,1)
plot2d(x,y, style=2:6, leg="I0@I1@I2@I3@I4", rect=[0,0,6,10])
xlabel("Some modified Bessel functions of the first kind")
subplot(2,1,2)
plot2d(x,ys, style=2:6, leg="I0s@I1s@I2s@I3s@I4s", rect=[0,0,6,1])
xlabel("Some modified scaled Bessel functions of the first kind")

// example #2 : display some J Bessel functions
x = linspace(0,40,5000)';
y = besselj(0:4,x);
clf()
plot2d(x,y, style=2:6, leg="J0@J1@J2@J3@J4")
xlabel("Some Bessel functions of the first kind")

// example #3 : use the fact that J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
// to compare the algorithm of besselj(0.5,x) with
// a more direct formula
x = linspace(0.1,40,5000)';
y1 = besselj(0.5, x);
y2 = sqrt(2 ./(%pi*x)).*sin(x);
er = abs((y1-y2)./y2);
ind = find(er > 0 & y2 ~= 0);
clf()
subplot(2,1,1)
plot2d(x,y1,style=2)
xlabel("besselj(0.5,x)")
subplot(2,1,2)
plot2d(x(ind), er(ind), style=2, logflag="nl")
xlabel("relative error between 2 formulae for besselj(0.5,x)")

// example #4: display some K Bessel functions
x = linspace(0.01,10,5000)';
y = besserk(0:4,x);
ys = besserk(0:4,x,1);
clf()
subplot(2,1,1)
plot2d(x,y, style=0:4, leg="K0@K1@K2@K3@K4", rect=[0,0,6,10])
xlabel("Some modified Bessel functions of the second kind")
subplot(2,1,2)
plot2d(x,ys, style=0:4, leg="K0s@K1s@K2s@K3s@K4s", rect=[0,0,6,10])
xlabel("Some modified scaled Bessel functions of the second kind")

// example #5: plot severals Y Bessel functions
```

```
x = linspace(0.1,40,5000)'; // Y Bessel functions are unbounded for x -> 0+
y = bessely(0:4,x);
clf()
plot2d(x,y, style=0:4, leg="Y0@Y1@Y2@Y3@Y4", rect=[0, -1.5, 40, 0.6])
xlabel("Some Bessel functions of the second kind")
```

Authors

W. J. Cody, L. Stoltz (code from Netlib (specfun))

Part XXXI. Strings

Name

ascii — string ascii conversions

```
a=ascii(txt)
txt=ascii(a)
```

Parameters

txt
A character string or a matrix of character strings.

a
A vector of integer ascii codes

Description

This function convert Scilab string to a vector of ascii code (the first 127 codes are ASCII) or vector of ascii code to Scilab strings.

If `txt` is a matrix of string, `ascii(txt)` is equivalent to `ascii(strcat(txt))`

Examples

```
ascii(["hello";"world"])
ascii("scilab")
ascii([115 99 105 108 97 98])
```

See Also

`code2str`, `str2code`

Name

blanks — Create string of blank characters

```
txt=blanks(n)
```

Parameters

txt

A single character string

n

number of blanks

Description

blanks(n) is a string of n blanks.

Examples

```
disp(['xxx' blanks(20) 'yyy'])
```

Name

`code2str` — returns character string associated with Scilab integer codes.

```
str=code2str(c)
```

Parameters

`str`

A character string

`c`

vector of character integer codes

Description

Returns character string associated with Scilab integer codes.`str` is such that `c(i)` is the Scilab integer code of `part(str,i)`

Examples

```
code2str([-28 12 18 21 10 11])  
str2code('Scilab')
```

See Also

`str2code`, `ascii`

Name

convstr — case conversion

```
[y]=convstr(str, [flag])
```

Parameters

str, y

A matrix of character strings

flag

A character option with possible values

'u'

for upper

'l'

for lower

Description

converts the matrix of strings `str-matrix` into lower case (for "l" ;default value) or upper case (for "u").

Examples

```
A=['this','is';'my','matrix'];  
convstr(A,'u')
```

Name

emptystr — zero length string

```
s=emptystr()  
s=emptystr(a)  
s=emptystr(m,n)
```

Parameters

a	Any type of matrix
s	A matrix of character strings
m,n	Integers

Description

Returns a matrix of zero length character strings

With no input argument returns a zero length character string.

With a matrix for input argument returns a zero length character strings matrix of the same size.

With two integer arguments returns a mxn zero length character strings matrix

Examples

```
x=emptystr();for k=1:10, x=x+', '+string(k);end
```

See Also

part , length , string

Name

grep — find matches of a string in a vector of strings

```
row=grep(haystack,needle )
[row,which]=grep(haystack,needle )
row=grep(haystack,needle ,[flag])
[row,which]=grep(haystack,needle ,[flag])
```

Parameters

haystack

A Row vector of character strings.

needle

A character string or a row vector of character strings . The string(s) to search in **haystack** .

row

vector of indices: row where a match has been found or an empty matrix if no match found.

which

vector of indices: index of needle string found or an empty matrix if no match found.

flag

Character ("r" for regular expression)

Description

Foreach entry of **haystack** , **grep** searches if at least a string in **needle** which matches a substring. The **haystack** entries index where at least a match has been found are returned in the **row** output argument. The optionnal **which** output argument gives the index of first string of **needle** found.

When using the third parameter **flag="r"**, the **needle** is expected to be a regular expression string. In this case, **grep** uses the **needle** as a regular expression and compares it against **haystack** according to the regular express rules. See the **regexp** function for details about regular expressions.

Example #1

In the following example, we search one or two strings in a text, which is stored in the **txt** variable.

```
txt=['find matches of a string in a vector of strings'
     'search position of a character string in an other string'
     'Compare Strings'];

grep(txt,'strings')
grep(txt,['strings' 'Strings'])

[r,w]=grep(txt,['strings' 'Strings'])
```

Example #2

In the following example, we perform regexp searches.

```
str = ["hat";"cat";"hhat";"chat";"hcat";"ccchat";"at";"dog"]
```

```
grep(str, '/[hc]+at/', 'r')  
grep(str, '/[hc]?at/', 'r')  
grep(str, '/cat|dog/', 'r')
```

See Also

[strindex](#), [regexp](#)

Name

isalphanum — check that characters of a string are alphanumerics

```
res = isalphanum(str)
```

Parameters

str
A character string.

res
A boolean matrix.

Description

`res = isalphanum(str)` returns an array the same size as `str` containing logical %t (true) where the elements of `str` are alphanumerics and logical %f (false) where they are not.

Examples

```
s = 'A1,B2,C3';  
isalphanum(s)
```

See Also

isletter , isdigit

Name

isascii — tests if character is a 7-bit US-ASCII character

```
res = isascii(str)
```

Parameters

str
A character string.

res
A Boolean matrix.

Description

`res = isascii(str)` returns TRUE (%T) if `c` is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive, otherwise returns FALSE (%F)

Examples

```
isascii(code2str(300))
isascii(code2str(-11))
letters = [115.    99.    105.    108.    97.    98.]
isascii(letters)
ascii(letters)
isascii('scilab')
```

See Also

isalphanum

Name

isdigit — check that characters of a string are digits between 0 and 9

```
res = isdigit(str)
```

Parameters

str
A character string.

res
A boolean matrix.

Description

`res = isdigit(str)` returns an array the same size as `str` containing logical %T (TRUE) where the elements of `str` are digits and logical %F (FALSE) where they are not.

Examples

```
s = 'A1,B2,C3';  
isdigit(s)
```

See Also

isalphanum , isletter

Name

isletter — check that characters of a string are alphabetic letters

```
res = isletter(str)
```

Parameters

str
A character string.

res
A boolean matrix.

Description

`res = isletter(str)` returns an array the same size as `str` containing logical %t (true) where the elements of `str` are letters of the alphabet and logical %f (false) where they are not.

Examples

```
s = 'A1,B2,C3';  
isletter(s)
```

See Also

isalphanum , isdigit

Name

isnum — tests if a string represents a number

```
res = isnum(str)
```

Parameters

str

A character string or a matrix of character strings.

res

A boolean matrix.

Description

`res = isnum(str)` returns %T if str represents a number

Examples

```
isnum(['1'      , ..  
      '-1.23' , ..  
      '+1e+23' , ..  
      '1d+23'  , ..  
      '%pi'   ])
```

See Also

isletter , isdigit , isalphanum

Authors

P.M

Name

`justify` — Justify character array.

```
Tj=justify(T,opt)
```

Parameters

- T**
A matrix of character string.
- Tj**
A matrix of character string, the justified result
- opt**
A character option with possible values
- 'r'
or 'right' for right justification
 - 'l'
or 'left' for left justification
 - 'c'
or 'center' for centering justification

Description

`justify` justify the column of a matrix of string according to the given option.

Examples

```
t=['1234','x','adfdfgdfghfgj'  
  '1','354556','dgf'  
  'sdfgd','','sdfsfs'];  
  
justify(t,'l')  
justify(t,'c')  
justify(t,'r')
```

See Also

`length` , `part`

Name

length — length of object

```
n=length(M)
```

Parameters

M
matrix (usual or polynomial or character string) or list

n
integer or integer matrix

Description

For usual or polynomial matrix **n** is the integer equal to number of rows times number of columns of M. (Also valid for M a boolean matrix)

For matrices made of character strings (and in particular for a character string) **length** returns in **n** the length of entries of the matrix of character strings M.

The length of a list is the number of elements in the list (also given by **size**).

`length('123')` is 3. `length([1,2;3,4])` is 4.

WARNING : length of a sparse matrix returns the max of dimensions and not the product of the dimensions. (example : `length(sparse(eye(12,2)))` returns `max(12,2)` and not 24)

please use `size(...,'*')` with sparse matrix.

Examples

```
length([123 ; 456 ])
length(['hello world',SCI])
```

See Also

[size](#)

Name

part — extraction of strings

```
[strings_out] = part(strings_in, v)
```

Parameters

strings_in, strings_out
Matrix of character strings.

v
Integer row vector.

Description

Let $S[k]$ stands for the k character of string S (or the white space character if $k > \text{length}(S)$).

part returns strings_out, a matrix of character strings, such that $\text{strings\_out}(i, j)$ is the string $s[v(1)] \dots s[v(n)]$ ($s = \text{strings\_in}(i, j)$).

Examples

```
// returns characters position 8 to 11
part("How to use ""part"" ?",8:11)

// returns characters position 2 to 4 for each element
// no characters replaced by ''
c = part(['a', 'abc', 'abcd'],2:4)

// returns character position 1 for each element and add characters position 4
c = part(['abcdefg', 'hijklmn', 'opqrstu'],[1,4:7]);

// returns character 4 for each element, add characters position 1 to 7 and add
c = part(['abcdefg', 'hijklmn', 'opqrstu'],[4,1:7,4]);

// returns character position 1,add again character position 1 and character po
c=part(['a', 'abc', 'abcd'],[1,1,2])

// a a a
part(['a', 'abc', 'abcd'],[1])

// aa aa aa
part(['a', 'abc', 'abcd'],[1,1])

// aa aab aab
part(['a', 'abc', 'abcd'],[1,1,2])
```

See Also

string, length

Name

`regexp` — find a substring that matches the regular expression string

```
[start]=regexp(input,pattern,[flag])  
[start,end,match]=regexp(input,pattern,[flag])  
[start,end]=regexp(input,pattern,[flag])  
[start,end,match]=regexp(input,pattern,[flag])
```

Parameters

`input`

a string.

`pattern`

a character string (under the rules of regular expression)

`start`

the starting index of each substring of `str` that matches the regular expression string `pattern`

`end`

the ending index of each substring of `str` that matches the regular expression string `pattern`

`match`

the text of each substring of `str` that matches `pattern`.

`[flag]`

'o' for matching the pattern once .

Description

The rules of regular expression are similar to perl language. For a quick start , see <http://perldoc.perl.org/perlrequick.html>. For a more in-depth tutorial on , see <http://perldoc.perl.org/perlretut.html> and for the reference page, see <http://perldoc.perl.org/perlre.html>

A difference with Perl is that matching a position but no character (for example, with `/^/` or `/(?=o)/`) is a successful match in Perl but not in Scilab.

Examples

```
regexp('xabyabbbz','/ab*/','o')  
regexp('a!','/((((((((a))))))))\041/')  
regexp('ABCC','/^abc$/i')  
regexp('ABC','/ab|cd/i')  
[a b c]=regexp('XABYABBBZ','/ab*/i')
```

See Also

`strindex`

Name

sci2exp — converts an expression to a string

```
t=sci2exp(a [,nam] [,lmax])
```

Parameters

a

a scilab expression, may be

- constant,
- polynomial
- string matrix
- list
- boolean matrix

nam

character string

t

vector of string, contains the expression or the affectation instruction

lmax

integer, contains the maximum line length. default value is 90, lmax=0 indicate no line length control a single string is returned

Description

sci2exp converts expression to an instruction string if nam is given or to an expression string.

Examples

```
a=[1 2;3 4]
sci2exp(a, 'aa')
sci2exp(a, 'aa', 0)
sci2exp(ssrand(2,2,2))
sci2exp(poly([1 0 3 4], 's'), 'fi')
```

Name

str2code — return scilab integer codes associated with a character string

```
c=str2code(str)
```

Parameters

str

A character string.

c

A vector of character integer codes

Description

Return c such that c(i) is the scilab integer code of part(str,i)

Examples

```
str2code('Scilab')'  
code2str([-28 12 18 21 10 11])
```

See Also

code2str, ascii

Name

strcat — concatenate character strings

```
txt=strcat(vector_of_strings [,string_added])
txt=strcat(vector_of_strings [,string_added],["flag"])
```

Parameters

vector_of_strings
vector of strings

string_added
string added, default value is the emptystr ""

txt
string

"flag"
string ("r" for return a column matrix, "c" for return a row matrix)

Description

txt=strcat(vector_of_strings) concatenates character strings :
txt=vector_of_strings(1)+...+vector_of_strings(n)

txt=strcat(vector_of_strings,string_added) returns
txt=vector_of_strings(1)+string_added+...+string_added
+vector_of_strings(n).

The plus symbol does the same: "a"+"b" is the same as strcat(["a","b"]).

If size of vector_of_strings is one, it returns
txt=vector_of_strings(1);

strcat('A','B') returns 'A' and not 'AB' as strcat(['A','B'])

Examples

```
strcat(string(1:10),',')
strcat(["a","b"])
strcat(["a","b"],'|')
strcat('A')
strcat('A','B')
strcat(['A','B'])
strcat(['A','B'],'')
```

See Also

string, strings

Name

strchr — Find the first occurrence of a character in a string

```
res = strchr(haystack, char)
```

Parameters

haystack

A character string or matrix of character strings

char

a character.

res

A character string or matrix of character strings

Description

`res = strchr(haystack, char)` Returns the first occurrence of character in the string `str`.

`num` must have same dimensions than `haystack` or only one char.

Examples

```
strchr('This is a sample string','s')  
strchr(['This is a sample string','in scilab'],'s')  
strchr(['This is a sample string','in scilab'],['s','a'])
```

See Also

strrchr , strstr

Name

strcmp — compare character strings

```
res = strcmp(string_one,string_two,[,'i'])
```

Parameters

string_one

A character string or matrix of character strings

string_two

A character string or matrix of character strings

'i'

parameter to do strcmp (case independent), default value is 's'

res

matrix.

Description

`res = strcmp(string_one,string_two)` returns an integral value indicating the relationship between the strings.

A value greater than zero indicates that the first character that does not match has a greater value in `string_one` than in `string_two`

And a value less than zero indicates the opposite.

Examples

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];  
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];  
strcmp(TXT1,TXT2)  
strcmp(TXT1,'scilab')  
strcmp(TXT1,'SciLab')  
strcmp(TXT1,TXT2,'i')
```

See Also

strcat, strcmpi

Name

strcmpi — compare character strings (case independent)

```
res = strcmpi(string_one,string_two)
```

Parameters

string_one

A character string or matrix of character strings

string_two

A character string or matrix of character strings

res

matrix.

Description

`res = strcmpi(string_one, string_two)` returns an integral value indicating the relationship between the strings.

A value greater than zero indicates that the first character that does not match has a greater value in `string_one` than in `string_two`

And a value less than zero indicates the opposite.

Examples

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];  
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];  
strcmpi(TXT1,TXT2)  
strcmpi(TXT1,'scilab')
```

See Also

strcat , strcmp

Name

strcspn — Get span until character in string

```
res = strcspn(string_one, string_two)
```

Parameters

string_one

A character string or matrix of character strings

string_two

A character string or matrix of character strings

res

matrix.

Description

`res = strcspn(string_one, string_two)` Scans `string_one` for the first occurrence of any of the characters that are part of `string_two`, returning the number of characters of `string_one` read before this first occurrence.

`string_one` must have same dimensions than `string_two` or `string_one` must be a string.

Examples

```
strcspn("fcba73", "1234567890")  
strcspn(["fcba73", "f7cba73"], "1234567890")  
strcspn(["fcba73", "f7cba73"], ["312", "34567890"])
```

See Also

strspn

Name

`strindex` — search position of a character string in an other string.

```
ind=strindex(haystack,needle,[flag])  
[ind,which]=strindex(haystack,needle,[flag])
```

Parameters

`haystack`

A character string. The string where to search occurrences of `needle`

`needle`

A character string or character string vector . The string(s) to search in `haystack`

`ind`

vector of indexes

`which`

vector of indexes

`flag`

string("r" for regular expression)

Description

`strindex` searches indexes where `needle (i)` is found in `haystack`

for each `k` it exist a `i` shuch that `part(haystack,ind(k)+(0:length(needle(i))-1))` is the same string than `needle(i)`. If `which` argument is required it contains these `i`. When using the third parameters "r", the needle should be a string of regular expression. And then `strindex` is going to match it with `haystack` according to the regular express rules.

`strindex` without regular expression argument is based on Knuth-Morris-Pratt algoritm.

This algorithm is more powerful than that used in `scilab 4.x`. In some special case, result can be different.

example:

```
// scilab 5.x
```

```
-->[k,w]=strindex('aab',['a','ab'])
```

```
w = 1. 1. 2. k = 1. 2. 2.
```

```
// scilab 4.x
```

```
-->[k,w]=strindex('aab',['a','ab'])
```

```
w = 1. 1. k = 1. 2.
```

The rules of regular expression are similar to perl language. For a quick start , see <http://perldoc.perl.org/perlrequick.html>. For a more in-depth tutorial on , see <http://perldoc.perl.org/perlretut.html> and for the reference page, see <http://perldoc.perl.org/perlre.html>

Examples

```
k=strindex('SCI/demos/scicos','/')
k=strindex('SCI/demos/scicos','SCI/')
k=strindex('SCI/demos/scicos','!')
k=strindex('aaaaa','aa')
k=strindex('SCI/demos/scicos',['SCI','sci'])
[k,w]=strindex('1+3*abc/2.33',['+','-','*','/'])
k=strindex('2','/2(]*?)$\\1/' , 'r')
```

See Also

string, strings, regexp, strsubst

Name

string — conversion to string

```
string(x)
[out,in,text]=string(x)
```

Parameters

x

Boolean, complex, real, integer, polynomial matrix, or function

Description

converts a matrix into a matrix of strings.

If *x* is a function `[out,in,text]=string(x)` returns three vectors strings : `out` is the vector of output variables, `in` is the vector of input variables, and `text` is the (column) vector of the source code of the function.

If *x* is a `lib` variable, `text` is a character string column vector. The first element contains the path of library file and the other the name of functions it defines.

Character strings are defined as `'string'` (between quotes) or `"string"` (between doublequotes); matrices of strings are defined as usual constant matrices.

Concatenation of strings is made by the `+` operation.

Examples

```
string(rand(2,2))
deff('y=mymacro(x)','y=x+1')
[out,in,text]=string(mymacro)
x=123.356;
disp('Result is '+string(x))

disp('/')+string(~%t)+'/'
disp('/')+string(%i+1)+'/'
disp('/')+string(int16(-123))+'/'
disp('/')+string(1+%s+%s^3)+'/'
```

See Also

`part` , `length` , `quote` , `evstr` , `execstr` , `strsubst` , `strcat` , `strindex` , `sci2exp`

Name

strings — Scilab Object, character strings

Description

Strings are defined as 'string' (between quotes) or "string" (between doublequotes); matrices of strings are defined as usual constant matrices.

Concatenation of two strings is made by a + : string1+string2.

Examples

```
['this','is'; 'a 2x2','matrix']  
"matrix"=="mat"+"rix"
```

See Also

part , length , strcat

Name

stripblanks — strips leading and trailing blanks (and tabs) of strings

```
txt=stripblanks(txt[, tabs])
```

Parameters

txt

A character string or matrix of character strings

tabs

if TRUE then tabs are also stripped (default value is FALSE)

Description

stripblanks strips leading and trailing blanks (and tabs) of strings

Examples

```
a=' 123  ';\n'!' +a+'!'\n'!' +stripblanks(a)+'!'\na=[' 123  ', ' xyz']\nstrcat(stripblanks(a))
```

Name

strncpy — Copy characters from strings

```
res = strncpy(str1,num)
```

Parameters

str1

A character string or matrix of character strings

num

matrix Maximum number of characters to be copied from source

res

A character string or matrix of character strings

Description

`res = strncpy(str1,num)` Copies the first num characters of source to destination.

num must have same dimensions than str1 or str2 must be a number.

Examples

```
strncpy('scilab',3)
strncpy(['scilab','SciLab';'strncpy','strstr'],3)
strncpy(['scilab','SciLab';'strncpy','strstr'],[1,2;3,4])
```

See Also

strcat , strcmp

Name

strrchr — Find the last occurrence of a character in a string

```
res = strrchr(str1,char)
```

Parameters

str1

A character string or matrix of character strings

char

a character.

res

A character string or matrix of character strings

Description

`res = strrchr(str1,char)` Returns the last occurrence of character in the string str.

num must have same dimensions than str1 or only one char.

Examples

```
strrchr('This is a sample string','s')  
strrchr(['This is a sample string','in scilab'],'s')  
strrchr(['This is a sample string','in scilab'],['s','a'])
```

See Also

strchr , strstr

Name

strrev — returns string reversed

```
res = strrev(str1)
```

Parameters

str1
A character string or matrix of character strings

res
A character string or matrix of character strings

Description

`res = strrev(str1)` Returns string reversed.

Examples

```
rev = strrev('This is a simple string')
strrev(rev)
strrev(['This is a simple string','scilab'])
```

Name

strsplit — split a string into a vector of strings

```
v = strsplit(str, ind)
[v, matched_separators] = strsplit(str)
[v, matched_separators] = strsplit(str, matrix_of_strings, limit)
[v, matched_separators] = strsplit(str, regexp_pattern, limit)
```

Parameters

str

A character string to split

ind

a vector of strictly increasing indices in the interval `[1 length(str)-1]` .

v

the resulting column vector of string (dimension > `size(ind, '*')+1`).

matched_separators

a column vector of the matched separators

matrix_of_strings

a matrix of strings to search in str

regexp_pattern

a regular expression pattern

limit

maximum of limit elements

Description

`v = strsplit(str, ind)` splits the string `str` into a vector of strings at the points given by the indices in `ind` (after each characters pointed to by the index in `ind`).

`strsplit(str)` returns same thing as `strsplit(str, 1:length(str)-1)`.

`strsplit(str, regexp_pattern, limit)` returns a column vector of strings, each of which is a substring of `str` formed by splitting it on boundaries formed by the case-sensitive regular expression pattern.

If there are `n` occurrences of pattern , the returned array will contain `n+1` items.

For example, if there is no occurrence of pattern , an array with only one element will be returned. Of course, this is also true if `str` is empty.

If `limit` is set, the returned array will contain a maximum of `limit` elements with the last element containing the whole rest of string.

`strsplit(str, matrix_of_strings, limit)` , `str` is splitted on any of elements. It allows to split on different separators for users without regex knowledge.

If `strsplit( )` is called with a second output argument, the column vector of the matched separators is returned.

Examples

```
S='strsplit splits a string into a vector of strings';
strsplit(S,[15 25 30])
ind=strindex(S, ' ')

[r_1, r_2] = strsplit("abcd")

[r_1, r_2] = strsplit("root:x:0:0:root:/root:/bin/bash",":",5)

[r_1, r_2] = strsplit("abc,def:ijk,:lmo","/:|,/")

[r_1, r_2] = strsplit("abc,def:ijk,:lmo",[":",";",","])

strsplit("abcdef2ghijkl3mnopqr6stuvw7xyz","\d+/")

[r_1, r_2] = strsplit("abcdef2ghijkl3mnopqr6stuvw7xyz","\d+/",2)
```

See Also

strcat, tokens, regexp

Authors

S. Steer, Allan CORNET
INRIA, DIGITEO

Name

strspn — Get span of character set in string

```
res = strspn(str1, str2)
```

Parameters

str1
A character string or matrix of character strings

str2
A character string or matrix of character strings

res
matrix.

Description

`res = strspn(str1, str2)` Returns the length of the initial portion of str1 which consists only of characters that are part of str2.

str1 must have same dimensions than str2 or str1 must be a string.

Examples

```
i = strspn("129th", "1234567890");  
printf ("The length of initial number is %d.\n", i);  
i = strspn(["129th", "130th"], ["1234567890", "130t"])
```

See Also

strcspn

Name

strstr — Locate substring

```
res = strstr(haystack,needle)
```

Parameters

haystack

A character string or matrix of character strings

needle

A character string or matrix of character strings

res

A character string or matrix of character strings

Description

`res = strstr(haystack,needle)` Returns a string matrix starting from where the first occurrence of needle in haystack to the end of haystack, or "" if there needle is not part of haystack.

Examples

```
strstr('This is a simple string','simple')
strstr('This is a simple string','sample')
strstr(['This is a simple string','in scilab'],'is')
strstr(['This is a sample string','in scilab'],['a','scilab'])
```

See Also

strchr, strchr

Name

`strsubst` — substitute a character string by another in a character string.

```
string_out=strsubst(string_in,searchStr,replaceStr)
string_out=strsubst(string_in,searchStr,replaceStr,[flag])
```

Parameters

`string_in`

a matrix of character string. The strings where to search occurrences of `searchStr`

`searchStr`

A character string. The string to search in `string`.

`replaceStr`

A character string. The replacement string.

`str_out`

A matrix of character strings. The result of the substitution on `searchStr` by `replaceStr` in `string`

`flag`

string("r" for regular expression)

Description

`strsubst` replaces all occurrences of `searchStr` in `string` by `replaceStr`.

When using the forth parameters "r", the `searchStr` should be a string of regular expression. And then `strsubst` is going to match it with `string` and replace according to the regular express rules.

Examples

```
strsubst('SCI/demos/scicos','SCI','.')
strsubst('SCI/demos/scicos','/',' ')
strsubst('2','/2(]*)?$\1/','0','r')
```

See Also

`string`, `strings`

Name

strtod — Convert string to double.

```
d = strtod(str)
[d,endstr] = strtod(str)
```

Parameters

str

A character string or matrix of character strings

d

A real or matrix of reals

endstr

A character string or matrix of character strings (next character in str after the numerical value).

Description

`[d,endstr] = strtod(str)` Parses strings str interpreting its content as a floating point number and returns its value as a real.

Examples

```
strtod('123.556This is a sample real')
[d,endstr] = strtod('123.556This is a sample real')
strtod(['123.556This is a sample real','888.666 here'])
[d,endstr] =strtod(['123.556This is a sample real','888.666 here'])
```

Name

strtok — Split string into tokens

```
res = strtok(str,delimiters)
```

Parameters

str
A character string

delimiters
A character string

res
A character string

Description

`res = strtok(str,delimiters)` sequence of calls to this function split `str` into tokens, which are sequences of contiguous characters speparated by any of the characters that are part of `delimiters`.

Examples

```
TOKENS = [];  
token = strtok("A string of ,,tokens and some more tokens"," ,");  
TOKENS = [TOKENS,token];  
while( token <> '' )  
    token = strtok(" ,");  
    TOKENS = [TOKENS,token];  
end  
disp(TOKENS);
```

See Also

strchr , strchr

Name

`tokenpos` — returns the tokens positions in a character string.

```
kdf=tokenpos(str [,delimiter])
```

Parameters

`str`

A character string. The string where to search the tokens.

`delimiter`

(optional) A character or a vector of characters. The tokens delimiters.

`kdf`

Two columns matrix, first column gives the index of the beginning of the tokens, the second gives the index of the last character of the tokens.

Description

`kdf=tokenpos(str [,delimiter])` searches the tokens included in the string `str`. The `delimiter` default value is `[" ",<Tab>]` where `<Tab>` stands for `ascii(9)`. It returns the indices of the first and last characters of each found tokens.

Examples

```
str='This is a character string';  
kdf=tokenpos(str)  
first=part(str,kdf(1,1):kdf(1,2))
```

See Also

`strindex` , `tokens`

Name

`tokens` — returns the tokens of a character string.

```
T=tokens(str [,delimiter])
```

Parameters

`str`

A character string. The string where to search the tokens.

`delimiter`

(optional) a character or a vector of characters. The tokens delimiters.

`T`

column vector of found tokens

Description

`T=tokens(str [,delimiter])` searches the tokens included in the string `str`. The `delimiter` default value is `[" ",<Tab>]` where `<Tab>` stands for `ascii(9)`.

Examples

```
tokens('This is a character string')  
  
tokens('SCI/demos/scicos','/')  
  
tokens('y=a+b*2',['=','+', '*'])
```

See Also

`strindex` , `tokenpos`

Name

tree2code — generates ascii definition of a Scilab function

```
txt=tree2code(tree,prettyprint)
```

Parameters

tree

a macro tree (coming from macr2tree)

prettyprint

optional boolean value

%T

generated code is indented and beautified

%F

generated code is not beautified (default)

txt

a column vector of strings, the text giving the Scilab instructions

Description

Given a loaded Scilab function "tree" (returned by macr2tree), tree2code allows to re-generate the code.

Examples

```
tree=macr2tree(cosh);
txt=tree2code(tree,%T);
write(%io(2),txt,'(a)');
```

See Also

macr2tree

Authors

V.C.

Part XXXII. Symbolic

Name

addf — symbolic addition

```
addf("a", "b")
```

Parameters

"a", "b"
character strings

Description

addf("a", "b") returns the character string "a+b". Trivial simplifications such as addf("0", "a") or addf("1", "2") are performed.

Examples

```
addf('0', '1')  
addf('1', 'a')  
addf('1', '2')  
'a'+ 'b'
```

See Also

mulf , subf , ldivf , rdivf , eval , evstr

Name

ldivf — left symbolic division

```
ldivf('d', 'c')
```

Description

returns the string 'c\d' Trivial simplifications such as '1\c' = 'c' are performed.

Examples

```
ldivf('1', '1')  
ldivf('a', '0')  
ldivf('a', 'x')  
ldivf('2', '4')
```

See Also

rdivf , addf , mulf , evstr

Name

mulf — symbolic multiplication

```
mulf('d', 'c')
```

Description

returns the string 'c*d' Trivial simplifications such as '1*c' = 'c' are performed.

Examples

```
mulf('1', 'a')  
mulf('0', 'a')  
'a'+ 'b' //Caution...
```

See Also

rdivf, addf, subf

Name

rdivf — right symbolic division

```
["r"]=ldivf("d","c")
```

Parameters

"d","c","r"
strings

Description

returns the string "c/d" Trivial simplifications such as "c/1" = "c" are performed.

Examples

```
ldivf('c','d')  
ldivf('1','2')  
ldivf('a','0')
```

See Also

ldivf

Name

subf — symbolic subtraction

```
["c"]=subf("a","b")
```

Parameters

"a","b","c"
strings

Description

returns the character string `c="a-b"` Trivial simplifications such as `subf("0","a")` or `subf("1","2")` are performed.

Examples

```
subf('0','a')  
subf('2','1')  
subf('a','0')
```

See Also

`mulf`, `ldivf`, `rdivf`, `eval`, `evstr`

Part XXXIII. Time and Date

Name

calendar — Calendar

```
c=calendar()  
c = calendar(y,m)
```

Description

`c = calendar` returns a list containing a calendar for the current month. The calendar runs Sunday to Saturday.

`c = calendar(y,m)`, where `y` and `m` are integers, returns a calendar for the specified month of the specified year.

Examples

```
calendar()  
calendar(1973,8)
```

See Also

`datevec` , `datenum`

Authors

Allan CORNET

Name

clock — Return current time as date vector

```
c = clock
```

Description

c = clock returns a 6-element date vector containing the current date and time in decimal form:

c = [year month day hour minute seconds]

the first five elements are integers. The seconds element is accurate to several digits beyond the decimal point.

Examples

```
clock
```

See Also

datenum , datevec , timer , etime , tic , toc

Authors

P.M

Name

date — Current date as date string

```
dt=date()
```

Parameters

dt
a string

Description

dt=date() returns a string containing the date in dd-mmm-yyyy format.

Examples

```
date()
```

See Also

getdate , toc , tic , timer , etime

Name

datenum — Convert to serial date number

```
N = datenum()  
N = datenum(DT)  
N = datenum(Y, M, D)  
N = datenum(Y, M, D, H, MI, S)
```

Description

The `datenum` function converts date vectors (defined by `datevec`) into serial date numbers. Date numbers are serial days elapsed from some reference date. By default, the serial day 1 corresponds to 1-Jan-0000.

`N = datenum()` returns the serial date numbers corresponding to current date.

`N = datenum(DT)` converts one or more date vectors to serial date number `N`. `DT` can be an `m`-by-6 or `m`-by-3 matrix containing `m` full or partial date vector respectively.

`N = datenum(Y, M, D)` returns the serial date numbers for corresponding elements of the `Y`, `M`, and `D` (year, month, day) arrays. `Y`, `M` and `D` must be arrays of the same size (or any can be a scalar).

`N = datenum(Y, M, D, H, MI, S)` returns the serial date numbers for corresponding elements of the `Y`, `M`, `D`, `H`, `MI`, and `S` (year, month, day, hour, minute, and second) array values. `Y`, `M`, `D`, `H`, `MI`, and `S` must be arrays of the same size (or any can be a scalar).

Examples

```
// N = datenum()  
datenum()  
  
// N = datenum(DT)  
A = [ 0 1 1 0 0 0 ; 2005 2 8 21 37 30 ]  
datenum(A)  
  
// N = datenum(Y, M, D)  
Years = [0; 1973; 2006]  
Months = [1; 8; 2]  
Days = [1; 4; 8]  
datenum(Years,Months,Days)  
  
Years = [0 0 0 ; 0 0 0]  
Months = [1 1 1 ; 1 1 1]  
Days = [1 2 3 ; 4 5 6]  
datenum(Years,Months,Days)  
  
// N = datenum(Y, M, D, H, MI, S)  
Years = grand(5,10,'uin',0,2006)  
Months = grand(5,10,'uin',1,12)  
Days = grand(5,10,'uin',1,28)  
Hours = grand(5,10,'uin',0,23)  
Minutes = grand(5,10,'uin',0,59)  
Seconds = grand(5,10,'uin',0,59)  
datenum(Years,Months,Days,Hours,Minutes,Seconds)
```

See Also

datevec , calendar

Authors

A.C

Name

datevec — Date components

```
V=datevec(DT)
[Y,M,D,H,MI,S]=datevec(DT)
```

Description

`V = datevec(DT)` converts a serial date number (defined by `datenum`) to a date vector `V` having elements [year, month, day, hour, minute, second]. The first five vector elements are integers. `DT` can be an array.

`[Y, M, D, H, MI, S] = datevec(DT)` returns the components of the date vector as individual variables. `DT` can be an array.

Examples

```
// First example
datevec(720840)

// Second example
datevec(datenum())

// Third example (With integers values)
A = grand(10,12,'uin',1,1000000)
datevec(A)

// Fourth example (With real values)
A = grand(10,12,'unf',1,1000000)
datevec(A)
```

See Also

`datenum` , `calendar`

Authors

A.C

Name

eomday — Return last day of month

```
E = eomday(Y, M)
```

Description

E = eomday(Y, M) returns the last day of the year and month given by corresponding elements of arrays Y and M.

Examples

```
eomday(2006,3);
```

See Also

datenum , datevec , weekday

Authors

P.M

Name

etime — Elapsed time

```
e = etime(t2,t1)
```

Parameters

t2

a vector with 6 or 10 values.

t1

a vector with 6 or 10 values.

e

number of seconds between t2 and t1.

Description

t1 and t2 with 10 values

t2 and t1 must have format returned by `getdate`. In this case, their third, fourth and fifth values are ignored.

t1 and t2 with 6 values

t2 and t1 must have format: T = [Year Month Day Hour Minute Second] with Second a number of seconds with milliseconds (e.g: 12.345).

t2 and t1 must have the same size.

t2 et t1 can be matrices with each line containing a format described above (all lines having same format).

Examples

```
t1=[2004 06 10 17 00 12.345]
t2=[2004 06 10 17 01 13.965]
E1=etime(t2,t1)
t1=[2004 06 24 162 5 10 17 00 12 345]
t2=[2004 06 24 162 5 10 17 01 13 965]
E2=etime(t2,t1)
```

See Also

tic , toc , getdate , datenum , datevec

Authors

V.C.

Name

getdate — get date and time information

```
dt=getdate()  
x=getdate("s")  
dt=getdate(x)
```

Parameters

- dt
an integer vector with 10 entries (see below)
- x
an integer containing a date coded in second from 1 Jan 1970

Description

- dt=getdate()
returns the current date in format given below:
- dt(1)
The year as a number (with the century) between 0000 and 9999.
- dt(2)
The month of the year as a number between 01 and 12.
- dt(3)
The ISO 8601 week number as a number between 01 and 53.
- dt(4)
The Julian day of the year as a number between 001 and 366.
- dt(5)
Specifies the weekday as a decimal number [1,7], with 1 representing Sunday.
- dt(6)
The day of the month as a number between 01 and 31.
- dt(7)
The hour of the day is output as a number between 00 and 23.
- dt(8)
The minute is output as a number between 00 and 59.
- dt(9)
The second is output as a number between 00 and 59.
- dt(10)
The millisecond is output as a number between 000 and 999.
- x=getdate("s")
returns a scalar with the number of seconds since Jan 1, 1970, 00:00 UTC (Unix Time Convention)
- dt=getdate(x)
formats the date given by x (number of seconds since Jan 1, 1970, 00:00 UTC) in format given above. In this case dt(10) is always equal to 0.

Examples

```
w=getdate()  
mprintf("Year:%d,Month:%d,Day:%d",w(1),w(2),w(6));  
  
x=getdate("s")  
getdate(x)
```

See Also

date , timer

Authors

V.C.

Name

now — Return current date and time

```
t = now()
```

Description

t = now() returns date and time as a serial date number. (See datenum)

Examples

```
realtimeinit(1);  
realtime(0);  
t1 = now()  
datevec(t1)  
realtime(10);  
t1 = now()  
datevec(t1)
```

See Also

clock , datenum , datevec

Authors

P.M

Name

`realtimeinit` — set time unit

`realtime` — set dates origin or waits until date

```
realtimeinit(time_unit)
realtime(t)
```

Parameters

`time_unit`

a real number. The number of seconds associated to the `realtime` argument

`t`

a real number. A date

Description

These two functions can be used to handle real time into Scilab.

`realtimeinit(time_unit)` defines the time unit associated to the `realtime` argument `t`

first call to `realtime(t0)` sets current date to (`t0`). subsequent calls to `realtime(t)` wait till date `t` is reached.

Examples

```
realtimeinit(1/2); //sets time unit to half a second
realtime(0); //sets current date to 0
for k=1:10, realtime(k); mprintf('current time is '+string(k/2)+'sec .\r\n'); end

//next instruction outputs a dot each 2 seconds
realtimeinit(2); realtime(0); for k=1:10, realtime(k); mprintf('.\r\n'); end

realtimeinit(1); realtime(0);
dt=getdate('s'); realtime(10); getdate('s')-dt
```

See Also

`getdate`

Name

sleep — suspend Scilab

```
sleep(milliseconds)
```

Description

`sleep` : Sleep process for specified number of miliseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

Examples

```
tic;sleep(6000);toc
```

See Also

`xpause` , `pause`

Authors

Allan CORNET

Name

tic — start a stopwatch timer

```
tic()
```

Description

The sequence of commands `tic(); operation; toc();` prints the number of seconds required for the operation.

Examples

```
tic();
realtimeinit(1);
realtime(0);
realtime(10);
toc();
```

See Also

`toc` , `timer` , `etime`

Authors

V.C.
A.C.

Name

timer — cpu time

```
timer()
```

Description

Returns the CPU time since the preceding call to `timer()`.

timer has a time precision of 100 nanoseconds.

NOTE: CPU time is the number of processor cycles used for a computation. This is not at all equivalent to real-world time.

CPU time can be used to compare CPU usage between different programs or functions , irrespective of background processes that might slow down the computer.

Examples

```
timer();A=rand(100,100);timer()
```

See Also

getdate, toc, tic, etime

Name

toc — Read the stopwatch timer

```
toc()  
t = toc()
```

Parameters

t
number of seconds since last call to `tic()` (Precision in order of millisecond).

Description

The sequence of commands `tic(); operation; toc();` prints the number of seconds required for the operation.

Examples

```
tic();  
realtimeinit(1);  
realtime(0);  
realtime(10);  
toc();
```

See Also

`tic` , `timer` , `etime`

Authors

V.C.
A.C.

Name

weekday — Return day of week

```
[N,S] = weekday(D)
[N,S] = weekday(D, form)
```

Description

`[N,S] = weekday(D)` returns the day of the week in numeric(N) and string(S) form for a given serial date number or date string D. Input argument D can represent more than one date in an array of serial date number.

`[N,S] = weekday(D, form)` returns the week in numeric(N) and string(S) form, where the content of S depends on the form argument. If form is 'long', then S contains the full name of the weekday (e.g., Tuesday). If form is 'short', then S contains an abbreviated name (e.g., Tue) from this table.

Examples

```
today = datenum();
[N,S] = weekday(today)
[N,S] = weekday(today, 'short')
[N,S] = weekday(today, 'long')
```

See Also

`datenum` , `datevec` , `weekday`

Authors

P.M

Part XXXIV. Statistics

Name

cdfbet — cumulative distribution function Beta distribution

```
[P,Q]=cdfbet("PQ",X,Y,A,B)
[X,Y]=cdfbet("XY",A,B,P,Q)
[A]=cdfbet("A",B,P,Q,X,Y)
[B]=cdfbet("B",P,Q,X,Y,A)
```

Parameters

P,Q,X,Y,A,B

five real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to X of the beta distribution (Input range: [0, 1].)

Q

1-P

X,Y (Y=1-X)

Upper limit of integration of beta density (Input range: [0,1], Search range: [0,1]) A,B : The two parameters of the beta density (input range: (0, +infinity), Search range: [1D-300,1D300])

Description

Calculates any one parameter of the beta distribution given values for the others (The beta density is proportional to $t^{A-1} * (1-t)^{B-1}$).

Cumulative distribution function (P) is calculated directly by code associated with the following reference.

DiDinato, A. R. and Morris, A. H. Algorithm 708: Significant Digit Computation of the Incomplete Beta Function Ratios. ACM Trans. Math. Softw. 18 (1993), 360-373.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfbin — cumulative distribution function Binomial distribution

```
[P,Q]=cdfbin("PQ",S,Xn,Pr,Ompr)
[S]=cdfbin("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfbin("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfbin("PrOmpr",P,Q,S,Xn)
```

Parameters

P,Q,S,Xn,Pr,Ompr

six real vectors of the same size.

P,Q (Q=1-P)

The cumulation from 0 to S of the binomial distribution. (Probability of S or fewer successes in XN trials each with probability of success PR.) Input range: [0,1].

S

The number of successes observed. Input range: [0, XN] Search range: [0, XN]

Xn

The number of binomial trials. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pr,Ompr (Ompr=1-Pr)

The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1]

Description

Calculates any one parameter of the binomial distribution given values for the others.

Formula 26.5.24 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the binomial distribution to the cumulative incomplete beta distribution.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfchi — cumulative distribution function chi-square distribution

```
[P,Q]=cdfchi("PQ",X,Df)
[X]=cdfchi("X",Df,P,Q);
[Df]=cdfchi("Df",P,Q,X)
```

Parameters

P,Q,Xn,Df

four real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to X of the chi-square distribution. Input range: [0, 1].

X

Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity).
Search range: [0,1E300]

Df

Degrees of freedom of the chi-square distribution. Input range: (0, +infinity). Search range:
[1E-300, 1E300]

Description

Calculates any one parameter of the chi-square distribution given values for the others.

Formula 26.4.19 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the chi-square distribution to the incomplete distribution.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfchn — cumulative distribution function non-central chi-square distribution

```
[P,Q]=cdfchn("PQ",X,Df,Pnonc)
[X]=cdfchn("X",Df,Pnonc,P,Q);
[Df]=cdfchn("Df",Pnonc,P,Q,X)
[Pnonc]=cdfchn("Pnonc",P,Q,X,Df)
```

Parameters

P,Q,X,Df,Pnonc

five real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to X of the non-central chi-square distribution. Input range: [0, 1-1E-16).

X

Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity).
Search range: [0,1E300]

Df

Degrees of freedom of the non-central chi-square distribution. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pnonc

Non-centrality parameter of the non-central chi-square distribution. Input range: [0, +infinity).
Search range: [0,1E4]

Description

Calculates any one parameter of the non-central chi-square distribution given values for the others.

Formula 26.4.25 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdff — cumulative distribution function F distribution

```
[P,Q]=cdff("PQ",F,Dfn,Dfd)
[F]=cdff("F",Dfn,Dfd,P,Q);
[Dfn]=cdff("Dfn",Dfd,P,Q,F);
[Dfd]=cdff("Dfd",P,Q,F,Dfn)
```

Parameters

P,Q,F,Dfn,Dfd

five real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to F of the f-density. Input range: [0,1].

F

Upper limit of integration of the f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn

Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Dfd

Degrees of freedom of the denominator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Description

Calculates any one parameter of the F distribution given values for the others.

Formula 26.6.2 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function for the F variate to that of an incomplete beta.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The value of the cumulative F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdffnc — cumulative distribution function non-central f-distribution

```
[P,Q]=cdffnc("PQ",F,Dfn,Dfd,Pnonc)
[F]=cdffnc("F",Dfn,Dfd,Pnonc,P,Q);
[Dfn]=cdffnc("Dfn",Dfd,Pnonc,P,Q,F);
[Dfd]=cdffnc("Dfd",Pnonc,P,Q,F,Dfn)
[Pnonc]=cdffnc("Pnonc",P,Q,F,Dfn,Dfd);
```

Parameters

P,Q,F,Dfn,Dfd,Pnonc

six real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to F of the non-central f-density. Input range: [0,1-1E-16).

F

Upper limit of integration of the non-central f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn

Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Dfd

Degrees of freedom of the denominator sum of squares. Must be in range: (0, +infinity). Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pnonc

The non-centrality parameter Input range: [0,infinity) Search range: [0,1E4]

Description

Calculates any one parameter of the Non-central F distribution given values for the others.

Formula 26.6.20 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

The value of the cumulative noncentral F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfgam — cumulative distribution function gamma distribution

```
[P,Q]=cdfgam("PQ",X,Shape,Scale)
[X]=cdfgam("X",Shape,Scale,P,Q)
[Shape]=cdfgam("Shape",Scale,P,Q,X)
[Scale]=cdfgam("Scale",P,Q,X,Shape)
```

Parameters

P,Q,X,Shape,Scale

five real vectors of the same size.

P,Q (Q=1-P)

The integral from 0 to X of the gamma density. Input range: [0,1].

X

The upper limit of integration of the gamma density. Input range: [0, +infinity). Search range: [0,1E300]

Shape

The shape parameter of the gamma density. Input range: (0, +infinity). Search range: [1E-300,1E300]

Scale

The scale parameter of the gamma density. Input range: (0, +infinity). Search range: (1E-300,1E300]

Description

Calculates any one parameter of the gamma distribution given values for the others.

Cumulative distribution function (P) is calculated directly by the code associated with:

DiDinato, A. R. and Morris, A. H. Computation of the incomplete gamma function ratios and their inverse. ACM Trans. Math. Softw. 12 (1986), 377-393.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The gamma density is proportional to $T^{(SHAPE - 1)} * \exp(-SCALE * T)$

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfnbn — cumulative distribution function negative binomial distribution

```
[P,Q]=cdfnbn("PQ",S,Xn,Pr,Ompr)
[S]=cdfnbn("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfnbn("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfnbn("PrOmpr",P,Q,S,Xn)
```

Parameters

P,Q,S,Xn,Pr,Ompr

six real vectors of the same size.

P,Q (Q=1-P)

The cumulation from 0 to S of the negative binomial distribution. Input range: [0,1].

S

The upper limit of cumulation of the binomial distribution. There are F or fewer failures before the XNth success. Input range: [0, +infinity). Search range: [0, 1E300]

Xn

The number of successes. Input range: [0, +infinity). Search range: [0, 1E300]

Pr

The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1].

Ompr

1-PR Input range: [0,1]. Search range: [0,1] PR + OMPR = 1.0

Description

Calculates any one parameter of the negative binomial distribution given values for the others.

The cumulative negative binomial distribution returns the probability that there will be F or fewer failures before the XNth success in binomial trials each of which has probability of success PR.

The individual term of the negative binomial is the probability of S failures before XN successes and is $\text{Choose}(S, XN+S-1) * PR^{(XN)} * (1-PR)^S$

Formula 26.5.26 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce calculation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfnor — cumulative distribution function normal distribution

```
[P,Q]=cdfnor("PQ",X,Mean,Std)
[X]=cdfnor("X",Mean,Std,P,Q)
[Mean]=cdfnor("Mean",Std,P,Q,X)
[Std]=cdfnor("Std",P,Q,X,Mean)
```

Parameters

P,Q,X,Mean,Std

six real vectors of the same size.

P,Q (Q=1-P)

The integral from -infinity to X of the normal density. Input range: (0,1].

X

Upper limit of integration of the normal-density. Input range: (-infinity, +infinity)

Mean

The mean of the normal density. Input range: (-infinity, +infinity)

Sd

Standard Deviation of the normal density. Input range: (0, +infinity).

Description

Calculates any one parameter of the normal distribution given values for the others.

A slightly modified version of ANORM from Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portabel FORTRAN Package of Special Function Routines and Test Drivers" acm Transactions on Mathematical Software. 19, 22-32. is used to calculate the cumulative standard normal distribution.

The rational functions from pages 90-95 of Kennedy and Gentle, Statistical Computing, Marcel Dekker, NY, 1980 are used as starting values to Newton's Iterations which compute the inverse standard normal. Therefore no searches are necessary for any parameter.

For $X < -15$, the asymptotic expansion for the normal is used as the starting value in finding the inverse standard normal. This is formula 26.2.12 of Abramowitz and Stegun.

The normal density is proportional to $\exp(-0.5 * ((X - MEAN)/SD)**2)$

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfpoi — cumulative distribution function poisson distribution

```
[P,Q]=cdfpoi("PQ",S,Xlam)
[S]=cdfpoi("S",Xlam,P,Q)
[Xlam]=cdfpoi("Xlam",P,Q,S);
```

Parameters

P,Q,S,Xlam

four real vectors of the same size.

P,Q (Q=1-P)

The cumulation from 0 to S of the poisson density. Input range: [0,1].

S

Upper limit of cumulation of the Poisson. Input range: [0, +infinity). Search range: [0,1E300]

Xlam

Mean of the Poisson distribution. Input range: [0, +infinity). Search range: [0,1E300]

Description

Calculates any one parameter of the Poisson distribution given values for the others.

Formula 26.4.21 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of computing a chi-square, hence an incomplete gamma function.

Cumulative distribution function (P) is calculated directly. Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdft — cumulative distribution function Student's T distribution

```
[P,Q]=cdft("PQ",T,Df)
[T]=cdft("T",Df,P,Q)
[Df]=cdft("Df",P,Q,T)
```

Parameters

P,Q,T,Df

six real vectors of the same size.

P,Q (Q=1-P)

The integral from -infinity to t of the t-density. Input range: (0,1].

T

Upper limit of integration of the t-density. Input range: (-infinity, +infinity). Search range: [-1E150, 1E150]

DF:

Degrees of freedom of the t-distribution. Input range: (0 , +infinity). Search range: [1e-300, 1E10]

Description

Calculates any one parameter of the T distribution given values for the others.

Formula 26.5.27 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

Name

center — center

```
s=center(x)
s=center(x, 'r') or s=center(x,1)
s=center(x, 'c') or s=center(x,2)
```

Parameters

x: real or complex vector or matrix

Description

This function computes S , the centred version of the numerical matrix x . For a vector or a matrix x , $s=center(x)$ returns in the (i, j) coefficient of the matrix S the value $(x(i, j) - \bar{x})$, where $\bar{x}$ is the mean of the values of the coefficients of x . $s=center(x, 'r')$ (or, equivalently, $s=center(x, 1)$) is the rowwise center reduction of the values of x . It returns in the entry $s(i, j)$ the value $(x(i, j) - \bar{x}_v(j))(j)$ with $\bar{x}_v(j)$ the mean of the values of the j column. $s=center(x, 'c')$ (or, equivalently, $s=center(x, 2)$) is the columnwise center reduction of the values of x . It returns in the entry $s(i, j)$ the value $(x(i, j) - \bar{x}_h(i))$ with $\bar{x}_h(i)$ the mean of the values of the i row.

Examples

```
x=[0.2113249  0.0002211  0.6653811;
   0.7560439  0.3303271  0.6283918]
s=center(x)
s=center(x, 'r')
s=center(x, 'c')
```

See Also

wcenter

Authors

Carlos Klimann

Name

wcenter — center and weight

```
s=wcenter(x)
s=wcenter(x, 'r') or s=wcenter(x,1)
s=wcenter(x, 'c') or s=wcenter(x,2)
```

Parameters

x: real or complex vector or matrix

Description

This function computes S , the weighed and centred version of the numerical matrix x .

For a vector or a matrix x , $s=wcenter(x)$ returns in the (i, j) coefficient of the matrix S the value $(x(i, j) - \bar{x})/\sigma$, where $\bar{x}$ is the mean of the values of the coefficients of x and σ his standard deviation.

$s=wcenter(x, 'r')$ (or, equivalently, $s=wcenter(x, 1)$) is the rowwise centre reduction of the values of x . It returns in the entry $s(i, j)$ the value $(x(i, j) - \bar{x}_v(j))/\sigma_v(j)$ with $\bar{x}_v(j)$ the mean of the values of the j column and $\sigma_v(j)$ the standard deviation of the j column of x .

$s=wcenter(x, 'c')$ (or, equivalently, $s=wcenter(x, 2)$) is the columnwise centre reduction of the values of x . It returns in the entry $s(i, j)$ the value $(x(i, j) - \bar{x}_h(i))/\sigma_h(i)$ with $\bar{x}_h(i)$ the mean of the values of the i row and $\sigma_h(i)$ the standard deviation of the i row of x .

Examples

```
x=[0.2113249 0.0002211 0.6653811;
    0.7560439 0.3303271 0.6283918]
s=wcenter(x)
s=wcenter(x, 'r')
s=wcenter(x, 'c')
```

See Also

center

Authors

Carlos Klimann

Name

cmoment — central moments of all orders

```
mom=cmoment(x,ord)
mom=cmoment(x,ord,'r') or mom=cmoment(x,ord,1)
mom=cmoment(x,ord,'c') or mom=cmoment(x,ord,2)
```

Parameters

x
real or complex vector or matrix

ord
positive integer

Description

cmoment(x,ord) is the central moment of order ord of the elements of x. If a third argument of type string 'r' (or 1) or 'c' (or 2) is used, we get in the first case, a row vector mom such that mom(j) contains the central moment of order ord of the j column of x. cmoment(x,ord,'c') is used in the same way for the central moments in the rows.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 0.3303271 0.6283918]
mom=cmoment(x,3)
mom=cmoment(x,2,'r')
mom=cmoment(x,3,'c')
```

See Also

sum , median , st_deviation , mean , meanf , moment , nanmean , nanmeanf , stdev , stdevf , variance , variancef , nanstdev

Authors

Carlos Klimann

Name

correl — correlation of two variables

```
rho=correl(x,y,fre)
```

Parameters

x
real or complex vector

y
real or complex vector

fre
matrix of type length(x) x length(y)

Description

`correl(x,y,fre)` computes the correlation of two variables `x` and `y`. `fre` is a matrix of dimensions `length(x) x length(y)`. In `fre` the element of indices `(i,j)` corresponds to the value or number or frequencies of `x_i&y_j`.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[2.5 7.5 12.5 17.5]
h=[0 1 2]
fre=[.03 .12 .07;.02 .13 .11;.01 .13 .14;.01 .09 .14]
rho=correl(x,h,fre)
```

See Also

covar

Authors

Carlos Klimann

Name

covar — covariance of two variables

```
s=covar(x,y, fre)
```

Parameters

x
real or complex vector

y
real or complex vector

fre
matrix of type length(x) x length(y)

Description

`covar(x,y, fre)` computes the covariance of two variables `x` and `y`. `fre` is a matrix of dimensions `length(x) x length(y)`. In `fre` the element of indices `(i,j)` corresponds to the value or number or frequencies of `x_i&y_j`.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[10 20 30 40]
y=[10 20 30 40]
fre=[.20 .04 .01 0;
      .10 .36 .09 0;
      0 .05 .10 0;
      0 0 0 .05]
s=covar(x,y, fre)
```

Authors

Carlos Klimann

Name

`ftest` — Fischer ratio

```
f=ftest(samples)
[f,p]=ftest(samples)
```

Parameters

`samples`
real or complex matrix of type `nr X nc`

Description

`f=ftest(samples)` computes the Fischer ratio of the `nc` samples whose values are in the columns of the matrix `samples`. Each one of these samples is composed of `nr` values. (The Fischer ratio is the ratio between `nr` times the variance of the means of samples and the mean of variances of each sample)

`[f,p]=ftest(samples)` gives in `p` the p-value of the computed Fischer ratio `f`.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
samples=[46 55 54;
          53 54 50;
          49 58 51;
          50 61 51;
          46 52 49]
[f,p]=ftest(samples)
```

See Also

`ftuneq`

Authors

Carlos Klimann

Name

ftuneq — Fischer ratio for samples of unequal size.

```
f=ftuneq(sample1[, sample2[, sample3]...])  
[f,p]=ftuneq(sample1[, sample2[, sample3]...])
```

Parameters

sample1, sample2, sample3,...
real or complex matrix of any type

Description

This function computes the F ratio for samples of unequal size.

"The most efficient design is to make all samples the same size n. However when this is not feasible, it still is possible to modify the ANOVA calculations." Note that the definition of xbarbar is no longer mean(xbar), but rather a weighted average with weights ni. Additionally it gives (in p) the p-value of the computed Fischer ratio.

Given a number a of samples each of them composed of n_i (i from 1 to a) observations this function computes in f the Fischer ratio (it is the ratio between nr times the variance of the means of samples and the mean of the variances of each sample).

f=ftest(samples) computes the Fischer ratio of the nc samples whose values are in the columns of the matrix samples. Each one of these samples is composed of nr values. (The Fischer ratio is the ratio between nr times the variance of the means of samples and the mean of variances of each sample)

[f,p]=ftest(samples) gives in p the p-value of the computed Fischer ratio f.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
samples=[46 55 54;53 54 50; 49 58 51;50 61 51;46 52 49]  
[f,p]=ftest(samples)
```

See Also

ftuneq

Authors

Carlos Klimann

Name

geomean — geometric mean

```
gm=geomean(x)
gm=geomean(x, 'r')(or, equivalently, gm=geomean(x,1))
gm=geomean(x, 'c')(or, equivalently, gm=geomean(x,2))
```

Parameters

x
real or complex vector or matrix

Description

This function computes the geometric mean of a vector or matrix x . For a vector or matrix x , `gm=geomean(x)` returns in scalar `gm` the geometric mean of all the entries of x .

`gm=geomean(x, 'r')` (or, equivalently, `gm=gmean(x,1)`) returns in each entry of the row vector `gm` the geometric mean of each column of x .

`gm=geomean(x, 'c')` (or, equivalently, `gm=gmean(x,2)`) returns in each entry of the column vector `gm` the geometric mean of each row of x .

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

harmean — harmonic mean

```
hm=harmean(x)
hm=harmean(x,'r')(or, equivalently, hm=harmean(x,1))
hm=harmean(x,'c')(or, equivalently, hm=harmean(x,2))
```

Parameters

x
real or complex vector or matrix

Description

This function computes the harmonic mean of a vector or matrix x . For a vector or matrix x , `hm=harmean(x)` returns in scalar `hm` the harmonic mean of all the entries of x .

`hm=harmean(x,'r')` (or, equivalently, `hm=harmean(x,1)`) returns in each entry of the row vector `hm` the harmonic mean of each column of x .

`hm=harmean(x,'c')` (or, equivalently, `hm=harmean(x,2)`) returns in each entry of the column vector `hm` the harmonic mean of each row of x .

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

iqr — interquartile range

```
q=iqr(x)
q=iqr(x,'r') (or, equivalently, q=iqr(x,1))
q=iqr(x,'c') (or, equivalently, q=iqr(x,2))
```

Parameters

x
real or complex vector or matrix

Description

This function computes the interquartile range $IQR = \text{upper quartile} - \text{lower quartile}$ of a vector or a matrix x .

For a vector or a matrix x , $q=iqr(x)$ returns in the scalar q the interquartile range of all the entries of x .

$q=iqr(x, 'r')$ (or, equivalently, $q=iqr(x, 1)$) is the rowwise interquartile range. It returns in each entry of the row vector q the interquartile range of each column of x .

$q=iqr(x, 'c')$ (or, equivalently, $q=iqr(x, 2)$) is the columnwise interquartile range. It returns in each entry of the column vector q the interquartile range of each row of x .

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. Wonacott, R.J.; Introductory Statistics, J.Wiley-Sons, 1990.

Name

labostat — Statistical toolbox for Scilab

Contents

```
centre: centering variables
centrered: centering and reducing variables
cmoment: central moments of all orders
correl: correlation
covar: covariance
ftest: fischer test and his p-value
geomean: geometric mean
harmean: harmonic mean
iqr: interquartile range
mad: mean absolute deviation
meanf: arithmetic mean of a vector or matrix with a table of frequencies
median: 50th percentile of a sample
mn: arithmetic mean of a vector or matrix
moment: moments of all orders
msd: mean squared deviation
mvvacov : multivariable matrix of variance-covariance
nand2mean: estimate of the difference of means of two independent samples
nanmax: maximum ignoring NaNs
nanmean: mean ignoring NaNs
nanmeanf: mean with frequency table ignoring NaNs
nanmedian: 50th percentile of a sample ignoring NaNs
nanmin: minimum ignoring NaNs
nanstdev: standard deviation ignoring NaNs
nanstdevf: standard deviation with frequency table ignoring NaNs
nansum: sum ignoring NaNs
nfreq: frequency of the values of a sample
pca: principal component analysis
pctl: vector of percentiles of a sample in decreasing order
perctl: vector of percentiles of a sample in decreasing order
quart: quartils
stdev: standard deviation
stdevf: standard deviation with frequencies
strange: distance between largest and smallest value
tabul: frequencies of values
var: variance
varf: variance with frequency table
```

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

mad — mean absolute deviation

```
s2=mad(x)
s2=mad(x, 'r') or s2=mad(x,1)
s2=mad(x, 'c') or s2=mad(x,2)
```

Parameters

x
real or complex vector or matrix

Description

This function computes the mean absolute deviation of a real or complex vector or matrix x .

For a vector or matrix x , $s2=mad(x)$ returns in scalar $s2$ the mean absolute deviation of all the entries of x .

$s2=mad(x, 'r')$ (or, equivalently, $s2=mad(x,1)$) returns in each entry of the column vector $s2$ the mean absolute deviation of each column of x .

$s2=mad(x, 'c')$ (or, equivalently, $s2=mad(x,2)$) returns in each entry of the column vector $s2$ the mean absolute deviation of each row of x .

Bibliography

Reference: Wonacott T.H.& Wonacott R.J. - Introductory Statistics, 5th edition, John Wiley, 1990.

Name

mean — mean (row mean, column mean) of vector/matrix entries

```
y=mean(x)
y=mean(x, 'r')
y=mean(x, 'c')
y=mean(x, 'm')
```

Parameters

x
real vector or matrix

y
scalar or vector

Description

For a vector or a matrix **x**, **y=mean(x)** returns in the scalar **y** the mean of all the entries of **x**.

y=mean(x, 'r') (or, equivalently, **y=mean(x, 1)**) is the rowwise mean. It returns a row vector:
y(j)= mean(x(:, j))

y=mean(x, 'c') (or, equivalently, **y=mean(x, 2)**) is the columnwise mean. It returns a column vector: **y(i)= mean(x(i, :))**

y=mean(x, 'm') is the mean along the first non singleton dimension of **x** (for compatibility with Matlab).

Examples

```
A=[1,2,10;7,7.1,7.01];
mean(A)
mean(A, 'r')
mean(A, 'c')
A=matrix(1:12, [1,1,2,3,2]);
// in this case mean(A, 'm') is equivalent to mean(A,3), the first non singleton
y=mean(A, 'm')
```

See Also

[sum](#) , [median](#) , [st_deviation](#)

Name

meanf — weighted mean of a vector or a matrix

```
m=meanf(val,fre)
m=meanf(val,fre,'r') or m=meanf(val,fre,1)
m=meanf(val,fre,'c') or m=meanf(val,fre,2)
```

Parameters

?

Description

This function computes the mean of a vector or matrix x . For a vector or matrix x , $m=mn(x)$ returns in scalar m the mean of all the entries of x . $m=mn(x, 'r')$ (or, equivalently, $m=mn(x, 1)$) returns in each entry of the row vector m the mean of each column of x . $m=mn(x, 'c')$ (or, equivalently, $m=mn(x, 2)$) returns in each entry of the column vector m the mean of each row of x .

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=meanf(x,rand(x))
m=meanf(x,[10 10 10;1 1 1],'r')
m=meanf(x,[10 10 10;1 1 1],'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

median — median (row median, column median,...) of vector/matrix/array entries

```
y=median(x)
y=median(x, 'r')
y=median(x, 'c')
y=median(x, 'm')
y=median(x, dim)
```

Parameters

x
real vector, matrix or an array

y
scalar,vector, matrix or an array

dim
positive integer

Description

For a vector or a matrix **x**, **y=median(x)** returns in the scalar **y** the median of all the entries of **x**.

y=median(x, 'r') (or, equivalently, **y=median(x, 1)**) is the median along the row index. It returns in each entry of the column vector **y** the median of each column of **x**.

y=median(x, 'c') (or, equivalently, **y=median(x, 2)**) is the median along the column index. It returns in each entry of the row vector **y** the median of each row of **x**.

y=median(x, 'm') is the median along the first non singleton dimension of **x** (for compatibility with matlab).

y=median(x, dim) is the median along the dimension **dim** of **x** (for compatibility with matlab).

Examples

```
A=[1,2,10;7,7.1,7.01];
median(A)
median(A, 'r')
median(A, 'c')
A=matrix([-9 3 -8 6 74 39 12 -6 -89 23 65 34],[2,3,2]);
median(A,3)
median(A, 'm')
```

See Also

sum , mean

Name

moment — non central moments of all orders

```
mom=moment(x,ord)
mom=moment(x,ord,'r') or mom=moment(x,ord,1)
mom=moment(x,ord,'c') or mom=moment(x,ord,2)
```

Parameters

x
real or complex vector or matrix

ord
positive integer

Description

`moment(x,ord)` is the non central moment of order `ord` of the elements of `x`.

If a third argument of type string `'r'` (or `1`) or `'c'` (or `2`) is used, we get in the first case, a row vector `mom` such that `mom(j)` contains the non central moment of order `ord` of the `j` column of `x`. `moment(x,ord,'c')` is used in the same way for the non central moments in the rows.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
mom=moment(x,3)
mom=moment(x,2,'r')
mom=moment(x,3,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

msd — mean squared deviation

```
y=msd(x)
y=msd(x, 'r') or m=msd(x,1)
y=msd(x, 'c') or m=msd(x,2)
```

Parameters

x
real or complex vector or matrix

Description

This function computes the mean squared deviation of the values of a vector or matrix x .

For a vector or a matrix x , $y=\text{msd}(x)$ returns in the scalar y the mean squared deviation of all the entries of x .

$y=\text{msd}(x, 'r')$ (or, equivalently, $y=\text{msd}(x, 1)$) is the rowwise mean squared deviation. It returns in each entry of the row vector y the mean squared deviation of each column of x .

$y=\text{msd}(x, 'c')$ (or, equivalently, $m=\text{msd}(x, 2)$) is the columnwise mean squared deviation. It returns in each entry of the column vector y the mean squared deviation of each row of x .

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=msd(x)
m=msd(x, 'r')
m=msd(x, 'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

mvvacov — computes variance-covariance matrix

```
v=mvvacov(x)
```

Parameters

x
real or complex vector or matrix

Description

This function computes v , the matrix of variance-covariance of the "tableau" x (x is a numerical matrix $n \times p$) who gives the values of p variables for n individuals: the (i,j) coefficient of v is $v(i,j)=E(x_i-x_{i\bar{}})(x_j-x_{j\bar{}})$, where E is the first moment of a variable, x_i is the i -th variable and $x_{i\bar{}}$ the mean of the x_i variable.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]
v=mvvacov(x)
```

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990. Mardia, K.V., Kent, J.T. & Bibby, J.M., Multivariate Analysis, Academic Press, 1979.

Name

nancumsum — Thos function returns the cumulative sum of the values of a matrix

```
s = nancumsum(x,orient)
```

Parameters

x

x is a numerical vector or matrix.

orient

is an optional parameter. The possible values are '*', 1, 2, 'r' or 'c'.

s

numerical scalar or vector. It contains the cumulative sum of the values of x, ignoring the NAN's.

Description

This function returns in scalar or vector s the cumulative sum of the values (ignoring the NANs) of a vector or matrix (real or complex) x.

This function for a vector or a matrix x, s=nancumsum(x) (or, equivalently s=nancumsum(x, '*') returns in scalar s the cumulative sum (ignoring the NANs) of all the entries of x taken columnwise.

s=nancumsum(x, 'r') (or, equivalently, s=nancumsum(x, 1)) returns in the cols(x) sized vector s the cumulative sum (ignoring the NANs) of the rows of x: s(:,i)=nancumsum(x(:,i))

s=nancumsum(x, 'c') (or, equivalently, s=nancumsum(x, 2)) returns in the rows(x) sized vector s the cumulative sum (ignoring NANs) of the columns of x: s(i,:)=nancumsum(x(i,:))

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

Examples

```
a=[1 2 3;4 5 6]
s=nancumsum(a)
s=nancumsum(a, 'r')
s=nancumsum(a, 'c')
```

See Also

nansum , cumsum

Authors

Carlos Klimann

Name

nand2mean — difference of the means of two independent samples

```
[dif]=nand2mean(sample1, sample2)
[dif]=nand2mean(sample1, sample2, conf)
```

Parameters

sample1
real or complex vector or matrix

sample2
real or complex vector or matrix

conf
real scalar between 0 and 1

Description

This function computes an estimate (dif(1)) for the difference of the means of two independent samples (arrays sample1 and sample2) and gives the half amplitude of the range of variability of dif with an indicated confidence level (dif(2)). The choice of the normal or t functions as the probability function depends on the sizes of sample1 and sample2. We suppose that the underlying variances of both populations are equal. NAN values are not counted.

In Labostat, NAN values stand for missing values in tables.

In absence of the confidence parameter a confidence level of 95% is assumed.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

Name

nanmax — max (ignoring Nan's)

```
[m,index]=nanmax(x)
[m,index]=nanmax(x,'r')
[m,index]=nanmax(x,'c')
```

Parameters

x
real or complex vector or matrix

Description

This function gives for a real or a numerical matrix **x** his largest element **m** (but ignoring the NaNs).

For **x**, a numerical vector or matrix, **m=nanmax(x)** returns in scalar **m** the largest element of **x** (ignoring the NaNs). The form **[m,index]=nanmax(x,orient)** gives in addition of the value of the largest element of **x** (ignoring the NaNs) in scalar **m**, the index of this element in **x**, as a 2-vector.

m=nanmax(x,'r') gives in the $1 \times \text{size}(x,2)$ matrix **m** the largest elements (ignoring the NaNs) of each column of **x**. If the form **[m,index]=nanmax(x,'r')** is used, the elements of the $1 \times \text{size}(x,2)$ matrix **index** are the indexes of the largest elements (ignoring the NaNs) of each column of **x** in the corresponding column.

m=nanmax(x,'c') gives in the $\text{size}(x,2) \times 1$ matrix **m** the largest elements (ignoring the NaNs) of each row of **x**. If the form **[m,index]=nanmax(x,'c')** is used, the elements of the $\text{size}(x,2) \times 1$ matrix **index** are the indexes of the largest elements (ignoring the NaNs) of each row of **x** in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmax(x)
m=nanmax(x,'r')
m=nanmax(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmean — mean (ignoring Nan's)

```
m=nanmean(val)
m=nanmean(val,'r') (or m=nanmean(val,1))
m=nanmean(val,'c') (or m=nanmean(val,2))
```

Parameters

val
real or complex vector or matrix

Description

This function returns in scalar `m` the mean of the values (ignoring the NaNs) of a vector or matrix `val`.

For a vector or matrix `val`, `m=nanmean(val)` or `m=nanmean(val, '*')` returns in scalar `m` the mean of all the entries (ignoring the NaNs) of `val`.

`m=nanmean(val, 'r')` (or, equivalently, `m=nanmean(val, 1)`) returns in each entry of the row vector `m` of type `1xsize(val,'c')` the mean of each column of `val` (ignoring the NaNs).

`m=nanmeanf(val, 'c')` (or, equivalently, `m=nanmean(val, 2)`) returns in each entry of the column vector `m` of type `size(val,'c')x1` the mean of each row of `val` (ignoring the NaNs).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmean(x)
m=nanmean(x,1)
m=nanmean(x,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmeanf — mean (ignoring Nan's) with a given frequency.

```
m=nanmean(val,fre)
m=nanmean(val,fre,'r') (or m=nanmean(val,fre,1))
m=nanmean(val,fre,'c') (or m=nanmean(val,fre,2))
```

Parameters

val

real or complex vector or matrix

fre

integer vector or matrix with same dimensions than val

Description

This function returns in scalar `m` the mean of the values (ignoring the NaNs) of a vector or matrix `val`, each counted with a frequency signaled by the corresponding values of the integer vector or matrix `fre` with the same type of `val`.

For a vector or matrix `val`, `m=nanmeanf(val,fre)` or `m=nanmeanf(val,fre,'*')` returns in scalar `m` the mean of all the entries (ignoring the NaNs) of `val`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'r')` (or, equivalently, `m=nanmeanf(val,fre,1)`) returns in each entry of the row vector `m` of type `1xsize(val,'c')` the mean of each column of `val` (ignoring the NaNs), each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'c')` (or, equivalently, `m=nanmeanf(val,fre,2)`) returns in each entry of the column vector `m` of type `size(val,'c')x1` the mean of each row of `val` (ignoring the NaNs), each value counted with the multiplicity indicated by the corresponding value of `fre`.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
fre=[34 12 25;12 23 5]
m=nanmeanf(x,fre)
m=nanmeanf(x,fre,1)
m=nanmeanf(x,fre,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmedian — median of the values of a numerical vector or matrix

```
m=nanmedian(x)
m=nanmedian(x, 'r') (or m=nanmedian(x,1))
m=nanmedian(x, 'c') (or m=nanmedian(x,2))
```

Parameters

x
real or complex vector or matrix

Description

For a vector or a matrix x , `[m]=nanmedian(x)` returns in the vector m the median of the values (ignoring the NaNs) of vector x .

`[m]=nanmedian(x, 'r')` (or, equivalently, `[m]=nanmedian(x,1)`) are the rowwise medians. It returns in each position of the row vector m the medians of data (ignoring the NaNs) in the corresponding column of x .

`[m]=nanmedian(x, 'c')` (or, equivalently, `[m]=nanmedian(x,2)`) are the columnwise medians. It returns in each position of the column vector m the medians of data (ignoring the NaNs) in the corresponding row of x .

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmedian(x)
m=nanmedian(x,1)
m=nanmedian(x,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmin — min (ignoring Nan's)

```
[m,index]=nanmin(x)
[m,index]=nanmin(x,'r')
[m,index]=nanmin(x,'c')
```

Parameters

x
real or complex vector or matrix

Description

This function gives for a real or a numerical matrix **x** his largest element **m** (but ignoring the NaNs).

For **x**, a numerical vector or matrix, **m=nanmin(x)** returns in scalar **m** the largest element of **x** (ignoring the NaNs). The form **[m,index]=nanmin(x,orient)** gives in addition of the value of the largest element of **x** (ignoring the NaNs) in scalar **m**, the index of this element in **x**, as a 2-vector.

m=nanmin(x,'r') gives in the $1 \times \text{size}(x,2)$ matrix **m** the largest elements (ignoring the NaNs) of each column of **x**. If the form **[m,index]=nanmin(x,'r')** is used, the elements of the $1 \times \text{size}(x,2)$ matrix **index** are the indexes of the largest elements (ignoring the NaNs) of each column of **x** in the corresponding column.

m=nanmin(x,'c') gives in the $\text{size}(x,2) \times 1$ matrix **m** the largest elements (ignoring the NaNs) of each row of **x**. If the form **[m,index]=nanmin(x,'c')** is used, the elements of the $\text{size}(x,2) \times 1$ matrix **index** are the indexes of the largest elements (ignoring the NaNs) of each row of **x** in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmin(x)
m=nanmin(x,'r')
m=nanmin(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanstdev — standard deviation (ignoring the NaNs).

```
s=nanstdev(x)
s=nanstdev(x, 'r') or m=nanstdev(x,1)
s=nanstdev(x, 'c') or m=nanstdev(x,2)
```

Parameters

x
real or complex vector or matrix

Description

This function computes the standard deviation of the values of a vector or matrix **x** (ignoring the NaNs).

For a vector or a matrix **x**, **s=nanstdev(x)** returns in the scalar **s** the standard deviation of all the entries of **x** (ignoring the NaNs).

s=nanstdev(x, 'r') (or, equivalently, **s=nanstdev(x,1)**) is the rowwise standard deviation. It returns in each entry of the row vector **s** the standard deviation of each column of **x** (ignoring the NaNs).

s=nanstdev(x, 'c') (or, equivalently, **s=nanstdev(x,2)**) is the columnwise standard deviation. It returns in each entry of the column vector **s** the standard deviation of each row of **x** (ignoring the NaNs).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 %nan    0.6283918;
   0.3        0.2      0.5        ];
s=nanstdev(x)
s=nanstdev(x, 'r')
s=nanstdev(x, 'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nansum — Sum of values ignoring NAN's

```
s = nansum(x,orient)
```

Parameters

x

numerical vector or matrix.

orient

nothing or '*' . 'r' or 1. 'c' or 2.

s

Numerical scalar or vector containig the value of the adding operation.

Description

This function returns in s the sum of the values (ignoring the NAN's) of a numerical vector or matrix x.

For a vector or matrix x, s=nansum(x) (or s=nansum(x,'*')) returns in scalar s the sum of values of all entries (ignoring the NAN's) of a vector or matrix x.

s=nansum(x,'r')(or, equivalently, s=nansum(x,1)) returns in each entry of the row vector s of type 1xsize(x,'c') the sum of each column of x (ignoring the NANs).

s=nansum(x,'c')(or, equivalently, s=nansum(x,2)) returns in each entry of the column vector s of type size(x,'c')x1 the sum of each row of x (ignoring the NANs).

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nansum(x)
m=nansum(x,1)
m=nansum(x,2)
```

See Also

nancumsum , sum

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. and Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley and Sons, 1990.

Name

`nfreq` — frequency of the values in a vector or matrix

```
m=nfreq(x)
```

Parameters

`x`
real or complex vector or matrix

Description

Frequency of the values in a real or complex vector or a real or complex matrix `x`.

For a real or complex vector or a real or complex matrix `x`, `m=nfreq(x)` returns in the first column of the `size(x, 's')×2` matrix `m` the values of `x` and in the second column of this matrix the frequencies of the corresponding values.

Note that the `tabul` function is more efficient, applies also to vector of strings and returns a sorted `m`.

Examples

```
x=[2 8 0 3 7 6 8 7 9 1]
m=nfreq(x)
```

See Also

`tabul` , `dsearch` , `histplot`

Authors

Carlos Klimann

Name

pca — Computes principal components analysis with standardized variables

```
[lambda, facpr, comprinc] = pca(x)
```

Parameters

x

is a $n \times p$ (n individuals, p variables) real matrix. Note that **pca** center and normalize the columns of **x** to produce principal components analysis with standardized variables.

lambda

is a $p \times 2$ numerical matrix. In the first column we find the eigenvalues of V , where V is the correlation $p \times p$ matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr

are the principal factors: eigenvectors of V . Each column is an eigenvector element of the dual of R^p .

comprinc

are the principal components. Each column ($c_i = Xu_i$) of this $n \times n$ matrix is the M -orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables $x_1, \dots, x_p$ with maximum variance under condition $u_i' M^{-1} u_i = 1$

Description

This function performs several computations known as "principal component analysis".

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k -dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k -dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

Warning, the graphical part of the old version of **pca** as been removed. It can now be performed using the **show_pca** function.

Examples

```
a=rand(100,10,'n');
[lambda,facpr,comprinc] = pca(a);
show_pca(lambda,facpr)
```

See Also

show_pca , **princomp**

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

perctl — computation of percentils

```
p=perctl(x,y)
```

Parameters

x

real or complex vector or matrix

y

vector of positif integer values between 0 and 100.

Description

Compute the matrix p of percentils (in increasing order, column first) of the real vector or matrix x.

The percentils are indicated by the entries of y, the values of entries of y must be positive integers between 0 and 100.

p is a matrix whose type is length(y) x 2 and the content of its first column are the percentils values.

The contents of its second column are the places of the computed percentiles in the input matrix x.

The minimum or maximum values in x are assigned to percentiles for percent values outside that range.

Examples

```
x=[ 6   7   0   7  10   4   2   2   7   1;  
    6   0   5   5   5   2   0   6   8  10;  
    8   6   4   3   5   9   8   3   4   7;  
    1   3   2   7   6   1   1   4   8   2;  
    6   3   5   1   6   5   9   9   5   5;  
    1   6   4   4   5   4   0   8   1   8;  
    7   1   3   7   8   0   2   8  10   8;  
    3   6   1   9   8   5   5   3   2   1;  
    5   7   6   2  10   8   7   4   0   8;  
   10   3   3   4   8   6   9   4   8   3]  
y=[10 20 30]  
p=perctl(x,y)
```

Authors

Carlos Klimann

Bibliography

HYNDMAN,Rob J. and FAN Yanan, Sample Quantiles in Statistical Packages, The American Statistician, Nov.1996, Vol 50, No.4

Name

princomp — Principal components analysis

```
[facpr,comprinc,lambda,tsquare] = princomp(x,eco)
```

Parameters

x

is a n -by- p (n individuals, p variables) real matrix.

eco

a boolean, use to allow economy size singular value decomposition.

facpr

A p -by- p matrix. It contains the principal factors: eigenvectors of the correlation matrix V .

comprinc

a n -by- p matrix. It contains the principal components. Each column of this matrix is the M -orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables $x_1, \dots, x_p$ with maximum variance under condition $u'_i M^{-1} u_i = 1$

lambda

is a p column vector. It contains the eigenvalues of V , where V is the correlation matrix.

tsquare

a n column vector. It contains the Hotelling's T^2 statistic for each data point.

Description

This function performs "principal component analysis" on the n -by- p data matrix x .

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k -dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k -dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

To compute principal component analysis with standardized variables may use `princomp(wcenter(x,1))` or use the `pca` function.

Examples

```
a=rand(100,10,'n');  
[facpr,comprinc,lambda,tsquare] = princomp(a);
```

See Also

`wcenter` , `pca`

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

quart — computation of quartiles

```
s=quart(x)
s=quart(x, 'r') or m=quart(x,1)
s=quart(x, 'c') or m=quart(x,2)
```

Parameters

x
real or complex vector or matrix

Description

For a vector or a matrix x, [q]=quart(x,y) returns in the vector q the quartiles of x. [q]=quart(x,'r') (or, equivalently, [q]=quart(x,1)) are the rowwise percentiles. It returns in each column of the matrix q the quartiles of data in the corresponding column of x.

[q]=quart(x,'c') (or, equivalently, [q]=quart(x,2)) are the columnwise quartiles. It returns in each row of the matrix q the quartiles of data in the corresponding row of x.

Examples

```
x=[6 7 0 7 10 4 2 2 7 1;
    6 0 5 5 5 2 0 6 8 10;
    8 6 4 3 5 9 8 3 4 7;
    1 3 2 7 6 1 1 4 8 2;
    6 3 5 1 6 5 9 9 5 5;
    1 6 4 4 5 4 0 8 1 8;
    7 1 3 7 8 0 2 8 10 8;
    3 6 1 9 8 5 5 3 2 1;
    5 7 6 2 10 8 7 4 0 8;
    10 3 3 4 8 6 9 4 8 3]
q=quart(x)
q=quart(x, 'r')
q=quart(x, 'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

regress — regression coefficients of two variables

```
coefs=regress(x,y)
```

Parameters

`x,y`
real or complex vector

Description

This function computes the regresion coefficients of two variables `x` and `y`, both numerical vectors of same number of elements `n`. `coefs=[a b]` be a 1x2 matrix such that $Y=a+bX$ will be the equation of the ordinary least square approximation to our data.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[0.5608486 0.6623569 0.7263507 0.1985144 0.5442573 0.2320748 0.2312237]
y=[0.3616361 0.2922267 0.5664249 0.4826472 0.3321719 0.5935095 0.5015342]
coefs=regress(x,y)
```

See Also

`covar`

Authors

Carlos Klimann

Name

sample — Sampling with replacement

```
s = sample(n,X,orient)
```

Parameters

- n
positive integer (size of sample)
- X
matrix. Samples will be extracted from this matrix.
- orient
Optional parameter. Admissible values are 1, 2, 'r' or 'c'
- s
vector or matrix containing sample

Description

This function gives a vector (or matrix) $n \times 1$. It contains a random sample of n extractions, with replacement, from the matrix X .

$s = \text{sample}(n,X)$ (or $s = \text{sample}(n,X,*)$) returns a vector s whose values are a random sample of n values from X , extracted with replacement, from X .

$s = \text{sample}(n,X,'r')$ (or, equivalently, $s = \text{sample}(n,X,1)$) returns a matrix of type $\text{size}(X,'r') \times n$. It contains a random sample of n rows, extracted with replacement, from the rows of X .

$s = \text{sample}(n,X,'c')$ (or, equivalently, $s = \text{sample}(n,X,2)$) returns a matrix of type $n \times \text{size}(X,'c')$. It contains a random sample of n columns, extracted with replacement from the columns of X .

Examples

```
X=['a' 'dd' 'arreu'; 'ber' 'car' 'zon']
s=sample(25,X)
s=sample(25,X,'r')
s=sample(25,X,'c')
```

See Also

samplef , samwr

Authors

Carlos Klimann

Name

samplef — sample with replacement from a population and frequencies of his values.

```
s = samplef(n,X,f,orient)
```

Parameters

- n**
positive integer (size of sample)
- X**
matrix. Samples will be extracted from this matrix
- f**
positive integer matrix with same type than X. It indicates frequencies of corresponding values of X.
- orient**
Optional parameter. Admissible values are 1, 2, 'r' or 'c'
- s**
vector or matrix containing sample

Description

This function gives s, a vector of lenght n. It contains a sample of n extractions, with replacement, from the vector (or matrix) X, each element counted with the frequency given by the corresponding value in vector f.

s=samplef(n,X,f) (or s=samplef(n,X,f,*)) returns a vector s whose values are a random sample of n values from X, each value with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from X. f must have same lenght than X.

s=samplef(n,X,f,'r') (or, equivalently, s=samplef(n,X,f,1)) returns a matrix of type size(X,'r')xn. It contains a random sample of n rows from X, each row with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the rows of X. The lenght of f must be equal to the number of rows of X.

s=samplef(n,X,f,'c') (or, equivalently, s=samplef(n,X,f,2)) returns a matrix of type nxsize(X,'c'). It contains a random sample of n columns from X, each column with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the columns of X. The lenght of f must be equal to the number of columns of X.

Examples

```
a=[3 7 9;22 4 2]
f1=[10 1 1 1 1 1]
f2=[1 ; 15]
f3=[10 1 1]
s=samplef(15,a,f1)
s=samplef(15,a,f2,'r')
s=samplef(15,a,f3,'c')
```

See Also

sample , samwr

Authors

Carlos Klimann

Name

samwr — Sampling without replacement

```
s = samwr(sizam, numsamp, X)
```

Parameters

sizam

integer. Size of a sample. It must be less or equal than size of X.

numsamp

integer. Number of samples to be extracted.

X

column vector. It contains the population.

s

matrix of type sizsam x numsamp. It contains numsamp random samples (the columns) each of sizsam (size(X, '*')) extractions, without replacement, from the column vector X.

Description

Gives samples without replacement from a column vector.

Examples

```
a=[0.33 1.24 2.1 1.03]
s=samwr(4,12,a)
```

See Also

sample , samplef

Authors

Carlos Klimann

Name

show_pca — Visualization of principal components analysis results

```
show_pca( lambda, facpr, N)
```

Parameters

lambda

is a $p \times 2$ numerical matrix. In the first column we find the eigenvalues of V , where V is the correlation $p \times p$ matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr

are the principal factors: eigenvectors of V . Each column is an eigenvector element of the dual of R^p .

N

Is a 2×1 integer vector. Its coefficients point to the eigenvectors corresponding to the eigenvalues of the correlation matrix p by p ordered by decreasing values of eigenvalues. If N is missing, we suppose $N = [1 \ 2]^T$.

Description

This function visualize the pca results.

Examples

```
a=rand(100,10,'n');  
[lambda,facpr,comprinc] = pca(a);  
show_pca(lambda,facpr)
```

See Also

pca , princomp

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

`st_deviation` — standard deviation (row or column-wise) of vector/matrix entries
`stdev` — standard deviation (row or column-wise) of vector/matrix entries

```
y=st_deviation(x)
y=st_deviation(x, 'r')
y=st_deviation(x, 'c')
y=stdev(x)
y=stdev(x, 'r')
y=stdev(x, 'c')
```

Parameters

`x`
real vector or matrix

`y`
scalar or vector

Description

`st_deviation` computes the "sample" standard deviation, that is, it is normalized by $N-1$, where N is the sequence length.

For a vector or a matrix `x`, `y=st_deviation(x)` returns in the scalar `y` the standard deviation of all the entries of `x`.

`y=st_deviation(x, 'r')` (or, equivalently, `y=st_deviation(x, 1)`) is the rowwise standard deviation. It returns in each entry of the column vector `y` the standard deviation of each row of `x`.

`y=st_deviation(x, 'c')` (or, equivalently, `y=st_deviation(x, 2)`) is the columnwise standard deviation. It returns in each entry of the row vector `y` the standard deviation of each column of `x`.

Examples

```
A=[1,2,10;7,7.1,7.01];
st_deviation(A)
st_deviation(A, 'r')
st_deviation(A, 'c')
```

See Also

`sum` , `median` , `mean` , `nanstdev` , `stdevf`

Name

stdevf — standard deviation

```
s=stdevf(x, fre)
s=stdevf(x, fre, 'r') or s=stdevf(x, fre, 1)
s=stdevf(x, fre, 'c') or s=stdevf(x, fre, 2)
```

Parameters

x
real or complex vector or matrix

Description

This function computes the standard deviation of the values of a vector or matrix **x**, each of them counted with a frequency given by the corresponding values of the integer vector or matrix **fre** who has the same type of **x**.

For a vector or matrix **x**, **s=stdevf(x, fre)** (or **s=stdevf(x, fre, '*')**) returns in scalar **s** the standard deviation of all the entries of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

s=stdevf(x, fre, 'r') (or, equivalently, **s=stdevf(x, fre, 1)**) returns in each entry of the row vector **s** of type `1xsize(x,'c')` the standard deviation of each column of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

s=stdevf(x, fre, 'c') (or, equivalently, **s=stdevf(x, fre, 2)**) returns in each entry of the column vector **s** of type `size(x,'c')x1` the standard deviation of each row of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=stdevf(x, fre)
m=stdevf(x, fre, 'r')
m=stdevf(x, fre, 'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

strange — range

```
[r]=strange(x)
[r]=strange(x,'r') (or, equivalently, [r]=strange(x,1))
[r]=strange(x,'c') (or, equivalently, [r]=strange(x,2))
```

Parameters

x
real or complex vector or matrix

Description

The range is the distance between the largest and smaller value, `[r]=strange(x)` computes the range of vector or matrix x .

`[r]=strange(x,'r')` (or equivalently `[r]=strange(x,1)`) give a row vector with the range of each column.

`[r]=strange(x,'c')` (or equivalently `[r]=strange(x,2)`) give a column vector with the range of each row.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Authors

Carlos klimann

Name

tabul — frequency of values of a matrix or vector

```
[m]=tabul(X [,order])
```

Parameters

X

vector or matrix (of real or complex numbers or strings)

order

(optionnal) a character equal to "d" or "i" (default value "d")

m

a 2 columns matrix (if X is a numerical vector or matrix) or a list with 2 members (if X is a string vector or matrix).

Description

This function computes the frequency of values of the components of a vector or matrix X of numbers or string characters :

if X is a numerical vector or matrix

then m is a two column matrix who contains in the first column the distinct values of X and in the other column the number of occurrences of those values (m(i,2) is the number of occurrences of m(i,1)).

if X is a string vector or matrix

then m is a list whose first member is a string (column) vector composed with the distinct values of X and the second member is a (column) vector whose components are the number of occurrences of those values (m(i)(2) is the number of occurrences of the string m(i)(1)).

The optional parameter **order** must be "d" or "i" (by default order="d") and gives the order (decreasing or increasing) the distinct values of X will be sorted.

Examples

```
// first example
X = [2 8 0 3 7 6 8 7 9 1 6 7 7 2 5 2 2 2 9 7]
m1 = tabul(X)
m2 = tabul(X, "i")

// second example
X = ["ba" "baba" "a" "A" "AA" "a" "aa" "aa" "aa" "A" "ba"]
m = tabul(X,"i")

// third example
n = 50000;
X = grand(n,1,"bin",70,0.5);
m = tabul(X,"i");
clf()
plot2d3(m(:,1), m(:,2)/n)
xtitle("empirical probabilities of B(70,0.5)")

// last example : computes the occurrences of words of the scilab license
```

```
text = read(SCI+"/license.txt",-1,1,"(A)"); // read the scilab license
bigstr = strcat(text," "); // put all the lines in a big string
sep = [" " ", " "." ", " *" " ":" "-" "" ""]; // words separators
words = tokens(bigstr, sep); // cut the big string into words
m = tabul(words); // computes occurrences of each word
[occ , p] = gsort(m(2)); // sort by decreasing frequencies
results = [m(1)(p) string(occ)] // display result
```

See Also

dsearch , histplot

Authors

Carlos Klimann (original author)
J.S. Giet and B. Pincon (new version)

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

thrownan — eliminates nan values

```
[nonan, numb]=thrownan(x)
```

Parameters

x
real or complex vector or matrix

Description

This function returns in vector **nonan** the values (ignoring the NaNs) of a vector or matrix **x** and in the corresponding places of vector **numb** the indexes of the value.

For a vector or matrix **x**, **[nonan, numb]=thrownan(x)** considers **x**, whatever his dimensions are, like a vector (columns first).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]  
[nonan numb]=thrownan(x)
```

Authors

Carlos Klimann

Name

trimmean — trimmed mean of a vector or a matrix

```
m=trimmean(x[,discard [,flag [,verbose]]])
```

Parameters

x

real or complex vector or matrix

discard

Optional real value between 0 and 100 representing the part of the data to discard. If discard is not in the [0,100] range, an error is generated. Default value for discard=50.

flag

Optional string or real parameter which controls the behaviour when x is a matrix. Available values for flag are : "all", 1, 2, r or c (default is flag="all"). The two values flag=r and flag=1 are equivalent. The two values flag=c and flag=2 are equivalent.

verbose

Optional integer. If set to 1, then enables verbose logging. Default is 0.

Description

A trimmed mean is calculated by discarding a certain percentage of the lowest and the highest scores and then computing the mean of the remaining scores. For example, a mean trimmed 50% is computed by discarding the lower and higher 25% of the scores and taking the mean of the remaining scores.

The median is the mean trimmed 100% and the arithmetic mean is the mean trimmed 0%.

A trimmed mean is obviously less susceptible to the effects of extreme scores than is the arithmetic mean. It is therefore less susceptible to sampling fluctuation than the mean for extremely skewed distributions. The efficiency of a statistic is the degree to which the statistic is stable from sample to sample. That is, the less subject to sampling fluctuation a statistic is, the more efficient it is. The efficiency of statistics is measured relative to the efficiency of other statistics and is therefore often called the relative efficiency. If statistic A has a smaller standard error than statistic B, then statistic A is more efficient than statistic B. The relative efficiency of two statistics may depend on the distribution involved. For instance, the mean is more efficient than the median for normal distributions but not for some extremely skewed distributions. The efficiency of a statistic can also be thought of as the precision of the estimate: The more efficient the statistic, the more precise the statistic is as an estimator of the parameter. The trimmed mean is less efficient than the mean for normal distributions.

For a vector or matrix **x**, **t=trimmean(x,discard)** returns in scalar **t** the mean of all the entries of **x**, after discarding **discard/2** highest values and **discard/2** lowest values.

t=trimmean(x,discard,'r') (or, equivalently, **t=trimmean(x,discard,1)**) returns in each entry of the row vector **t** the trimmed mean of each column of **x**.

t=trimmean(x,discard,'c') (or, equivalently, **t=trimmean(x,discard,2)**) returns in each entry of the column vector **t** the trimmed mean of each row of **x**.

This function computes the trimmed mean of a vector or matrix **x**.

For a vector or matrix **x**, **m=trimmean(x,discard)** returns in scalar **m** the trimmed mean of all the entries of **x**.

m=trimmean(x,'r') (or, equivalently, **m=trimmean(x,1)**) returns in each entry of the row vector **m** the trimmed mean of each column of **x**.

`m=trimmean(x,'c')` (or, equivalently, `m=trimmean(x,2)`) returns in each entry of the column vector `m` the trimmed mean of each row of `x`.

Example with x as vector

In the following example, one computes the trimmed mean of one data vector, with the default discard value equals to 50 and verbose logging. The data is made of 9 entries. The algorithm sorts the vector and keeps only indices from 3 to 7, skipping indices 1, 2, 8 and 9. The value 4000, which is much larger than the others is not taken into account. The computed trimmed mean is therefore 50.

```
data = [10, 20, 30, 40, 50, 60, 70, 80, 4000];
computed = trimmean(data,verbose=1);
```

Example with x as matrix

In the following example, one computes the trimmed mean of one data matrix. The chosen discard value is 50. The orientation is "r", which means that the data is sorted row by row. For each column of the matrix, one computes a trimmed mean. The trimmed mean is the line vector [25 25 25 25].

```
data = [10 10 10 10
        20 20 20 20
        30 30 30 30
        4000 4000 4000 4000];
computed = trimmean(data,50,orien="r");
```

References

Luis Angel Garcia-Escudero and Alfonso Gordaliza, Robustness Properties of Means and Trimmed Means, JASA, Volume 94, Number 447, Sept 1999, pp956-969

Trimmed Mean, <http://davidmlane.com/hyperstat/A11971.html>

Authors

Carlos Klimann

Name

variance — variance of the values of a vector or matrix

```
s=variance(x[,orien[,w]])  
s=variance(x,'r') or m=variance(x,1)  
s=variance(x,'c') or m=variance(x,2)
```

Parameters

x

real or complex vector or matrix

orien

the orientation of the computation. Valid values or the orien parameter are 1, "r", 2 and "c".

w

w : type of normalization to use. Valid values are 0 and 1. This depends on the number of columns of x (if orien = 1 is chosen), the number of rows (if orien = 2 is chosen). If w = 0, normalizes with m-1, provides the best unbiased estimator of the variance (this is the default). If w = 1, normalizes with m, this provides the second moment around the mean. If no orien option is given, the normalization is done with $n * m - 1$, where $n * m$ is the total number of elements in the matrix.

Description

This function computes the variance of the values of a vector or matrix x.

For a vector or a matrix x, `s=variance(x)` returns in the scalar s the variance of all the entries of x.

`s=variance(x,'r')` (or, equivalently, `s=variance(x,1)`) is the rowwise variance. It returns in each entry of the row vector s the variance of each column of x. The generalized formulae is used, which manages complex values.

`s=variance(x,'c')` (or, equivalently, `s=variance(x,2)`) is the columnwise standard deviation. It returns in each entry of the column vector s the variance of each row of x. The generalized formulae is used, which manages complex values.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]  
s=variance(x)  
s=variance(x,'r')  
s=variance(x,'c')
```

See Also

mtlb_var

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

variancef — standard deviation of the values of a vector or matrix

```
s=variancef(x,fre)
s=variancef(x,fre,'r') or s=variancef(x,fre,1)
s=variancef(x,fre,'c') or s=variancef(x,fre,2)
```

Parameters

x
real or complex vector or matrix

Description

This function computes the variance of the values of a vector or matrix **x**, each of them counted with a frequency signaled by the corresponding values of the integer vector or matrix **fre** with the same type of **x**.

For a vector or matrix **x**, **s=variancef(x,fre)** (or **s=variancef(x,fre,'*')**) returns in scalar **s** the variance of all the entries of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

s=variancef(x,fre,'r') (or, equivalently, **s=variancef(x,fre,1)**) returns in each entry of the row vector **s** of type `1xsize(x,'c')` the variance of each column of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

s=variancef(x,fre,'c') (or, equivalently, **s=variancef(x,fre,2)**) returns in each entry of the column vector **s** of type `size(x,'c') x1` the variance of each row of **x**, each value counted with the multiplicity indicated by the corresponding value of **fre**.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=variancef(x,fre)
m=variancef(x,fre,'r')
m=variancef(x,fre,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Part XXXV. ARnoldi PACKage

Name

dnaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real linear operator

`[ ID0, RESID, V, IPARAM, IPNTR, WORKD, WORKL, INFO ] = dnaupd( ID0, BMAT, N, WHICH, NEV, TOL, R`

Parameters

ID0

Integer. (INPUT/OUTPUT) Reverse communication flag. ID0 must be zero on the first call to dnaupd. ID0 will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dnaupd with the result. The operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)).

- ID0 = 0: first call to the reverse communication interface
- ID0 = -1: compute $Y = OP * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.
- ID0 = 1: compute $Y = OP * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3 and 4, the vector $B * X$ is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming $OP * X$.
- ID0 = 2: compute $Y = B * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.
- ID0 = 3: compute the IPARAM(8) real and imaginary parts of the shifts where IPNTR(14) is the pointer into WORKL for placing the shifts. See Remark 5 below.
- ID0 = 99: done

BMAT

Character, specifies the type of the matrix B that defines the semi-inner product for the operator OP.

- 'I' - standard eigenvalue problem $A*x = \lambda*x$
- 'G' - generalized eigenvalue problem $A*x = \lambda*B*x$

N

Integer, dimension of the eigenproblem.

WHICH

string of length 2, Specify which of the Ritz values of OP to compute.

- 'LM' - want the NEV eigenvalues of largest magnitude.
- 'SM' - want the NEV eigenvalues of smallest magnitude.
- 'LR' - want the NEV eigenvalues of largest real part.
- 'SR' - want the NEV eigenvalues of smallest real part.
- 'LI' - want the NEV eigenvalues of largest imaginary part.
- 'SI' - want the NEV eigenvalues of smallest imaginary part.

NEV

Integer, number of eigenvalues of OP to be computed. $0 < NEV < N-1$.

TOL

scalar. Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if $\text{BOUNDS}(I) \leq \text{TOL} * \text{ABS}(\text{RITZ}(I))$. If $\text{TOL} \leq 0$, the machine precision is set.

RESID

array of length N (INPUT/OUTPUT)

On INPUT: If $\text{INFO}=0$, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV

Integer, number of columns of the matrix V. NCV must satisfy the two inequalities $2 \leq \text{NCV} - \text{NEV}$ and $\text{NCV} \leq N$.

This will indicate how many Arnoldi vectors are generated at each iteration.

After the startup phase in which NEV Arnoldi vectors are generated, the algorithm generates approximately $\text{NCV} - \text{NEV}$ Arnoldi vectors at each subsequent update iteration. Most of the cost in generating each Arnoldi vector is in the matrix-vector operation $\text{OP} * \mathbf{x}$.

NOTE: $2 \leq \text{NCV} - \text{NEV}$ in order that complex conjugate pairs of Ritz values are kept together. (See remark 4 below)

V

N by NCV array. Contains the final set of Arnoldi basis vectors.

IPARAM

array of length 11. (INPUT/OUTPUT)

- $\text{IPARAM}(1) = \text{ISHIFT}$: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.
 - $\text{ISHIFT} = 0$: the shifts are provided by the user via reverse communication. The real and imaginary parts of the NCV eigenvalues of the Hessenberg matrix H are returned in the part of the WORKL array corresponding to RITZR and RITZI. See remark 5 below.
 - $\text{ISHIFT} = 1$: exact shifts with respect to the current Hessenberg matrix H. This is equivalent to restarting the iteration with a starting vector that is a linear combination of approximate Schur vectors associated with the "wanted" Ritz values.

- $\text{IPARAM}(2) = \text{LEVEC}$ No longer referenced.

- $\text{IPARAM}(3) = \text{MXITER}$

On INPUT: maximum number of Arnoldi update iterations allowed.

On OUTPUT: actual number of Arnoldi update iterations taken.

- $\text{IPARAM}(4) = \text{NB}$: blocksize to be used in the recurrence. The code currently works only for $\text{NB} = 1$.
- $\text{IPARAM}(5) = \text{NCONV}$: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.
- $\text{IPARAM}(6) = \text{IUPD}$ No longer referenced. Implicit restarting is ALWAYS used.
- $\text{IPARAM}(7) = \text{MODE}$ On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4; See under Description of dnaupd for the five modes available.
- $\text{IPARAM}(8) = \text{NP}$ When $\text{ido} = 3$ and the user provides shifts through reverse communication ($\text{IPARAM}(1)=0$), dnaupd returns NP, the number of shifts the user is to provide.

$0 < NP \leq NCV - NEV$. See Remark 5 below.

- IPARAM(9) = NUMOP,
IPARAM(10) = NUMOPB,
IPARAM(11) = NUMREO,

On OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR

array of length 14. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Arnoldi iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.
- IPNTR(2): pointer to the current result vector Y in WORKD.
- IPNTR(3): pointer to the vector $B * X$ in WORKD when used in the shift-and-invert mode.
- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.
- IPNTR(5): pointer to the NCV by NCV upper Hessenberg matrix H in WORKL.
- IPNTR(6): pointer to the real part of the ritz value array RITZR in WORKL.
- IPNTR(7): pointer to the imaginary part of the ritz value array RITZI in WORKL.
- IPNTR(8): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZR and RITZI in WORKL.
- IPNTR(14): pointer to the NP shifts in WORKL. See Remark 5 below.

Note: IPNTR(9:13) is only referenced by dneupd . See Remark 2.

- IPNTR(9): pointer to the real part of the NCV RITZ values of the original system.
- IPNTR(10): pointer to the imaginary part of the NCV RITZ values of the original system.
- IPNTR(11): pointer to the NCV corresponding error bounds.
- IPNTR(12): pointer to the NCV by NCV upper quasi-triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by dneupd if RVEC = 1 See Remark 2 below.

WORKD

Double precision work array of length $3*N$. (REVERSE COMMUNICATION)

Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If an invariant subspace associated with the converged Ritz values is desired, see remark 2 below, subroutine dneupd uses this output. See Data Distribution Note below.

WORKL

work array of length at least $3*NCV**2 + 6*NCV$. (OUTPUT/WORKSPACE) Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below.

INFO

Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

- 0: Normal exit.
- 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
- 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
- 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV ≥ 2 and less than or equal to N.
- -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work array WORKL is not sufficient.
- -8: Error return from LAPACK eigenvalue calculation;
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: IPARAM(1) must be equal to 0 or 1.
- -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

Description

Reverse communication interface for the Implicitly Restarted Arnoldi iteration. This subroutine computes approximations to a few eigenpairs of a linear operator "OP" with respect to a semi-inner product defined by a symmetric positive semi-definite real matrix B. B may be the identity matrix. NOTE: If the linear operator "OP" is real and symmetric with respect to the real positive semi-definite symmetric matrix B, i.e. $B*OP = (OP^T)*B$, then subroutine dsaupd should be used instead.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dnaupd is usually called iteratively to solve one of the following problems:

- Mode 1: $A*x = \lambda*x$. $OP = A$, $B = I$.
- Mode 2: $A*x = \lambda*M*x$, M symmetric positive definite $OP = \text{inv}[M]*A$, $B = M$. (If M can be factored see remark 3 below)
- Mode 3: $A*x = \lambda*M*x$, M symmetric positive semi-definite. $OP = \text{Real\_Part}\{ \text{inv}[A - \sigma*M]*M \}$, $B = M$. shift-and-invert mode (in real arithmetic)

If $OP*x = \text{amu}*x$, then

$$\text{amu} = 1/2 * [1/(\lambda - \sigma) + 1/(\lambda - \text{conj}(\sigma))].$$

Note: If sigma is real, i.e. imaginary part of sigma is zero; $\text{Real\_Part}\{ \text{inv}[A - \text{sigma}*M]*M \} == \text{inv}[A - \text{sigma}*M]*M \text{amu} == 1/(\text{lambda}-\text{sigma})$.

- Mode 4: $A*x = \text{lambda}*M*x$, M symmetric semi-definite $OP = \text{Imaginary\_Part}\{ \text{inv}[A - \text{sigma}*M]*M \}$, $B = M$. shift-and-invert mode (in real arithmetic)

If $OP*x = \text{amu}*x$, then $\text{amu} = 1/2i * [1/(\text{lambda}-\text{sigma}) - 1/(\text{lambda}-\text{conj}(\text{sigma}))]$.

Both mode 3 and 4 give the same enhancement to eigenvalues close to the (complex) shift sigma. However, as lambda goes to infinity, the operator OP in mode 4 dampens the eigenvalues more strongly than does OP defined in mode 3.

NOTE: The action of $w \leftarrow \text{inv}[A - \text{sigma}*M]*v$ or $w \leftarrow \text{inv}[M]*v$ should be accomplished either by a direct method using a sparse matrix factorization and solving $[A - \text{sigma}*M]*w = v$ or $M*w = v$, or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

Example

```
// The following sets dimensions for this problem.

nx      = 10;

nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workev  = zeros(3*ncv,1);
workl   = zeros(3*ncv*ncv+6*ncv,1);

// Build the test matrix

A       = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol     = 0;
ido     = 0;

ishfts  = 1;
maxitr  = 300;
mode1   = 1;

iparam(1) = ishfts;
```

```
iparam(3) = maxitr;
iparam(7) = mode1;

sigmar = 0; // the real part of the shift
sigmai = 0; // the imaginary part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxitr)
  info_dnaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dnaupd] = dnaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL dneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DNAUPD again.
    continue
  end

  if (info_dnaupd < 0) then
    printf('\nError with dnaupd, info = %d\n',info_dnaupd);
    printf('Check the documentation of dnaupd\n\n');
  else
    // Post-Process using DNEUPD.

    rvec      = 1;
    howmany   = 'A';

    info_dneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dneupd] = dneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i

    if (info_dneupd~=0) then
      printf('\nError with dneupd, info = %d\n', info_dneupd);
      printf('Check the documentation of dneupd.\n\n');
    end

    // Print additional convergence information.
    if (info_dneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end
end
```

```

printf('\nDNSIMP\n');
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam);
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);

```

Remarks

1. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3 and 4. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dneupd.
2. If a basis for the invariant subspace corresponding to the converged Ritz values is needed, the user must call dneupd immediately following completion of dnaupd. This is new starting with release 2 of ARPACK.
3. If M can be factored into a Cholesky factorization $M = LL^T$ then Mode = 2 should not be selected. Instead one should use Mode = 1 with $OP = \text{inv}(L) * A * \text{inv}(L^T)$. Appropriate triangular linear systems should be solved with L and L^T rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving $L^T z = x$ where x is a Ritz vector of OP.
4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requirement is that $NCV > NEV + 2$. However, it is recommended that $NCV \geq 2 * NEV + 1$. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically. See Chapter 8 of Reference 2 for further information.
5. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) real and imaginary parts of the shifts in locations

real part	imaginary part
-----	-----
1 WORKL(IPNTR(14))	WORKL(IPNTR(14)+NP)
2 WORKL(IPNTR(14)+1)	WORKL(IPNTR(14)+NP+1)
.	.
.	.
.	.
NP WORKL(IPNTR(14)+NP-1)	WORKL(IPNTR(14)+2*NP-1) .

Only complex conjugate pairs of shifts may be applied and the pairs must be placed in consecutive locations. The real part of the eigenvalues of the current upper Hessenberg matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1) and the imaginary part in WORKL(IPNTR(7)) through WORKL(IPNTR(7)+NCV-1). They are ordered according to the order defined by WHICH. The complex conjugate pairs are kept together and the associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

See Also

dsaupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett, "The Symmetric Eigenvalue Problem". Prentice-Hall, 1980.
4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.
5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.
6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).
7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.
8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

Used Functions

Based on ARPACK routine dnaupd

Name

dneupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute the converged approximations to eigenvalues of $A*z = \lambda B*z$ approximations to a few eigenpairs of a real linear operator

```
[Dr, Di, Z, RESID, V, IPARAM, IPNTR, WORKD, WORKL, INFO] = dneupd(RVEC, HOWMANY, SELECT, DR, DI, Z, RESID, V, IPARAM, IPNTR, WORKD, WORKL, INFO, BMAT, N, WHICH, NEV, TOL, R0)
```

Parameters

RVEC

Integer (INPUT)

Specifies whether a basis for the invariant subspace corresponding to the converged Ritz value approximations for the eigenproblem $A*z = \lambda B*z$ is computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute the Ritz vectors or Schur vectors.

See Remarks below.

HOWMANY

Character (INPUT)

Specifies the form of the basis for the invariant subspace corresponding to the converged Ritz values that is to be computed.

- 'A': Compute NEV Ritz vectors;
- 'P': Compute NEV Schur vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT

SELECT

Integer array of dimension NCV. (INPUT)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value (DR(j), DI(j)), SELECT(j) must be set to 1.

If HOWMANY = 'A' or 'P', SELECT is used as internal workspace.

DR

Double precision array of dimension NEV+1. (OUTPUT)

If IPARAM(7) = 1, 2 or 3 and SIGMAI=0.0 then on exit: DR contains the real part of the Ritz approximations to the eigenvalues of $A*z = \lambda B*z$.

If IPARAM(7) = 3, 4 and SIGMAI is not equal to zero, then on exit: DR contains the real part of the Ritz values of OP computed by DNAUPD .

A further computation must be performed by the user to transform the Ritz values computed for OP by DNAUPD to those of the original system $A*z = \lambda B*z$. See remark 3 below

DI

Double precision array of dimension NEV+1. (OUTPUT)

On exit, DI contains the imaginary part of the Ritz value approximations to the eigenvalues of $A*z = \lambda B*z$ associated with DR.

NOTE: When Ritz values are complex, they will come in complex conjugate pairs. If eigenvectors are requested, the corresponding Ritz vectors will also come in conjugate pairs and the real and imaginary parts of these are represented in two consecutive columns of the array Z (see below).

Z

Double precision N by NEV+1 array

if RVEC = 1 and HOWMANY = 'A'. (OUTPUT)

On exit, if RVEC = 1 and HOWMANY = 'A', then the columns of Z represent approximate eigenvectors (Ritz vectors) corresponding to the NCONV=IPARAM(5) Ritz values for eigensystem $A*z = \lambda*B*z$. The complex Ritz vector associated with the Ritz value with positive imaginary part is stored in two consecutive columns. The first column holds the real part of the Ritz vector and the second column holds the imaginary part. The Ritz vector associated with the Ritz value with negative imaginary part is simply the complex conjugate of the Ritz vector associated with the positive imaginary part.

If RVEC = 0 or HOWMANY = 'P', then Z is not referenced.

NOTE: If if RVEC = 1 and a Schur basis is not required, the array Z may be set equal to first NEV+1 columns of the Arnoldi basis array V computed by DNAUPD . In this case the Arnoldi basis will be destroyed and overwritten with the eigenvector basis.

SIGMAR

Double precision (INPUT)

If IPARAM(7) = 3 or 4, represents the real part of the shift.

Not referenced if IPARAM(7) = 1 or 2.

SIGMAI

Double precision (INPUT)

If IPARAM(7) = 3 or 4, represents the imaginary part of the shift.

Not referenced if IPARAM(7) = 1 or 2. See remark 3 below.

WORKEV

Double precision work array of dimension 3*NCV. (WORKSPACE)

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, INFO must be passed directly to DNEUPD following the last call to DNAUPD .

These arguments MUST NOT BE MODIFIED between the the last call to DNAUPD and the call to DNEUPD .

Three of these parameters (V, WORKL, INFO) are also output parameters.

V

Double precision N by NCV array. (INPUT/OUTPUT)

Upon INPUT: the NCV columns of V contain the Arnoldi basis vectors for OP as constructed by DNAUPD.

Upon OUTPUT: If RVEC = 1 the first NCONV=IPARAM(5) columns contain approximate Schur vectors that span the desired invariant subspace. See Remark 2 below.

NOTE: If the array Z has been set equal to first NEV+1 columns of the array V and RVEC=1 and HOWMANY= 'A', then the Arnoldi basis held by V has been overwritten by the desired Ritz vectors. If a separate array Z has been passed then the first NCONV=IPARAM(5) columns of V will contain approximate Schur vectors that span the desired invariant subspace.

WORKL

Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:ncv*ncv+3*ncv) contains information obtained in dnaupd . They are not changed by dneupd .

WORKL(ncv*ncv+3*ncv+1:3*ncv*ncv+6*ncv) holds the real and imaginary part of the untransformed Ritz values, the upper quasi-triangular matrix for H, and the associated matrix representation of the invariant subspace for H.

Note: IPNTR(9:13) contains the pointer into WORKL for addresses of the above information computed by dneupd .

- IPNTR(9): pointer to the real part of the NCV RITZ values of the original system.
- IPNTR(10): pointer to the imaginary part of the NCV RITZ values of the original system.
- IPNTR(11): pointer to the NCV corresponding error bounds.
- IPNTR(12): pointer to the NCV by NCV upper quasi-triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by dneupd if RVEC = 1 See Remark 2 below.

INFO

Error flag on output.

- 0: Normal exit.
- 1: The Schur form computed by LAPACK routine dlahqr could not be reordered by LAPACK routine dtrsen . Re-enter subroutine dneupd with IPARAM(5)=NCV and increase the size of the arrays DR and DI to have dimension at least dimension NCV and allocate at least NCV columns for Z.

NOTE: Not necessary if Z and V share the same space. Please notify the authors if this error occurs.

- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV ≥ 2 and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'T' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from calculation of a real Schur form. Informational error from LAPACK routine dlahqr .
- -9: Error return from calculation of eigenvectors. Informational error from LAPACK routine dtrevc .
- -10: IPARAM(7) must be 1,2,3,4.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: HOWMANY = 'S' not yet implemented
- -13: HOWMANY must be one of 'A' or 'P' if RVEC = 1
- -14: DNAUPD did not find any eigenvalues to sufficient accuracy.
- -15: DNEUPD got a different count of the number of converged Ritz values than DNAUPD got. This indicates the user probably made an error in passing data from DNAUPD to DNEUPD or that the data was modified before entering DNEUPD

Description

This subroutine returns the converged approximations to eigenvalues of $A*z = \lambda*B*z$ and (optionally):

1. The corresponding approximate eigenvectors;
2. An orthonormal basis for the associated approximate invariant subspace;
3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal basis is always computed.

There is an additional storage cost of $n*nev$ if both are requested (in this case a separate array Z must be supplied).

The approximate eigenvalues and eigenvectors of $A*z = \lambda*B*z$ are derived from approximate eigenvalues and eigenvectors of the linear operator OP prescribed by the $MODE$ selection in the call to $DNAUPD$. $DNAUPD$ must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Schur basis.

See documentation in the header of the subroutine $DNAUPD$ for definition of OP as well as other terms and the relation of computed Ritz values and Ritz vectors of OP with respect to the given problem $A*z = \lambda*B*z$.

For a brief description, see definitions of $IPARAM(7)$, $MODE$ and $WHICH$ in the documentation of $DNAUPD$.

Remarks

1. Currently only $HOWMNY = 'A'$ and $'P'$ are implemented.

Let $\text{trans}(X)$ denote the transpose of X .

2. Schur vectors are an orthogonal representation for the basis of Ritz vectors. Thus, their numerical properties are often superior. If $RVEC = 1$ then the relationship

$$A * V(:,1:IPARAM(5)) = V(:,1:IPARAM(5)) * T, \text{ and } \text{trans}(V(:,1:IPARAM(5))) * V(:,1:IPARAM(5)) = I$$

are approximately satisfied.

Here T is the leading submatrix of order $IPARAM(5)$ of the real upper quasi-triangular matrix stored $\text{workl}(\text{ipntr}(12))$. That is, T is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign. Corresponding to each 2-by-2 diagonal block is a complex conjugate pair of Ritz values. The real Ritz values are stored on the diagonal of T .

3. If $IPARAM(7) = 3$ or 4 and $SIGMAI$ is not equal zero, then the user must form the $IPARAM(5)$ Rayleigh quotients in order to transform the Ritz values computed by $DNAUPD$ for OP to those of $A*z = \lambda*B*z$. Set $RVEC = 1$ and $HOWMNY = 'A'$, and compute

$$\text{trans}(Z(:,I)) * A * Z(:,I) \text{ if } DI(I) = 0.$$

If $DI(I)$ is not equal to zero and $DI(I+1) = -D(I)$, then the desired real and imaginary parts of the Ritz value are

$$\text{trans}(Z(:,I)) * A * Z(:,I) + \text{trans}(Z(:,I+1)) * A * Z(:,I+1),$$
$$\text{trans}(Z(:,I)) * A * Z(:,I+1) - \text{trans}(Z(:,I+1)) * A * Z(:,I),$$

respectively.

Another possibility is to set RVEC = 1 and HOWMANY = 'P' and compute

$$\text{trans}(V(:,1:\text{IPARAM}(5))) * A * V(:,1:\text{IPARAM}(5))$$

and then an upper quasi-triangular matrix of order IPARAM(5) is computed. See remark 2 above.

Example

```
// The following sets dimensions for this problem.

nx      = 10;

nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workev  = zeros(3*ncv,1);
workl   = zeros(3*ncv*ncv+6*ncv,1);

// Build the test matrix

A          = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol      = 0;
ido      = 0;

ishfts   = 1;
maxitr   = 300;
mode1    = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigmar = 0; // the real part of the shift
sigmai = 0; // the imaginary part of the shift
```

```

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_dnaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dnaupd] = dnaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL dneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DNAUPD again.
    continue
  end

  if (info_dnaupd < 0) then
    printf('\nError with dnaupd, info = %d\n',info_dnaupd);
    printf('Check the documentation of dnaupd\n\n');
  else
    // Post-Process using DNEUPD.

    rvec      = 1;
    howmany = 'A';

    info_dneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dneupd] = dneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i

    if (info_dneupd~=0) then
      printf('\nError with dneupd, info = %d\n', info_dneupd);
      printf('Check the documentation of dneupd.\n\n');
    end

    // Print additional convergence information.
    if (info_dneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

printf('\nDNSIMP\n');
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);

```

```
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam);
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

See Also

dsaupd, dnaupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett, "The Symmetric Eigenvalue Problem". Prentice-Hall, 1980.
4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.
5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.
6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).
7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.
8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

Used Functions

Based on ARPACK routine dneupd

Name

dsaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real and symmetric linear operator

```
[IDO, RESID, V, IPARAM, IPNTR, WORKD, WORKL, INFO] = dsaupd(ID0, BMAT, N, WHICH, NEV, TOL, R)
```

Parameters

IDO

Integer. (INPUT/OUTPUT)

Reverse communication flag. IDO must be zero on the first call to dsaupd . IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dsaupd with the result. T

he operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)). (If Mode = 2 see remark 5 below)

- IDO = 0: first call to the reverse communication interface
- IDO = -1: compute $Y = OP * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.
- IDO = 1: compute $Y = OP * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3,4 and 5, the vector $B * X$ is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming $OP * X$.
- IDO = 2: compute $Y = B * X$ where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.
- IDO = 3: compute the IPARAM(8) shifts where IPNTR(11) is the pointer into WORKL for placing the shifts. See remark 6 below.
- IDO = 99: done

BMAT

Character. (INPUT)

Specifies the type of the matrix B that defines the semi-inner product for the operator OP.

- 'I': standard eigenvalue problem $A*x = \lambda*x$
- 'G': generalized eigenvalue problem $A*x = \lambda*B*x$

N

Integer.

Dimension of the eigenproblem.

WHICH

String of length 2.

Specify which of the Ritz values of OP to compute.

- 'LA' - compute the NEV largest (algebraic) eigenvalues.
- 'SA' - compute the NEV smallest (algebraic) eigenvalues.

- 'LM' - compute the NEV largest (in magnitude) eigenvalues.
- 'SM' - compute the NEV smallest (in magnitude) eigenvalues.
- 'BE' - compute NEV eigenvalues, half from each end of the spectrum. When NEV is odd, compute one more from the high end than from the low end. (see remark 1 below)

NEV

Integer.

Number of eigenvalues of OP to be computed. $0 < \text{NEV} < N$.

TOL

scalar.

Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if $\text{BOUNDS}(I) \leq \text{TOL} * \text{ABS}(\text{RITZ}(I))$. If $\text{TOL} \leq 0$, the machine precision is set.

RESID

Array of length N (INPUT/OUTPUT)

On INPUT: If $\text{INFO}=0$, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV

Integer.

Number of columns of the matrix V (less than or equal to N).

This will indicate how many Lanczos vectors are generated at each iteration. After the startup phase in which NEV Lanczos vectors are generated, the algorithm generates $\text{NCV}-\text{NEV}$ Lanczos vectors at each subsequent update iteration. Most of the cost in generating each Lanczos vector is in the matrix-vector product $\text{OP} * x$. (See remark 4 below).

V

N by NCV array.

The NCV columns of V contain the Lanczos basis vectors.

IPARAM

array of length 11. (INPUT/OUTPUT)

- $\text{IPARAM}(1) = \text{ISHIFT}$: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.
- $\text{ISHIFT} = 0$: the shifts are provided by the user via reverse communication. The NCV eigenvalues of the current tridiagonal matrix T are returned in the part of WORKL array corresponding to RITZ. See remark 6 below.
- $\text{ISHIFT} = 1$: exact shifts with respect to the reduced tridiagonal matrix T. This is equivalent to restarting the iteration with a starting vector that is a linear combination of Ritz vectors associated with the "wanted" Ritz values.
- $\text{IPARAM}(2) = \text{LEVEC}$ No longer referenced. See remark 2 below.
- $\text{IPARAM}(3) = \text{MXITER}$

On INPUT: maximum number of Arnoldi update iterations allowed.

On OUTPUT: actual number of Arnoldi update iterations taken.

- IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.
- IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.
- IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.
- IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4,5; See under Description of dsaupd for the five modes available.
- IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), dsaupd returns NP, the number of shifts the user is to provide. $0 < NP \leq NCV - NEV$. See Remark 6 below.
- IPARAM(9) = NUMOP,
IPARAM(10) = NUMOPB,
IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR

array of length 11. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Lanczos iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.
- IPNTR(2): pointer to the current result vector Y in WORKD.
- IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.
- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.
- IPNTR(5): pointer to the NCV by 2 tridiagonal matrix T in WORKL.
- IPNTR(6): pointer to the NCV RITZ values array in WORKL.
- IPNTR(7): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZ in WORKL.
- IPNTR(11): pointer to the NP shifts in WORKL. See Remark 6 below.

Note: IPNTR(8:10) is only referenced by dseupd . See Remark 2.

- IPNTR(8): pointer to the NCV RITZ values of the original system.
- IPNTR(9): pointer to the NCV corresponding error bounds.
- IPNTR(10): pointer to the NCV by NCV matrix of eigenvectors of the tridiagonal matrix T. Only referenced by dseupd if RVEC = 1. See Remarks

WORKD

work array of length 3*N. (REVERSE COMMUNICATION)

Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If the Ritz vectors are desired subroutine dseupd uses this output. See Data Distribution Note below.

WORKL

work array of length at least $NCV**2 + 8*NCV$. (OUTPUT/WORKSPACE)

Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below. add here the parameter description

INFO

Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

- 0: Normal exit.
- 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
- 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
- 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV must be greater than NEV and less than or equal to N.
- -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.
- -5: WHICH must be one of 'LM', 'SM', 'LA', 'SA' or 'BE'.
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work array WORKL is not sufficient.
- -8: Error return from trid. eigenvalue calculation; Informatinal error from LAPACK routine dsteqr .
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4,5.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatable.
- -12: IPARAM(1) must be equal to 0 or 1.
- -13: NEV and WHICH = 'BE' are incompatable.
- -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

Description

Reverse communication interface for the Implicitly Restarted Arnoldi Iteration. For symmetric problems this reduces to a variant of the Lanczos method. This method has been designed to compute approximations to a few eigenpairs of a linear operator OP that is real and symmetric with respect to a real positive semi-definite symmetric matrix B, i.e. $B^*OP = (OP^*)^*B$.

Another way to express this condition is $\langle x, OPy \rangle = \langle OPx, y \rangle$ where $\langle z, w \rangle = z^*Bw$.

In the standard eigenproblem B is the identity matrix. (A^T denotes transpose of A)

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dsauupd is usually called iteratively to solve one of the following problems:

- Mode 1: $A^T x = \lambda x$, A symmetric $\implies OP = A$ and $B = I$.
- Mode 2: $A^T x = \lambda M x$, A symmetric, M symmetric positive definite $\implies OP = \text{inv}[M]^T A$ and $B = M$. $\implies$ (If M can be factored see remark 3 below)
- Mode 3: $K^T x = \lambda M x$, K symmetric, M symmetric positive semi-definite $\implies OP = (\text{inv}[K - \sigma M])^T M$ and $B = M$. $\implies$ Shift-and-Invert mode
- Mode 4: $K^T x = \lambda K G x$, K symmetric positive semi-definite, KG symmetric indefinite $\implies OP = (\text{inv}[K - \sigma KG])^T K$ and $B = K$. $\implies$ Buckling mode
- Mode 5: $A^T x = \lambda M x$, A symmetric, M symmetric positive semi-definite $\implies OP = \text{inv}[A - \sigma M]^T [A + \sigma M]$ and $B = M$. $\implies$ Cayley transformed mode

NOTE: The action of $w \leftarrow \text{inv}[A - \sigma M]^T v$ or $w \leftarrow \text{inv}[M]^T v$ should be accomplished either by a direct method using a sparse matrix factorization and solving $[A - \sigma M]^T w = v$ or $M^T w = v$,

or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

Remarks

1. The converged Ritz values are always returned in ascending algebraic order. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3,4,5. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dseupd .

2. If the Ritz vectors corresponding to the converged Ritz values are needed, the user must call dseupd immediately following completion of dsauupd . This is new starting with version 2.1 of ARPACK.

3. If M can be factored into a Cholesky factorization $M = LL^T$ then Mode = 2 should not be selected. Instead one should use Mode = 1 with $OP = \text{inv}(L)^T A \text{inv}(L)$. Appropriate triangular linear systems should be solved with L and L^T rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving $L^T z = x$ where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requirement is that $NCV > NEV$. However, it is recommended that $NCV \geq 2 \cdot NEV$. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of $OP^T x$ operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically.

5. If IPARAM(7) = 2 then in the Reverse communication interface the user must do the following. When IDO = 1, $Y = OP^T X$ is to be computed. When IPARAM(7) = 2 $OP = \text{inv}(B)^T A$. After computing $A^T X$ the user must overwrite X with $A^T X$. Y is then the solution to the linear set of equations $B^T Y = A^T X$.

6. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) shifts in locations: 1 WORKL(IPNTR(11)) 2 WORKL(IPNTR(11)+1) . . . NP WORKL(IPNTR(11)+NP-1). The eigenvalues of the current tridiagonal matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1). They are in the order defined by WHICH. The associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

Example

```
// The following sets dimensions for this problem.

nx      = 10;

nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays

iparam   = zeros(11,1);
ipntr    = zeros(14,1);
_select  = zeros(ncv,1);
d        = zeros(nev+1,1);
resid    = zeros(nx,1);
v        = zeros(nx,ncv);
workd    = zeros(3*nx+1,1);
workl    = zeros(ncv*ncv+8*ncv,1);

// Build the symmetric test matrix

A          = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(6*ones(nx-1,1));

tol       = 0;
ido       = 0;

ishfts    = 1;
maxitr    = 300;
mode1     = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
    info_dsaupd = 0;
    iter = iter + 1;
    // Repeatedly call the routine DSAUPD and take actions indicated by parameter
    // either convergence is indicated or maxitr has been exceeded.

    [ido,resid,v,iparam,ipntr,workd,workl,info_dsaupd] = dsaupd(ido,bmat,nx,which

    if (ido==99) then
```

```

    // BE CAREFUL: DON'T CALL dseupd IF ido == 99 !!
    break;
end

if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DSAUPD again.
    continue
end

if (info_dsaupd < 0) then
    printf('\nError with dsaupd, info = %d\n',info_dsaupd);
    printf('Check the documentation of dsaupd\n\n');
else
    // Post-Process using DSEUPD.

    rvec      = 1;
    howmany = 'A';

    info_dseupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dseupd] = dseupd(rvec,howmany,bmat,nx,workl,iparam,info_dseupd);

    if (info_dseupd~=0) then
        printf('\nError with dseupd, info = %d\n', info_dseupd);
        printf('Check the documentation of dseupd.\n\n');
    end

    if (info_dseupd==1) then
        printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dseupd==3) then
        printf('\nNo shifts could be applied during implicit Arnoldi update, try\n\n');
    end
end
end

// Done with program dssimp.
printf('\nDSSIMP\n');
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam(8));
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);

```

See Also

dnaupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

Used Functions

Based on ARPACK routine dsaupd

Name

dsaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to the converged approximations to eigenvalues of $A*z = \lambda B*z$

```
[D, Z, RESID, V, IPARAM, IPNTR, WORKD, WORKL, INFO] = dseupd(RVEC, HOWMANY, SELECT, D, Z, SI  
NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, INFO)
```

Parameters

RVEC

Integer (INPUT)

Specifies whether Ritz vectors corresponding to the Ritz value approximations to the eigenproblem $A*z = \lambda B*z$ are computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute Ritz vectors.

HOWMNY

Character*1 (INPUT)

Specifies how many Ritz vectors are wanted and the form of Z the matrix of Ritz vectors. See remark 1 below.

- 'A': compute NEV Ritz vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT.

SELECT

Integer array of dimension NCV. (INPUT/WORKSPACE)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value $D(j)$, SELECT(j) must be set to 1.

If HOWMANY = 'A', SELECT is used as a workspace for reordering the Ritz values.

D

Double precision array of dimension NEV. (OUTPUT)

On exit, D contains the Ritz value approximations to the eigenvalues of $A*z = \lambda B*z$. The values are returned in ascending order.

If IPARAM(7) = 3,4,5 then D represents the Ritz values of OP computed by dsaupd transformed to those of the original eigensystem $A*z = \lambda B*z$.

If IPARAM(7) = 1,2 then the Ritz values of OP are the same as the those of $A*z = \lambda B*z$.

Z

Double precision N by NEV array.

If HOWMNY = 'A'. (OUTPUT) On exit, Z contains the B-orthonormal Ritz vectors of the eigensystem $A*z = \lambda B*z$ corresponding to the Ritz value approximations.

If RVEC = 0 then Z is not referenced.

NOTE: The array Z may be set equal to first NEV columns of the Arnoldi/Lanczos basis array V computed by DSAUPD.

SIGMA

Double precision (INPUT)

If IPARAM(7) = 3,4,5 represents the shift. Not referenced if IPARAM(7) = 1 or 2.

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, INFO must be passed directly to DSEUPD following the last call to DSAUPD .

These arguments MUST NOT BE MODIFIED between the the last call to DSAUPD and the call to DSEUPD.

Two of these parameters (WORKL, INFO) are also output parameters.

WORKL

Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:4*ncv) contains information obtained in dsaupd. They are not changed by dseupd.

WORKL(4*ncv+1:ncv*ncv+8*ncv) holds the untransformed Ritz values, the computed error estimates, and the associated eigenvector matrix of H.

Note: IPNTR(8:10) contains the pointer into WORKL for addresses of the above information computed by dseupd .

- IPNTR(8): pointer to the NCV RITZ values of the original system.
- IPNTR(9): pointer to the NCV corresponding error bounds.
- IPNTR(10): pointer to the NCV by NCV matrix of eigenvectors of the tridiagonal matrix T. Only referenced by dseupd if RVEC = 1 See Remarks.

INFO

Integer. (OUTPUT)

Error flag on output.

- 0: Normal exit.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV must be greater than NEV and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LA', 'SA' or 'BE'.
- -6: BMAT must be one of 'T' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from trid. eigenvalue calculation; Information error from LAPACK routine dsteqr .
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4,5.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: NEV and WHICH = 'BE' are incompatible.
- -14: DSAUPD did not find any eigenvalues to sufficient accuracy.
- -15: HOWMNY must be one of 'A' or 'S' if RVEC = 1
- -16: HOWMNY = 'S' not yet implemented
- -17: DSEUPD got a different count of the number of converged Ritz values than DSAUPD got. This indicates the user probably made an error in passing data from DSAUPD to DSEUPD or that the data was modified before entering DSEUPD.

Description

This subroutine returns the converged approximations to eigenvalues of $A*z = \lambda*B*z$ and (optionally):

1. the corresponding approximate eigenvectors,
2. an orthonormal (Lanczos) basis for the associated approximate invariant subspace,
3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal (Lanczos) basis is always computed. There is an additional storage cost of $n*nev$ if both are requested (in this case a separate array Z must be supplied).

These quantities are obtained from the Lanczos factorization computed by DSAUPD for the linear operator OP prescribed by the $MODE$ selection (see $IPARAM(7)$ in DSAUPD documentation.) DSAUPD must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Lanczos basis.

See documentation in the header of the subroutine DSAUPD for a definition of OP as well as other terms and the relation of computed Ritz values and vectors of OP with respect to the given problem $A*z = \lambda*B*z$.

The approximate eigenvalues of the original problem are returned in ascending algebraic order.

The user may elect to call this routine once for each desired Ritz vector and store it peripherally if desired. There is also the option of computing a selected set of these vectors with a single call.

Remarks

1. The converged Ritz values are always returned in increasing (algebraic) order. c 2. Currently only $HOWMNY = 'A'$ is implemented. It is included at this stage for the user who wants to incorporate it.

Example

```
// The following sets dimensions for this problem.

nx      = 10;

nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
```

```
resid    = zeros(nx,1);
v        = zeros(nx,ncv);
workd    = zeros(3*nx+1,1);
workl    = zeros(ncv*ncv+8*ncv,1);

// Build the symmetric test matrix

A        = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(6*ones(nx-1,1));

tol      = 0;
ido      = 0;

ishfts   = 1;
maxitr   = 300;
model    = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = model;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxitr)
    info_dsaupd = 0;
    iter = iter + 1;
    // Repeatedly call the routine DSAUPD and take actions indicated by parameter
    // either convergence is indicated or maxitr has been exceeded.

    [ido,resid,v,iparam,ipntr,workd,workl,info_dsaupd] = dsaupd(ido,bmat,nx,which

    if (ido==99) then
        // BE CAREFUL: DON'T CALL dseupd IF ido == 99 !!
        break;
    end

    if (ido==-1 | ido==1) then
        // Perform matrix vector multiplication
        workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
        // L O O P   B A C K to call DSAUPD again.
        continue
    end

    if (info_dsaupd < 0) then
        printf('\nError with dsaupd, info = %d\n',info_dsaupd);
        printf('Check the documentation of dsaupd\n\n');
    else
        // Post-Process using DSEUPD.

        rvec    = 1;
        howmany = 'A';

        info_dseupd = 0;
```

```
[d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dseupd] = dseupd(rvec,how,
                                                                    bmat,nx,
                                                                    iparam,i

if (info_dseupd~=0) then
    printf('\nError with dseupd, info = %d\n', info_dseupd);
    printf('Check the documentation of dseupd.\n\n');
end

if (info_dseupd==1) then
    printf('\nMaximum number of iterations reached.\n\n');
elseif (info_dseupd==3) then
    printf('\nNo shifts could be applied during implicit Arnoldi update, try
end
end
end

// Done with program dssimp.
printf('\nDSSIMP\n');
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

See Also

dsaupd, dneupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

Used Functions

Based on ARPACK routine dsaupd

Name

znaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute a few eigenpairs of a complex linear operator OP with respect to a semi-inner product defined by a hermitian positive semi-definite real matrix B .

`[ IDO, RESID, V, IPARAM, IPNTR, WORKD, WORKL, RWORKINFO ] = znaupd( ID0, BMAT, N, WHICH, NEV,`

Parameters

IDO

Integer. (INPUT/OUTPUT)

Reverse communication flag. IDO must be zero on the first call to znaupd. IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call znaupd with the result.

The operand is given in `WORKD(IPNTR(1))`, the result must be put in `WORKD(IPNTR(2))`.

- IDO = 0: first call to the reverse communication interface
- IDO = -1: compute $Y = OP * X$ where `IPNTR(1)` is the pointer into `WORKD` for X , `IPNTR(2)` is the pointer into `WORKD` for Y .

This is for the initialization phase to force the starting vector into the range of OP .

- IDO = 1: compute $Y = OP * X$ where `IPNTR(1)` is the pointer into `WORKD` for X , `IPNTR(2)` is the pointer into `WORKD` for Y .

In mode 3, the vector $B * X$ is already available in `WORKD(ipntr(3))`. It does not need to be recomputed in forming $OP * X$.

- IDO = 2: compute $Y = M * X$ where `IPNTR(1)` is the pointer into `WORKD` for X , `IPNTR(2)` is the pointer into `WORKD` for Y .
- IDO = 3: compute and return the shifts in the first `NP` locations of `WORKL`.
- IDO = 99: done

After the initialization phase, when the routine is used in the "shift-and-invert" mode, the vector $M * X$ is already available and does not need to be recomputed in forming $OP * X$.

BMAT

Character. (INPUT)

BMAT specifies the type of the matrix B that defines the semi-inner product for the operator OP .

- 'I': standard eigenvalue problem $A * x = \lambda * x$
- 'G': generalized eigenvalue problem $A * x = \lambda * M * x$

N

Integer. (INPUT)

Dimension of the eigenproblem.

WHICH

Characters. (INPUT)

- 'LM': want the NEV eigenvalues of largest magnitude.
- 'SM': want the NEV eigenvalues of smallest magnitude.

- 'LR': want the NEV eigenvalues of largest real part.
- 'SR': want the NEV eigenvalues of smallest real part.
- 'LI': want the NEV eigenvalues of largest imaginary part.
- 'SI': want the NEV eigenvalues of smallest imaginary part.

NEV

Integer. (INPUT)

Number of eigenvalues of OP to be computed. $0 < \text{NEV} < N-1$.

TOL

Double precision scalar. (INPUT)

Stopping criteria: the relative accuracy of the Ritz value is considered acceptable if $\text{BOUNDS(I)} \cdot \text{LE} \cdot \text{TOL} \cdot \text{ABS}(\text{RITZ(I)})$ where $\text{ABS}(\text{RITZ(I)})$ is the magnitude when RITZ(I) is complex. $\text{DEFAULT} = \text{dlamch}(\text{'EPS'})$ (machine precision as computed by the LAPACK auxiliary subroutine `dlamch`).

RESID

Complex*16 array of length N. (INPUT/OUTPUT)

On INPUT: If $\text{INFO} \cdot \text{EQ} \cdot 0$, a random initial residual vector is used. If $\text{INFO} \cdot \text{NE} \cdot 0$, RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV

Integer. (INPUT)

Number of columns of the matrix V. NCV must satisfy the two inequalities $1 \leq \text{NCV} - \text{NEV}$ and $\text{NCV} \leq N$.

This will indicate how many Arnoldi vectors are generated at each iteration. After the startup phase in which NEV Arnoldi vectors are generated, the algorithm generates approximately $\text{NCV} - \text{NEV}$ Arnoldi vectors at each subsequent update iteration. Most of the cost in generating each Arnoldi vector is in the matrix-vector operation $\text{OP} \cdot \mathbf{x}$. (See remark 4 below.)

V

Complex*16 array N by NCV. (OUTPUT)

Contains the final set of Arnoldi basis vectors.

IPARAM

Integer array of length 11. (INPUT/OUTPUT)

- $\text{IPARAM}(1) = \text{ISHIFT}$: method for selecting the implicit shifts. The shifts selected at each iteration are used to filter out the components of the unwanted eigenvector.
- $\text{ISHIFT} = 0$: the shifts are to be provided by the user via reverse communication. The NCV eigenvalues of the Hessenberg matrix H are returned in the part of WORKL array corresponding to RITZ.
- $\text{ISHIFT} = 1$: exact shifts with respect to the current Hessenberg matrix H. This is equivalent to restarting the iteration from the beginning after updating the starting vector with a linear combination of Ritz vectors associated with the "wanted" eigenvalues.
- $\text{ISHIFT} = 2$: other choice of internal shift to be defined.
- $\text{IPARAM}(2) = \text{No longer referenced}$

- IPARAM(3) = MXITER

On INPUT: maximum number of Arnoldi update iterations allowed.

On OUTPUT: actual number of Arnoldi update iterations taken.

- IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.
- IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.
- IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.
- IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3; See under Description of znaupd for the four modes available.
- IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), _naupd returns NP, the number of shifts the user is to provide. $0 < NP < NCV-NEV$.
- IPARAM(9) = NUMOP,
IPARAM(10) = NUMOPB,

IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR

Integer array of length 14. (OUTPUT)

Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Arnoldi iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.
- IPNTR(2): pointer to the current result vector Y in WORKD.
- IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.
- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.
- IPNTR(5): pointer to the NCV by NCV upper Hessenberg matrix H in WORKL.
- IPNTR(6): pointer to the ritz value array RITZ
- IPNTR(7): pointer to the (projected) ritz vector array Q
- IPNTR(8): pointer to the error BOUNDS array in WORKL.
- IPNTR(14): pointer to the NP shifts in WORKL. See Remark 5 below.

Note: IPNTR(9:13) is only referenced by zneupd. See Remark 2 below.

- IPNTR(9): pointer to the NCV RITZ values of the original system.
- IPNTR(10): Not Used
- IPNTR(11): pointer to the NCV corresponding error bounds.
- IPNTR(12): pointer to the NCV by NCV upper triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by zneupd if RVEC = 1 See Remark 2 below.

WORKD

Complex*16 work array of length $3*N$. (REVERSE COMMUNICATION)

Distributed array to be used in the basic Arnoldi iteration for reverse communication.

The user should not use WORKD as temporary workspace during the iteration !!!!!!!!!!!

See Data Distribution Note below.

WORKL

Complex*16 work array of length $3*NCV**2 + 5*NCV$. (OUTPUT/WORKSPACE)

Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below.

RWORK

Double precision work array of length NCV (WORKSPACE) Private (replicated) array on each PE or array allocated on the front end.

INFO

Integer. (INPUT/OUTPUT)

If $INFO == 0$, a randomly initial residual vector is used.

If $INFO \neq 0$, RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

- 0: Normal exit.
- 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
- 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
- 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: $NCV - NEV \geq 1$ and less than or equal to N.
- -4: The maximum number of Arnoldi update iteration must be greater than zero.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work array is not sufficient.
- -8: Error return from LAPACK eigenvalue calculation;
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: IPARAM(1) must be equal to 0 or 1.
- -9999: Could not build an Arnoldi factorization. User input error highly likely. Please check actual array dimensions and layout. IPARAM(5) returns the size of the current Arnoldi factorization.

Description

Reverse communication interface for the Implicitly Restarted Arnoldi iteration. This is intended to be used to find a few eigenpairs of a complex linear operator OP with respect to a semi-inner product defined by a hermitian positive semi-definite real matrix B . B may be the identity matrix.

NOTE: if both OP and B are real, then `dsaupd` or `dnaupd` should be used.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors. `znaupd` is usually called iteratively to solve one of the following problems:

- Mode 1: $A*x = \lambda*x$.
====> $OP = A$ and $B = I$.
- Mode 2: $A*x = \lambda*M*x$, M hermitian positive definite
====> $OP = \text{inv}[M]*A$ and $B = M$.
====> (If M can be factored see remark 3 below)
- Mode 3: $A*x = \lambda*M*x$, M hermitian semi-definite
====> $OP = \text{inv}[A - \sigma*M]*M$ and $B = M$.
====> shift-and-invert mode If $OP*x = \sigma*x$, then $\lambda = \sigma + 1/\mu$.

NOTE: The action of $w \leftarrow \text{inv}[A - \sigma*M]*v$ or $w \leftarrow \text{inv}[M]*v$ should be accomplished either by a direct method using a sparse matrix factorization and solving

$[A - \sigma*M]*w = v$ or $M*w = v$,

or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

Remarks

1. The computed Ritz values are approximate eigenvalues of OP . The selection of `WHICH` should be made with this in mind when using Mode = 3. When operating in Mode = 3 setting `WHICH = 'LM'` will compute the NEV eigenvalues of the original problem that are closest to the shift `SIGMA`. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine `zneupd`.
2. If a basis for the invariant subspace corresponding to the converged Ritz values is needed, the user must call `zneupd` immediately following completion of `znaupd`. This is new starting with release 2 of ARPACK.
3. If M can be factored into a Cholesky factorization $M = LL^T$ then Mode = 2 should not be selected. Instead one should use Mode = 1 with $OP = \text{inv}(L)*A*\text{inv}(L^T)$. Appropriate triangular linear systems should be solved with L and L^T rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving $L^T z = x$ where x is a Ritz vector of OP .
4. At present there is no a-priori analysis to guide the selection of `NCV` relative to `NEV`. The only formal requirement is that `NCV > NEV + 1`. However, it is recommended that `NCV .ge. 2*NEV`. If many problems of the same type are to be solved, one should experiment with increasing `NCV` while keeping `NEV` fixed for a given test problem. This will usually decrease the required number of $OP*x$ operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically. See Chapter 8 of Reference 2 for further information.

5. When $\text{IPARAM}(1) = 0$, and $\text{IDO} = 3$, the user needs to provide the $\text{NP} = \text{IPARAM}(8)$ complex shifts in locations

$\text{WORKL}(\text{IPNTR}(14)), \text{WORKL}(\text{IPNTR}(14)+1), \dots, \text{WORKL}(\text{IPNTR}(14)+\text{NP})$.

Eigenvalues of the current upper Hessenberg matrix are located in $\text{WORKL}(\text{IPNTR}(6))$ through $\text{WORKL}(\text{IPNTR}(6)+\text{NCV}-1)$. They are ordered according to the order defined by **WHICH**. The associated Ritz estimates are located in

$\text{WORKL}(\text{IPNTR}(8)), \text{WORKL}(\text{IPNTR}(8)+1), \dots, \text{WORKL}(\text{IPNTR}(8)+\text{NCV}-1)$.

Example

```
// The following sets dimensions for this problem.

nx      = 10;

nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1) + 0*%i;
v       = zeros(nx,ncv) + 0*%i;
workd   = zeros(3*nx+1,1) + 0*%i;
workev  = zeros(3*ncv,1);
rwork   = zeros(ncv,1);
workl   = zeros(3*ncv*ncv+5*ncv,1) + 0*%i;

// Build the complex test matrix
A       = diag(10*ones(nx,1)+%i*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol     = 0;
ido     = 0;

ishfts  = 1;
maxitr  = 300;
model   = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = model;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)
```

```

iter = 0;
while(iter<maxiter)
  info_znaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine ZNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_znaupd] = znaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL zneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call ZNAUPD again.
    continue
  end

  if (info_znaupd < 0) then
    printf('\nError with znaupd, info = %d\n',info_znaupd);
    printf('Check the documentation of znaupd\n\n');
  else
    // Post-Process using ZNEUPD.

    rvec      = 1;
    howmany = 'A';

    info_zneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_zneupd] = zneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i

    if (info_zneupd~=0) then
      printf('\nError with zneupd, info = %d\n', info_zneupd);
      printf('Check the documentation of zneupd.\n\n');
    end

    if (info_zneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_zneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program znsimp.
printf('\nZNSIMP\n');
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);

```

```
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam  
printf('The number of OP*x is %d\n', iparam(9));  
printf('The convergence criterion is %d\n', tol);
```

See Also

dnaupd, dneupd, zneupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

Used Functions

Based on ARPACK routine znaupd

Name

zneupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to the converged approximations to eigenvalues of $A*z = \lambda*B*z$

```
[D, Z, RESID, IPARAM, IPNTR, WORKD, WORKL, RWORK, INFO] = zneupd(RVEC, HOWMANY, SELECT, D,  
BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM,
```

Parameters

RVEC

Integer (INPUT)

Specifies whether a basis for the invariant subspace corresponding to the converged Ritz value approximations for the eigenproblem $A*z = \lambda*B*z$ is computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute Ritz vectors or Schur vectors. See Remarks below.

HOWMANY

Character (INPUT)

Specifies the form of the basis for the invariant subspace corresponding to the converged Ritz values that is to be computed.

- 'A': Compute NEV Ritz vectors;
- 'P': Compute NEV Schur vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT.

SELECT

Integer array of dimension NCV. (INPUT)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value $D(j)$, SELECT(j) must be set to 1.

If HOWMANY = 'A' or 'P', SELECT need not be initialized but it is used as internal workspace.

D

Complex*16 array of dimension NEV+1. (OUTPUT)

On exit, D contains the Ritz approximations to the eigenvalues λ for $A*z = \lambda*B*z$.

Z

Complex*16 N by NEV array (OUTPUT)

On exit,

If RVEC = 1 and HOWMANY = 'A', then the columns of Z represents approximate eigenvectors (Ritz vectors) corresponding to the NCONV=IPARAM(5) Ritz values for eigensystem $A*z = \lambda*B*z$.

If RVEC = 0 or HOWMANY = 'P', then Z is NOT REFERENCED.

NOTE: If if RVEC = 1 and a Schur basis is not required, the array Z may be set equal to first NEV+1 columns of the Arnoldi basis array V computed by ZNAUPD. In this case the Arnoldi basis will be destroyed and overwritten with the eigenvector basis.

SIGMA

Complex*16 (INPUT)

If IPARAM(7) = 3 then SIGMA represents the shift.

Not referenced if $\text{IPARAM}(7) = 1$ or 2 .

WORKEv

Complex*16 work array of dimension $2 \times \text{NCV}$. (WORKSPACE)

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, RWORK, INFO must be passed directly to ZNEUPD following the last call to ZNAUPD.

These arguments MUST NOT BE MODIFIED between the the last call to ZNAUPD and the call to ZNEUPD.

Three of these parameters (V, WORKL and INFO) are also output parameters.

V

Complex*16 N by NCV array. (INPUT/OUTPUT)

Upon INPUT: the NCV columns of V contain the Arnoldi basis vectors for OP as constructed by ZNAUPD.

Upon OUTPUT: If RVEC = 1 the first $\text{NCONV} = \text{IPARAM}(5)$ columns contain approximate Schur vectors that span the desired invariant subspace.

NOTE: If the array Z has been set equal to first $\text{NEV} + 1$ columns of the array V and RVEC=1 and HOWMANY= 'A', then the Arnoldi basis held by V has been overwritten by the desired Ritz vectors. If a separate array Z has been passed then the first $\text{NCONV} = \text{IPARAM}(5)$ columns of V will contain approximate Schur vectors that span the desired invariant subspace.

WORKL

Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:ncv*ncv+2*ncv) contains information obtained in znaupd. They are not changed by zneupd.

WORKL(ncv*ncv+2*ncv+1:3*ncv*ncv+4*ncv) holds the untransformed Ritz values, the untransformed error estimates of the Ritz values, the upper triangular matrix for H, and the associated matrix representation of the invariant subspace for H.

Note: IPNTR(9:13) contains the pointer into WORKL for addresses of the above information computed by zneupd.

- IPNTR(9): pointer to the NCV RITZ values of the original system.
- IPNTR(10): Not used
- IPNTR(11): pointer to the NCV corresponding error estimates.
- IPNTR(12): pointer to the NCV by NCV upper triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by zneupd if RVEC = 1 See Remark 2 below.

INFO

Integer. (OUTPUT)

Error flag on output.

- 0: Normal exit.
- 1: The Schur form computed by LAPACK routine csheqr could not be reordered by LAPACK routine ztrsen. Re-enter subroutine zneupd with $\text{IPARAM}(5) = \text{NCV}$ and increase the size of the array D to have dimension at least dimension NCV and allocate at least NCV columns for Z.

NOTE: Not necessary if Z and V share the same space. Please notify the authors if this error occurs.

- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV ≥ 1 and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from LAPACK eigenvalue calculation. This should never happened.
- -9: Error return from calculation of eigenvectors. Informational error from LAPACK routine `ztrevc`
- -10: IPARAM(7) must be 1,2,3
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: HOWMANY = 'S' not yet implemented
- -13: HOWMANY must be one of 'A' or 'P' if RVEC = .true.
- -14: ZNAUPD did not find any eigenvalues to sufficient accuracy.
- -15: ZNEUPD got a different count of the number of converged Ritz values than ZNAUPD got. This indicates the user probably made an error in passing data from ZNAUPD to ZNEUPD or that the data was modified before entering ZNEUPD

Description

This subroutine returns the converged approximations to eigenvalues of $A*z = \lambda*B*z$ and (optionally):

1. The corresponding approximate eigenvectors;
2. An orthonormal basis for the associated approximate invariant subspace;
3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal basis is always computed.

There is an additional storage cost of $n*nev$ if both are requested (in this case a separate array Z must be supplied).

The approximate eigenvalues and eigenvectors of $A*z = \lambda*B*z$ are derived from approximate eigenvalues and eigenvectors of the linear operator OP prescribed by the MODE selection in the call to ZNAUPD.

ZNAUPD must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Schur basis.

The definition of OP as well as other terms and the relation of computed Ritz values and vectors of OP with respect to the given problem $A*z = \lambda*B*z$ may be found in the header of ZNAUPD. For a brief description, see definitions of IPARAM(7), MODE and WHICH in the documentation of ZNAUPD.

Remarks

1. Currently only HOWMNY = 'A' and 'P' are implemented.
2. Schur vectors are an orthogonal representation for the basis of Ritz vectors. Thus, their numerical properties are often superior.

If RVEC = 1 then the relationship

$$A * V(:,1:IPARAM(5)) = V(:,1:IPARAM(5)) * T,$$

and

$$\text{transpose}(V(:,1:IPARAM(5))) * V(:,1:IPARAM(5)) = I$$

are approximately satisfied. Here T is the leading submatrix of order IPARAM(5) of the upper triangular matrix stored workl(ipntr(12)).

Example

```
// The following sets dimensions for this problem.
nx      = 10;
nev     = 3;
ncv     = 6;
bmat    = 'I';
which   = 'LM';

maxiter = 10;

// Local Arrays
iparam   = zeros(11,1);
ipntr    = zeros(14,1);
_select  = zeros(ncv,1);
d        = zeros(nev+1,1);
resid    = zeros(nx,1) + 0*i;
v        = zeros(nx,ncv) + 0*i;
workd    = zeros(3*nx+1,1) + 0*i;
workev   = zeros(3*ncv,1);
rwork    = zeros(ncv,1);
workl    = zeros(3*ncv*ncv+5*ncv,1) + 0*i;

// Build the complex test matrix
A        = diag(10*ones(nx,1)+%i*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol      = 0;
ido      = 0;

ishfts   = 1;
maxitr   = 300;
mode1    = 1;

iparam(1) = ishfts;
```



```

iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxitr)
  info_znaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine ZNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_znaupd] = znaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL zneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call ZNAUPD again.
    continue
  end

  if (info_znaupd < 0) then
    printf('\nError with znaupd, info = %d\n',info_znaupd);
    printf('Check the documentation of znaupd\n\n');
  else
    // Post-Process using ZNEUPD.

    rvec      = 1;
    howmany = 'A';

    info_zneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_zneupd] = zneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i

    if (info_zneupd~=0) then
      printf('\nError with zneupd, info = %d\n', info_zneupd);
      printf('Check the documentation of zneupd.\n\n');
    end

    if (info_zneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_zneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program znsimp.
printf('\nZNSIMP\n');

```

```
printf('=====\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam);
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

See Also

znaupd, dnaupd, dneupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.
2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.
3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

Used Functions

Based on ARPACK routine zneupd

Part XXXVI. Compatibility Functions

Name

asciimat — string matrix ascii conversions

```
a=asciimat(txt)
txt=asciimat(a)
```

Parameters

txt
character string or matrix of strings.

a
vector or matrix of integer ASCII codes

Description

This function convert Scilab string to ASCII code or a matrix of ASCII code to Scilab string. Output is a matrix having same number of lines than input, what is not the case with `ascii`.

See Also

`ascii`

Authors

V.C.

Name

firstnonsingleton — Finds first dimension which is not 1

```
[dim]=firstnonsingleton(A[,opt])
```

Parameters

dim

first dimension of A which is not 1

A

a variable of any Scilab data type

opt

a character string giving the type of output we want

"num"

returned value is a numerical value

"str"

returned value is a character string if possible ("r" instead of 1 and "c" instead of 2)

Description

This function is used by `mfile2sci` to emulate Matlab behavior under Scilab, particularly for functions which treat the values along the first non-singleton dimension of A in Matlab while in Scilab they treat all values of A.

Examples

```
A = [1 2 3;4 5 6];  
// Scilab max  
M = max(A)  
// Matlab max emulation  
M = max(A,firstnonsingleton(A))
```

Authors

V.C.

Nom

`makecell` — Creates a cell array.

```
s = makecell(dims,a1,a2,...an)
```

Parameters

`dims`

a vector with positive integer elements, the cell array dimension

`a1,a2,...,an`

Sequence of Scilab variables, n must be equal to `prod(dims)`

`s`

resulting cell array

Description

`s = makecell(dims,a1,a2,...an)` creates a cell array of dimensions given by `dims` with the given input arguments. The `a1` are stored along the last dimension first.

Examples

```
makecell([2,3],1,2,3,'x','y','z')
```

See Also

`cell`

Authors

Farid Belhacen, INRIA

Name

mstr2sci — character string matrix to character matrix conversion

```
a=mstr2sci(txt)
```

Parameters

txt
character string or string matrix

a
character vector or matrix

Description

This function converts a Scilab character string to a vector of characters. Result is the Scilab equivalent for a Matlab string.

Caution: `mstr2sci` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

`mtlb_0` — Matlab non-conjugate transposition emulation function

Description

Matlab and Scilab non-conjugate transposition behave differently in some particular cases:

- With character strings operands: The `.` operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_0(A)` is used by `mfile2sci` to replace `A.'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_0` calls:

- If `A` is not a character string matrix `mtlb_0(A)` may be replaced by `A.'`

Caution: `mtlb_0` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings`

Authors

V.C.

Name

`mtlb_a` — Matlab addition emulation function

Description

Matlab and Scilab addition behave differently in some particular cases:

- With character string operands: The `+` operator is used into Scilab to concatenate character strings, while Matlab realizes the sum of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the addition is the other operand. In Matlab if one of the operand is an empty matrix the result of the addition should be an error or an empty matrix.

The function `mtlb_a(A,B)` is used by `mfile2sci` to replace `A+B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_a` calls:

- If `A` and `B` are character strings, `mtlb_a(A,B)` may be replaced by `asciimat(A)+asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_a(A,B)` may be replaced by `A+B`, else `mtlb_a(A,B)` may be replaced by `[]`.
- If `mtlb_mode==%T`, then `mtlb_a(A,B)` may be replaced by `A+B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_a` has not to be used for hand coded functions.

See Also

`mtlb_mode`

Authors

V.C.

Name

`mtlb_all` — Matlab all emulation function

Description

Matlab `all` and Scilab `and` behave differently in some particular cases:

- When used with one input (`all(A)`), Matlab `all` treats the values along the first non-singleton dimension of `A` as vectors while Scilab `and` treats all `A` values.
- When used with two inputs (`all(A, dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of `A` while Scilab returns an error message in this case.

The function `R = mtlb_all(A[, dim])` is used by `mfile2sci` to replace `all(A[, dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_all` calls:

- If `A` is a scalar or a vector `R = mtlb_all(A)` may be replaced by `R = and(A)`
- If `A` is a matrix `R = mtlb_all(A)` may be replaced by `R = and(A, 1)`
- If `A` is an hypermatrix `R = mtlb_all(A)` may be replaced by `R = and(A, firstnonsingleton(A))` or by `R = and(A, user_defined_value)` if the first non-singleton dimensions of `A` is known.
- If `dim` is less equal to the number of dimensions of `A` `R = mtlb_all(A, dim)` may be replaced by `R = and(A, dim)`
- If `dim` is greater than then number of dimensions of `A` `R = mtlb_all(A, dim)` may be replaced by `R = A<>0` if `A` is not an empty matrix or by `R = A` if `A` is an empty matrix.

Caution: `mtlb_all` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_any` — Matlab any emulation function

Description

Matlab `any` and Scilab `or` behave differently in some particular cases:

- When used with one input (`any(A)`), Matlab `any` treats the values along the first non-singleton dimension of `A` as vectors while Scilab `or` treats all `A` values.
- When used with two inputs (`any(A, dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of `A` while Scilab returns an error message in this case.

The function `R = mtlb_any(A[, dim])` is used by `mfile2sci` to replace `any(A[, dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_any` calls:

- If `A` is a scalar or a vector `R = mtlb_any(A)` may be replaced by `R = or(A)`
- If `A` is a matrix `R = mtlb_any(A)` may be replaced by `R = or(A, 1)`
- If `A` is an hypermatrix `R = mtlb_any(A)` may be replaced by `R = or(A, firstnonsingleton(A))` or by `R = or(A, user_defined_value)` if the first non-singleton dimensions of `A` is known.
- If `dim` is less equal to the number of dimensions of `A` `R = mtlb_any(A, dim)` may be replaced by `R = or(A, dim)`
- If `dim` is greater than then number of dimensions of `A` `R = mtlb_any(A, dim)` may be replaced by `R = A<>0` if `A` is not an empty matrix or by `R = A` if `A` is an empty matrix.

Caution: `mtlb_any` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_axis` — Matlab axis emulation function

```
mtlb_axis(X)
mtlb_axis(st)
mtlb_axis(axeshandle, ...)
[mode,visibility,direction]=mtlb_axis('state')
```

Parameters

`X`
a vector of reals ([xmin xmax ymin ymax] or [xmin xmax ymin ymax zmin zmax])

`st`
a string ('auto', 'manual', 'tight', 'ij', 'xy', 'equal', 'square', 'vis3d', 'off', 'on')

`axeshandle`
handle of the current axes entity

Description

Matlab `axis` have not a Scilab equivalent function.

The function `mtlb_axis(...)` is used by `mfile2sci` to replace `axis(...)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time (`axis fill`, `axis image` and `axis normal` are not implemented). If you want to have a more efficient code it is possible to replace `mtlb_axis` call by `get(axeshandle, prop)` call (`prop` is a axes property, see `axis_properties`)

Caution: `mtlb_axis` has not to be used for hand coded functions.

Authors

F.B.

Name

`mtlb_beta` — Matlab beta emulation function

Description

Matlab and Scilab `beta` behave differently in some particular cases:

- With inputs having different sizes: Matlab `beta` input parameters must have the same size unless one of them is a scalar. In Scilab `beta` input parameters must have the same size even if one of them is a scalar.

The function `mtlb_beta(A,B)` is used by `mfile2sci` to replace `beta(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_beta` calls:

- If `A` is a scalar but not `B` `Y = mtlb_beta(A,B)` may be replaced by `C=B;C(:)=A;Y = mtlb_beta(C,B);`
- If `B` is a scalar but not `A` `Y = mtlb_beta(A,B)` may be replaced by `C=A;C(:)=B;Y = mtlb_beta(A,C);`

Caution: `mtlb_beta` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_box — Matlab box emulation function

Description

There is no Scilab equivalent function for Matlab `box` but it can be easily replaced.

The function `mtlb_box([axes_handle[,val]])` is used by `mfile2sci` to replace `box([axes_handle[,val]]l)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_box` calls:

- When called without input parameters, `mtlb_box()` may be replaced by `a=gca();if a.box=="on" then a.box="off";else a.box="on";end;`
- If `axes_handle` is a character string, `mtlb_box(axes_handle)` may be replaced by `a=gca();a.box=convstr(axes_handle,"l");`
- If `axes_handle` is a graphic handle `mtlb_box(axes_handle)` may be replaced by `if axes_handle.box=="on" then axes_handle.box="off";else axes_handle.box="on";end;`
- If `axes_handle` is a graphic handle and `val` is a character string `mtlb_box(axes_handle,val)` may be replaced by `axes_handle.box=convstr(val,"l");`

Caution: `mtlb_box` has not to be used for hand coded functions.

See Also

`axes_properties`

Authors

V.C.

Name

`mtlb_close` — Matlab close emulation function

Description

Scilab equivalent for Matlab `close` is different according to the current figure type (uicontrol or graphic one).

- When current figure is a uicontrol one: Scilab equivalent for Matlab `close` is `close`
- When current figure is a graphic one: Scilab equivalent for Matlab `close` is `xdel` or `delete`
- In Scilab we do not get an error status.

The function `mtlb_close([h])` is used by `mfile2sci` to replace `close([h])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_close` calls:

- If `h` is a uicontrol figure `mtlb_close(h)` may be replaced by `close(h)`
- If `h` is a graphic figure `mtlb_close(h)` may be replaced by `delete(h)`

Caution: `mtlb_close` has not to be used for hand coded functions.

See Also

`xdel` , `delete` , `winsid` , `close`

Authors

V.C.

Name

mtlb_colordef — Matlab colordef emulation function

Description

There is no Scilab equivalent function for Matlab `colordef` but there are equivalent instructions.

The function `h = mtlb_colordef(color_option)` or `h = mtlb_colordef(fig,color_option)` is used by `mfile2sci` to replace `colordef(color_option)` or `colordef(fig,color_option)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_colordef` calls:

- When called with one input parameter, if `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -1;`
- When called with one input parameter, if `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -2;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig.background = -1;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig.background = -2;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid()+1));fig.background = -1;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid()+1));fig.background = -2;`
- When called with one output parameter `h`, just add `h = fig;` after equivalent instructions.

Caution: `mtlb_colordef` has not to be used for hand coded functions.

See Also

`figure_properties`

Authors

V.C.

Name

`mtlb_conv` — Matlab conv emulation function

Description

Matlab `conv` and Scilab `convol` behave differently in some particular cases:

- With column vector inputs: if at least input is a column vector Matlab `conv` returns a column vector but Scilab `convol` always returns a row vector.

The function `mtlb_conv(u, v)` is used by `mfile2sci` to replace `conv(u, v)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_conv` calls:

- If `u` and `v` are row vectors, `mtlb_conv(u, v)` may be replaced by `convol(u, v)`
- If `u` or `v` is a column vector, `mtlb_conv(u, v)` may be replaced by `convol(u, v) . '`
- If `u` and `v` are column vectors, `mtlb_conv(u, v)` may be replaced by `convol(u, v) . '`

Scilab `convol` sometimes returns values that may be rounded using `clean` to have a closer result from Matlab one.

Caution: `mtlb_conv` has not to be used for hand coded functions.

See Also

`clean`

Authors

V.C.

Name

`mtlb_cumprod` — Matlab `cumprod` emulation function

Description

Matlab and Scilab `cumprod` behave differently in some particular cases:

- When used with one input (`cumprod(A)`), Matlab `cumprod` treats the values along the first non-singleton dimension of `A` as vectors while Scilab `cumprod` treats all `A` values.
- When used with two inputs (`cumprod(A, dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of `A` while Scilab returns an error message in this case.

The function `R = mtlb_cumprod(A[,dim])` is used by `mfile2sci` to replace `cumprod(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumprod` calls:

- If `dim` is less equal to the number of dimensions of `A` `R = mtlb_cumprod(A, dim)` may be replaced by `R = cumprod(A, dim)`
- If `dim` is greater than then number of dimensions of `A` `R = mtlb_cumprod(A, dim)` may be replaced by `R = A`.

Caution: `mtlb_cumprod` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

mtlb_cumsum — Matlab cumsum emulation function

Description

Matlab and Scilab cumsum behave differently in some particular cases:

- When used with one input (`cumsum(A)`), Matlab `cumsum` treats the values along the first non-singleton dimension of `A` as vectors while Scilab `cumsum` treats all `A` values.
- When used with two inputs (`cumsum(A, dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of `A` while Scilab returns an error message in this case.

The function `R = mtlb_cumsum(A[,dim])` is used by `mfile2sci` to replace `cumsum(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumsum` calls:

- If `dim` is less equal to the number of dimensions of `A` `R = mtlb_cumsum(A, dim)` may be replaced by `R = cumsum(A, dim)`
- If `dim` is greater than then number of dimensions of `A` `R = mtlb_cumsum(A, dim)` may be replaced by `R = A`.

Caution: `mtlb_cumsum` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

mtlb_dec2hex — Matlab dec2hex emulation function

Description

Matlab and Scilab `dec2hex` behave differently in some particular cases:

- With empty matrix: Matlab `dec2hex` returns "" but in Scilab you get [].
- With complex inputs: Matlab `dec2hex` automatically removes complex part of inputs but not Scilab one.
- Matlab `dec2hex` always returns a row vector but in Scilab `dec2hex` returns a value which have the same size as the input.
- Matlab `dec2hex` can have two inputs but not Scilab one.

The function `mtlb_dec2hex(D[,N])` is used by `mfile2sci` to replace `dec2hex(D[,N])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_dec2hex` calls:

- If `D` is not an empty matrix, `mtlb_dec2hex(D)` may be replaced by `matrix(dec2hex(real(D)), -1, 1)` if `D` is complex and by `matrix(dec2hex(D), -1, 1)` else.

Caution: `mtlb_dec2hex` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_delete` — Matlab delete emulation function

Description

The function `mtlb_delete(A)` is used by `mfile2sci` to replace `delete(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_delete` calls:

- If `A` is a character string `mtlb_delete(A)` may be replaced by `mdelete(A)`
- If `A` is a graphic handle `mtlb_delete(A)` may be replaced by `delete(A)`

Caution: `mtlb_delete` has not to be used for hand coded functions.

See Also

`mdelete`

Authors

V.C.

Name

mtlb_diag — Matlab diag emulation function

Description

Matlab and Scilab `diag` behave differently in some particular cases:

- With character string matrices: Scilab `diag` function considers each character string as an object while Matlab considers each character individually.

The function `y = mtlb_diag(x[,dim])` is used by `mfile2sci` to replace `y = diag(x[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diag` calls:

- If `x` is not a character string matrix `y = mtlb_diag(x[,dim])` may be replaced by `y = diag(x[,dim])`

Caution: `mtlb_diag` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

`mtlb_diff` — Matlab diff emulation function

Description

Matlab and Scilab `diff` behave differently in some particular cases:

- With two input parameters: Scilab `diff` considers all values of first input as a vector what Matlab does not.
- With three input parameters: If dimension of first input along dimension given by third parameter reaches 1 before `n` iterations have been made, Matlab switches to next non-singleton dimension what Scilab does not.

The function `mtlb_diff(A[,n[,dim]])` is used by `mfile2sci` to replace `diff(A[,n[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diff` calls:

- With two inputs, if `A` is a scalar or a vector `mtlb_diff(A,n)` may be replaced by `diff(A,n)`
- With three inputs, if size of `A` along dimension given by `dim` can not reach 1 `mtlb_diff(A,n,dim)` may be replaced by `diff(A,n,dim)`

Caution: `mtlb_diff` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_dir` — Matlab dir emulation function

Description

Matlab and Scilab `dir` behave differently:

- When result is stored in a variable: Matlab `dir` returns a structure but Scilab `dir` returns a 'dir' tlist, so data can not be extracted in the same way.

The function `mtlb_dir([path])` is used by `mfile2sci` to replace `dir([path])` when output is stored in a variable. There is no replacement possibility for it, else (when `mtlb_dir` is replaced by `dir`) data can not be extracted like in Matlab. For example, Scilab equivalent for Matlab `L=dir;file_name=L(1).name;` is `L=dir();file_name=L.name(1);`.

Caution: `mtlb_dir` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_double` — Matlab double emulation function

Description

Matlab and Scilab `double` behave differently in some particular cases:

- With character string input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of ASCII codes.
- With boolean input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of double values.

The function `mtlb_double(A)` is used by `mfile2sci` to replace `double(A)` when it was not possible to know what were the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_double` calls:

- If `A` is a character string, `mtlb_double(A)` may be replaced by `asciimat(A)`
- If `A` is a boolean matrix, `mtlb_double(A)` may be replaced by `bool2s(A)`
- If `A` is a double matrix, `mtlb_double(A)` may be replaced by `A`

Caution: `mtlb_double` has not to be used for hand coded functions.

See Also

`asciimat` , `bool2s`

Authors

V.C.

Name

`mtlb_e` — Matlab extraction emulation function

Description

Matlab and Scilab extraction behave differently in some particular cases:

- When extracting data from a matrix with a vector as index: Matlab returns a row vector and Scilab returns a column vector.
- When extracting data from a character string matrix: due to the fact that character string matrices in Matlab can be addressed as any other matrix (each character can be addressed), extraction in such a type of matrix does not differ from other. But in Scilab it can't be done so and `part` function has to be used.

The function `mtlb_e(B, k)` is used by `mfile2sci` to replace `A=B(k)` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_e` calls:

- If `B` is a vector `A=mtlb_e(B, k)` may be replaced by `A=B(k)`
- If `B` is a matrix `A=mtlb_e(B, k)` may be replaced by `A=B(k) . '`
- If `B` is a character string matrix and `k` is a scalar or a vector `A=mtlb_e(B, k)` may be replaced by `A=part(B, k)`

Caution: `mtlb_e` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings` , `part`

Authors

V.C.

Name

`mtlb_echo` — Matlab echo emulation function

Description

There is no equivalent for Matlab `echo` in Scilab but some cases can be replaced by calls to Scilab `mode`:

- `echo`: is equivalent to Scilab `mode(abs(mode()-1))` for scripts and non-compiled functions
- `echo on`: is equivalent to Scilab `mode(1)` for scripts and non-compiled functions
- `echo off`: is equivalent to Scilab `mode(0)`

The function `mtlb_echo(arg1[,arg2])` is used by `mfile2sci` to replace `echo arg1 [arg2]` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_echo` calls:

- If `arg1` is equal to "on" `mtlb_echo(arg1)` may be replaced by `mode(1)`
- If `arg1` is equal to "off" `mtlb_echo(arg1)` may be replaced by `mode(0)`

Caution: `mtlb_echo` has not to be used for hand coded functions.

See Also

`mode`

Authors

V.C.

Name

mtlb_eval — Matlab eval emulation function

Description

Scilab equivalent for Matlab `eval` is different according to its inputs and outputs

The function `mtlb_eval(str1, str2)` is used by `mfile2sci` to replace `eval` because it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eval` calls:

- When called with one input and no output, `mtlb_eval(str1)` may be replaced by `evstr(str1)` if `str1` is a valid Scilab expression or by `execstr(str1)` if `str1` is a valid Scilab instruction.
- When called with one input and one output, `val=mtlb_eval(str1)` may be replaced by `val=evstr(str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and no output, `mtlb_eval(str1, str2)` may be replaced by: `if execstr(str1, "errcatch") <> 0 then execstr(str2); end` if `str1` and `str2` are valid Scilab instructions.
- When called with more than one output and one input, `[val1, val2, ...]=mtlb_eval(str1)` may be replaced by `execstr("[val1, val2, ...]" + str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and more than one output, `[val1, val2, ...]=mtlb_eval(str1, str2)` may be replaced by:

```
if execstr("[val1, val2, ...]" + str1, "errcatch") <> 0 then
    execstr("[val1, val2, ...]" + str2);
end
```

if `str1` and `str2` are valid Scilab instructions.

Caution: `mtlb_eval` has not to be used for hand coded functions.

See Also

`evstr`, `execstr`

Authors

V.C.

Name

`mtlb_exist` — Matlab exist emulation function

Description

There is no Scilab equivalent for Matlab `exist` except when input is a variable. Scilab `mtlb_exist` is a partial emulation of of this function.

The function `r = mtlb_exist(nam[,tp])` is used by `mfile2sci` to replace `exist(nam[,tp])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_exist` calls:

- When called with one input and if `nam` is a variable name, `mtlb_exist(nam)` may be replaced by `exists(nam)`

Caution: `mtlb_exist` has not to be used for hand coded functions.

See Also

`exists`

Authors

V.C.

Name

mtlb_eye — Matlab eye emulation function

Description

Matlab and Scilab `eye` behave differently in some particular cases:

- With a scalar input: Matlab `eye` returns a $n \times n$ matrix while Scilab returns a 1.

The function `mtlb_eye(A)` is used by `mfile2sci` to replace `eye(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eye` calls:

- If `A` is a scalar `mtlb_eye(A)` may be replaced by `eye(A,A)`
- If `A` is not a scalar `mtlb_eye(A)` may be replaced by `eye(A)`

Caution: `mtlb_eye` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_false` — Matlab false emulation function

Description

There is no Scilab equivalent for Matlab `false`. However, Scilab `zeros` returns a result interpreted in an equivalent way for Scilab.

Matlab `false` and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `false` returns a $n \times n$ matrix of zeros while Scilab `zeros` returns a 0.

The function `mtlb_false(A)` is used by `mfile2sci` to replace `false(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_false` calls:

- If `A` is a scalar `mtlb_false(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_false(A)` may be replaced by `zeros(A)`

Caution: `mtlb_false` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fft` — Matlab fft emulation function

Description

Matlab and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `fft` is `fft(..., -1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(..., -1, 2, 1)`

The function `mtlb_fft(X[,n,[job]])` is used by `mfile2sci` to replace `fft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fft` calls:

- If `X` is a scalar or a vector `mtlb_fft(X, -1)` may be replaced by `fft(X, -1)`
- If `X` is a matrix `mtlb_fft(X, -1)` may be replaced by `fft(X, -1, 2, 1)`

Caution: `mtlb_fft` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fftshift` — Matlab `fftshift` emulation function

Description

Matlab and Scilab `fftshift` behave differently in some particular cases:

- With character string input: due to the fact that character strings are not considered in the same way in Matlab and Scilab, results can be different for this kind of input.
- With two inputs: Matlab `fftshift` can work even if `dim` parameter is greater than number of dimensions of first input.

The function `mtlb_fftshift(A[,dim])` is used by `mfile2sci` to replace `fftshift(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fftshift` calls:

- If `A` is not a character string matrix, `mtlb_fftshift(A)` may be replaced by `fftshift(A)`
- If `A` is not a character string matrix and `dim` is not greater than `size(size(a), "*")`, `mtlb_fftshift(A, dim)` may be replaced by `fftshift(A, dim)`

Caution: `mtlb_fftshift` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_find` — Matlab find emulation function

Description

Matlab and Scilab `find` behave differently in some particular cases:

- With column vectors and matrices as input: Matlab `find` returns column vectors while Scilab returns row vectors.
- When called with three outputs: Matlab `find` can have three outputs but Scilab not.

The function `[i[,j[,v]]] = mtlb_find(B)` is used by `mfile2sci` to replace `[i[,j[,v]]] = find(B)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_find` calls:

- When called with one output, if `B` is a row vector `i = mtlb_find(B)` may be replaced by `i = find(B)`
- When called with one output, if `B` is not a row vector `i = mtlb_find(B)` may be replaced by `i = matrix(find(B), -1, 1)`
- When called with two outputs, if `B` is a row vector `[i, j] = mtlb_find(B)` may be replaced by `[i, j] = find(B)`
- When called with two outputs, if `B` is not a row vector `[i, j] = mtlb_find(B)` may be replaced by `[i, j] = find(B); i = matrix(i, -1, 1); j = matrix(j, -1, 1);`

Caution: `mtlb_find` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_findstr` — Matlab `findstr` emulation function

Description

There is no Scilab equivalent for Matlab `findstr`.

The function `mtlb_findstr(A,B)` is used by `mfile2sci` to replace `findstr(A,B)` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_findstr` calls:

- If `A` is always longer than `B` (i.e. `findstr` can be replaced by `strfind` in Matlab, `mtlb_findstr(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_findstr` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fliplr` — Matlab `fliplr` emulation function

Description

Matlab and Scilab `fliplr` behave differently in some particular cases:

- With character string matrices: due to the fact that Scilab and Matlab do not consider character string matrices in the same way, result can be different for input of this type.

The function `mtlb_fliplr(A)` is used by `mfile2sci` to replace `fliplr(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fliplr` calls:

- If `A` is not a character string matrix, `mtlb_fliplr(A)` may be replaced by `A(:, $:-1:1)`

Caution: `mtlb_fliplr` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fopen` — Matlab fopen emulation function

Description

Matlab `fopen` and Scilab `mopen` behave differently in some particular cases:

- Scilab function returns no usable error message
- Scilab file identified does not exist in case of error but Matlab one is set to `-1`.
- Matlab function can work with inputs which do not exist in Scilab such as permission options...

The function `mtlb_fopen(filename,permission)` is used by `mfile2sci` to replace `mopen(filename,permission)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fopen` calls:

- If error message is not used and no error can occurs, `mtlb_fopen(filename,permission)` may be replaced by `mopen(filename,permission,0)`

Caution: `mtlb_fopen` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_format` — Matlab format emulation function

Description

Matlab and Scilab `format` behave differently in some particular cases:

- Some Matlab formats do not exist in Scilab
- Calling sequence for `format` is different in Scilab and Matlab

The function `mtlb_format(type, prec)` is used by `mfile2sci` to replace `format([type prec])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_format` calls:

- If `type=""` and `prec=""` `mtlb_format("", "")` may be replaced by `format("v", 6)`
- If `type="+"` and `prec=""` `mtlb_format("+", "")` may be replaced by `format(6)`
- If `type="long"` and `prec=""` `mtlb_format("long", "")` may be replaced by `format(16)`
- If `type="long"` and `prec="e"` `mtlb_format("long", "e")` may be replaced by `format("e"16)`
- If `type="long"` and `prec="g"` `mtlb_format("long", "g")` may be replaced by `format("e"16)`
- If `type="short"` and `prec=""` `mtlb_format("short", "")` may be replaced by `format(6)`
- If `type="short"` and `prec="e"` `mtlb_format("short", "e")` may be replaced by `format("e"6)`
- If `type="short"` and `prec="g"` `mtlb_format("short", "g")` may be replaced by `format("e"6)`

Caution: `mtlb_format` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fprintf` — Matlab `fprintf` emulation function

Description

There is no Scilab exact equivalent for Matlab `fprintf`. Scilab `mfprintf` and Matlab `fprintf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fprintf(varargin)` is used by `mfile2sci` to replace `fprintf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fprintf` by calls to `mfprintf`.

Caution: `mtlb_fprintf` has not to be used for hand coded functions.

See Also

`mfprintf`

Authors

V.C.

Name

`mtlb_fread` — Matlab fread emulation function

Description

There is no Scilab equivalent for Matlab `fread`. Scilab `mget` and Matlab `fread` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fread(varargin)` is used by `mfile2sci` to replace `fread`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fread` by calls to `mget`.

Caution: `mtlb_fread` has not to be used for hand coded functions.

See Also

`mget`

Authors

V.C.

Name

`mtlb_fscanf` — Matlab `fscanf` emulation function

Description

There is no Scilab exact equivalent for Matlab `fscanf`. Scilab `mfscanf` and Matlab `fscanf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fscanf(varargin)` is used by `mfile2sci` to replace `fscanf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fscanf` by calls to `mfscanf`.

Caution: `mtlb_fscanf` has not to be used for hand coded functions.

See Also

`mfscanf`

Authors

V.C.

Name

`mtlb_full` — Matlab full emulation function

Description

Matlab and Scilab `full` behave differently in some particular cases:

- With character strings input: Matlab `full` can be used with character string input while Scilab function cannot.
- With boolean input: Matlab `full` can be used with boolean input while Scilab function cannot.

The function `mtlb_full(A)` is used by `mfile2sci` to replace `full(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_full` calls:

- If `A` is a double matrix `mtlb_full(A)` may be replaced by `full(A)`
- If `A` is a boolean matrix `mtlb_full(A)` may be replaced by `full(bool2s(A))`

Caution: `mtlb_full` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_fwrite` — Matlab `fwrite` emulation function

Description

There is no Scilab equivalent for Matlab `fwrite`. Scilab `mput` and Matlab `fwrite` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fwrite(varargin)` is used by `mfile2sci` to replace `fwrite`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fwrite` by calls to `mput`.

Caution: `mtlb_fwrite` has not to be used for hand coded functions.

See Also

`mput`

Authors

V.C.

Name

`mtlb_grid` — Matlab grid emulation function

Description

There is no Scilab equivalent function for Matlab `grid` but there are equivalent instructions.

The function `mtlb_grid(flag_or_handle[,flag])` is used by `mfile2sci` to replace `grid(flag_or_handle[,flag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_grid` calls:

- With one input, if `flag` is equal to "on" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[1 1])`
- With one input, if `flag` is equal to "off" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[-1 -1])`
- With two inputs, if `flag` is equal to "on" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[1 1]`
- With two inputs, if `arg2` is equal to "off" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[-1 -1]`

Caution: `mtlb_grid` has not to be used for hand coded functions.

See Also

`axes_properties`

Authors

V.C.

Name

`mtlb_hold` — Matlab hold emulation function

Description

There is no Scilab equivalent function for Matlab `hold` but there are equivalent instructions.

The function `mtlb_hold(flag)` is used by `mfile2sci` to replace `hold flag` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_hold` calls:

- If `flag` is equal to "on" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","off")`
- If `flag` is equal to "off" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","on")`

Caution: `mtlb_hold` has not to be used for hand coded functions.

See Also

`axes_properties`

Authors

V.C.

Name

`mtlb_i` — Matlab insertion emulation function

Description

Matlab and Scilab insertion behave differently in some particular cases:

- When inserting a matrix in a variable: Matlab automatically adjusts output variable to fit with matrix to insert but not Scilab. For example, with `A=1`, `A([1,2,3,4])=[1,2;3,4]` returns an error in Scilab while in Matlab we get `A=[1,2,3,4]`. If values miss comparing to indexes, Matlab fills output value with 0.
- When inserting data into a character string matrix: due to the fact that character string matrices in Matlab can be addressed as any other matrix (each character can be addressed), insertion in such a type of matrix does not differ from other. But in Scilab it can't be done so...`mtlb_is` is an alternative.

The function `A=mtlb_i(A, k, B)` is used by `mfile2sci` to replace `A(k)=B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_i` calls:

- If `A` is not a vector `A=mtlb_i(A, k, B)` may be replaced by `A(k)=B`
- If `A` and `B` are both row or column vectors `A=mtlb_i(A, k, B)` may be replaced by `A(k)=B`

Caution: `mtlb_i` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings` , `mtlb_is`

Authors

V.C.

Name

`mtlb_ifft` — Matlab ifft emulation function

Description

Matlab `ifft` and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `ifft(A)` is `fft(A,1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(A,1,2,1)`

The function `mtlb_ifft(X[,n,[job]])` is used by `mfile2sci` to replace `ifft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ifft` calls:

- If `X` is a scalar or a vector `mtlb_ifft(X,1)` may be replaced by `fft(X,1)`
- If `X` is a matrix `mtlb_ifft(X,1)` may be replaced by `fft(X,1,2,1)`

Caution: `mtlb_ifft` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_imp` — Matlab colon emulation function

Description

Matlab and Scilab colon behave differently in some particular cases:

- With empty matrices: The `:` operator must be used with scalars in Scilab and gives an error message when used with empty matrices while Matlab returns `[]` in these cases.

The function `mtlb_imp(A,B[,C])` is used by `mfile2sci` to replace `A:B[:C]` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_imp` calls:

- If A, B and C are not empty matrices `mtlb_imp(A,B[,C])` may be replaced by `A:B[:C]`

Caution: `mtlb_imp` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_int16` — Matlab int16 emulation function

Description

Matlab and Scilab `int16` behave differently in some particular cases:

- With complex input: Matlab `int16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int16` returns 32767 and Scilab returns -32768.
- With `%nan`: Matlab `int16` returns 0 and Scilab returns -32768.
- With `-%nan`: Matlab `int16` returns 0 and Scilab returns -32768.

The function `mtlb_int16(A)` is used by `mfile2sci` to replace `int16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int16` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int16(A)` may be replaced by `int16(A)`

Caution: `mtlb_int16` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_int32` — Matlab `int32` emulation function

Description

Matlab and Scilab `int32` behave differently in some particular cases:

- With complex input: Matlab `int32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int32` returns 2147483647 and Scilab returns -2147483648.
- With `%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.
- With `-%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.

The function `mtlb_int32(A)` is used by `mfile2sci` to replace `int32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int32` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int32(A)` may be replaced by `int32(A)`

Caution: `mtlb_int32` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_int8` — Matlab `int8` emulation function

Description

Matlab and Scilab `int8` behave differently in some particular cases:

- With complex input: Matlab `int8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int8` returns 127 and Scilab returns 0.
- With `-%inf`: Matlab `int8` returns -128 and Scilab returns 0.

The function `mtlb_int8(A)` is used by `mfile2sci` to replace `int8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int8` calls:

- If `A` does not contain `%inf` or `-%inf` values `mtlb_int8(A)` may be replaced by `int8(A)`

Caution: `mtlb_int8` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_is` — Matlab string insertion emulation function

Description

Matlab and Scilab insertion behave differently for strings due to the fact that they do not consider character strings in the same way.

The function `str = mtlb_is(sto,sfrom,i,j)` is used by `mfile2sci` to replace insertion operations for character strings. This function will determine the correct semantic at run time. There is no replacement possibility for it.

Caution: `mtlb_is` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings` , `mtlb_i`

Authors

V.C.

Name

`mtlb_isa` — Matlab isa emulation function

Description

There is no Scilab equivalent function for Matlab `isa` but some equivalent expressions can be used when the object "class" exists in Scilab.

The function `mtlb_isa(OBJ, class)` is used by `mfile2sci` to replace `isa(OBJ, class)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isa` calls:

- If `class` is equal to "logical", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==4`
- If `class` is equal to "char", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==10`
- If `class` is equal to "numeric", `mtlb_isa(OBJ, class)` may be replaced by `or(type(OBJ)==[1, 5, 8])`
- If `class` is equal to "intX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ, class)` may be replaced by `typeof(OBJ)=="intX"`
- If `class` is equal to "uintX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ, class)` may be replaced by `typeof(OBJ)=="uintX"`
- If `class` is equal to "single", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "double", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "cell", `mtlb_isa(OBJ, class)` may be replaced by `typeof(OBJ)=="ce"`
- If `class` is equal to "struct", `mtlb_isa(OBJ, class)` may be replaced by `typeof(OBJ)=="st"`
- If `class` is equal to "function_handle", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==13`
- If `class` is equal to "sparse", `mtlb_isa(OBJ, class)` may be replaced by `type(OBJ)==5`
- If `class` is equal to "lti", `mtlb_isa(OBJ, class)` may be replaced by `typeof(OBJ)=="state-space"`

Caution: `mtlb_isa` has not to be used for hand coded functions.

See Also

`type` , `typeof`

Authors

V.C.

Name

`mtlb_isfield` — Matlab `isfield` emulation function

Description

There is no Scilab equivalent function for Matlab `isfield(st,f)` and equivalent expressions behave differently in some particular cases:

- If `st` is not a structure: Scilab equivalent returns an error message but Matlab returns `0`.

The function `mtlb_isfield(st,f)` is used by `mfile2sci` to replace `isfield(st,f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isfield` calls:

- If `st` is a structure `tf = mtlb_isfield(st,f)` may be replaced by `allf=getfield(1,st);tf=or(allf(3:$)==f);`
- If `st` is not a structure `tf = mtlb_isfield(st,f)` may be replaced by `tf=%F;`

Caution: `mtlb_isfield` has not to be used for hand coded functions.

See Also

`getfield`

Authors

V.C.

Name

`mtlb_isletter` — Matlab isletter emulation function

Description

There is no Scilab equivalent function for Matlab `isletter` and equivalent instructions are quite ugly, so `mfile2sci` uses `mtlb_isletter(A)` to replace `isletter(A)`. If you want to have a more efficient code it is possible to replace `mtlb_isletter` calls:

- If `A` is not a character string `tf = mtlb_isletter(A)` may be replaced by `tf = zeros(A)`
- If `A` is a character string `tf = mtlb_isletter(A)` may be replaced by `tf = (asciimat(A)>=65&asciimat(A)<=90)|(asciimat(A)>=97&asciimat(A)<=122)`

Caution: `mtlb_isletter` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_isspace` — Matlab isspace emulation function

Description

There is no Scilab function equivalent for Matlab `isspace` but its behavior can be reproduced.

The function `mtlb_isspace(A)` is used by `mfile2sci` to replace `isspace(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isspace` calls:

- If `A` is a character string matrix `mtlb_isspace(A)` may be replaced by `asciimat(A)==32`
- If `A` is not a character string matrix `mtlb_isspace(A)` may be replaced by `zeros(A)`

Caution: `mtlb_isspace` has not to be used for hand coded functions.

See Also

`asciimat`

Authors

V.C.

Name

`mtlb_l` — Matlab left division emulation function

Description

Matlab and Scilab left division behave differently in some particular cases:

- With character string operands: The `\` operator can not be used into Scilab with character strings, while in Matlab it can. And in this case, result is transposed in a very strange way...

The function `mtlb_l(A,B)` is used by `mfile2sci` to replace `A\B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_l` calls:

- If both `A` and `B` are not character strings `mtlb_l(A,B)` may be replaced by `A\B`.

Caution: `mtlb_l` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_legendre — Matlab legendre emulation function

```
P = mtlb_legendre(n,X)
P = mtlb_legendre(n,X[,normflag])
```

Parameters

X
a scalar, vector, matrix or hypermatrix with elements in [-1,1]

n
a positive scalar integer

normflag
a string ('sch' or 'norm')

Description

Matlab and Scilab `legendre` behave differently in some particular cases:

- Scilab `legendre(m,n,X)` evaluates the `legendre` function of degree `n` and order `n` for the `X` elements. Matlab `legendre(n,X)` evaluates the Legendre functions of degree `n` and order `m=0,1,...,n`. (emulated by `mtlb_legendre`) for the `X` elements.
- The option `normflag= 'sch'` doesn't exist for Scilab `legendre` (emulated)
- If `X` is a hypermatrix then Scilab `legendre(n,X)` doesn't work (emulated)

The function `mtlb_legendre(n,X[,normflag])` is used by `mfile2sci` to replace `legendre(n,X[,normflag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_legendre` call:

- If `X` is a scalar, a vector or a matrix `mtlb_legendre(n,X[, 'norm'])` may be replaced by `legendre(n,0:n,X[, 'norm'])`

Caution: `mtlb_legendre` has not to be used for hand coded functions.

Authors

F.B.

Name

mtlb_linspace — Matlab linspace emulation function

Description

Matlab and Scilab `linspace` behave differently in some particular cases:

- With character string inputs: Matlab `linspace(A,B[,n])` returns a character string vector if `A` and/or `B` are character strings, but Scilab function does not work with such inputs.

The function `mtlb_linspace(A,B[,n])` is used by `mfile2sci` to replace `linspace(A,B[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_linspace` calls:

- If `A` and `B` are not character strings `mtlb_linspace(A,B[,n])` may be replaced by `linspace(A,B[,n])`
- If `A` or `B` are character strings `mtlb_linspace(A,B[,n])` may be replaced by `ascii(linspace(ascii(A),ascii(B)[,n]))`

Caution: `mtlb_linspace` has not to be used for hand coded functions.

See Also

`ascii`

Authors

V.C.

Name

`mtlb_logic` — Matlab logical operators emulation function

Description

Matlab and Scilab logical operator behave differently in some particular cases:

- With complex operands: The `<`, `<=`, `>` and `>=` operators can not be used into Scilab with complex operands, while in Matlab they can. And in this case, only real part of complex operands is compared.
- With empty matrices: If both operands of `<`, `<=`, `>` and `>=` operators are empty matrices, Scilab returns an error message, while Matlab returns an empty matrix. For operators `==` and `~=`, if at least one operand is an empty matrix, Matlab returns `[]` while Scilab returns a boolean value `%T` or `%F`.

The function `mtlb_logic(A, symbol, B)` (with "symbol" a character string containing the operator symbol) is used by `mfile2sci` to replace `A symbol B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logic` calls:

- If both `A` and `B` are not complex values nor empty matrices `mtlb_logic(A, symbol, B)` may be replaced by `A symbol B`.

Caution: `mtlb_logic` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_logical` — Matlab logical emulation function

Description

There is no Scilab equivalent function for Matlab `logical` but its behavior can be easily reproduced.

The function `mtlb_logical(A)` is used by `mfile2sci` to replace `logical(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logical` calls:

- If `A` is a boolean matrix `mtlb_logical(A)` may be replaced by `A`
- If `A` is not an empty matrix `mtlb_logical(A)` may be replaced by `A<>[]`
- If `A` is an empty matrix `mtlb_logical(A)` may be replaced by `[]`

Caution: `mtlb_logical` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_lower` — Matlab lower emulation function

Description

Matlab `lower(A)` and Scilab `convstr(A, "l")` behave differently in some particular cases:

- If `A` is not a character string matrix: Matlab `lower` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_lower(A)` is used by `mfile2sci` to replace `lower(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_lower` calls:

- If `A` is a character string matrix `mtlb_lower(A)` may be replaced by `convstr(A, "l")`
- If `A` is not a character string matrix `mtlb_lower(A)` may be replaced by `A`

Caution: `mtlb_lower` has not to be used for hand coded functions.

See Also

`convstr`

Authors

V.C.

Name

`mtlb_max` — Matlab max emulation function

Description

Matlab and Scilab `max` behave differently in some particular cases:

- With complex values: Matlab `max` can be used with complex values but not Scilab function.
- When called with one input: Matlab `max` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns `[]`.
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_max(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = max(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_max` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_max(A)` may be replaced by `max(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_max(A)` may be replaced by `max(A, "r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_max(A, B)` may be replaced by `max(A, B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_max(A, [], dim)` may be replaced by `max(A, dim)`

Caution: `mtlb_max` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_min` — Matlab min emulation function

Description

Matlab and Scilab `min` behave differently in some particular cases:

- With complex values: Matlab `min` can be used with complex values but not Scilab function.
- When called with one input: Matlab `min` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns `[]`.
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_min(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = min(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_min` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_min(A)` may be replaced by `min(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_min(A)` may be replaced by `min(A, "r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_min(A, B)` may be replaced by `min(A, B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_min(A, [], dim)` may be replaced by `min(A, dim)`

Caution: `mtlb_min` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_more` — Matlab more emulation function

Description

Matlab `more` and Scilab `lines` behave differently in some particular cases:

- With character strings as input: Matlab `more` can take "on" and "off" as input but not Scilab `lines` but there are equivalents (0 and 60).

The function `mtlb_more(in)` is used by `mfile2sci` to replace `more(in)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_more` calls:

- If `in` is equal to "on" `mtlb_more(in)` may be replaced by `lines(60)`
- If `in` is equal to "off" `mtlb_more(in)` may be replaced by `lines(0)`
- If `in` is a double value `mtlb_more(in)` may be replaced by `lines(in)`

Caution: `mtlb_more` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_num2str` — Matlab `num2str` emulation function

Description

Matlab `num2str` and Scilab equivalents (`string`, `msprintf`) behave differently in some particular cases:

- With two input parameters, the second giving precision: There is no Scilab equivalent function, but `msprintf` can be used to emulate.
- With two input parameters, the second giving format: Scilab equivalent for Matlab `num2string(a, format)` is `msprintf(format, a)`.

The function `mtlb_num2str(x, f)` is used by `mfile2sci` to replace `num2str(x, f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_num2str` calls:

- If `f` is a character string `mtlb_num2str(x, f)` may be replaced by `msprintf(f, x)`

Caution: `mtlb_num2str` has not to be used for hand coded functions.

See Also

`string` , `msprintf`

Authors

V.C.

Name

`mtlb_ones` — Matlab ones emulation function

Description

Matlab and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `ones` returns a $n \times n$ matrix while Scilab returns a 1.

The function `mtlb_ones(A)` is used by `mfile2sci` to replace `ones(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ones` calls:

- If `A` is a scalar `mtlb_ones(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_ones(A)` may be replaced by `ones(A)`

Caution: `mtlb_ones` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_plot` — Matlab plot emulation function

Description

Matlab `plot` and Scilab `plot2d` behave differently.

The function `mtlb_plot(varargin)` is used by `mfile2sci` to replace `plot(varargin)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_plot` calls when there is no output value, however in this case, you have to set colors manually:

- With one input, if `Y` is real, `mtlb_plot(Y)` may be replaced by `plot2d(Y)`
- With one input, if `Y` is complex, `mtlb_plot(Y)` may be replaced by `plot2d(real(Y), imag(Y))`
- With two inputs `X` and `Y`, if `Y` is not a character string, `mtlb_plot(X, Y)` may be replaced by `plot2d(X, Y)`

Caution: `mtlb_plot` has not to be used for hand coded functions.

See Also

`plot2d`

Authors

V.C.

Name

`mtlb_prod` — Matlab prod emulation function

Description

Matlab and Scilab `prod` behave differently in some particular cases:

- When called with one input: Matlab `prod` threats the values along the first non-singleton dimension of input while Scilab `prod` threats all values of input.
- When called with two inputs: Matlab `prod` can be used with second input giving a non-existing dimension of first input while Scilab `prod` returns an error message.

The function `mtlb_prod(A[, dim])` is used by `mfile2sci` to replace `prod(A[, dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_prod` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_prod(A)` may be replaced by `prod(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_prod(A)` may be replaced by `prod(A, 1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_prod(A)` may be replaced by `prod(A, firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_prod(A, dim)` may be replaced by `prod(A, dim)`

Caution: `mtlb_prod` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_rand` — Matlab rand emulation function

Description

Matlab and Scilab `rand` behave differently in some particular cases:

- With a scalar input: Matlab `rand` returns a $n \times n$ matrix while Scilab returns a scalar.

The function `mtlb_rand(A)` is used by `mfile2sci` to replace `rand(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rand` calls:

- If `A` is a scalar `mtlb_rand(A)` may be replaced by `rand(A,A)`
- If `A` is not a scalar `mtlb_rand(A)` may be replaced by `rand(A)`

Caution: `mtlb_rand` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_randn — Matlab randn emulation function

Description

Matlab `rand` and Scilab `rand(..., "normal")` behave differently in some particular cases:

- With a scalar input: Matlab `randn` returns a $n \times n$ matrix while Scilab `rand(..., "normal")` returns a scalar.

The function `mtlb_randn(A)` is used by `mfile2sci` to replace `randn(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_randn` calls:

- If `A` is a scalar `mtlb_randn(A)` may be replaced by `rand(A, A, "normal")`
- If `A` is not a scalar `mtlb_randn(A)` may be replaced by `rand(A, "normal")`

Caution: `mtlb_randn` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_rcond` — Matlab `rcond` emulation function

Description

Matlab and Scilab `rcond` behave differently in some particular cases:

- With empty matrix: Matlab `rcond` returns `Inf` and Scilab `rcond` returns `[]`

The function `mtlb_rcond(A)` is used by `mfile2sci` to replace `rcond(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rcond` calls:

- If `A` is not an empty matrix, `mtlb_rcond(A)` may be replaced by `rcond(A)`

Caution: `mtlb_rcond` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_realmax` — Matlab `realmax` emulation function

Description

Scilab equivalent for Matlab `realmax` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmax` or `realmax('double')` is `number_properties("huge")`.
- There is no Scilab equivalent for Matlab `realmax('single')`

The function `mtlb_realmax(prec)` is used by `mfile2sci` to replace `realmax(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmax` calls:

- If `prec` is equal to `"double"` `mtlb_realmax(prec)` may be replaced by `number_properties("huge")`

Caution: `mtlb_realmax` has not to be used for hand coded functions.

See Also

`number_properties`

Authors

V.C.

Name

`mtlb_realmin` — Matlab realmin emulation function

Description

Scilab equivalent for Matlab `realmin` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmin` or `realmin('double')` is `number_properties("tiny")`.
- There is no Scilab equivalent for Matlab `realmin('single')`

The function `mtlb_realmin(prec)` is used by `mfile2sci` to replace `realmin(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmin` calls:

- If `prec` is equal to `"double"` `mtlb_realmin(prec)` may be replaced by `number_properties("tiny")`

Caution: `mtlb_realmin` has not to be used for hand coded functions.

See Also

`number_properties`

Authors

V.C.

Name

mtlb_repmat — Matlab repmat emulation function

Description

There is no Scilab equivalent function for Matlab `repmat` but there are equivalent instructions.

The function `mtlb_repmat(M,m[,n])` is used by `mfile2sci` to replace `repmat(M,m[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_repmat` calls:

- If `A` is of Double type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*M` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*M`
- If `A` is of Boolean type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*bool2s(M)` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*bool2s(M)`
- If `A` is of String type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m,m).*asciimat(M))` and `mtlb_repmat(M,m,n)` may be replaced by `asciimat(ones(m,n).*asciimat(M))`
- If `A` is of Double type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*M`
- If `A` is of Boolean type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*bool2s(M)`
- If `A` is of String type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m(1),m(2),...).*asciimat(M))`

Caution: `mtlb_repmat` has not to be used for hand coded functions.

See Also

`ones`, `kron`

Authors

V.C.

Name

`mtlb_s` — Matlab subtraction emulation function

Description

Matlab and Scilab subtraction behave differently in some particular cases:

- With character string operands: The `-` operator can not be used into Scilab with character strings, while Matlab realizes the subtraction of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the subtraction is the other operand. In Matlab if one of the operand is an empty matrix the result of the subtraction should be an error or an empty matrix.

The function `mtlb_s(A,B)` is used by `mfile2sci` to replace `A-B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_s` calls:

- If `A` and `B` are character strings, `mtlb_s(A,B)` may be replaced by `asciimat(A)-asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_s(A,B)` may be replaced by `A-B`, else `mtlb_s(A,B)` may be replaced by `[]`.
- If `mtlb_mode()==%T`, then `mtlb_s(A,B)` may be replaced by `A-B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_s` has not to be used for hand coded functions.

See Also

`mtlb_mode`

Authors

V.C.

Name

`mtlb_setstr` — Matlab `setstr` emulation function

Description

Matlab `setstr` and Scilab `ascii` behave differently in some particular cases:

- With character string input: Matlab `setstr` returns a character string while Scilab `ascii` returns ASCII codes.
- With double matrix input: Matlab `setstr` returns a character matrix having the same size as input while Scilab `ascii` returns a single character string

The function `mtlb_setstr(A)` is used by `mfile2sci` to replace `setstr(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_setstr` calls:

- If `A` is a character string or a character string matrix `mtlb_setstr(A)` may be replaced by `A`
- If `A` is a double row vector `mtlb_setstr(A)` may be replaced by `ascii(A)`

Caution: `mtlb_setstr` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_size` — Matlab size emulation function

Description

Matlab and Scilab `size` behave differently in some particular cases:

- With two inputs: Matlab `size` can be used with second parameter giving a not-existing dimension of first parameter (returns 1 in this case) but not Scilab one.
- With more than one output: if number of output is lesser than number of dimensions, last output is the product of all remaining dimensions in Matlab but not in Scilab. If number of output is greater than number of dimensions, outputs corresponding to a not-existing dimension are set to 1 in Matlab but Scilab gives an error in this case.

The function `[d1, [d2, ...]] = mtlb_size(X[, dim])` is used by `mfile2sci` to replace `[d1, [d2, ...]] = size(X[, dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_size` calls:

- With two inputs: if `dim` gives an existing dimension of `X` `mtlb_size(X, dim)` may be replaced by `size(X, dim)`
- With more than one outputs: if the number of outputs is equal to the number of dimensions of `X` `[d1, [d2, ...]] = mtlb_size(X)` may be replaced by `[d1, [d2, ...]] = size(X)`

Caution: `mtlb_size` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_sort` — Matlab sort emulation function

```
P = mtlb_sort(X)
P = mtlb_sort(X, dim[, mode])
```

Parameters

`X`
a scalar, vector, matrix of reals, booleans or a string

`dim`
a positive scalar integer

`mode`
a string ("ascend" or "descend")

Description

Matlab `sort` and Scilab `gsort` behave differently in some particular cases:

- For a vector `X` the Matlab `sort(X, 'g', 'i')` function call is equivalent to the Scilab `gsort(X)` function call.
- The value 1 (resp. 2) of the Matlab `dim` is equivalent to the Scilab "r" flag (resp. "c").
- The Matlab "ascend" (resp. "descend") mode is equivalent to the Scilab "i" (resp. "d") flag.

The function `mtlb_sort(X[, dim[, mode]])` is used by `mfile2sci` to replace `sort(X[, dim[, mode]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sort` call by `gsort` call.

Caution: `mtlb_sort` has not to be used for hand coded functions.

Authors

F.B.

Name

`mtlb_strcmp` — Matlab strcmp emulation function

Description

There is no Scilab function equivalent for Matlab `strcmp`, there is equivalent instructions.

The function `mtlb_strcmp(A, B)` is used by `mfile2sci` to replace `strcmp(A, B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmp` calls:

- If `A` and `B` are character strings `mtlb_strcmp(A, B)` may be replaced by `A==B`
- If `A` and/or `B` is not a character string `mtlb_strcmp(A, B)` may be replaced by `0`

Caution: `mtlb_strcmp` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_strcmpi` — Matlab `strcmpi` emulation function

Description

There is no Scilab function equivalent for Matlab `strcmpi`, there is equivalent instructions.

The function `mtlb_strcmpi(A,B)` is used by `mfile2sci` to replace `strcmpi(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmpi` calls:

- If `A` and `B` are character strings `mtlb_strcmpi(A,B)` may be replaced by `convstr(A)==convstr(B)`
- If `A` and/or `B` is not a character string `mtlb_strcmpi(A,B)` may be replaced by `0`

Caution: `mtlb_strcmpi` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_strfind` — Matlab `strfind` emulation function

Description

Matlab `strfind` and Scilab `strindex` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strfind` can be used with not character strings inputs but not Scilab `strindex`.

The function `mtlb_strfind(A,B)` is used by `mfile2sci` to replace `strfind(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strfind` calls:

- If A and B are character strings `mtlb_strfind(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_strfind` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_strep` — Matlab `strep` emulation function

Description

Matlab `strep` and Scilab `strsubst` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strep` can be used with not character strings inputs but not Scilab `strsubst`.

The function `mtlb_strep(A,B,C)` is used by `mfile2sci` to replace `strep(A,B,C)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strep` calls:

- If `A`, `B` and `C` are character strings `mtlb_strep(A,B,C)` may be replaced by `strsubst(A,B,C)`

Caution: `mtlb_strep` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_sum` — Matlab sum emulation function

Description

Matlab and Scilab `sum` behave differently in some particular cases:

- When called with one input: Matlab `sum` threats the values along the first non-singleton dimension of input while Scilab `sum` threats all values of input.
- When called with two inputs: Matlab `sum` can be used with second input giving a non-existing dimension of first input while Scilab `sum` returns an error message.

The function `mtlb_sum(A[, dim])` is used by `mfile2sci` to replace `sum(A[, dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sum` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_sum(A)` may be replaced by `sum(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_sum(A)` may be replaced by `sum(A, 1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_sum(A)` may be replaced by `sum(A, firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_sum(A, dim)` may be replaced by `sum(A, dim)`

Caution: `mtlb_sum` has not to be used for hand coded functions.

See Also

`firstnonsingleton`

Authors

V.C.

Name

`mtlb_t` — Matlab transposition emulation function

Description

Matlab and Scilab transposition behave differently in some particular cases:

- With character strings operands: The `'` operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_t(A)` is used by `mfile2sci` to replace `A'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_t` calls:

- If `A` is not a character string matrix `mtlb_t(A)` may be replaced by `A'`

Caution: `mtlb_t` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings`

Authors

V.C.

Name

`mtlb_toeplitz` — Matlab toeplitz emulation function

Description

Matlab and Scilab `toeplitz` behave differently in some particular cases:

- With one input parameter: if this parameter is complex or is a matrix, output value of Matlab and Scilab `toeplitz` can be different.
- With two input parameters: if they are vectors and their first elements are not equal, Scilab returns an error but Matlab gives priority to the column element. If they are matrices, output value of Matlab and Scilab `toeplitz` are different.

The function `mtlb_toeplitz(c[,r])` is used by `mfile2sci` to replace `toeplitz(c[,r])` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_toeplitz` calls:

- When called with one input, if `c` is a real scalar or vector `mtlb_toeplitz(c)` may be replaced by `toeplitz(c)`
- When called with two inputs, if `c` and `r` are scalars or vectors and their first elements are equal `mtlb_toeplitz(c,r)` may be replaced by `toeplitz(c,r)`

Caution: `mtlb_toeplitz` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_tril` — Matlab tril emulation function

Description

Matlab and Scilab `tril` behave differently in some particular cases:

- With complex input: Matlab `tril` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `tril` do not give the same results for this type of input.
- With boolean inputs: Matlab `tril` can be used with boolean data but not Scilab one.

The function `mtlb_tril(x, k)` is used by `mfile2sci` to replace `tril(x, k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_tril` calls:

- If `X` contains real double values `mtlb_tril(x, k)` may be replaced by `tril(x, k)`
- If `X` contains boolean values `mtlb_tril(x, k)` may be replaced by `tril(bool2s(x), k)`

Caution: `mtlb_tril` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings`

Authors

V.C.

Name

`mtlb_triu` — Matlab triu emulation function

Description

Matlab and Scilab `triu` behave differently in some particular cases:

- With complex input: Matlab `triu` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `triu` do not give the same results for this type of input.
- With boolean inputs: Matlab `triu` can be used with boolean data but not Scilab one.

The function `mtlb_triu(x, k)` is used by `mfile2sci` to replace `triu(x, k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_triu` calls:

- If `X` contains real double values `mtlb_triu(x, k)` may be replaced by `triu(x, k)`
- If `X` contains boolean values `mtlb_triu(x, k)` may be replaced by `triu(bool2s(x), k)`

Caution: `mtlb_triu` has not to be used for hand coded functions.

See Also

`Matlab-Scilab_character_strings`

Authors

V.C.

Name

`mtlb_true` — Matlab true emulation function

Description

There is no Scilab equivalent for Matlab `true`. However, Scilab `ones` returns a result interpreted in an equivalent way for Scilab.

Matlab `true` and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `true` returns a $n \times n$ matrix of ones while Scilab `ones` returns a 1.

The function `mtlb_true(A)` is used by `mfile2sci` to replace `true(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_true` calls:

- If `A` is a scalar `mtlb_true(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_true(A)` may be replaced by `ones(A)`

Caution: `mtlb_true` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_uint16` — Matlab `uint16` emulation function

Description

Matlab and Scilab `uint16` behave differently in some particular cases:

- With complex input: Matlab `uint16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint16` returns 65535 and Scilab returns 0.

The function `mtlb_uint16(A)` is used by `mfile2sci` to replace `uint16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint16` calls:

- If `A` does not contain `%inf` values `mtlb_uint16(A)` may be replaced by `uint16(A)`

Caution: `mtlb_uint16` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_uint32` — Matlab `uint32` emulation function

Description

Matlab and Scilab `uint32` behave differently in some particular cases:

- With complex input: Matlab `uint32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint32` returns 4294967295 and Scilab returns 0.

The function `mtlb_uint32(A)` is used by `mfile2sci` to replace `uint32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint32` calls:

- If `A` does not contain `%inf` values `mtlb_uint32(A)` may be replaced by `uint32(A)`

Caution: `mtlb_uint32` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_uint8` — Matlab uint8 emulation function

Description

Matlab and Scilab `uint8` behave differently in some particular cases:

- With complex input: Matlab `uint8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint8` returns 255 and Scilab returns 0.

The function `mtlb_uint8(A)` is used by `mfile2sci` to replace `uint8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint8` calls:

- If `A` does not contain `%inf` values `mtlb_uint8(A)` may be replaced by `uint8(A)`

Caution: `mtlb_uint8` has not to be used for hand coded functions.

Authors

V.C.

Name

`mtlb_upper` — Matlab upper emulation function

Description

Matlab `upper(A)` and Scilab `convstr(A, "u")` behave differently in some particular cases:

- If `A` is not a character string matrix: Matlab `upper` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_upper(A)` is used by `mfile2sci` to replace `upper(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_upper` calls:

- If `A` is a character string matrix `mtlb_upper(A)` may be replaced by `convstr(A, "u")`
- If `A` is not a character string matrix `mtlb_upper(A)` may be replaced by `A`

Caution: `mtlb_upper` has not to be used for hand coded functions.

See Also

`convstr`

Authors

V.C.

Name

mtlb_var — Matlab var emulation function

Parameters

x

a real or a complex vector or matrix.

s

a real scalar or real vector.

- If **x** is a vector, **s** is the variance of **x**.
- If **x** is a matrix, **s** is a row vector containing the variance of each column of **x**.

w

type of normalization to use. Valid values are, depending on the number of columns **m** of **x** :

- **w** = 0 : normalizes with **m**-1, provides the best unbiased estimator of the variance (this is the default).
- **w** = 1: normalizes with **m**, this provides the second moment around the mean.

dim

the dimension along which the variance is computed (default is 1, i.e. column by column). If **dim** is 2, the variance is computed row by row.

Description

This function computes the variance of the values of a vector or matrix **x**. It provides the same service as Octave and Matlab. It differs from Scilab's variance primitive:

- **mtlb_var** returns a real (i.e. with a zero imaginary part) variance, even if **x** is a complex vector or matrix. The Scilab variance primitive returns a complex value if the input vector **x** is complex and if no option additionnal is used.
- Whatever the type of the input data **x** (i.e. vector or matrix), **mtlb_var** computes the variance either on dimension 1 or on dimension 2 while, if no option is passed to the Scilab's variance primitive, the variance is computed on all dimension at once.

Examples

The following 3 examples illustrates the use of the **mtlb_var** function. In the first case, a column vector is passed to the function, which returns the value 750. In the second case, a matrix is passed to the function, which returns the row vector [0.16 0.09]. In the third case, a complex column vector is passed to the function, which returns a value close to 2.

```
x = [10; 20; 30; 40; 50; 60; 70; 80; 90];
computed = mtlb_var(x);

x = [0.9    0.7
     0.1    0.1
     0.5    0.4];
computed = mtlb_var(x);

N=1000;
x = grand(N,1,'nor',0,1) + %i*grand(N,1,'nor',0,1);
```

```
computed = mtlb_var(x);
```

See Also

[variance](#)

Authors

Michael Baudin

Name

mtlb_zeros — Matlab zeros emulation function

Description

Matlab and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `zeros` returns a $n \times n$ matrix while Scilab returns a 0.

The function `mtlb_zeros(A)` is used by `mfile2sci` to replace `zeros(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_zeros` calls:

- If `A` is a scalar `mtlb_zeros(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_zeros(A)` may be replaced by `zeros(A)`

Caution: `mtlb_zeros` has not to be used for hand coded functions.

Authors

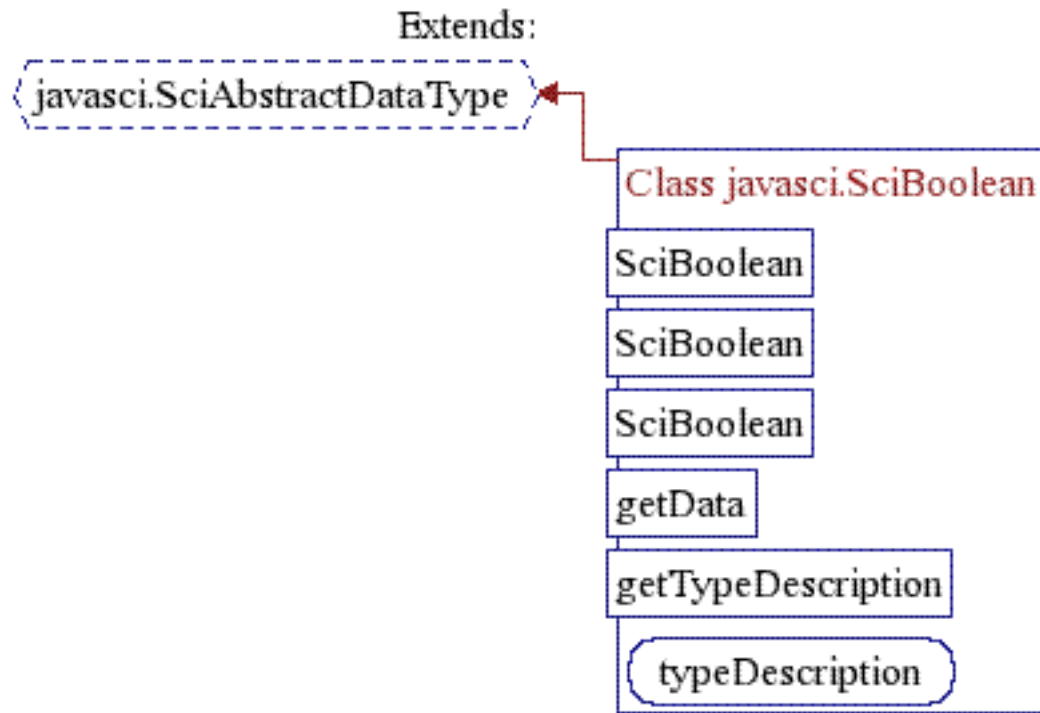
V.C.

Part XXXVII. Java Interface

Name

javasci.SciBoolean — Class to use boolean object with scilab

Description



Method Summary :

```
public SciBoolean(String name,SciBoolean Obj)
```

```
public SciBoolean(String name) Constructor (if name exists in Scilab and has the same  
type, variable is imported from Scilab)
```

```
public SciBoolean(String name,boolean Value )
```

```
public String getName()Get Name of scilab object
```

```
public boolean getData()Get Value of scilab object
```

```
public void Get()Get in java object , value of scilab object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

```
public void Send()Send to scilab object , value of java object
```

```
public void disp()disp object
```

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciBooleanArray , SciDoubleArray , SciString ,
SciStringArray

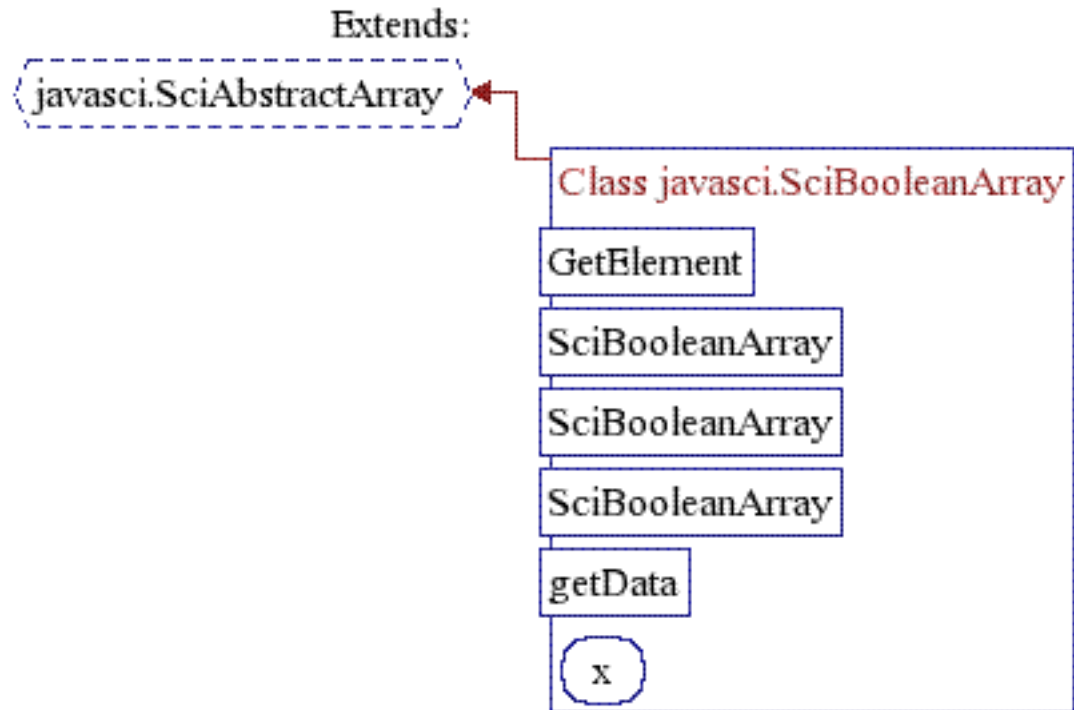
Authors

A.C

Name

javasci.SciBooleanArray — Class to use boolean matrix in Scilab.

Description



Method Summary :

```
public SciBooleanArray(String name,SciBooleanArray Obj)
public SciBooleanArray(String name,int r,int c)
public SciBooleanArray(String name,int r,int c,boolean [] x )Constructor
public int getNumbersOfRows()Get number of rows
public int getNumbersOfCols() Get number of colons
public String getName()Get Name of scilab object
public boolean[] getData() Get Value of scilab object
public void disp() disp object
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
public void Get()Get in java object , value of scilab object
public void Send()Send to scilab object , value of java object
public boolean GetElement(int indr, int indc)Get a specific element of scilab
object
```

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciBoolean , SciDouble , SciString , SciStringArray

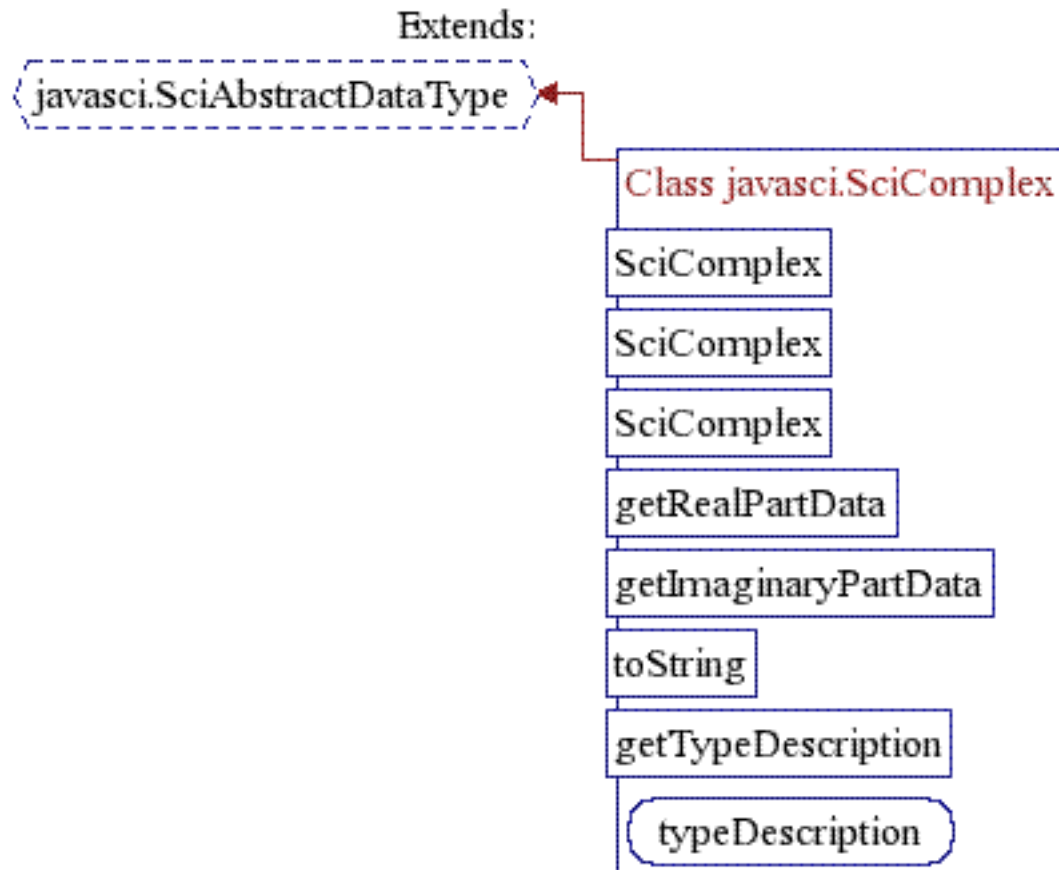
Authors

A.C

Name

javasci.SciComplex — Class to use complex object with scilab

Description



Method Summary :

```
public SciComplex(String name,SciComplex Obj)
```

```
public SciComplex(String name) Constructor (if name exists in Scilab and has the same  
type, variable is imported from Scilab)
```

```
public SciComplex(String name,double realpart,double imaginary-  
part )Constructor
```

```
public String getName()Get Name of scilab object
```

```
public double getRealPartData()Get Real Part Value of scilab object
```

```
public double getImaginaryPartData()Get Imaginary Part Value of scilab object
```

```
public void Get()Get in java object , value of scilab object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

```
public void Send()Send to scilab object , value of java object
```

```
public void disp()disp object
```

```
public void toString()convert complex to a string
```

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciComplexArray , SciString , SciStringArray , Sci-DoubleArray , SciDouble

Authors

A.C

Name

javasci.SciComplexArray — Class to use complex matrix in Scilab.

Descr

GetRealPartElement

GetImaginaryPartElement

SciComplexArray

SciComplexArray

SciComplexArray

getData

getName

getRealPartData

getImaginaryPartData

Job

Get

Send

getNumberOfRows

getNumberOfCols

getRow

getCol

disp

x

y

m

n

name

Initialize

getNumberOfRowsFromScilab

getNumberOfColsFromScilab

Method Summary :

```
public SciComplexArray(String name,SciComplexArray Obj)
public SciComplexArray(String name,int r,int c)
public SciComplexArray(String name,int r,int c,double []
realpart,double [] imaginarypart)Constructor
public int getNumbersOfRows()Get number of rows
public int getNumbersOfCols() Get number of colons
public String getName()Get Name of scilab object
public double[] getRealPartData()Get Real Part Value of scilab object
public double[] getImaginaryPartData()Get Imaginary Part Value of scilab object
public void disp() disp object
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
public void Get()Get in java object , value of scilab object
public void Send()Send to scilab object , value of java object
public double GetRealPartElement(int indr, int indc)Get a specific element
of scilab object
public double GetImaginaryPartElement(int indr, int indc)Get a specific
element of scilab object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciComplex , SciDouble , SciDoubleArray , SciString , SciStringArray

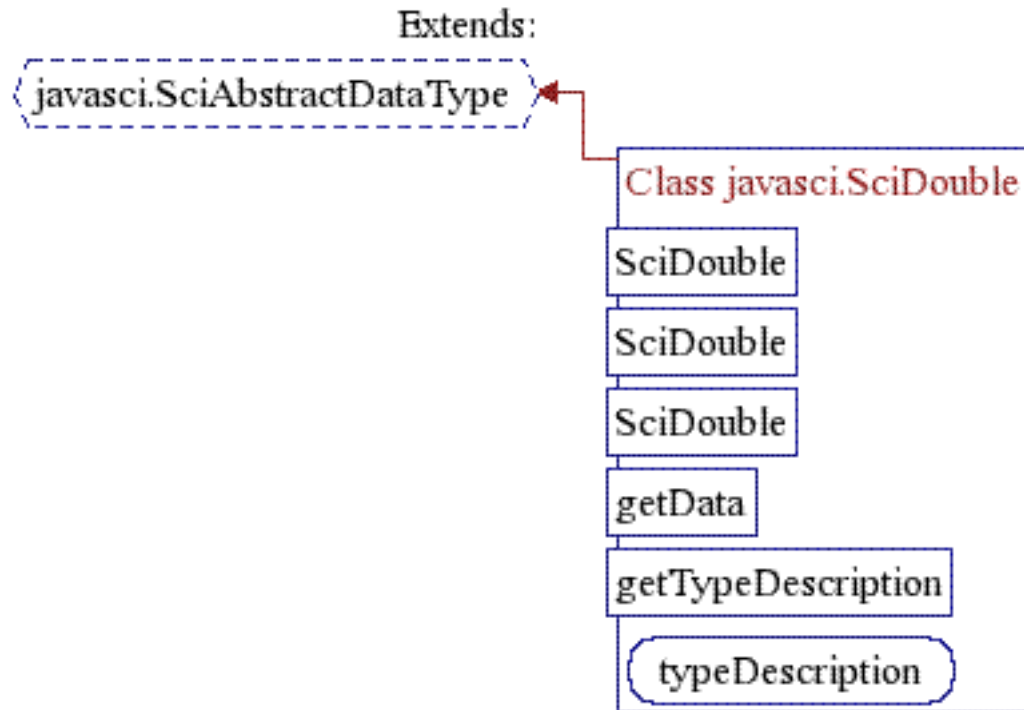
Authors

A.C

Name

javasci.SciDouble — Class to use double object with scilab

Description



Method Summary :

```
public SciDouble(String name,SciDouble Obj)
```

```
public SciDouble(String name) Constructor (if name exists in Scilab and has the same  
type, variable is imported from Scilab)
```

```
public SciDouble(String name,double Value )
```

```
public String getName()Get Name of scilab object
```

```
public double getData()Get Value of scilab object
```

```
public void Get()Get in java object , value of scilab object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

```
public void Send()Send to scilab object , value of java object
```

```
public void disp()disp object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDoubleArray , SciString , SciStringArray

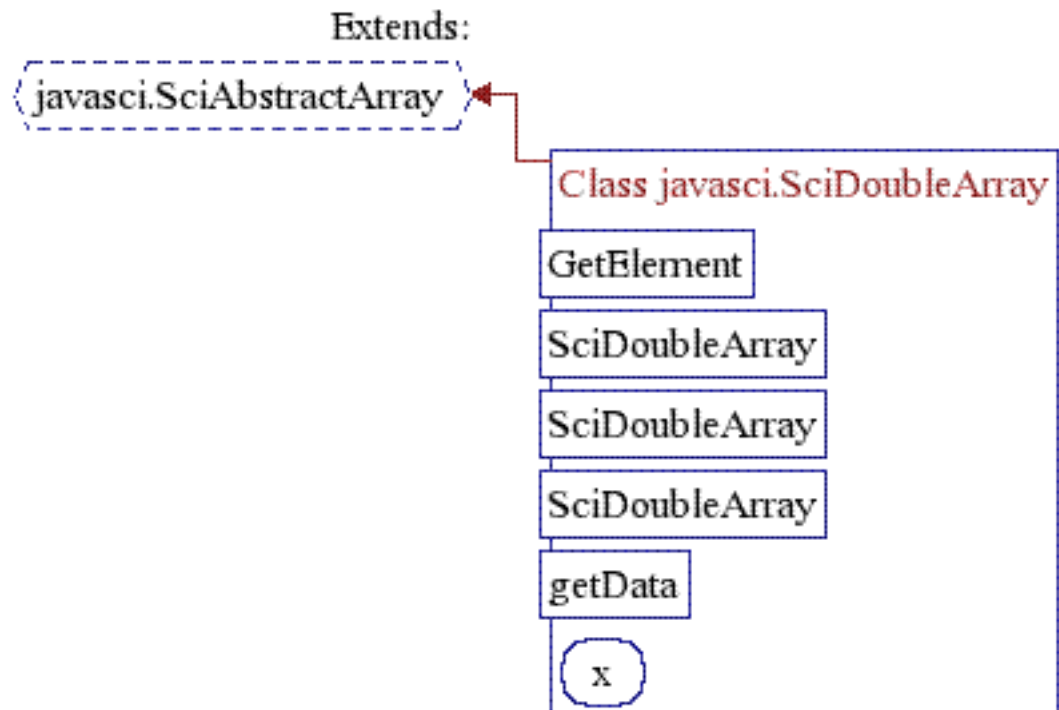
Authors

A.C

Name

javasci.SciDoubleArray — Class to use real matrix in Scilab.

Description



Method Summary :

```
public SciDoubleArray(String name,SciDoubleArray Obj)
```

```
public SciDoubleArray(String name,int r,int c)
```

```
public SciDoubleArray(String name,int r,int c,double [] x )Constructor
```

```
public int getNumbersOfRows()Get number of rows
```

```
public int getNumbersOfCols() Get number of colons
```

```
public String getName()Get Name of scilab object
```

```
public double[] getData() Get Value of scilab object
```

```
public void disp() disp object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

```
public void Get()Get in java object , value of scilab object
```

```
public void Send()Send to scilab object , value of java object
```

```
public double GetElement(int indr, int indc)Get a specific element of scilab object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciString , SciStringArray

Authors

A.C

Name

javasci.SciInteger — Class to use integer object with scilab

Description

Method Summary :

```
public SciInteger(String name,SciInteger Obj)
```

```
public SciInteger(String name) Constructor (if name exists in Scilab and has the same  
type, variable is imported from Scilab)
```

```
public SciInteger(String name,int Value )
```

```
public String getName()Get Name of scilab object
```

```
public int getData()Get Value of scilab object
```

```
public void Get()Get in java object , value of scilab object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

```
public void Send()Send to scilab object , value of java object
```

```
public void disp()disp object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciIntegerArray , SciString , SciStringArray

Authors

A.C

Name

javasci.SciIntegerArray — Class to use int matrix in Scilab.

Description

Method Summary :

```
public SciIntegerArray(String name,SciIntegerArray Obj)
public SciIntegerArray(String name,int r,int c)
public SciIntegerArray(String name,int r,int c,int [] x )Constructor
public int getNumbersOfRows() Get number of rows
public int getNumbersOfCols() Get number of colons
public String getName() Get Name of scilab object
public int[] getData() Get Value of scilab object
public void disp() disp object
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
public void Get() Get in java object , value of scilab object
public void Send() Send to scilab object , value of java object
public int GetElement(int indr, int indc) Get a specific element of scilab object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciInteger , SciString , SciStringArray

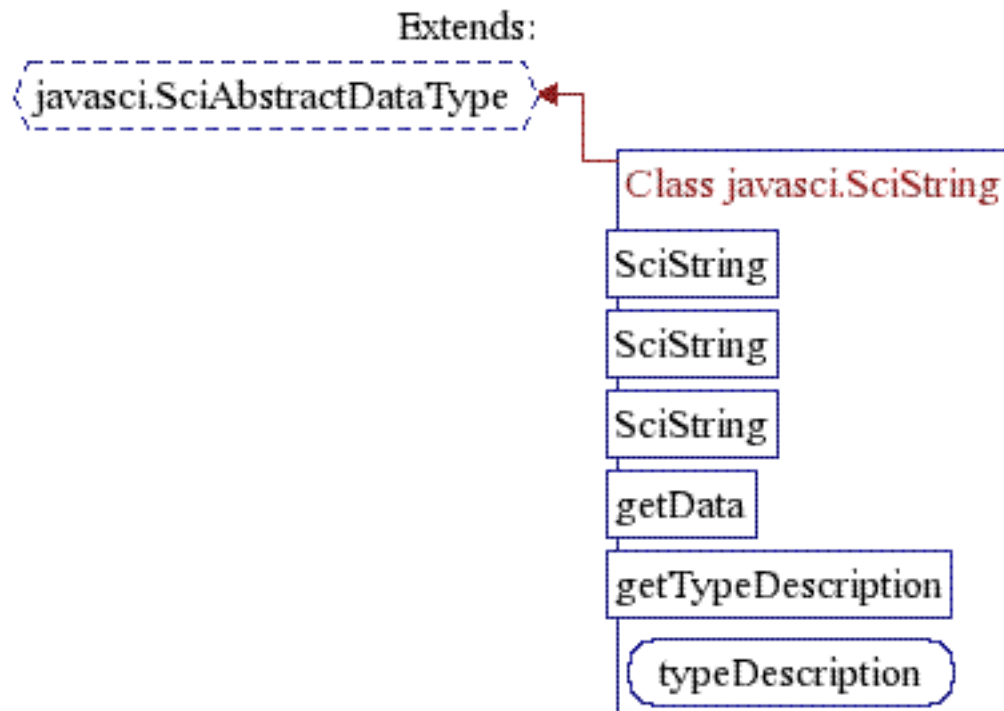
Authors

A.C

Name

javasci.SciString — Map a Java String into a Scilab string.

Description



Method Summary :

```
public SciString(String name,SciString Obj)
```

```
public SciString(String name) Constructor (if name exists in Scilab and has the same  
type, variable is imported from Scilab)
```

```
public String getName()Get Name of scilab object
```

```
public void Get()Get in java object , value of scilab object
```

```
public String getData() :Get Value of scilab object
```

```
public void Send()Send to scilab object , value of java object
```

```
public void disp() disp object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciStringArray

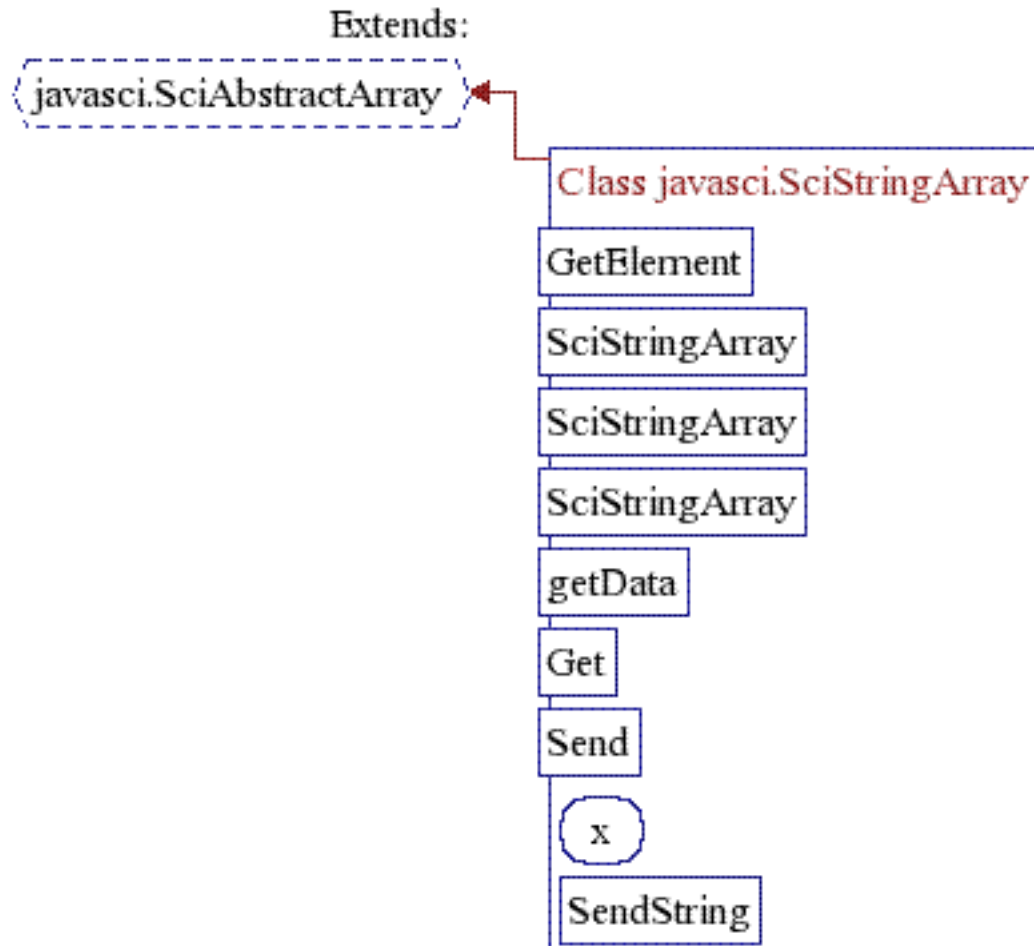
Authors

A.C

Name

javasci.SciStringArray — Classe to use String matrix in Scilab.

Description



Method Summary :

```
public SciDoubleArray(String name,SciDoubleArray Obj)
```

```
public SciStringArray(String name,int r,int c)
```

```
public SciStringArray(String name,int r,int c,String [] x )Constructor
```

```
public int getNumbersOfRows()Get number of rows
```

```
public int getNumbersOfCols() Get number of colons
```

```
public String getName()Get Name of scilab object
```

```
public String[] getData()Get Value of scilab object
```

```
public void Get() Get in java object , value of scilab object
```

```
public void Send() Send to scilab object , value of java object
```

```
public void disp() disp object
```

```
public boolean Job(String job)(deprecated see Scilab.Exec) Execute a job in scilab  
public String GetElement(int indr, int indc)Get a specific element of scilab object
```

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciString

Authors

A.C

Name

javasci.Scilab — This class provides the basic methods to execute Scilab code and scripts.

Description

This class is static. Since the Scilab environnement is persistent, all variables will remain accessible with the Java application.

Method Summary :

- *public static void Events()* - Execute a Scilab event
- *public static boolean HaveAGraph()* - Check if there is any scilab graphic window open (return True if it is the case).
- *public static boolean Exec(String job)* - Execute a job in scilab. return true if there is no error.

Note that the given expression must be consistent by itself. Then, a serie of **Exec** defining a function will not work. Please consider calling it with a single string instruction or using a temporary file with the method **ExecuteScilabScript**.

For example: `Scilab.Exec("function foo(); disp('bar'); end-function");` will work when `Scilab.Exec("function foo();");`
`Scilab.Exec("disp('bar');");` `Scilab.Exec("endfunction;");` will not work since each statement being processed independently

- *public static native boolean ExistVar(String VarName)* - Detect if VarName exists in Scilab. return true if Varname exist.
- *public static native int TypeVar(String VarName)* - Return Scilab type of VarName. See type
- *public static native int GetLastErrorCode()* - Return last Error code. See lasterror
- *public static boolean ExecuteScilabScript(String scilabscriptfilename)* - Execute a scilab script (.sce) return true if there is no error.
- *public static native boolean Finish()* - Terminate scilab (call scilab.quit , close a scilab object)

Examples

```
// A Scilab / Java example
// Filename: ScilabExample.java

import javasci.Scilab;

public class ScilabExample {
    public static void main(String []args){
        String myVar="myVariable";
        Scilab.Exec(myVar+"=(%pi^4)/90;disp(myVariable);"); // Simple display
        if (Scilab.ExistVar(myVar)) {
            System.out.println("Variable "+myVar+" exists. Type: "+Scilab.TypeVar(myVar))
        }
        if (!Scilab.Exec("disp(plop);")) { // Error
```

```
        System.err.println("Last error: "+Scilab.GetLastErrorCode()); // Error
    }
    Scilab.Finish();
}
}
```

See Also

Compile and run with Javasci, SciDouble, SciDoubleArray, SciString, SciStringArray, type, lasterror

Authors

A.C

Name

Compile and run with javasci — How to compile a Java application using Javasci

Description

To compile a Java code based on Javasci, it is only necessary to have javasci.jar defined in the class-path.

For example, with the code defined in the example of this page, the command would be:

on Linux/Unix/MacOSX:

```
$ javac -cp $SCI/modules/javasci/jar/javasci.jar BasicExample.java
```

on Windows:

```
D:\> javac -cp %SCI%\modules\javasci\jar\javasci.jar BasicExample.java
```

To run Scilab, there are a few other things to set up.

Some global variables must be set:

- SCI - Path to Scilab files
 - Linux/Unix/MacOSX:
 - In the binary version of Scilab, SCI will point to /path/to/scilab/share/scilab/
 - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
 - Windows
 - path of scilab root directory:
set SCI=C:\program files\scilab-5.1
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .jnilib...)
 - Linux/Unix/MacOSX:
 - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
 - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/javasci/.libs/ and /path/to/scilab/.libs/
 - Windows: Path to libscilab.dll and javasci.dll
 - equivalent to LD_LIBRARY_PATH on Windows is PATH
set PATH="%SCI%\bin";%PATH%

To launch the Java Application, you can either provide them with environment variable

- Linux/Unix/MacOSX:
 - LD_LIBRARY_PATH=/path/to/libjavasci/ SCI=/path/to/scilab/ java -cp modules/javasci/jar/javasci.jar:. BasicExample
 - SCI=/path/to/scilab/ java -Djava.library.path=/path/to/libjavasci/ -cp modules/javasci/jar/javasci.jar:. BasicExample

or with the arguments

- Windows:

```
set SCI=c:\program files\scilab-5.1
```

```
set PATH="%SCI%\bin";%PATH%
```

```
D:\java -cp "%SCI%\modules\javasci\jar\javasci.jar";. BasicExample
```

optional options to launch java : -Djava.compiler=JIT -Xmx256m

(With these arguments , you launch javasci with same initial options that standard scilab).

Note that two environnement variables are taken in account for specific needs:

- SCI_DISABLE_TK=1 Disables Tk (Tcl's GUI)
- SCI_JAVA_ENABLE_HEADLESS=1 Launch Java in headless mode (no AWT/Swing)

Examples

```
// A simple Java example
// Filename: BasicExample.java

import javasci.Scilab;

public class BasicExample {
    public static void main(String []args){
        Scilab.Exec("disp((%pi^2)/6);");
    }
}
```

See Also

Javasci, Javasci FAQ, SciDouble, SciDoubleArray, SciString, SciStringArray

Authors

Allan Cornet

Sylvestre Ledru

Name

javasci — Call Scilab engine from a Java application

Description

Scilab offers the possibility to be called from a Java application.

This help describes the features of the javasci API.

Examples

```
// A simple Java example
// Filename: DisplayPI.java

import javasci.Scilab;

public class DisplayPI {
    public static void main(String []args){
        Scilab.Exec("disp(%pi);");
    }
}
```

See Also

Compile and run with Javasci, Javasci FAQ, SciDouble, SciDoubleArray, SciString, SciStringArray

Authors

Allan Cornet

Sylvestre Ledru

Name

javasci FAQ — The frequently asked questions

Questions / Answers

1. Running an application based on javasci, I get the error Exception in thread "main" java.lang.NoClassDefFoundError: javasci/Scilab

javasci.jar is probably not defined in the CLASSPATH. See

See Also

Javasci, Compile and run with javasci

Authors

Sylvestre Ledru

Part XXXVIII. Maple Interface

Name

sci2map — Scilab to Maple variable conversion

```
txt=sci2map(a, Map-name)
```

Parameters

a
Scilab object (matrix, polynomial, list, string)

Map-name
string (name of the Maple variable)

txt
vector of strings containing the corresponding Maple code

Description

Makes Maple code necessary to send the Scilab variable **a** to Maple: the name of the variable in Maple is **Map-name**. A Maple procedure `maple2scilab` can be found in `SCI/modules/maple2scilab/src/maple/maple2scilab.mpl` directory.

Examples

```
txt=[sci2map([1 2;3 4], 'a');  
      sci2map(%s^2+3*%s+4, 'p')]
```

Part XXXIX. Matlab to Scilab Conversion Tips

Name

About M2SCI tools — Generally speaking about tools to convert Matlab files to Scilab

Description

Scilab includes useful tools to convert Matlab M-files to Scilab.

Taking a Matlab M-file, `mfile2sci` modifies this files so that it can be compiled by Scilab. After that this compiled code is converted to a "tree" of instructions by `macr2tree`. This "tree" is an imbrication of Scilab lists and tlists and is the basis for conversion. Each instruction of this "tree" is converted to Scilab and inference is done to know what are the variables. Once this "tree" is converted to Scilab, code is generated using `tree2code`.

All tlists used for coding this tree (and we call "MSCI tlists") are listed below:

- `funcall`
tlist representing a function call created by `Funcall`
- `operation`
tlist representing an operation created by `Operation`
- `variable`
tlist representing a variable created by `Variable`
- `cste`
tlist representing a constant created by `Cste`
- `equal`
tlist representing an instruction created by `Equal`
- `ifthenelse`
tlist representing an IF/THEN/ELSE control instruction created inside M2SCI kernel functions
- `while`
tlist representing a WHILE control instruction created inside M2SCI kernel functions
- `selectcase`
tlist representing a SELECT/CASE control instruction created inside M2SCI kernel functions
- `for`
tlist representing a FOR control instruction created inside M2SCI kernel functions

The contents of these tlists is described in corresponding help pages.

Operations are converted using a fonction named `%<opcode>2sci` with opcode the Scilab code for this operator. See help page for overloading to have these codes. All these functions are already written and are in directory `SCI/modules/m2sci/macros/percent/`.

Function calls are converted using a function called `sci_<Matlab_function_name>`. Some of these functions have been written and are in directory `SCI/modules/m2sci/macros/sci_files/`. We are working on increasing the set of Matlab functions converted. However, everybody can written such functions using help page `sci_files`.

Inference is done using tlists of type "infer" containing fields:

- `dims`
list of dimensions
- `type`
"type" tlist

contents

"contents" tlist if a Cell or a Struct

Type is a tlist of type "type" containing fields:

- vtype
data type

property

property

To have more details about inference see help page for m2scideclare.

See Also

mfile2sci , translatepaths , overloading , sci_files , Funcall , Operation , Variable , Cste , Infer , Type , Equal , m2scideclare

Authors

V.C.

Name

Contents — Create a tree containing contents inference data

```
contents=Contents(list_of_index, list_of_infer)
```

Parameters

`list_of_index`

list of indexes similar to indexes returned by `macr2tree`.

`list_of_infer`

list of "infer" tlists containing inference data for matching index.

`contents`

a "contents" tlist

Description

This function create a `tlist` representing inference data for the contents of a Cell or a Struct when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI) Please ensure that for each entry you insert in `list_of_index`, you also insert an entry in `list_of_infer`.

See Also

`get_contents_infer` , `Funcall` , `Operation` , `Variable` , `Cste` , `Infer` , `Type` , `Equal`

Authors

V.C.

Name

Cste — Create a tree representing a constant

```
const=Cste(value)
```

Parameters

value
 constante value

const
 a "cste" tlist

Description

This function create a `t list` representing a constant when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

[Funcall](#) , [Operation](#) , [Variable](#) , [Infer](#) , [Contents](#) , [Type](#) , [Equal](#)

Authors

V.C.

Name

Equal — Create a tree representing an instruction

```
eq=Equal(lhslist,expression)
```

Parameters

`lhslist`
list of lhs parameters (list of "M2SCI tlists")

`expression`
right member of equal (an "M2SCI tlist")

`eq`
an "equal" tlist

Description

This function create a `tlist` representing an instruction when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

[Funcall](#) , [Operation](#) , [Variable](#) , [Cste](#) , [Infer](#) , [Contents](#) , [Type](#)

Authors

V.C.

Name

Funcall — Create a tree representing a function call

```
fc=Funcall(name, lhsnb, rhslist, lhslist)
```

Parameters

name

function name (character string)

lhsnb

number of outputs (constant)

rhslist

list of inputs (list of "M2SCI tlists")

lhslist

list of outputs (list of "M2SCI tlists")

fc

a "funcall" tlist

Description

This function create a `tlist` representing a function call when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Operation , Variable , Cste , Infer , Contents , Type , Equal

Authors

V.C.

Name

Infer — Create a tree containing inference data

```
infer=Infer(varargin)
```

Parameters

varargin

data for inference

varargin(1)

list of dimensions default value is list(Unknown,Unknown)

varargin(2)

type ("type" tlist, see Type help page) default value is Type(Unknown,Unknown)

varargin(3)

contents ("contents" tlist, see Contents help page) default value is Contents(list(),list()). This field is only used if represented data is a Cell or a Struct.

infer

an "infer" tlist

Description

This function create a `tList` representing inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

See Also

Funcall , Operation , Variable , Cste , Contents , Type , Equal

Authors

V.C.

Name

Matlab-Scilab_character_strings — Generally speaking about...

Description

Matlab and Scilab character strings are not considered in the same way. Here is a little talk about differences between them.

Matlab considers a character string as Scilab considers a matrix of characters. For example, a Scilab equivalent for Matlab 'mystring' could be ["m","y","s","t","r","i","n","g"]. So in Scilab, a character string is a object of type string (10) and always have size 1 x 1 but in Matlab, a character string have size equal to 1 x number_of_characters.

Considering this, we can see that a Matlab character string matrix column can only be made of same-size character strings what is not true in Scilab. We can say that a Scilab character string matrix is equivalent to a Matlab cell of character strings.

All these differences can lead to different results while executing same commands in Scilab or in Matlab, particularly for "dimension" functions such as length() or size().

See Also

mstr2sci

Authors

V.C.

Name

Operation — Create a tree representing an operation

```
op=Operation(operator, operands, out)
```

Parameters

operator

operator symbol (character string)

operands

list of operands (list of "M2SCI tlists")

out

list of outputs (list of "M2SCI tlists")

op

an "operation" tlist

Description

This function create a `tlist` representing an operation when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

`Funcall` , `Variable` , `Cste` , `Infer` , `Contents` , `Type` , `Equal`

Authors

V.C.

Name

Type — Create a tree containing type inference data

```
tp=Type(vtype,property)
```

Parameters

vtype

data type (see m2scideclare)

property

property of data (see m2scideclare)

tp

a "type" tlist

Description

This function create a `tlist` representing type inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

See Also

Funcall , Operation , Variable , Cste , Infer , Contents , Equal , m2scideclare

Authors

V.C.

Name

Variable — Create a tree representing a variable

```
var=Variable(name,infer)
```

Parameters

var

variable name (character string)

infer

inference data (a `tlist` of type "infer", see Infer help page)

var

a "variable" `tlist`

Description

This function create a `tlist` representing a variable when using M2SCI. All input parameters values are verified to be compatible with "M2SCI `tlists`".

See Also

`Funcall` , `Operation` , `Cste` , `Infer` , `Contents` , `Type` , `Equal`

Authors

V.C.

Name

`get_contents_infer` — Search for information in a "M2SCi tlist" contents

```
[infer, pos]=get_contents_infer(m2scitlist, index)
```

Parameters

`m2scitlist`

a "M2SCI tlist"

`index`

an index similar to indexes returned by `macr2tree`.

`infer`

an "infer" tlist

`pos`

position of information in list

Description

This functions searches for inference informations of a given index in the contents of a Cell or a Struct taken in account the *. If no information has been found, returned values are `infer=infer()` and `pos=0`.

See Also

`Infer` , `Contents`

Authors

V.C.

Name

m2scideclare — Giving tips to help M2SCI...

Description

The main difficulty for M2SCI (`mfile2sci`) is to find what variables are: dimensions, type...

To help this tool, just add comments beginning with `%m2scideclare` in the M-file to convert, (`%m2sciassume` was used in previous Scilab versions and is now obsolete).

The syntax of this command is:

```
%m2scideclare variable_name|dimensions|data_type|property
```

with :

- `variable_name`: name of the variable declared. It can be a Struct field (e.g. `x(1,2).name`) or describe the contents of a Cell using syntax `x(1,2).entries`. NOTE that for Cells and Structs, `*` can be used as an index (see examples below).
- `dimensions`: dimensions of the variable declared separated by blanks, if a dimension is unknown, replace it by `?`. NOTE that String dimensions must be similar to Matlab ones e.g. `1 6` for character string `'string'`.
- `data_type`: data type of the variable which can be:

m2scideclare data type	Scilab "equivalent" type
Double	1
Boolean	4
Sparse	5
Int	8
Handle	9
String	10
Struct	Matlab struct (16)
Cell	Matlab cell (17)
Void	No type (0)
?	Unknown type

- `property`: property of the variable which can be:

m2scideclare property	Scilab "equivalent"
Real	Real data
Complex	Complex data
?	Unknown property

This field is ignored for following datatypes: `Cell`, `Struct`, `String` and `Boolean`.

All data given by `m2scideclare` are compared with inferred data, in case of conflict, inferred data are kept and a warning message is displayed. If you are sure about your data, report a bug.

Some examples are given below:

- `%m2scideclare var1|2 3|Double|Real`var1 is declared as a 2x3 Double matrix containing real data

- `%m2scideclare var2|2 3 10|Double|Complex`var2 is declared as a 2x3x10 Double hypermatrix containing complex data
- `%m2scideclare var3(1,2).name|1 10|String|?`var3 is declared as a Struct array containing a 1x10 character string in field 'name' of struct at index (1,2)
- `%m2scideclare var4(1,5).entries|1 ?|Boolean|?`var4 is declared as a Cell containing a row boolean vector at index (1,5)
- `%m2scideclare var4(1,6).entries|? ?|Int|?`var4 is declared as a Cell containing a row boolean vector at index (1,5) and integer data at index (1,6)
- `%m2scideclare var5(*,*).name|1 ?|String|?`var5 is declared as a Struct array containing a 1xn character string in all fields 'name'
- `%m2scideclare var6(2,*).entries|1 3|Double|Real`var6 is declared as a Cell array containing a 1x3 double vector in each element of its second row

Authors

V.C.

Name

matfile2sci — converts a Matlab 5 MAT-file into a Scilab binary file

```
matfile2sci(mat_file_path,sci_file_path)
```

Parameters

mat_file_path

character string containing the path of the Matlab input file

sci_file_path

character string containing the path of the Scilab output file

Description

Converts a Matlab 5 MAT-file into a Scilab binary file compatible with the function `load`. The Matlab data types are converted into the Scilab equivalents.

See Also

`loadmatfile` , `load` , `mfile2sci`

Authors

Serge Steer (INRIA)

Bibliography

This function has been developed according to the document "MAT-File Format": >Mat-File Format
[http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]

Name

mfile2sci — Matlab M-file to Scilab conversion function

```
mfile2sci([M-file-path [,result-path [,Recmode [,only-double [,verbose-mode [,p
```

Parameters

M-file-path

a character string which gives the path of Matlab M-file to convert

result-path

a character string which gives the directory where the result has to be written. Default value is current directory.

Recmode

Boolean flag, used by translatepaths function for recursive conversion. Must be %F to convert a single mfile. Default value : %f

only-double

Boolean flag, if %T mfile2sci considers that numerical function have been used only with numerical data (no Scilab overloading function is needed). Default value: %F

verbose-mode

display information mode

0

no information displayed

1

information written as comment is resulting SCI-file

2

information written as comment is resulting SCI-file and in logfile

3

information written as comment is resulting SCI-file, in logfile and displayed in Scilab window

prettyprintoutput

Boolean flag, if %T generated code is beautified. Default value: %F

Description

M2SCI (and particularly mfile2sci) is Matlab M-file to Scilab function conversion tools. It tries whenever possible to replace call to Matlab functions by the equivalent Scilab primitives and functions.

To convert a Matlab M-file just enter the Scilab instruction: mfile2sci(file)

where file is a character string giving the path name of the M-file mfile2sci will generate three files in the same directory

<function-name>.sci

the Scilab equivalent of the M-file

<function-name>.cat

the Scilab help file associated to the function

sci_<function-name>.sci

the Scilab function required to convert the calls to this Matlab M-file in other Matlab M-files. This function may be improved "by hand". This function is only useful for conversion not for use of translated functions.

Some functions like eye, ones, size, sum,... behave differently according to the dimension of their arguments. When mfile2sci cannot infer dimensions it replaces these function call by a call to an emulation function named mtlb_<function_name>. For efficiency these functions may be replaced by the proper scilab equivalent instructions. To get information about replacement, enter: help mtlb_<function_name> in Scilab command window

Some other functions like plot, has no straightforward equivalent in scilab. They are also replaced by an emulation function named mtlb_<function_name>.

When translation may be incorrect or may be improved mfile2sci adds a comment which begins by "/*!" (according to verbose-mode)

When called without rhs, mfile2sci() launches a GUI to help to select a file/directory and options.

Examples

```
// Create a simple M-file
write(TMPDIR+'/rot90.m', ['function B = rot90(A,k)'
    '[m,n] = size(A);'
    'if nargin == 1'
    '    k = 1;'
    'else'
    '    k = rem(k,4);'
    '    if k < 0'
    '        k = k + 4;'
    '    end'
    'end'
    'if k == 1'
    '    A = A.';
    '    B = A(n:-1:1,:);'
    'elseif k == 2'
    '    B = A(m:-1:1,n:-1:1);'
    'elseif k == 3'
    '    B = A(m:-1:1,:);'
    '    B = B.';
    'else'
    '    B = A;'
    'end']);

// Convert it to scilab
mfile2sci(TMPDIR+'/rot90.m',TMPDIR)

// Show the new code
write(%io(2),read(TMPDIR+'/rot90.sci',-1,1,'(a)'))

// get it into scilab
exec(TMPDIR+'/rot90.sci')

// Execute it
m=rand(4,2);rot90(m,1)
```

See Also

`translatepaths`

Authors

V. Couvert

S. Steer

Name

sci_files — How to write conversion functions

Description

To convert calls to Matlab functions, `mfile2sci` uses a function called `sci_<Matlab_function_name>`. All these functions are defined in `sci_files` in directory `SCI/modules/m2sci/macros/sci_files/`. The set of `sci_files` given in Scilab distribution does not allow to convert calls to all Matlab functions yet. However, a Scilab user can add `sci_files` (for Matlab functions or for user defined functions) to Scilab using the following tips.

In M2SCI, a function call is considered as a "tree" (it is also the case for the instructions of the file to convert), represented in Scilab by a `tlist` with following fields:

- `name`
Matlab function name
- `lhsnb`
number of Matlab function output parameters
- `lhs`
list of Matlab function output parameters
- `rhs`
list of Matlab function input parameters

A `sci_function` has one input called `tree` which is also the output of the function. A `sci_function` has to convert this incoming "tree" so that it is compatible with Scilab by changing name, `lhsnb`, `lhs` and/or `rhs`. The other task that has to be done by this function is inference. Incoming `tree` contains inference data in its `lhs` that have to be updated with what can be inferred for the outputs of this function.

Some useful functions have been written to help to create M2SCI `tlists` while writing this conversion function:

- `Funcall`
create a tree representing a function call
- `Operation`
create a tree representing an operation
- `Variable`
create a tree representing a variable
- `Cste`
create a tree representing a constante value
- `Infer`
create a tree representing inference data
- `Type`
create a tree representing type for inference
- `Equal`
create a tree representing an instruction

Some other functions have been designed to get properties of operands/inputs. Considering `A` is `tlist` used in macro tree, you can use the following functions:

Function	returns %T if...

<code>is_empty(A)</code>	all dimensions of A are 0
<code>not_empty(A)</code>	all dimensions of A are known and at least one dimension of A is not 0
<code>is_a_scalar(A)</code>	all dimensions of A are 1
<code>not_a_scalar(A)</code>	all dimensions of A are known and at least one dimension of A is not 1
<code>is_a_vector(A)</code>	all dimensions of A are known and all dimensions of A but one are equal to 1
<code>not_a_vector(A)</code>	all dimensions of A are known and at least two dimensions of A are greater than one
<code>is_real(A)</code>	A is real
<code>is_complex(A)</code>	A is complex
<code>isdefinedvar(A)</code>	A is a variable already created in M-file currently converted
<code>allunknown(A)</code>	all dimensions of A are unknown

Some other functions have been written for specific needs while writing conversion files:

- `first_non_singleton`
is an equivalent to `firstnonsingleton` for an M2SCI tlist. Calling sequence: `dim = first_non_singleton(A)`

`gettempvar`
generates a temporary variable having a name which does not already exist. Calling sequence:
`v = gettempvar()`

`insert`
allows to insert instructions. Calling sequence: `insert(Equal(...),opt)` with `opt~=1` to insert before current instruction and `opt=1` to insert after it.

`getoperands`
can be used to get each operand as a variable. Calling sequence: `[A,B] = getoperands(operation_tlist)`

`getrhs`
can be used to get each parameter as a variable. Calling sequence: `[A,...] = getrhs(funcall_tlist)`

`convert2double`
change type of input when this type is not implemented for a particular function in Scilab.
Calling sequence: `A = convert2double(A)`

To have more information about how to write such files, refer to directory `SCI/modules/m2sci/macros/sci_files/` which gives many examples from very simple ones (e.g. `sci_abs.sci`) to very complex ones (e.g. `sci_zeros.sci`).

See Also

`m2scideclare` , `Funcall` , `Operation` , `Variable` , `Cste` , `Infer` , `Type` , `Equal`

Authors

V.C.

Name

`translatepaths` — convert a set of Matlab M-files directories to Scilab

```
translatepaths(dirs_path [, res_path])
```

Parameters

`dirs_path`

a character string vector which gives the paths of Matlab M-file directories to convert

`res_path`

a character string which gives the path of the directory where the Scilab functions are written to.
Default value is current directory.

Description

`translatepaths`, converts all Matlab M-file contained in a set of directories to Scilab functions.
Each function is converted by `mfile2sci`.

Trace of conversion information is stored in a file named "log" in the `res_path` directory

When called without rhs, `translatepaths()` launches a GUI to help to select a file/directory and options.

See Also

`mfile2sci`

Authors

V. Couvert

S. Steer

Part XL. Tcl/Tk Interface

Name

ScilabEval — tcl instruction : Evaluate a string with scilab interpreter

```
ScilabEval instruction
ScilabEval instruction "seq"
ScilabEval instruction "sync"
ScilabEval instruction "sync" "seq"
ScilabEval "flush"
```

Parameters

instruction

tcl string character contains a Scilab instruction to evaluate with the current Scilab interpreter.

Description

This function must be called in a tcl/tk script executed from Scilab. It allows to associate Scilab actions to tcl/tk widgets (graphic objects) or to use Scilab to perform some computations within a tcl script.

ScilabEval instruction

If the `ScilabEval instruction` syntax is used, the `instruction` is first stored in a FIFO queue, `ScilabEval` returns immediately. Scilab executes the queued `instructions` when possible (it should be at the prompt but also at the end of each instructions of the currently running function) in the order they were submitted. This syntax can be used to associate Scilab actions to tcl/tk widgets but not into a tcl script executed by `TCL_EvalFile` or `TCL_EvalStr` because in this situation the Scilab interpreter is blocked up to the end of the script. Note that with the `ScilabEval instruction` syntax, if there are many `ScilabEval` commands stored in the queue the execution of the second one can be started in the middle of the execution of the first one (in particular if the first one contains more than a simple expression).

If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before the next queued instruction evaluation can be started. The next callback stored in the command queue will only be taken into account when the current one will be finished or paused.

ScilabEval instruction "sync"

If the `ScilabEval instruction "sync"` syntax is used, the instruction is executed immediately (not queued) and the `ScilabEval` returns when the `instruction` evaluation is finished. The scilab `instruction` evaluation may be interrupted by new or queued commands.

If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before any queued instruction evaluation can be started. The scilab `instruction` evaluation may not be interrupted by new or queued commands (except if it is paused).

ScilabEval "flush"

If the `ScilabEval "flush"` syntax is used, all the previously queued `instructions` are executed immediately and the `ScilabEval` returns when the execution is finished. Each `instruction` is executed with the option used at the time of queuing up (i.e. `seq` or no option).

The evaluation context of all these cases is the current Scilab context when the `instruction` evaluation starts.

Examples

```
//Callbacks and "seq" option usage
```

```

//create tcl instructions
tcl_script=['toplevel .w1'
            'button .w1.b -text ""Click here to execute without seq option"" -co
            'button .w1.b1 -text ""Click here to execute with seq option"" -co
            'pack .w1.b .w1.b1'
            'proc WithoutSeq {} { '
            '   ScilabEval ""cont=%f;;cont=%t;"" '
            '   ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;
            '}'
            'proc WithSeq {} { '
            '   ScilabEval ""cont=%f;;cont=%t;"" ""seq""
            '   ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;
            '}''];
mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
cont=%f;
TCL_EvalFile(TMPDIR+'/test.tcl');;

//scripts and "sync" option usage

//-----without "sync"-----
tcl_script=[' set t ""0""
            ' while {$t != ""10""} {'
            '     ScilabEval ""a=$t;mprintf(''%d ''',a);""
            '     incr t'
            ' }'];
mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');

// The ScilabEval are executed after the and of TCL_EvalFile

//-----with "sync"-----
tcl_script=[' set t ""0""
            ' while {$t != ""10""} {'
            '     ScilabEval ""a=$t;mprintf(''%d ''',a);"" ""sync""
            '     incr t'
            ' }'];
mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');

// The ScilabEval are executed synchronously with TCL_EvalFile

```

See Also

TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar

Authors

Bertrand Guiheneuf

Name

TCL_CreateSlave — Create a TCL slave interpreter

```
TCL_CreateSlave(slaveName[, isSafe])
```

Parameters

slaveName

String: Name of the TCL slave interpreter to create.

isSafe

Boolean: %T to create a safe slave interpreter, %F otherwise. The default value is %F. A safe slave is not allowed to perform some operations, see the TCL documentation for more information.

Description

This routine allows to create a TCL slave interpreter.

Examples

```
TCL_CreateSlave("TCLinterp")
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")

TCL_CreateSlave("TCLinterp", %T)
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")
```

See Also

TCL_SetVar , TCL_ExistVar , TCL_DeleteInterp

Authors

Allan CORNET

V.C.

Name

TCL_DeleteInterp — delete TCL interpreter

```
TCL_DeleteInterp(interp)
TCL_DeleteInterp()
```

Parameters

interp

character string parameter. Name of the slave tcl interpreter to delete. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

This routine allows to delete a TCL slave interpreter or the main scilab TCL interpreter.

Examples

```
TCL_SetVar("Scilab", "OK")
TCL_ExistVar("Scilab")
TCL_DeleteInterp()
TCL_ExistVar("Scilab")
TCL_CreateSlave('BisInterp')
TCL_ExistInterp('BisInterp')
TCL_SetVar("Scilab", "OK", 'BisInterp')
TCL_ExistVar("Scilab", 'BisInterp')
TCL_DeleteInterp('BisInterp')
TCL_ExistInterp('BisInterp')
```

See Also

TCL_SetVar , TCL_ExistVar , TCL_CreateSlave , TCL_ExistInterp

Authors

Allan CORNET

Name

TCL_EvalFile — Reads and evaluate a tcl/tk file

```
TCL_EvalFile(filename [,interp])
```

Parameters

filename

character string. Contains the name of the file to read and evaluate.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

With this routine, one can read and evaluate the content of a file containing tcl/tk scripts. This allows to create powerful tk interfaces.

The filename might be relative or absolute.

Advantages and drawbacks of this functionality

This routines allows to use directly tcl/tk scripts. This thus allows, for instance to use Interface Builders such as SpecTcl to design the interface. The interfaces built directly with tcl/tk scripts are much faster than the ones built with the Scilab Graphic Object library provided with tksci (see uicontrol for example). Indeed, those Objects are warpings around tk graphic widgets. Nevertheless, this way of creating graphic user interface should only be used when one aims at addressing directly specific tk/tcl features. There are two main reasons for this. First of all, there is no simple way to manipulate Scilab objects from within a tcl/tk script. Thus, the interface designer has to write two sets of callbacks routines. One to describe the changes occuring in the interface when the user acts on the widgets. The second set of call routines will perform the (pure) Scilab reactions to the user actions.

Here is an example: Suppose you design a scrollbar corresponding to a spline tension value. You want the spline to be displayed in a graphic windows and updated each time the user moves the scrollbar. At the same time, you want the value of this tension parameter to be displayed within the Interface. You will have to write a first tcl/tk (callback) function which will be automatically called by the tk scrollbar ("-command" option). This callback function will update the displayed value of the parameter in the interface and will then call the scilab routine ("ScilabEval" command) to update the graph.

Remarks on the tcl/tk script style

Because Scilab manages the tcl/tk events, it creates the root window ".", this window should not be destroyed nor directly used by your tcl/tk scripts. You should thus always create your own toplevel windows. Moreover, since this module was written at a time when namespaces didn't exist, some variables defined by scilab tcl/tk scripts could collide your code. Running your scripts in a slave interpreter may help in such a case.

Examples

```
TCL_EvalFile(SCI+"/modules/tclsci/demos/tk/puzzle");
```

See Also

ScilabEval , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar

Authors

Allan CORNET

Name

TCL_EvalStr — Evaluate a string within the Tcl/Tk interpreter

```
TCL_EvalStr(str [,interp])  
res = TCL_EvalStr(str [,interp])
```

Parameters

str

string or matrix of strings, contains a Tcl/Tk script in each element.

interp

optional character string parameter. Name of the slave Tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main Tcl interpreter created by Scilab.

res

result of the evaluation, if it is successful. This is a character string matrix giving the evaluation result for each element of the input argument str

Description

This routine allows to evaluate Tcl/Tk instructions with the Tcl/Tk interpreter launched with Scilab (when the `interp` parameter is not given), or in a slave interpreter.

When Tcl/Tk support is enabled in Scilab, you can evaluate Tcl/Tk expression from Scilab interpreter. In fact, Scilab launches a main Tcl/Tk interpreter. The Scilab instruction `TCL_EvalStr` can be used to evaluate expressions without having to write Tcl/Tk instructions in a separated file (this capability is provided by `TCL_EvalFile`).

Examples

```
//with one call  
TCL_EvalStr(["toplevel .foo1"  
            "label .foo1.l -text ""TK married Scilab !!!"""  
            "pack .foo1.l"  
            "button .foo1.b -text close -command {destroy .foo1}"  
            "pack .foo1.b"])  
  
//step by step (debugging)  
TCL_EvalStr("toplevel .foo2");  
  
// creates a toplevel TK window.  
TCL_EvalStr("label .foo2.l -text ""TK married Scilab !!!""");  
  
// create a static label  
TCL_EvalStr("pack .foo2.l");  
  
// pack the label widget. It appears on the screen.  
text="button .foo2.b -text close -command {destroy .foo2}";  
TCL_EvalStr(text);  
TCL_EvalStr("pack .foo2.b");  
  
//kill the windows by program  
TCL_EvalStr("destroy .foo1");  
TCL_EvalStr("destroy .foo2");
```

```
//with one call, and in a slave interpreter
TCL_CreateSlave('TCLSlave');
TCL_EvalStr('set test ""in Slave TCL Interp""', 'TCLSlave');
TCL_GetVar('test', 'TCLSlave')

TCL_DeleteInterp('TCLSlave')

// return a result
res = TCL_EvalStr("expr 1+1")
res = TCL_EvalStr("tk_messageBox -message Hello -type okcancel")
res = TCL_EvalStr(["expr 4+5" "lsearch -all {a b c a b c} c" ; "list [list a b c]"])
```

See Also

ScilabEval , TCL_EvalFile , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar ,
TCL_UpVar

Authors

Allan CORNET

Name

TCL_ExistArray — Return %T if a tcl array exists

```
OK=TCL_ExistArray(arrayname [,interp])
```

Parameters

arrayname

character string. Contains the name of the tcl/tk array.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok

boolean. %T if arrayname exists.

Description

This routine allows to test if a tcl array exists.

Examples

```
TCL_ExistVar("A")
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
TCL_ExistVar("A")
TCL_ExistArray("A")
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar ,
TCL_UpVar , TCL_CreateSlave

Authors

Allan CORNET

Name

TCL_ExistInterp — Return %T if a tcl slave interpreter exists

```
OK=TCL_ExistInterp(interp)
```

Parameters

interp

character string parameter. Name of the slave tcl interpreter.

ok

boolean. %T if TCL interpreter exists.

Description

This routine allows to test if TCL interpreter exists.

Examples

```
TCL_ExistInterp('SlaveInterp')
TCL_CreateSlave('SlaveInterp')
TCL_ExistInterp('SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

See Also

TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_ExistVar — Return %T if a tcl variable exists

```
OK=TCL_ExistVar(varname [,interp])
```

Parameters

varname

character string. Contains the name of the tcl/tk variable.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok

boolean. %T if varname exists.

Description

This routine allows to test if a tcl variable exists.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_SetVar("aa",1)
TCL_CreateSlave('SlaveInterp');
TCL_SetVar("aa",2,'SlaveInterp')
TCL_ExistVar("aa")
TCL_GetVar("aa")
TCL_UnsetVar("aa")
TCL_GetVar("aa",'SlaveInterp')
TCL_UnsetVar("aa",'SlaveInterp')
TCL_ExistVar("aa",'SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar ,
TCL_UpVar , TCL_CreateSlave

Authors

Allan CORNET

Name

TCL_GetVar — Get a tcl/tk variable value

```
value=TCL_GetVar(Varname [,interp])
```

Parameters

varname

character string. Contains the name of the tcl/tk variable.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

value

may be a character string or a strings matrix. Contains the value of the tcl/tk variable `varname` in the interpreter `interp`.

Description

When tcl/tk support is enabled in Scilab, this routine can be used to retrieve the value of a tcl/tk variable.

Examples

```
TCL_EvalStr("toplevel .tst1");

// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");

// create an editable entry
TCL_EvalStr("set tvar foobar");

// set the entry value
TCL_EvalStr("pack .tst1.e");

// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")

// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")

//-----
a=["A", "B", "C"; "D", "E", "F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")

//-----
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")

//-----
```

```
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")

//-----
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")

//-----
// Examples with a slave interpreter
//-----
a=['AA','BB','CC';'DD','EE','FF'];
TCL_CreateSlave('SlaveInterp')
TCL_SetVar("A_slave",a,'SlaveInterp')
AfromTCL_slave=TCL_GetVar('A_slave','SlaveInterp')
TCL_DeleteInterp('SlaveInterp')

//-----
b=[66,44,11;22,33,55];
TCL_CreateSlave('SlaveInterp1')
TCL_SetVar("B_slave",b,'SlaveInterp1')
BfromTCL_slave=TCL_GetVar('B_slave','SlaveInterp1')
TCL_DeleteInterp('SlaveInterp1')

//-----
TCL_CreateSlave('SlaveInterp2')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'SlaveInterp2')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'SlaveInterp2')
TCL_DeleteInterp('SlaveInterp2')

//-----
TCL_CreateSlave('SlaveInterp3')
TCL_SetVar("ScalarTCL_slave",1.22,'SlaveInterp3')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'SlaveInterp3')
TCL_DeleteInterp('SlaveInterp3')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar ,
TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_GetVersion — get the version of the TCL/TK library at runtime.

```
TCL_GetVersion()  
ret=TCL_GetVersion('numbers')
```

Description

get the version of the TCL/TK library at runtime.

ret=TCL_GetVersion('numbers') returns a matrix with the version of the TCL/TK library at runtime.

Examples

```
TCL_GetVersion()  
TCL_GetVersion("numbers")
```

Authors

Allan CORNET

Name

TCL_SetVar — Set a tcl/tk variable value

```
TCL_SetVar(varname, value [,interp])
```

Parameters

varname

character string. Contains the name of the tcl/tk variable to set.

value

may be a character string, a scalar, a real or string matrix (m x n). Contains the value to give to the tcl/tk variable.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

This routine allows to set a variable within a tcl/tk interpreter. When tcl/tk support is enabled in scilab, this routine can be used to set up the value of a tcl/tk variable. This can be useful to change some value in the tcl/tk interpreter without having to build a tcl/tk instruction (and use `TCL_EvalStr`).

Examples

```
TCL_EvalStr("toplevel .tst1");

// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");

// create an editable entry
TCL_EvalStr("set tvar foobar");

// set the entry value
TCL_EvalStr("pack .tst1.e");

// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")

// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")

//-----
a=["A", "B", "C"; "D", "E", "F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")

//-----
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")
```

```
//-----
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")

//-----
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")

//-----
// Examples with a slave interpreter
//-----
TCL_CreateSlave('TCLSlave')
a=['AA','BB','CC';'DD','EE','FF'];
TCL_SetVar("A_slave",a,'TCLSlave')
AfromTCL_slave=TCL_GetVar('A_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----
TCL_CreateSlave('TCLSlave')
b=[66,44,11;22,33,55];
TCL_SetVar("B_slave",b,'TCLSlave')
BfromTCL_slave=TCL_GetVar('B_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----
TCL_CreateSlave('TCLSlave')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'TCLSlave')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----
TCL_CreateSlave('TCLSlave')
TCL_SetVar("ScalarTCL_slave",1.22,'TCLSlave')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_ExistVar , TCL_UnsetVar ,
TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_UnsetVar — Remove a tcl variable

```
OK=TCL_UnsetVar(varname [,interp])
```

Parameters

varname

character string. Contains the name of the tcl/tk variable to unset.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok

boolean. %T if varname was deleted.

Description

This routine allows to unset a tcl variable.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_CreateSlave('InterpSlave');
TCL_SetVar("Scilab","Good",'InterpSlave')
TCL_GetVar("Scilab",'InterpSlave')
TCL_UnsetVar("Scilab",'InterpSlave')
TCL_ExistVar("Scilab",'InterpSlave')
TCL_DeleteInterp('InterpSlave')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar ,
TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_UpVar — Make a link from a tcl source variable to a tcl destination variable

```
OK=TCL_UpVar( varname1, varname2, [interp])
```

Parameters

varname1

character string. Contains the name of the tcl source variable.

varname2

character string. Contains the name of the tcl destination variable.

interp

optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok

boolean. %T if it is ok.

Description

Make a link from a tcl source variable to a tcl destination variable.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_UpVar("Scilab","ScilabBis")
TCL_GetVar("ScilabBis")
TCL_SetVar("Scilab","NOK")
TCL_GetVar("ScilabBis")
TCL_SetVar("ScilabBis","modified")
TCL_GetVar("ScilabBis")
TCL_GetVar("Scilab")

TCL_CreateSlave('InterpBis')
TCL_SetVar("Scilab","Good",'InterpBis')
TCL_UpVar("Scilab","ScilabBis",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("Scilab","Not good",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("ScilabBis","modified again",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_GetVar("Scilab",'InterpBis')
TCL_DeleteInterp('InterpBis')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar ,
TCL_UnsetVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

browsevar — Scilab variable browser

```
browsevar()
```

Description

`browsevar` is an embedded Scilab variable browser written in TCL/TK.

`browsevar` can show all variables and function (like `who`). `browsevar` can be costumized to show all or some type of variable. It's also possible to exclude variable names.

Examples

```
browsevar();
```

Authors

Jaime Urzua

Name

`config` — Scilab general configuration.

```
config()
```

Description

`config()` allows configure scilab parameters like lines to display, stacksize, %ODEOPTIONS.

Authors

Jaime Urzua

Name

editvar — Scilab variable editor

```
editvar varname
```

Parameters

varname

variable name. The variable must exist in scilab.

Description

editvar is an embedded Scilab variable editor written in TCL/TK.

editvar can edit the following variable type: real or complex constant matrix (type 1), boolean matrix (type 4) an matrix of character strings (type 10).

Examples

```
a=rand(10,10);
editvar a;
b=['hello';'good bye'];
editvar b;
```

Authors

Jaime Urzua

Name

tk_getdir — dialog to get a directory path

```
path=tk_getdir([Title="string"])
path=tk_getdir(startdir,[Title="string"])
path=tk_getdir(startdir>windowtitle)
```

Parameters

startdir

a character string which gives the initial directory used for directory search. By default tk_getdir uses the previously selected directory.

path

is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title="string"

Optional argument which gives the title for the tk_getdir window. **Warning:** Use the new variable `Title` instead of the old variable `title`.

Description

WARNING: this function is deprecated (see `uigetdir` as a replacement of `tk_getdir`). It will be removed in Scilab 5.3.

Creates a dialog window for file selection.

Examples

```
tk_getdir()
tk_getdir("SCI/modules/")
tk_getdir(Title="Choose a directory name")
```

See Also

`uigetdir` , `uigetfile` , `file` , `fileinfo`

Name

tk_savefile — dialog to get a file path for writing

```
path=tk_savefile([Title='string'])
path=tk_savefile(file_mask,[Title='string'])
path=tk_savefile(file_mask,dir,[Title='string'])
path=tk_savefile(file_mask,dir,'string')
```

Parameters

file_mask

a character string which gives the file mask to use for file selection. `file_mask` is written with Unix convention. the default value is '*'.

dir

a character string which gives the initial directory used for file search. by default `tk_savefile` uses the previously selected directory.

path

is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title='string'

Optional argument which gives the title for the `tk_savefile` window. **Warning:** Use the new variable `Title` instead of the old variable `title`.

Description

WARNING: this function is obsolete (see `uigetfile` as a replacement of `tk_savefile`). It will be removed in Scilab 5.3.

Creates a dialog window for output file selection

Examples

```
tk_savefile()
tk_savefile('*.sci','SCI/modules/graphics/macros')
tk_savefile(Title='Choose a file name ')
```

See Also

`uigetfile` , `uigetdir` , `file` , `fileinfo`

Name

winclose — close windows created by sciGUI

```
winclose(winIds)
```

Parameters

winIds

matrix of integer greater than 0, window identificator.

Description

winclose(winIds) close windows created by sciGUI.

Examples

```
//CREATE SOME WINDOWS  
win1=waitbar('This is an example');  
win2=waitbar('HELLO!');  
winclose([win1,win2]);
```

Authors

Jaime Urzua

Name

winlist — Return the winId of current window created by sciGUI

```
winIds=winlist()
```

Parameters

winIds

matrix of integer greater than 0, window identifier.

Description

winlist() Return the winId of current window created by sciGUI.

Authors

Jaime Urzua

Part XLI. Texmacs

Name

pol2tex — convert polynomial to TeX format. This function is obsolete and will be removed in Scilab 5.3. Please use prettyprint

```
[y]=pol2tex(x)
```

Parameters

x
polynomial

y
list

Description

Latex source code for the polynomial x. (For use with `texprint`)

This function is obsolete and will be removed in Scilab 5.3. Please use `prettyprint`

Examples

```
s=poly(0, 's');  
p=s^3+2*s-5;  
pol2tex(p)
```

See Also

`prettyprint`

Name

`texprint` — TeX output of Scilab object. This function is obsolete and will be removed in Scilab 5.3. Please use `prettyprint`

```
[text]= texprint(a)
```

Parameters

`a`
Scilab object

`text`
list

Description

returns the Tex source code of the Scilab variable `a`. `a` is a matrix (constant, polynomial, rational) or a linear system (`syslin` list).

This function is obsolete and will be removed in Scilab 5.3. Please use `prettyprint`

Examples

```
s=poly(0, 's');  
texprint([1/s, s^2])
```

See Also

`prettyprint` , `pol2str`

Part XLII. Sound file handling

Name

analyze — frequency plot of a sound signal

Parameters

fmin,fmax,rate,points

scalars. default values fmin=100,fmax=1500,rate=22050,points=8192;

Description

Make a frequency plot of the signal **w** with sampling rate **rate**. The data must be at least **points** long. The maximal frequency plotted will be **fmax**, the minimal **fmin**.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
analyze(s);
```

Name

auread — load .au sound file

```
y=auread(aufile)
y=auread(aufile,ext)
[y,Fs,bits]=auread(aufile)
[y,Fs,bits]=auread(aufile,ext)
```

Parameters

aufile

string (The .au extension is appended if no extension is given)

Fs

...

[]

integer, frequency sampling in Hz.

ext

string ('size' or 'snd') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

Description

Utility function to read .au sound file. `auread(aufile)` loads a sound file specified by the string `aufile`, returning the sampled data in `y`. Amplitude values are in the range [-1,+1].

Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=auread(aufile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`auread(aufile,n)` returns the first n samples from each channel.

`auread(aufile,[n1,n2])` returns samples n1 to n2.

`auread(aufile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [samples channels].

`auread(aufile,'snd')` returns information about the sample and data as a tlist.

Examples

```
y=wavread('SCI/modules/sound/demos/chimes.wav');

// default is 8-bits mu-law
auwrite(y,TMPDIR+'/tmp.au');
y1=auread(TMPDIR+'/tmp.au');
maxi(abs(y-y1))
```

See Also

savewave , analyze , mapsound

Name

auwrite — writes .au sound file

```
auwrite(y, aufile)
auwrite(y, Fs, aufile)
auwrite(y, Fs, bits, aufile)
auwrite(y, Fs, bits, method, aufile)
```

Parameters

y
real vector or matrix with entries in $[-1,1]$.

aufile
string (The .au extension is appended if no extension is given)

Fs
integer, frequency sampling in Hz.

bits
integer, number of bits in the encoding.

method
string, 'mu' (default) or 'linear', encoding method.

Description

Utility function to save .au sound file. `auwrite(y, aufile)` writes a sound file specified by the string `aufile`. The data should be arranged with one channel per column. Amplitude values outside the range $[-1, +1]$ are ignored. Supports multi-channel data for 8-bit mu-law, and 8, 16, 32, 64 bits linear formats.

`auwrite(y, Fs, aufile)` specifies in `Fs` the sample rate of the data in Hertz.

`auwrite(y, Fs, bits, aufile)` selects the number of bits in the encoder. Allowable settings are bits in $[8, 16, 32, 64]$. `auwrite(y, Fs, bits, method, aufile)` allows selection of the encoding method, which can be either 'mu' or 'linear'. Note that bits must be 8 for 'mu' choice. The default method is 8-bits mu-law encoding.

Examples

```
A=matrix(1:6,2,3);
auwrite(A/6,22050,64,'linear',TMPDIR+'/foo.au');
B=auread(TMPDIR+'/foo.au');
maxi(abs(A- round(B*6)))
```

See Also

`auread`, `wavread`, `savewave`, `analyze`, `mapsound`

Name

beep — Produce a beep sound

```
beep();  
beep('on')  
beep('off')  
s=beep()
```

Description

`beep()` produces your computer's default beep sound.

`beep('on')` turns the beep on

`beep('off')` turns the beep off

`s=beep()` returns the current beep mode (on or off).

Authors

A.C

Name

lin2mu — linear signal to mu-law encoding

```
mu=lin2mu(y)
```

Parameters

y
real vector

mu
real vector

Description

Utility fct: converts linear signal to mu-law encoding. `mu = lin2mu(y)` converts linear audio signal amplitudes in the range $-1 \leq y \leq 1$ to mu-law in the range $0 \leq \mu \leq 255$.

See Also

mu2lin

Name

loadwave — load a sound wav file into scilab

```
x=loadwave(filename);  
[x,y]=loadwave(filename);
```

Parameters

filename

a string. The path of the wav file to be loaded

x

a matrix one line for each channel

y

vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

Description

Reads a .wav sound file into Scilab as a matrix. If y is given, it is filled with information about the samples (See the message sent by loadwave).

Examples

```
// At first we create 0.5 seconds of sound parameters.  
t=soundsec(0.5);  
  
// Then we generate the sound: a two channels sound.  
s=[sin(2*pi*440*t);sin(2*pi*350*t)];  
savewave(TMPDIR+'/foo.wav',s);  
s1=loadwave(TMPDIR+'/foo.wav');  
max(abs(s1-s))
```

See Also

savewave, analyze, mapsound

Name

mapsound — Plots a sound map

```
mapsound (w,dt,fmin,fmax,simpl,rate)
```

Parameters

dt,fmin,fmax,simpl,rate

scalars. default values dt=0.1,fmin=100,fmax=1500,simpl=1,rate=22050;

Description

Plots a sound map for a sound. It does FFT at time increments dt. **rate** is the sampling rate. **simpl** points are collected for speed reasons. **fmin** and **fmax** are used for graphic boundaries.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
mapsound(s);
```

Name

mu2lin — mu-law encoding to linear signal

```
mu=lin2mu(y)
```

Parameters

y
real vector

mu
real vector

Description

Utility fct: `y=mu2lin(mu)` converts mu-law encoded 8-bit audio signals, stored in the range $0 \leq \mu \leq 255$, to linear signal amplitude in the range $-s < y < s$ where $s = 32124/32768 \approx .9803$. The input `mu` is often obtained using `mget(...,'uc')` to read byte-encoded audio files. Translation of C program by Craig Reese: IDA/Supercomputing Research Center Joe Campbell: Department of Defense

See Also

mu2lin

Name

playsnd — sound player facility

```
[]=playsnd(y)
[]=playsnd(y,rate,bits [,command])
```

Parameters

y
A matrix. Each line describe a channel

fs
real number, sampling frequency (default value is 22050).

bits
real number, number of bits (usually 8 or 16). Unused yet.

command
Only used on Unix systems it gives the name of the command to use for playing sound (wav) files.
The default value is `play`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

Description

Plays a multi channel signal given by a Scilab matrix were sound is sampled at rate given by **rate**.

Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
playsnd(y)
```

See Also

lin2mu , wavread

Name

savewave — save data into a sound wav file.

```
savewave(filename,x [, rate , nbits]);
```

Parameters

filename

a string. The path of the output wav file

x

a mxn matrix where m is the number of channels and n the number of samples for each channel

rate

a scalar giving the sampling rate (number of sample per second) 22050 is the default value.

Description

save x into a wav sound file. you can transform other sound files into wav file with the SOX program.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(2*pi*440*t)+sin(2*pi*220*t)/2+sin(2*pi*880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(2*pi*330*t(nc/2:nc));
savewave(TMPDIR+'/foo.wav',s);
```

See Also

loadwave, analyze, mapsound

Name

sound — sound player facility

```
sound(y [,fs,bits,command)
```

Parameters

y

real vector

fs

real number, sampling frequency in sample per second (default value is 22050)

bits

real number, number of bits (unused)

command

Only used on Unix systems it gives the name of the command to use for playing sound (wav) files. The default value is `aplay`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

Description

`sound(y, fs)` plays the sound signal given by matrix *y* (with sample frequency *fs*). In fact this function is just a wrapper for `playsnd`. Values in *y* are assumed to be in the range $-1.0 \leq y \leq 1.0$. Values outside that range are truncated. The number of rows of *y* gives the number of channels. `sound(y)` plays the sound at the default sample rate of 22050 sample per second. `sound(y, fs, nbits)` plays the sound using *nbits* bits/sample if possible (it is in fact unused). Most platforms support bits=8 or 16.

Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
sound(y)
```

See Also

`playsnd`

Name

soundsec — generates n sampled seconds of time parameter

```
t=soundsec(n [, rate])
```

Parameters

n
an integer, the number of seconds to generate.

rate
an integer, the number of samples per second.

Description

generates a vector coding time from 0 to nseconds at sampled rate rate.

Examples

```
// At first we create 0.5 seconds of sound parameters.  
t=soundsec(0.5);  
  
// Then we generate the sound.  
s=sin(2*pi*440*t);  
analyze(s,200,600,22050);
```

See Also

playsnd , analyze

Name

wavread — load .wav sound file

```
y=wavread(wavfile)
y=wavread(wavfile,ext)
[y,Fs,bits]=wavread(wavfile)
[y,Fs,bits]=wavread(wavfile,ext)
```

Parameters

wavfile

string (The .wav extension is appended if no extension is given)

Fs

integer, frequency sampling in Hz (number of samples per second).

ext

string ('size') or string('info') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

Description

Utility function to read .wav sound file. `wavread(wavfile)` loads a sound file specified by the string `wavfile`, returning the sampled data in `y`. Amplitude values are in the range [-1,+1]. Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=wavread(wavfile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`wavread(wavfile,n)` returns the first n samples from each channel.

`wavread(wavfile,[n1,n2])` returns samples n1 to n2.

`wavread(wavfile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [channels samples].

`wavread(wavfile,'info')` returns information about the audio data contained in the file in place of the actual audio data, returning the vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

Examples

```
wavread("SCI/modules/sound/demos/chimes.wav","size")
[y,Fs,bits]=wavread("SCI/modules/sound/demos/chimes.wav");Fs,bits
subplot(2,1,1)
plot2d(y(1,:)) // first channel
subplot(2,1,2)
plot2d(y(2,:)) // second channel
y=wavread("SCI/modules/sound/demos/chimes.wav",[1 5]) //the first five samples
```

See Also

`auread`, `savewave`, `analyze`, `mapsound`

Name

wavwrite — writes .wav sound file

```
wavwrite(y, wavfile)
wavwrite(y, Fs, wavfile)
wavwrite(y, Fs, nbits, wavfile)
```

Parameters

y
real vector or matrix with entries in $[-1,1]$.

wavfile
string (The .wav extension is appended if no extension is given)

Fs
integer, frequency sampling in Hz. 22500 is the default value.

nbits
bit-depth 8, 16, 24, 32 bits. it describes the number of bits of information recorded for each sample.
16 is the default value.

Description

Utility function to save .wav sound file. `wavwrite(y,wavfile)` writes a sound file specified by the string `wavfile`. The data should be arranged with one channel per column. Amplitude values outside the range $[-1,+1]$ are ignored.

`wavwrite(y,Fs,wavfile)` specifies in `Fs` the sample rate of the data in Hertz.

Examples

```
A=matrix(1:6,2,3);
wavwrite(A/6,TMPDIR+'/foo.wav');
B=wavread(TMPDIR+'/foo.wav');
```

See Also

`auread`, `wavread`, `savewave`, `analyze`, `mapsound`

Part XLIII. Randlib

Name

grand — Random number generator(s)

```
Y=grand(m, n, dist_type [,p1,...,pk])
Y=grand(X, dist_type [,p1,...,pk])
Y=grand(n, dist_type [,p1,...,pk])
S=grand(action [,q1,...,ql])
```

Parameters

m, n
integers, size of the wanted matrix Y

X
a matrix whom only the dimensions (say $m \times n$) are used

dist_type
a string given the distribution which (independants) variates are to be generated ('bin', 'nor', 'poi', etc ...)

p1, ..., pk
the parameters (reals or integers) required to define completely the distribution **dist_type**

Y
the resulting $m \times n$ random matrix

action
a string given the action onto the base generator(s) ('setgen' to change the current base generator, 'getgen' to retrieve the current base generator name, 'getsd' to retrieve the state (seeds) of the current base generator, etc ...)

q1, ..., ql
the parameters (generally one string) needed to define the action

S
output of the action (generally a string or a real column vector)

Description

This function may be used to generate random numbers from various distributions. In this case you must apply one of the **three first forms** of the possible calling sequences to get an $m \times n$ matrix. The two firsts are equivalent if X is a $m \times n$ matrix, and the third form corresponds to 'multivalued' distributions (e.g. multinomial, multivariate gaussian, etc...) where a sample is a column vector (says of dim m) and you get then n such random vectors (as an $m \times n$ matrix).

The last form is used to undertake various manipulations onto the base generators like changing the base generator (since v 2.7 you may choose between several base generators), changing or retrieving its internal state (seeds), etc ... These base generators give random integers following a uniform distribution on a large integer interval (lgi), all the others distributions being gotten from it (in general via a scheme $lgi \rightarrow U([0,1]) \rightarrow$ wanted distribution).

Getting random numbers from a given distribution

beta
Y=grand(m, n, 'bet ', A, B) generates random variates from the beta distribution with parameters A and B. The density of the beta is ($0 < x < 1$):

$$x^{\frac{A-1}{x}} (1-x)^{\frac{B-1}{1-x}} / \text{beta}(A, B)$$

A and B must be reals $> 10^{-37}$. Related function(s) : cdfbet.

binomial

`Y=grand(m, n, 'bin', N, p)` generates random variates from the binomial distribution with parameters N (positive integer) and p (real in [0,1]) : number of successes in N independant Bernouilli trials with probability p of success. Related function(s) : binomial, cdfbin.

negative binomial

`Y=grand(m, n, 'nbn', N, p)` generates random variates from the negative binomial distribution with parameters N (positive integer) and p (real in (0,1)) : number of failures occurring before N successes in independant Bernouilli trials with probability p of success. Related function(s) : cdfnbn.

chisquare

`Y=grand(m, n, 'chi', Df)` generates random variates from the chisquare distribution with Df (real > 0.0) degrees of freedom. Related function(s) : cdfchi.

non central chisquare

`Y=grand(m, n, 'nch', Df, Xnon)` generates random variates from the non central chisquare distribution with Df degrees of freedom (real ≥ 1.0) and noncentrality parameter Xnonc (real ≥ 0.0). Related function(s) : cdfchn.

exponential

`Y=grand(m, n, 'exp', Av)` generates random variates from the exponential distribution with mean Av (real ≥ 0.0).

F variance ratio

`Y=grand(m, n, 'f', Dfn, Dfd)` generates random variates from the F (variance ratio) distribution with Dfn (real > 0.0) degrees of freedom in the numerator and Dfd (real > 0.0) degrees of freedom in the denominator. Related function(s) : cdf.

non central F variance ratio

`Y=grand(m, n, 'nf', Dfn, Dfd, Xnon)` generates random variates from the noncentral F (variance ratio) distribution with Dfn (real ≥ 1) degrees of freedom in the numerator, and Dfd (real > 0) degrees of freedom in the denominator, and noncentrality parameter Xnonc (real ≥ 0). Related function(s) : cdfnc.

gamma

`Y=grand(m, n, 'gam', shape, scale)` generates random variates from the gamma distribution with parameters shape (real > 0) and scale (real > 0). The density of the gamma is :

$$\frac{\text{shape}^{\text{shape}}}{\text{scale}^{\text{shape}}} \frac{x^{\text{shape}-1} e^{-x/\text{scale}}}{\text{gamma}(\text{shape})}$$

Related function(s) : gamma, cdfgam.

Gauss Laplace (normal)

`Y=grand(m, n, 'nor', Av, Sd)` generates random variates from the normal distribution with mean Av (real) and standard deviation Sd (real ≥ 0). Related function(s) : cdfnor.

multivariate gaussian (multivariate normal)

`Y=grand(n, 'mn', Mean, Cov)` generates n multivariate normal random variates ; Mean must be a m x 1 matrix and Cov a m x m symmetric positive definite matrix (Y is then a m x n matrix).

geometric

`Y=grand(m,n,'geom',p)` generates random variates from the geometric distribution with parameter `p` : number of Bernoulli trials (with probability success of `p`) until a success is met. `p` must be in `[pmin,1]` (with `pmin = 1.3 10^(-307)`).

`Y` contains positive real numbers with integer values, which are the "number of trials to get a success".

markov

`Y=grand(n,'markov',P,x0)` generate `n` successive states of a Markov chain described by the transition matrix `P`. Initial state is given by `x0`. If `x0` is a matrix of size `m=size(x0,'*')` then `Y` is a matrix of size `m x n`. `Y(i,:)` is the sample path obtained from initial state `x0(i)`.

multinomial

`Y=grand(n,'mul',nb,P)` generates `n` observations from the Multinomial distribution : class `nb` events in `m` categories (put `nb` "balls" in `m` "boxes"). `P(i)` is the probability that an event will be classified into category `i`. `P` the vector of probabilities is of size `m-1` (the probability of category `m` being `1-sum(P)`). `Y` is of size `m x n`, each column `Y(:,j)` being an observation from multinomial distribution and `Y(i,j)` the number of events falling in category `i` (for the `j`th observation) (`sum(Y(:,j)) = nb`).

Poisson

`Y=grand(m,n,'poi',mu)` generates random variates from the Poisson distribution with mean `mu` (real ≥ 0.0). Related function(s) : `cdfpoi`.

random permutations

`Y=grand(n,'prm',vect)` generate `n` random permutations of the column vector (`m x 1`) `vect`.

uniform (def)

`Y=grand(m,n,'def')` generates random variates from the uniform distribution over `[0,1]` (1 is never return).

uniform (unf)

`Y=grand(m,n,'unf',Low,High)` generates random reals uniformly distributed in `[Low,High]`.

uniform (uin)

`Y=grand(m,n,'uin',Low,High)` generates random integers uniformly distributed between `Low` and `High` (included). `High` and `Low` must be integers such that `(High-Low+1) < 2,147,483,561`.

uniform (lgi)

`Y=grand(m,n,'lgi')` returns the basic output of the current generator : random integers following a uniform distribution over :

- `[0, 2^32 - 1]` for `mt`, `kiss` and `fsultra`
- `[0, 2147483561]` for `clcg2`
- `[0, 2^31 - 2]` for `clcg4`
- `[0, 2^31 - 1]` for `urand`.

Set/get the current generator and its state

Since Scilab-2.7 you have the possibility to choose between different base generators (which give random integers following the 'lgi' distribution, the others being gotten from it) :

mt

the Mersenne-Twister of M. Matsumoto and T. Nishimura, period about 2^{19937} , state given by an array of 624 integers (plus an index onto this array); this is the default generator.

kiss

The Keep It Simple Stupid of G. Marsaglia, period about 2^{123} , state given by 4 integers.

clcg2

a Combined 2 Linear Congruential Generator of P. L'Ecuyer, period about 2^{61} , state given by 2 integers ; this was the only generator previously used by grand (but slightly modified)

clcg4

a Combined 4 Linear Congruential Generator of P. L'Ecuyer, period about 2^{121} , state given by 4 integers ; this one is splitted in 101 different virtual (non over-lapping) generators which may be useful for different tasks (see 'Actions specific to clcg4' and 'Test example for clcg4').

urand

the generator used by the scilab function rand, state given by 1 integer, period of 2^{31} (based on theory and suggestions given in d.e. knuth (1969), vol 2. State). This is the faster of this list but a little outdated (don't use it for serious simulations).

fsultra

a Subtract-with-Borrow generator mixing with a congruential generator of Arif Zaman and George Marsaglia, period more than 10^{356} , state given by an array of 37 integers (plus an index onto this array, a flag (0 or 1) and another integer).

The differents actions common to all the generators, are:

action= 'getgen'

`S=grand('getgen')` returns the current base generator (S is a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra'.

action= 'setgen'

`grand('setgen', gen)` sets the current base generator to be gen a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra' (notes that this call returns the new current generator, ie gen).

action= 'getsd'

`S=grand('getsd')` gets the current state (the current seeds) of the current base generator ; S is given as a column vector (of integers) of dimension 625 for mt (the first being an index in [1, 624]), 4 for kiss, 2 for clcg2, 40 for fsultra, 4 for clcg4 (for this last one you get the current state of the current virtual generator) and 1 for urand.

action= 'setsd'

`grand('setsd', S), grand('setsd', s1[, s2, s3, s4])` sets the state of the current base generator (the new seeds) :

for mt

S is a vector of integers of dim 625 (the first component is an index and must be in [1, 624], the 624 last ones must be in $[0, 2^{32}]$ (but must not be all zeros) ; a simpler initialisation may be done with only one integer s1 (s1 must be in $[0, 2^{32}]$;

for kiss

4 integers s1, s2, s3, s4 in $[0, 2^{32}]$ must be provided ;

for clcg2

2 integers s1 in [1, 2147483562] and s2 in [1, 2147483398] must be given ;

for clcg4

4 integers s1 in [1, 2147483646], s2 in [1, 2147483542], s3 in [1, 2147483422], s4 in [1, 2147483322] are required ; CAUTION : with clcg4 you set the seeds of the current virtual generator but you may lost the synchronisation between

this one and the others virtuals generators (ie the sequence generated is not warranty to be non over-lapping with a sequence generated by another virtual generator)=> use instead the 'setall' option.

for urand

1 integer s1 in $[0, 2^{31}[$ must be given.

for fsultra

S is a vector of integers of dim 40 (the first component is an index and must be in $[0, 37]$, the 2d component is a flag (0 or 1), the 3d an integer in $[1, 2^{32}[$ and the 37 others integers in $[0, 2^{32}[$); a simpler (and recommended) initialisation may be done with two integers s1 and s2 in $[0, 2^{32}[$.

action= 'phr2sd'

Sd=grand('phr2sd', phrase) given a phrase (character string) generates a 1×2 vector Sd which may be used as seeds to change the state of a base generator (initially suited for clcg2).

Options specific to clcg4

The clcg4 generator may be used as the others generators but it offers the advantage to be splitted in several (101) virtual generators with non over-lapping sequences (when you use a classic generator you may change the initial state (seeds) in order to get another sequence but you are not warranty to get a complete different one). Each virtual generator corresponds to a sequence of 2^{72} values which is further split into $V=2^{31}$ segments (or blocks) of length $W=2^{41}$. For a given virtual generator you have the possibility to return at the beginning of the sequence or at the beginning of the current segment or to go directly at the next segment. You may also change the initial state (seed) of the generator 0 with the 'setall' option which then change also the initial state of the other virtual generators so as to get synchronisation (ie in function of the new initial state of gen 0 the initial state of gen 1 . . 100 are recomputed so as to get 101 non over-lapping sequences).

action= 'setcgn'

grand('setcgn', G) sets the current virtual generator for clcg4 (when clcg4 is set, this is the virtual (clcg4) generator number G which is used); the virtual clcg4 generators are numbered from 0, 1, . . , 100 (and so G must be an integer in $[0, 100]$); by default the current virtual generator is 0.

action= 'getcgn'

S=grand('getcgn') returns the number of the current virtual clcg4 generator.

action= 'initgn'

grand('initgn', I) reinitializes the state of the current virtual generator

I = -1

sets the state to its initial seed

I = 0

sets the state to its last (previous) seed (i.e. to the beginning of the current segment)

I = 1

sets the state to a new seed W values from its last seed (i.e. to the beginning of the next segment) and resets the current segment parameters.

action= 'setall'

grand('setall', s1, s2, s3, s4) sets the initial state of generator 0 to s1, s2, s3, s4. The initial seeds of the other generators are set accordingly to have synchronisation. For constraints on s1, s2, s3, s4 see the 'setsd' action.

action= 'advnst'

grand('advnst', K) advances the state of the current generator by 2^K values and resets the initial seed to that value.

Test example for clcg4

An example of the need of the splitting capabilities of clcg4 is as follows. Two statistical techniques are being compared on data of different sizes. The first technique uses bootstrapping and is thought to be as accurate using less data than the second method which employs only brute force. For the first method, a data set of size uniformly distributed between 25 and 50 will be generated. Then the data set of the specified size will be generated and analyzed. The second method will choose a data set size between 100 and 200, generate the data and analyze it. This process will be repeated 1000 times. For variance reduction, we want the random numbers used in the two methods to be the same for each of the 1000 comparisons. But method two will use more random numbers than method one and without this package, synchronization might be difficult. With clcg4, it is a snap. Use generator 0 to obtain the sample size for method one and generator 1 to obtain the data. Then reset the state to the beginning of the current block and do the same for the second method. This assures that the initial data for method two is that used by method one. When both have concluded, advance the block for both generators.

See Also

rand

Authors

randlib

The codes to generate sequences following other distributions than def, unf, lgi, uin and geom are from "Library of Fortran Routines for Random Number Generation", by Barry W. Brown and James Lovato, Department of Biomathematics, The University of Texas, Houston.

mt

The code is the mt19937int.c by M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January, pp.3-30 1998.

kiss

The code was given by G. Marsaglia at the end of a thread concerning RNG in C in several news-groups (whom sci.math.num-analysis) "My offer of RNG's for C was an invitation to dance..." only kiss have been included in Scilab (kiss is made of a combinaison of several others which are not visible at the scilab level).

clcg2

The method is from P. L'Ecuyer but the C code is provided at the Luc Devroye home page (<http://cgm.cs.mcgill.ca/~luc/rng.html>).

clcg4

The code is from P. L'Ecuyer and Terry H. Andres and provided at the P. L'Ecuyer home page (<http://www.iro.umontreal.ca/~lecuyer/papers.html>) A paper is also provided and this new package is the logical successor of an old 's one from : P. L'Ecuyer and S. Cote. Implementing a Random Number Package with Splitting Facilities. ACM Transactions on Mathematical Software 17:1,pp 98-111.

fsultra

code from Arif Zaman (arif@stat.fsu.edu) and George Marsaglia (geo@stat.fsu.edu)

scilab packaging

By Jean-Philippe Chancelier and Bruno Pincon

Part XLIV. Development tools

Name

bench_run — Launch benchmark tests

```
bench_run()  
bench_run(module[, test_name[, options]])
```

Parameters

module

a vector of string. It can be the name of a module or the absolute path of a toolbox.

test_name

a vector of string

options

a vector of string

- list : list of the benchmark tests available in a module
- help : displays some examples of use in the Scilab console
- nb_run=value : repeat the benchmark test value times

Description

Search for .tst files in benchmark test library execute them, and display a report about execution time. The .tst files are searched in directories SCI+"/modules/*/tests/benchmark".

Special tags may be inserted in the .tst file, which help to control the processing of the corresponding test. These tags are expected to be found in Scilab comments.

These are the available tags :

- <-- BENCH NB RUN : 10 --> This test will be repeated 10 times.
- <-- BENCH START --> <-- BENCH END --> The interesting part of the benchmark must be enclosed by these tags.

Examples

Some simple examples of invocation of bench_run

```
// Launch all tests  
bench_run();  
bench_run([]);  
bench_run([], []);  
  
// Test one or several module  
bench_run('core');  
bench_run('core', []);  
bench_run(['core', 'string']);  
  
// Launch one or several test in a specified module  
bench_run('core', ['trycatch', 'opcode']);
```

```
// With options
bench_run([],[], 'list');
bench_run([],[], 'help');
bench_run([],[], 'nb_run=2000');
```

An example of a benchmark file. This file corresponds to the file SCI/modules/linear_algebra/tests/benchmarks/bench_chol.tst.

```
// =====
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) 2007-2008 - INRIA
//
// This file is distributed under the same license as the Scilab package.
// =====

//=====
// Benchmark for chol function
//=====

// <-- BENCH NB RUN : 10 -->

stacksize(30000000);

a = 0;
b = 0;
a = rand(900, 900, 'n');
a = a'*a;

// <-- BENCH START -->
b = chol(a);
// <-- BENCH END -->
```

The result of the test

```
-->bench_run('linear_algebra','bench_chol')

          Boucle For ..... 0.19 µs [
001/001 - [linear_algebra] bench_chol ..... 151576.71 µs [
```

See Also

test_run

Authors

Yann Collette

Name

tbx_build_cleaner — Generate a cleaner.sce script (toolbox compilation process)

```
tbx_build_loader(toolbox_name)
tbx_build_loader(toolbox_name, toolbox_path)
```

Parameters

toolbox_name

Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

toolbox_path

Root directory of toolbox sources ; the script will be generated here (default: current directory).

Examples

```
// Recommended usage
tbx_build_cleaner("mytoolbox", get_absolute_file_path('builder.sce'))
```

Authors

Allan CORNET

Name

tbx_build_gateway — Build a gateway (toolbox compilation process)

```
tbx_build_gateway(libname, names, files, [gateway_path [, libs [, ldflags
```

Parameters

libname

a character string, the generic name of the library without path and extension.

names

2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files

string matrix giving objects files needed for shared library creation

gateway_path

Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway_(lang).sce script (which should be the script calling this function). Default is current directory.

libs

string matrix giving extra libraries needed for shared library creation

ldflags,cflags,fflags

character strings to provide options for the loader, the C compiler and the Fortran compiler.

cc

character string. The name of or path to the compiler.

makename

character string. The path of the Makefile file without extension.

This parameter is useless since Scilab 5.0. Default value to use: [].

A warning will be displayed in Scilab 5.3 if you use another value than the default.

ismex

Internal variable to specify if we are working with mex or not.

Examples

```
// Recommended usage
tbx_build_gateway('mytoolbox', ['c_sum','sci_csum';'c_sub','sci_csub'], ['sci_c
    get_absolute_file_path('builder_gateway_c.sce'), ..
    ['../../src/c/libcsum']]);
```

See Also

ilib_build

Authors

SL

Name

tbx_build_gateway_clean — Generate a cleaner_gateway.sce script (toolbox compilation process)

```
tbx_build_gateway_loader(langs)
tbx_build_gateway_loader(langs, gateway_path)
```

Parameters

langs

Languages of source files.

gateway_path

Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_gateway_clean(['c', 'fortran'], get_absolute_file_path('builder_gateway.sce'))
```

Authors

Allan CORNET

Name

tbx_build_gateway_loader — Generate a loader_gateway.sce script (toolbox compilation process)

```
tbx_build_gateway_loader(langs)
tbx_build_gateway_loader(langs, gateway_path)
```

Parameters

langs

Languages of source files.

gateway_path

Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_gateway_loader(['c', 'fortran'], get_absolute_file_path('builder_gate
```

Authors

SL

Name

tbx_build_help — Generate help files (toolbox compilation process)

```
tbx_build_help(title)
tbx_build_help(title, help_path)
```

Parameters

title

Title of the chapter.

help_path

Directory where the files will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_help("Toolbox Example", get_absolute_file_path('build_help.sce'))
```

Authors

SL

Name

tbx_build_help_loader — Generate a addchapter.sce script (toolbox compilation process)

```
tbx_build_help_loader(title)
tbx_build_help_loader(title, help_path)
```

Parameters

title

Title of the chapter.

help_path

Directory where the script will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_help_loader("Toolbox Example", get_absolute_file_path('build_help.sce'))
```

Authors

SL

Name

tbx_build_loader — Generate a loader.sce script (toolbox compilation process)

```
tbx_build_loader(toolbox_name)
tbx_build_loader(toolbox_name, toolbox_path)
```

Parameters

toolbox_name

Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

toolbox_path

Root directory of toolbox sources ; the script will be generated here (default: current directory).

Examples

```
// Recommended usage
tbx_build_loader("mytoolbox", get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_build_macros — Compile macros (toolbox compilation process)

```
tbx_build_macros(toolbox_name)
tbx_build_macros(toolbox_name, macros_path)
```

Parameters

toolbox_name

Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

macros_path

Directory where the macros files can be found and where the compiled macros will be placed into ; in a normal toolbox it should be the directory containing the buildmacros.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_macros("toolbox_example", get_absolute_file_path('buildmacros.sce'))
```

Authors

SL

Name

tbx_build_src — Build sources (toolbox compilation process)

```
tbx_build_src(names, files, flag, [src_path [, libs [, ldflags [, cflags
```

Parameters

names

a string matrix giving the entry names which are to be linked.

files

string matrix giving objects files needed for shared library creation

flag

a string flag ("c" or "f") for C or Fortran entry points.

src_path

Path to the source files ; in a normal toolbox it should be the directory containing the builder_src_(lang).sce script (which should be the script calling this function). Default is current directory.

libs

string matrix giving extra libraries needed for shared library creation

ldflags

optional character string. It can be used to add specific linker options in the generated Makefile. Default value is "

cflags

optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

fflags

optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

cc

optional character string. It can be used to specify a C compiler. Default value is "

libname

optional character string. The name of the generated shared library (default value is "", and in this case the name is derived from names(1)).

loadername

character string. The pathname of the loader file (default value is loader.sce).

makename

character string. The pathname of the Makefile file without extension.

This parameter is useless since Scilab 5.0. Default value to use: [].

A warning will be displayed in Scilab 5.3 if you use another value than the default.

Examples

```
// Recommended usage
tbx_build_src(['csum','csub'], ['csum.c','csub.c'], 'c', ..
               get_absolute_file_path('builder.c.sce'));
```

See Also

[ilib_for_link](#)

Authors

SL

Name

tbx_builder_gateway — Run builder_gateway.sce script if it exists (toolbox compilation process)

```
tbx_builder_gateway(toolbox_path)
```

Parameters

toolbox_path

Root directory of toolbox sources ; builder_gateway.sce script will be searched in the sci_gateway subdirectory of this directory.

Examples

```
// Recommended usage  
tbx_builder_gateway(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

`tbx_builder_gateway_lang` — Run `builder_gateway_(language).sce` script if it exists (toolbox compilation process)

```
tbx_builder_gateway_lang(lang)
tbx_builder_gateway_lang(lang, gw_path)
```

Parameters

`lang`

Language of sources files ; the `builder_gateway_(lang).sce` script will be searched in the subdirectory `lang` (e.g. `fortran`) of the `gw_path` directory.

`gw_path`

Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the `builder_gateway.sce` script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_gateway_lang("fortran", get_absolute_file_path('builder_gateway.sce'))
```

Authors

SL

Name

tbx_builder_help — Run builder_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help(toolbox_path)
```

Parameters

toolbox_path

Root directory of toolbox sources ; builder_help.sce script will be searched in the help subdirectory of this directory.

Examples

```
// Recommended usage  
tbx_builder_help(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_help_lang — Run build_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help_lang(lang)
tbx_builder_help_lang(lang, help_path)
```

Parameters

lang

Language of help files to use ; the build_help.sce script will be searched in the subdirectory lang (e.g. en_US) of the help_path directory

help_path

Path to help directory ; in a normal toolbox it should be the directory containing the builder_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_help_lang("en_US", get_absolute_file_path('builder_help.sce'))
```

Authors

SL

Name

tbx_builder_macros — Run buildmacros.sce script if it exists (toolbox compilation process)

```
tbx_builder_macros(toolbox_path)
```

Parameters

toolbox_path

Root directory of toolbox sources ; buildmacros.sce script will be searched in the macros subdirectory of this directory.

Examples

```
// Recommended usage  
tbx_builder_macros(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_src — Run builder_src.sce script if it exists (toolbox compilation process)

```
tbx_builder_src(toolbox_path)
```

Parameters

toolbox_path

Root directory of toolbox sources ; builder_src.sce script will be searched in the src subdirectory of this directory.

Examples

```
// Recommended usage  
tbx_builder_src(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_src_lang — Run builder_(language).sce script if it exists (toolbox compilation process)

```
tbx_builder_src_lang(lang)
tbx_builder_src_lang(lang, src_path)
```

Parameters

lang

Language of sources files ; the builder_(lang).sce script will be searched in the subdirectory lang (e.g. fortran) of the src_path directory.

src_path

Path to the sources ; in a normal toolbox it should be the directory containing the builder_src.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_src_lang("fortran", get_absolute_file_path('builder_src.sce'))
```

Authors

SL

Name

test_run — Launch tests

```
N = test_run()  
N = test_run(module[,test_name[,options]])
```

Parameters

module

a vector of string. It can be the name of a module or the absolute path of a toolbox.

test_name

a vector of string

options

a vector of string

- no_check_ref : does not check if the .dia and .dia.ref are equal
- no_check_error_output
- create_ref : create the .dia.ref file and does not check if the .dia and .dia.ref are equal
- list : does not perform the tests but displays a list of available tests
- help : display some examples about how to use this command
- mode_nw : add the "-nw" option to the launch
- mode_nwni : add the "-nwni" option to the launch
- nonreg_test : runs only the non-regression tests, skipping unit tests
- unit_test : runs only the unit tests, skipping non-regression tests
- skip_tests
- enable_lt : Enable long-time execution tests

Description

Search for .tst files in the unit test and non-regression test library execute them, and display a report about success of failures. The .tst files are searched in directories SCI+"/modules/*/tests/unit_tests" and SCI+"/modules/*/tests/nonreg_tests". Whenever a test is executed, a .dia file is generated which contains the full list of commands executed along with message which appears in the console. When the script is done, the .dia file is compared with the .dia.ref file which is expected to be in the same directory as the .tst file. If the two file are different, the test fails.

Special tags may be inserted in the .tst file, which help to control the processing of the corresponding test. These tags are expected to be found in Scilab comments.

These are the available tags :

- <-- INTERACTIVE TEST --> This test will be skipped because it is interactive.
- <-- LONG TIME EXECUTION --> This test will be skipped because it needs long-time duration. To enable the test, call test_run with the following option: "enable_lt"
- <-- NOT FIXED --> This test will be skipped because it is a known, but unfixed bug.
- <-- TEST WITH GRAPHIC --> This test will not be executed if the option "mode_nwni" is used.

- <-- NO TRY CATCH -->
- <-- NO CHECK ERROR OUTPUT --> The error output file is not checked
- <-- NO CHECK REF --> The .dia and the .dia.ref files are not compared.
- <-- ENGLISH IMPOSED --> This test will be executed with the -l en_US option.
- <-- FRENCH IMPOSED --> This test will be executed with the -l fr_FR option.
- <-- JVM NOT MANDATORY --> This test will be executed with the nwni mode by default.
- <-- WINDOWS ONLY --> If the operating system isn't Windows, the test is skipped.
- <-- UNIX ONLY --> If the operating system isn't an unix OS, the test is skipped.
- <-- LINUX ONLY --> If the operating system isn't Linux, the test is skipped.
- <-- MACOSX ONLY --> If the operating system isn't MacOSX, the test is skipped.

Each test is executed in a separated process, created with the "host" command. That enables the current command to continue, even if the test as created an unstable environment. It also enables the tests to be independent from one another.

Platform-specific tests

It may happen that the output of a test depends on the platform on which it is executed. In this case, the .ref file cannot be correct for all platforms and unit tests may fail for some platform. In this case, we can create a default .ref and create additionnal .ref file for each platform.

The various platform-specific .ref files must have one of the following extensions.

- .unix.dia.ref for Unix platform,
- .linux.dia.ref for Linux platform,
- .win.dia.ref for Windows platform,
- .macosx.dia.ref for Mac OS X platform.

The algorithm is the following. First, the .ref is considered. If this file does not exist, the platform-specific .ref file is examined depending on the current platform.

- on windows platforms: .win.dia.ref,
- on Max OS X platforms: .unix.dia.ref, .macosx.dia.ref,
- on Linux platforms: .unix.dia.ref, .linux.dia.ref.

Examples

```
// Launch all tests
// =====

test_run();
test_run([]);
test_run([], []);

// Test one or several module
// =====

// Test one module
```

```
test_run('time');

// Test several modules
test_run(['time','string']);

// Test a submodule
test_run(['optimization|neldermead']);

// Refer to a module by its path
test_run(SCI+'/modules/core');

// Launch a specific test
// =====

// One specific test
test_run('time','datenum');

// Several tests
test_run('time',['datenum','calendar']);

// Skip some tests
// =====

test_run('time',['datenum','calendar'],'skip_tests');

// Options
// =====

// does not check if the .dia and .dia.ref are equal
test_run('time','datenum','no_check_ref');

// Create the .dia.ref file and does not check if the .dia and .dia.ref are equal
test_run([],[],'create_ref');

// Does not perform the tests but displays a list of available tests
test_run([],[],'list');

// Display some examples about how to use this command
test_run([],[],'help');

// Runs only the non-regression tests, skipping unit tests
test_run([],[],'nonreg_test');

// Runs only the unit tests, skipping non-regression tests
test_run([],[],'unit_test');

// Do not check the error output (std err)
test_run('boolean','bug_2799','no_check_error_output');

// Combine several options
test_run([],[],['no_check_ref','mode_nw']);
```

Internal Design

The tests are performed in the temporary directory, not in the directory which originally contain the tests files. The .tst file is copied into the temporary directory, the test is performed and the .dia.ref is copied back into the original location.

The .tst script is not run as is. Instead, a header and a footer are inserted at the beginning and at the end of the .tst at the time the script is copied into the temporary directory. The role of this modification is to redirect the output messages into the .dia file, so that the user can have a log file once the test is performed.

Authors

Pierre Maréchal
Michael Baudin

Part XLV. Demo Tools

Nom

demo_begin — begin a demonstration

```
demo_begin()
```

Description

The function `demo_begin` is used to begin a demonstration. It sets the script and the values in mode of non display on the console, save the environment variables in a temporary file and save the width of the console. This function shall be used with the function `demo_end`.

Voir Aussi

`demo_end` , `demo_run` , `demo_message`

Auteurs

G.H

Name

demo_choose — create a dialog box for the choice of options

```
num = demo_choose(fil)
```

Description

The function demo_choose create a dialog box for the choice of options. It takes as argument the binary file 'fil'. This file is built by a .sce file written like below. It shall contain the variables 'choice', an array of text within bracket (the different options), and 'titl', the title of the dialog box. After that, the .sce file shall save both variables in the binary file in argument. Before the use of demo_choose, the .sce file shall be executed. The function demo_choose returns the number of line chosen in the options array.

Examples

```
exec('SCI/demos/control/pid/pid_ch_2.sce');
[n]=demo_choose('SCI/demos/control/pid/pid_ch_2.bin');
select n
    case 0
        break
    case 1
        mode(1)
    case 2
        mode(-1)
end
```

See Also

x_choose , demo_mdialog

Authors

G.H

Name

demo_compiler — test the presence of a compiler

```
status = demo_compiler()
```

Description

The function `demo_compiler` asks the computer if it owns a compiler C or not. It returns a boolean indicating whether the compiler exists or not.

Examples

```
select num,
  case 0
    return;
  case 2 then
    st = demo_compiler(); //The compiler will be used
    if (st==%t) then
      mode(0);
      wheel_build_and_load()
    end
  case 1 then // A precomputed value
    x=read(path+'/x.wrt',8,301);
    wheelg=wheelgs;
    show(x);
  end
```

See Also

`findmsvccompiler`

Authors

G.H

Nom

demo_end — completes a demonstration

```
demo_end( )
```

Description

The function `demo_end()` is used to complete a demonstration. It shall be used complementarily with the function `demo_begin`. It resets the state of the environment as it was before to use the function `demo_begin` : width of the console and the variables value.

Voir Aussi

`demo_begin` , `demo_run` , `demo_message`

Auteurs

G.H

Name

demo_file_choice — choose and executes an item within a list

```
demo_file_choice(path,ch)
```

Description

The function demo_file_choice choose and executes an item chosen in the 'demolist' variable, that shall be written above. The variable 'demolist' is a text matrix whose first column contains names of items displayed in an options window and whose second column contains the names of the files that will be executed. The title of the options window is 'Choose a demo'. The 'path' variable is the access to the files called in the second column. The 'ch' variable allows to avoid the special cases 'root' and 'anim' that are used in peculiar demonstrations of Scilab. Then you have to enter another word than 'root' or 'ch', 'no' for example. If you choose to cancel the options window, the console is put back to its previous width.

Examples

```
demolist=['n-pendulum','npend/npend_gateway.sce';  
         'Wheel simulation','wheel2/wheel_gateway.sce';  
         'Bike Simulation','bike/bike_gateway.sce';  
         'ODE'S','ode/ode.dem';  
         'DAE'S','dae/dae.dem']  
  
demo_file_choice('SCI/demos/simulation/','no');
```

See Also

demo_function_choice

Authors

G.H

Name

demo_function_choice — choose and execute an item within a list

```
demo_function_choice()
```

Description

The function `demo_function_choice` choose and execute an item chosen in the variable 'demolist' that shall appear above. The variable 'demolist' is a text matrix whose first column contains item names displayed in an options window and whose second column contains the function that will be called. The title of the options window is 'Choose a demo'. If the options window is cancelled, the console is put back to its previous width.

Examples

```
demolist=[
'Simulation of a binomial random variable','clf();BinomialT();';
'Simulation of a discrete random variable','clf();RndDiscT();';
'Simulation of a geometric random variable','clf();GeomT(1000);';
'Simulation of a Poisson random variable','clf();PoissonT() ';
'Simulation of an exponential random variable','clf();ExpT();';
'Simulation of a Weibull random variable','clf();WeibullT();';
'Simulation of an hyper geometric random variable','clf();HyperGeomT();';
'Simulation of an Erlang random variable','clf();ErlangT();'];

demo_function_choice();
```

See Also

`demo_file_choice`

Authors

G.H

Name

demo_mdialog — create a dialog box

```
resp = demo_mdialog(fil)
```

Description

The function demo_mdialog create a dialog box. It takes as argument a binary file. This file is built by a .sce file written like below. It shall contain the variables 'titl', the title a the dialog box, 'namevar', the name of the fields to fill, and 'value', the values written by default. After this, these three variables shall be saved in the binary file. The use of demo_mdialog shall be preceded by the execution of the .sce associated. The function demo_mdialog returns 'resp', the values chosen by the user.

Examples

```
exec('SCI/demos/control/pid/pid_dial_4.sce');  
[defv]=demo_mdialog('SCI/demos/control/pid/pid_dial_4.bin');  
  
if defv==[] then warning('Demo stops!');return;end
```

See Also

demo_choose , x_mdialog , x_dialog

Authors

G.H

Name

demo_message — display a message

```
demo_message(fil)
```

Description

The function demo_message displays the text message within the file 'fil' given in argument.

Examples

```
demo_message('SCI/demos/control/pid/pid_3.sce');
```

See Also

demo_run , messagebox , demo_begin , demo_end

Authors

G.H

Name

demo_run — script file execution

```
demo_run(fil)
```

Description

The function demo_run executes a script in the file 'fil' given in argument.

See Also

exec , demo_message , demo_begin , demo_end

Authors

G.H

Part XLVI. Spreadsheet

Name

excel2sci — obsolete see read_csv

See Also

read_csv , write_csv

Name

read_csv — Read comma-separated value file

```
M = read_csv(fname [, sep])
```

Parameters

fname

a character string. The file path.

sep

a character string. Field separator used, default value is ","

M

a matrix of strings.

Description

Given an ascii file with delimited fields, for instance created by a spreadsheet using "Text and comma" format, read_csv(fname) returns the corresponding Scilab matrix of strings. Use read_csv(fname,sep) for another choice of separator.

Note: You may eval all or part of M using function evstr in order to convert string variables into numeric variables.

Examples

```
// create a file with some data separated with tab
A = 1:50;
mputl(strcat(string(A),ascii(9)), TMPDIR + '/foo.csv');

// read csv file
B = read_csv(TMPDIR + '/foo.csv');

// eval B
C = evstr(B);

// compares original data and result
and(A == C)
```

See Also

write_csv, evstr

Authors

Allan CORNET

Name

readxls — reads an Excel file

```
sheets = readxls(file_path)
```

Parameters

file_path

a character string: the path of the Excel file.

sheets

an `mlist` of type `xls`, with one field named `sheets`

Description

Given an Excel file path this function returns an `mlist` data structure of type `xls`, with one field named `sheets`. The `sheets` field itself contains a list of sheet data structure.

`sheet=mlist(['xlssheet', 'name', 'text', 'value'], sheetname, Text, Value)`
where `sheetname` is a character string containing the name of the sheet, `Text` is a matrix of string which contains the cell's strings and `Value` is a matrix of numbers which contains the cell's values.

Warning only BIFF8 Excel files (last Excel file version) are handled

Examples

```
Sheets = readxls('SCI/modules/spreadsheet/demos/xls/t1.xls')
// some basic operations on Sheets
typeof(Sheets)
s1=Sheets(1) //get the first sheet
typeof(s1)
s1.value //get the first sheet value field
s1.text //get the first sheet text field
s1(2,:) //get the 2 row of the sheet
typeof(s1(2,:))

editvar s1
```

See Also

`xls_open` , `xls_read`

Authors

Pierrick Mode
INRIA

Serge Steer
INRIA

Used Functions

This function is based on the Scilab functions `xls_open` and `xls_read`

Name

write_csv — Read comma-separated value file

```
write_csv(M, filename)
```

Parameters

filename

a character string. The file path.

M

a matrix of strings.

Description

write_csv(M, filename) writes matrix M into filename as comma-separated values. The filename input is a string.

Examples

```
// save a matrix as csv file format
A = [1:10] * 0.1;
write_csv(A, TMPDIR + '/datas.csv');

// read as text
mgetl(TMPDIR + '/datas.csv')

r = read_csv(TMPDIR + '/datas.csv',ascii(9));
r = strsubst(r,',','.',');
evstr(r)
```

See Also

read_csv, evstr, mgetl

Authors

Allan CORNET

Name

`xls_open` — Open an Excel file for reading

```
[fd, SST, Sheetnames, Sheetpos] = xls_open(file_path)
```

Parameters

`file_path`

a character string: the path of the Excel file.

`fd`

a number, the logical unit on the Excel stream.

`SST`

A vector of all character strings which appear in the Excel sheets.

`Sheetnames`

a vector of strings: the sheet names.

`Sheetpos`

a vector of numbers: the position of the beginning of sheets in the Excel stream.

Description

This function first analyzes the ole2 data structure associated with the given file to extract the Excel stream which is included in. After that the Excel stream is saved in the `TMDIR` directory and opened. The `fd` logical unit points to this temporary file. Then the first sheet in this stream is read to get the global information like number of sheets, sheet names `Sheetnames`, sheet addresses within the stream `Sheetpos` and the `SST` which contains all the strings used in the following sheets.

The `fd` and `Sheetpos` data have to be passed to `xls_read` to read the data sheets.

The `readxls` function can be used to read all an Excel file in one function with a single function call.

Warning only BIFF8 Excel files (last Excel file version (2003)) are handled

Examples

```
//Decode ole file, extract and open Excel stream
[fd, SST, Sheetnames, Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/Test.
//Read first data sheet
[Value, TextInd] = xls_read(fd, Sheetpos(1))
//close the spreadsheet stream
mclose(fd)
```

See Also

`xls_read` , `readxls`

Authors

Pierrick Mode
INRIA

Serge Steer
INRIA

Bibliography

This function is based on the Microsoft ole2 file documentation (<http://chicago.sourceforge.net/devel/docs/ole/>) and on Excel stream description from OpenOffice (<http://sc.openoffice.org/spreadsheetfileformat.pdf>).

Used Functions

The ripole-0.1.4 procedure (<http://www.pldaniels.com/ripole>) is used to extract the spreadsheet stream out of the ole file.

Name

xls_read — read a sheet in an Excel file

```
[Value,TextInd] = xls_read(fd,Sheetpos)
```

Parameters

fd

a number, the logical unit on the Excel stream returned by xls_open.

Sheetpos

a number: the position of the beginning of the sheet in the Excel stream. This position is one of those returned by xls_open.

Value

a matrix of numbers, the numerical data found in the sheet. The cell without numerical data are represented by NaN values.

TextInd

a matrix of indices with the same size as Value. The 0 indices indicates that no string exists in the correspondin Excel cell. a positive index *i* points to the string SST(*i*) where SST is given by xls_open.

Description

This function reads an Excel sheet given a logical unit on an Excel stream ant the position of the beginning of the sheet within this stream. It returns the numerical data and the strings contained by the Excel cells.

The readxls function can be used to read all an Excel file in one function with a single function call.

Warning only BIFF8 Excel files (last Excel file version) are handled

Examples

```
//Decode ole file, extract and open Excel stream
[fd,SST,Sheetnames,Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/Test.
//Read first data sheet
[Value,TextInd] = xls_read(fd,Sheetpos(1))
//close the spreadsheet stream
mclose(fd)
```

See Also

xls_open , readxls

Authors

Pierrick Mode
INRIA

Serge Steer
INRIA

Bibliography

This function is based on Excel stream description from OpenOffice (<http://sc.openoffice.org/spreadsheetfileformat.pdf>).

Used Functions

This function uses the xls.c file which can be found in a Scilab source version in the directory SCIDIR/modules/spreadsheet/src/c

Part XLVII. call_scilab API

Name

Boolean management — How to manage Scilab's boolean read and write process using call_scilab and api_scilab

Description

This help describes how boolean and matrix of booleans can be handle through the Call Scilab API and api Scilab.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

Note: Access to variables is done through api Scilab (named variable).

Examples

```
/*
 * Write a matrix of boolean into Scilab
 * B=[F F T F;
 *   F F F T ]
 * Note that it is done column by column
 */
int B[]={0,0,0,0,1,0,0,1}; /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";
SciErr sciErr;

/* Write it into Scilab's memory */
sciErr = createNamedMatrixOfBoolean(pvApiCtx, variableNameB, rowB, colB, B);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(cwritebmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB));
 */

printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

```
/* Read the previously declared matrix of boolean B */
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;
int *matrixOfBooleanB = NULL; /* Int instead of double */

char variableToBeRetrievedB[] = "B";
SciErr sciErr;
/* First, retrieve the size of the matrix */
sciErr = readNamedMatrixOfBoolean(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_, &lp_);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}
```

```
}

/*
 * Prior to Scilab 5.2:
 * C2F(cmatbptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeRetrievedB));
 */

/* Alloc the memory */
matrixOfBooleanB=(int*)malloc((rowB_*colB_)*sizeof(int));

/* Load the matrix */
sciErr = readNamedMatrixOfBoolean(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(creadbmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfBooleanB,strlen(variableToBeRetrievedB));
 */

printf("\n");
printf("Display from B formatted (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user expects */
        printf("%d ",matrixOfBooleanB[i * rowB_ + j]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

API_Scilab: Boolean reading, API_Scilab: Boolean writing, SendScilabJob, StartScilab, Call_Scilab: Double Management, Call_Scilab: Complex Management, Call_Scilab: String Management, API_Scilab: Double Reading, API_Scilab: Double Writing, API_Scilab: String Reading, API_Scilab: String Writing

Authors

Sylvestre Ledru

Name

Complex management — How to manage Scilab's complex variable read and write process using call_scilab

Description

This help describes how doubles and matrix of complex can be handle through the Call Scilab API.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

Note: Access to variables is done through api Scilab (named variable).

Examples

```
/*
 * Write a matrix into Scilab
 * B=[1+%i 4-%i 2+1/2*%i 3;
 *    3 9 8+42*%i 2 ]
 * Note that it is done column by column
 */

double B[]={1,3,4,9,2,8,3,2};
double B_img[]={1,0.233,-1,-0.2,0.5,42,-23,123}; /* Declare the matrix */

/*
 * Prior to Scilab 5.2:
 * double B[]={1,3,4,9,2,8,3,2,1,0.233,-1,-0.2,0.5,42,-23,123};
 */

int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";
SciErr sciErr;

/* Write it into Scilab's memory */
sciErr = createNamedComplexMatrixOfDouble(pvApiCtx,variableNameB,rowB,colB, B, B_img);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * 2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB));
 */
printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */

int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;

double *matrixOfComplexB = NULL;
```

```
double *matrixOfComplexB_img = NULL;
char variableToBeRetrievedB[] = "B";
SciErr sciErr;

/* First, retrieve the size of the matrix */
readNamedComplexMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_)
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(cmatcptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeRetrievedB))
 */

/* Alloc the memory */
matrixOfComplexB = (double*)malloc((rowB_*colB_)*sizeof(double));
matrixOfComplexB_img = (double*)malloc((rowB_*colB_)*sizeof(double));

/* Load the matrix */
sciErr = readNamedComplexMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_)
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(creadcmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfComplexB,strlen(variableToBeRetrievedB))
 */

printf("\n");
printf("Display from B formatted (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user
         * expect */
        printf("%5.2f + %5.2f.i  ",matrixOfComplexB[i * rowB_ + j],matrixOfComplexB_img[i * rowB_ + j]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

[Call_Scilab](#), [api_Scilab](#), [cwritecmat](#), [creadcmat](#), [SendScilabJob](#), [StartScilab](#), [Call_Scilab: Complex Management](#), [Call_Scilab: Boolean Management](#), [Call_Scilab: String Management](#), [API_Scilab: Boolean Reading](#), [API_Scilab: Boolean Writing](#), [API_Scilab: String Reading](#), [API_Scilab: String Writing](#)

Authors

Sylvestre Ledru

Name

DisableInteractiveMode — Disables some features (plotting, gui creation, Tcl/Tk...) and leaves only the computing engine

Synopsis

```
void DisableInteractiveMode(void);
```

Description

Calling this fonction will disable some features of Scilab.

Examples

```
// A simple DisableInteractiveMode example

DisableInteractiveMode();
int code=SendScilabJob("plot3d()"); /* This will failed since plot3d is
                                     among the disable features*/
if (code!=0){
    char lastjob[4096];
    if (GetLastJob(lastjob,4096)) {
        printf("Failed command: %s\n",lastjob);
    }
}
```

See Also

Call_Scilab,api Scilab,Compile and run with call_scilab SendScilabJob TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

Double management — How to manage Scilab's variable read and write process using call_scilab and api_scilab

Description

This help describes how doubles and matrix of doubles can be handle through the Call Scilab API and api Scilab.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

Note that the default type of a numeric value in Scilab is double. For example, in $a=1$ or $a=[1,2]$, a will be consider as a double.

Note: Access to variables is done through api Scilab (named variable).

Examples

```
/*
 * Write a matrix into Scilab
 * B=[1 4 2 3;
 *    3 9 8 2 ]
 * Note that it is done column by column
 */
double B[] = {1,3,4,9,2,8,3,2}; /* Declare the matrix */
int rowB = 2, colB = 4; /* Size of the matrix */
char variableNameB[] = "B";
SciErr sciErr;

/*
 * Write it into Scilab's memory
 */
sciErr = createNamedMatrixOfDouble(pvApiCtx,variableNameB,rowB,colB, B); /* pvA
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB));
 */

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */

/* Read the previously declared matrix of double B */
int rowB_ = 0, colB_ = 0, lp_ = 0;
double *matrixOfDoubleB = NULL;
int i = 0, j = 0;
char variableToBeRetrievedB[] = "B";
SciErr sciErr;
```

```
/* First, retrieve the size of the matrix */
sciErr = readNamedMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(cmatptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeRetrievedB));
 */

/* Alloc the memory */
matrixOfDoubleB = (double*)malloc((rowB_*colB_)*sizeof(double));

/* Load the matrix */
sciErr = readNamedMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2:
 * C2F(creadmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfDoubleB,strlen(variableToBeRetrievedB));
 */

printf("\n");
printf("Display from B formatted (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user
         * expect */
        printf("%5.2f ",matrixOfDoubleB[i * rowB_ + j]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

Call_Scilab,api Scilab,API_Scilab: Double reading, API_Scilab: Double writing, SendScilabJob, StartScilab, Call_Scilab: BooleanManagement, Call_Scilab: ComplexManagement, Call_Scilab: StringManagement, API_Scilab: Boolean Reading, API_Scilab: Boolean Writing, API_Scilab: String Reading, API_Scilab: String Writing

Authors

Sylvestre Ledru

Name

GetLastJob — Returns the latest job sent to Scilab engine

Synopsis

```
BOOL GetLastJob(char *JOB,int nbcharsJOB);
```

Description

This fonction returns the latest job sent to Scilab engine with the command SendScilabJob or SendScilabJobs. This can be used to display a command which failed.

BOOL is just a simple typedef on int (typedef int BOOL). TRUE is defined on 1 (#define TRUE 1) and FALSE is defined on 0 (#define FALSE 0).

Parameters

JOB

a standard C char* which will be filled with the latest job

nbcharsJOB

The number of char of JOB

returns

1 (TRUE) if the operation is successfull.

0 (FALSE) if an error during initialization occurred.

Examples

```
// A simple GetLastJob example
// See SCI/modules/call_scilab/examples/basicExamples/GetLastJob.c for
// the full code
int code=SendScilabJob("failedMyCurrentJob=%pi*3/0");
if (code!=0){
    char lastjob[4096];
    if (GetLastJob(lastjob,4096)) {
        printf("Failed command: %s\n",lastjob);
    }
}
```

See Also

Call_Scilab,api Scilab,Compile and run with call_scilab, SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

ScilabHaveAGraph — Check if any Scilab graphics have been opened.

Synopsis

```
int ScilabHaveAGraph(void);
```

Description

Returns if there is any Scilab graphics opened.

Examples

```
// A simple ScilabHaveAGraph example
int code=SendScilabJob("plot3d()"); /* This will failed since plot3d is
                                     among the disable features*/
if (ScilabHaveAGraph()){
    printf("Graphics\n");
}else{
    printf("NO Graphics\n");
}
```

See Also

Call_Scilab,api Scilab,Compile and run with call_scilab SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

SendScilabJob — Send a Scilab task from a C/C++ code (call_scilab)

Synopsis

```
int SendScilabJob(char *job);
```

Description

This fonction is provided in call_scilab. This function send a task which will be processed by the Scilab engine.

Parameters

job

a standard C char* containing the Scilab instructions

returns

0 is the operation is successfull.

-1 if Call Scilab has not been able to write the job into Scilab.

-2 or -3 if Call Scilab has not been able to read the error code from Scilab.

-4 if Call Scilab has not been able to allocate the job.

Examples

```
// A simple SendScilabJob example

if (SendScilabJob("disp('unfinished quote')")!=0){ // This will fail
    printf("An error occured\n");
}
SendScilabJob("myMatrix=['sample','for the help']");
```

See Also

Call_Scilab,api Scilab,Compile and run with call_scilab, SendScilabJob, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

SendScilabJobs — Send Scilab tasks from a C/C++ code (call_scilab)

Synopsis

```
int SendScilabJobs(char **jobs, int numberjobs);
```

Description

This fonction is provided in call_scilab. This function send many tasks which will be processed by the Scilab engine.

Note that the ending ";" is not mandatory at the end of a command.

Parameters

jobs

an array of standard C char* containing the Scilab instructions

numberjobs

The number of the Scilab instructions

returns

0 is all the operations are successful.

-10 if .

<0 and > -10 when an error occurred in the execution.

Examples

```
// A simple SendScilabJobs example

char* jobs[3];
jobs[0]="a = 1";
jobs[1]="b = 3";
jobs[2]="c = a + b;";
SendScilabJobs(jobs,3);
SendScilabJob("disp(c);"); // Will display 4.
```

See Also

Call_Scilab, api Scilab, SendScilabJob, Compile and run with call_scilab, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

StartScilab — Initializes and starts Scilab engine in Call Scilab

Synopsis

```
BOOL StartScilab(char *SCIpath, char *ScilabStartup, int
                 *Stacksize);
```

Description

This function starts the Scilab engine. This is mandatory to use SendScilabJob functions and to manage Scilab's data.

BOOL is just a simple typedef on int (typedef int BOOL). TRUE is defined on 1 (#define TRUE 1) and FALSE is defined on 0 (#define FALSE 0).

Parameters

SCIpath

a standard C char* containing the path to Scilab data

This argument is mandatory under Linux, Unix or Mac OS X.

Under Windows, if SCIpath is NULL, Scilab will find the path.

ScilabStartup

a standard C char* containing the path to Scilab startup script (scilab.start)

If ScilabStartup is NULL, Scilab will use the default path (detected from SCIpath).

Stacksize

a standard int* defining the size of the Scilab stack

If Stacksize is NULL, Scilab will use the default stacksize of Scilab.

returns

1 (TRUE) if the operation is successful.

0 (FALSE) if an error during initialization occurred.

Examples

```
// A simple StartScilab example
if ( StartScilab(getenv("SCI"),NULL,NULL) == FALSE )
{
    fprintf(stderr,"Error while calling StartScilab\n");
    return -1;
}
```

See Also

Call_Scilab, api Scilab, Compile and run with call_scilab, SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

String management — How to manage Scilab's String read and write process using `call_scilab` and `api_scilab`

Description

This help describes how strings and matrix of strings can be handle through the Call Scilab API and `api Scilab`.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

Note: Access to variables is done through `api Scilab` (named variable).

Examples

```
// This example shows how to write a Scilab string in Scilab engine
// It is the equivalent to A="my Message"; in Scilab interpreter
// See: modules/call_scilab/examples/basicExamples/readwritestring.c

// StartScilab
int row = 1, col = 1; /* Size of the matrix */
/* Declare the string */
char **myMatrixOfString = (char**)malloc(sizeof(char*) * row * col);
myMatrixOfString[0]="my Message";
char variableName[] = "A";
SciErr sciErr;

/* Write it into Scilab's memory */
sciErr = createNamedMatrixOfString(pvApiCtx, variableName, row, col, myMatrixOfString);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

/*
 * Prior to Scilab 5.2
 * C2F(cwritechain)(variableName, &sizeOfMyString , myString, strlen(variableName))
 */

printf("Display from Scilab of A:\n");
SendScilabJob("disp(A);"); /* Display A */
```

```
/* Load the previously set variable A */
// See: modules/call_scilab/examples/basicExamples/readwritestring.c

char variableToBeRetrieved[]="A";
int iRows      = 0;
int iCols      = 0;
int i,j;
int* piAddr    = NULL;
int* piLen     = NULL;
char** pstData = NULL;
SciErr sciErr;
```

```
//first call to retrieve dimensions
sciErr = readNamedMatrixOfString(pvApiCtx,variableToBeRetrieved,&iRows, &iCols,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

piLen = (int*)malloc(sizeof(int) * iRows * iCols);
//second call to retrieve length of each string
sciErr = readNamedMatrixOfString(pvApiCtx,variableToBeRetrieved, &iRows, &iCols
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
}
//third call to retrieve data
sciErr = readNamedMatrixOfString(pvApiCtx, variableToBeRetrieved, &iRows, &iCol
if(sciErr.iErr)
{
    printError(&sciErr, 0);
}

printf("\n");
printf("Load and display of A:\n");
for(j = 0 ; j < iCols ; j++)
{
    for(i = 0 ; i < iRows ; i++)
    {
        /* Display the formatted matrix with same scilab indice */
        printf("[%d,%d] = %s\n",j+1,i+1,pstData[j* iRows + i]);
    }
}

printf("\n");
free(piLen);
for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pstData[i]);
}
free(pstData);
```

See Also

[Call_Scilab](#),[api Scilab](#), [cwritechain](#), [creadchain](#), [SendScilabJob](#), [StartScilab](#), [Call_Scilab: Boolean Management](#), [Call_Scilab: Double Management](#), [API_Scilab: Boolean Reading](#), [API_Scilab: Boolean Writing](#), [API_Scilab: Double Reading](#), [API_Scilab: Double Writing](#), [Complex Management](#)

Authors

Sylvestre Ledru

Name

TerminateScilab — Stops and terminates Scilab engine in Call Scilab

Synopsis

```
BOOL TerminateScilab(char *ScilabQuit);
```

Description

This fonction stops the Scilab engine. It is strongly recommended to call this function at the end when using Call Scilab .

BOOL is just a simple typedef on int (typedef int BOOL). TRUE is defined on 1 (#define TRUE 1) and FALSE is defined on 0 (#define FALSE 0).

Parameters

ScilabQuit

a standard C char* containing the path to Scilab quit script (scilab.quit)

If ScilabStartup is NULL, Scilab will use the default path (detected from SCIpath).

returns

1 (TRUE) if the operation is successful.

0 (FALSE) if an error during initialization occurred.

Examples

```
// A simple TerminateScilab example
if ( TerminateScilab(NULL) == FALSE ) {
    fprintf(stderr, "Error while calling TerminateScilab\n");
    return -2;
}
```

See Also

Call_Scilab, api Scilab, Compile and run with call_scilab, StartScilab, SendScilabJob, Double Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

call_scilab — call_scilab is an interface which provides the ability to call Scilab engine from C/C++ code

Description

Scilab offers the possibility to be called from a native (C/C++) application.

Thanks to this module, it is possible to call Scilab from C/C++ in order to interface Scilab's features from an other code/application or to be able to interface Scilab's features from an other language.

Since Scilab 5.2.0, all Scilab datatype can be handle by call_scilab. This is done thanks to API_Scilab

This help describes the features of the call_scilab API.

Note: The javasci module is based on call_scilab.

Note: old APIs (stackX.h) will not be available after Scilab 6.0 (included).

Examples

```
// A simple call_scilab example

#include <stdio.h> /* stderr */

#include "stack-c.h" /* Provide functions to access to the memory of Scilab */
#include "call_scilab.h" /* Provide functions to call Scilab engine */

// Filename: simple_call_scilab.c

int main(void)
{
    /****** INITIALIZATION *****/
    #ifdef _MSC_VER
        if ( StartScilab(NULL,NULL,NULL) == FALSE )
    #else
        if ( StartScilab(getenv("SCI"),NULL,NULL) == FALSE )
    #endif
    {
        fprintf(stderr,"Error while calling StartScilab\n");
        return -1;
    }

    /****** ACTUAL Scilab TASKS *****/

    SendScilabJob("myMatrix=['sample','for the help']");
    SendScilabJob("disp(myMatrix);"); // Will display !sample for the help !
    SendScilabJob("disp([2,3]+[-44,39]);"); // Will display - 42. 42.

    /****** TERMINATION *****/
    if ( TerminateScilab(NULL) == FALSE ) {
        fprintf(stderr,"Error while calling TerminateScilab\n");
        return -2;
    }
    return 0;
}
```

See Also

api Scilab, Compile and run with call_scilab, Matrix Management, Boolean Management, Complex Management, String Management

Authors

Sylvestre Ledru

Name

Compile and run with Call Scilab — How to compile a native application based on or using Scilab

Compilation

To compile a native code based on Call Scilab, it is necessary to define some arguments, variables and paths.

- CFLAGS. Call Scilab needs to have access to the headers of Scilab core and call Scilab module.
 - Linux/Unix/MacOSX:
 - In the binary version of Scilab, CFLAGS must be set to `/path/to/scilab/include/scilab/core/` and `/path/to/scilab/include/scilab/call_scilab/`
 - In the source tree of Scilab, CFLAGS must be set to `/path/to/scilab/modules/core/includes/` and `/path/to/scilab/modules/call_scilab/includes/`
 - Windows
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .dll, .jnilib...)
 - Linux/Unix/MacOSX:
 - In the binary version of Scilab, SCI will point to `/path/to/scilab/lib/scilab/`
 - In the source tree of Scilab, SCI will point to the root of the source tree `/path/to/scilab/modules/call_scilab/.libs/` and `/path/to/scilab/.libs/`
- LDFLAGS - The name of the library to link against
 - Linux/Unix/MacOSX: It is only mandatory to link against scilab. This should include the other libraries.

```
# A sample Makefile building a C code using Call Scilab using Scilab built in i
PATH_SCILAB = /path/to/scilab/sources.
SCILAB_CFLAGS = -I$(PATH_SCILAB)/modules/core/includes/ -I$(PATH_SCILAB)/module
SCILAB_LDFLAGS = -lscilab
PATH_TO_LIB_SCILAB = $(PATH_SCILAB)/modules/.libs/
PATH_TO_LIB_CALL_SCILAB = $(PATH_SCILAB)/modules/call_scilab/.libs/

all: simple_call_scilab.c
export LD_LIBRARY_PATH=$(PATH_TO_LIB_SCILAB):$(PATH_TO_LIB_CALL_SCILAB)
gcc -o myExample $(SCILAB_LDFLAGS) -L$(PATH_TO_LIB_SCILAB) -L$(PATH_TO_LIB_CAL
```

Running

To run an application based on Call Scilab, there are a few other things to set up.

Some global variables must me set:

- SCI - Path to Scilab files
 - Linux/Unix/MacOSX:
 - In the binary version of Scilab, SCI will point to `/path/to/scilab/share/scilab/`

- In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
- Windows:
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .dll, .jnilib...)
- Linux/Unix/MacOSX:
 - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
 - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/javasci/.libs/ and /path/to/scilab/.libs/
- LD_LIBRARY_PATH (Java) - Paths to Java native libraries (libjava.so, libjvm.so, libhpi.so)... It is usually provided by the operating system or by Scilab distribution. Please note that won't be necessary in Scilab 5.2
- Linux/Unix:
 - JAVA_HOME/jre/lib/<arch>/, JAVA_HOME/jre/lib/<arch>/server, JAVA_HOME/jre/lib/<arch>/native_threads/ (<arch> can be i386, etc...)
- Mac OS X:
- Windows

Note that two environment variables are taken in account for specific needs:

- SCI_DISABLE_TK=1 Disables Tk (Tcl's GUI)
- SCI_JAVA_ENABLE_HEADLESS=1 Launch Java in headless mode (no AWT/Swing)

Examples

```
# Serie of declarations to execute my binary.

# With a Scilab source tree:
$ SCI=<path to Scilab source tree>
$ export LD_LIBRARY_PATH=$SCI/modules/.libs/:$SCI/modules/call_scilab/.libs/:$SCI/modules/javasci/.libs/

# With a Scilab binary:
$ SCI=<path to Scilab binary>
$ export LD_LIBRARY_PATH=$SCI/lib/scilab/

# Don't forget to update i386 to whatever is necessary
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/jvm/java-6-openjdk//jre/lib/
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/jvm/java-6-openjdk//jre/lib/
$ export SCI=/path/to/scilab/
$ ./myExample
!sample for the help !

- 42.      42.
```

See Also

api Scilab, call_scilab, StartScilab, SendScilabJob, SendScilabJobs, Double Management, Boolean Management, Complex Management, String Management

Authors

Allan Cornet

Sylvestre Ledru

Name

`creadbmat` (obsolete) — Read a single boolean or a matrix of boolean from Scilab memory using `call_scilab`. Starting with Scilab 5.2, this function is obsolete. See [API_Scilab: Boolean reading for replacement](#).

Synopsis

```
int C2F(creadbmat)(char *name, int *m, int *n, int *scimat, unsigned long name_
```

Parameters

- `name`
The name of the future Scilab variable
- `m`
Number of rows
- `n`
Number of columns
- `scimat`
The actual matrix of boolean (array of int). Note that it is going to be stored in Scilab columnwise.
- `name_len`
The length of the variable *name* (fortran compatibility)
- `C2F`
C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `creadbmat`.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *int* *.

Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwriteboolean.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;
int *matrixOfBooleanB = NULL; /* Int instead of double */

char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatbptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeRetrievedB));

/* Alloc the memory */
matrixOfBooleanB=(int*)malloc((rowB_*colB_)*sizeof(int));

/* Load the matrix */
C2F(creadbmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfBooleanB,strlen(variableToBeRetrievedB));

printf("\n");
printf("Display from B formatted (size: %d, %d):\n",rowB_, colB_);
```

```
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user
         * expect */
        printf("%d ",matrixOfBooleanB[i * rowB_ + j]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

[Call_Scilab](#), [api Scilab](#), [API_Scilab](#): Boolean reading, [API_Scilab](#): Boolean writing, [SendScilabJob](#), [StartScilab](#), [cwritebmat](#), [Boolean Management](#)

Authors

Sylvestre Ledru

Name

creadchain (obsolete) — Read a single string from Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: String reading for replacement.

Synopsis

```
int C2F(creadchain)(char *name, int *itslen, char *myString, unsigned long name.
```

Parameters

name

The name of the future Scilab variable

itslen

The length of the future buffer (usually, use *bsiz*, it skips the need to detect the size of a char). This variable will be altered to contain the actual size of *myString*

myString

The actual String (char *) which is going to content the content of the Scilab variable *name*

name_len

The length of the variable *name* (fortran compatibility)

myString_len

The length of the string (fortran compatibility)

C2F

C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function creadchain.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *char **.

Examples

```
/* Load the previously set variable A */
// See: modules/call_scilab/examples/basicExamples/readwritestring.c
int sizeA = 0;
char myStringFromScilab[bsiz]; /* Static char */
int length_myStringFromScilab = bsiz; /* Max size (it is going to be changed by
char variableToBeRetrieved[]="A";

/* We are loading a single string from Scilab */
C2F(creadchain)(variableToBeRetrieved,&length_myStringFromScilab,myStringFromSc
printf("\n");
printf("Display of A (size %d): %s\n", length_myStringFromScilab, myStringFromS
```

See Also

Call_Scilab,api Scilab,API_Scilab: String reading, API_Scilab: String writing, SendScilabJob, StartScilab, cwritechain, String Management

Authors

Sylvestre Ledru

Name

`creadcmat` (obsolete) — Read a single complex or a matrix of complex from Scilab memory using `call_scilab`. Starting with Scilab 5.2, this function is obsolete. See `API_Scilab: Complex double reading` for replacement.

Synopsis

```
int C2F(creadcmat)(char *name, int *m, int *n, double *scimat, unsigned long na
```

Parameters

`name`

The name of the future Scilab variable

`m`

Number of rows

`n`

Number of columns

`scimat`

The actual matrix of complex (array of double). Note that it is going to be stored in Scilab column-wise and the second half of the array is used for complex values.

`name_len`

The length of the variable *name* (fortran compatibility)

`C2F`

C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `creadcmat`.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *double* *.

Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwritecomplexmatrix.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;

double *matrixOfComplexB = NULL;
char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatcptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBe

/* Alloc the memory */
matrixOfComplexB = (double*)malloc((rowB_*colB_*2)*sizeof(double));

/* Load the matrix */
C2F(creadcmat)(variableToBeRetrievedB, &rowB_, &colB_, matrixOfComplexB, strlen(var

printf("\n");
```

```
printf("Display from B formatted (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user
         * expect */
        printf("%5.2f + %5.2f.i  ",matrixOfComplexB[i * rowB_ + j],matrixOfComplexB[i * rowB_ + j + 1]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

[Call_Scilab](#),[api Scilab](#),[API_Scilab](#): Complex double reading, [API_Scilab](#): Double writing, [SendScilabJob](#), [StartScilab](#), [cwritecmat](#), [Complex Management](#)

Authors

Sylvestre Ledru

Name

`creadmat` (obsolete) — Read a single double or a matrix of double from Scilab memory using `call_scilab`. Note that it is the default datatype in Scilab. Starting with Scilab 5.2, this function is obsolete. See `API_Scilab: Double reading` for replacement.

Synopsis

```
int C2F(creadmat)(char *name, int *m, int *n, double *scimat, unsigned long nam
```

Parameters

`name`

The name of the future Scilab variable

`m`

Number of rows

`n`

Number of columns

`scimat`

The actual matrix of double (array of double). Note that it is going to be stored in Scilab column-wise.

`name_len`

The length of the variable *name* (fortran compatibility)

`C2F`

`C2F` is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `creadmat`.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *double* *.

Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwritematrix.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
double *matrixOfDoubleB = NULL;
int i = 0, j = 0;

char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeR

/* Alloc the memory */
matrixOfDoubleB = (double*)malloc((rowB_*colB_)*sizeof(double));

/* Load the matrix */
C2F(creadmat)(variableToBeRetrievedB, &rowB_, &colB_, matrixOfDoubleB, strlen(varia

printf("\n");
```

```
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
    for(i = 0 ; i < colB_ ; i++)
    {
        /* Display the formatted matrix ... the way the user
         * expect */
        printf("%5.2f ",matrixOfDoubleB[i * rowB_ + j]);
    }
    printf("\n"); /* New row of the matrix */
}
```

See Also

Call_Scilab,api_Scilab,API_Scilab: Double reading,API_Scilab: Double writing, SendScilabJob, StartScilab, cwriteat, Double Management

Authors

Sylvestre Ledru

Name

`cwritebmat` (obsolete) — Write a single boolean or a matrix of boolean into Scilab memory using `call_scilab`. Starting with Scilab 5.2, this function is obsolete. See [API_Scilab: Boolean writing for replacement](#).

Synopsis

```
int C2F(cwritebmat)(char *name, int *m, int *n, int *mat, unsigned long name_len)
```

Parameters

`name`

The name of the future Scilab variable

`m`

Number of rows

`n`

Number of columns

`mat`

The actual matrix of boolean (array of int). Note that it is going to be stored in Scilab columnwise.

`name_len`

The length of the variable *name* (fortran compatibility)

`C2F`

C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `cwritebmat`.

Using this function will basically do the same as $A = [T F F T]$; but straight into Scilab memory with `call_scilab`.

Examples

```
// This example shows how to write a Scilab matrix of boolean in Scilab engine
// It is the equivalent to
// B=[F F T F;
//   F F F T] in Scilab interpreter
// See: modules/call_scilab/examples/basicExamples/readwriteboolean.c

// StartScilab
int B[]={0,0,0,0,1,0,0,1}; /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";
C2F(cwritebmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Write B */
printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

See Also

[Call_Scilab](#), [api Scilab](#), [API_Scilab: Boolean reading](#), [API_Scilab: Boolean writing](#), [SendScilabJob](#), [StartScilab](#), [creadbmat](#), [Boolean Management](#)

Authors

Sylvestre Ledru

Name

`cwritchain` (obsolete) — Write a single string into Scilab memory using `call_scilab`. Starting with Scilab 5.2, this function is obsolete. See [API_Scilab: String writing for replacement](#).

Synopsis

```
int C2F(cwritchain)(char *name, int *myStringSize, char *myString, unsigned long
```

Parameters

`name`

The name of the future Scilab variable

`myStringSize`

The length of the string which is going to be write into Scilab memory

`myString`

The actual String (char *)

`name_len`

The length of the variable *name* (fortran compatibility)

`myString_len`

The length of the string (fortran compatibility)

`C2F`

C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `cwritchain`.

Using this function will basically do the same as `A = "my own String"`; but straight into Scilab memory with `call_scilab`.

Examples

```
// This example shows how to write a Scilab string in Scilab engine
// It is the equivalent to A="my Message"; in Scilab interpreter
// See: modules/call_scilab/examples/basicExamples/readwritestring.c

// StartScilab
char *myString = "my Message"; /* Declare the string */
char variableName[] = "A"; /* The name of the future variable in Scilab */
int sizeofMyString=strlen(myString);

C2F(cwritchain)(variableName, &sizeofMyString , myString, strlen(variableName));

printf("Display from Scilab of A:\n");
SendScilabJob("disp(A);"); /* Display A */
```

See Also

[Call_Scilab,api Scilab,API_Scilab: String reading](#), [API_Scilab: String writing](#), [SendScilabJob](#)
[StartScilab](#) [creadchains](#) [String Management](#)

Authors

Sylvestre Ledru

Name

`cwritemat` (obsolete) — Write a single complex or a matrix of complex into Scilab memory using `call_scilab`. Starting with Scilab 5.2, this function is obsolete. See [API_Scilab: Complex double writing](#) for replacement.

Synopsis

```
int C2F(cwritemat) (char *name, int *m, int *n, double *mat, unsigned long na
```

Parameters

- `name`
The name of the future Scilab variable
- `m`
Number of rows
- `n`
Number of columns
- `mat`
The actual matrix of complex (array of double). Note that it is going to be stored in Scilab column-wise and the second half of the array is used for complex values.
- `name_len`
The length of the variable *name* (fortran compatibility)
- `C2F`
C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `cwritemat`.

Using this function will basically do the same as $A = [1 + i \ 3i \ 4 \ 2 + 2i]$; but straight into Scilab memory with `call_scilab`.

Examples

```
// This example shows how to write a Scilab matrix of complex in Scilab engine
/*
 * Write a matrix into Scilab
 * B=[1+%i 4-%i 2+1/2*%i 3;
 *   3 9 8+42*%i 2 ]
 * Note that it is done column by column
 */

double B[]={1,3,4,9,2,8,3,2,1,0,-1,0,0.5,42,0,0}; /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";

C2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Write

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

See Also

Call_Scilab,api Scilab,API_Scilab: Complex double reading, API_Scilab: Complex double writing, SendScilabJob, StartScilab, Complex Management, creadcmat

Authors

Sylvestre Ledru

Name

`cwritemat` (obsolete) — Write a single double or a matrix of double into Scilab memory using `call_scilab`. Note that it is the default datatype in Scilab. Starting with Scilab 5.2, this function is obsolete. See `API_Scilab: Double writing` for replacement.

Synopsis

```
int C2F(cwritemat)(char *name, int *m, int *n, int *mat, unsigned long name_len
```

Parameters

- `name`
The name of the future Scilab variable
- `m`
Number of rows
- `n`
Number of columns
- `mat`
The actual matrix of double (array of double). Note that it is going to be stored in Scilab column-wise.
- `name_len`
The length of the variable *name* (fortran compatibility)
- `C2F`
C2F is just a macro which provides to this function the ability to be called from fortran

Description

This help describes how to use the function `cwritemat`.

Using this function will basically do the same as `A=[ 1 3 4 2 ]`; but straight into Scilab memory with `call_scilab`.

Examples

```
// This example shows how to write a Scilab matrix of double in Scilab engine
// * Write a matrix into Scilab
// * B=[1 4 2 3;
// *   3 9 8 2 ]
// * Note that it is done column by column
// See: modules/call_scilab/examples/basicExamples/readwritematrix.c

// StartScilab
double B[] = {1,3,4,9,2,8,3,2}; /* Declare the matrix */
int rowB = 2, colB = 4; /* Size of the matrix */
char variableNameB[] = "B";

C2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Write

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display A */
```

See Also

Call_Scilab,api Scilab, API_Scilab: Double reading,API_Scilab: Double writing, SendScilabJob, StartScilab, Double Management, creadmat

Authors

Sylvestre Ledru

Name

fromc — Checks if current Scilab is called from an external C program (by StartScilab).

```
r=fromc()
```

Parameters

r
a boolean

Description

Returns %t Checks if current Scilab is called from an external C program or %f if not.

See Also

Call_Scilab,api Scilab,fromjava

Name

fromjava — Checks if current Scilab is called from javasci

```
r = fromjava()
```

Parameters

r
a boolean

Description

Returns %t if Scilab is called from javasci or %f if not.

See Also

Call_Scilab,api Scilab,fromc

Part XLVIII. FFTW

Name

fftw — Fast fourier transform based on the fftw library

```
[y]=fftw(x)
[y]=fftw(x,sign)
[y]=fftw(x,sign,dim,incr)
[y]=fftw(x,sign,[dim1 dim2 ...dimN],[incr1 incr2 ...incrN])
```

Parameters

y,x
matrix/vector of real/complex data. Input/output data to be transformed.

sign
Integer. 1 or -1. Set direct or inverse transform.

dim
integer. Set the dimension (the length) of the transform.

incr
integer. Set the stride (the span) of the transform.

Description

This function realizes direct/inverse Discrete Fourier Transform (DFT) with the help of the FFTW library.

One can compute vector, 2D, M-D transform with this function.

For more details of fftw syntax see fft scilab function.

For more details about FFTW library see FFTW Web site : <http://www.fftw.org>

Remark: fftw function automatically stores his last parameters in memory to re-use it in a second time.

This improves greatly the time computation when consecutives calls (with same parameters) are performed.

Examples

```
//simple vector direct transform
a = rand(50,1)+%i*rand(50,1);
y = fftw(a);
y = fftw(a,-1);
//inverse transform
b = fftw(y,1);

//2D transform
a = rand(512,512)+%i*rand(512,512);
y = fftw(a);

//M-D transform -old calling sequence-
a = rand(120,1);
y = a;
dim=[5 6 4];incr=[1 5 30];
for i=1:3
```

```
y = fftw(y, -1, dim(i), incr(i));  
end  
  
//M-D transform -new calling sequence-  
//More efficient than old  
y = fftw(a, -1, [5 6 4], [1 5 30]);  
b = fftw(y, 1, [5 6 4], [1 5 30]);
```

See Also

fftw_flags, get_fftw_wisdom, set_fftw_wisdom, fftw_forget_wisdom

Bibliography

Matteo Frigo and Steven G. Johnson, "FFTW Manual fo version 3.1.2" June 2006. Available : <http://www.fftw.org>

Name

fftw_flags — set computation method of fast fourier transform of the fftw function

```
[a, [S]]=fftw_flags([x1;x2;...])
```

Parameters

[x1;x2;...]

Matrix of string or integers. Entry to switch the method of fft computation for fftw.

a

Integer. Give the current value of the flag of the fftw function.

S

String matrix. Give the string value of the fftw flag.

Description

This function enables the change of the unsigned flags parameter of the `fftw_plan_guru_split_dft` function that is used in `fftw` function.

Default value is `FFTW_ESTIMATE`

Accepted entries are :

- `FFTW_MEASURE` or 0
- `FFTW_DESTROY_INPUT` or 1
- `FFTW_UNALIGNED` or 2
- `FFTW_CONSERVE_MEMORY` or 4
- `FFTW_EXHAUSTIVE` or 8
- `FFTW_PRESERVE_INPUT` or 16
- `FFTW_PATIENT` or 32
- `FFTW_ESTIMATE` or 64
- `FFTW_ESTIMATE_PATIENT` or 128
- `FFTW_BELIEVE_PCost` or 256
- `FFTW_NO_DFT_R2HC` or 512
- `FFTW_NO_NONTHREADED` or 1024
- `FFTW_NO_BUFFERING` or 2048
- `FFTW_NO_INDIRECT_OP` or 4096
- `FFTW_ALLOW_LARGE_GENERIC` or 8192
- `FFTW_NO_RANK_SPLITS` or 16384
- `FFTW_NO_VRANK_SPLITS` or 32768
- `FFTW_NO_VRECURSE` or 65536

- FFTW_NO_SIMD or 131072
- FFTW_NO_SLOW or 262144
- FFTW_NO_FIXED_RADIX_LARGE_N or 524288
- FFTW_ALLOW_PRUNING or 1048576

Rmk : when using FFTW_MEASURE/FFTW_PATIENT/FFTW_EXHAUSTIVE you must call two times fftw. (first call for initialisation, second and others calls for computation)

Examples

```
//return the integer value of the flag
fftw_flags()

//change flags
fftw_flags(["FFTW_MEASURE";"FFTW_CONSERVE_MEMORY"]);

//change flags and display current value of fftw flags (both integer and string)
[a,S]=fftw_flags("FFTW_PATIENT")
```

See Also

[fftw](#)

Name

fftw_forget_wisdom — Reset fftw wisdom

```
fftw_forget_wisdom()
```

Description

This function reset the current fftw wisdom.

Examples

```
//return fftw wisdom  
txt=get_fftw_wisdom();  
//set fftw wisdom  
set_fftw_wisdom(txt);  
//reset fftw wisdom  
fftw_forget_wisdom();
```

See Also

fftw , get_fftw_wisdom , set_fftw_wisdom

Name

get_fftw_wisdom — return fftw wisdom

```
[txt]=get_fftw_wisdom()
```

Parameters

txt
String matrix that contains fftw wisdom.

Description

This function return the fftw wisdom in a string matrix.

Examples

```
//return fftw wisdom  
txt=get_fftw_wisdom();  
//set fftw wisdom  
set_fftw_wisdom(txt);  
//reset fftw wisdom  
fftw_forget_wisdom();
```

See Also

fftw , set_fftw_wisdom , fftw_forget_wisdom

Name

set_fftw_wisdom — set fftw wisdom

```
set_fftw_wisdom(txt)
```

Parameters

txt
String matrix that contains fftw wisdom.

Description

This function set the fftw wisdom with a string matrix.

Examples

```
//return fftw wisdom  
txt=get_fftw_wisdom();  
//set fftw wisdom  
set_fftw_wisdom(txt);  
//reset fftw wisdom  
fftw_forget_wisdom();
```

See Also

fftw , get_fftw_wisdom , fftw_forget_wisdom

Part XLIX. UMFPACK Interface

Name

PlotSparse — plot the pattern of non nul elements of a sparse matrix

```
PlotSparse(A [, style])
```

Parameters

A

a sparse matrix

style

(optional) a string given the color and/or the marker type of the form "[color][mark]" where color may be a number referring the color you want to use (in the current colormap). If you use the std colormap then color may be one of the following letters :

```
k  for black      b  for blue
r  for red        g  for green
c  for cyan       m  for magenta
y  for yellow     t  for turquoise
G  a dark green
```

mark must be one of the following :

```
.  point          +  plus
x  cross          *  circled plus
D  filled diamond d  diamond
^  upper triangle v  down triangle
o  circle
```

by default you have "b." (in fact the 2d color) and this is also forced in case of error.

Description

plot the pattern of non nul elements of a sparse matrix : each non nul element is drawn with a marker.
For "big" matrix use essentially the point . as marker

Examples

```
[A,description,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.
PlotSparse(A,"y+")
xtitle(ref + "." + mtype + " : " + description)
```

See Also

ReadHBSparse

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

ReadHBSparse — read a Harwell-Boeing sparse format file

```
[A, description, ref, mtype] = ReadHBSparse([filename])
```

Parameters

filename

(optional) a string given the filename (eventually preceeding by the path), if filename is not given then the function use uigetfile to get filename interactively

A

the sparse matrix

description

a string given some information about the matrix

ref

a string given the reference of the matrix

mtype

a string given the type of the matrix

Description

An utility to read the Harwell-Boeing sparse matrix format. Currently don't work for unassembled matrix. Also the eventual rhs vectors of the file are not returned. Generally the file name is of the form ref.mtype where mtype is a 3 letters word abc given some information (already inside the file) on the matrix :

```
a = R|C|P   for real|complex|pattern (no values given)
b = S|H|Z|U for symmetric|hermitian|skew symmetric|asymmetric
c = A|E     for assembled|unassembled matrix
           (case E is not treated by this func)
```

References

Users' Guide for the Harwell-Boeing Sparse Matrix Collection Iain S. Duff, Roger G. Grimes, John G. Lewis. You may found this guide and numerous sparse matrices (in the Harwell-Boeing format) at the University of Florida Sparse Matrix Collection

web site : <http://www.cise.ufl.edu/research/sparse/matrices/>

maintained by Tim Davis (<http://www.cise.ufl.edu/~davis/>)

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
```

See Also

PlotSparse

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

cond2sp — computes an approximation of the 2-norm condition number of a s.p.d. sparse matrix

```
[K2, lm, vm, LM, vM] = cond2sp(A, C_ptr [, rtol, itermax, verb])
```

Parameters

- A**
a real symmetric positive definite sparse matrix
- C_ptr**
a pointer to a Cholesky factorization (got with taucs_chfact)
- rtol**
(optional) relative tolerance (default 1.e-3) (see details in DESCRIPTION)
- itermax**
(optional) maximum number of iterations in the underlying algorithms (default 30)
- verb**
(optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.
- K2**
estimated 2-norm condition number $K2 = \|A\|_2 \|A^{-1}\|_2 = LM/lm$
- lm**
(real positive scalar) minimum eigenvalue
- vm**
associated eigenvector
- LM**
(real positive scalar) maximum eigenvalue
- vM**
associated eigenvector

Description

This quick and dirty function computes (LM, vM) using the iterative power method and (lm, vm) with the inverse iterative power method, then $K2 = LM/lm$. For each method the iterations are stopped until the following condition is met :

```
| (l_new - l_old)/l_new | < rtol
```

but 4 iterations are nevertheless required and also the iterations are stopped if itermax is reached (and a warning message is issued). As the matrix is symmetric this is the rayleigh quotient which gives the estimated eigenvalue at each step ($\lambda = v' * A * v$). You may called this function with named parameter, for instance if you want to see the intermediary result without setting yourself the rtol and itermax parameters you may called this function with the syntax :

```
[K2, lm, vm, LM, vM] = cond2sp(A , C_ptr, verb=%t )
```

Caution

Currently there is no verification for the input parameters !

Remark

This function is intended to get an approximation of the 2-norm condition number (K2) and with the methods used, the precision on the obtained eigenvectors (vM and vm) are generally not very good. If you look for a smaller residual $||Av - l*v||$, you may apply some inverse power iterations from v0 with the matrix :

```
B = A - l0*speye(A)
```

For instance, applied 5 such iterations for (lM, vM) is done with :

```
l0 = lM ; v0 = vM; // or l0 = lM ; v0 = vM; // to polish (lM,vM)
B = A - l0*speye(A);
LUp = umf_lufact(B);
vr = v0; nstep = 5;
for i=1:nstep, vr = umf_lusolve(LUp, vr, "Ax=b", B); vr = vr/norm(vr) ; end
umf_ludel(LUp); // if you don't use anymore this factorization
lr = vr'*A*vr;
norm_r0 = norm(A*v0 - l0*v0);
norm_rr = norm(A*vr - lr*vr);
// Bauer-Fike error bound...
mprintf(" first estimated eigenvalue : l0 = %e \n\t", l0)
mprintf(" |l-l0| <= ||Av0-l0v0|| = %e , |l-l0|/l0 <= %e \n\r", norm_r0, norm_r0)
mprintf(" raffined estimated eigenvalue : lr = %e \n\t", lr)
mprintf(" |l-lr| <= ||Avr-lrvr|| = %e , |l-lr|/lr <= %e \n\r", norm_rr, norm_rr)
```

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
[K2, lM, vM, lM, vM] = cond2sp(A , C_ptr, 1.e-5, 50, %t );
taucs_chdel(C_ptr)
```

See Also

condestsp , taucs_chfact , rcond

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

condestsp — estimate the condition number of a sparse matrix

```
[K1] = condestsp(A, LUp, t)
[K1] = condestsp(A, LUp)
[K1] = condestsp(A, t)
[K1] = condestsp(A)
```

Parameters

A

a real or complex square sparse matrix

LUp

(optional) a pointer to (umf) LU factors of A obtained by a call to umf_lufact ; if you have already computed the LU (= PAQ) factors it is recommended to give this optional parameter (as the factorization may be time consuming)

t

(optional) a positive integer (default value 2) by increasing this one you may hope to get a better (even exact) estimate

K1

estimated 1-norm condition number of A

Description

Give an estimate of the 1-norm condition number of the sparse matrix A by Algorithm 2.4 appearing in :

```
"A block algorithm for matrix 1-norm estimation
with an application to 1-norm pseudospectra"
Nicholas J. Higham and Francoise Tisseur
Siam J. Matrix Anal. Appl., vol 21, No 4, pp 1185-1201
```

Noting the exact condition number $K1e = ||A||_1 ||A^{(-1)}||_1$, we have allways $K1 \leq K1e$ and this estimate gives in most case something superior to $1/2 K1e$

Examples

```
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
K1 = condestsp(A)
// verif by direct computation
K1e = norm(A,1)*norm(inv(full(A)),1)

// another example
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
K1 = condestsp(A)
// this example is not so big so that we can do the verif
```

```
K1e = norm(A,1)*norm(inv(full(A)),1)

// if you have already the lu factors condestsp(A,Lup) is faster
// because lu factors are then not computed inside condestsp
Lup = umf_lufact(A);
K1 = condestsp(A,Lup)
umf_ludel(Lup)          // clear memory
```

See Also

umf_lufact , rcond

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

rafiter — (obsolete) iterative refinement for a s.p.d. linear system

```
[xn, rn] = rafiter(A, C_ptr, b, x0, [, nb_iter, verb])
```

Parameters

- A
a real symmetric positive definite sparse matrix
- C_ptr
a pointer to a Cholesky factorization (got with taucs_chfact)
- b
column vector (r.h.s of the linear system) but "matrix" (multiple r.h.s.) are allowed.
- x0
first solution obtained with taucs_chsolve(C_ptr, b)
- nb_iter
(optional) number of raffinement iterations (default 2)
- verb
(optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.
- xn
new refined solution
- rn
residual ($A \cdot x_n - b$)

Description

This function is somewhat obsolete, use `x = taucs_chsolve(C_ptr, b, A)` (see taucs_chsolve) which do one iterative refinement step.

To use if you want to improve a little the solution got with taucs_chsolve. Note that with verb=%t the displayed internal steps are essentially meaningful in the case where b is a column vector.

Caution

Currently there is no verification for the input parameters !

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
[xn, rn] = rafiter(A, C_ptr, b, x0, verb=%t);
norm(A*xn - b)
taucs_chdel(C_ptr)
```


See Also

taucs_chsolve , taucs_chfact

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

`res_with_prec` — computes the residual $r = Ax - b$ with precision

```
[r, norm2_r] = res_with_prec(A, x, b)
```

Parameters

- A
real or complex sparse matrix (m x n)
- x
column vector (n x 1) or matrix (n x p)
- b
column vector (m x 1) or matrix (m x p)
- r
column vector (m x 1) or matrix (m x p)
- norm2_r
scalar or vector (1 x p) when b is a m x p matrix

Description

This function computes the residual of a linear system $r = Ax - b$ (together with its 2-norm) with the additional precision provided on "Intel like" FPU (80 bits in place of 64) if the compiler translates "long double" to use it. Else one must get the same than using $A*x - b$ at the scilab level. In both cases using `[r, nr] = res_with_prec(A, x, b)` is faster than $r = A*x - b$ (and faster than $r = A*x - b$; $nr = \text{norm}(r)$).

When $p > 1$, `norm2_r(i)` is the 2-norm of the vector `r(:, i)`.

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
norm(res_with_prec(A, x0, b))
```

See Also

`rafit`

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

taucs_chdel — utility function used with taucs_chfact

```
taucs_chdel(C_ptr) or taucs_chdel()
```

Parameters

C_ptr
a pointer to a Cholesky factorization

Description

This function is used in conjunction with taucs_chfact and taucs_chsolve. It clears the internal memory space used to store the Cholesky factorization (got with taucs_chfact). Use without argument it frees the memory for all the current scilab (taucs) Cholesky factorizations.

Examples

see the example section of taucs_chfact

See Also

taucs_chfact , taucs_chsolve , taucs_chinfo , taucs_chget

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chfact — cholesky factorisation of a sparse s.p.d. matrix

```
C_ptr = taucs_chfact(A)
```

Parameters

- A
a sparse real symmetric positive definite (s.p.d.) matrix
- C_ptr
a pointer to the Cholesky factors (C,p : $A(p,p)=CC'$)

Description

This function computes a Cholesky factorization of the sparse symmetric positive definite (s.p.d.) matrix A and retrieves at the scilab level, a pointer (C_ptr) to an handle of the Cholesky factors (C,p) (the memory used for them is "outside" scilab space).

If your matrix is s.p.d. this function must be used in place of umf_lufact or in place of the scilab function chfact for a gain in speed (also as chfact uses the scilab memory for the factors the user must set the stacksize with a large value because of the fill-in occuring in computing the factor C which then may take more memory than the initial matrix A).

When such a factorisation have been computed, a linear system must be solved with taucs_chsolve. **To free the memory used by the Cholesky factors, use taucs_chdel(C_ptr);** to retrieve the Cholesky factors at the scilab level (for example to display their sparse patterns), use taucs_chget; to get some information (number of non zeros in C), use taucs_chinfo. To compute an approximation of the condition number in norm 2 use cond2sp.

Remarks

- taucs_chfact works only with the upper triangle of A, and the matrix A must be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle;
- currently taucs_chfact uses the genmmd (generalized minimum degree) algorithm of Liu to find in a first step the permutation p (so as to minimize the fill-in in the factorization); future versions will let the user choose his/her own reordering by providing a supplementary argument p.

Examples

```
// Example #1 : a small linear test system
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2 -1 0 0 0;
             -1 2 -1 0 0;
              0 -1 2 -1 0;
              0 0 -1 2 -1;
              0 0 0 -1 2] );
b = [0 ; 0; 0; 0; 6];
Cp = taucs_chfact(A);
x = taucs_chsolve(Cp,b)
// don't forget to clear memory with
taucs_chdel(Cp)

// Example #2 a real example
```

```
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cp = taucs_chfact(A);
b = rand(size(A,1),1); // a random rhs
// use taucs_chsolve for solving Ax=b
x = taucs_chsolve(Cp,b);
norm(A*x - b)
// the same with one iterative refinement step
x = taucs_chsolve(Cp,b,A);
norm(A*x - b)
// don't forget to clear memory
taucs_chdel(Cp)
```

See Also

taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chget — retrieve the Cholesky factorization at the scilab level

```
[Ct,p] = taucs_chget(C_ptr)
```

Parameters

C_ptr

a pointer to the Cholesky factorization (C,p : A(p,p)=CC')

Ct

a scilab sparse matrix (you get the upper triangle i.e. Ct is equal to C')

p

column vector storing the permutation

Description

This function may be used if you want to plot the sparse pattern of the Cholesky factorization (or if you code something which use the factors). Traditionnaly, the factorization is written :

$$P A P' = C C'$$

with P' the permutation matrix associated to the permutation p. As we get the upper triangle Ct (= C'), in scilab syntax we can write :

$$A(p,p) = Ct' * Ct$$

Examples

```
// Example #1 : a small linear test system
A = sparse( [ 2 -1 0 0 0;
             -1 2 -1 0 0;
              0 -1 2 -1 0;
              0 0 -1 2 -1;
              0 0 0 -1 2] );
Cp = taucs_chfact(A);
[Ct, p] = taucs_chget(Cp);
full(A(p,p) - Ct'*Ct) // this must be near the null matrix
taucs_chdel(Cp)

// Example #2 a real example
stacksize(3000000) // the last PlotSparse need memory
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cptr = taucs_chfact(A);
// retrieve the factor at scilab level
[Ct, p] = taucs_chget(Cptr);
// plot the initial matrix
xset("window",0) ; clf()
```

```
PlotSparse(A) ; xtitle("Initial matrix A (bcsstk24.rsa)")
// plot the permuted matrix
B = A(p,p);
xset("window",1) ; clf()
PlotSparse(B) ; xtitle("Permuted matrix B = A(p,p)")
// plot the upper triangle Ct
xset("window",2) ; clf()
PlotSparse(Ct) ; xtitle("The pattern of Ct (A(p,p) = C*Ct)")
// retrieve cnz
[OK, n, cnz] = taucs_chinfo(Cptr)
// cnz is superior to the realnumber of non zeros elements of C :
cnz_exact = nnz(Ct)
// don't forget to clear memory
taucs_chdel(Cptr)
```

See Also

taucs_chfact , taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chinfo — get information on Cholesky factors

```
[OK, n, cnz] = taucs_chinfo(C_ptr)
```

Parameters

C_ptr
a pointer to a Cholesky factorization

OK
a scalar boolean

n
a scalar integer

cnz
a scalar integer

Description

This function may be used to know basic information about the Cholesky factor created with `taucs_chfact` :

- first **OK** is %t if **C_ptr** is a valid pointer to an Cholesky factorization (and %f else)
- if **OK** is %t then **n** and **cnz** are respectively the matrix order and the number of non zeros elements in the supernodal structure storing **C** ; if **OK** is %f, **n** and **cnz** are set to the void matrix [].

Details

Due to the supernodal structure used for **C**, **cnz** is larger than the exact number of non-zeros elements in **C** (and so this **cnz** is a mesure of the memory used internally). To get the exact **cnz** you may retrieve the Cholesky factor with `taucs_chget` then apply the `nnz scilab` function (see the 2d example in `taucs_chget`).

See Also

`taucs_chfact` , `taucs_chsolve` , `taucs_chdel` , `taucs_chget`

Authors

taucs by Sivan Toledo (see `taucs_license`)
scilab interface by Bruno Pincon

Name

taucs_chsolve — solve a linear sparse (s.p.d.) system given the Cholesky factors

```
[x] = taucs_chsolve(C_ptr, b [, A])
```

Parameters

C_ptr

a pointer to a handle of the Cholesky factors (C,p with $A(p,p)=CC'$)

b

a real column vector or a matrix (multiple rhs)

x

a real column vector or a matrix in case of multiple rhs ($x(:,i)$ is solution of $A x(:,i) = b(:,i)$)

A

(optional) the real s.p.d. matrix A (to use for iterative refinement step)

Description

This function must be used in conjunction with taucs_chfact which computes the Cholesky factorization of a sparse real s.p.d. matrix. When the matrix A is provided, one iterative refinement step is done (the refined solution is accepted if it improves the 2-norm of the residual $Ax - b$).

Like in taucs_chfact the matrix A may be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle.

Examples

see the example section of taucs_chfact

See Also

taucs_chfact , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)

scilab interface by Bruno Pincon

Name

taucs_license — display the taucs license

Copyright

TAUCS Version 1.0, November 29, 2001. Copyright (c) 2001 by Sivan Toledo, Tel-Aviv Univesity, stoledo@tau.ac.il. All Rights Reserved.

TAUCS License

Your use or distribution of TAUCS or any derivative code implies that you agree to this License. THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY EXPRESSED OR IMPLIED. ANY USE IS AT YOUR OWN RISK. Permission is hereby granted to use or copy this program, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any derivative code must cite the Copyright, this License, the Availability note, and "Used by permission." If this code or any derivative code is accessible from within MATLAB, then typing "help taucs" must cite the Copyright, and "type taucs" must also cite this License and the Availability note. Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. This software is provided to you free of charge.

Availability

<http://www.tau.ac.il/~stoledo/taucs/>

Name

umf_license — display the umfpack license

Copyright

UMFPACK, Copyright (c) 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK is available under alternate licences; contact T. Davis for details.

UMFPACK License

Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included.

Availability

<http://www.cise.ufl.edu/research/sparse>

Name

umf_ludel — utility function used with umf_lufact

```
umf_ludel(LU_ptr) or umf_ludel()
```

Parameters

LU_ptr

a pointer to an handle of umf lu factors (L,U,p,q,R)

Description

This function must be used in conjunction with umf_lufact and umf_lusolve. It clears the internal memory space used to store the LU factors (got with umf_lufact). Use without argument it frees the memory for all the current scilab umfpack LU factors.

Examples

see the example section of umf_lufact

See Also

umfpack , umf_lufact , umf_lusolve , umf_luget , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)

scilab interface by Bruno Pincon

Name

umf_lufact — lu factorisation of a sparse matrix

```
LU_ptr = umf_lufact(A)
```

Parameters

A
a sparse, real or complex, square or rectangular, matrix

LU_ptr
a pointer to umf lu factors (L,U,p,q,R)

Description

This function computes a LU factorisation of the sparse matrix A () and return at the scilab level, a pointer (LU_ptr) to an handle of the LU factors (L,U,p,q,R) (the memory used for them is "outside" scilab stack).

This function must be used in place of umfpack if you have multiple linear systems with the same matrix to solve when the rhs are not known at the same time (for instance $A x_1 = b_1$ and $A x_2 = b_2$ but b_2 depends on x_1 , etc...).

When such a factorisation have been computed, a linear system must be solved with umf_lusolve (in general $x = \text{umf\_lusolve}(\text{LU\_ptr}, b)$ but others options are possible, see umf_lusolve. **To free the memory used by the LU factors, use umf_ludel(LU_ptr) (umf_ludel);** to retrieve the LU factors at the scilab level (for example to display their sparse patterns), use umf_luget; to get some information (number of non zeros in L and U), use umf_luinfo. To compute an approximation of the condition number use condetsp

Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
Lup = umf_lufact(A);
x = umf_lusolve(Lup,b)

// solve now A'x=b
x = umf_lusolve(Lup,b,"A'x=b")
norm(A'*x - b)

// don't forget to clear memory with
umf_ludel(Lup)

// a real (but small) example
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
// compute the factorisation
Lup = umf_lufact(A);
```

```
b = rand(size(A,1),1); // a random rhs
// use umf_lusolve for solving Ax=b
x = umf_lusolve(Lup,b);
norm(A*x - b)

// now the same thing with iterative refinement
x = umf_lusolve(Lup,b,"Ax=b",A);
norm(A*x - b)

// solve now the system A'x=b
x = umf_lusolve(Lup,b,"A'x=b"); // without refinement
norm(A'*x - b)
x = umf_lusolve(Lup,b,"A'x=b",A); // with refinement
norm(A'*x - b)

// don't forget to clear memory
umf_ludel(Lup)
```

See Also

umfpack , umf_luget , umf_lusolve , umf_ludel , umf_luinfo , condestsp

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

Name

umf_luget — retrieve lu factors at the scilab level

```
[L,U,p,q,Rd] = umf_luget(LU_ptr)
```

Parameters

LU_ptr

a pointer to umf lu factors (L,U,p,q,R)

L,U

scilab sparse matrix

p,q

column vectors storing the permutations

Rd

vector storing the (row) scaling factors

Description

This function may be used if you want to plot the sparse pattern of the lu factors (or if you code something which use the lu factors). The factorization provided by umfpack is of the form:

$$P R^{(-1)} A Q = LU$$

where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix. The function provides the matrices L and U as Sparse scilab matrices but P and Q are given as permutation vectors p and q (in fact p is the permutation associated to P') and Rd is the vector corresponding to the diagonal of R.

Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
B = A;
for i=1:5, B(i,:) = B(i,+)/R(i); end // apply the row scaling
B(p,q) - L*U // must be a (quasi) nul matrix

umf_ludel(Lup) // clear memory

// the same with a complex matrix
A = sparse( [ 2+%i  3+2*%i  0  0  0;
              3-%i  0  4+%i  0  6-3*%i;
              0  -1+%i  -3+6*%i  2-%i  0;
              0  0  1-5*%i  0  0;
              0  4  2-%i  0  1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
```

```
B = A;  
for i=1:5, B(i,:) = B(i,+)/R(i); end // apply the row scaling  
B(p,q) - L*U // must be a (quasi) nul matrix  
  
umf_ludel(Lup) // clear memory
```

See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

Name

umf_luinfo — get information on LU factors

```
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(LU_ptr)
```

Parameters

LU_ptr

a pointer to umf lu factors (L,U,p,q, R)

OK

a scalar boolean

nrow, ncol, lnz, unz, udiag_nz, it

scalars (integers)

Description

This function may be used to know basic information about LU factors created with umf_lufact :

first OK is %t if LU_ptr is a valid pointer to an umfpack LU numeric handle (and %f else)

if OK is %t then:

nrow, ncol

are the matrix size (L is nrow x n and U is n x ncol where $n = \min(\text{nrow}, \text{ncol})$)

lnz, unz

are the number of non zeros elements in L and in U;

udiag_nz

are the number of non zeros elements on the diagonal of U; if the matrix is square ($\text{nrow} = \text{ncol} = n$) then it is not inversible if $\text{udiag\_nz} < n$ (more precisely it appears to be numerically not inversible through the LU factorization).

it

0 if the factors are real and 1 if they are complex.

if OK is %f then all the others outputs are set to the empty matrix [].

Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(Lup) // OK must be %t, n
[L,U,p,q,R] = umf_luget(Lup);
nnz(L) // must be equal to lnz
nnz(U) // must be equal to unz
umf_ludel(Lup) // clear memory
```

See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luget

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

Name

umf_lusolve — solve a linear sparse system given the LU factors

```
[x] = umf_lusolve(LU_ptr, b [, st, A])
```

Parameters

LU_ptr

a pointer to umf lu factors (L,U,p,q,R)

b

a real or complex column vector or a matrix (multiple rhs)

st

(optional) a string "Ax=b" (default) or "Ax'=b" (to be written "Ax"=b" in scilab langage: a quote in a string must be doubled !)

A

(optional) the sparse square matrix corresponding to the LU factors (LU_ptr must be got with LU_ptr = umf_lufact(A))

x

a column vector or a matrix in case of multiple rhs (x(:,i) is solution of A x(:,i) = b(:,i) or A'x(:,i) = b(:,i))

Description

This function must be used in conjunction with umf_lufact which computes the LU factors of a sparse matrix. The optional st argument lets us choose between the solving of Ax=b (general case) or of A'x=b (sometimes useful). If you give the 4th argument then iterative refinement will be also proceeded (as in umfpack) to give a better numerical solution.

Examples

see the example section of umf_lufact

See Also

umfpack , umf_lufact , umf_luget , umf_ludel , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)

scilab interface by Bruno Pincon with contributions from Antonio Frasson

Name

umfpack — solve sparse linear system

```
x = umfpack(A, "\", b)
x = umfpack(b, "/", A)
```

Parameters

- A
a sparse (real or complex) square matrix $n \times n$
- b
in the first case, a column vector ($n \times 1$) or a $n \times m$ matrix ; in the second case, a row vector ($1 \times n$) or a $m \times n$ matrix
- x
in the first case , a column vector ($n \times 1$) or a $n \times m$ matrix ; in the second case, a row vector ($1 \times n$) or a $m \times n$ matrix
- 2d arg
string specifier "\" or "/"

Description

This function is intended to work like the classic operators $\backslash$ and $/$ $x = A \backslash b$ and $x = b / A$ i.e. it solves a linear system $Ax = b$ or $xA = b$ with a sparse square (says $n \times n$) real or complex matrix and with a compatible rhs $b : n \times m$ in the first case and $m \times n$ in the second.

Details

First an LU factorisation of the matrix is computed ($P R^{(-1)} A Q = LU$ where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix) then a first solution is computed with forward/backward substitutions ; finally the solution is improved by iterative refinement.

Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
x = umfpack(A, "\", b)

// test the other form x A = b
b = [8 20 13 6 17];
x = umfpack(b, "/", A) // solution must be [1 2 3 4 5]

// test multiple rhs
b = rand(5,3);
x = umfpack(A, "\", b)
norm(A*x - b)
```

```
// test multiple rhs for x A = b
b = rand(3,5);
x = umfpack(b,"/",A)
norm(x*A - b)

// solve a complex system
A = sparse( [ 2+%i  3+2*i  0      0      0;
              3-%i  0      4+%i  0      6-3*i;
              0     -1+%i  -3+6*i  2-%i  0;
              0      0      1-5*i  0      0;
              0      4      2-%i  0      1] );
b = [ 3+13*i ; 58+32*i ; -19+13*i ; 18-12*i ; 22+16*i ];
x = umfpack(A,"\",b) // x must be [1+i; 2+2i; 3+3i; 4 + 4i; 5+5i]

// A benchmark of several linear solvers

[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa")

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\ ',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

See Also

[umf_lufact](#) , [umf_lusolve](#) , [umf_ludel](#) , [umf_luinfo](#) , [umf_luget](#) , [linsolve](#) , [backslash](#)

Authors

umfpack by Timothy A. Davis (see [umf_license](#))
scilab interface by Bruno Pincon with contributions from Antonio Frasson

Part L. Genetic Algorithms

Name

`coding_ga_binary` — A function which performs conversion between binary and continuous representation

```
pop_out = coding_ga_binary(pop_in,direction,param)
```

Parameters

`pop_in`

a list which contains all the individuals in the current population.

`direction`

'code' or 'decode'. If `direction == 'code'` then we perform a continuous to binary encoding. Else, we convert from binary to continuous.

`param`

a parameter list.

- 'binary_length': the number of bits by variables. If `binary_length = 8` and the variable `X` is of dimension 2 then the binary code will be 16 bits length.
- 'minbound': a vector of minimum bounds for the variable `X`.
- 'maxbound': a vector of maximum bounds for the variable `X`.

`pop_out`

the population coded to binary or decoded to continuous values.

Description

- This function allows to code or decode a population of individuals from (resp. to) continuous variables to (resp. from) binary.

See Also

`optim_ga` , `mutation_ga_binary` , `crossover_ga_binary`

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

`coding_ga_identity` — A "no-operation" conversion function

```
pop_out = coding_ga_identity(pop_in,direction,param)
```

Parameters

`pop_in`

the population to be converted.

`direction`

'code' or 'decode'. This value has no influence of the state of `pop_in`.

`param`

a parameter list. For this function, there are no useful parameters set.

`pop_out`

a population identical to `pop_in`.

Description

- This function is a do-nothing function. It is essentially useful to implement an evolutionary algorithm. In an evolutionary algorithm, we work directly on the variable and not on a binary code.

See Also

`mutation_func_default` , `crossover_func_default` , `init_func_default` , `optim_ga`

Authors

Yann COLLETTE

ycollet@freesurf.fr

Name

crossover_ga_binary — A crossover function for binary code

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_binary(Indiv1,Indiv2,param)
```

Parameters

Indiv1

the first individual (here a binary code) to be crossed-over.

Indiv2

the second individual to be crossed-over.

param

a list of parameters.

- 'binary_length': the length of the binary code.
- 'multi_cross': a boolean. If %T then we allow several cuts in the binary code.
- 'multi_cross_nb': the number of cuts in the binary code. Only used when multi_cross is set to %T.

Crossed_Indiv1

The first individual obtained by the cross-over function.

Crossed_Indiv2

The second individual obtained by the cross-over function.

Description

- This function implements a classical binary cross-over.

See Also

crossover_ga_binary , crossover_ga_default , mutation_ga_binary , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

crossover_ga_default — A crossover function for continuous variable functions

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_default(Indiv1,Indiv2,param)
```

Parameters

Indiv1

The first individual to be crossed-over.

Indiv2

The second individual to be crossed-over.

param

a list of parameters.

- 'beta': the range of the random generator. A random value will be sampled between -beta and 1+beta. This sampled value will be used to perform a convex combination between Indiv1 and Indiv2.
- 'minbound': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

Crossed_Indiv1

The first individual resulting from the crossover.

Crossed_Indiv2

The second individual resulting from the crossover.

Description

crossover_ga_default is a crossover function for functions with continuous variables. This crossover function is an extension of a convex combination. The crossed individuals are computed with the following equations :

```
mix = (1 + 2*Beta)*rand(1,1) - Beta;  
Crossed_Indiv1 = mix*Indiv1 + (1-mix)*Indiv2;  
Crossed_Indiv2 = (1-mix)*Indiv1 + mix*Indiv2;
```

The Beta parameter should be set to a positive value. If Beta is set to 0, the resulting crossover is a simple convex combination between the two parents. That may lead to a too fast convergence of the genetic algorithm and may decrease the diversity of the individuals of the population. If Beta is chosen strictly positive, that may allow children to explore the domain beyond the domain explored by their parents.

See Also

crossover_ga_binary , mutation_ga_default , init_ga_default , optim_ga

References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

init_ga_default — A function a initialize a population

```
Pop_init = init_ga_default(popsiz, param)
```

Parameters

popsiz

the number of individuals to generate.

param

a list of parameters.

- 'dimension': the size of the vector X.
- 'minbound': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

Pop_init

a list which contains the initial population of individuals.

Description

- This function generate an initial population containing pop_size individuals.

See Also

crossover_ga_default , mutation_ga_default , mutation_ga_binary , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

mutation_ga_binary — A function which performs binary mutation

```
Mut_Indiv = mutation_ga_binary(Indiv,param)
```

Parameters

Indiv

the individual on which we will perform the mutation.

param

a list of parameters.

- 'binary_length': the size of the binary code.
- 'multi_mut': a boolean. If %T, several random bits will be flipped.
- 'multi_mut_nd': the number of bits to be flipped. Works only when multi_mut is set to %T.

Mut_Indiv

The mutated individual.

Description

- This function performs a classical multi-bits binary mutation.

See Also

mutation_ga_default , crossover_ga_binary , init_func_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

mutation_ga_default — A continuous variable mutation function

```
Mut_Indiv = mutation_ga_default(Indiv,param)
```

Parameters

Indiv

The individual to be mutated.

param

a list of parameters.

- 'delta': a random perturbation will be sampled via an uniform distribution between -delta and + delta.
- 'minbound': a vector of minimum bound for the variable X.
- 'maxbound': a vector of maximum bound for the variable X.

Mut_Indiv

The resulting mutated individual.

Description

- This function performs the classical continuous variable mutation function.

See Also

mutation_ga_binary , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

optim_ga — A flexible genetic algorithm

```
[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_ga(ga_f, pop_size, nb_generations, param)
```

Parameters

ga_f

the function to be optimized. The prototype is $y = f(x)$ or $y = \text{list}(f, p_1, p_2, \dots)$.

pop_size

the size of the population of individuals (default value: 100).

nb_generation

the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut

the mutation probability (default value: 0.1).

p_cross

the crossover probability (default value: 0.7).

Log

if %T, we will display an information message during the run of the genetic algorithm.

param

a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: `codage_identity`).
- 'init_func': the function which will perform the initialization of the population (default function: `init_ga_default`).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: `crossover_ga_default`).
- 'mutation_func': the function which will perform the mutation of one individual (default function: `mutation_ga_default`).
- 'selection_func': the function which will perform the selection of individuals at the end of a generation (default function: `selection_ga_elitist`).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value of the efficiency of the worst individual (default value: 0.05).

pop_opt

the population of optimal individuals.

fobj_pop_opt

the set of objective function values associated to `pop_opt` (optional).

pop_init

the initial population of individuals (optional).

fobj_pop_init

the set of objective function values associated to `pop_init` (optional).

Description

- This function implements the classical genetic algorithm.

Examples

```
deff('y=f(x)', 'y = sum(x.^2)');

PopSize      = 100;
Proba_cross  = 0.7;
Proba_mut    = 0.1;
NbGen        = 10;
NbCouples    = 110;
Log          = %T;
nb_disp      = 10; // Nb point to display from the optimal population
pressure     = 0.05;

ga_params = init_param();
// Parameters to adapt to the shape of the optimization problem
ga_params = add_param(ga_params, 'minbound', [-2; -2]);
ga_params = add_param(ga_params, 'maxbound', [2; 2]);
ga_params = add_param(ga_params, 'dimension', 2);
ga_params = add_param(ga_params, 'beta', 0);
ga_params = add_param(ga_params, 'delta', 0.1);
// Parameters to fine tune the Genetic algorithm. All these parameters are optional
// If you need to adapt the GA to a special problem, you
ga_params = add_param(ga_params, 'init_func', init_ga_default);
ga_params = add_param(ga_params, 'crossover_func', crossover_ga_default);
ga_params = add_param(ga_params, 'mutation_func', mutation_ga_default);
ga_params = add_param(ga_params, 'codage_func', codage_ga_identity);
ga_params = add_param(ga_params, 'selection_func', selection_ga_elitist);
//ga_params = add_param(ga_params, 'selection_func', selection_ga_random);
ga_params = add_param(ga_params, 'nb_couples', NbCouples);
ga_params = add_param(ga_params, 'pressure', pressure);

Min = get_param(ga_params, 'minbound');
Max = get_param(ga_params, 'maxbound');
x0 = (Max - Min) .* rand(size(Min,1),size(Min,2)) + Min;

[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_ga(f, PopSize, NbGen, 1, 1, x0, ga_params, nb_disp, pressure);
```

See Also

[optim_moga](#) , [optim_nsga](#) , [optim_nsga2](#)

References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

optim_moga — multi-objective genetic algorithm

```
[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_moga(ga_f, pop_size, nb_gen)
```

Parameters

ga_f

the function to be optimized. The header of the function is the following :

```
y = f(x)
```

or

```
y = list(f, p1, p2, ...)
```

pop_size

the size of the population of individuals (default value: 100).

nb_generation

the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut

the mutation probability (default value: 0.1).

p_cross

the crossover probability (default value: 0.7).

Log

if %T, we will display to information message during the run of the genetic algorithm.

param

a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function which will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
the population of optimal individuals.

fobj_pop_opt
the set of multi-objective function values associated to pop_opt (optional).

pop_init
the initial population of individuals (optional).

fobj_pop_init
the set of multi-objective function values associated to pop_init (optional).

Description

- This function implements the classical "Multi-Objective Genetic Algorithm". For a demonstration: see `SCI/modules/genetic_algorithms/examples/MOGAdemo.sce`.

See Also

optim_ga , optim_nsga , optim_nsga2

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

optim_nsga — A multi-objective Niche Sharing Genetic Algorithm

```
[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_nsga(ga_f, pop_size, nb_gen)
```

Parameters

ga_f

the function to be optimized. The prototype if $y = f(x)$ or $y = \text{list}(f, p1, p2, \dots)$.

pop_size

the size of the population of individuals (default value: 100).

nb_generation

the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut

the mutation probability (default value: 0.1).

p_cross

the crossover probability (default value: 0.7).

Log

if %T, we will display to information message during the run of the genetic algorithm.

param

a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: `codage_identity`).
- 'init_func': the function which will perform the initialization of the population (default function: `init_ga_default`).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: `crossover_ga_default`).
- 'mutation_func': the function which will perform the mutation of one individual (default function: `mutation_ga_default`).
- 'selection_func': the function which will perform the selection of individuals at the end of a generation (default function: `selection_ga_elitist`).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

sigma

the radius of the sharing area.

pow

the power coefficient of the penalty formula.

pop_opt

the population of optimal individuals.

fobj_pop_opt

the set of objective function values associated to pop_opt (optional).

pop_init

the initial population of individuals (optional).

fobj_pop_init

the set of objective function values associated to pop_init (optional).

Description

- This function implements the classical "Niche Sharing Genetic Algorithm". For a demonstration, see SCI/modules/genetic_algorithms/examples/NSGAdemo.sce.

See Also

optim_moga , optim_ga , optim_nsga2

Authors

Yann COLLETTE

ycollet@freesurf.fr

Name

optim_nsga2 — A multi-objective Niche Sharing Genetic Algorithm version 2

```
[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_nsga2(ga_f, pop_size, nb_ge
```

Parameters

ga_f

the function to be optimized. The prototype is $y = f(x)$ or $y = \text{list}(f, p_1, p_2, \dots)$.

pop_size

the size of the population of individuals (default value: 100).

nb_generation

the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut

the mutation probability (default value: 0.1).

p_cross

the crossover probability (default value: 0.7).

Log

if %T, we will display an information message during the run of the genetic algorithm.

param

a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: `codage_identity`).
- 'init_func': the function which will perform the initialization of the population (default function: `init_ga_default`).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: `crossover_ga_default`).
- 'mutation_func': the function which will perform the mutation of one individual (default function: `mutation_ga_default`).
- 'selection_func': the function which will perform the selection of individuals at the end of a generation (default function: `selection_ga_elitist`).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value of the efficiency of the worst individual (default value: 0.05).

pop_opt

the population of optimal individuals.

fobj_pop_opt

the set of objective function values associated to `pop_opt` (optional).

pop_init

the initial population of individuals (optional).

fobj_pop_init

the set of objective function values associated to `pop_init` (optional).

Description

- This function implements the classical "Niche Sharing Genetic Algorithm". For a demonstration, see `SCI/modules/genetic_algorithms/examples/NSGA2demo.sce`.

See Also

`optim_moga` , `optim_ga` , `optim_nsga`

Authors

Yann COLLETTE
`ycollet@freesurf.fr`

Name

pareto_filter — A function which extracts non dominated solution from a set

```
[F_out,X_out,Ind_out] = pareto_filter(F_in,X_in)
```

Parameters

F_in

the set of multi-objective function values from which we want to extract the non dominated solutions.

X_in

the associated values in the parameters space.

F_out

the set of non dominated multi-objective function values.

X_out

the associated values in the parameters space.

Ind_out

the set of indexes of the non dominated individuals selected from the set X_in.

Description

- This function applies a Pareto filter to extract non dominated solutions from a set of values.

See Also

optim_moga , optim_nsga , optim_nsga2

Authors

Yann COLLETTE
ycollet@freesurf.Fr

Name

selection_ga_elitist — An 'elitist' selection function

```
[Pop_out, FObj_Pop_out, Efficiency, MO_Total_FObj_out] = selection_ga_elitist(Pop_in,
```

Parameters

Pop_in

The initial population of individuals.

Indiv1

a first set of childs generated via crossover + mutation.

Indiv2

a second set of childs generated via crossover + mutation.

FObj_Pop_in

a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1

a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2

a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in

a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1

a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2

a matrix of multi-objective function values associated to each individuals of Indiv2.

param

a list of parameters. - 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out

all the selected individuals in a population of size pop_size.

FObj_Pop_out

all the objective function values associated to each individuals of Pop_out.

Efficiency

all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out

all the multi-objective function values associated to each individuals of Pop_out.

Description

- This function performs the elitist selection function. We select the best individuals in the set of parents and childs individuals.

See Also

selection_ga_random , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

selection_ga_random — A function which performs a random selection of individuals

```
[Pop_out, FObj_Pop_out, Efficiency, MO_Total_FObj_out] = selection_ga_random(Pop_in,
```

Parameters

Pop_in

The initial population of individuals.

Indiv1

a first set of childs generated via crossover + mutation.

Indiv2

a second set of childs generated via crossover + mutation.

FObj_Pop_in

a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1

a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2

a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in

a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1

a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2

a matrix of multi-objective function values associated to each individuals of Indiv2.

param

a list of parameters.

- 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out

all the selected individuals in a population of size pop_size.

FObj_Pop_out

all the objective function values associated to each individuals of Pop_out.

Efficiency

all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out

all the multi-objective function values associated to each individuals of Pop_out.

Description

- This function performs the random selection function. We select pop_size individuals in the set of parents and childs individuals at random.

See Also

selection_ga_elitist , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Part LI. Simulated Annealing

Name

`compute_initial_temp` — A SA function which allows to compute the initial temperature of the simulated annealing

```
T_init = compute_initial_temp(x0,f,proba_init,ItMX,neigh_func,param_neigh_func)
```

Parameters

`x0`
the starting point

`f`
the objective function which will be send to the simulated annealing for optimization

`proba_init`
the initial probability of accepting a bad solution (usually around 0.7)

`ItMX`
the number of iterations of random walk (usually around 100)

`neigh_func`
a function which returns a neighbor of a given point (see the help page of `neigh_func` to see the prototype of this function)

`param_neigh_func`
some parameters (can be a list) which will be sent as parameters to `neigh_func`

`T_init`
The initial temperature corresponding to the given probability of accepting a bad solution

Description

- This function computes an initial temperature given an initial probability of accepting a bad solution. This computation is based on some iterations of random walk.

Examples

```
deff('y=f(x)', 'y=sum(x.^2)');

x0 = [2 2];
Proba_start = 0.7;
It_Pre = 100;
x_test = neigh_func_default(x0);

comp_t_params = init_param();
comp_t_params = add_param(comp_t_params, 'neigh_func', neigh_func_default);

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, comp_t_sa);
```

See Also

`optim_sa`, `neigh_func_default`, `temp_law_default`

Authors

collette
Yann COLLETTE (ycollet@freesurf.fr)

Name

`neigh_func_csa` — The classical neighborhood relationship for the simulated annealing

```
x_neigh = neigh_func_csa(x_current, T, param)
```

Parameters

`x_current`

the point for which we want to compute a neighbor

`T`

the current temperature

`param`

a vector with the same size than `x_current`. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

`x_neigh`

the computed neighbor

Description

- This function implements the classical neighborhood relationship for the simulated annealing. The neighbors distribution is a gaussian distribution which is more and more peaked as the temperature decrease.

See Also

`neigh_func_default` , `temp_law_huang` , `optim_sa`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

`neigh_func_default` — A SA function which computes a neighbor of a given point

```
x_neigh = neigh_func_default(x_current, T, param)
```

Parameters

`x_current`

the point for which we want to compute a neighbor

`T`

the current temperature

`param`

a two columns vector. The first column correspond to the negative amplitude of variation and the second column corresponds to the positive amplitude of variation of the neighborhood. By default, the first column is a column of -0.1 and the second column is a column of 0.1.

`x_neigh`

the computed neighbor

Description

- This function computes a neighbor of a given point. For example, for a continuous vector, a neighbor will be produced by adding some noise to each component of the vector. For a binary string, a neighbor will be produced by changing one bit from 0 to 1 or from 1 to 0.

Examples

```
// We produce a neighbor by adding some noise to each component of a given vect  
function x_neigh = neigh_func_default(x_current, T)  
    sa_min_delta = -0.1*ones(size(x_current,1),size(x_current,2));  
    sa_max_delta = 0.1*ones(size(x_current,1),size(x_current,2));  
    x_neigh = x_current + (sa_max_delta - sa_min_delta).*rand(size(x_current,1),s  
endfunction
```

See Also

`optim_sa` , `compute_initial_temp` , `temp_law_default`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_fsa — The Fast Simulated Annealing neighborhood relationship

```
x_neigh = neigh_func_fsa(x_current, T, param)
```

Parameters

x_current

the point for which we want to compute a neighbor

T

the current temperature

param

a vector with the same size than x_current. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

x_neigh

the computed neighbor

Description

- This function computes the FSA neighborhood of a given point. The corresponding distribution is a Cauchy distribution which is more and more peaked as the temperature decrease.

See Also

optim_sa , temp_law_fsa , neigh_func_default

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_vfsa — The Very Fast Simulated Annealing neighborhood relationship

```
x_neigh = neigh_func_vfsa(x_current, T, param)
```

Parameters

x_current

the point for which we want to compute a neighbor

T

the current temperature

param

a ones column vector. The column correspond to the amplitude of variation of the neighborhood. By default, the column is a column of 0.1.

x_neigh

the computed neighbor

Description

- This function implements the Very Fast Simulated Annealing relationship. This distribution is more and more peaked as the temperature decrease.

See Also

optim_sa , neigh_func_vfsa , temp_law_huang

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

optim_sa — A Simulated Annealing optimization method

```
[x_best, f_best, mean_list, var_list, f_history, temp_list, x_history] = optim_sa(x0,
```

Parameters

- x0**
the initial solution
- f**
the objective function to be optimized (the prototype if $f(x)$)
- ItExt**
the number of temperature decrease
- ItInt**
the number of iterations during one temperature stage
- T0**
the initial temperature (see `compute_initial_temp` to compute easily this temperature)
- Log**
if %T, some information will be displayed during the run of the simulated annealing
- temp_law**
the temperature decrease law (see `temp_law_default` for an example of such a function)
- param_temp_law**
a structure (of any kind - it depends on the temperature law used) which is transmitted as a parameter to `temp_law`
- neigh_func**
a function which computes a neighbor of a given point (see `neigh_func_default` for an example of such a function)
- param_neigh_func**
a structure (of any kind like vector, list, it depends on the neighborhood function used) which is transmitted as a parameter to `neigh_func`
- x_best**
the best solution found so far
- f_best**
the objective function value corresponding to `x_best`
- mean_list**
the mean of the objective function value for each temperature stage. A vector of float (optional)
- var_list**
the variance of the objective function values for each temperature stage. A vector of float (optional)
- f_history**
the computed objective function values for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of float which gathers all the objective function values computed during the corresponding temperature stage - (optional)
- temp_list**
the list of temperature computed for each temperature stage. A vector of float (optional)

x_history

the parameter values computed for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of input variables which corresponds to all the variables computed during the corresponding temperature stage - (optional - can slow down a lot the execution of optim_sa)

Description

- A Simulated Annealing optimization method.

Examples

```
function y = rastrigin(x)
    y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0          = [2 2];
Proba_start = 0.7;
It_Pre      = 100;
It_extern   = 100;
It_intern   = 1000;
x_test = neigh_func_default(x0);

comp_t_params = init_param();
comp_t_params = add_param(comp_t_params, 'neigh_func', neigh_func_default);

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, comp_t_sa);

[x_opt, f_opt, sa_mean_list, sa_var_list] = optim_sa(x0, rastrigin, It_extern,

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

t = 1:length(sa_mean_list);
plot(t, sa_mean_list, 'r', t, sa_var_list, 'g');
```

See Also

compute_initial_temp , neigh_func_default , temp_law_default

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_csa — The classical temperature decrease law

```
T_out = temp_law_csa(T_in, step_mean, step_var, temp_stage, n, param)
```

Parameters

T_in

the temperature at the current stage

step_mean

the mean value of the objective function computed during the current stage

step_var

the variance value of the objective function computed during the current stage

temp_stage

the index of the current temperature stage

n

the dimension of the decision variable (the x in $f(x)$)

param

not used for this temperature law

T_out

the temperature for the temperature stage to come

Description

- This function implements the classical annealing temperature schedule (the one for which the convergence of the simulated annealing has been proven).

Examples

```
function y = rastrigin(x)
    y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre = 100;

printf('SA: the CSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_default);
printf('Initial temperature T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, T0, It_extern, It_intern, Proba_start, neigh_func_default);

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Classical simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

`optim_sa` , `temp_low_huang` , `neigh_func_default`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_default — A SA function which computed the temperature of the next temperature stage

```
T_next = temp_law_default(T, step_mean, step_var, temp_stage, n, param)
```

Parameters

- T
the temperature applied during the last temperature stage
- step_mean
the mean of the objective function values computed during the last temperature stage
- step_var
the variance of the objective function values computed during the last temperature stage
- temp_stage
the index of the current temperature stage
- n
the dimension of the decision variable (the x in $f(x)$)
- param
a float between 0 and 1. Corresponds to the decrease in temperature of the geometric law (0.9 by default)
- T_next
the new temperature to be applied for the next temperature stage

Description

- A SA function which computed the temperature of the next temperature stage

Examples

```
// This function implements the simple geometric temperature law
function T = temp_law_default(T, step_mean, step_var)
    _alpha = 0.9;
    T = _alpha*T;
endfunction
```

See Also

optim_sa , compute_initial_temp , neigh_func_default

Authors

collette
Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_fsa — The Szu and Hartley Fast simulated annealing

```
T_out = temp_law_fsa(T_in, step_mean, step_var, temp_stage, n, param)
```

Parameters

T_in
the temperature at the current stage

step_mean
the mean value of the objective function computed during the current stage

step_var
the variance value of the objective function computed during the current stage

temp_stage
the index of the current temperature stage

n
the dimension of the decision variable (the x in $f(x)$)

param
not used for this temperature law

T_out
the temperature for the temperature stage to come

Description

- This function implements the Fast simulated annealing of Szu and Hartley.

Examples

```
function y = rastrigin(x)
    y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre = 100;

printf('SA: the FSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_default);
printf('Initial temperature T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, T0, It_extern, It_intern, Proba_start, neigh_func_default);

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('Fast simulated annealing','Iteration','Mean / Variance');  
t = 1:length(sa_mean_list);  
plot(t,sa_mean_list,'r',t,sa_var_list,'g');  
legend(['Mean','Variance']);  
subplot(2,1,2);  
xtitle('Temperature evolution','Iteration','Temperature');  
plot(t,temp_list,'k-');
```

See Also

`optim_sa`, `temp_low_huang`, `neigh_func_default`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_huang — The Huang temperature decrease law for the simulated annealing

```
T_out = temp_law_huang(T_in, step_mean, step_var, temp_stage, n, param)
```

Parameters

T_in

the temperature at the current stage

step_mean

the mean value of the objective function computed during the current stage

step_var

the variance value of the objective function computed during the current stage

temp_stage

the index of the current temperature stage

n

the dimension of the decision variable (the x in $f(x)$)

param

a float corresponding to the lambda parameter of the Huang temperature decrease law (0.01 by default)

T_out

the temperature for the temperature stage to come

Description

- This function implements the Huang temperature decrease law for the simulated annealing.

Examples

```
function y = rastrigin(x)
    y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre = 100;

printf('SA: the Huang temperature decrease law\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_default);
printf('Initial temperature T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, T0, It_extern, It_intern, Proba_start, neigh_func_default);

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Huang simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

`optim_sa` , `temp_low_csa` , `neigh_func_csa`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_low_vfsa — This function implements the Very Fast Simulated Annealing from L. Ingber

```
T_out = temp_low_vfsa(T_in, step_mean, step_var, temp_stage, n, param)
```

Parameters

T_in

the temperature at the current stage

step_mean

the mean value of the objective function computed during the current stage

step_var

the variance value of the objective function computed during the current stage

temp_stage

the index of the current temperature stage

n

the dimension of the decision variable (the x in f(x))

param

a float: the 'c' parameter of the VFSA method (0.01 by default)

T_out

the temperature for the temperature stage to come

Description

- This function implements the Very Fast Simulated Annealing from L. Ingber.

Examples

```
function y = rastrigin(x)
    y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre = 100;

printf('SA: the VFSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_default);
printf('Initial temperature T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, T0, It_extern, It_intern, Proba_start, neigh_func_default);

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('VFSA simulated annealing','Iteration','Mean / Variance');  
t = 1:length(sa_mean_list);  
plot(t,sa_mean_list,'r',t,sa_var_list,'g');  
legend(['Mean','Variance']);  
subplot(2,1,2);  
xtitle('Temperature evolution','Iteration','Temperature');  
plot(t,temp_list,'k-');
```

See Also

`optim_sa`, `neigh_func_vfsa`, `temp_low_huang`

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Part LII. Parameters

Name

add_param — Add a parameter to a list of parameters

```
[param_list,err] = add_param(list_name,param_name,param_value)
```

Parameters

list_name

the list of parameters. This list must have been initialize by a call to init_param.

param_name

a string. The name of the parameter to be added in the list of parameters.

param_value

the value associated to the parameter param_name. This parameter is optional. You can set the value of this parameter via a call to set_param.

param_list

the updated list of parameters.

err

an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

This function creates a new parameter in a list of parameters. You can set the value of the parameter using this function or you can set it via a call to set_param.

Examples

```
mylist = init_param();  
mylist = add_param(mylist,'minbound',[0 0 0]);
```

See Also

init_param , set_param , get_param , remove_param , is_param

Authors

collette

ycollet@freesurf.fr

Name

get_param — Get the value of a parameter in a parameter list

```
[res,err] = get_param(list_name,param_name,default_value)
```

Parameters

list_name

the list of parameters. This list must have been initialized by a call to init_param.

param_name

a string. The name of the parameter to be add in the list of parameters.

default_value

the default value to be stored in the parameter if param_name has not been found.

res

the value of the parameter. If the parameter doesn't exist, res = [].

err

an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

This function returns the value of the parameter param_name in a parameter list. If the err output parameter is not present, when an error occurs, a message is printed in the console.

Examples

```
mylist = init_param();  
mylist = add_param(mylist,'minbound',[0 0 0]);  
disp(get_param(mylist,'minbound',-[1 1 1]));  
disp(get_param(mylist,'maxbound',[1 1 1]));
```

See Also

init_param , set_param , add_param , remove_param , is_param

Authors

collette

ycollet@freesurf.fr

Name

`init_param` — Initialize the structure which will handles the parameters list

```
param_list = init_param()
```

Parameters

`param_list`

an initialized list of parameters (this list is empty and is of type `plist`).

Description

This function initialize an empty list of parameters. You must initialize the list of parameters before using it.

Examples

```
mylist = init_param();  
mylist = add_param(mylist, 'minbound', [0 0 0]);
```

See Also

`add_param` , `set_param` , `get_param` , `remove_param` , `is_param`

Authors

collette

ycollet@freesurf.fr

Name

is_param — Check if a parameter is present in a parameter list

```
[res,err] = is_param(list_name,param_name)
```

Parameters

list_name

the list of parameters. This list must have been initialize by a call to init_param.

param_name

a string. The name of the parameter to be add in the list of parameters.

res

the result: %T is the parameter is present, %F otherwise.

err

an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

This function checks if a parameter is present in a parameter list. If the err output parameter is not present, when an error occurs, a message is printed in the console.

Examples

```
mylist = init_param();  
mylist = add_param(mylist,'minbound',[0 0 0]);  
disp(is_param(mylist,'minbound'));  
disp(is_param(mylist,'maxbound'));
```

See Also

init_param , set_param , get_param , remove_param , add_param

Authors

collette

ycollet@freesurf.fr

Name

list_param — List all the parameters name in a list of parameters

```
[string_list,err] = list_param(list_name)
```

Parameters

list_name

the list of parameters. This list must have been initialize by a call to init_param.

string_list

the list of parameters name.

err

an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

List all the parameters name in a list of parameters. If the err output parameter is not present, when an error occurs, a message is printed in the console.

Examples

```
mylist = init_param();  
mylist = add_param(mylist, 'minbound', [0 0 0]);  
mylist = add_param(mylist, 'maxbound', [1 1 1]);  
disp(list_param(mylist));
```

See Also

init_param , set_param , get_param , remove_param , is_param

Authors

collette

ycollet@freesurf.fr

Name

`remove_param` — Remove a parameter and its associated value from a list of parameters

```
[param_list,err] = remove_param(list_name,param_name)
```

Parameters

`list_name`

the list of parameters. This list must have been initialize by a call to `init_param`.

`param_name`

a string. The name of the parameter to be removed from the list of parameters. If the parameter doesn't exist, nothing happens.

`param_list`

the updated list of parameters.

`err`

an error flag which is set to %T if `list_name` is not of type plist (this list hasn't been initialized by a call to `init_param`).

Description

This function allows to remove a parameter and its associated value from a list of parameters. If the `err` output parameter is not present, when an error occurs, a message is printed in the console.

Examples

```
mylist = init_param();  
mylist = add_param(mylist,'minbound',[0 0 0]);  
mylist = add_param(mylist,'maxbound',[0 0 0]);  
mylist = remove_param(mylist,'minbound');
```

See Also

`init_param` , `set_param` , `get_param` , `add_param` , `is_param`

Authors

collette

ycollet@freesurf.fr

Name

set_param — Set the value of a parameter in a parameter list

```
[param_list,err] = set_param(list_name,param_name,param_value)
```

Parameters

list_name

the list of parameters. This list must have been initialize by a call to init_param.

param_name

a string. The name of the parameter to be added in the list of parameters.

param_value

the value to be associated to the parameter param_name.

param_list

the updated list of parameters.

err

an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

This function sets the value of an already existing parameter. If the parameter doesn't exist, err is set to %T. If the err output parameter is not present, when an error occurs, a message is printed in the console.

Examples

```
mylist = init_param();  
mylist = add_param(mylist,'minbound',[0 0 0]);  
[mylist,err] = set_param(mylist,'minbound',[1 1 1]); disp(err);  
[mylist,err] = set_param(mylist,'maxbound',[1 1 1]); disp(err);
```

See Also

init_param , add_param , get_param , remove_param , is_param

Authors

collette

ycollet@freesurf.fr

Part LIII. Atoms

Name

Getting started

Introduction

This page teaches how to get started with ATOMS module manager on the scilab platform towards a session example. It describes how one can install a module and load it in Scilab environment.

Atoms

- **Configure ATOMS to display extra-informations**

```
-->atomsSetConfig('Verbose','True');
```

- **List available modules**

```
-->atomsList();

ampl_toolbox - An interface to load .nl files created by AMPL
ANN_Toolbox - ANN Toolbox
conmin - A Scilab interface to the conmin optimization method
CUTer - Testing environment for optimization and linear algebra s
dace_scilab - This is a conversion of the well known DACE kriging toolb
dde_toolbox - Dynamic Data Exchange client for Scilab
HYDROGRv50 - Models and function for operational hydrology
lolimot - A fast neural network - L0cal LLinear Model Tree
module_lycee - Scilab pour les lycées
NISP - Non Intrusive Spectral Projection
plotlib - "Matlab-like" Plotting library for Scilab
scilab2c - Translate Scilab code into C code
scipad - Scipad 7.20
simplex - This package contains the simplex optimization method
sndfile_toolbox - Read & write sound files
stixbox - Statistics toolbox for Scilab 5.2
```

- **Install a module** Installing a module download and extract it.

```
-->atomsInstall('NISP');

NISP (2.1) will be installed in the 'allusers' section
Installing NISP (2.1) ... success
```

- **Load a module** The module is installed but it's not loaded in the scilab environment and its functionalities are not available yet.



By default, a module is added to the list of modules to load at Scilab start when it's installed.
(>> *More information on the autoload system* : `atomsAutoloadList` , `atomsAutoloadAdd` , `atomsAutoloadDel`).

```
-->atomsLoad('NISP');  
  
    Start NISP Toolbox  
      Load gateways  
      Load help  
      Load demos
```

- **Remove a module:**

```
-->atomsRemove NISP  
  
    NISP (2.1) will be removed from the 'allusers' section  
    the package NISP (2.1) is currently loaded, It will removed at next Scilab
```

Authors

Pierre MARECHAL - DIGITEO

Name

Functions Summary

Install/Remove/Update modules ...

atomsInstall	Install one or several modules
atomsUpdate	Update one or several modules
atomsRemove	Remove one or several modules
atomsGetInstalled	Get the list of installed external modules
atomsIsInstalled	Determines whether the module is installed or not

Load modules ...

atomsLoad	Load one or several modules
atomsGetLoaded	Get the list of loaded modules
atomsIsLoaded	Determines whether the module is loaded or not
atomsAutoloadList	Get the list of modules registered to autoload at Scilab start.
atomsAutoloadAdd	Add one or several modules to autoload system.
atomsAutoloadDel	Remove one or several modules from autoload system.

Get information ...

atomsList	List available modules
atomsSearch	Searches for modules
atomsShow	Show the characteristics of a module
atomsDepTreeShow	Show the dependency tree of a module

Manage repositories ...

atomsRepositoryList	Get the list of managed repositories
atomsRepositoryAdd	Add one or several URLs to the list of managed repositories
atomsRepositoryDel	Remove one or several URLs from the list of managed repositories

Manage ATOMS system ...

atomsSystemUpdate	Update the list of available modules
atomsSetConfig	Manage ATOMS parameters

Authors

Pierre MARECHAL - DIGITEO

Name

atomsAutoloadAdd — Add one or several modules to autoload

```
nbAdd = atomsAutoloadAdd(modules[,section])
```

Parameters

modules

mx1, mx2 or mx3 Matrix of strings:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, the most recent version is used
<i>3rd Col.</i>	Installed section	Optionnal	If this field is empty or is not present, and <code>module</code> is installed in both "user" and "allusers" sections, the section of autoload list is used.

section

This argument controls which autoload list is changed.

`section` is a single-string and its value should be :

- "allusers": modules are added to the "allusers" autoload list and all users of scilab are affected.
- "user": modules are added to the "user" autoload list and only the current user is affected.

If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

nbAdd

An integer : the number of modules successfully added.

Description

Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');
atomsSetConfig("autoloadAddAfterInstall","False");

atomsInstall("toolbox_5","user");
atomsAutoloadList()

atomsAutoloadAdd("toolbox_5","user");
atomsAutoloadList()
```

```
atomsAutoloadDel(["toolbox_5"]);
atomsAutoloadAdd(["toolbox_5" "1.0"], "user");
atomsAutoloadList()

atomsAutoloadDel("toolbox_5");
atomsAutoloadAdd(["toolbox_5" "1.0" "user"], "user");
atomsAutoloadList()

atomsRemove("toolbox_5", "user");
atomsSetConfig("autoloadAddAfterInstall", "True");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');

atomsAutoloadList()
```

See Also

atomsAutoloadDel , atomsAutoloadList

Authors

Pierre MARECHAL - DIGITEO

Name

atomsAutoloadDel — Remove one or several modules from the autoload system

```
nbDel = atomsAutoloadDel(modules[,section])
```

Parameters

modules

mx1, mx2 or mx3 character string matrix:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, all versions of the module are removed from the autoload system
<i>3rd Col.</i>	Installed section	Optionnal	If this field is empty or is not present, modules installed in both "user" and "allusers" sections are searched.

section

This argument controls the list of sections where search modules(s) to remove from autoload system.

section is a single-string and its value should be :

- "all": module(s) to remove from autotoload list are searched in both "user" and "allusers" sections.
- "allusers": module(s) to remove from autotoload system are only searched in the "allusers" autoload list.
- "user": module(s) to remove from autotoload system are only searched in the "user" autoload list.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

nbDel

An integer : the number of modules successfully removed from the autoload system.

Description

atomsAutoloadDel remove one or several modules from the autoload system.

Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');  
atomsInstall("toolbox_5", "user");  
atomsAutoloadList()
```

```
atomsAutoloadDel("toolbox_5");
atomsAutoloadList()

atomsAutoloadAdd(["toolbox_5" "1.0" "user"], "user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0"]);
atomsAutoloadList()

atomsAutoloadAdd(["toolbox_5" "1.0" "user"], "user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0" "user"]);
atomsAutoloadList()

atomsAutoloadAdd(["toolbox_5" "1.0" "user"], "user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0" "user"], "user");
atomsAutoloadList()

atomsRemove("toolbox_5", "user");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');
```

See Also

`atomsAutoloadAdd` , `atomsAutoloadList`

Authors

Pierre MARECHAL - DIGITEO

Name

atomsAutoloadList — Get the list of modules registered to autoload

```
modules = atomsAutoloadList([section])
```

Parameters

section

This argument controls the list of section where search URL(s).

section is a single-string and its value should be :

- "all": module(s) present on both "user" and "allusers" autoload lists are returned.
- "allusers": only module(s) present on the "allusers" autoload lists are returned.
- "user": only module(s) present on the "user" autoload lists are returned.

The default value is "all".

modules

4xn character string matrix:

<i>1st Col.</i>	Module's technical name
<i>2nd Col.</i>	Module's version
<i>3rd Col.</i>	Module's installed section
<i>4th Col.</i>	Autoload list section

Description

atomsAutoloadList returns the list of modules registered to autoload

Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');
atomsInstall("toolbox_1");

atomsAutoloadList('user')
atomsAutoloadList('allusers')
atomsAutoloadList('all')

atomsRemove("toolbox_1");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');
```

See Also

atomsAutoloadAdd , atomsAutoloadDel

Authors

Pierre MARECHAL - DIGITEO

Name

atomsDepTreeShow — Show the dependency tree of a module

```
atomsDepTreeShow(module)
```

Parameters

module

1x1 or 1x2 Matrix of strings:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, the most recent version is used

Description

atomsDepTreeShow shows the dependency tree of an external module.

Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
atomsDepTreeShow("toolbox_6")
```

See Also

atomsShow , atomsList

Authors

Pierre MARECHAL - DIGITEO

Name

atomsGetInstalled — Get the list of installed external modules

```
installed = atomsGetInstalled(section)
```

Parameters

installed

5xn String matrix :

- 1st column : External module's technical name
- 2nd column : External module's version
- 3rd column : allusers/user, this parameter tell if the external module has been installed for all users or only for the current user.
- 4th column : External module's installation path
- 5th column : I/A, this parameter tell if the external module has been automatically or intentionnaly installed.

section

This argument filter the output list.

section is a single-string and its value should be :

- "all": atomsGetInstalled() lists external modules installed in both "user" and "allusers" sections.
- "allusers": atomsGetInstalled() only lists external modules installed in the "allusers" section.
- "user": atomsGetInstalled() only lists external modules installed in the "user" section.

The default value is "all".

Description

atomsGetInstalled returns the list of installed external modules

Examples

```
atomsSetConfig("Verbose", "True");  
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
atomsInstall("toolbox_5");  
atomsGetInstalled();  
atomsRemove("toolbox_5");
```

See Also

atomsInstall , atomsIsInstalled

Authors

Pierre MARECHAL - DIGITEO

Name

atomsGetLoaded — Get the list of loaded external modules

```
loaded = atomsGetLoaded()
```

Parameters

loaded

5xn String matrix :

- 1st column : External module's technical name
- 2nd column : External module's version
- 3rd column : allusers/user, this parameter tell if the external module has been installed for all users or only for the current user.
- 4th column : External module's installation path
- 5th column : I/A, this parameter tell if the external module has been automatically or intentionnaly installed.

Description

atomsGetLoaded returns the list of loaded external modules

Examples

```
atomsSetConfig("Verbose", "True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsInstall("toolbox_5");
atomsLoad("toolbox_5");

atomsGetLoaded("toolbox_5");

atomsRemove("toolbox_5");
```

See Also

atomsLoad , atomsIsLoaded

Authors

Pierre MARECHAL - DIGITEO

Name

atomsInstall — Install one or several external modules

```
result = atomsInstall(modules[,section])
result = atomsInstall(file)
```

Parameters

modules

mx1, mx2 Matrix of strings:

1st Col.	Technical name	Mandatory	
2nd Col.	Version	Optionnal	If this field is empty or is not present, the most recent version is used

file

mx1 Matrix of strings:

1st Col.	File-system path	Mandatory	

section

This argument controls where the external module is installed.

section is a single-string and its value should be :

- "allusers": the external module is installed for all users of the computer and is located in SCI/contrib ("allusers zone").
- "user": the external module is only installed for the current user and is located in SCI-HOME/atoms ("user zone").

If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

result

5xn character string matrix:

1st Col.	Technical name	
2nd Col.	Version	
3rd Col.	Installation section	this parameter determines whether the module has been installed for all users or only for the current user.
4th Col.	Installation path	
5th Col.	Status	"I" stands for "Intentionnaly", "A" stands for "Automatically"

Description

atomsInstall install one or more external modules.

Examples

Example 1: Installing a module from a repository

```
// Display some additionnal informations
atomsSetConfig("Verbose","True");

// Load the test repository
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");

// Install a module
atomsInstall("toolbox_1");

// Install a specific version
atomsInstall(["toolbox_2" "2.0"]);

// Install several modules
atomsInstall(["toolbox_4" "1.0" ; "toolbox_2" "1.0"]);

// Install a module in the user section
atomsInstall(["toolbox_5"],"user");

// Install a module in the allusers section
// (write access on SCI directory is needed):
atomsInstall(["toolbox_6" "1.0";"toolbox_3" "1.0"], ..
             "allusers");

// Get the list of installed modules:
disp( atomsGetInstalled() );

// Cleaning :)
atomsRemove(["toolbox_1"; ..
            "toolbox_2"; ..
            "toolbox_3"; ..
            "toolbox_4"; ..
            "toolbox_5"; ..
            "toolbox_6"]);
```

Example 2: Installing a local module. As well as installing modules from the central repository, you can also install modules directly from your own machine.

```
// Display some additionnal informations
atomsSetConfig("Verbose","True");

// Install a module
atomsInstall(SCI+"/modules/atoms/tests/unit_tests/toolbox_7_1.0-1.bin.zip");

// Get the list of installed modules:
disp( atomsGetInstalled() );

// Cleaning :)
atomsRemove("toolbox_7");
```

See Also

[atomsIsInstalled](#) , [atomsGetInstalled](#) , [atomsRemove](#)

Authors

Pierre MARECHAL - DIGITEO

Name

atomsIsInstalled — Determines whether the module is installed. Returns true if the module is installed, false otherwise.

```
res = atomsIsInstalled(modules[,section])
```

Parameters

modules

mx1, mx2 Matrix of strings:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, module's version is ignored.
<i>3rd Col.</i>	Section	Optionnal	If this field is empty or is not present, module's section is ignored.

section

This argument controls the list of searched modules.

section is a single-string and its value should be :

- "all": atomsIsInstalled() searches external modules installed in both "user" and "allusers" sections.
- "allusers": atomsIsInstalled() searches external modules installed in the "allusers" section.
- "user": atomsIsInstalled() searches external modules installed in the "user" section.

The default value is "all".

res

Description

Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");

// Install the needed module for the purpose of the example
atomsInstall("toolbox_5","user");

// simplest way
atomsIsInstalled("toolbox_5");

// Check several modules ...
atomsIsInstalled(["toolbox_5" "toolbox_4"])
```

```
// ... with a specific version
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"])

// ... installed in a specific section
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"], "u
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"], "a

// Some cleaning ...
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
atomsRemove("toolbox_2");
```

See Also

atomsInstall , atomsGetInstalled

Authors

Pierre MARECHAL - DIGITEO

Name

atomsIsLoaded — determines whether a module is loaded or not

```
result = atomsIsLoaded(modules)
```

Parameters

modules

mx1, mx2 Matrix of strings:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, module's version is ignored.
<i>3rd Col.</i>	Section	Optionnal	If this field is empty or is not present, module's section is ignored.

result

mx1 boolean matrix

Description

atomsIsLoaded determines whether a module is loaded or not.

Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");

// Install toolbox_4, both 1.0 and 1.1 version
atomsInstall(["toolbox_4" "1.0"],"user");

// Load the version 1.0
atomsLoad(["toolbox_4" "1.0"]);

// Ignore the module's version
atomsIsLoaded("toolbox_4")

// With a specific version
atomsIsLoaded(["toolbox_4" "1.0"])

// Check several modules
atomsIsLoaded( ["toolbox_4" "1.0" ; "toolbox_4" "1.1" ; "toolbox_1" "1.0" ] )

// Some cleaning ...
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
atomsRemove("toolbox_2");
```

See Also

atomsLoad , atomsGetLoaded

Authors

Pierre MARECHAL - DIGITEO

Name

atomsList — List available external modules

```
atomsList()
```

Description

List available external modules

Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
atomsList()
```

See Also

atomsShow , atomsDepTreeShow

Authors

Pierre MARECHAL - DIGITEO

Name

atomsLoad — Install one or several external modules

```
result = atomsLoad(name[,version])
```

Parameters

name

Matrix of strings : External module name

version

Matrix of strings : External module version. This is an optional parameter. If it's not defined, the most recent version of the module is used.

result

Description

atomsLoad install one or more external modules.

Examples

```
atomsSetConfig("Verbose", "True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsInstall(["toolbox_2" "1.0"]);
atomsInstall(["toolbox_2" "2.0"]);

atomsLoad(["toolbox_2" "1.0"]);
t2_version()
t2_function1()

atomsRemove("toolbox_2");
```

See Also

atomsIsLoaded , atomsGetLoaded

Authors

Pierre MARECHAL - DIGITEO

Name

atomsRemove — Remove one or several modules

```
result = atomsRemove(modules[,section])
```

Parameters

modules

mx1 or mx2 character string matrix:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, all versions of the module are removed.

section

This argument controls the list of sections where search modules to remove.

section is a single-string and its value should be :

- "all": Modules to remove are searched in both "user" and "allusers" sections.
- "allusers": Modules to remove are only searched in the "allusers" section.
- "user": Modules to remove are only searched in the "user" section.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

result

5xn character string matrix:

<i>1st Col.</i>	Technical name	
<i>2nd Col.</i>	Version	
<i>3rd Col.</i>	Installation section	this parameter determines whether the module has been installed for all users or only for the current user.
<i>4th Col.</i>	Installation path	
<i>5th Col.</i>	Status	"I" stands for "Intentionnaly", "A" stands for "Automatically"

Description

atomsRemove remove one or more modules.

Examples

```
// Display some additionnal informations
atomsSetConfig("Verbose","True");
```

```
// Load the test repository
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");

// install toolbox_4 : both 1.0 and 1.1 versions

// Remove all versions of a module
atomsInstall(["toolbox_4" "1.0";"toolbox_4" "1.1"],"user");
atomsRemove(["toolbox_4"]);

// Remove a specific version
atomsInstall(["toolbox_4" "1.0";"toolbox_4" "1.1"],"user");
atomsRemove(["toolbox_4" "1.0"]);
atomsRemove(["toolbox_4" "1.1"]);

// Remove several modules
atomsInstall(["toolbox_4";"toolbox_3"],"user");
atomsRemove(["toolbox_4";"toolbox_3"]);

// Remove a module from a specific section
// ! This example needs write access on SCI directory

atomsInstall("toolbox_4","user");
atomsInstall("toolbox_4","allusers");

disp(atomsGetInstalled());
atomsRemove("toolbox_4","user");
disp(atomsGetInstalled());
atomsRemove("toolbox_4","allusers");
disp(atomsGetInstalled());

// Unload the test repository
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
```

See Also

[atomsRemove](#) , [atomsIsInstalled](#) , [atomsGetInstalled](#)

Authors

Pierre MARECHAL - DIGITEO

Name

atomsRepositoryAdd — Add one or several URLs to the list of managed repositories

```
nbAdd = atomsRepositoryAdd(url[,section])
```

Parameters

url

Matrix of strings : list of the URLs to add

section

This argument controls where the repository is added.

section is a single-string and its value should be :

- "allusers": the repository is added to the "allusers" list and all user of scilab are affected.
- "user": the repository is added to the "user" list and only the current user is affected.

If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

nbAdd

An integer : the number of repositories successfully added.

Description

Examples

```
atomsRepositoryList()  
atomsRepositoryAdd(["http://scene1.test.atoms.scilab.org"])  
atomsRepositoryList()
```

See Also

atomsRepositoryDel , atomsRepositoryList

Authors

Pierre MARECHAL - DIGITEO

Name

atomsRepositoryDel — Remove one or several URLs from the list of managed repositories

```
nbDel = atomsRepositoryDel(url[,section])
```

Parameters

url

character string matrix: list of the URLs to remove

section

This argument controls the list of sections where search URL(s) to remove.

section is a single-string and its value should be :

- "all": URL(s) to remove are searched in both "user" and "allusers" sections.
- "allusers": URL(s) to remove are only searched in the "allusers" section.
- "user": URL(s) to remove are only searched in the "user" section.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

nbDel

An integer : the number of repositories successfully removed.

Description

Examples

```
atomsRepositoryList()  
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org")  
atomsRepositoryList()  
atomsRepositoryDel("http://scene1.test.atoms.scilab.org")  
atomsRepositoryList()
```

See Also

atomsRepositoryAdd , atomsRepositoryList

Authors

Pierre MARECHAL - DIGITEO

Name

atomsRepositoryList — Get the list of managed repositories

```
repositories = atomsRepositoryList([section])
```

Parameters

section

This argument controls the list of section where search URL(s).

section is a single-string and its value should be :

- "all": URL(s) present in the "user", "allusers" and "official" section are listed.
- "allusers": only URL(s) present in the "allusers" section are listed.
- "user": only URL(s) present in the "user" section are listed.
- "official": only URL(s) present in the "official" section are listed.

The default value is "all".

repositories

Matrix of strings : the first column give the list of repositories managed by ATOMS and the second column indicate if the repository is a official repository, if the repository has been added for all users or only for the current user.

Description

atomsRepositoryList return a matrix that give the list of available repositories.

Examples

```
atomsRepositoryAdd('http://scene1.atoms.scilab.org');
atomsRepositoryList()
atomsRepositoryList('all')
atomsRepositoryList('official')
atomsRepositoryList('allusers')
atomsRepositoryList('user')
atomsRepositoryDel('http://scene1.atoms.scilab.org');
```

See Also

atomsRepositoryAdd , atomsRepositoryDel

Authors

Pierre MARECHAL - DIGITEO

Name

atomsSearch — Searches for external modules.

```
result = atomsSearch(pattern)
```

Parameters

pattern

String : The pattern to search for.

result

Description

atomsSearch searches for packages matching one of the patterns supplied as input argument.

Examples

```
atomsSetConfig("Verbose", "True");  
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
atomsSearch("toolbox");
```

See Also

atomsList

Authors

Pierre MARECHAL - DIGITEO

Name

atomsSetConfig — Manage ATOMS system parameters

```
result = atomsSetConfig(parameter,value)
```

Parameters

parameter
single-string matrix

value
single-string matrix

result
Number of changed parameters

Proxy

Parameter	Description	Values
useProxy	Use/Don't use proxies	True /False
proxyHost	the hostname (IP or DNS name)	
proxyPort	the port	
proxyUser	Specify the username for authentication on a proxy server	
proxyPassword	Specify the password for authentication on a proxy server	

Network

Parameter	Description	Values
offLine	If set to "True", the system only works with local repositories. The offline mode permits the user to install modules from a local repository or a local package (hard disk, USB keys, ...) even if the network is unreachable.	True/ False

Autoload System

Parameter	Description	Values
autoload	Enable/Disable autoload system	True /False
autoloadAddAfterInstall	Automatically add a module to the list of module to autoload at Scilab start	True /False

Miscellenous

Parameter	Description	Values

verbose	Display or not extra-informations	True/ False

Description

atomsSetConfig returns the list of modules registered to autoload

Examples

```
// Display extra-informations
atomsSetConfig('Verbose', 'True')

// Disable autoload system
atomsSetConfig('autoload', 'False')
```

Authors

Pierre MARECHAL - DIGITEO

Name

atomsShow — Show the characteristics of a module

```
atomsShow(module)
```

Parameters

module

1x1 or 1x2 Matrix of strings:

<i>1st Col.</i>	Technical name	Mandatory	
<i>2nd Col.</i>	Version	Optionnal	If this field is empty or is not present, the most recent version is used

Description

Show the characteristics of a module

Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
atomsShow("toolbox_5");
```

See Also

atomsList , atomsDepTreeShow

Authors

Pierre MARECHAL - DIGITEO

Name

atomsSystemUpdate — Update the list of available modules

```
atomsSystemUpdate()
```

Description

atomsSystemUpdate update the list of available modules.

Examples

```
atomsSystemUpdate();
```

Authors

Pierre MARECHAL - DIGITEO

Name

atomsUpdate — Update one or several external modules

```
result = atomsUpdate()  
result = atomsUpdate(name[, section])
```

Parameters

name

1xn character string matrix : module's technical name

section

This argument controls the list of sections where search modules to update.

section is a single-string and its value should be :

- "all": Modules to remove are searched in both "user" and "allusers" sections.
- "allusers": Modules to remove are only searched in the "allusers" section.
- "user": Modules to remove are only searched in the "user" section.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

result

Description

atomsUpdate update one or more external modules.

Examples

```
atomsSetConfig("Verbose", "True");  
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");  
  
// Install toolbox_5  
atomsInstall("toolbox_5");  
  
disp(atomsGetInstalled());  
  
// Load the 2nd scenario in which toolbox_4 has been updated:  
// toolbox_4 version 1.1 has been added  
// (toolbox_4 is a dependency of toolbox_5)  
  
atomsRepositoryDel("http://scene1.test.atoms.scilab.org");  
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");  
  
// Update toolbox_5  
atomsUpdate("toolbox_5");  
disp(atomsGetInstalled());  
  
// Some cleaning
```

```
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");  
atomsRemove("toolbox_5");
```

See Also

[atomsInstall](#) , [atomsRemove](#) , [atomsGetInstalled](#)

Authors

Pierre MARECHAL - DIGITEO

Part LIV. Matlab binary files I/O

Name

loadmatfile — loads a Matlab V6 MAT-file (binary or ASCII) into Scilab

```
loadmatfile(format,filename[,var1[,var2[,...]]])  
loadmatfile(filename[,format[,var1[,var2[,...]]]])  
loadmatfile(filename[,var1[,var2[,...[,format]]]])
```

Parameters

filename

character string containing the path of the file (needed)

format

file format (if not given and file has extension ".mat", file is considered to be binary)

"-mat"

binary file

"-ascii"

option to force Scilab to read file as an ASCII file

var1, var2

character strings containing the name of the variables to load (only for binary files)

Description

loads a Matlab MAT-file into Scilab. The Matlab data types are converted into the Scilab equivalents.

Examples

```
A = rand(10,10);  
B = sprand(100,100,0.1);  
savematfile('test_matfile.mat','A','B','-v6');  
clear;  
loadmatfile('test_matfile.mat');  
disp(A)  
disp(B)
```

See Also

load , savematfile , save , mfile2sci , matfile2sci

Authors

Serge Steer (INRIA)
V.C

Bibliography

This function has been developped following the "MAT-File Format" description: **Mat-File Format** [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]

Name

`matfile_close` — Closes a Matlab V5 binary MAT-file.

```
status = matfile_close(fd)
```

Parameters

`fd`
Real: file descriptor (returned by `matfile_open`).

`status`
Boolean: *%T* if closure succeeds, *%F* otherwise.

Description

Closes a Matlab binary MAT-file opened by `matfile_open`.

See Also

`matfile_open` , `matfile_varreadnext` , `matfile_varwrite` , `matfile_listvar`

Authors

V.C

Bibliography

This function uses MATIO library (<http://sourceforge.net/projects/matio/>).

Name

matfile_listvar — Lists variables of a Matlab V5 binary MAT-file.

```
[names[, classes[, types]]] = matfile_listvar(fd)
```

Parameters

fd
Real: file descriptor (returned by matfile_open).

names
String vector: names of the variables.

classes
Real vector: classes of the variables.

types
Real vector: data types of the variables.

Description

Lists variables of a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_close , matfile_varwrite , matfile_varreadnext

Authors

V.C

Bibliography

This function uses MATIO library (<http://sourceforge.net/projects/matio/>).

Name

matfile_open — Opens a Matlab V5 binary MAT-file.

```
fd = matfile_open(filename[, mode])
```

Parameters

filename

String: the path of the file. Must contain only ANSI character.

mode

String: file access type ("r" by default).

- "r": opens the file for reading.
- "w": opens the file for writing.

fd

Real: file descriptor (-1 if opening failed).

Description

Opens a Matlab binary MAT-file for reading or writing data.

See Also

matfile_close , matfile_varreadnext , matfile_varwrite , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (<http://sourceforge.net/projects/matio/>).

Name

matfile_varreadnext — Reads next variable in a Matlab V5 binary MAT-file.

```
[name[, value[, vartype]]] = matfile_varreadnext(fd)
```

Parameters

fd

Real: file descriptor (returned by `matfile_open`).

name

String: name of the variable read or "" if reading failed.

value

Any Scilab type: value of the variable read or an empty matrix if reading failed.

vartype

Real: type of the variable if reading succeeds or:

- 0: if the variable type is unknown.
- -1: if end of file has been reached.

Description

Reads next variable in a Matlab binary MAT-file opened by `matfile_open`.

See Also

`matfile_open` , `matfile_close` , `matfile_varwrite` , `matfile_listvar`

Authors

V.C

Bibliography

This function uses MATIO library (<http://sourceforge.net/projects/matio/>).

Name

matfile_varwrite — Write a variable in a Matlab V5 binary MAT-file.

```
status = matfile_varreadnext(fd, name, value, compressionflag)
```

Parameters

fd

Real: file descriptor (returned by matfile_open).

name

String: name of the variable to write in the file.

value

Any Scilab type: value of the variable to write in the file.

compressionflag

Boolean: indicate if data compression has to be used (flag equaled to %T) or not.

status

Boolean: %T if writing succeeds, %F otherwise.

Description

Writes a variable in a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_close , matfile_varreadnext , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (<http://sourceforge.net/projects/matio/>).

Name

`saveumatfile` — write a Matlab MAT-file (binary or ASCII)

```
saveumatfile('filename')
saveumatfile('filename', 'var1', 'var2', ...)
saveumatfile('filename', '-struct', 's')
saveumatfile('filename', '-struct', 's', 'f1', 'f2', ...)
saveumatfile(..., '-v4')
saveumatfile(..., '-v6')
saveumatfile(..., '-v7')
saveumatfile(..., '-v7.3')
saveumatfile filename var1 var2 ...
```

Parameters

`filename`

character string containing the path of the file (needed)

`format`

data format to use

"-mat"

binary MAT-file (default)

"-ascii"

8-bit ASCII format

"-ascii" "-double"

16-bit ASCII format

"-ascii" "-tabs"

delimits with tabs

"-ascii" "-double" "-tabs"

16-digit ASCII format, tab delimited

"-v4"

A format that MATLAB Version 4 can open

"-v6"

A format that MATLAB Version 6 and earlier can open

"-v7"

A format that MATLAB Version 7 and earlier can open (default)

"-v7.3"

A format that MATLAB Version 7.3 and earlier can open

`var1, var2`

character strings containing the name of the variables to load (only for binary files)

"-struct" "s"

saves all fields of the scalar structure `s` as individual variables within the file `filename`.

"-struct" "s" "f1" "f2"

saves as individual variables only those structure fields specified (`s.f1`, `s.f2`, ...).

Description

saves variables in a Matlab MAT-file from Scilab. The Scilab data types are converted into the Matlab equivalents.

Examples

```
A = rand(10,10);
B = sprand(100,100,0.1);
savematfile('test_matfile.mat','A','B','-v6');
clear;
loadmatfile('test_matfile.mat');
disp(A)
disp(B)
```

See Also

load , save , loadmatfile , mfile2sci

Authors

Serge Steer (INRIA)
V.C

Bibliography

This function has been developped following the "MAT-File Format" description: Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]

Part LV. xcos

Table of Contents

6. Batch functions	2412
lincos	2413
scicos	2414
scicos_simulate	2415
scicosim	2417
steadycos	2419
7. palettes	2421
1. Annotations palette	2421
2. Commonly used blocks palette	2424
3. Continuous time systems palette	2432
4. Demonstrations blocks palette	2457
5. Discontinuities palette	2474
6. Discrete time systems palette	2487
7. Electrical palette	2504
8. Event handling palette	2562
9. Implicit palette	2608
10. Integer palette	2613
11. Lookup tables palette	2644
12. Math operations palette	2651
13. Matrix operation palette	2698
14. Port & Subsystem palette	2761
15. Signal processing palette	2768
16. Signal routing palette	2773
17. Sinks palette	2809
18. Sources palette	2864
19. Thermohydraulics palette	2917
20. User defined functions palette	2931
21. Zero crossing detection palette	2957
8. Programming xcos Blocks	2967
1. C Computational Functions	2967
2. Scilab Computational Functions	2981
3. Utilities Functions	2985
9. Scilab Data Structures	2996
1. Blocks	2996
2. Compilation/Simulation	3007
3. Diagram	3015
4. Links	3020
10. Scilab Utilities Functions	3023
buildouttb	3024
create_palette	3025
get_scicos_version	3026
scicos_debug	3027
var2vec	3028
vec2var	3029

Chapter 6. Batch functions

Name

lincos — Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram

```
sys= lincos(scs_m [,x0,u0 [,param] ])
```

Module

- xcos

Description

Construct a linear state-space system by linearizing a model given as a xcos diagram.

The output is a Scilab data structure of type continuous-time state-space linear system.

```
sys= lincos(scs_m [,x0,u0 [,param] ])
```

Parameters

- **scs_m** : a xcos data structure
- **x0** : column vector. Continuous state around which linearization to be done (default 0)
- **u0** : column vector. Input around which linearization to be done (default 0)
- **param** : param: list with two elements (default list(1.d-6,0)) param(1): scalar. Perturbation level for linearization; the following variation is used $\text{del}([x;u])_i = \text{param}(1) + \text{param}(1) * 1d-4 * \text{abs}([x;u])_i$. param(2): scalar. Time t.
- **sys** : state-space system

File content

- SCI/modules/scicos/macros/scicos_auto/lincos.sci

See Also

- **steadycos** - Finds an equilibrium state of a general dynamical system described by a xcos diagram (Scilab Function)
- **scicos_simulate** - Function for running xcos simulation in batch mode (Scilab Function)

Authors

Ramine Nikoukhah - INRIA

Name

scicos — OBSOLETE - see xcos

```
scs_m = scicos([toto])
```

Module

- xcos

See Also

- xcos

Name

scicos_simulate — Function for running xcoss simulation in batch mode

```
Info=scicos_simulate(scs_m, Info[, %scicos_context][, flag])
```

Module

- xcoss

Description

This function is used to simulate xcoss diagrams in batch mode. It requires the scs_m structure which can be obtained by loading in Scilab the .cos file (e.g. load mydiagram.cos).

Contrary to the function , the diagram need not be compiled before being saved.

```
Info=scicos_simulate(scs_m, Info[, %scicos_context][, flag])
```

Parameters

- **scs_m** : xcoss diagram (obtained by "load file.cos"). Note that the version of file.cos must be the current version. If not, load into xcoss and save.
- **Info** : a list. It must be list() at the first call, then use output Info as input Info for the next calls. Info contains compilation and simulation information and is used to avoid recompilation when not needed.
- **%scicos_context** : a Scilab struct containing values of symbolic variables used in the context and xcoss blocks. This is often used to change a parameter in the diagram context. In that case, make sure that in the diagram context the variable is defined such that it can be modified. Say a variable "a" is to be defined in the context having value 1, and later in batch mode, we want to change the value of "a". In that case, in the context of the diagram place: if exists('a') then a=1,end If you want then to run the simulation in batch mode using the value a=2, set: %scicos_context.a=2
- **flag** : string. If it equals 'nw' (no window), then blocks using graphical windows are not executed. Note that the list of such blocks must be updated as new blocks are added.
- **Info** : contains compilation and simulation information and is used to avoid recompilation when not needed. Normally it should not be modified.

File content

- SCI/modules/scicos/macros/scicos_auto/scicos_simulate.sci

See Also

- scicosim - Scicos (batch) simulation function (Scilab Function)
- xcoss - Block diagram editor and GUI for the hybrid simulator (Scilab Function)
- steadyccos - Finds an equilibrium state of a general dynamical system described by a xcoss diagram (Scilab Function)

- lincos - Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram (Scilab Function)

Authors

Ramine Nikoukhah - INRIA

Name

scicosim — xcos (batch) simulation function

```
[state,t] = scicosim(state,tcur,tf,sim,str,tol)
```

Module

- xcos

Description

Simulator for xcos compiled diagram.

Usually scicosim is called by xcos to perform simulation of a diagram.

But scicosim may also be called outside xcos. Typical usage in such a case may be :

For advanced user it is possible to "manually" change some parameters or state values.

```
[state,t] = scicosim(state,tcur,tf,sim,str,tol)
```

Parameters

- **state** : Scilab tlist containing initial state. Usually generated by xcos Compile. After loading a compiled .cos file, it can be found in %cpr.state.
- **tcur** : starting time of simulation. At the beginning it must be zero.
- **tf** : final simulation time.
- **sim** : Scilab tlist containing compilation results. Usually generated by xcos Compile. After loading a compiled .cos file, it can be found in %cpr.sim.
- **str** : 'start' , 'run' or 'finish'. Function must be first called with 'start', then with 'run' one or more times, and finally with 'finish'.
- **tol** : vector [atol,rtol,ttol,deltat,realtimescale,solver,hmax] where atol, rtol are respectively the absolute and relative tolerances for ode or dae solver, ttol is the precision on event dates (must be very small), deltat is maximum integration interval for each call to ode solver (sometimes needed to force restarting the call to solver), realtimescale is the correspondance between simulation time and real time (0 means no slowing down), solver is the choice of solver (0: lsodar, 100: daskr), hmax: max step size used by solver. Default: [0.0001,1.000E-06,1.000E-10,100001,0,0]
- **state** : state after simulation
- **t** : final reached time. Usually tf but not always.

See Also

- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)
- xcos - Block diagram editor and GUI for the hybrid simulator (Scilab Function)

Authors

- **Ramine Nikoukhah** INRIA
- **Alan Layec** INRIA

Name

steadycos — Finds an equilibrium state of a general dynamical system described by a xcos diagram

```
[X,U,Y,XP]=steadycos(scs_m,X,U,Y,Indx,Indu,Indy [,Indxp [,param ] ])
```

Module

- xcos

Description

This function finds the steady state for a given system described by a xcos diagram. The diagram consists in general of a Super block with input and output port blocks. The steady state concern only the continuous-time dynamics.

```
[X,U,Y,XP]=steadycos(scs_m,X,U,Y,Indx,Indu,Indy [,Indxp [,param ] ])
```

Parameters

- **scs_m** : a xcos data structure
- **X** : column vector. Continuous state. Can be set to [] if zero.
- **U** : column vector. Input. Can be set to [] if zero.
- **Y** : column vector. Output. Can be set to [] if zero.
- **Indx** : index of entries of X that are not fixed. If all can vary, set to 1:\$
- **Indu** : index of entries of U that are not fixed. If all can vary, set to 1:\$
- **Indy** : index of entries of Y that are not fixed. If all can vary, set to 1:\$
- **Indxp** : index of entries of XP (derivative of x) that need not be zero. If all can vary, set to 1:\$. Default [].
- **param** : list with two elements (default list(1.d-6,0)). param(1): scalar. Perturbation level for linearization; the following variation is used $\text{del}([x;u])_i = \text{param}(1) + \text{param}(1) * 1d-4 * \text{abs}([x;u])_i$. param(2): scalar. Time t.
- **X** : steady state X
- **U** : stationary input U
- **Y** : output corresponding to steady state found
- **XP** : derivative of the state corresponding to steady state found

File content

- SCI/modules/scicos/macros/scicos_auto/steadycos.sci

See Also

- lincos - Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram (Scilab Function)

- `scicos_simulate` - Function for running xcos simulation in batch mode (Scilab Function)
- `xcos` - Block diagram editor and GUI for the hybrid simulator (Scilab Function)

Authors

Ramine Nikoukhah - INRIA

Chapter 7. palettes

1. Annotations palette

Name

Annotations_pal — Annotations palette

Block Screenshot

A small, light gray rectangular block with the word "Text" centered inside in a dark gray font.

Module

- xCOS

Description

This palette includes blocks used for annotations.

Blocks

- TEXT_f - Text

Name

TEXT_f — Text

Block Screenshot



Contents

- Text
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

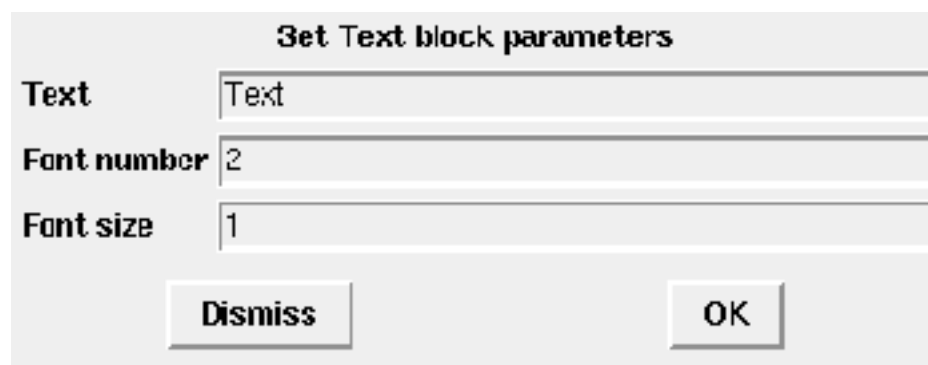
Palette

- Annotations palette

Description

This special block is only use to add text at any point of the diagram window. It has no effect on the simulation.

Dialog box



Set Text block parameters	
Text	Text
Font number	2
Font size	1
Dismiss OK	

- **Text**
a character string, Text to be displayed
Properties : Type 'str' of size -1
- **Font number**
a positive integer less than 6, number of selected font (see**xset**)

Properties : Type 'vec' of size 1

- **Font size**

a positive integer, selected font size (see **xset**).

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *text*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/TEXT_f.sci

Authors

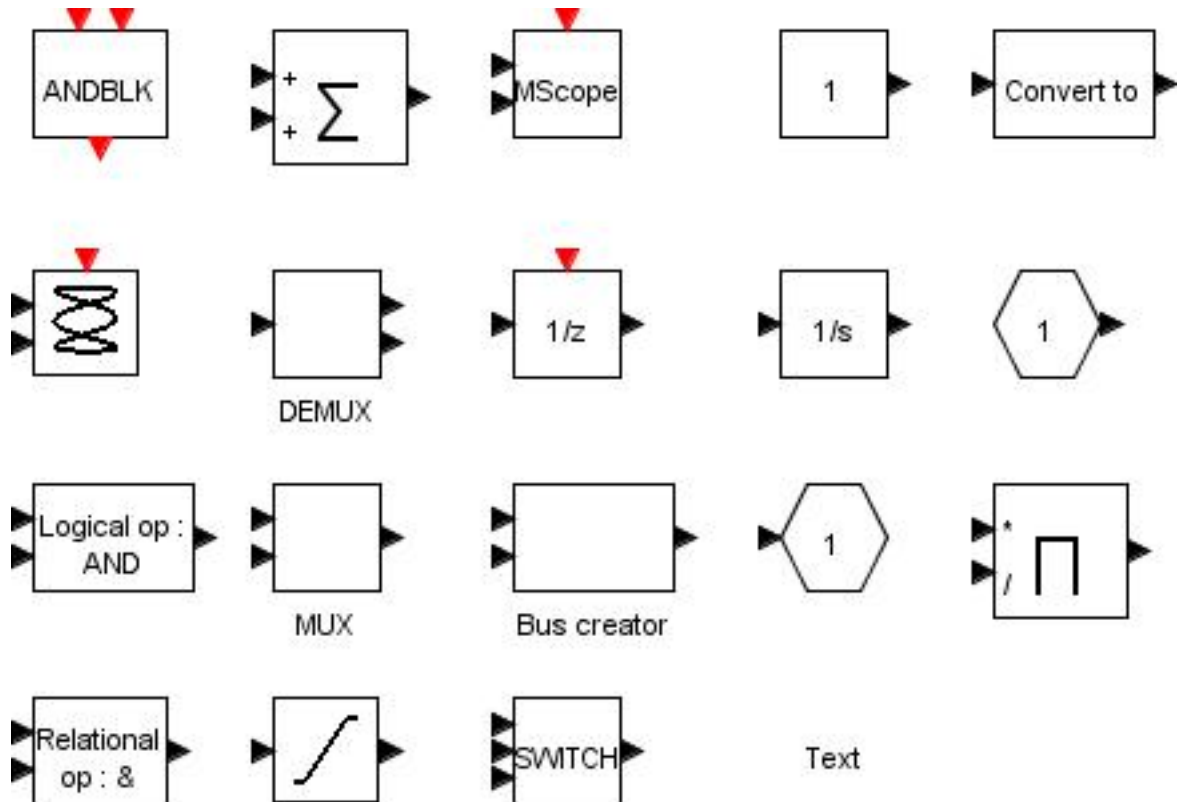
Ramine Nikoukhah - INRIA

2. Commonly used blocks palette

Name

Commonlyusedblocks_pal — Commonly used blocks palette

Block Screenshot



Module

- xcoss

Description

In the Commonly used blocks palette, you can find blocks from other palettes that most models use.

Blocks

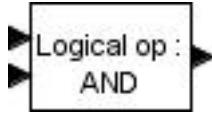
- ANDBLK - Activation and
- BIGSOM_f - Sum
- CMSCOPE - Multi display scope
- CONST_m - Constant
- CONVERT - CONVERT Data Type Conversion
- CSCOPXY - $y=f(x)$ permanent viewer
- DEMUX - Demultiplexer
- DOLLAR_f - Delay operator
- INTEGRAL_f - Integration

- IN_f - Input Port
- LOGICAL_OP - Logical operation
- MUX - Multiplexer
- NRMSOM_f - Merge data
- OUT_f - Output Port
- PRODUCT - Product
- RELATIONALOP — Relational operation
- SATURATION - Saturation
- SWITCH2_m - Switch2
- TEXT_f - Text

Name

LOGICAL_OP — Logical operation

Block Screenshot



Contents

- Logical operation
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

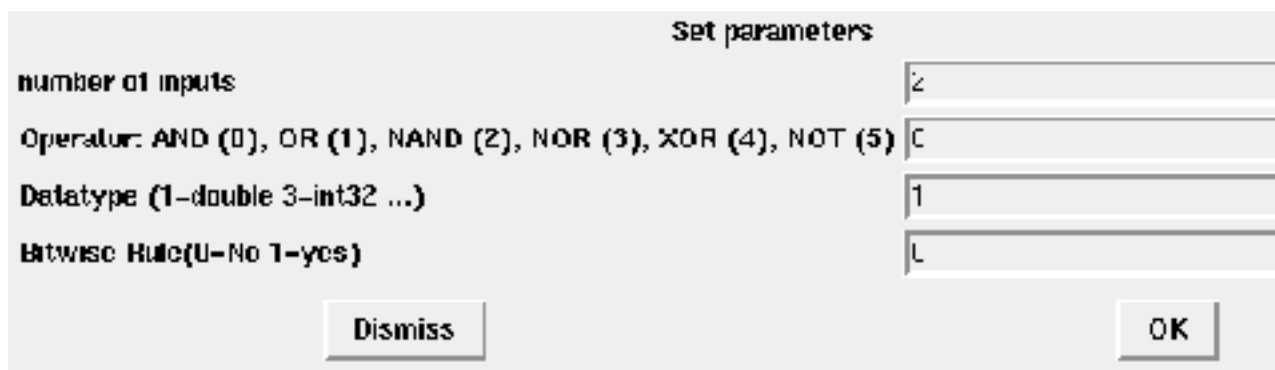
Palette

- Commonly used blocks palette

Description

The Logical Operator block performs the specified logical operation on its inputs. An input value is TRUE (1) if it is nonzero and FALSE (0) if it is zero.

Dialog box



- **number of inputs**

The number of block inputs. The value must be appropriate for the selected operator.

Properties : Type 'vec' of size 1

- **Operator: AND**

The logical operator to be applied to the block inputs. Valid choices are the operators from the list.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logicalop*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/LOGICAL_OP.sci

Computational function

- SCI/modules/scicos_blocks/src/c/logicalop.c
- SCI/modules/scicos_blocks/src/c/logicalop_i32.c
- SCI/modules/scicos_blocks/src/c/logicalop_i16.c
- SCI/modules/scicos_blocks/src/c/logicalop_i8.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui32.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui16.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui8.c

Authors

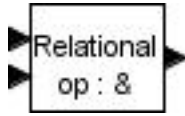
- **Fady NASSIF** INRIA

- **Ramine Nikoukhah** INRIA

Name

RELATIONALOP — Relational operation

Block Screenshot



Contents

- Relational operation
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Commonly used blocks palette

Description

It performs logical comparison of its two inputs.

Dialog box

 A screenshot of a 'Set parameters' dialog box. The title bar at the top right says 'Set parameters'. The dialog has three rows of labels and input fields. The first row is 'Operator: == (0), ~= (1), < (2), <= (3), >= (4), > (5)' with a dropdown menu showing '2'. The second row is 'Use zero crossing (no: 0), (yes: 1)' with a dropdown menu showing '0'. The third row is 'Datatype (1=double 3=int32 ...)' with a dropdown menu showing '1'. At the bottom left is a 'Dismiss' button, and at the bottom right is an 'OK' button.

- **Operator: ==**
Designate the relational operator used to compare the two inputs.
Properties : Type 'vec' of size 1
- **Use zero crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *relationalop*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/RELATIONALOP.sci

Computational function

- SCI/modules/scicos_blocks/src/c/relational_op.c
- SCI/modules/scicos_blocks/src/c/relational_op_i32.c
- SCI/modules/scicos_blocks/src/c/relational_op_i16.c
- SCI/modules/scicos_blocks/src/c/relational_op_i8.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui32.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui16.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui8.c

Authors

- **Fady NASSIF** INRIA

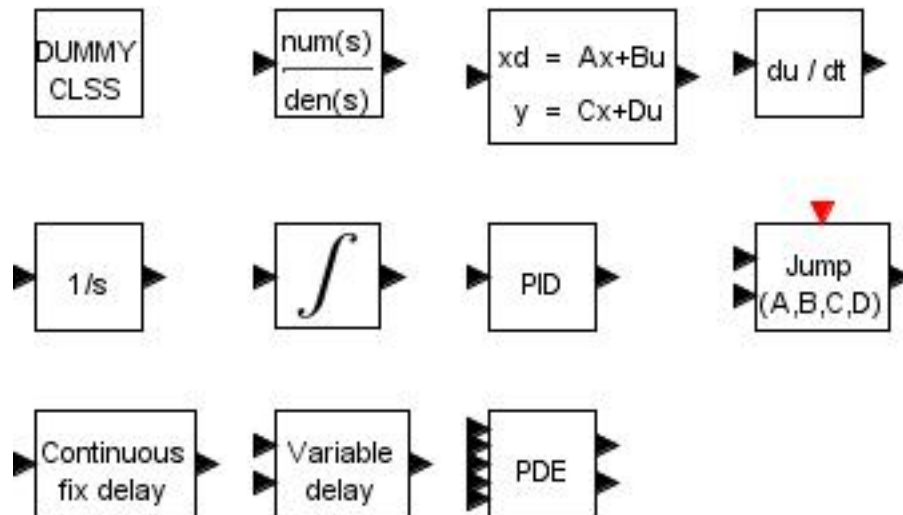
- **Ramine Nikoukhah** INRIA

3. Continuous time systems palette

Name

Continuous_pal — Continuous time systems palette

Block Screenshot



Module

- xcos

Description

The Continuous time systems palette includes basic linear blocks .

Blocks

- CLINDUMMY_f — Dummy
- CLR — Continuous transfer function
- CLSS - Continuous state-space system
- DERIV - Derivative
- INTEGRAL_f - Integration
- INTEGRAL_m — Integration
- PID - PID regulator
- TCLSS — Continuous linear system with jump
- TIME_DELAY - Time delay
- VARIABLE_DELAY — Variable delay

Name

CLINDUMMY_f — Dummy

Block Screenshot



Contents

- Dummy
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- Continuous time systems palette

Description

This block should be placed in any block diagram that contains a zero-crossing block but no continuous system with state. The reason for that is that it is the ode solver that find zero crossing surfaces.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cdummy*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLINDUMMY_f.sci

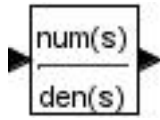
Authors

Ramine Nikoukhah - INRIA

Name

CLR — Continuous transfer function

Block Screenshot



Contents

- Continuous transfer function
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Continuous time systems palette

Description

This block realizes a SISO linear system represented by its rational transfer function **Numerator/Denominator**. The rational function must be proper.

Dialog box

Set continuous SISO transfer parameters

Numerator (s)

Denominator (s)

- **Numerator**

This parameter sets the numerator of the transfer function.

This must be a polynomial in s.

Properties : Type 'pol' of size 1.

- **Denominator**

This parameter sets the denominator of the transfer function.

This must be a polynomial in s .

Properties : Type 'pol' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csslti4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLR.sci

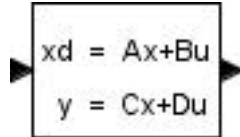
Computational function

- SCI/modules/scicos_blocks/src/c/csslti4.c (Type 4)

Name

CLSS — Continuous state-space system

Block Screenshot



Contents

- Continuous state-space system
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Continuous time systems palette

Description

This block realizes a continuous-time linear state-space system.

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C} * \mathbf{x} + \mathbf{D} * \mathbf{u}\end{aligned}$$

$$(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) \quad \mathbf{x}_0$$

The system is defined by the matrices and the initial state $\mathbf{x}_0$. The dimensions must be compatible.

Dialog box

Set continuous linear system parameters

A matrix

B matrix

C matrix

D matrix

Initial state

- **A matrix**
A square matrix.
Properties : Type 'mat' of size [-1,-1].
- **B matrix**
The B matrix, [] if system has no input.
Properties : Type 'mat' of size ["size(%1,2)","-1"].
- **C matrix**
The C matrix, [] if system has no output.
Properties : Type 'mat' of size ["-1","size(%1,2)"].
- **D matrix**
The D matrix, [] if system has no D term.
Properties : Type 'mat' of size [-1,-1].
- **Initial state**
A vector/scalar initial state of the system.
Properties : Type 'vec' of size "size(%1,2)".

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no

- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csslti4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLSS.sci

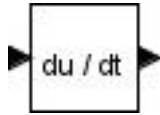
Computational function

- SCI/modules/scicos_blocks/src/c/csslti4.c (Type 4)

Name

DERIV — Derivative

Block Screenshot



Contents

- Derivative
- • the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Continuous time systems palette

Description

The Derivative block approximates the derivative of its input by computing:

$$\frac{\Delta u}{\Delta t}$$

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *deriv*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DERIV.sci

Computational function

- SCI/modules/scicos_blocks/src/c/deriv.c (Type 4)

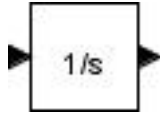
Authors

Ramine Nikoukhah - INRIA

Name

INTEGRAL_f — Integration

Block Screenshot



Contents

- Integration
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

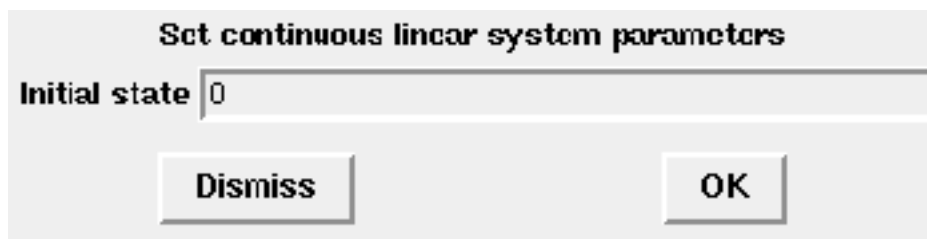
Palette

- Continuous time systems palette

Description

This block is an integrator. The output is the integral of the input.

Dialog box



- **Initial Condition**

A scalar that gives the initial condition.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *integr*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/INTEGRAL_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/integr.f (Type 0)

Name

INTEGRAL_m — Integration

Block Screenshot



Contents

- Integration
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Continuous time systems palette

Description

This block is an integrator. The output is the integral of the input.

Dialog box

Set Integral block parameters	
Initial Condition	0
With re-initialization (1:yes, 0:no)	0
With saturation (1:yes, 0:no)	0
Upper limit	1
Lower limit	-1
DismissOK	

- **Initial Condition**
A vector/scalar initial conditions.

With that parameter, one can define the datatype of the input/output. It can be a real or a complex type.

Properties : Type 'mat' of size [-1,-1].

- **With re-initialization**

To reset its state to the specified initial condition based on an external signal select1 .

Properties : Type 'vec' of size 1.

- **With saturation**

If selected, limits the states to a value between the Lower saturation limit and Upper saturation limit parameters.

Properties : Type 'vec' of size 1.

- **Upper limit**

The upper limit for the integral.

Properties : Type 'mat' of size [-1,-1].

- **Lower limit**

The lower limit for the integral.

Properties : Type 'mat' of size [-1,-1].

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *integral_func*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/INTEGRAL_m.sci

Computational function

- `SCI/modules/scicos_blocks/src/c/integral_func.c`
- `SCI/modules/scicos_blocks/src/c/integralz_func.c`

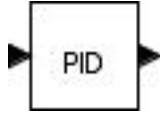
Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

Name

PID — PID regulator

Block Screenshot



Contents

- PID regulator
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “Authors”

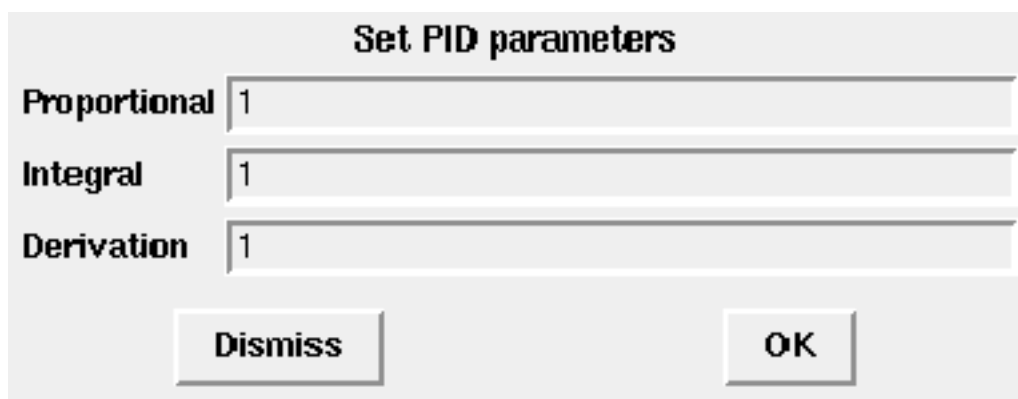
Palette

- Continuous time systems palette

Description

This block implements a PID controller. The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element.

Dialog box



Set PID parameters	
Proportional	1
Integral	1
Derivation	1
DismissOK	

- **Proportional**

The value of the gain that multiply the error.

Properties : Type 'vec' of size -1.

- **Integral**

The value of the integral time of the error.(1/Integral)

Properties : Type 'vec' of size -1.

- **Derivation**

The value of the derivative time of the error.

Properties : Type 'vec' of size -1.

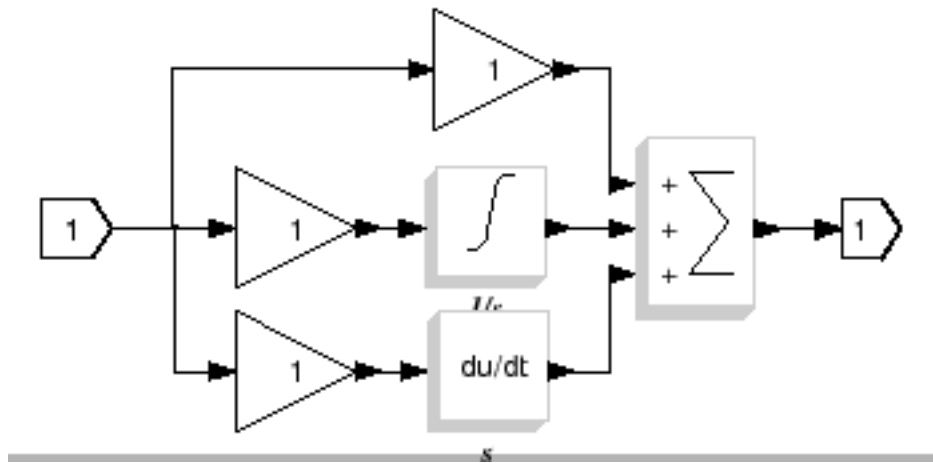
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/PID.sci

Compiled Super Block content



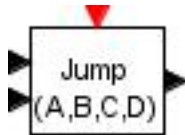
Authors

Fady NASSIF - INRIA

Name

TCLSS — Continuous linear system with jump

Block Screenshot



Contents

- Continuous linear system with jump
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Continuous time systems palette

Description

This block realizes a continuous-time linear state-space system with the possibility of jumps in the state. The number of inputs to this block is two. The first input is the regular input of the linear system, the second carries the new value of the state which is copied into the state when an event arrives at the unique event input port of this block. That means the state of the system jumps to the value present on the second input (of size equal to that of the state). The system is defined by the matrices and the initial state. The dimensions must be compatible. The sizes of inputs and outputs are adjusted automatically.

Dialog box

A dialog box titled "Set continuous linear system parameters". It contains five input fields with labels: "A matrix" (value 0), "B matrix" (value 1), "C matrix" (value 1), "D matrix" (value 0), and "Initial state" (value 0). At the bottom, there are two buttons: "Dismiss" and "OK".

Set continuous linear system parameters	
A matrix	0
B matrix	1
C matrix	1
D matrix	0
Initial state	0
DismissOK	

- **A matrix**

A square matrix.

Properties : Type 'mat' of size [-1,-1].

- **B matrix**

The B matrix, [] if system has no input.

Properties : Type 'mat' of size ["size(%1,2)","-1"].

- **C matrix**

The C matrix, [] if system has no output.

Properties : Type 'mat' of size ["-1","size(%1,2)"].

- **D matrix**

The D matrix, [] if system has no D term.

Properties : Type 'mat' of size [-1,-1].

- **Initial state**

A vector/scalar initial state of the system.

Properties : Type 'vec' of size "size(%1,2)".

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *tcslti4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/TCLSS.sci

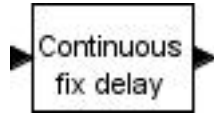
Computational function

- SCI/modules/scicos_blocks/src/c/tcslti4.c (Type 4)

Name

TIME_DELAY — Time delay

Block Screenshot



Contents

- Time delay
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Continuous time systems palette

Description

The Transport Delay block delays the input by a specified amount of time. It can be used to simulate a time delay. At the start of the simulation, the block outputs the Initial input parameter until the simulation time exceeds the Time delay parameter, when the block begins generating the delayed input.

The Time delay parameter must be non-negative.

Dialog box

A dialog box titled "Set delay parameters". It contains three input fields: "Delay" with the value "1", "initial input" with the value "0", and "Buffer size" with the value "1024". At the bottom, there are two buttons: "Dismiss" and "OK".

Set delay parameters	
Delay	1
initial input	0
Buffer size	1024
Dismiss OK	

- Delay

The amount of simulation time that the input signal is delayed before being propagated to the output. The value must be nonnegative.

Properties : Type 'vec' of size 1

- **initial input**

The output generated by the block between the start of the simulation and the Time delay.

Properties : Type 'vec' of size 1

- **Buffer size**

The initial memory allocation for the number of points to store.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *time_delay*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/TIME_DELAY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/time_delay.c (Type 4)

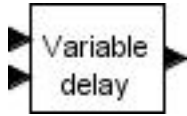
Authors

Ramine Nikoukhah - INRIA

Name

VARIABLE_DELAY — Variable delay

Block Screenshot



Contents

- Variable delay
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

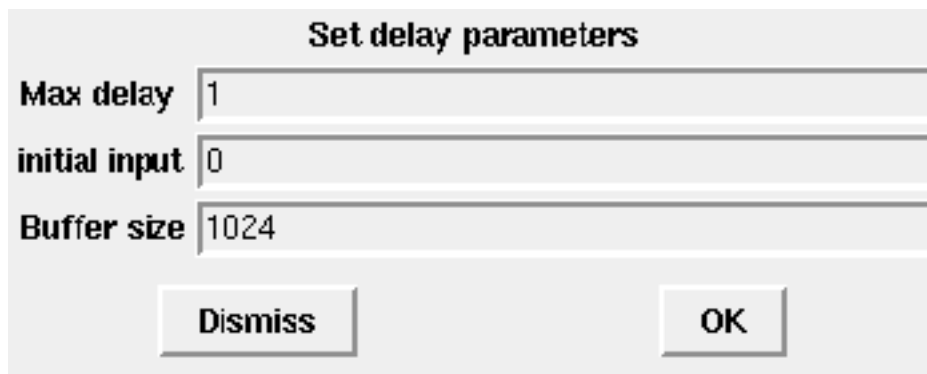
Palette

- Continuous time systems palette

Description

The Variable Transport Delay block can be used to simulate a variable time delay. The block might be used to model a system with a pipe where the speed of a motor pumping fluid in the pipe is variable. The block accepts two inputs: the first input is the signal that passes through the block; the second input is the time delay.

Dialog box

A dialog box titled 'Set delay parameters'. It contains three input fields: 'Max delay' with the value '1', 'initial input' with the value '0', and 'Buffer size' with the value '1024'. At the bottom, there are two buttons: 'Dismiss' and 'OK'.

- **Max delay**

It defines the largest value the time delay input can have. The value cannot be negative.

Properties : Type 'vec' of size 1.

- **initial input**

The output generated by the block until the simulation time first exceeds the time delay input.

Properties : Type 'vec' of size 1.

- **Buffer size**

The number of points the block can store.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *variable_delay*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/VARIABLE_DELAY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/variable_delay.c (Type 4)

Authors

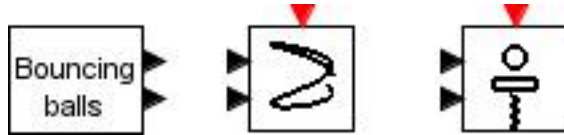
Ramine Nikoukhah - INRIA

4. Demonstrations blocks palette

Name

Demonstrationsblocks_pal — Demonstrations blocks palette

Block Screenshot



Module

- xcos

Description

The Demonstrations blocks palette contains blocks used in some Xcos demonstration diagrams.

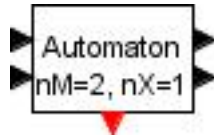
Blocks

- AUTOMAT — hybrid automata
- BOUNCE - Balls coordinates generator
- BOUNCEXY — Balls viewer
- BPLATFORM — Balls under a platform viewer
- PDE — 1D PDE block

Name

AUTOMAT — automata (finite state machine)

Block Screenshot



Contents

- automata (finite state machine)
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Demonstrations blocks palette

Description

This block gives the possibility to construct hybrid automata, i.e., a hybrid system whose discrete part is defined via modes and transitions between modes, and the continuous part is defined via DAE (differential algebraic equations).

The automaton block provides a switching mechanism between subsystems corresponding to control modes of an automaton. Subsystems are constructed in such a way that they have the state vector as input (coming from the automaton block) and compute the flow and jump functions (zero-crossing) and pass them back to the automaton block. The state variables are defined in the automaton block and the subsystems are static functions.

Suppose that a hybrid automaton consists of M control modes. The continuous-time dynamics in mode is defined with DAE () where and the dimension of is () for any . Suppose that in control mode , there are jump conditions indicating jumps toward other modes. The jump conditions are defined by functions where .

When a jump function changes sign and becomes positive, a mode transition will happen. When transition function becomes positive, a transition to mode happens and state vector is reset to , for .

In order to develop an automaton containing a mode with multiple reset functions, the value of the current and previous active modes should be used. These values are available at the first output port of the block.

The automaton block has the following input/output ports.

- **Output 1:** The first output port is a vector of size two consisting of the current and the previous active control modes, i.e.,

$$OUT_1 = [current - mode, previous - mode]^t$$

- **Output 2:** The second output port is a vector of size $2N$ providing the state vector and its first

$$[x, \dot{x}]^t$$

time derivative,

- **Inputs:** The automaton block has M vector input ports corresponding to M modes or subsystems of the automaton. Each input defines the dynamic behavior in the control each mode as well as the reset functions and the transition functions. The input port i which is the output of the i^{th}

$$2N + Z_i$$

subsystem is a vector of size . Each input is composed of the following vector.

- $Input_i$
The first N elements of the are the continuous-time dynamics. The dynamics of the system in the control mode i is described by a smooth index-1 DAE ($0 = F_i(\dot{x}, x, u, t)$).

- $Input_i$
The next N elements of are the values used to reset the continuous-time state when a transition to control mode i is activated.

- Z_i $Input_i$ j^{th}
The next elements of are the jump or zero-crossing functions. If the zero-crossing function of mode i crosses zero with negative to positive direction, a transition j^{th} to destination mode happens.

- **Event Output:** This is an event output port, which is activated whenever a mode transition happens. This event is useful when an event is needed to activate or initialize a part of the subsystem not included in the internal dynamics of the automaton block.

In the interface window, the number of control modes, the initial control mode and the initial value of continuous-time state at the beginning of the simulation should be given.

Find more documentation and demos about the Automaton block at www.scicos.org. Interested users are referred to the paper "Modeling Hybrid Automata in Scicos", Masoud Najafi, Ramine Nikoukhah, 2007 IEEE Multi-conference on Systems and Control, Singapore.

Dialog box

Set finite state machine model	
Number (finite state) Modes	2
Initial Mode	1
Number of continuous-time states	1
Continuous-time states initial values	0
Xproperties of continuous time states in each Mode	[1;1]
Jump from Mode 1:[...;M_final(Guard=In(1).i);...]	2
Jump from Mode 2:[...;M_final(Guard=In(2).i);...]	1
Dismiss OK	

- **Number of (finite-state) Modes**

Number of modes in the automation.

Properties : Type 'vec' of size [1,1].

- **Initial Mode**

Initial active mode at the beginning of the simulation.

Properties : Type 'vec' of size [1,1].

- **Number of continuous-time states** Number of continuous-time states at modes. Note that the number of continuous-time states is the same in all modes.

Properties : Type 'vec' of size [-1,1].

- **Continuous-time states initial values**

Initial value of continuous-time states at the beginning of the simulation.

Properties : Type 'vec' of size [-1,1].

- **Xproperties of continuous-time states in each Mode**

In this field the state types in mode are given. A state in an index 1 DAE can be either differential state or algebraic state. vector is coded in an M*N matrix, where M is the number of modes and N is the number of states. This matrices indicates whether a continuous-time state is algebraic or

differential in each control mode. If in the i^{th} mode, j^{th} state is differential, the (i,j)-th element of the Xproperty matrix should set to "+1", otherwise it should set to "-1". Xproperty can be given as a 1*N vector if type of states remain the same in all modes.

Properties : Type 'mat' of size [-1,-1].

- **Jump from Mode 1:[...;M_final(Guard=In(1).i);...]**

The fields express the mode transition information. Suppose that all control modes are labeled from 1 to M. Then, in the field corresponding to control mode i , destination modes of mode i are defined in a vector. j -th element of this vector gives the destination mode when j -th jump function :

becomes positive. For example, if in the field of the mode 2, the user defines [1;3;4], it means that in mode 2, there are three active jump functions. When, for example, the third jump function becomes positive, a mode transition to mode 4 will be activated.

Properties : Type 'vec' of size [-1,1].

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - port 1 : size [3,1] / type 1
 - port 2 : size [3,1] / type 1
- **regular outputs:**
 - port 1 : size [2,1] / type 1
 - port 2 : size [2,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *automat*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/AUTOMAT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/automat.c (Type 10004)

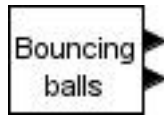
Authors

Masoud Najafi - INRIA

Name

BOUNCE — Balls coordinates generator

Block Screenshot



Contents

- Balls coordinates generator
- • the section called “Palette”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Demonstrations blocks palette

Dialog box

Set Bounce Block	
Mass	[1;1]
Radius	[1;1]
[xmin,xmax,ymin,ymax]	[0;5;0;5]
xpos	[2;2.5]
xdpos	[0;0]
ypos	[3;5]
ydpos	[0;0]
g (gravity)	9.81
C (aerodynamic coeff	0
DismissOK	

- **Mass**

The parameter description 1.

Properties : Type 'vec' of size -1.

- **Radius**

The parameter description 2.

Properties : Type 'vec' of size -1.

- **[xmin,xmax,ymin,ymax]**

The parameter description 3.

Properties : Type 'vec' of size -1.

- **xpos**

The parameter description 4.

Properties : Type 'vec' of size -1.

- **xdpos**

The parameter description 5.

Properties : Type 'vec' of size -1.

- **ypos**

The parameter description 6.

Properties : Type 'vec' of size -1.

- **ydpos**

The parameter description 7.

Properties : Type 'vec' of size -1.

- **g (gravity)**

The parameter description 8.

Properties : Type 'vec' of size 1.

- **C (aerodynamic coeff**

The parameter description 9.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** no

- **regular outputs:**
 - port 1 : size [2,1] / type 1
 - port 2 : size [2,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bounce_ball*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BOUNCE.sci

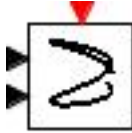
Computational function

- SCI/modules/scicos_blocks/src/c/bounce_ball.c (Type 4)

Name

BOUNCEXY — Balls viewer

Block Screenshot



Contents

- Balls viewer
 - the section called “Palette”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Demonstrations blocks palette

Dialog box

Set Scope parameters	
colors	[1;2]
radii	[1;1]
window number (1 for automatic)	1
animation mode (0,1)	1
Xmin	-5
Xmax	5
Ymin	0
Ymax	15
DismissOK	

- colors

The parameter description 1.

Properties : Type 'vec' of size -1.

- **radii**

The parameter description 2.

Properties : Type 'vec' of size -1.

- **window number (-1 for automatic)**

The parameter description 3.

Properties : Type 'vec' of size 1.

- **animation mode (0,1)**

The parameter description 4.

Properties : Type 'vec' of size 1.

- **Xmin**

The parameter description 5.

Properties : Type 'vec' of size 1.

- **Xmax**

The parameter description 6.

Properties : Type 'vec' of size 1.

- **Ymin**

The parameter description 7.

Properties : Type 'vec' of size 1.

- **Ymax**

The parameter description 8.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 1

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *bouncexy*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BOUNCXY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/bouncexy.c (Type 4)

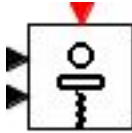
Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

BPLATFORM — Balls under a platform viewer

Block Screenshot



Contents

- Balls under a platform viewer
 - the section called “Palette”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- Demonstrations blocks palette

Dialog box

Set Scope parameters	
pendulum length	2
cart size (square side)	2
slope	11
Xmin	-5
Xmax	5
Ymin	0
Ymax	15
Dismiss OK	

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no

- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *bplatform2*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BPLATFORM.sci

Authors

M. Najafi - INRIA

Name

PDE — 1D PDE block

Block Screenshot



Contents

- 1D PDE block
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- Demonstrations blocks palette

Description

This block is an implementation of several numerical schemes (Finite Elements (1st and 2nd order), Finite Differences (1st and 2nd order), Finite Volumes (1st order)) to solve mono dimensional PDE (Partial Differential Equation) within SCICOS. The mathematical framework was restricts in PDEs linear scalars with maximum order 2 in time and space. The goal is to provide engineers and physicists with an easy to use toolbox in SCICOS that will let them graphically describe the PDE to be solved. A decision system selects the most efficient numerical scheme depending on the type of the PDE and runs the solvers.

Dialog box

- **a et b**

(double) The two edges of the discretization field.

- **specification de l'EDP**

check box to select the PDE operators. $a_i(x)$, $b_i(t)$ ($i=1:7$) are the operator coefficients. type of PDE discriminant (constant or variable, in the later case, the sign should be given)

- **Discretization methode**

choix (check box) : is the choice for the manual or the automatic mode. type : in the manual mode we can give the method type (Finite differences, finite elements or finite volumes). degré : method degree (1 or 2 for the FD and FE methods, 1 for the FV method). Nombre de noeuds : to give the number of the nodal points.

- **Conditions initiales**

$u(x,t_0)=, du/dt$ at $t_0=$: to give the initial conditions.

- **Conditions aux limites**

type : two type of the boundray conditions are possible : Dirichlet or Neumann. expressions : to give then boundray conditions expressions.

- **Points de mesures**

To give the list of mesurment points.

- **Name**

A getvalue box to give the block name's.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 0
 - port 3 : size [1,1] / type 0
 - port 4 : size [1,1] / type 0
 - port 5 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [10,1] / type 1
 - port 2 : size [0,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *PDE*

Interfacing function

- SCI/modules/scicos_blocks/macros/PDE/PDE.sci

Authors

- EADS 2005-01-16

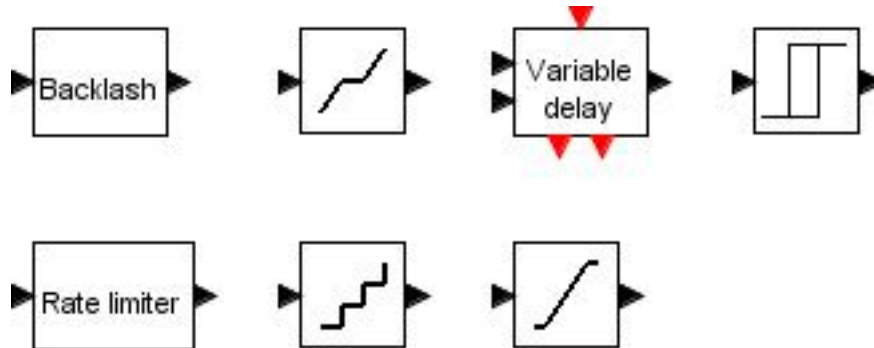
EADS 2005-01-16

5. Discontinuities palette

Name

discontinuities_pal — discontinuities palette

Block Screenshot



Module

- xcOS

Description

discontinuities palette includes blocks whose outputs are discontinuities functions of their inputs.

Blocks

- BACKLASH - Backlash
- DEADBAND - Deadband
- DELAYV_f — Variable time delay
- HYSTHERESIS — Hysteresis
- RATELIMITER - Rate limiter
- QUANT_f - Quatization
- SATURATION — Saturation

Name

BACKLASH — Backlash

Block Screenshot



Contents

- Backlash
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- discontinuities palette

Description

The Backlash block implements a system in which a change in input causes an equal change in output. However, when the input changes direction, an initial change in input has no effect on the output. The amount of side-to-side play in the system is referred to as the .

Dialog box

Set backlash parameters	
initial output	0
gap	1
use zero-crossing (0:no, 1:yes)	1
DismissOK	

- **initial output**

The initial output value.

Properties : Type 'vec' of size 1

- **gap**

The width of the dead-band.

Properties : Type 'vec' of size 1

- **use zero-crossing**

Select to enable use of zero crossing detection to detect engagement with lower and upper thresholds.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *backlash*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BACKLASH.sci

Computational function

- SCI/modules/scicos_blocks/src/c/backlash.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

DEADBAND — Deadband

Block Screenshot



Contents

- Deadband
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- discontinuities palette

Description

Provide a region of zero output

Dialog box

Set Deadband parameters	
End of dead band	0.5
Start of dead band	-0.5
zero crossing (0:no, 1:yes)	1
Dismiss OK	

- **End of dead band**

The upper limit of the dead band.

Properties : Type 'vec' of size 1
- **Start of dead band**

The lower limit of the dead band.

Properties : Type 'vec' of size 1

- **zero crossing**

Select to enable zero crossing detection to detect when the limits are reached.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *deadband*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/DEADBAND.sci

Computational function

- SCI/modules/scicos_blocks/src/c/deadband.c (Type 4)

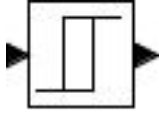
Authors

Ramine Nikoukhah - INRIA

Name

HYSTHERESIS — Hysteresis

Block Screenshot



Contents

- Hysteresis
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- discontinuities palette

Description

Switch output between two constants. The Relay block allows its output to switch between two specified values. When the relay is on, it remains on until the input drops below the value of the Switch off point parameter. When the relay is off, it remains off until the input exceeds the value of the Switch on point parameter. The block accepts one input and generates one output.

Dialog box

Set parameters	
switch on at	1
switch off at	0
output when on	1
output when off	0
use zero crossing: yes (1), no (0)	1

- **switch on at**

The Switch on point parameter is converted to the input data type offline using round-to-nearest and saturation.

Properties : Type 'vec' of size 1

- **switch off at**

The Switch off point parameter is converted to the input data type offline using round-to-nearest and saturation.

Properties : Type 'vec' of size 1

- **output when on**

The output when the relay is on.

Properties : Type 'vec' of size 1

- **output when off**

The output when the relay is off.

Properties : Type 'vec' of size 1

- **use zero crossing: yes**

Select to enable zero crossing detection to detect switch-on and switch-off points.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *hysteresis*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/HYSTHERESIS.sci

Computational function

- SCI/modules/scicos_blocks/src/c/hystheresis.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

RATELIMITER — Rate limiter

Block Screenshot



Contents

- Rate limiter
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

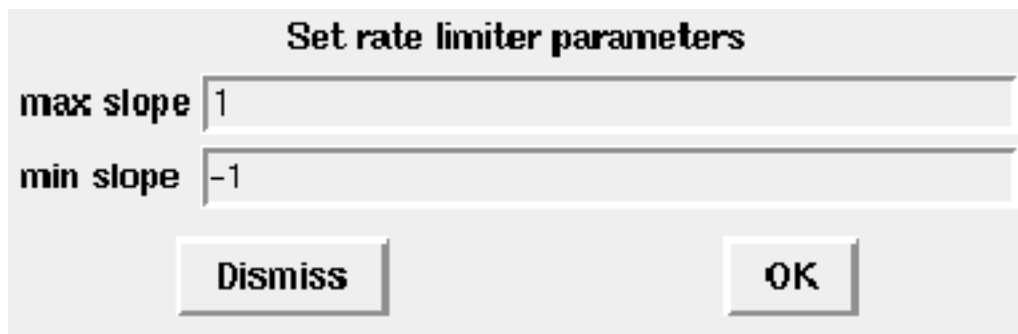
Palette

- discontinuities palette

Description

The Rate Limiter block limits the first derivative of the signal passing through it. The output changes no faster than the specified limit.

Dialog box



- **max slope**
The limit of the derivative of an increasing input signal.
Properties : Type 'vec' of size 1
- **min slope**

The limit of the derivative of a decreasing input signal.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ratelimiter*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/RATELIMITER.sci

Computational function

- SCI/modules/scicos_blocks/src/c/ratelimiter.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

SATURATION — Saturation

Block Screenshot



Contents

- Saturation
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

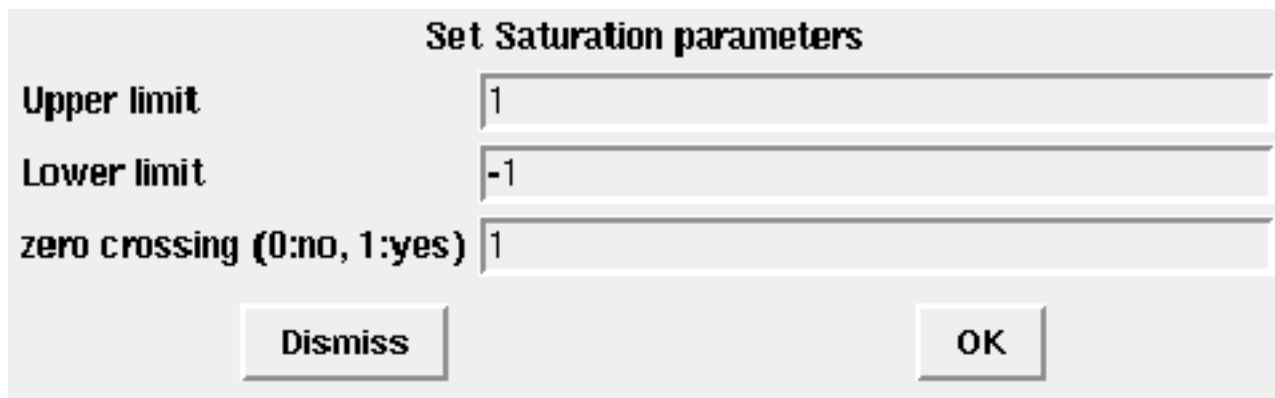
Palette

- discontinuities palette

Description

The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the Lower limit and Upper limit parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound. When the Lower limit and Upper limit parameters are set to the same value, the block outputs that value.

Dialog box



Set Saturation parameters	
Upper limit	1
Lower limit	-1
zero crossing (0:no, 1:yes)	1
DismissOK	

- Upper limit

Specify the upper bound on the input signal. When the input signal to the Saturation block is above this value, the output of the block is clipped to this value.

Properties : Type 'vec' of size 1

- **Lower limit**

Specify the lower bound on the input signal. When the input signal to the Saturation block is below this value, the output of the block is clipped to this value.

Properties : Type 'vec' of size 1

- **zero crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *satur*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SATURATION.sci

Computational function

- SCI/modules/scicos_blocks/src/c/satur.c (Type 4)

Authors

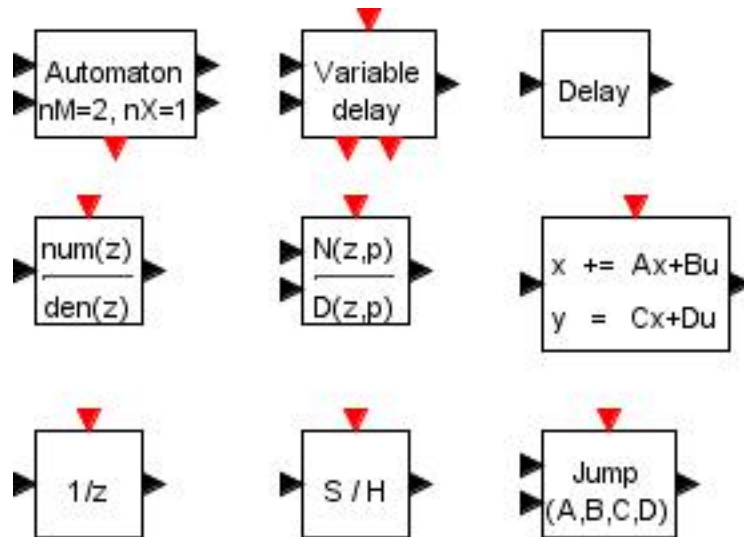
Ramine Nikoukhah - INRIA

6. Discrete time systems palette

Name

Discrete_pal — Discrete time systems palette

Block Screenshot



Module

- xcOS

Description

The Discrete time systems palette contains blocks for modeling discrete time systems.

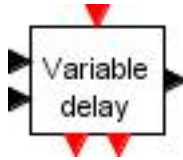
Blocks

- DELAY_f — Discrete time delay
- DELAYV_f — Variable delay
- DLR — Discrete transfer function
- DLRADAPT_f — Discrete Zero-Pole function
- DLSS — Discrete state-space system
- DOLLAR_f — Delay operator
- REGISTER — Shift Register
- SAMPHOLD_m — Sample and hold
- TCLSS — Continuous linear system with jump

Name

DELAYV_f — Variable delay

Block Screenshot



Contents

- Variable delay
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Discrete time systems palette

Description

Add here a paragraph of the function description

Dialog box

Set delay parameters	
Number of inputs	1
Register initial condition	0;0;0;0;0;0;0;0;0
Max delay	1
DismissOK	

- **Number of inputs**

Set the vector size of the first regular input and the vector size of the regular output port.

Properties : Type 'vec' of size 1.

- **Register initial condition**

Set the length and the initial conditions of the register.

Properties : Type 'vec' of size -1.

- **Max delay**

It defines the largest value the time delay input can have. The value cannot be negative.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *delayv*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DELAYV_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/delayv.f (Type 1)

Name

DELAY_f — Discrete time delay

Block Screenshot



Contents

- Discrete time delay
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content

Palette

- Discrete time systems palette

Description

This compiled super-block implements a discretized delay. It is built with a shift register and a clock. The value of the delay is given by the discretization time step multiplied by the number-1 of state of the register.

Dialog box

**This block implements as a discretised delay
it is consist of a shift register and a clock
value of the delay is given by;
the discretisation time step multiplied by the
number-1 of state of the register**

Discretisation time step

Register initial state

- **Discretization time step**

Set the time period of the integrated clock.

Properties : Type 'vec' of size 1.

- **Register initial state**

Set the length and the initial conditions of the register.

Properties : Type 'vec' of size -1.

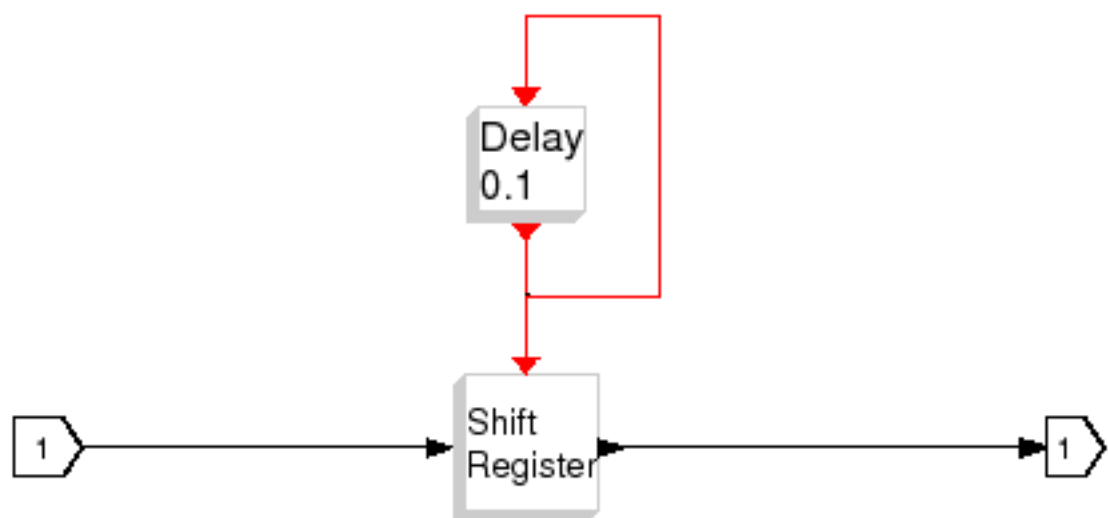
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DELAY_f.sci

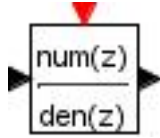
Compiled Super Block content



Name

DLR — Discrete transfer function

Block Screenshot



Contents

- Discrete transfer function
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Discrete time systems palette

Description

This block realizes a SISO linear system represented by its rational transfer function (in the symbolic variable z). The rational function must be proper.

Dialog box

- **Numerator (z)**

This parameter sets the numerator of the transfer function.

This must be a polynomial in z .

Properties : Type 'pol' of size 1.

- **Denominator (z)**

This parameter sets the denominator of the transfer function.

This must be a polynomial in z .

Properties : Type 'pol' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dsslti4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DLR.sci

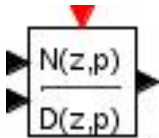
Computational function

- SCI/modules/scicos_blocks/src/c/dsslti4.c (Type 4)

Name

DLRADAPT_f — Discrete Zero-Pole

Block Screenshot



Contents

- Discrete Zero-Pole
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

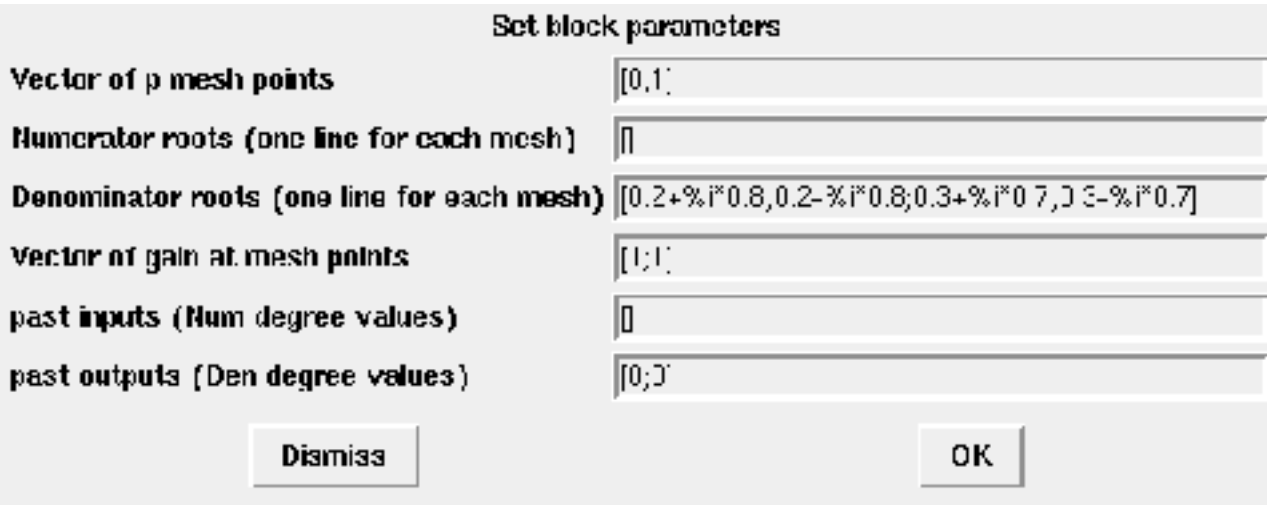
Palette

- Discrete time systems palette

Description

This block models system represented by zeros and poles of discrete transfer function.

Dialog box



- Vector of p mesh points
- The parameter description 1.

Properties : Type 'vec' of size -1.

- **Numerator roots (one line for each mesh)**

The parameter description 2.

Properties : Type 'mat' of size [-1,-1].

- **Denominator roots (one line for each mesh)**

The parameter description 3.

Properties : Type 'mat' of size ["size(%1,"*")," -1"].

- **Vector of gain at mesh points**

The parameter description 4.

Properties : Type 'vec' of size "size(%1,"*")".

- **past inputs (Num degree values)**

The parameter description 5.

Properties : Type 'vec' of size "size(%2,2)".

- **past outputs (Den degree values)**

The parameter description 6.

Properties : Type 'vec' of size "size(%3,2)".

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *dlradp*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/DLRADAPT_f.sci

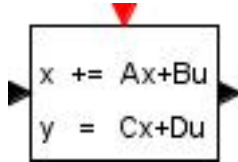
Computational function

- SCI/modules/scicos_blocks/src/fortran/dlradp.f (Type 0)

Name

DLSS — Discrete state-space system

Block Screenshot



Contents

- Discrete state-space system
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Discrete time systems palette

Description

This block realizes a discrete-time linear state-space system. The system is defined by the matrices and the initial state . The dimensions must be compatible. At the arrival of an input event on the unique input event port, the state is updated.

Dialog box

Set discrete linear system parameters

A matrix	-1
B matrix	1
C matrix	1
D matrix	0
Initial state	0

Dismiss OK

- **A matrix**

A square matrix.

Properties : Type 'mat' of size [-1,-1].

- **B matrix**

The B matrix, [] if system has no input.

Properties : Type 'mat' of size ["size(%1,2)","-1"].

- **C matrix**

The C matrix, [] if system has no output.

Properties : Type 'mat' of size ["-1","size(%1,2)"].

- **D matrix**

The D matrix, [] if system has no D term.

Properties : Type 'mat' of size [-1,-1].

- **Initial state**

A vector/scalar initial state of the system.

Properties : Type 'vec' of size "size(%1,2)".

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dsslti4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DLSS.sci

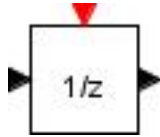
Computational function

- `SCI/modules/scicos_blocks/src/c/dsslti4.c` (Type 4)

Name

DOLLAR_f — Delay operator

Block Screenshot



Contents

- Delay operator
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Discrete time systems palette

Description

The Unit Delay block delays its input by the specified sample period. This block is equivalent to the z^{-1} discrete-time operator. The block accepts one input and generates one output, which can be either both scalar or both vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

Dialog box

A dialog box titled "Set 1/z block parameters". It has two input fields. The first field is labeled "initial condition" and contains the value "0". The second field is labeled "Inherit (no:0, yes:1)" and contains the value "0". At the bottom of the dialog box, there are two buttons: "Dismiss" and "OK".

- **initial condition**

The output of the simulation for the first sampling period, during which the output of the Unit Delay block is otherwise undefined.

Properties : Type 'vec' of size -1.

- **Inherit (no:0, yes:1)**

When "Inherit" is yes, the block inherit its event input.

Properties : Type 'vec' of size -1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dollar*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DOLLAR_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/dollar.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

REGISTER — Shift Register

Block Screenshot



Contents

- Shift Register
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Discrete time systems palette

Description

This block realizes a shift register. At every input event, the register is shifted one step.

Dialog box

Set delay parameters

Register initial condition 0;0;0;0;0;0;0;0;0

Datatype (1=double 3=int32 ...) 1

Dismiss OK

- **Register initial condition**

A column vector. It contains the initial state of the register.

Properties : Type 'vec' of size -1
- **Datatype**

This block support all datatypes besides complex.

Properties : Type 'vec' of size -1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [1,1] / type 1
- **regular outputs:**
 - **port 1 :** size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *delay4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/REGISTER.sci

Computational function

- SCI/modules/scicos_blocks/src/c/delay4_i32.c
- SCI/modules/scicos_blocks/src/c/delay4_i16.c
- SCI/modules/scicos_blocks/src/c/delay4_i8.c
- SCI/modules/scicos_blocks/src/c/delay4_ui32.c
- SCI/modules/scicos_blocks/src/c/delay4_ui16.c
- SCI/modules/scicos_blocks/src/c/delay4_ui8.c

Authors

Fady NASSIF - INRIA

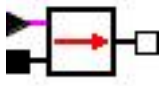
7. Electrical palette

- Inductor - Electrical inductor
- NMOS - Simple NMOS Transistor
- NPN - NPN transistor
- OpAmp - Ideal opamp (norator-nullator pair)
- PMOS - Simple PMOS Transistor
- PNP - PNP transistor
- PotentialSensor - Potential sensor
- Resistor - Electrical resistor
- SineVoltage - Sine voltage source
- Switch - Non-ideal electrical switch
- VariableResistor - Electrical variable resistor
- VoltageSensor - Electrical voltage sensor
- VsourceAC - Electrical AC voltage source
- VVsourceAC - Variable AC voltage source

Name

CCS — Controllable Modelica current source

Block Screenshot



Contents

- Controllable Modelica current source
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model
- the section called “See also”
- the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

This block is an ideal current source. The current value is controlled through the explicit input of the block (connected to standard Xcos blocks). The voltage across the block is independent of the current value.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'Iin'
Explicit **variable**.
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
- **File name of the model :** CCS

Interfacing function

- `SCI/modules/scicos_blocks/macros/Electrical/CCS.sci`

Modelica model

- `SCI/modules/scicos_blocks/macros/Electrical/CCS.mo`

See also

- CVS - Controllable Modelica voltage source

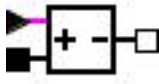
Authors

Masoud Najafi - INRIA

Name

CVS — Controllable Modelica voltage source

Block Screenshot



Contents

- Controllable Modelica voltage source
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “See also”
- the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

This block is an ideal voltage source. The voltage value is controlled through the explicit input of the block (connected to standard Xcos blocks). The current passing through the block is independent of the voltage across the block terminals.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'vin'
Explicit **variable**.
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
- **File name of the model :** CVS

Interfacing function

- `SCI/modules/scicos_blocks/macros/Electrical/CVS.sci`

Modelica model

- `SCI/modules/scicos_blocks/macros/Electrical/CVS.mo`

See also

- CCS - Controllable Modelica current source

Authors

Masoud Najafi - INRIA

Name

Capacitor — Electrical capacitor

Block Screenshot



Contents

- Electrical capacitor
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

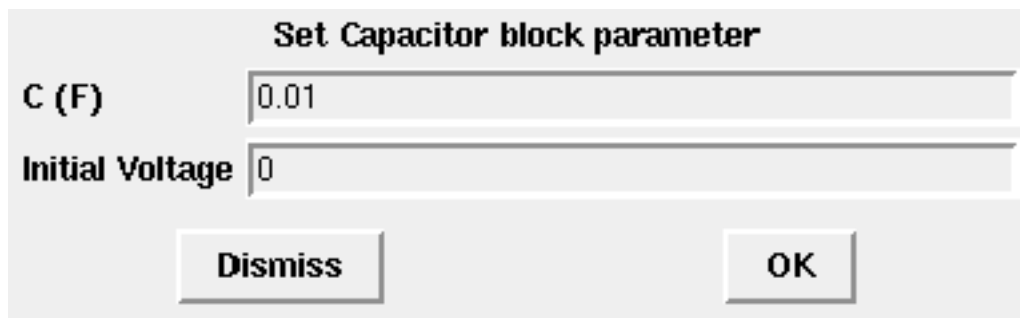
Description

A capacitor is an electrical component that can store energy in electrical circuits. The relationship between the voltage across a capacitor with capacitance and the current passing through it is given by the:

$$I = C \frac{dV}{dt}$$

Capacitors can also be used to differentiate between high-frequency and low-frequency signals and this makes them useful in electronic filters. A capacitor has a high impedance when a signal is low frequency signals.

Dialog box

The image shows a dialog box titled "Set Capacitor block parameter". It contains two input fields: "C (F)" with a value of "0.01" and "Initial Voltage" with a value of "0". At the bottom, there are two buttons: "Dismiss" and "OK".

Set Capacitor block parameter	
C (F)	0.01
Initial Voltage	0
DismissOK	

- **C (F)**

Capacitance

Properties : Type 'vec' of size 1.

- **Initial Voltage**

Initial Voltage

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name : 'p'**

Implicit variable.

- **Outputs :**

- **Modelica variable name : 'n'**

Implicit variable.

- **Parameters :**

- **Modelica parameter name : 'C'**

Default value : 0.01

Is a state variable : no.

- **Modelica parameter name : 'v'**

Default value : 0

Is a state variable : yes.

- **File name of the model :** Capacitor

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Capacitor.sci

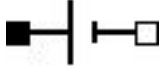
Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Capacitor.mo

Name

ConstantVoltage — Electrical DC voltage source

Block Screenshot



Contents

- Electrical DC voltage source
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

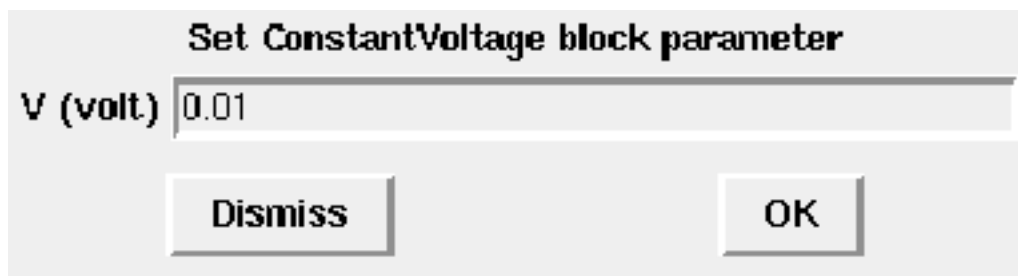
Palette

- Electrical.cosf - Electrical toolbox

Description

This component is a model for any device or system that produces a constant electromotive force between its port. The output voltage of this DC voltage source is defined by the user. The black port indicates the positive voltage. The ohmic resistance of this DC voltage source is zero.

Dialog box



- **V (volt)**
Output voltage
Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
- **Parameters :**
 - **Modelica parameter name :** 'V'
Default value : 0.01
Is a state variable : no.
- **File name of the model :** ConstantVoltage

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/ConstantVoltage.sci

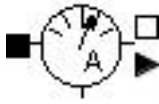
Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/ConstantVoltage.mo

Name

CurrentSensor — Electrical current sensor

Block Screenshot



Contents

- Electrical current sensor
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

This block is inserted in series in an electrical circuit to measure the current passing through the component. The measure is given to the explicit part of the model via an explicit pout. Conventionally, current flowing into the black port is considered positive. The ohmic resistance of this block is zero.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
 - **Modelica variable name :** 'i'
Explicit **variable**.
- **File name of the model :** CurrentSensor

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/CurrentSensor.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/CurrentSensor.mo

Name

Diode — Electrical diode

Block Screenshot



Contents

- Electrical diode
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

This component consists of a simple diode parallel with an ohmic resistance (). The current passing through this component is defined as a function of the voltage across the ports, ,

$$i = I_{ds}(\exp^{v/V_t} - 1) + \frac{v}{R}$$

where and are the saturation current and the voltage equivalent of temperature, respectively. If the exponent reaches a certain limit (), the diode characteristic becomes linear to avoid overflow.

Dialog box

Set Diode block parameter	
Saturation current (A)	0.000001
Voltage equivalent to temperature (Volt)	0.04
Max exponent for linear continuation	15
R (ohm)	1.000E+08
Dismiss OK	

- **Saturation current (A)**

Saturation current

Properties : Type 'vec' of size 1.

- **Voltage equivalent to temperature (Volt)**

Voltage equivalent of temperature

Properties : Type 'vec' of size 1.

- **Max exponent for linear continuation**

Max exponent for linear continuation

Properties : Type 'vec' of size 1.

- **R (ohm)**

Parallel ohmic resistance.

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name : 'p'**

Implicit **variable**.

- **Outputs :**

- **Modelica variable name : 'n'**

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name : 'Ids'**

Default value : 0.000001

Is a state variable : no.

- **Modelica parameter name : 'Vt'**

Default value : 0.04

Is a state variable : no.

- **Modelica parameter name : 'Maxexp'**

Default value : 15

Is a state variable : no.

- **Modelica parameter name : 'R'**

Default value : 1.000E+08

Is a state variable : no.

- **File name of the model :** Diode

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Diode.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Diode.mo

Name

Ground — Ground (zero potential reference)

Block Screenshot



Contents

- Ground (zero potential reference)
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

The Ground element is a single port component providing a reference voltage in electrical circuits. The potential at the ground node is zero. Every electrical circuit has to contain at least one ground element.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **File name of the model :** Ground

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Ground.sci

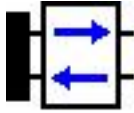
Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Ground.mo

Name

Gyrator — Modelica Gyrator

Block Screenshot



Contents

- Modelica Gyrator
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - Modelica model
 - the section called “Authors”

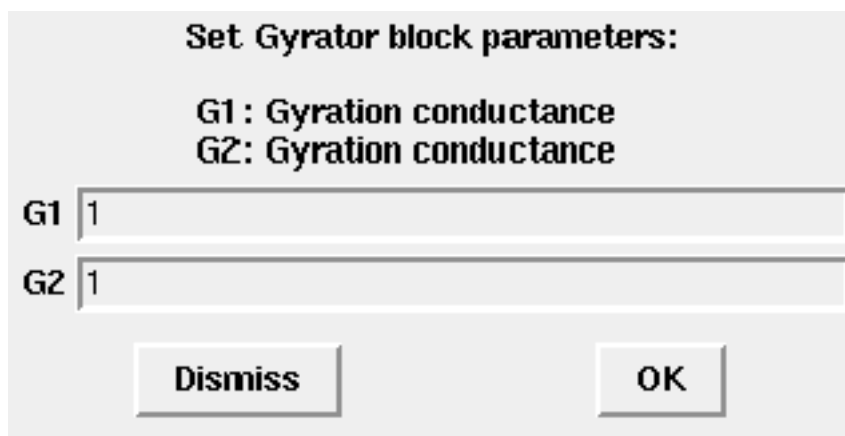
Palette

- Electrical.cosf - Electrical toolbox

Description

A gyrator is a two-port element defined by the following equations: $i_1 = G_2 * v_2$ $i_2 = -G_1 * v_1$ where the constants G_1 , G_2 are called the gyration conductance.

Dialog box



Set Gyrator block parameters:

G1 : Gyration conductance
G2 : Gyration conductance

G1

G2

- G1
Gyration conductance ($-i_2/v_1$) .

Properties : Type 'vec' of size 1.

- **G2**

Gyration conductance ($i1/v2$).

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name :** 'p1'

Implicit **variable**.

- **Modelica variable name :** 'n1'

Implicit **variable**.

- **Modelica variable name :** 'p2'

Implicit **variable**.

- **Modelica variable name :** 'n2'

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name :** 'G1'

Default value : 1

Is a state variable : no.

- **Modelica parameter name :** 'G2'

Default value : 1

Is a state variable : no.

- **File name of the model :** Gyrator

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Gyrator.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Gyrator.mo

Authors

Masoud Najafi - INRIA

Name

IdealTransformer — Ideal Transformer

Block Screenshot



Contents

- Ideal Transformer
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “Authors”

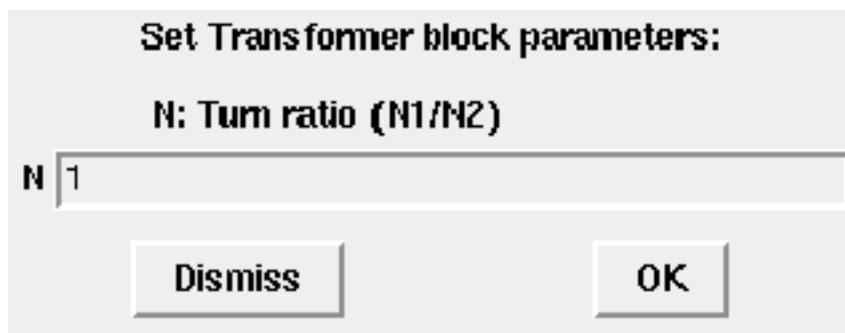
Palette

- Electrical.cosf - Electrical toolbox

Description

The ideal transformer is an ideal two-port resistive circuit element which is characterized by the following two equations: $v_1 = n * v_2$ $i_2 = -n * i_1$ where n is a real number called the turns ratio.

Dialog box



- N
Turns ratio (N1/N2)
Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p1'
Implicit **variable**.
 - **Modelica variable name :** 'n1'
Implicit **variable**.
 - **Modelica variable name :** 'p2'
Implicit **variable**.
 - **Modelica variable name :** 'n2'
Implicit **variable**.
- **Parameters :**
 - **Modelica parameter name :** 'N'
Default value : 1
Is a state variable : no.
- **File name of the model :** IdealTransformer

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/IdealTransformer.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/IdealTransformer.mo

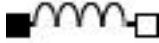
Authors

Masoud Najafi - INRIA

Name

Inductor — Electrical inductor

Block Screenshot



Contents

- Electrical inductor
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

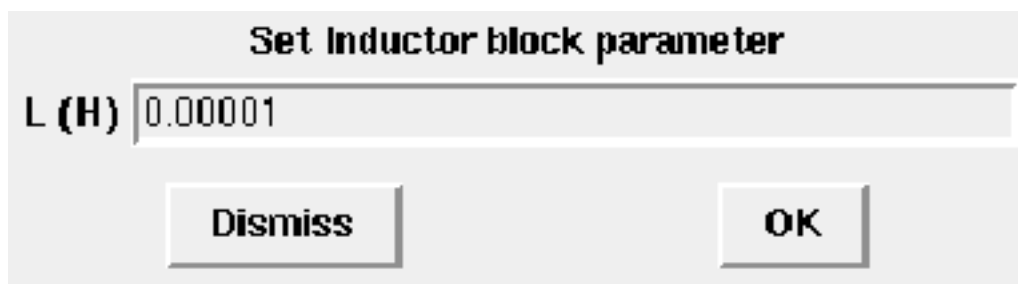
Description

Inductor is an electrical component that can store energy in electrical circuits. The relationship between the voltage across the ports of an inductor of inductance and the current passing through it is given by:

$$v = L \frac{dI}{dt}$$

Inductors can also be used to differentiate between high-frequency and low-frequency signals and this makes them useful in electronic filters. An inductor shows a high impedance for high frequency signals.

Dialog box



- **L (H)**

Inductance

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name :** 'p'

Implicit **variable**.

- **Outputs :**

- **Modelica variable name :** 'n'

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name :** 'L'

Default value : 0.00001

Is a state variable : no.

- **File name of the model :** Inductor

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Inductor.sci

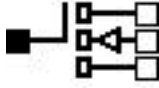
Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Inductor.mo

Name

NMOS — Simple NMOS Transistor

Block Screenshot



Contents

- Simple NMOS Transistor
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “See also”
- the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

Description

The NMos model is a simple model of a n-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A small fixed drain-source resistance is included (to avoid numerical difficulties).

W [m]	L [m]	Beta [1/V ²]	Vt [V]	K2	K5	DW [m]	DL[m]	
12.e-6	4.e-6	.062	-4.5	.24	.61	-1.2e-6	-.9e-6	depletion
60.e-6	3.e-6	.048	.1	.08	.68	-1.2e-6	-.9e-6	enhancement
12.e-6	4.e-6	.0625	-.8	.21	.78	-1.2e-6	-.9e-6	zero
50.e-6	8.e-6	.0299	.24	1.144	.7311	-5.4e-6	-4.e-6	
20.e-6	6.e-6	.041	.8	1.144	.7311	-2.5e-6	-1.5e-6	
30.e-6	9.e-6	.025	-4.	.861	.878	-3.4e-6	-1.74e-6	
30.e-6	5.e-6	.031	.6	1.5	.72	0	-3.9e-6	

50.e-6	6.e-6	.0414	-3.8	.34	.8	-1.6e-6	-2.e-6	depletion
50.e-6	5.e-6	.03	.37	.23	.86	-1.6e-6	-2.e-6	enhancement
50.e-6	6.e-6	.038	-.9	.23	.707	-1.6e-6	-2.e-6	zero
20.e-6	4.e-6	.06776	.5409	.065	.71	-.8e-6	-.2e-6	
20.e-6	4.e-6	.06505	.6209	.065	.71	-.8e-6	-.2e-6	
20.e-6	4.e-6	.05365	.6909	.03	.8	-.3e-6	-.2e-6	
20.e-6	4.e-6	.05365	.4909	.03	.8	-.3e-6	-.2e-6	
12.e-6	4.e-6	.023	-4.5	.29	.6	0	0	depletion
60.e-6	3.e-6	.022	.1	.11	.65	0	0	enhancement
12.e-6	4.e-6	.038	-.8	.33	.6	0	0	zero
20.e-6	6.e-6	.022	.8	1	.66	0	0	

Dialog box

Set NMOS Transistor block parameters

Width [m]

7.777777

Length [m]

3.3000008

Transconductance parameter [A/(V*V)]

3.3000041

Zero bias threshold voltage [V]

1.3

Bulk threshold parameter

1.144

Reduction of pinch-off region

7.7311

Narrowing of channel [m]

-0.0000025

Shortening of channel [m]

-0.0000015

Drain-Source-Resistance [Ohm]

10000000

Dismiss
OK

- Width [m]

W

Properties : Type 'vec' of size 1.

- Length [m]

L

Properties : Type 'vec' of size 1.

- Transconductance parameter [A/(V*V)]

Beta

Properties : Type 'vec' of size 1.

- **Zero bias threshold voltage [V]**

Vt

Properties : Type 'vec' of size 1.

- **Bulk threshold parameter**

K2

Properties : Type 'vec' of size 1.

- **Reduction of pinch-off region**

K5

Properties : Type 'vec' of size 1.

- **Narrowing of channel [m]**

dW

Properties : Type 'vec' of size 1.

- **Shortening of channel [m]**

dL

Properties : Type 'vec' of size 1.

- **Drain-Source-Resistance [Ohm]**

RDS

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name : 'G'**

Implicit variable.

- **Outputs :**

- **Modelica variable name : 'D'**

Implicit variable.

- **Modelica variable name : 'B'**

Implicit variable.

- **Modelica variable name : 'S'**

Implicit variable.

- **Parameters :**

- **Modelica parameter name : 'W'**

Default value : 0.00002

Is a state variable : no.

- **Modelica parameter name :** 'L'

Default value : 0.000006

Is a state variable : no.

- **Modelica parameter name :** 'Beta'

Default value : 0.000041

Is a state variable : no.

- **Modelica parameter name :** 'Vt'

Default value : 0.8

Is a state variable : no.

- **Modelica parameter name :** 'K2'

Default value : 1.144

Is a state variable : no.

- **Modelica parameter name :** 'K5'

Default value : 0.7311

Is a state variable : no.

- **Modelica parameter name :** 'dW'

Default value : -0.0000025

Is a state variable : no.

- **Modelica parameter name :** 'dL'

Default value : -0.0000015

Is a state variable : no.

- **Modelica parameter name :** 'RDS'

Default value : 10000000

Is a state variable : no.

- **File name of the model :** NMOS

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/NMOS.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/NMOS.mo

See also

- PMOS - Simple PMOS Transistor

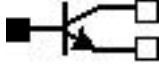
Authors

- www.modelica.org

Name

NPN — NPN transistor

Block Screenshot



Contents

- NPN transistor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - Modelica model
 - the section called “See also”
 - the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

This model is a simple model of a bipolar NPN junction transistor according to Ebers-Moll.

Dialog box

Set NPN block parameters:

Bf : Forward beta	50
Br : Reverse beta	0.1
Is : Transport saturation current	1.e-16
Vak : Early voltage (inverse), 1/Volt	0.02
Tauf : Ideal forward transit time	0.12e-9
Taur : Ideal reverse transit time	5e-9
Ccs : Collector-substrat(ground) cap.	1e-12
Cje : Base-emitter zero bias depletion cap.	0.4e-12
Cjc : Base coll. zero bias depletion cap.	0.5e-12
PhiE : Base-emitter diffusion voltage	0.6
Me : Base-emitter gradation exponent	0.4
PhiC : Base-collector diffusion voltage	0.6
Mc : Base-collector gradation exponent	0.333
Gbc : Base-collector conductance	1e-15
Gbe : Base-emitter conductance	1e-15
Vt : Voltage equivalent of temperature	0.02585
CLminmax : if $x > CLminMax$, the $\exp(x)$ is linearized	40

Dismiss
OK

Parameter	Default value	Description
Bf	50	Forward beta
Br	0.1	Reverse beta
Is	1.e-16	Transport saturation current [A]
Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Tauf	0.12e-9	Ideal forward transit time [s]
Taur	5e-9	Ideal reverse transit time [s]
Ccs	1e-12	Collector-substrat(ground) cap. [F]
Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]

Phie	0.8	Base-emitter diffusion voltage [V]
Me	0.4	Base-emitter gradation exponent
Phic	0.8	Base-collector diffusion voltage [V]
Mc	0.333	Base-collector gradation exponent
Gbc	1e-15	Base-collector conductance [S]
Gbe	1e-15	Base-emitter conductance [S]
Vt	0.02585	Voltage equivalent of temperature [V]
EMin	-100	if $x < E_{Min}$, the $\exp(x)$ function is linearized
EMax	40	if $x > E_{Max}$, the $\exp(x)$ function is linearized

Default properties

- **Inputs :**

- **Modelica variable name :** 'B'

Implicit variable.

- **Outputs :**

- **Modelica variable name :** 'C'

Implicit variable.

- **Modelica variable name :** 'E'

Implicit variable.

- **Parameters :**

- **Modelica parameter name :** 'Bf'

Default value : 50

Is a state variable : no.

- **Modelica parameter name :** 'Br'

Default value : 0.1

Is a state variable : no.

- **Modelica parameter name :** 'Is'

Default value : 0

Is a state variable : no.

- **Modelica parameter name :** 'Vak'

Default value : 0.02

Is a state variable : no.

- **Modelica parameter name :** 'Tauf'

Default value : 1.200E-10

Is a state variable : no.

- **Modelica parameter name :** 'Taur'

Default value : 5.000E-09

Is a state variable : no.

- **Modelica parameter name :** 'Ccs'

Default value : 1.000E-12

Is a state variable : no.

- **Modelica parameter name :** 'Cje'

Default value : 4.000E-13

Is a state variable : no.

- **Modelica parameter name :** 'Cjc'

Default value : 5.000E-13

Is a state variable : no.

- **Modelica parameter name :** 'Phie'

Default value : 0.8

Is a state variable : no.

- **Modelica parameter name :** 'Me'

Default value : 0.4

Is a state variable : no.

- **Modelica parameter name :** 'Phic'

Default value : 0.8

Is a state variable : no.

- **Modelica parameter name :** 'Mc'

Default value : 0.333

Is a state variable : no.

- **Modelica parameter name :** 'Gbc'

Default value : 1.000E-15

Is a state variable : no.

- **Modelica parameter name :** 'Gbe'
Default value : 1.000E-15
Is a state variable : no.
- **Modelica parameter name :** 'Vt'
Default value : 0.02585
Is a state variable : no.
- **Modelica parameter name :** 'EMinMax'
Default value : 40
Is a state variable : no.
- **File name of the model :** NPN

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/NPN.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/NPN.mo

See also

- PNP - PNP transistor

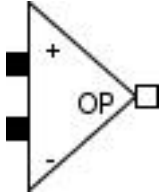
Authors

- www.modelica.org

Name

OpAmp — Ideal opamp (norator-nullator pair)

Block Screenshot



Contents

- Ideal opamp (norator-nullator pair)
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

The ideal OpAmp is a two-port. The left port is fixed to $v_1=0$ and $i_1=0$ (nullator). At the right port both any voltage v_2 and any current i_2 are possible (norator).

Default properties

- **Inputs :**
 - **Modelica variable name :** 'in_p'
Implicit **variable**.
 - **Modelica variable name :** 'in_n'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'out'
Implicit **variable**.
- **File name of the model :** OpAmp

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/OpAmp.sci

Modelica model

- `SCI/modules/scicos_blocks/macros/Electrical/OpAmp.mo`

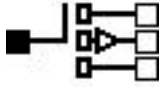
Authors

- www.modelica.org

Name

PMOS — Simple PMOS Transistor

Block Screenshot



Contents

- Simple PMOS Transistor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - Modelica model
 - the section called “See also”
 - the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

The PMOS model is a simple model of a p-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A small fixed drain-source resistance is included (to avoid numerical difficulties).

Some typical parameter sets are:

W [m]	L[m]	Beta [1/V ²]	Vt [V]	K2	K5	DW [m]	DL [m]
50.e-6	8.e-6	.0085	-.15	.41	.839	-3.8e-6	-4.0e-6
20.e-6	6.e-6	.0105	-1.0	.41	.839	-2.5e-6	-2.1e-6
30.e-6	5.e-6	.0059	-.3	.98	1.01	0	-3.9e-6
30.e-6	5.e-6	.0152	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0163	-.69	.104	1.1	-.8e-6	-.4e-6
30.e-6	5.e-6	.0182	-.69	.086	1.06	-.1e-6	-.6e-6
20.e-6	6.e-6	.0074	-1.	.4	.59	0	0

Dialog box

Set PMOS Transistor parameters	
Width [m]	1.100005
Length [m]	0.000008
Transconductance parameter [A/(V*V)]	0.000005
Zero bias threshold voltage [V]	-1
Bulk threshold parameter	0.41
Reduction of pinch-off region	1.339
Narrowing of channel [m]	-0.0000025
Shortening of channel [m]	-0.0000021
Drain-Source-Resistance [Ohm]	10000000
Dismiss OK	

- **Width [m]**
W
Properties : Type 'vec' of size 1.
- **Length [m]**
L
Properties : Type 'vec' of size 1.
- **Transconductance parameter [A/(V*V)]**
Beta
Properties : Type 'vec' of size 1.
- **Zero bias threshold voltage [V]**
Vt
Properties : Type 'vec' of size 1.
- **Bulk threshold parameter**
K2
Properties : Type 'vec' of size 1.
- **Reduction of pinch-off region**
K5
Properties : Type 'vec' of size 1.

- **Narrowing of channel [m]**

dW

Properties : Type 'vec' of size 1.

- **Shortening of channel [m]**

dL

Properties : Type 'vec' of size 1.

- **Drain-Source-Resistance [Ohm]**

RDS

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name : 'G'**

Implicit variable.

- **Outputs :**

- **Modelica variable name : 'D'**

Implicit variable.

- **Modelica variable name : 'B'**

Implicit variable.

- **Modelica variable name : 'S'**

Implicit variable.

- **Parameters :**

- **Modelica parameter name : 'W'**

Default value : 0.00005

Is a state variable : no.

- **Modelica parameter name : 'L'**

Default value : 0.000006

Is a state variable : no.

- **Modelica parameter name : 'Beta'**

Default value : 0.0000105

Is a state variable : no.

- **Modelica parameter name : 'Vt'**

Default value : -1

Is a state variable : no.

- **Modelica parameter name :** 'K2'

Default value : 0.41

Is a state variable : no.

- **Modelica parameter name :** 'K5'

Default value : 0.839

Is a state variable : no.

- **Modelica parameter name :** 'dW'

Default value : -0.0000025

Is a state variable : no.

- **Modelica parameter name :** 'dL'

Default value : -0.0000021

Is a state variable : no.

- **Modelica parameter name :** 'RDS'

Default value : 10000000

Is a state variable : no.

- **File name of the model :** PMOS

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PMOS.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PMOS.mo

See also

- NMOS - Simple NMOS Transistor

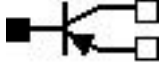
Authors

- www.modelica.org

Name

PNP — PNP transistor

Block Screenshot



Contents

- PNP transistor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model
 - the section called “See also”
 - the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

This model is a simple model of a bipolar PNP junction transistor according to Ebers-Moll.

Dialog box

Set PNP block parameters:

Bf : Forward beta	50
Br : Reverse beta	0.1
Is : Transport saturation current	1e-16
Vak : Early voltage (inverse), 1/Volt	0.02
Tauf : Ideal forward transit time	0.12e-9
Taur : Ideal reverse transit time	5e-9
Ccs : Collector-substrat(ground) cap.	1e-12
Cje : Base-emitter zero bias depletion cap.	0.4e-12
Cjc : Base-coll. zero bias depletion cap.	0.5e-12
Phie : Base-emitter diffusion voltage	0.8
Me : Base-emitter gradation exponent	0.4
Phic : Base-collector diffusion voltage	0.8
Mc : Base-collector gradation exponent	0.011
Gbc : Base-collector conductance	1e-15
Gbe : Base-emitter conductance	1e-5
Vt : Voltage equivalent of temperature	0.02585
LMkMax : if $x > LMkMax$, the $\exp(x)$ function is linearized	40

Dismiss OK

Parameter	Default value	Description
Bf	50	Forward beta
Br	0.1	Reverse beta
Is	1.e-16	Transport saturation current [A]
Vak	0.02	Early voltage (inverse), 1/Volt [1/V]
Tauf	0.12e-9	Ideal forward transit time [s]
Taur	5e-9	Ideal reverse transit time [s]
Ccs	1e-12	Collector-substrat(ground) cap. [F]
Cje	0.4e-12	Base-emitter zero bias depletion cap. [F]
Cjc	0.5e-12	Base-coll. zero bias depletion cap. [F]
Phie	0.8	Base-emitter diffusion voltage [V]
Me	0.4	Base-emitter gradation exponent

Phic	0.8	Base-collector diffusion voltage [V]
Mc	0.333	Base-collector gradation exponent
Gbc	1e-15	Base-collector conductance [S]
Gbe	1e-15	Base-emitter conductance [S]
Vt	0.02585	Voltage equivalent of temperature [V]
EMin	-100	if $x < E_{Min}$, the $\exp(x)$ function is linearized
EMax	40	if $x > E_{Max}$, the $\exp(x)$ function is linearized

Default properties

- **Inputs :**
 - **Modelica variable name :** 'B'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'C'
Implicit **variable**.
 - **Modelica variable name :** 'E'
Implicit **variable**.
- **Parameters :**
 - **Modelica parameter name :** 'Bf'
Default value : 50
Is a state variable : no.
 - **Modelica parameter name :** 'Br'
Default value : 0.1
Is a state variable : no.
 - **Modelica parameter name :** 'Is'
Default value : 0
Is a state variable : no.
 - **Modelica parameter name :** 'Vak'
Default value : 0.02
Is a state variable : no.
 - **Modelica parameter name :** 'Tauf'

Default value : 1.200E-10

Is a state variable : no.

- **Modelica parameter name :** 'Taur'

Default value : 5.000E-09

Is a state variable : no.

- **Modelica parameter name :** 'Ccs'

Default value : 1.000E-12

Is a state variable : no.

- **Modelica parameter name :** 'Cje'

Default value : 4.000E-13

Is a state variable : no.

- **Modelica parameter name :** 'Cjc'

Default value : 5.000E-13

Is a state variable : no.

- **Modelica parameter name :** 'Phie'

Default value : 0.8

Is a state variable : no.

- **Modelica parameter name :** 'Me'

Default value : 0.4

Is a state variable : no.

- **Modelica parameter name :** 'Phic'

Default value : 0.8

Is a state variable : no.

- **Modelica parameter name :** 'Mc'

Default value : 0.333

Is a state variable : no.

- **Modelica parameter name :** 'Gbc'

Default value : 1.000E-15

Is a state variable : no.

- **Modelica parameter name :** 'Gbe'

Default value : 1.000E-15

Is a state variable : no.

- **Modelica parameter name :** 'Vt'

Default value : 0.02585

Is a state variable : no.

- **Modelica parameter name :** 'EMinMax'

Default value : 40

Is a state variable : no.

- **File name of the model :** PNP

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PNP.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PNP.mo

See also

- NPN - NPN transistor

Authors

- www.modelica.org

Name

PotentialSensor — Potential sensor

Block Screenshot



Contents

- Potential sensor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

This block is used to measure the voltage with respect to the reference voltage (**Ground** block) in an electrical circuit. The voltage is given to the explicit part of the model via an explicit output port.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'v'
Explicit **variable**.
- **File name of the model :** PotentialSensor

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PotentialSensor.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PotentialSensor.mo

Name

Resistor — Electrical resistor

Block Screenshot



Contents

- Electrical resistor
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

A resistor is a two-port electrical component that resists an electric current by producing a voltage drop () between its terminals according to the Ohm's law.

$$R = \frac{V}{I}$$

The electrical resistance () is equal to the voltage drop across the resistor divided by the current through the resistor ().

Dialog box



- R (ohm)

Resistance

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
 - Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
 - Implicit **variable**.
- **Parameters :**
 - **Modelica parameter name :** 'R'
 - Default value :** 0.01
 - Is a state variable :** no.
- **File name of the model :** Resistor

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Resistor.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Resistor.mo

Name

SineVoltage — Sine voltage source

Block Screenshot



Contents

- Sine voltage source
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “Authors”

Palette

- Electrical.cosf - Electrical toolbox

Description

This Modelica block a general sine voltage source. The internal ohmic resistance is zero.

Dialog box

Set voltage source parameter	
Amplitude (Volt)	1
phase (rad)	0
Frequency (Hz)	1
Voltageoffset (V)	0
Timeoffset (s)	0
DismissOK	

- **Amplitude (Volt)**

Sine voltage amplitude

Properties : Type 'vec' of size 1.

- **phase (rad)**

phase shift of the sine voltage

Properties : Type 'vec' of size 1.

- **Frequency (Hz)**

Sine voltage frequency

Properties : Type 'vec' of size 1.

- **Voltageoffset (V)**

Offset Voltage of the sine voltage

Properties : Type 'vec' of size 1.

- **Timeoffset (s)**

Start time. During the start time, the signal value is equal to the voltage offset.

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name :** 'p'

Implicit **variable**.

- **Outputs :**

- **Modelica variable name :** 'n'

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name :** 'V'

Default value : 1

Is a state variable : no.

- **Modelica parameter name :** 'phase'

Default value : 0

Is a state variable : no.

- **Modelica parameter name :** 'freqHz'

Default value : 1

Is a state variable : no.

- **Modelica parameter name :** 'offset'
Default value : 0
Is a state variable : no.
- **Modelica parameter name :** 'startTime'
Default value : 0
Is a state variable : no.
- **File name of the model :** SineVoltage

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/SineVoltage.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/SineVoltage.mo

Authors

- www.modelica.org

Name

Switch — Non-ideal electrical switch

Block Screenshot



Contents

- Non-ideal electrical switch
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model
- the section called “Authors”

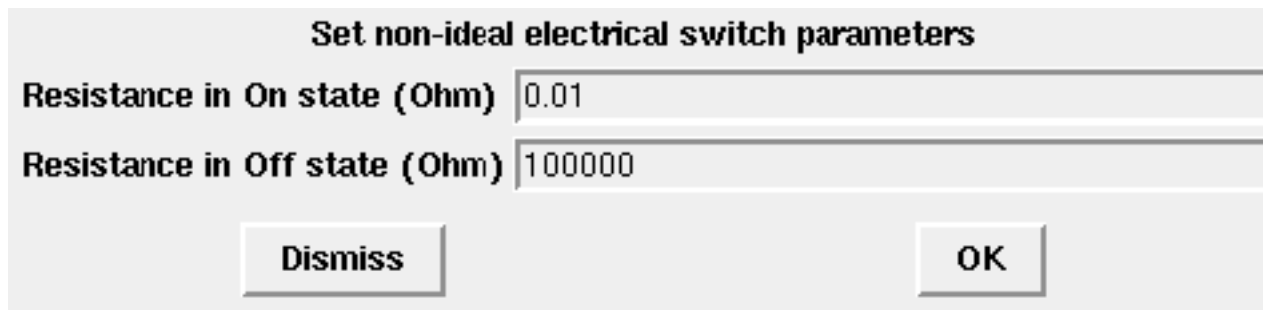
Palette

- Electrical.cosf - Electrical toolbox

Description

This is a non-ideal two-pole switch. If the explicit input become positive, two pins are connected via a resistor of resistance R_{ON}). Otherwise, two pins are connected via R_{OFF} resistance. Note that using this block may result in a stiff model, so try to choose proper error tolerances.

Dialog box



Set non-ideal electrical switch parameters

Resistance in On state (Ohm)

Resistance in Off state (Ohm)

- **Resistance in On state (Ohm)**
Switch resistance when the Switch is closed
Properties : Type 'vec' of size 1.

- **Resistance in Off state (Ohm)**

Switch resistance when the switch is open

Properties : Type 'vec' of size 1.

Default properties

- **Inputs :**

- **Modelica variable name : 'p'**

Implicit **variable**.

- **Modelica variable name : 'inp'**

Explicit **variable**.

- **Outputs :**

- **Modelica variable name : 'n'**

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name : 'Ron'**

Default value : 0.01

Is a state variable : no.

- **Modelica parameter name : 'Roff'**

Default value : 100000

Is a state variable : no.

- **File name of the model :** Switch

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Switch.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Switch.mo

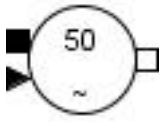
Authors

Masoud Najafi - INRIA

Name

VVsourceAC — Variable AC voltage source

Block Screenshot



Contents

- Variable AC voltage source
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

The variable voltage source block is a model for a controlled AC voltage source. This component provides a sinusoid voltage across its ports. The amplitude of the output voltage is governed by the explicit input and the frequency is defined by the user. The ohmic resistance of the block is zero.

Dialog box



- **Frequency (Hz)**

Frequency of the output sinusoid voltage

Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'

Implicit **variable**.

- **Modelica variable name** : 'VA'

Explicit **variable**.

- **Outputs** :

- **Modelica variable name** : 'n'

Implicit **variable**.

- **Parameters** :

- **Modelica parameter name** : 'f'

Default value : 50

Is a state variable : no.

- **File name of the model** : VVsourceAC

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VVsourceAC.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/VVsourceAC.mo

Name

VariableResistor — Electrical variable resistor

Block Screenshot



Contents

- Electrical variable resistor
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

This component represents a variable ohmic resistor. The resistance () is controlled via an explicit input port.

$$R_x = \frac{V}{I}$$

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit **variable**.
 - **Modelica variable name :** 'R'
Explicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
- **File name of the model :** VariableResistor

Interfacing function

- `SCI/modules/scicos_blocks/macros/Electrical/VariableResistor.sci`

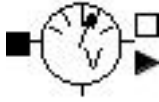
Modelica model

- `SCI/modules/scicos_blocks/macros/Electrical/VariableResistor.mo`

Name

VoltageSensor — Electrical voltage sensor

Block Screenshot



Contents

- Electrical voltage sensor
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Electrical.cosf - Electrical toolbox

Description

This component is used to measure the voltage difference between two nodes in an electrical circuit. The output signal is the difference between the voltages of the black port and the white port, ,

$$V_{output} = V_{black} - V_{white}$$

The ohmic conductance of this block is zero.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit **variable**.
 - **Modelica variable name :** 'v'
Explicit **variable**.
- **File name of the model :** VoltageSensor

Interfacing function

- `SCI/modules/scicos_blocks/macros/Electrical/VoltageSensor.sci`

Modelica model

- `SCI/modules/scicos_blocks/macros/Electrical/VoltageSensor.mo`

Name

VsourceAC — Electrical AC voltage source

Block Screenshot



Contents

- Electrical AC voltage source
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

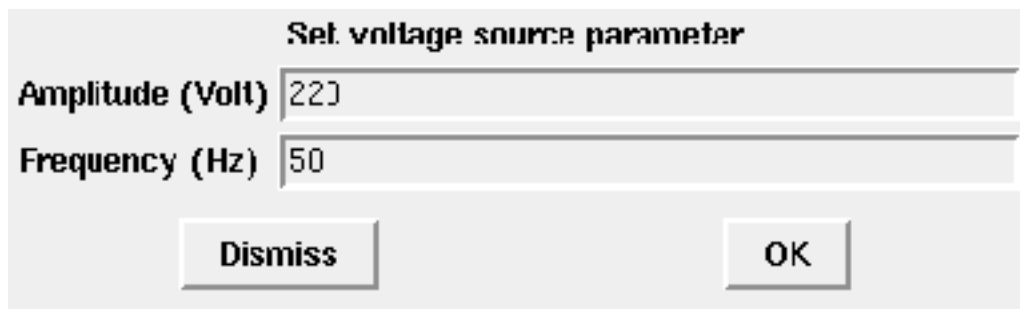
Palette

- Electrical.cosf - Electrical toolbox

Description

This component is an AC voltage source with sinusoid output voltage. The amplitude and the frequency of the output voltage is set by the user. The ohmic resistance of this block is zero.

Dialog box



- **Amplitude (Volt)**
Amplitude of the output sinusoid voltage
Properties : Type 'vec' of size -1.
- **Frequency (Hz)**
Frequency of the output sinusoid voltage

Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'p'
Implicit variable.
- **Outputs :**
 - **Modelica variable name :** 'n'
Implicit variable.
- **Parameters :**
 - **Modelica parameter name :** 'VA'
Default value : 220
Is a state variable : no.
 - **Modelica parameter name :** 'f'
Default value : 50
Is a state variable : no.
- **File name of the model :** VsourceAC

Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VsourceAC.sci

Modelica model

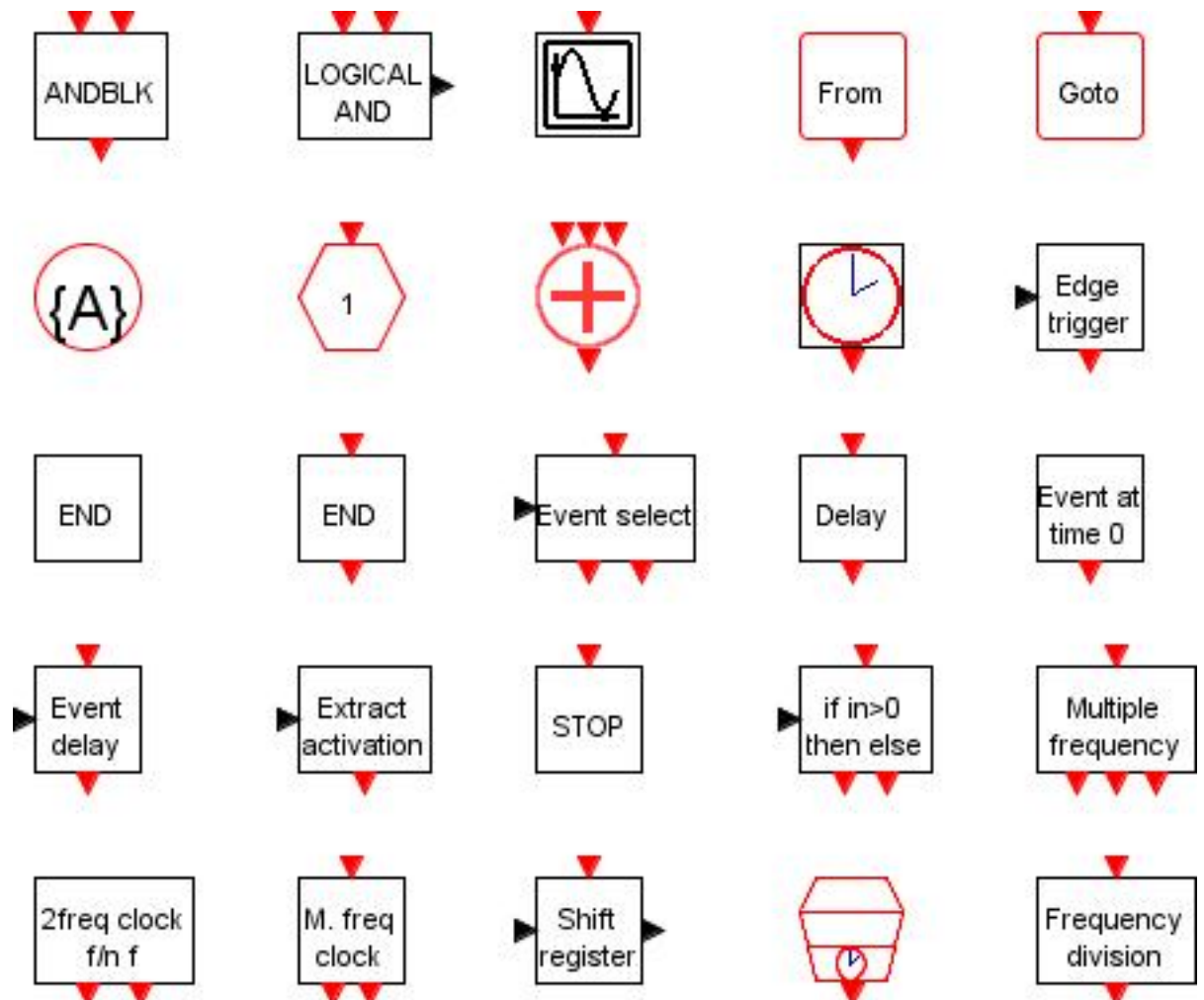
- SCI/modules/scicos_blocks/macros/Electrical/VsourceAC.mo

8. Event handling palette

Name

Events_pal — Event handling palette

Block Screenshot



Module

- xc0s

Description

The Event handling palette is used to handle events in the diagram. It contains several activation clocks, synchronous blocks and blocks to gather events in a single link.

Blocks

- ANDBLK - Activation and
- ANDLOG_f - Logical and
- CEVENTSCOPE — Activation scope
- CLKFROM — Receives data from a corresponding CLKGOTO
- CLKGOTO — Pass block input to CLKFROM block

- CLKGotoTagVisibility — Define Scope of CLKGOTO tag visibility
- CLKOUTV_f — Output activation port
- CLKSOMV_f — Activation union
- CLOCK_c — Activation clock
- EDGE_TRIGGER — EDGE_TRIGGER block
- ENDBLK — END block
- END_c — END_c block
- ESELECT_f — Synchronous block Event-Select
- EVTDLY_c — Event delay
- EVTGEN_f — Event generator
- EVTWARDLY — Event variable delay
- Extract_Activation — Extract_Activation block
- HALT_f — Stop simulation
- IFTHEL_f — Synchronous block If-Then-Else
- M_freq — Multiple Frequencies
- MCLOCK_f — Clock Frequency division
- MFCLCK_f — Clock Multiple Frequencies
- REGISTER — Shift Register
- SampleCLK — Sample Time Clock
- freq_div — Frequency division

Name

ANDBLK — Activation and

Block Screenshot



Contents

- Activation and
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “Authors”

Palette

- Event handling palette

Description

The Bus Creator block combines a set of signals, i.e., a group of signals represented by a single line in a block diagram. It allows you to reduce the number of lines required to route signals from one part of a diagram to another. This makes your easier to understand.

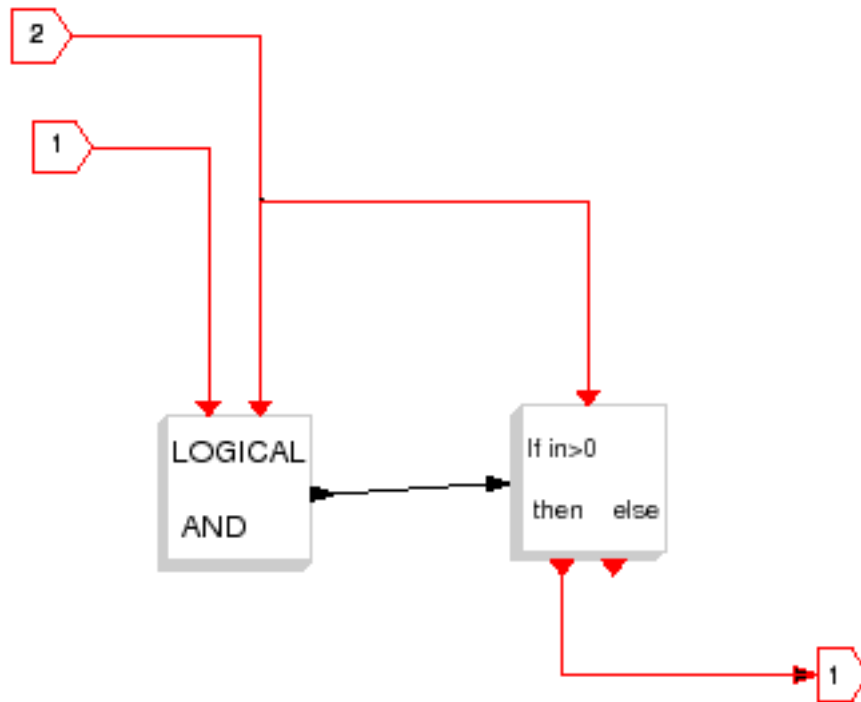
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/ANDBLK.sci

Compiled Super Block content



Authors

Ramine Nikoukhah - INRIA

Name

ANDLOG_f — Logical and

Block Screenshot



Contents

- Logical and
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- Event handling palette

Description

This block, with two event inputs and a regular output, outputs **+1** or **-1** on its regular output depending on input events.

- **+1** : When events are synchronously (present on both event input ports),
- **-1** : When only one event is present.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - **port 1** : size [1,1] / type 1
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *andlog*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/ANDLOG_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

CEVENTSCOPE — Activation scope

Block Screenshot



Contents

- Activation scope
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Event handling palette

Description

This block realizes the visualization of the input event signals.

Dialog box

Set Scope parameters	
Number of event inputs	1
colors c (>0) or mark (<0)	1
Output window number (-1 for automatic)	-1
Output window position	0
Output window sizes	[0C0;40C]
Refresh period	30
DismissOK	

- Number of event inputs

an integer giving the number of event input ports colors : a vector of integers. The i-th element is

≥ 0 < 0

the color number () or dash type () used to draw the evolution of the i-th input port signal. See **set** for color (dash type) definitions.

Properties : Type 'vec' of size 1

- **colors c**

> 0 < 0

an integer. It is the color number () or dash type () used to draw the evolution of the input port signal. See **plot2d** for color (dash type) definitions.

Properties : Type 'vec' of size -1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired). Output window position : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size 1

- **Output window position**

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Properties : Type 'vec' of size -1

- **Refresh period**

Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *cevscpe*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CEVENTSCOPE.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cevscpe.c (Type 4)

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CLKFROM — Receives data from a corresponding CLKGOTO

Block Screenshot



Contents

- Receives data from a corresponding CLKGOTO
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

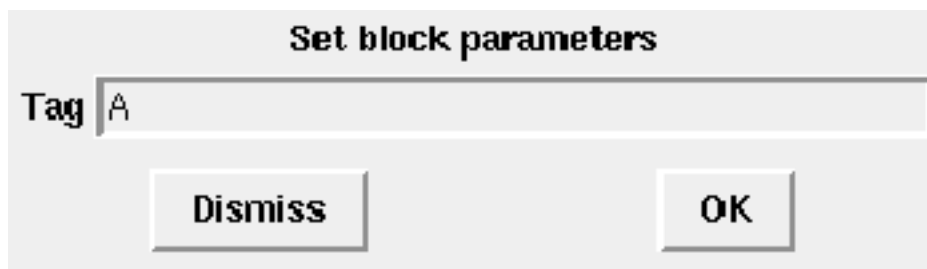
Palette

- Event handling palette

Description

This block is used to connect events ports. For more information on how it works please refer to the documentation of the FROM block by clicking on the link in the "See also" field.

Dialog box



- **Tag**

The tag of the CLKGOTO block passing the signal to this CLKFROM block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkfrom*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKFROM.sci

See also

- FROM - FROM Receives data from a corresponding GOTO

Authors

Fady NASSIF - INRIA

Name

CLKGOTO — Pass block input to CLKFROM block

Block Screenshot



Contents

- Pass block input to CLKFROM block
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

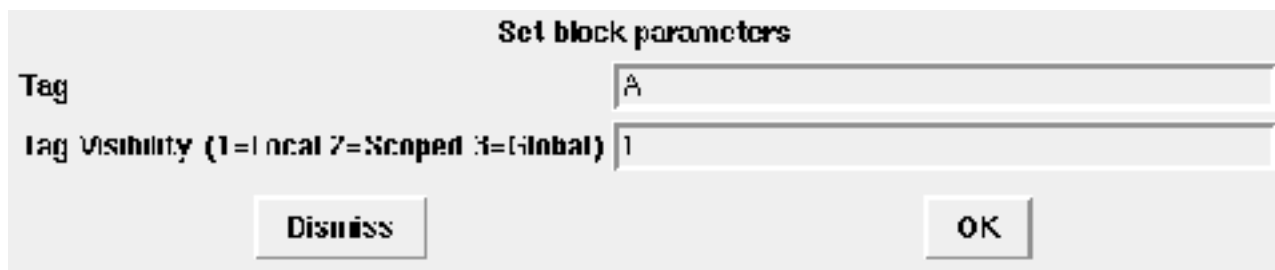
Palette

- Event handling palette

Description

This block is used to connect events ports. For more information on how it works please refer to the documentation of the GOTO block by clicking on the link in the "See also" field.

Dialog box



Set block parameters

Tag

Tag Visibility (1=Local 2=Scoped 3=Global)

- **Tag**
This parameter identifies the Goto block whose scope is defined in this block.
Properties : Type 'str' of size -1.
- **Tag Visibility (1=Local 2=Scoped 3=Global)**

This parameter identifies the visibility of the block. It can be local(1), scoped(2) or global(3).

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkgoto*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKGOTO.sci

See also

- GOTO - GOTO Pass block input to From block

Authors

Fady NASSIF - INRIA

Name

CLKGotoTagVisibility — Define Scope of CLKGOTO tag visibility

Block Screenshot



Contents

- Define Scope of CLKGOTO tag visibility
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “See also”
 - the section called “Authors”

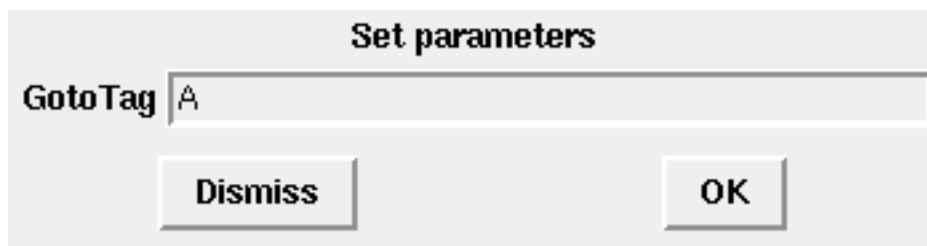
Palette

- Event handling palette

Description

This block is used in the event case. For more information on how it works please refer to the documentation of the GotoTagVisibility block by clicking on the link in the "See also" field.

Dialog box



- **GotoTag**

The Goto block tag whose visibility is defined by the location of this block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkgototagvisibility*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKGotoTagVisibility.sci

See also

- GotoTagVisibility - Define Scope of GOTO tag visibility

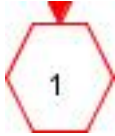
Authors

Fady NASSIF - INRIA

Name

CLKOUTV_f — Output activation port

Block Screenshot



Contents

- Output activation port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

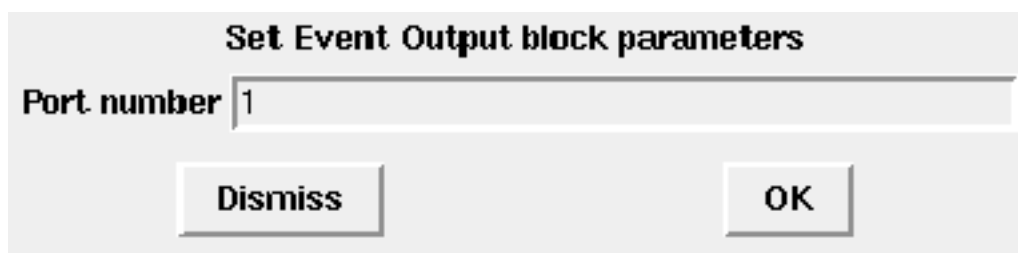
Palette

- Event handling palette

Description

This block must only be used inside Xcos Super Blocks to represent an event output port. In a Super Block, the event output ports must be numbered from 1 to the number of event output ports.

Dialog box



- **Port number**
an integer defining the port number.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *output*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CLKOUTV_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

CLKSOMV_f — Activation union

Block Screenshot



Contents

- Activation union
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Event handling palette

Description

This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOMV is not a Xcos block because it is discarded at the compilation phase. The inputs and output of CLKSOMV are synchronized.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 3
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sum*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/CLKSOMV_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/sum.c (Type 0)

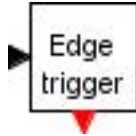
Authors

Ramine Nikoukhah - INRIA

Name

EDGE_TRIGGER — EDGE_TRIGGER block

Block Screenshot



Contents

- EDGE_TRIGGER block
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content

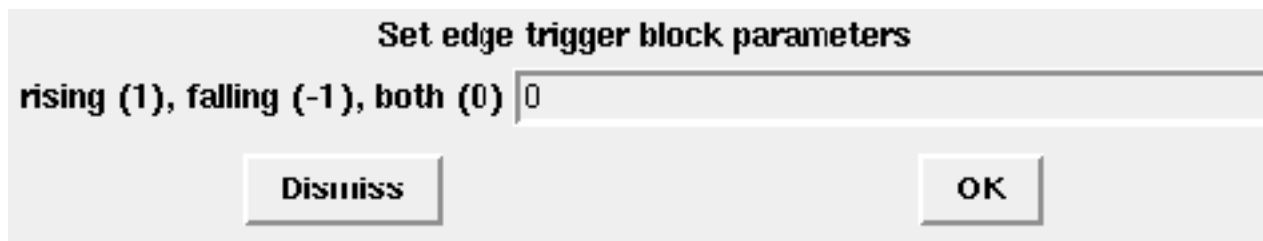
Palette

- Event handling palette

Description

This block generates an event on rising, falling or both edges of the input signal (depending on block parameter). A rising edge is a change in value from strictly negative to positive or zero, or a change in value from zero to strictly positive. A falling edge is the opposite. Note that this block only generates an event if the input jumps due to an event. The generated event is synchronous with the event causing the jump. This block does not detect continuous-time zero-crossings.

Dialog box



- **rising (1), falling (-1), both (0)**

Properties : Type 'vec' of size 1.

Default properties

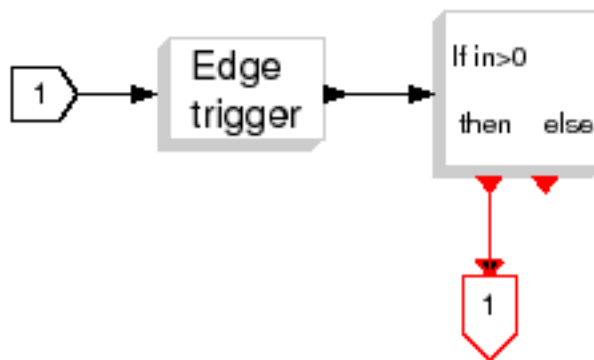
- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/EDGE_TRIGGER.sci

Compiled Super Block content



Name

ESELECT_f — Synchronous block Event-Select

Block Screenshot



Contents

- Synchronous block Event-Select
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

Palette

- Event handling palette

Description

Special block similar to If-Then-Else. Input and output are synchronized. The incoming event is directed to one of the output event ports depending on the value of the regular input.

For example, when the input value is between 0 and 1, the control input is redirected to the first command output; when the input value is between 1 and 2, the control input is redirected to the second command output; et cætera...

Dialog box

Set ESELECT block parameters

number of output event ports	2
Inherit (1: no, 0: yes)	1
zero-crossing (0: no, 1: yes)	0

- number of output event ports

A scalar. Number of output event ports.

Properties : Type 'vec' of size 1

- **Inherit**

If no, then it inherits the event from event input port, elseif yes, then event is activated by regular input port.

Properties : Type 'vec' of size 1

- **zero-crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [1,1] / type -1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *eselect*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/ESELECT_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

EVTDLY_c — Event delay

Block Screenshot



Contents

- Event delay
- • the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Event handling palette

Description

One event is generated **Delay** after an event enters the unique input event port. Block may also generate an initial output event. The event date of that block is computed by the formula :

$$t_i = t_{\text{init}} + i * T_{\text{delay}},$$

where t_{init} the date of initial output event, T_{delay} the delay and i and internal integer discrete counter.

Dialog box

Set Event Delay block parameters
Delay is the delay between an input event and the generated output event
Block may initially generate an output event before any input event. "Date of initial output event" gives the date of this event. Set a negative value if no initial event required

Delay

Date of initial output event

- **Delay**

scalar. Time delay between input and output event.

Properties : Type 'vec' of size 1

- **Date of initial output event**

scalar. If **Auto-exec** ≥ 0 block initially generates an output event at date **Auto-exec** .

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evtdly4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTDLY_c.sci

Computational function

- `SCI/modules/scicos_blocks/src/c/evtdly4.c` (Type 4)

See also

- `CLOCK_c` - Activation clock

Authors

Alan Layec - INRIA

Name

EVTGEN_f — Event generator

Block Screenshot



Contents

- Event generator
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Event handling palette

Description

One event is generated on the unique output event port if Event time is larger than equal to zero, if not, no event is generated.

Dialog box



- **Event Time**
scalar. date of the initial event.
Properties : Type 'vec' of size 1

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *trash*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTGEN_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/trash.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

EVTVARDLY — Event variable delay

Block Screenshot



Contents

- Event variable delay
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

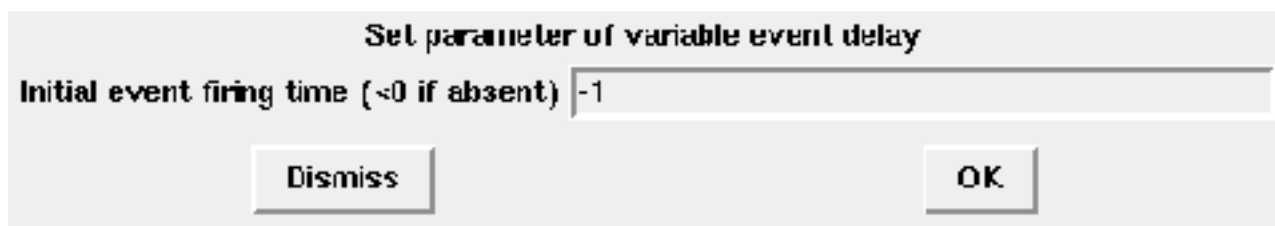
Palette

- Event handling palette

Description

One event is generated Delay after an event enters the unique input event port. The value of the delay is read from the regular input port. Block may also generate an initial output event.

Dialog box



- **Initial event firing time**

One event is generated on the unique output event port if Event time is larger than equal to zero, if not, no event is generated.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evtvardly*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTVARDLY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/evtvardly.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

Extract_Activation — Extract_Activation block

Block Screenshot



Contents

- Extract_Activation block
- • the section called “Palette”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content

Palette

- Event handling palette

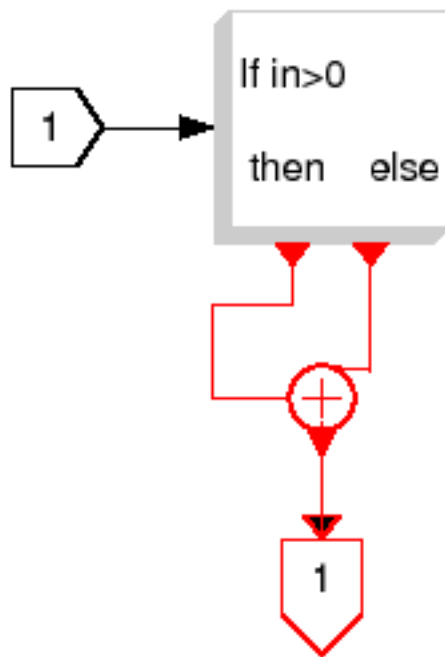
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/Extract_Activation.sci

Compiled Super Block content



Name

HALT_f — Halt

Block Screenshot



Contents

- Halt
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Event handling palette

Description

This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Xcos window is activated. Simulation can be restarted or continued (Run button).

Dialog box



- **State on halt**

A scalar value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *hltblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/HALT_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/hltblk.f (Type 0)

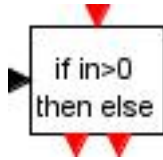
Authors

Ramine Nikoukhah - INRIA

Name

IFTHEL_f — Synchronous block If-Then-Else

Block Screenshot



Contents

- Synchronous block If-Then-Else
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

Palette

- Event handling palette

Description

One event is generated on one of the output event ports when an input event arrives. Depending on the sign of the regular input, the event is generated on the first or second output. This is a Synchro block, , input and output event are synchronized.

Dialog box

Set parameters	
Inherit (1: no, 0: yes)	1
zero-crossing (0: no, 1: yes)	1
Dismiss OK	

- **Inherit**

If no, then it inherits the event from event input port, elseif yes, then event is activated by regular input port.

Properties : Type 'vec' of size 1.

- **zero-crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [1,1] / type -1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ifthel*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/IFTHEL_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

MCLOCK_f — MCLOCK_f title

Block Screenshot



Contents

- Multiple Frequencies f/n
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Compiled Super Block content

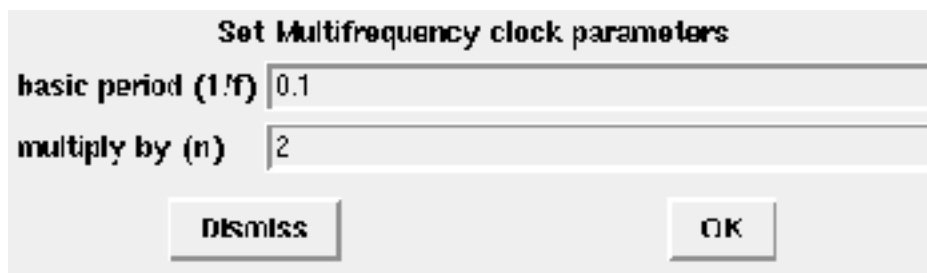
Palette

- Event handling palette

Description

Add here a paragraph of the function description

Dialog box



- **basic period (1/f)**

The parameter description 1.

Properties : Type 'vec' of size 1.
- **multiply by (n)**

The parameter description 2.

Properties : Type 'vec' of size 1.

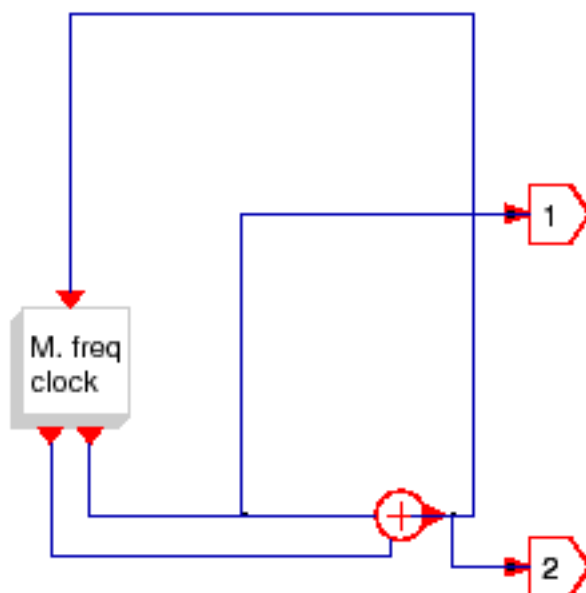
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/MCLOCK_f.sci

Compiled Super Block content



Name

MFCLCK_f — MFCLCK_f title

Block Screenshot



Contents

- Clock Frequency division
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

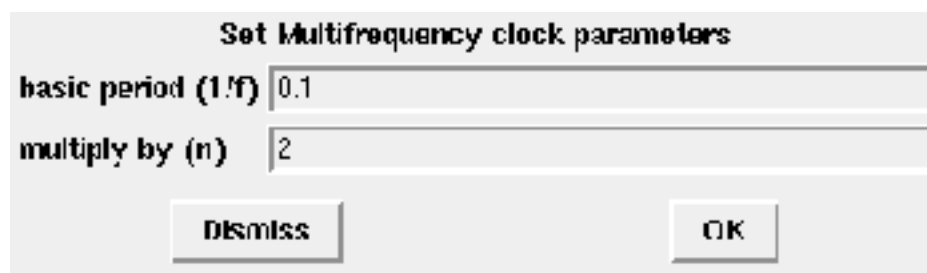
Palette

- Event handling palette

Description

Add here a paragraph of the function description

Dialog box



- **basic period (1/f)**
The parameter description 1.
Properties : Type 'vec' of size 1.
- **multiply by (n)**
The parameter description 2.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *mfclck*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/MFCLCK_f.sci

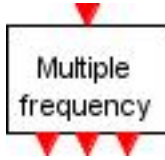
Computational function

- SCI/modules/scicos_blocks/src/fortran/mfclck.f (Type 0)

Name

M_freq — Multiple Frequencies

Block Screenshot



Contents

- Multiple Frequencies
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Event handling palette

Description

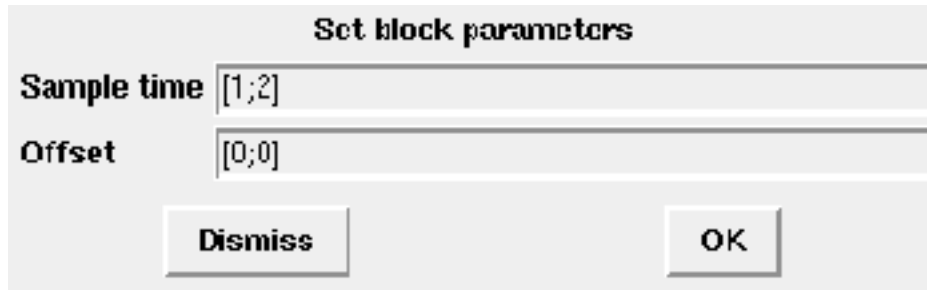
This block generates events at specific sample time of the simulation time. The sample time is given in the "Sample Time" field and the offset is given in the "Offset" field. This block has one event input, the number of event outputs depends on the number of different sample time. For example if the vector of sample time is [1 1 2] and the vector of offset is [0 .5 0] then the block has 7 outputs.

- The first output is activated when the simulation time is equal to a multiple of the first sample time plus the first offset
- The second output is activated when the simulation time is equal to a multiple of the second sample time plus the second offset.
- The third output is activated when we have both cases, first case and second case.
- The fourth output is activated when the simulation time is equal to a multiple of the third sample time plus the third offset.
- The fifth output is activated when we have both cases, first case and fourth case.
- The sixth output is activated when we have both cases, second case and fourth case.
- The seventh output is activated when we have both cases, third case and fourth case.

etc...

So the number of outputs is equal to $2^{\text{number of different time values}}$. Each of these time values is represented by a binary number associated to the output's number in decimal.

Dialog box

A dialog box titled "Set block parameters" with two input fields. The first field is labeled "Sample time" and contains the text "[1;2]". The second field is labeled "Offset" and contains the text "[0;0]". At the bottom of the dialog box, there are two buttons: "Dismiss" on the left and "OK" on the right.

- **Sample time**

Vector of sample time values.

Properties : Type 'vec' of size -1.

- **Offset**

Vector of offset values. Must have the same size as the Sample time and each offset value must be less than its corresponding sample time.

Properties : Type 'vec' of size -1.

Example

```
Let us take the example where the sample time is equal to [1 1 2] and the offset is [0 0 0].
When t=0, the fifth output is activated (001 + 100).
When t=0.5, the second output is activated (010).
When t=1, the first output is activated (001).
When t=1.5, the second output is activated (010).
When t=2 we loop back to 0.
```

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 3
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *m_frequ*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/M_freq.sci

Computational function

- SCI/modules/scicos_blocks/src/c/m_frequ.c (Type 4)

See also

- MFCLCK_f - MFCLCK_f title

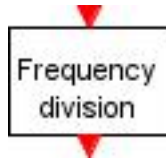
Authors

Fady NASSIF - INRIA

Name

freq_div — Frequency division

Block Screenshot



Contents

- Frequency division
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “Authors”

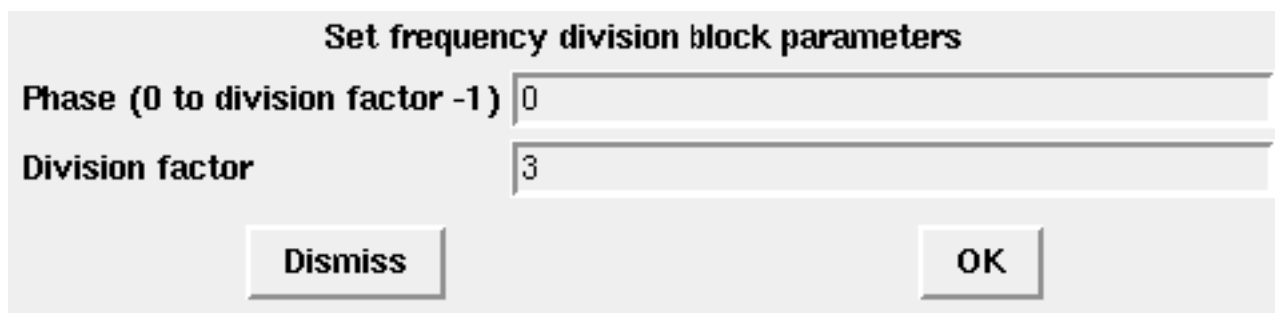
Palette

- Event handling palette

Description

This block is a Super Block. The input event is directed once every n times to output. The input is driven by an event clock.

Dialog box



Set frequency division block parameters

Phase (0 to division factor -1)

Division factor

- **Phase**
positive scalar.
Properties : Type 'vec' of size 1
- **Division factor**

an integer greater than 1.

Properties : Type 'vec' of size 1

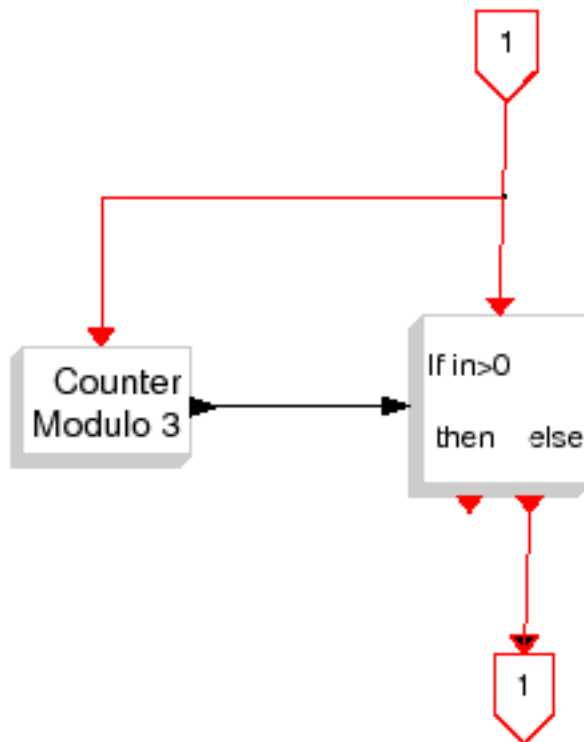
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/freq_div.sci

Compiled Super Block content



Authors

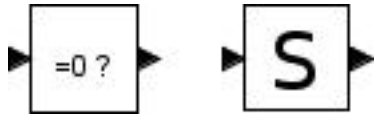
Ramine Nikoukhah - INRIA

9. Implicit palette

Name

Implicit_pal — Implicit palette

Block Screenshot



Module

- xcOS

Description

In the Implicit palette, you can find blocks used to model implicit systems.

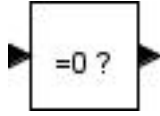
Blocks

- CONSTRAINT_f - Constraint
- DIFF_f - Sum

Name

CONSTRAINT_f — Constraint

Block Screenshot



Contents

- Constraint
- the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Implicit palette

Description

Defines implicit algebraic relations.

Dialog box



- **Set number of constraints**
no of algebraic relations to be defined.
Properties : Type 'vec' of size 1

Default properties

- **always active:** yes
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *constraint*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/CONSTRAINT_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/constraint.f (Type 10001)

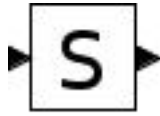
Authors

Ramine Nikoukhah - INRIA

Name

DIFF_f — Derivative

Block Screenshot



Contents

- Derivative
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

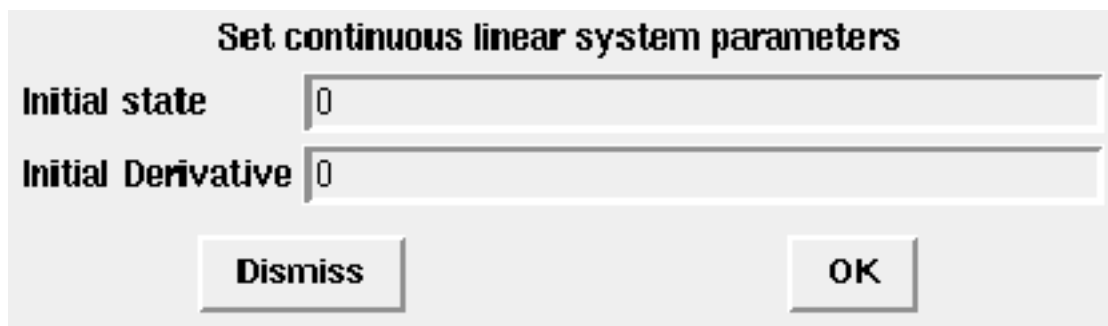
Palette

- Implicit palette

Description

This block computes the derivative of the input.

Dialog box



Set continuous linear system parameters

Initial state

Initial Derivative

Dismiss **OK**

- **Initial state**
The initial continuous state.
Properties : Type 'vec' of size 1.
- **Initial Derivative**
The initial derivative state.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *diffblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/DIFF_f.sci

Computational function

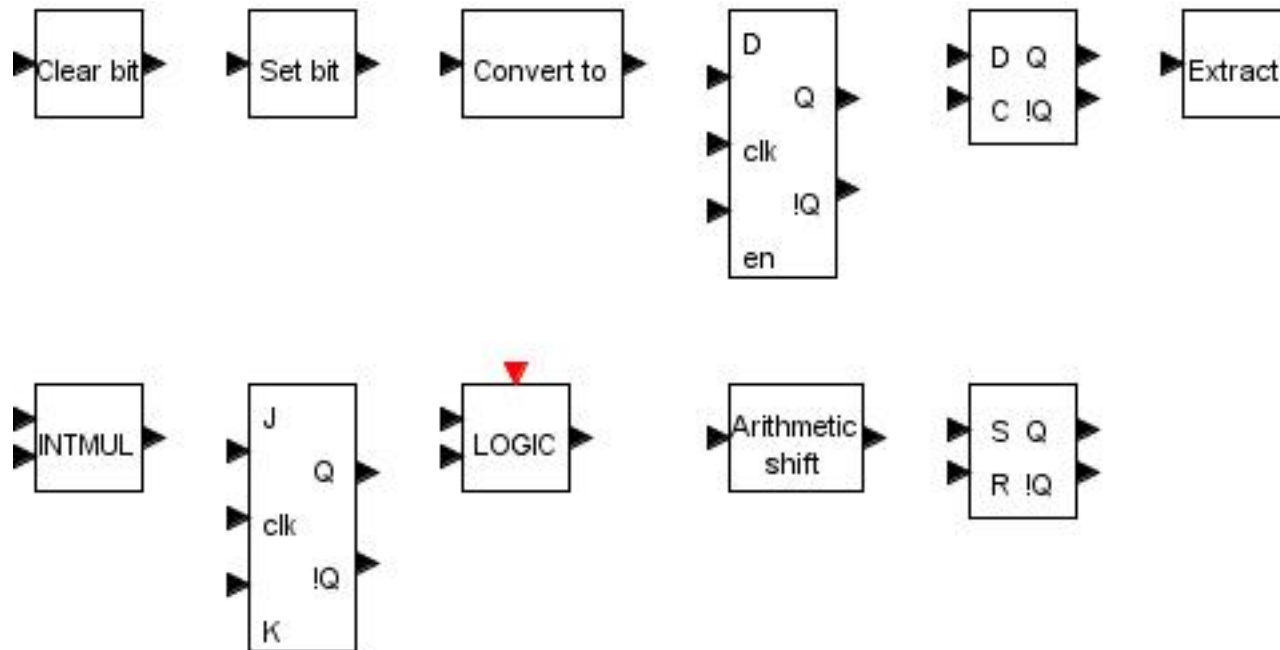
- SCI/modules/scicos_blocks/src/fortran/diffblk.f (Type 10001)

10. Integer palette

Name

Integer_pal — Integer palette

Block Screenshot



Module

- xcios

Description

That palette is dedicated to handle integer numbers. Various basic operators for the management of bit fields and for logic are implemented as well as common gates encountered in digital circuits.

Blocks

- BITCLEAR - BITCLEAR Clear a Bit
- BITSET - BITSET Set a Bit
- CONVERT - CONVERT Data Type Conversion
- DFLIPFLOP - D flip-flop
- DLATCH - D latch flip-flop
- EXTRACTBITS - EXTRACTBITS
- INTMUL - INTMUL integer matrix multiplication
- JKFLIPFLOP - JK flip-flop
- LOGIC - Combinational Logic
- SHIFT - SHIFT Shift Bits
- SRFLIPFLOP - SR flip-flop

Name

BITCLEAR — Clear a Bit

Block Screenshot



Contents

- Clear a Bit
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

This blocks set the specified bit of the integer input to 0. The user can specify the bit in the field: "index of bit". Bit 0 is the least significant bit.

Dialog box

A dialog box titled "Set parameters" with a light gray background. It contains two input fields. The first field is labeled "Datatype(3=int32 4=int16 5=int8 ...)" and has the value "3" entered. The second field is labeled "index of bit (0 is least significant)" and has the value "0" entered. At the bottom of the dialog box, there are two buttons: "Dismiss" on the left and "OK" on the right.

- **Datatype(3=int32 4=int16 5=int8 ...)**

It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

Properties : Type 'vec' of size 1.

- **index of bit (0 is least significant)**

It indicate the index of the bit to clear. When the type is int32 or uint32 the number must be positive and less than 32. When the type is int16 or uint16 the number must be positive and less than 16. When the type is int8 or uint8 the number must be positive and less than 8.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 3
- **regular outputs:**
 - port 1 : size [1,1] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bit_clear_32*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/BITCLEAR.sci

Computational function

- SCI/modules/scicos_blocks/src/c/bit_clear_32.c
- SCI/modules/scicos_blocks/src/c/bit_clear_16.c
- SCI/modules/scicos_blocks/src/c/bit_clear_8.c

See also

- BITSET - BITSET Set a Bit

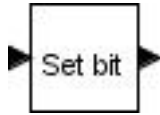
Authors

Fady NASSIF - INRIA

Name

BITSET — Set a Bit

Block Screenshot



Contents

- Set a Bit
- the section called "Palette"
- the section called "Description"
- the section called "Dialog box"
- the section called "Default properties"
- the section called "Interfacing function"
- the section called "Computational function"
- the section called "See also"
- the section called "Authors"

Palette

- Integer palette

Description

This blocks set the specified bit of the integer input to 1. The user can specify the bit in the field: "index of bit". Bit 0 is the least significant bit.

Dialog box

A dialog box titled "Set parameters" with a light gray background. It contains two input fields. The first field is labeled "Datatype(3=int32 4=int16 5=int8 ...)" and has the value "3" entered. The second field is labeled "index of bit (0 is least significant)" and has the value "0" entered. At the bottom of the dialog, there are two buttons: "Dismiss" on the left and "OK" on the right.

- **Datatype(3=int32 4=int16 5=int8 ...)**

It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

Properties : Type 'vec' of size 1.

- **index of bit (0 is least significant)**

It indicates the index of the bit to clear. When the type is int32 or uint32 the number must be positive and less than 32. When the type is int16 or uint16 the number must be positive and less than 16. When the type is int8 or uint8 the number must be positive and less than 8.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 3
- **regular outputs:**
 - port 1 : size [1,1] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bit_set_32*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/BITSET.sci

Computational function

- SCI/modules/scicos_blocks/src/c/bit_set_32.c
- SCI/modules/scicos_blocks/src/c/bit_set_16.c
- SCI/modules/scicos_blocks/src/c/bit_set_8.c

See also

- BITCLEAR - BITCLEAR Clear a Bit

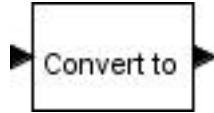
Authors

Fady NASSIF - INRIA

Name

CONVERT — Data Type Conversion

Block Screenshot



Contents

- CONVERT Data Type Conversion
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Integer palette

Description

This block converts an input signal of any data type to a specified data type. The input can be real, complex or integer. When the output is an integer and when overflow occurs the block three different forms of results :

- 1- A normal non saturated result.
- 2- A saturated result.
- 3- An error message warning the user about the overflow..

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

Dialog box

Set CONVERT block parameters

input type (1= double 3=int32 4=int16 5=int8 ...)	1
output type (1= double 3=int32 4=int16 5=int8 ...)	3
Do on Overflow (0=Nothing 1=Saturate 2=Error)	0

- **input type (1= double 3=int32 4=int16 5=int8 ...)**

It indicates the input data type, it can be a double or an integer.

Properties : Type 'vec' of size 1.

- **output type (1= double 3=int32 4=int16 5=int8 ...)**

It indicates the output data type, it can be a double or an integer.

Properties : Type 'vec' of size 1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

When this parameter is set to zero the result is similar to a normal multiplication of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *convert*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/CONVERT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/convert.c

Authors

Fady NASSIF - INRIA

Name

DFLIPFLOP — D flip-flop

Block Screenshot



Contents

- D flip-flop
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

The DFLIPFLOP block outputs the input state when the enable is set and on the rising edge of the clock. The input is D the enable is en and the clock is clk. Q and !Q are the outputs of this block. This block is almostly used with digital number, the input data type is int8.

The truth table of this block is

en	D	Q	!Q
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Default properties

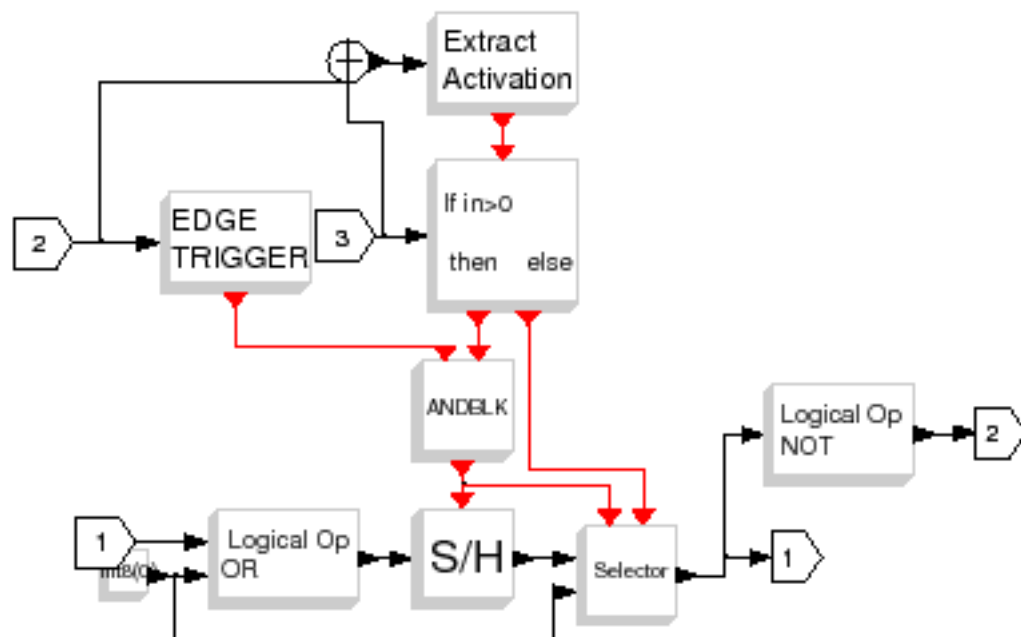
- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

- port 1 : size [1,1] / type 5
- port 2 : size [1,1] / type 1
- port 3 : size [1,1] / type 1
- regular outputs:
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5
- number/sizes of activation inputs: 0
- number/sizes of activation outputs: 0
- continuous-time state: no
- discrete-time state: no
- object discrete-time state: no
- name of computational function: *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/DFLIPFLOP.sci

Compiled Super Block content



See also

- DLATCH - D latch flip-flop
- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

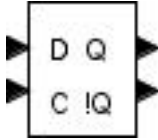
Authors

Fady NASSIF - INRIA

Name

DLATCH — D latch flip-flop

Block Screenshot



Contents

- D latch flip-flop
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

This block outputs the input state when the input gate is high. The input is D the enable is C. Q and !Q are the outputs of this block. This block is almost used with digital number, the input data type is int8.

The truth table of this block is

C	D	Q	!Q
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Default properties

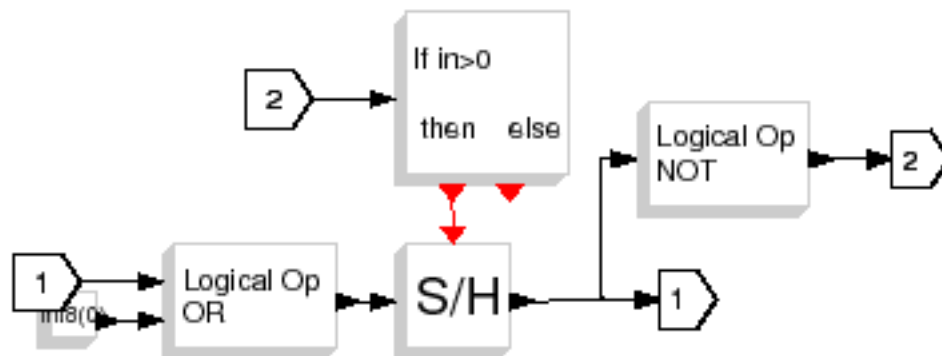
- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

- port 1 : size [1,1] / type 5
- port 2 : size [1,1] / type -1
- regular outputs:
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5
- number/sizes of activation inputs: 0
- number/sizes of activation outputs: 0
- continuous-time state: no
- discrete-time state: no
- object discrete-time state: no
- name of computational function: *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/DLATCH.sci

Compiled Super Block content



See also

- DFLIPFLOP - D flip-flop
- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

Authors

Fady NASSIF - INRIA

Name

EXTRACTBITS — EXTRACTBITS

Block Screenshot



Contents

- EXTRACTBITS
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

This block outputs a contiguous selection of bits from the input integer number. The Bits to extract defines the method by which the user select the output bits.

1- When "Upper Half" is selected the block outputs the half of the input that contain the most significant bit. In this case the third parameter "number of bits or index of bit" is ignored.

2- When "Lower Half" is selected the block outputs the half of the input that contain the least significant bit. In this case the third parameter "number of bits or index of bit" is ignored.

3- When "Range starting with most significant bit" is selected the block outputs certain number of bits of the input that contain the most significant bit. In this case the third parameter "number of bits or index of bit" defines the number of bits to extract.

4- When "Range ending with least significant bit" is selected the block outputs certain number of bits of the input that contain the least significant bit. In this case the third parameter "number of bits or index of bit" defines the number of bits to extract.

5- When "Range of bits" is selected the block outputs a range of bits of the input. In this case the third parameter "number of bits or index of bit" defines the range of bits to extract, it must be a vector with the format [start,end].

The extracted value depends on the forth parameter "Treat bit field as an integer". When it is set to 0 the input scaling is used to determine the output scaling. When it is set to 1, only the extracted bits forms the output number.

Dialog box

Set parameters

-Datatype : set the integer type
3=int32, 4=int16, 5=int8, ...

-Bits to extract :
1=Upper Half
2=Lower Half
3=Range from MSB
4=Range to LSB
5=Range of bits

-Number of bits or index of bit :
case range of bits:[start,end],0 is LSB

-Treat bit field as an integer (0=no 1=yes)

Datatype	3
Bits to extract	1
Number of bits or index of bit	0
Treat bit field as an integer	0

- **Datatype(3=int32 4=int16 5=int8 ...)**

It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

Properties : Type 'vec' of size 1.

- **Bits to extract(1=Upper Half 2=Lower Half 3=Range starting with most significant bit 4=Range ending with least significant bit 5=Range of bits)**

It indicates the mode used to extract bits from the input data.

Properties : Type 'vec' of size 1.

- **number of bits or index of bit (case range of bits:[start,end],0 is least significant bit)**

When the "Bits to extract" field is set to 3 or 4, this parameter is used to determine the number of bits to extract and it must be a number. When the "Bits to extract" field is set to 5, this parameter is used to determine range of bits to extract and it must have the [start,end] form vector. When the "Bits to extract" field is set to 1 or 2, this parameter is ignored.

Properties : Type 'vec' of size -1.

- **Treat bit field as an integer(0=no 1=yes)**

It indicates the scaling mode to use on the output bits selection.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 3
- **regular outputs:**
 - port 1 : size [1,1] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extract_bit_32_UH0*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/EXTRACTBITS.sci

Computational function

- SCI/modules/scicos_blocks/src/c/extract_bit_32_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_UH1.c

- SCI/modules/scicos_blocks/src/c/extract_bit_u16_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u16_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u16_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_RB1.c

See also

- BITSET - BITSET Set a Bit
- BITCLEAR - BITCLEAR Clear a Bit

Authors

Fady NASSIF - INRIA

Name

INTMUL — Integer matrix multiplication

Block Screenshot



Contents

- Integer matrix multiplication
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Integer palette

Description

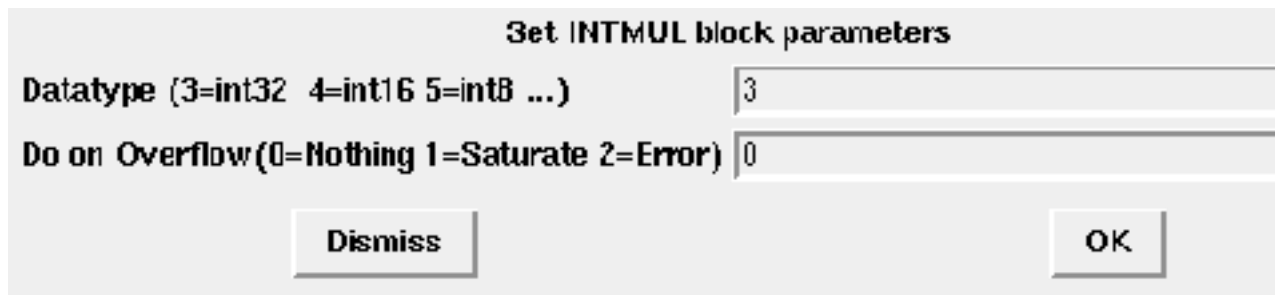
The INTMUL block computes the matrix multiplication of two integers inputs matrices. The number of rows of the second matrix must be equal to the number of columns of the first matrix. The output is a matrix where the number of rows is equal to the number of rows of the first input matrix and the number of columns is equal to the number of columns of the second input matrix. This block support all the integer data type.

On overflow, the result can take different forms:

- 1- A normal non saturated result.
- 2- A saturated result.
- 3- An error message warning the user about the overflow.

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

Dialog box



- **Datatype (3=int32 4=int16 5=int8 ...)**

It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

Properties : Type 'vec' of size 1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

When this parameter is set to zero the result is similar to a normal multiplication of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 3
 - port 2 : size [-2,-3] / type 3
- **regular outputs:**
 - port 1 : size [-1,-3] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matmul_i32*

Interfacing function

- `SCI/modules/scicos_blocks/macros/IntegerOp/INTMUL.sci`

Computational function

- `SCI/modules/scicos_blocks/src/c/matmul_i32n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i16n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i8n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui32n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui16n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui8n.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i32s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i16s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i8s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui32s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui16s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui8s.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i32e.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i16e.c`
- `SCI/modules/scicos_blocks/src/c/matmul_i8e.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui32e.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui16e.c`
- `SCI/modules/scicos_blocks/src/c/matmul_ui8e.c`

See also

- `MATMUL` - `MATMUL` Matrix Multiplication

Authors

Fady NASSIF - INRIA

Name

JKFLIPFLOP — JK flip-flop

Block Screenshot



Contents

- JK flip-flop
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

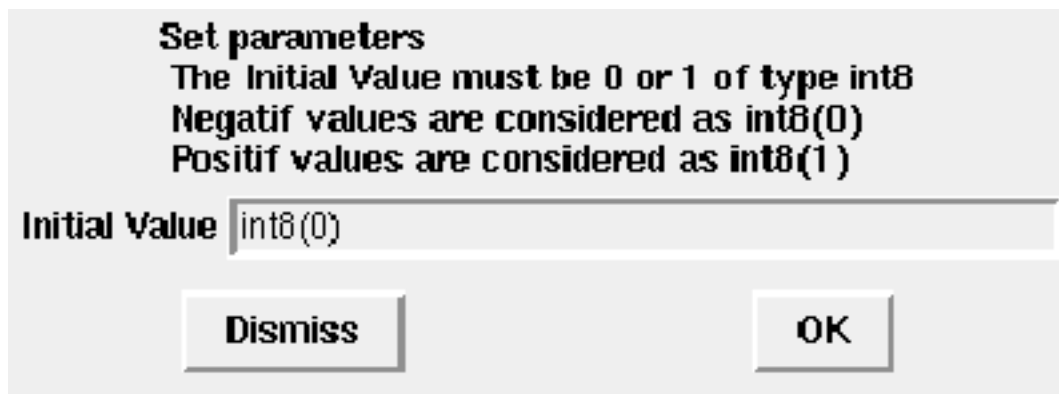
Description

The JK flip flop is the most versatile of the basic flip-flops. It has two inputs traditionally labeled J and K. When J and K are different, the output takes the value of J at the next falling edge. When J and K are both low, no change occurs in the output state, when they are both high the output will toggle from one state to other. It can perform the functions of the set/reset flip-flop and has the advantage that there are no ambiguous states. It can also act as a T flip-flop to accomplish toggling action if J and K are tied together. This toggle application finds extensive use in binary counters. This block is almost used with digital number, the input data type is int8.

The truth table of this block is

J	K	Q(t)	!Q(t)
0	0	Q(t-1)	!Q(t-1)
0	1	0	1
1	0	1	0
1	1	!Q(t-1)	Q(t-1)

Dialog box



- **Initial Value**

Initial Value of the state Q.

Properties : Type 'vec' of size 1.

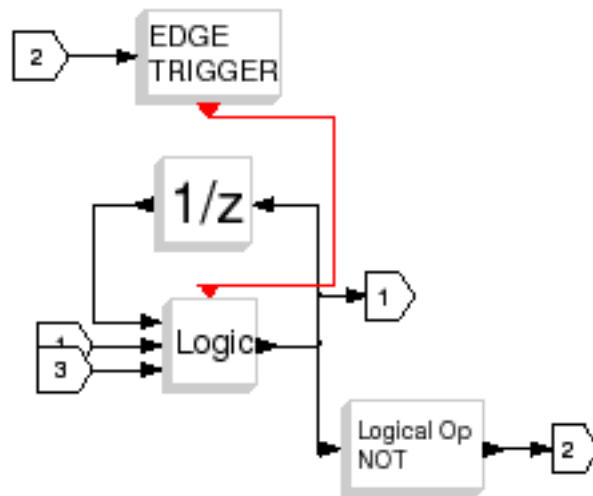
Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 1
 - port 3 : size [1,1] / type 5
- **regular outputs:**
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/JKFLIPFLOP.sci

Compiled Super Block content



See also

- DLATCH - D latch flip-flop
- DFLIPFLOP - D flip-flop
- SRFLIPFLOP - SR flip-flop

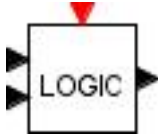
Authors

Fady NASSIF - INRIA

Name

LOGIC — Combinational Logic

Block Screenshot



Contents

- Combinational Logic
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

This block implements a standard truth table for modeling programming array, digital circuit and any other boolean expressions. The user can specify a matrix that defines all the possible block output in the Truth table field. Each row of the matrix contains the output of different combination of input elements. The number of rows must be a power of two, it defines the number of inputs using the equation:

number of row = 2 (number of input)

The number of outputs is equal to the number of columns of the matrix.

This block support only the int8 data type. When the input is positive, the input is considered as logical 1, When it is negative or zero it is considered as logical 0.

This block can be activated by an implicit input event or it can inherit the clock from the regular input.

This block is used to implement SR and JK flip-flops.

Dialog box

Set Logic parameters

Truth table

Inherit(0=no 1=yes)

- **Truth table**

The matrix of outputs. For more information see the description part.

Properties : Type 'mat' of size [-1,-2].

- **Inherit(0=no 1=yes)**

Specifies if the clock is inherit or not.

Properties : Type 'vec' of size 1.

Example

The easiest example to consider is the OR example. In this case we have two inputs and one output.

input 1	input 2	output
0	0	0
0	1	1
1	0	1
1	1	1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5

- **regular outputs:**
 - port 1 : size [1,1] / type 5
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logic*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/LOGIC.sci

Computational function

- SCI/modules/scicos_blocks/src/c/logic.c

See also

- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

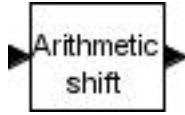
Authors

Fady NASSIF - INRIA

Name

SHIFT — Shift Bits

Block Screenshot



Contents

- Shift Bits
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

Description

This block shifts the bits of the input signal. In this operation the digits are moved to the right or to the left. The user can choose the rule to shifts the bits. It can be normal or cycle by setting the "Shifttype" parameter to "0" or "1".

When the Shifttype is 0, an arithmetic shift is applied to the input signal. In this case, the bits that are shifted out of either end are discarded. Zeros are shifted in on the right, in the case of left shift; in the case of right shifts, copies of the sign bit is shifted in on the left.

When the "Shifttype" is 1, a circular shift is applied to the input signal. In this case, the bits are rotated as if the left and right ends of the register are joined. The value that is shifted in on the right during a left-shift is whatever values was shifted out on the left, and vice versa.

Dialog box

Set Shift block parameters

Datatype (3=int32 4=int16 5=int8 ...)

Number of bits to shift left (use negative number to shift right)

Shifttype(0=Arithmetic 1=Circular)

- **Datatype (3=int32 4=int16 5=int8 ...)**

It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

Properties : Type 'vec' of size 1.

- **Number of bits to shift left (use negative number to shift right)**

It indicates the number of bits the input signal is shifted. A positive value indicates a shift left, negative values indicates shift right.

Properties : Type 'vec' of size 1.

- **Shifttype(0=Arithmetic 1=Circular)**

It indicate the rule used to shift the bits. It can be arithmetic or circular. When the Shifttype is normal, an arithmetic shift is applied to the input signal. In this case, the bits that are shifted.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 3
- **regular outputs:**
 - port 1 : size [-1,-2] / type 3
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *shift_32_LA*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/SHIFT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/shift_32_LA.c
- SCI/modules/scicos_blocks/src/c/shift_32_LC.c
- SCI/modules/scicos_blocks/src/c/shift_32_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u32_RA.c
- SCI/modules/scicos_blocks/src/c/shift_32_RC.c
- SCI/modules/scicos_blocks/src/c/shift_16_LA.c
- SCI/modules/scicos_blocks/src/c/shift_16_LC.c
- SCI/modules/scicos_blocks/src/c/shift_16_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u16_RA.c
- SCI/modules/scicos_blocks/src/c/shift_16_RC.c
- SCI/modules/scicos_blocks/src/c/shift_8_LA.c
- SCI/modules/scicos_blocks/src/c/shift_8_LC.c
- SCI/modules/scicos_blocks/src/c/shift_8_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u8_RA.c
- SCI/modules/scicos_blocks/src/c/shift_8_RC.c

See also

- BITSET - BITSET Set a Bit
- BITCLEAR - BITCLEAR Clear a Bit

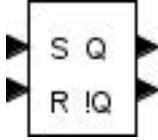
Authors

Fady NASSIF - INRIA

Name

SRFLIPFLOP — SR flip-flop

Block Screenshot



Contents

- SR flip-flop
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

- Integer palette

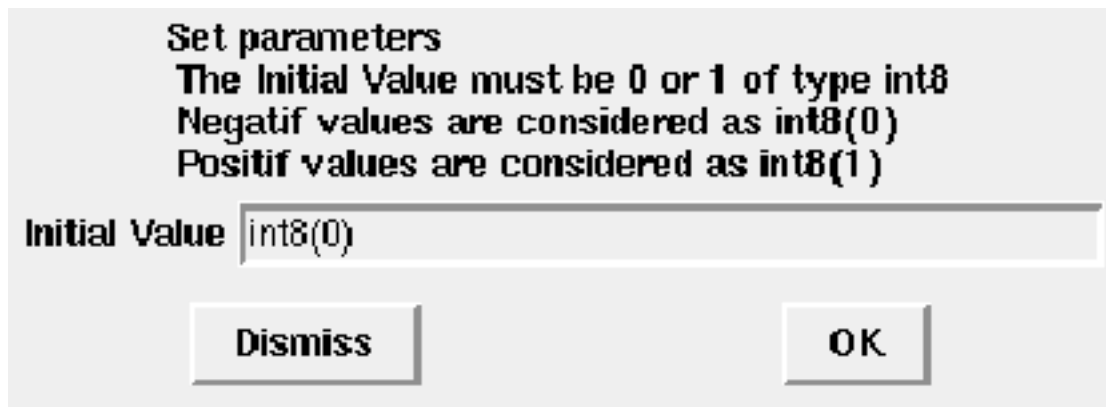
Description

This block describe the simplest and the most fundamental latch the SR flip flop. Where S and R are the input and Q and !Q are the outputs.If S (Set) is pulsed high while R is held low, then the Q output is forced high, and stays high when S returns low; similarly, if R (Reset) is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns low. When both are low, Q(t) takes the same state as Q(t-1). When they are both high, both Q and !Q take the low values we are in an unstable state. Practically we have to avoid this case.This block is almost used with digital number, the input data type is int8.

The truth table of this block is

S	R	Q(t)	!Q(t)
0	0	Q(t-1)	!Q(t-1)
0	1	0	1
1	0	1	0
1	1	0	0
-> This case is to avoid			

Dialog box



- **Initial Value**

Initial Value of the state Q.

Properties : Type 'vec' of size 1.

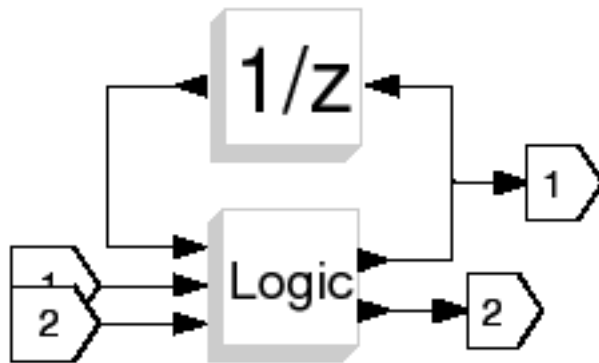
Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5
- **regular outputs:**
 - port 1 : size [1,1] / type 5
 - port 2 : size [1,1] / type 5
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/SRFLIPFLOP.sci

Compiled Super Block content



See also

- DLATCH - D latch flip-flop
- DFLIPFLOP - D flip-flop
- JKFLIPFLOP - JK flip-flop

Authors

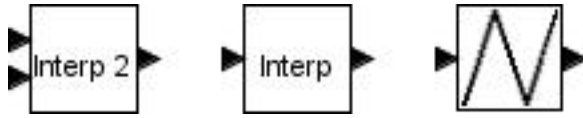
Fady NASSIF - INRIA

11. Lookup tables palette

Name

Lookuptables_pal — Lookup tables palette

Block Screenshot



Module

- xcOS

Description

The lookup tables palette includes blocks that compute output approximations from inputs.

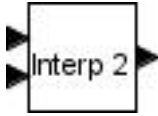
Blocks

- INTRP2BLK_f — 2D interpolation
- INTRPLBLK_f — Interpolation
- LOOKUP_f — Lookup table

Name

INTRP2BLK_f — 2D interpolation

Block Screenshot



Contents

- 2D interpolation
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

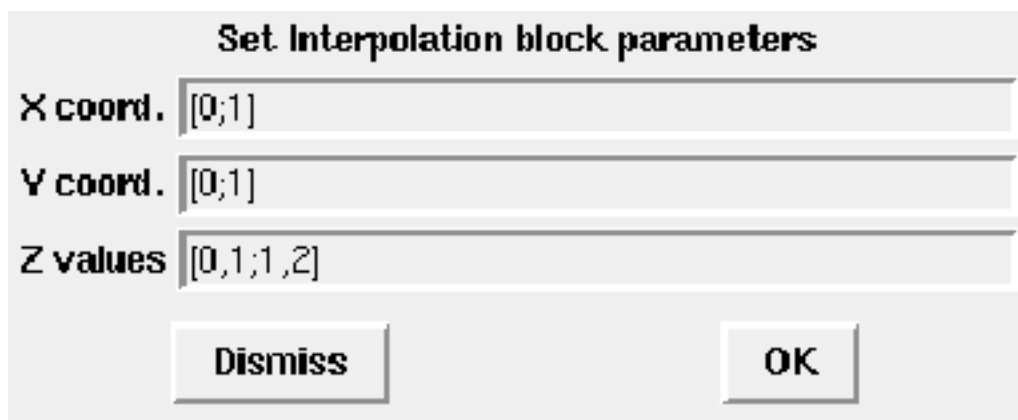
Palette

- Lookup tables palette

Description

The output of this block is a function of the inputs obtained by bilinear interpolation. This block has two scalar inputs and a single scalar output. The and give respectively the coordinate and the coordinate of the -th data point to be interpolated and its value.

Dialog box



Set Interpolation block parameters

X coord. [0;1]

Y coord. [0;1]

Z values [0,1;1,2]

Dismiss OK

- **X coord.**
an n-vector (strictly increasing).

Properties : Type 'vec' of size -1

- **Y coord.**

an m-vector (strictly increasing).

Properties : Type 'vec' of size -1

- **Z values**

$m \times n$

an matrix.

Properties : Type 'mat' of size [-1,-1]

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *intrp2*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INTRP2BLK_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/intrp2.f (Type 1)

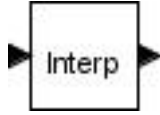
Authors

Ramine Nikoukhah - INRIA

Name

INTRPLBLK_f — Interpolation

Block Screenshot



Contents

- Interpolation
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

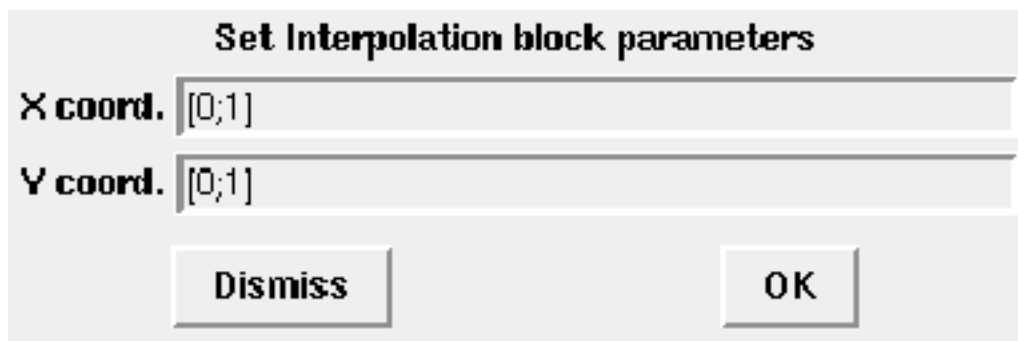
Palette

- Lookup tables palette

Description

The output of this block is a function of the input obtained by linear interpolation. This block has a single scalar input and a single scalar output port. The coord. and coord. give respectively the coordinate and the coordinate of the data points to be interpolated. coord must be strictly increasing.

Dialog box



- **X coord.**
A vector (strictly increasing).
Properties : Type 'vec' of size -1

- **Y coord.**

A vector (same size as X coord).

Properties : Type 'vec' of size -1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *intrpl*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INTRPLBLK_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/intrpl.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

LOOKUP_f — Lookup table

Block Screenshot



Contents

- Lookup table
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Lookup tables palette

Description

This block realizes a non-linear function defined using a graphical editor.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *lookup*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/LOOKUP_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/lookup.f (Type 0)

Authors

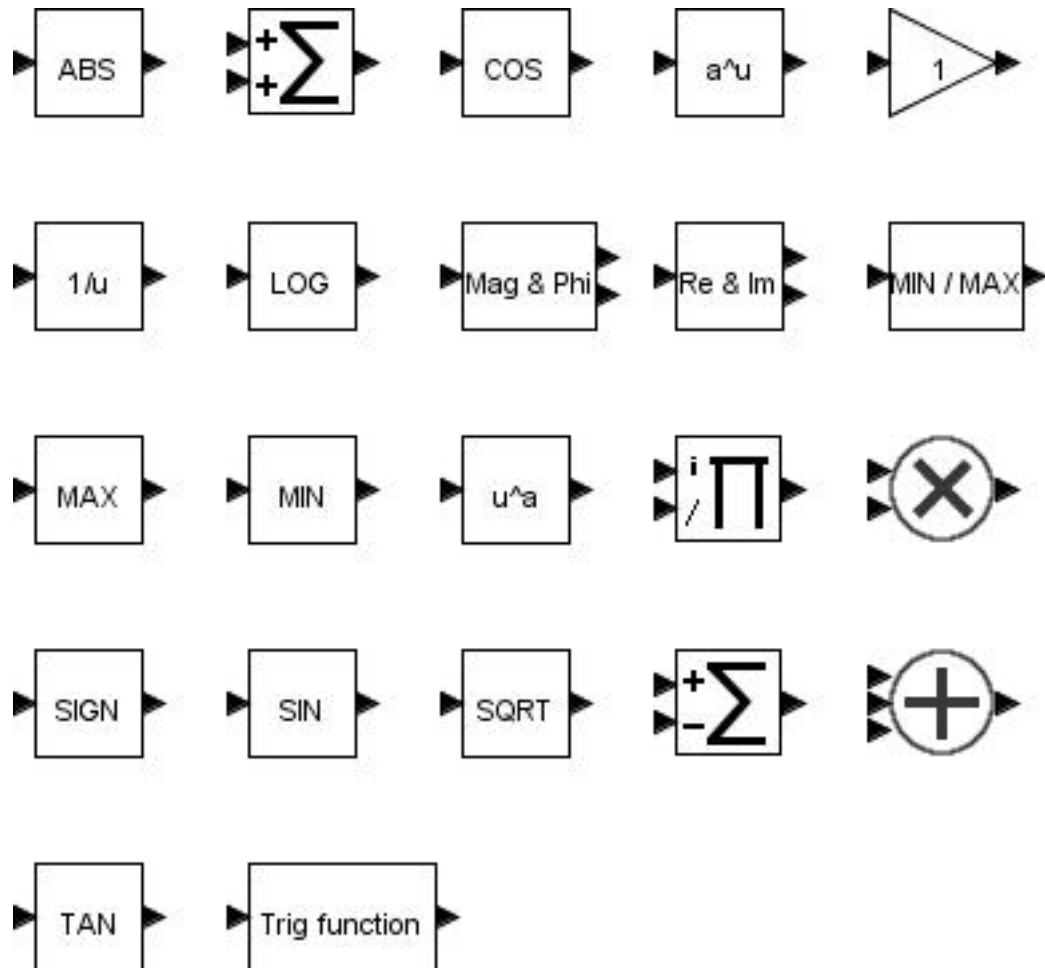
Ramine Nikoukhah - INRIA

12. Math operations palette

Name

Mathoperations_pal — Math operations palette

Block Screenshot



Module

- xcos

Description

The Math operations palette contains blocks that model general mathematical functions.

Blocks

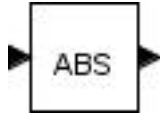
- ABS_VALUE — Absolute value
- BIGSOM_f — Sum
- COSBLK_f — Cosinus Block
- EXPBLK_m — Exponential
- GAINBLK_f — Gain
- INVBLK — Inverse

- LOGBLK_f — Common logarithm
- MATMAGPHI — Complex to Magnitude and Angle Conversion
- MATZREIM — Complex decomposition
- MAX_f — MAX
- MAXMIN — Max and Min
- MIN_f — MIN
- POWBLK_f — Array power
- PROD_f — Multiplication
- PRODUCT — Product
- SIGNUM — Signum
- SINBLK_f — Sinus block
- SQRT — Square root
- SUM_f — Addition
- SUMMATION — Summation
- TANBLK_f — Tangent block
- TrigFun — Trigonometric function

Name

ABS_VALUE — Absolute value

Block Screenshot



Contents

- Absolute value
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

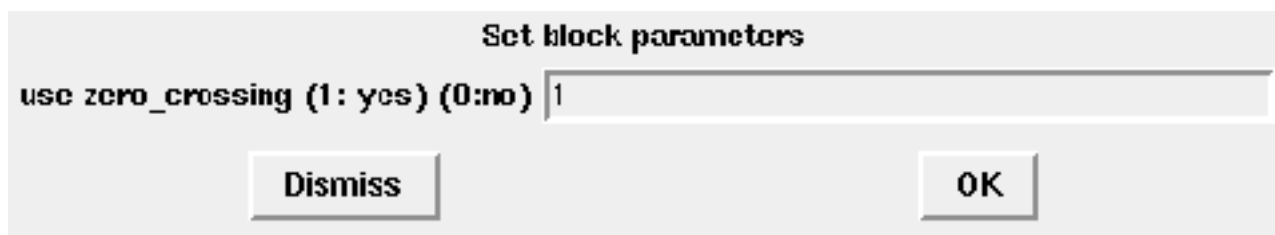
Palette

- Math operations palette

Description

The Abs block outputs the absolute value of the input.

Dialog box



- **use zero_crossing**
Select to enable zero crossing detection.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *absolute_value*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/ABS_VALUE.sci

Computational function

- SCI/modules/scicos_blocks/src/c/absolute_value.c (Type 4)

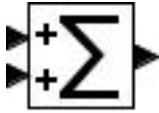
Authors

Ramine Nikoukhah - INRIA

Name

BIGSOM_f — Sum

Block Screenshot



Contents

- Sum
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

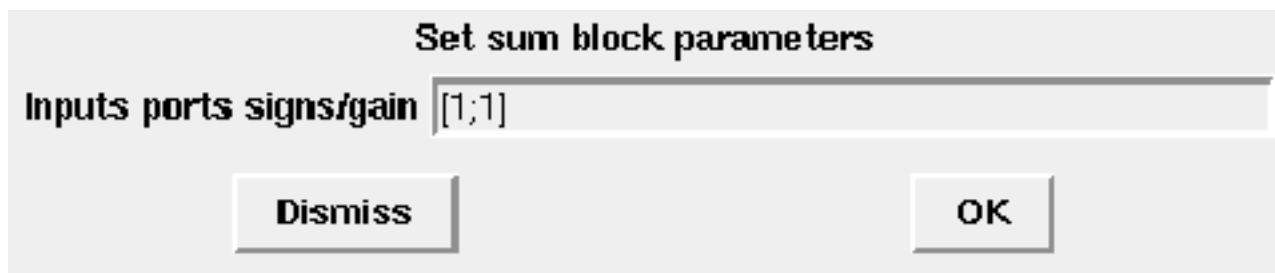
Palette

- Math operations palette

Description

The Sum block performs addition on its inputs. This block can add scalar or vector inputs.

Dialog box



- **Inputs ports signs/gain**
Set sign and a gain for each inputs.
Properties : Type 'vec' of size -1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sum*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/BIGSOM_f.sci

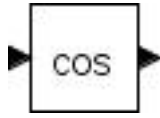
Computational function

- SCI/modules/scicos_blocks/src/c/sum.c (Type 2)

Name

COSBLK_f — COSBLK

Block Screenshot



Contents

- COSBLK
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”

Palette

- Math operations palette

Description

Description

$$y = \cos(u)$$

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *cosblk*

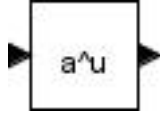
Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/COSBLK_f.sci

Name

EXPBLK_m — Exponential

Block Screenshot



Contents

- Exponential
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

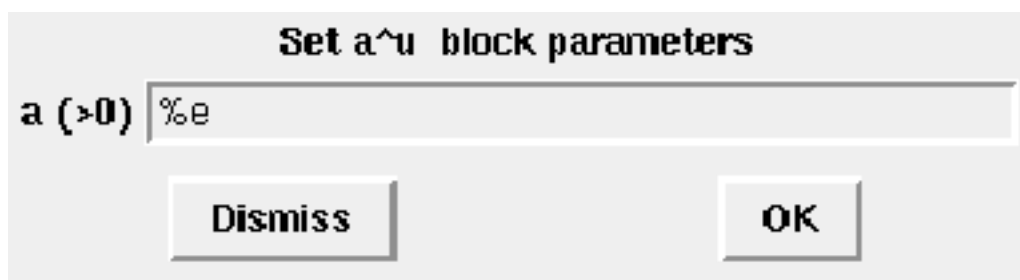
Palette

- Math operations palette

Description

This block realizes a^u . The input and output port sizes are determined by the compiler.

Dialog box



- **a**
A real positive scalar.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *expblk_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/EXPBLK_m.sci

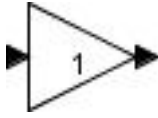
Computational function

- SCI/modules/scicos_blocks/src/c/expblk_m.c (Type 4)

Name

GAINBLK_f — Gain

Block Screenshot



Contents

- Gain
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

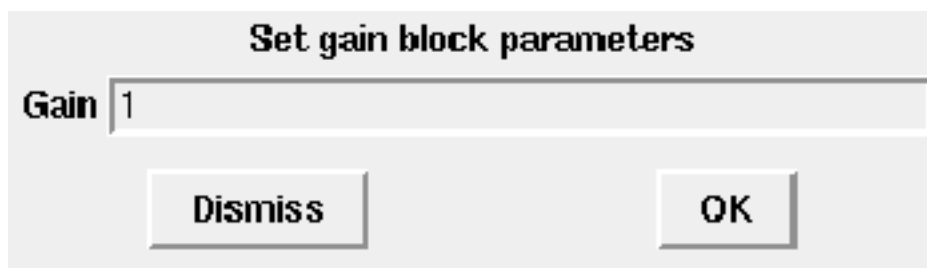
Palette

- Math operations palette

Description

The GAINBLK computes the product of a square matrix A by the input matrix U , where the number of rows/cols of A is equal to the number of rows of U .

Dialog box



- **Gain**
This parameter defined the square matrix A .
Properties : Type 'mat' of size [-1,-1].

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gain*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/GAINBLK_f.sci

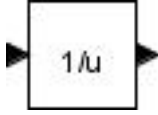
Authors

Ramine Nikoukhah - INRIA

Name

INVBLK — Inverse

Block Screenshot



Contents

- Inverse
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Math operations palette

Description

This block computes $\frac{1}{u}$. The input (output) size is determined by the context.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *invblk4*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INVBLK.sci

Computational function

- SCI/modules/scicos_blocks/src/c/invblk4.c (Type 4)

Name

LOGBLK_f — Log

Block Screenshot



Contents

- Log
- the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Math operations palette

Description

This block realizes . The input and output port sizes are determined by the context.

Dialog box



- **Basis**
A real scalar greater than 1.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/LOGBLK_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/logblk.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

MATMAGPHI — Complex to Magnitude and Angle Conversion

Block Screenshot



Contents

- Complex to Magnitude and Angle Conversion
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Math operations palette

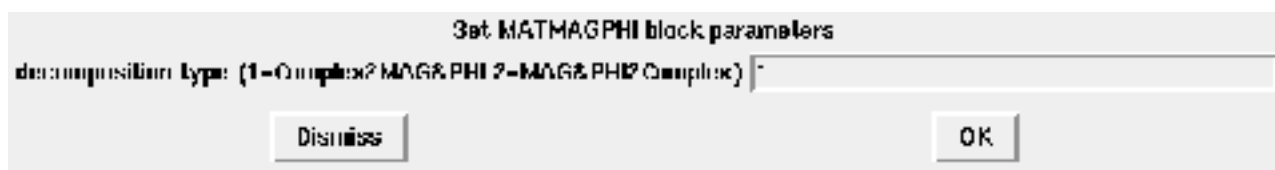
Description

MATMAGPHI Block has two types of decomposotions.

When the type is set to one, the block converts a complex number to the magnitude and the radian angle, in this case the input is complex and the outputs are real double. If the input is real double, the angle will be zero or PI and the magnitude will be equal to the absolute of the input number.

When the type is set to two, the block outputs a complex number given the magnitude and the radian angle. In this case the inputs are real double and the output is complex.

Dialog box



- **decomposition type (1=Complex2MAG&PHI 2=MAG&PHI2Complex)**

It indicates the rule of the conversion.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [-1,-2] / type 2
- **regular outputs:**
 - **port 1 :** size [-1,-2] / type 1
 - **port 2 :** size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_abs*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATMAGPHI.sci

Computational function

- SCI/modules/scicos_blocks/src/c/matz_abs.c
- SCI/modules/scicos_blocks/src/c/matz_absc.c

See also

- MATZREIM - Complex decomposition

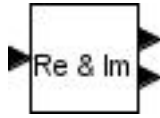
Authors

Fady NASSIF - INRIA

Name

MATZREIM — Complex decomposition

Block Screenshot



Contents

- Complex decomposition
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Math operations palette

Description

This block decomposes a complex number by separating the real and imaginary parts or compose a complex number by joining the two parts. The user can select even to separate or to join real and imaginary part by setting the decomposition type to 1 or 2. When it is set to 1, the input is a complex matrix and the outputs are the real and imaginary parts of the input. When it set to 2, The inputs are two real matrices, the output is a complex number with real part the first input and imaginary part the second input.

Dialog box



- **decomposition type (1=Complex2Real&Imag 2=Real&Imag2Complex)**

Indicates the type to use for the decomposition. See the description part for more information.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 2
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_reim*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATZREIM.sci

Computational function

- SCI/modules/scicos_blocks/src/c/matz_reim.c
- SCI/modules/scicos_blocks/src/c/matz_reimc.c

See also

- MATMAGPHI - MATMAGPHI Complex to Magnitude and Angle Conversion

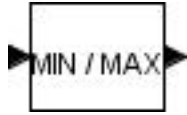
Authors

Fady NASSIF - INRIA

Name

MAXMIN — Max and Min

Block Screenshot



Contents

- Max and Min
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Math operations palette

Description

The MinMax block outputs either the minimum or the maximum element or elements of the inputs. You can choose the function to apply by selecting one of the choices from the Function parameter list.

Dialog box

Set Max/Min block parameters	
Min (1) or Max (2)	2
Number of input vectors (1 or 2)	1
zero-crossing (1: yes, 0;no)	1
DismissOK	

- **Min or Max**

The function (min or max) to apply to the input.

Properties : Type 'vec' of size 1.
- **Number of input vectors**

The number of inputs to the block.

Properties : Type 'vec' of size 1.

- **zero-crossing**

Select to enable zero crossing detection to detect minimum and maximum values.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *minmax*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MAXMIN.sci

Computational function

- SCI/modules/scicos_blocks/src/c/minmax.c (Type 4)

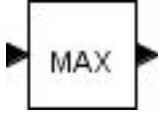
Authors

Ramine Nikoukhah - INRIA

Name

MAX_f — MAX

Block Screenshot



Contents

- MAX
- the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Math operations palette

Description

That block find the max value in the elements of its input vector.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *maxblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MAX_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/maxblk.f (Type 0)

Name

MIN_f — MIN

Block Screenshot



Contents

- MIN
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

Palette

- Math operations palette

Description

That block find the min value in the elements of its input vector.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *minblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MIN_f.sci

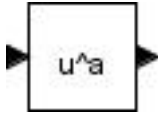
Computational function

- SCI/modules/scicos_blocks/src/fortran/minblk.f (Type 0)

Name

POWBLK_f — Power

Block Screenshot



Contents

- Power
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

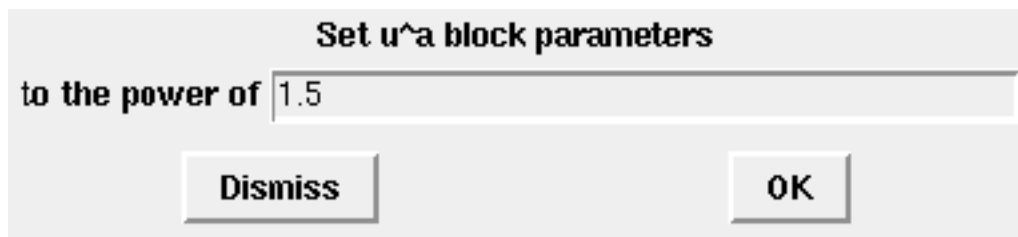
Palette

- Math operations palette

Description

This block realizes u^a . The input and output port sizes are determined by the compiler according to the connected blocks port sizes.

Dialog box



- **to the power of**
real scalar.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *powblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/POWBLK_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/powblk.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

PRODUCT — Product

Block Screenshot



Contents

- Product
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

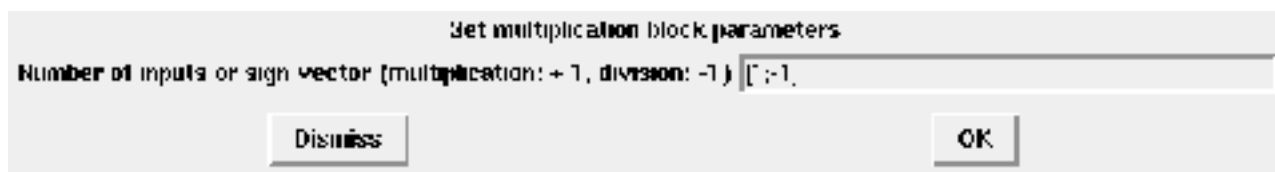
Palette

- Math operations palette

Description

The Product block performs multiplication or division of its inputs. This block produces outputs using either element-wise or matrix multiplication, depending on the value of the Multiplication parameter. You specify the operations with the Number of inputs parameter. Multiply(+1) and divide (-1) characters indicate the operations to be performed on the inputs.

Dialog box



- **Number of inputs or sign vector**

Enter the number of inputs.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *product*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/PRODUCT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/product.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

PROD_f — Multiplication

Block Screenshot



Contents

- Multiplication
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Math operations palette

Description

The output is the element wise product of the inputs.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *prod*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/PROD_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/prod.c (Type 2)

Authors

Ramine Nikoukhah - INRIA

Name

SIGNUM — Signum

Block Screenshot



Contents

- Signum
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

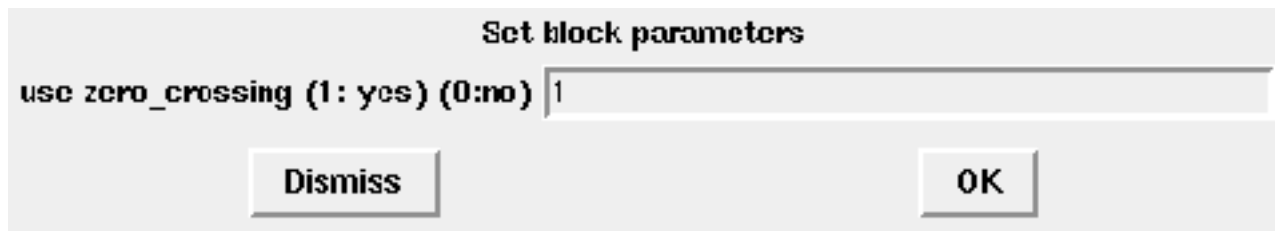
- Math operations palette

Description

The Sign block indicates the sign of the input:

- The output is 1 when the input is greater than zero.
- The output is 0 when the input is equal to zero.
- The output is -1 when the input is less than zero.

Dialog box



- **use zero_crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *signum*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SIGNUM.sci

Computational function

- SCI/modules/scicos_blocks/src/c/signum.c (Type 4)

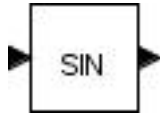
Authors

Ramine Nikoukhah - INRIA

Name

SINBLK_f — SINBLK

Block Screenshot



Contents

- SINBLK
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Math operations palette

Description

Description

$$y = \sin(u)$$

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sinblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SINBLK_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/sinblk.f (Type 0)

Name

SQRT — Square root

Block Screenshot



Contents

- Square root
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Math operations palette

Description

This block computes the square root of each element of the input matrix. It supported real and complex data types.

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sqrt*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/SQRT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_sqrt.c
- SCI/modules/scicos_blocks/src/c/matz_sqrt.c

See also

- POWBLK_f - Pow

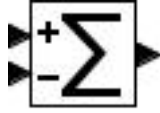
Authors

Fady NASSIF - INRIA

Name

SUMMATION — Matrix Summation

Block Screenshot



Contents

- Matrix Summation
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Math operations palette

Description

The Sum block performs addition or subtraction on its inputs. This block can add or subtract scalar, vector, or matrix inputs. It can also collapse the elements of a single input vector.

The number of inputs is given by the second parameter. This parameter can be a vector of +1 and -1 or it can be a positive value. In the first case the size of the vector indicates the number of inputs and the signs indicates whether it is a summation or a subtraction. In the second case, the block is a summation block and the value indicates the number of inputs.

On overflow, the result can take different forms:

- 1- A normal non saturated result.
- 2- A saturated result.
- 3- An error message warning the user about the overflow.

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

Dialog box

Set sum block parameters

Datatype (1=real double 2=complex 3=int32 ...)

Number of inputs or sign vector (of +1, -1)

Do on Overflow(0=Nothing 1=Saturate 2=Error)

- **Datatype (1=real double 2=complex 3=int32 ...)**

It indicates the type of the input/output data. It support all datatype, number must be between 1 and 8.

Properties : Type 'vec' of size 1.

- **Number of inputs or sign vector (of +1, -1)**

It indicates the number of inputs and the operation see the description for more detail.

Properties : Type 'vec' of size -1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

When this parameter is set to zero the result is similar to a normal summation of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears. If the Data type is double or complex this parameter is not used.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *summation*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SUMMATION.sci

Computational function

- SCI/modules/scicos_blocks/src/c/summation.c
- SCI/modules/scicos_blocks/src/c/summation_z.c
- SCI/modules/scicos_blocks/src/c/summation_i32n.c
- SCI/modules/scicos_blocks/src/c/summation_i16n.c
- SCI/modules/scicos_blocks/src/c/summation_i8n.c
- SCI/modules/scicos_blocks/src/c/summation_ui32n.c
- SCI/modules/scicos_blocks/src/c/summation_ui16n.c
- SCI/modules/scicos_blocks/src/c/summation_ui8n.c
- SCI/modules/scicos_blocks/src/c/summation_i32s.c
- SCI/modules/scicos_blocks/src/c/summation_i16s.c
- SCI/modules/scicos_blocks/src/c/summation_i8s.c
- SCI/modules/scicos_blocks/src/c/summation_ui32s.c
- SCI/modules/scicos_blocks/src/c/summation_ui16s.c
- SCI/modules/scicos_blocks/src/c/summation_ui8s.c
- SCI/modules/scicos_blocks/src/c/summation_i32e.c
- SCI/modules/scicos_blocks/src/c/summation_i16e.c
- SCI/modules/scicos_blocks/src/c/summation_i8e.c
- SCI/modules/scicos_blocks/src/c/summation_ui32e.c
- SCI/modules/scicos_blocks/src/c/summation_ui16e.c
- SCI/modules/scicos_blocks/src/c/summation_ui8e.c

Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

Name

SUM_f — Addition

Block Screenshot



Contents

- Addition
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Math operations palette

Description

The Sum block performs addition on its inputs. This block can add scalar, vector, or matrix inputs.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 0
 - port 3 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *plusblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SUM_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/plusblk.c (Type 2)

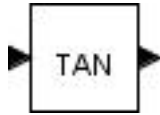
Authors

Ramine Nikoukhah - INRIA

Name

TANBLK_f — TANBLK

Block Screenshot



Contents

- TANBLK
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Math operations palette

Description

Description

$$y = \tan(u)$$

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tanblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/TANBLK_f.sci

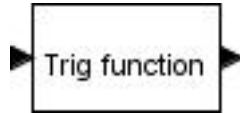
Computational function

- SCI/modules/scicos_blocks/src/fortran/tanblk.f (Type 0)

Name

TrigFun — Trigonometric function

Block Screenshot



Contents

- Trigonometric function
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

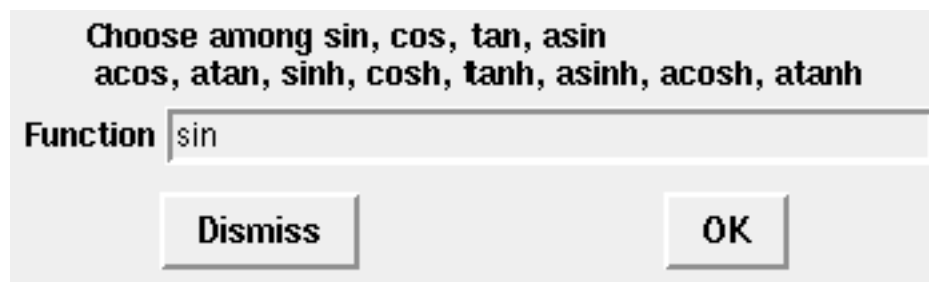
Palette

- Math operations palette

Description

The Trigonometric Function block performs numerous common trigonometric functions. You can select one of these functions from the Function list: sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, and tanh. The block output is the result of the operation of the function on the input or inputs.

Dialog box



- **Function**

The trigonometric function.

Properties : Type 'str' of size 1

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sin_blk*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/TrigFun.sci

Computational function

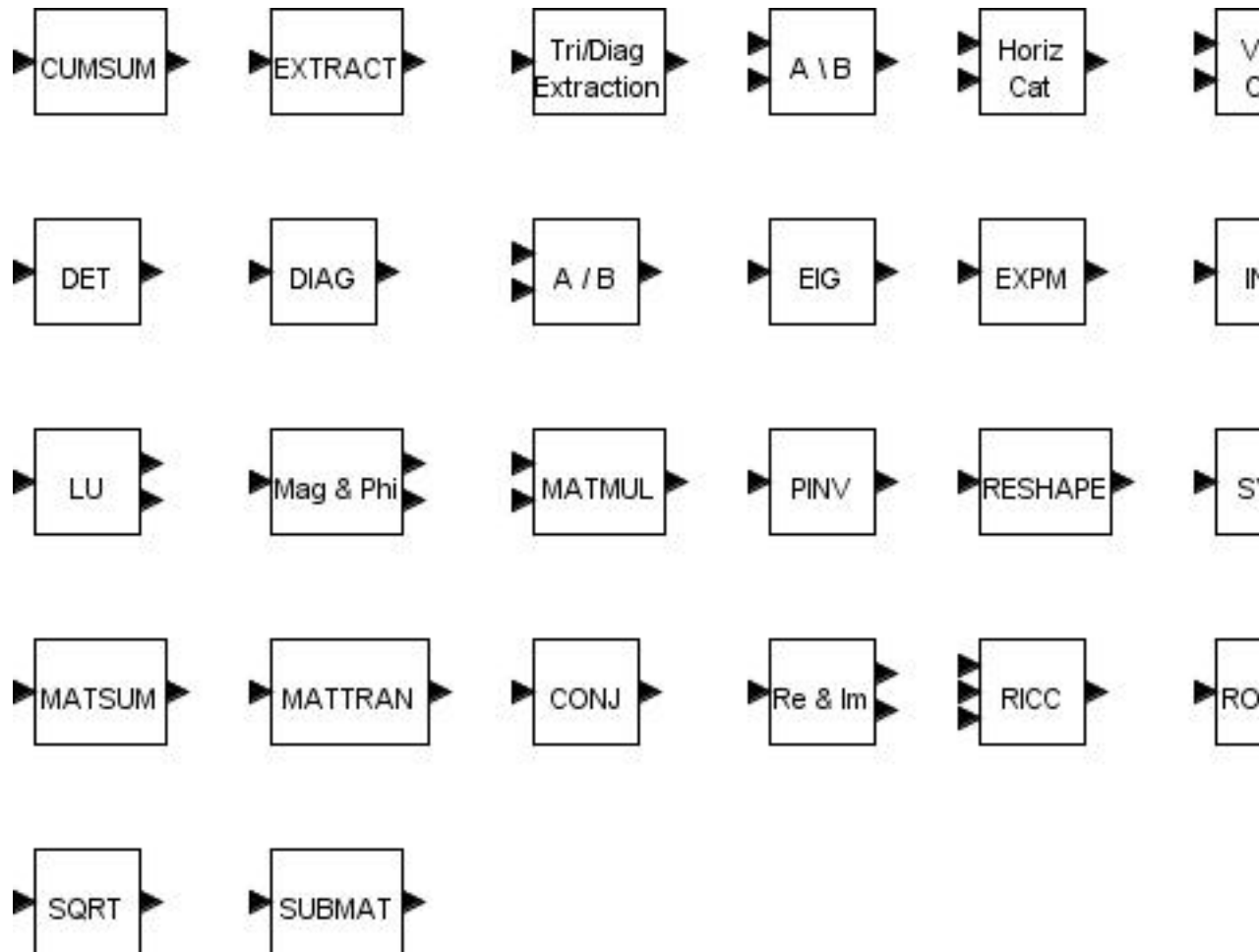
- SCI/modules/scicos_blocks/src/c/sin_blk.c (Type 4)

13. Matrix operation palette

Name

Matrix_pal — Matrix operation palette

Block Screenshot



Module

- xcOS

Description

Matrix palette contains all blocks that you need to do simple and complex matrix operations.

Blocks

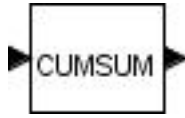
- CUMSUM - CUMSUM: Cumulative Sum
- EXTRACT - EXTRACT: Matrix Extractor
- EXTTRI - EXTTRI: Triangular or Diagonal extraction
- MATBKSL - MATBKSL:left matrix division
- MATCATH - MATCATH: Horizontal Concatenation
- MATCATV - MATCATV Vertical Concatenation

- MATDET - MATDET Matrix Determinant
- MATDIAG - MATDIAG Create Diagonal Matrix
- MATDIV - MATDIV Matrix division
- MATEIG - MATEIG Matrix Eigenvalues
- MATEXPM - MATEXPM Matrix Exponential
- MATINV - MATINV Matrix Inverse
- MATLU - MATLU LU Factorization
- MATMAGPHI - MATMAGPHI Complex to Magnitude and Angle Conversion
- MATMUL - MATMUL Matrix Multiplication
- MATPINV - MATPINV Matrix PseudoInverse
- MATRESH - MATRESH Matrix Reshape
- MATSING - MATSING SVD decomposition
- MATSUM - Matrix Sum
- MATTRAN - Matrix Transpose
- MATZCONJ - Matrix Conjugate
- MATZREIM - Complex decomposition
- RICC - RICC Equation de Riccati
- ROOTCOEF - Coefficient computation
- SQRT - SQRT Square root
- SUBMAT - SUBMAT Sub-matrix extraction

Name

CUMSUM — Cumulative Sum

Block Screenshot



Contents

- Cumulative Sum
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

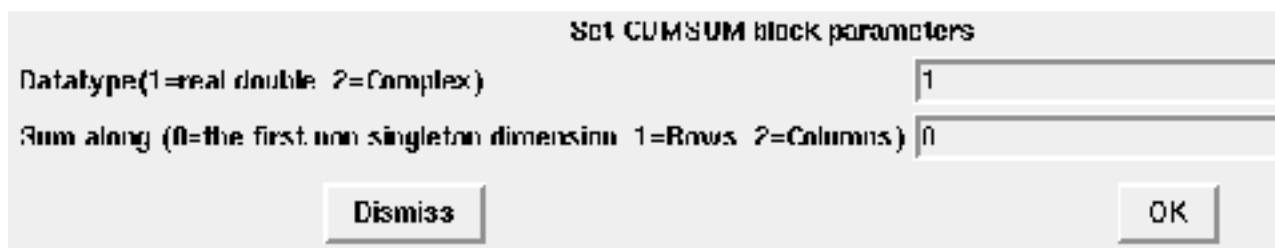
Description

The CUMSUM block sums the element of an input $m \times n$ matrix u along either the rows, the columns or the first non singleton dimension. When the "Sum along" parameter is set to "1", the block sums across the elements of each row. The result will be displayed as a $m \times 1$ matrix.

When the "Sum along" parameter is set to "2", the block sums across the elements of each column. The result will be display as a $1 \times n$ matrix.

When the "Sum along" parameter is set to "0", the block sums across the first non singleton dimension. The result will be displayed as one element. This block is equivalent to `cumsum` in scilab.

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Sum along (0=the first non singleton dimension 1=Rows 2=Columns)**

Indicate whether to sum across the rows, the columns or the first non singleton dimension.

Properties : Type 'vec' of size 1.

Example

```
A=[1 2 3;4 5 6;7 8 9]
If the sum is along the row the result will be
B=[12;15;18]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cumsum_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/CUMSUM.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cumsum_m.c
- SCI/modules/scicos_blocks/src/c/cumsum_r.c

- `SCI/modules/scicos_blocks/src/c/cumsum_c.c`
- `SCI/modules/scicos_blocks/src/c/cumsumz_m.c`
- `SCI/modules/scicos_blocks/src/c/cumsumz_r.c`
- `SCI/modules/scicos_blocks/src/c/cumsumz_c.c`

See also

- MATSUM - Matrix Sum (xcos Block)

Authors

Fady NASSIF - INRIA

Name

EXTRACT — Matrix Extractor

Block Screenshot



Contents

- Matrix Extractor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - Example
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Matrix operation palette

Description

The EXTRACT block extracts some elements from the matrix. The size of the output depends on the number of rows and number of columns to extract.

Dialog box

The dialog box is titled "Set EXTRACT Block". It contains three input fields with the following labels and values:

Label	Value
Datatype (1=real double 2=Complex)	1
Lines to extract	1
Columns to extract	1

At the bottom of the dialog box, there are two buttons: "Dismiss" and "OK".

- Datatype (1=real double 2=Complex)

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Lines to extract**

It indicates the numbers of the lines to extract.

Properties : Type 'mat' of size [1,-1].

- **Columns to extract**

It indicates the numbers of the columns to extract.

Properties : Type 'mat' of size [1,-1].

Example

```
A=[1 2 3;4 5 6;7 8 9]
```

If the "Lines to extract" is 1 and 2 and the "Column to extract" is 1 and 3 the

```
B=[1 3;4 6]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extract*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/EXTRACT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/extract.c

- `SCI/modules/scicos_blocks/src/c/extractz.c`

See also

- EXTTRI - EXTTRI: Triangular or Diagonal extraction (xcos Block)

Authors

Fady NASSIF - INRIA

Name

EXTTRI — Triangular or Diagonal extraction

Block Screenshot



Contents

- Triangular or Diagonal extraction
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The EXTTRI block extracts some elements from the input matrix u .

When the "Extraction type" is set to "1", the block copies the elements on and above the main diagonal to an output matrix of the same size. The elements below the main diagonal are set to zero.

When the "Extraction type" is set to "2", the block copies the elements on and below the main diagonal to an output matrix of the same size. The elements above the main diagonal are set to zero.

When the "Extraction type" is set to "3", the block copies the elements on the main diagonal to an output matrix of the same size. The elements above and below the main diagonal are set to zero.

Dialog box

Set EXTTRI block parameters

Datatype(1=real double 2=Complex)

extraction type (1=lower 2=upper 3=diagonal)

- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **extraction type (1=lower 2=upper 3=diagonal)**

It indicates the form of the output matrix. It can be an upper triangle, a lower triangle or a diagonal matrix.

Properties : Type 'vec' of size 1.

Example

```
A=[1 2 3;4 5 6;7 8 9;10 11 12]
If the extraction type is 2 then the output is
B=[1 0 0;4 5 0;7 8 9;10 11 12]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *extrilz*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/EXTTRI.sci

Computational function

- SCI/modules/scicos_blocks/src/c/extril.c
- SCI/modules/scicos_blocks/src/c/extriu.c
- SCI/modules/scicos_blocks/src/c/extdiag.c
- SCI/modules/scicos_blocks/src/c/extrilz.c
- SCI/modules/scicos_blocks/src/c/extriuiz.c
- SCI/modules/scicos_blocks/src/c/extdiagz.c

See also

- EXTRACT - EXTRACT: Matrix Extractor (xcos Block)

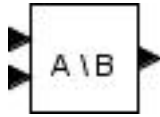
Authors

Fady NASSIF - INRIA

Name

MATBKSL — left matrix division

Block Screenshot



Contents

- left matrix division
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

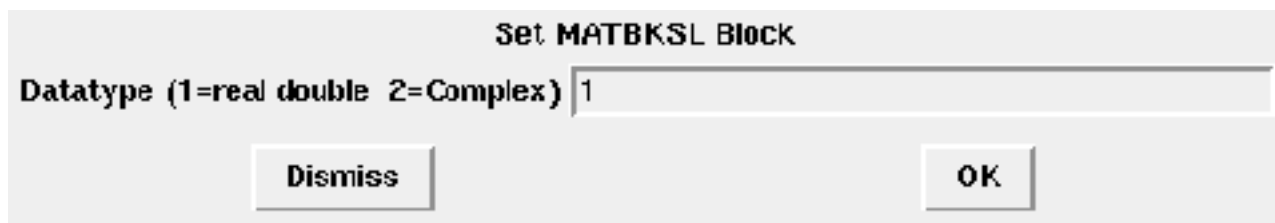
- Matrix operation palette

Description

The MATBKSL block outputs the left matrix division. It is a solution to $A*x=B$. The higher input is the A matrix, the lower one is the B matrix, and the output is x. If A is an M-by-N1 matrix, B must be a M-by-N2 where N1 and N2 can be different or equal. The output x is a N1-by-N2 matrix.

The equivalent of BACKSLASH is "in Scilab.

Dialog box



- **Datatype (1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example

```
A=[1 7 3;23 32 29]
B=[21 18;13 10]
then the result of the A*x=B equation is (A\B):
x=[-4.504 -3.922;3.643 3.132;0.000 0.000]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [-1,-2] / type 1
 - **port 2 :** size [-1,-3] / type 1
- **regular outputs:**
 - **port 1 :** size [-2,-3] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_bksl*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATBKSL.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_bksl.c
- SCI/modules/scicos_blocks/src/c/matz_bksl.c

See also

- MATDIV - MATDIV Matrix division (xcos Block)
- MATMUL - MATMUL Matrix Multiplication (xcos Block)

Authors

Fady NASSIF - INRIA

Name

MATCATH — Horizontal Concatenation

Block Screenshot



Contents

- Horizontal Concatenation
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The MATCATH Block outputs the horizontal concatenation of multiple matrices. It is also called concatenation according to the columns. The inputs $U_1, U_2, \dots, U_n$ must have the same number of rows. The output is a M -by- $(N_1+N_2+\dots+N_n)$ matrix, where $N_1, N_2, \dots, N_n$ are the numbers of columns of the inputs matrices, and M is the number of rows.

The equivalent of MATCATH in Scilab is $y=[U_1 \ U_2 \ \dots \ U_n]$.

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **number of columns of each matrix**

It indicates the number of columns of the inputs matrices.

Properties : Type 'mat' of size [-1,1].

Example

```
A=[1 2 3;4 5 6]
B=[7 8;9 10]
The result of the horizontal concatenation is:
C=[1 2 3 7 8;4 5 6 9 10]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-3] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_cath*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATCATH.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_cath.c

- `SCI/modules/scicos_blocks/src/c/matz_cath.c`

See also

- `MATCATV` - `MATCATV` Vertical Concatenation (xcos Block)

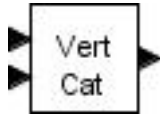
Authors

Fady NASSIF - INRIA

Name

MATCATV — Vertical Concatenation

Block Screenshot



Contents

- Vertical Concatenation
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The MATCATV Block outputs the vertical concatenation of multiple matrices. It is also called concatenation according to the rows. The inputs $U_1, U_2, \dots, U_n$ must have the same number of columns. The output is a $(M_1 + M_2 + \dots + M_n)$ -by- N matrix, where $M_1, M_2, \dots, M_n$ are the numbers of rows of the inputs matrices, and N is the number of columns.

The equivalent of MATCATH in Scilab is $y = [U_1; U_2; \dots; U_n]$

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **number of line of each matrix**

It indicates the number of rows of the inputs matrices.

Properties : Type 'mat' of size [-1,1].

Example

```
A=[1 2;3 4;5 6]
B=[7 8;9 10]
The result of the horizontal concatenation is:
C=[1 2;3 4;5 6;7 8;9 10]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-3] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_catv*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATCATV.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_catv.c

- `SCI/modules/scicos_blocks/src/c/matz_catv.c`

See also

- `MATCATH` - `MATCATH`: Horizontal Concatenation (xcos Block)

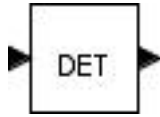
Authors

Fady NASSIF - INRIA

Name

MATDET — Matrix Determinant

Block Screenshot



Contents

- Matrix Determinant
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

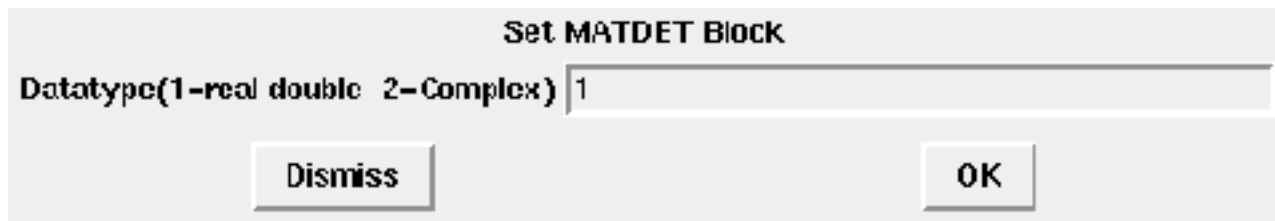
- Matrix operation palette

Description

The MATDET outputs the determinant of a square input matrix. If the input $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$ then the output of the block has the form of: $y = A_{11} \cdot (A_{22} \cdot A_{33} - A_{23} \cdot A_{32}) - A_{12} \cdot (A_{21} \cdot A_{33} - A_{23} \cdot A_{31}) + A_{13} \cdot (A_{21} \cdot A_{32} - A_{22} \cdot A_{31})$.

The equivalent of MATDET in Scilab is "det"

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example

```
U=[1 0 63;2 -2 5;9 9 4]
y=2215
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_det*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDET.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_det.c
- SCI/modules/scicos_blocks/src/c/matz_det.c

See also

- MATINV - MATINV Matrix Inverse (xcos Block)

Authors

Fady NASSIF - INRIA

Name

MATDIAG — Create Diagonal Matrix

Block Screenshot



Contents

- Create Diagonal Matrix
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The MATDIAG block create a diagonal matrix from a 1D vector. If the input is a M-by-1 vector than the output is an M-by-M matrix.

Dialog box



- **Datatype (1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example

```
if the input of the block is U=ones(5,1) then the output is:  
y=[1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_diag*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDIAG.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_diag.c
- SCI/modules/scicos_blocks/src/c/matz_diag.c

See also

- EXTTRI - EXTTRI: Triangular or Diagonal extraction (xcos Block)

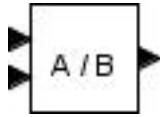
Authors

Fady NASSIF - INRIA

Name

MATDIV — Matrix division

Block Screenshot



Contents

- Matrix division
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

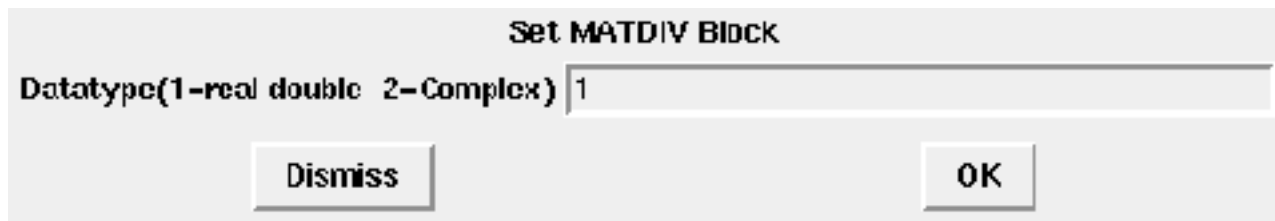
- Matrix operation palette

Description

The MATDIV block outputs the right matrix division. It is a solution to $x*B=A$. The higher input is the A matrix, the lower one is the B matrix, and the output is x. If A is an M1-by-N matrix, B must be a M2-by-N where M1 and M2 can be different or equal. The output x is a M1-by-M2 matrix.

The equivalent of BACKSLASH is "/" in Scilab.

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example

```
A=[1 7 3;23 32 29]
B=[21 18 34;13 10 19;11 54 36]
then the result of the x*B=A equation is (A/B):
x=[-0.475 0.712 0.156;-4.350 8.381 0.491]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [-1,-3] / **type 1**
 - **port 2 :** size [-2,-3] / **type 1**
- **regular outputs:**
 - **port 1 :** size [-1,-2] / **type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_div*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDIV.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_div.c
- SCI/modules/scicos_blocks/src/c/matz_div.c

See also

- MATBKSL - MATBKSL:left matrix division
- MATMUL - MATMUL Matrix Multiplication

Authors

Fady NASSIF - INRIA

Name

MATEIG — Matrix Eigenvalues

Block Screenshot



Contents

- Matrix Eigenvalues
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

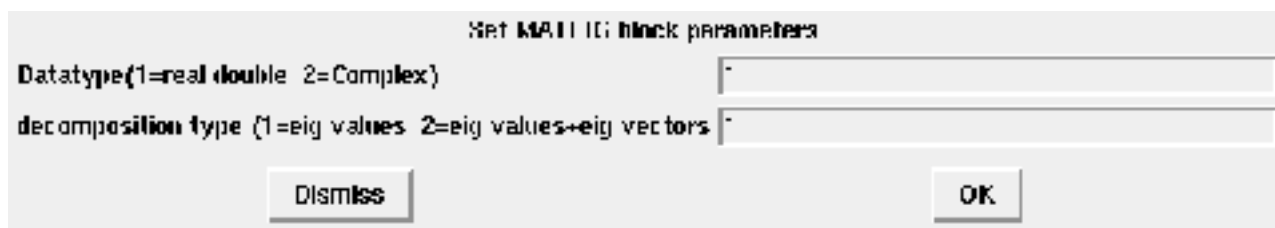
The MATEIG calculate the eigenvalues and the eigenvectors of a square input matrix U .

When the "Decomposition type" is set to 1, the block outputs the eigenvalues in a vector form, if the input is a M-by-M matrix the output is a M-by-1 vector.

When the "Decomposition type" is set to 2, the block outputs two matrices. for an M-by-M input matrix, the first output is a M-by-M diagonal matrix composed by the eigenvalues, and the second is a M-by-M matrices composed by the eigenvectors; the eigenvectors are represented by the columns of the matrix.

The equivalent of the MATEIG block in Scilab is "spec(A)"

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **decomposition type (1=eig values 2=eig values+eig vectors)**

To select the form of the output.

Properties : Type 'vec' of size 1.

Example

```
A=[1 12 32;21 35 46;70 8 19]
When the "Decomposition type" is set to one the output is y=[-35.649;14.279;76.369]
When the "Decomposition type" is set to two the outputs are:
y1=[-35.649 0.000 0.000;0.000 14.279 0.000;0.000 0.000 76.369]
y2=[0.557 0.080 0.349;0.330 -0.922 0.770;-0.762 0.379 0.533]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 2
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_vps*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATEIG.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_vps.c

- `SCI/modules/scicos_blocks/src/c/mat_vpv.c`
- `SCI/modules/scicos_blocks/src/c/matz_vps.c`
- `SCI/modules/scicos_blocks/src/c/matz_vpv.c`

See also

- MATSING - MATSING SVD decomposition (xcos Block)

Authors

Fady NASSIF - INRIA

Name

MATEXPM — Matrix Exponential

Block Screenshot



Contents

- Matrix Exponential
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

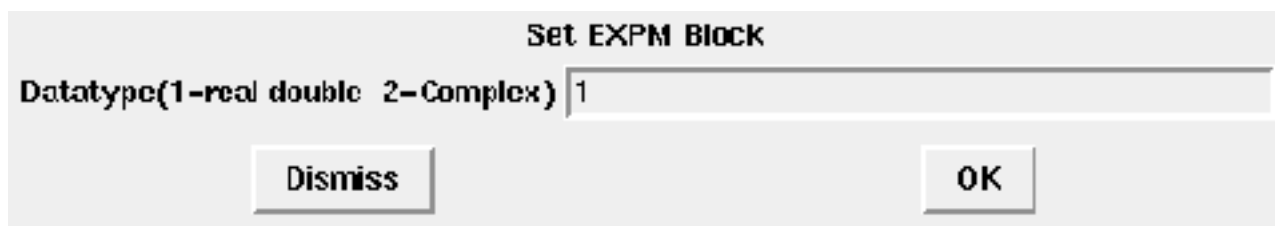
Palette

- Matrix operation palette

Description

The MATEXPM outputs the matrix exponential of a square matrix input by the pade's approximants. The output is a square matrix with the same size of the input. The equivalent of this block in Scilab is "expm".

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example

```
u=[1 2 3;2 3 1;4 2 1]
y=[182.612 196.518 141.735;172.973 190.770 133.577;204.677 220.063 159.067]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [-1,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_expm*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATEXPM.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_expm.c
- SCI/modules/scicos_blocks/src/c/matz_expm.c

See also

- MATMUL - MATMUL Matrix Multiplication (xcos Block)

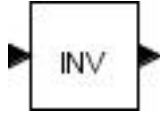
Authors

Fady NASSIF - INRIA

Name

MATINV — Matrix Inverse

Block Screenshot



Contents

- Matrix Inverse
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

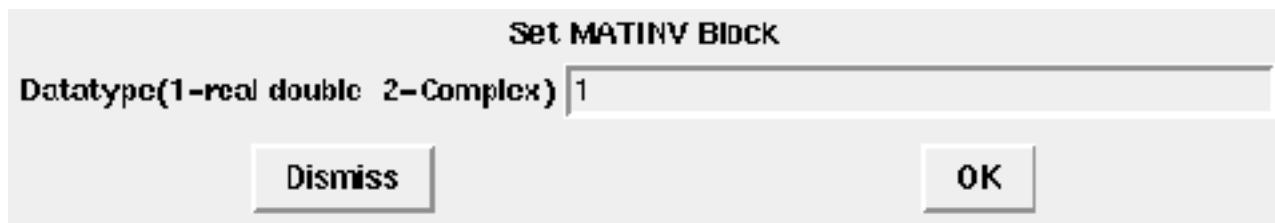
Palette

- Matrix operation palette

Description

The MATINV Block outputs the inverse of a square input matrix using the LU factorization. A warning message is printed if the input is badly scaled or nearly singular. The equivalent function of this block in Scilab is "inv".

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [-1,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_inv*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATINV.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_inv.c
- SCI/modules/scicos_blocks/src/c/matz_inv.c

See also

- MATLU - MATLU LU Factorization (xcos Block)
- MATPINV - MATPINV Matrix PseudoInverse (xcos Block)

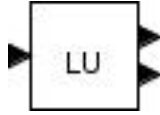
Authors

Fady NASSIF - INRIA

Name

MATLU — LU Factorization

Block Screenshot



Contents

- LU Factorization
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

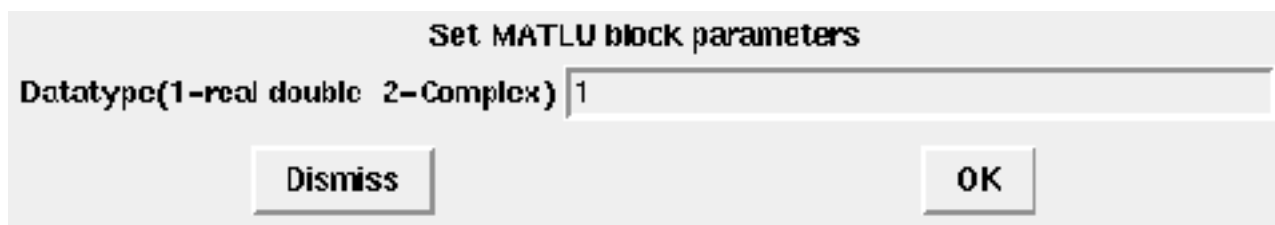
Palette

- Matrix operation palette

Description

The MATLU Block outputs two matrices L and U, with row pivoting, from the LU factorization of a square input matrix. If A is the input matrix then $E*A=L*U$ where E is the permutation matrix. The equivalent function of this block in Scilab is "[l,u,e]=lu(A)"

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [-1,-1] / type 1
 - port 2 : size [-1,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_lu*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATLU.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_lu.c
- SCI/modules/scicos_blocks/src/c/matz_lu.c

See also

- MATINV - MATINV Matrix Inverse (xcos Block)

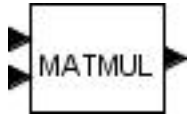
Authors

Fady NASSIF - INRIA

Name

MATMUL — Matrix Multiplication

Block Screenshot



Contents

- Matrix Multiplication
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

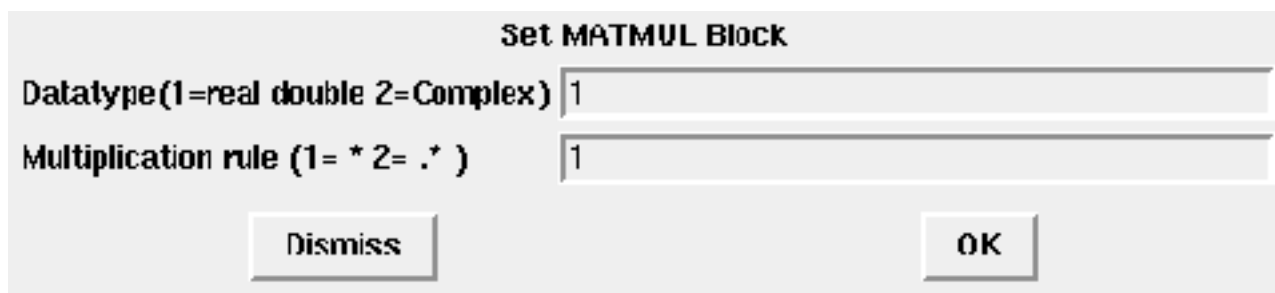
Palette

- Matrix operation palette

Description

The MATMUL block computes the matrix multiplication of two inputs matrices. The number of rows of the second matrix must be equal to the number of columns of the first matrix. The output is a matrix where the number of rows is equal to that of the first input matrix and the number of columns is equal to that of the second input matrix.

Dialog box

A dialog box titled "Set MATMUL Block". It contains two input fields. The first field is labeled "Datatype(1=real double 2=Complex)" and has the value "1" entered. The second field is labeled "Multiplication rule (1= * 2= .*)" and also has the value "1" entered. At the bottom, there are two buttons: "Dismiss" on the left and "OK" on the right.

Set MATMUL Block	
Datatype(1=real double 2=Complex)	1
Multiplication rule (1= * 2= .*)	1
Dismiss	OK

- Datatype(1=real double 2=Complex)

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Multiplication rule** ($1 = * 2 = .*$)

?

Properties : Type 'vec' of size 1.

Example

```
A=[1 2 3;4 5 6]
B=[9 8 7 6;5 4 3 2;9 7 5 3]
y=[46 37 28 19;115 94 73 52]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-2,-3] / type 1
- **regular outputs:**
 - port 1 : size [-1,-3] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matmul_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATMUL.sci

Computational function

- SCI/modules/scicos_blocks/src/c/matmul_m.c
- SCI/modules/scicos_blocks/src/c/matmul_m.c

- `SCI/modules/scicos_blocks/src/c/matmul2_m.c`
- `SCI/modules/scicos_blocks/src/c/matzmul2_m.c`

See also

- INTMUL - INTMUL integer matrix multiplication (xcos Block)
- MATDIV - MATDIV Matrix division (Scicos Block)
- MATBKSL - MATBKSL:left matrix division (xcos Block)
- MATEXPM - MATEXPM Matrix Exponential (xcos Block)
- SUMMATION - SUMMATION Matrix Summation (xcos Block)

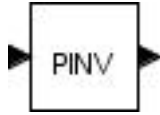
Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

Name

MATPINV — Matrix PseudoInverse

Block Screenshot



Contents

- Matrix PseudoInverse
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The MATPINV Block outputs the inverse of a non square input matrix using the SVD theory.if the SVD decomposition of A is equal to:

$$A=USV'$$

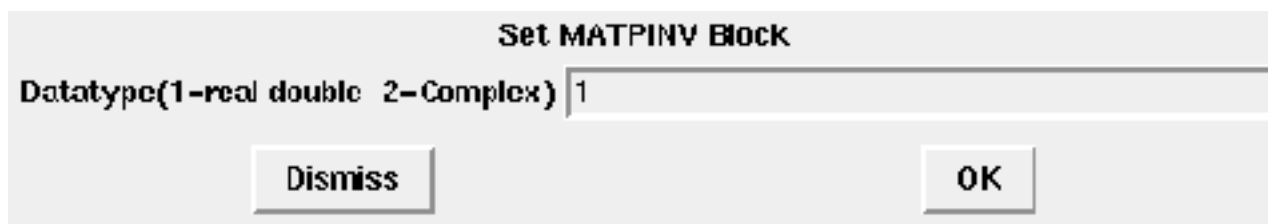
The pseudoinverse x of A is given by:

$X=VS''U'$ where $S''(i,j)=1/S(i,j)$ (if $S(i,j) \neq 0$), U' and V are respectively the transpose of U and V' .

and we have $A*X*A=A$ and $X*A*X=X$. Both $A*X$ and $X*A$ are Hermitian . A warning message is printed if the input is badly scaled or nearly singular.

When the input is a M-by-N matrix the output is a N-by-M matrix. The equivalent function of this block in Scilab is "pinv".

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 : size [-1,-2] / type 1**
- **regular outputs:**
 - **port 1 : size [-2,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_pinv*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATPINV.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_pinv.c
- SCI/modules/scicos_blocks/src/c/matz_pinv.c

See also

- MATINV - MATINV Matrix Inverse (xcos Block)
- MATSING - MATSING SVD decomposition (xcos Block)

Authors

Fady NASSIF - INRIA

Name

MATRESH — Matrix Reshape

Block Screenshot



Contents

- Matrix Reshape
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

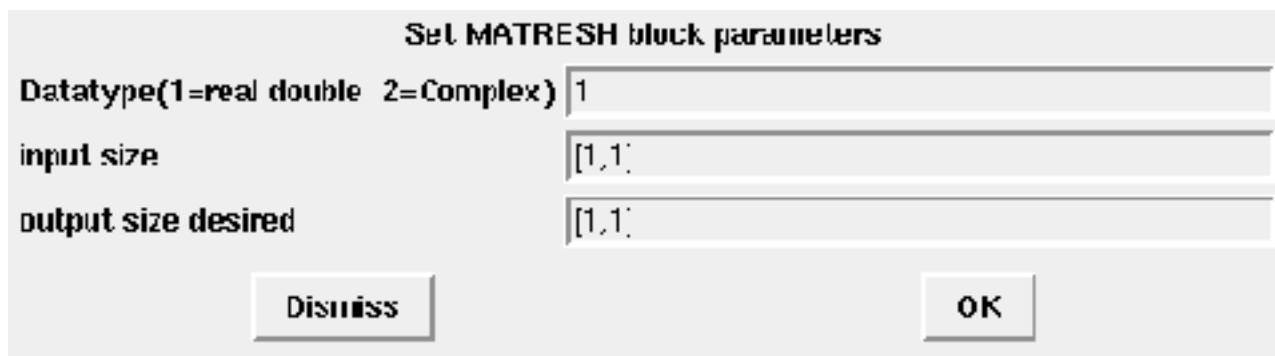
Palette

- Matrix operation palette

Description

The RESHAPE block changes the dimensions of a matrix or a vector to another dimensions specified by the user in the "output size desired" label. The output size must be less or equal to the input size.

Dialog box



Set MATRESH block parameters

Datatype(1=real double 2=Complex) | 1

input size | [1,1]

output size desired | [1,1]

Dismiss OK

- Datatype(1=real double 2=Complex)

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size -1.

- **input size**

It indicates the size of the input matrix.

Properties : Type 'vec' of size -1.

- **output size desired**

It indicates the desired output's size.

Properties : Type 'vec' of size -1.

Example

```
u=[1 2 3 4;5 6 7 8]
When the output desired is [1;6] the output is:
y=[1 2 3 4 5 6]
```

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_reshape*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATRESH.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_reshape.c

- `SCI/modules/scicos_blocks/src/c/matz_reshape.c`

See also

- `EXTRACT` - `EXTRACT`: Matrix Extractor (xcos Block)

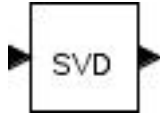
Authors

Fady NASSIF - INRIA

Name

MATSING — SVD decomposition

Block Screenshot



Contents

- SVD decomposition
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Matrix operation palette

Description

The MATSING block computes the economy sized SVD of the M-by-N input matrix A by finding U,S and V such that

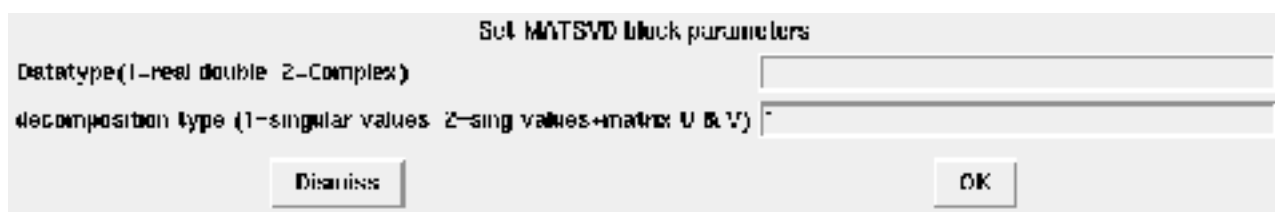
$$A=U*S*V'.$$

When the decomposition type is set to one, the output is a vector composed by the singular values.

When the decomposition type is set to two, we have three outputs: the second output is a diagonal matrix S composed by the singular values and the other two outputs are the unitary matrices U and V.

The equivalent function of this block in Scilab is "svd".

Dialog box



- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **decomposition type (1=singular values 2=sing values+matrix U & V)**

It indicates the form of the output. When it is set to one, we have a unique vector output (singular values). When it is set to two we have three same sizes matrices(U,S,V).

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sing*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATSING.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_sing.c
- SCI/modules/scicos_blocks/src/c/mat_svd.c
- SCI/modules/scicos_blocks/src/c/matz_sing.c
- SCI/modules/scicos_blocks/src/c/matz_svd.c

See also

- MATEIG - MATEIG Matrix Eigenvalues (xcos Block)

- MATLU - MATLU LU Factorization (xcos Block)

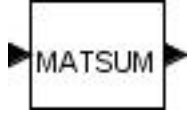
Authors

Fady NASSIF - INRIA

Name

MATSUM — Matrix Sum

Block Screenshot



Contents

- Matrix Sum
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

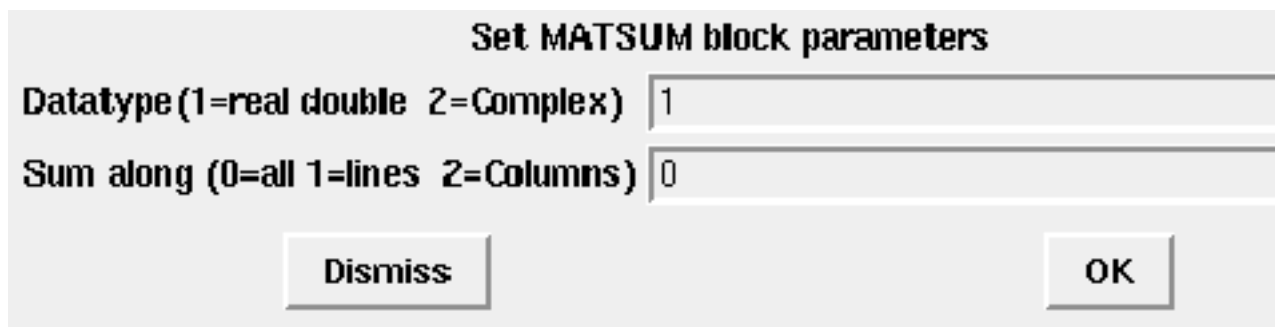
Palette

- Matrix operation palette

Description

The MATSUM block returns the sum of the element of an input matrix/vector. When the Sum along is set to all the block outputs the sum of all the elements of the matrix. The output is then a scalar. When the Sum along is set to lines the block is a row-wise sum. The output is a row vector. When the Sum along is set to Columns the block is a column-wise sum. The output is a column vector. The equivalent function of this block in Scilab is: "sum".

Dialog box



- Datatype(1=real double 2=Complex)

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Sum along (0=all 1=lines 2=Columns)**

Indicates the used rule to sum. For more information see the description part.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sum*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATSUM.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mat_sum.c
- SCI/modules/scicos_blocks/src/c/mat_suml.c
- SCI/modules/scicos_blocks/src/c/mat_sumc.c
- SCI/modules/scicos_blocks/src/c/matz_sum.c
- SCI/modules/scicos_blocks/src/c/matz_suml.c
- SCI/modules/scicos_blocks/src/c/matz_sumc.c

See also

- SUBMAT - SUBMAT Sub-matrix extraction (xcos Block)

- SUMMATION - SUMMATION Matrix Summation (xcos Block)

Authors

Fady NASSIF - INRIA

Name

MATTRAN — Matrix Transpose

Block Screenshot



Contents

- Matrix Transpose
- the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
- Example
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

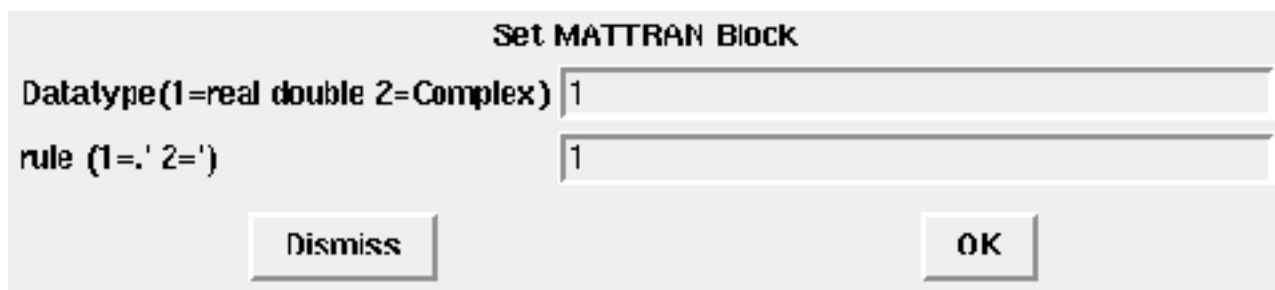
Palette

- Matrix operation palette

Description

This Block transposes an $M \times N$ matrix to a $N \times M$ matrix. For complex data type it uses the hermitian transpose. The equivalent of this block in Scilab is $y=u'$.

Dialog box

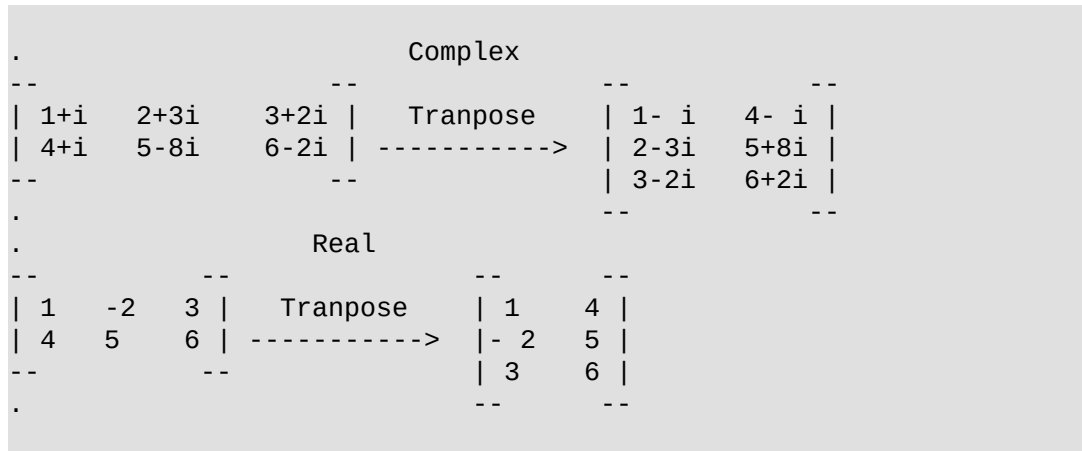


- **Datatype(1=real double 2=Complex)**

It indicates the data type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

Example



Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-2,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mattran_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATTRAN.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mattran_m.c
- SCI/modules/scicos_blocks/src/c/matztran_m.c

See also

- `Matrix_pal` - Matrix operation palette (xcos Palette)

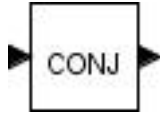
Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

Name

MATZCONJ — Matrix Conjugate

Block Screenshot



Contents

- Matrix Conjugate
- • the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Matrix operation palette

Description

This blocks computes the conjugate of a complex input matrix.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 2
- **regular outputs:**
 - port 1 : size [-1,-2] / type 2
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_conj*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATZCONJ.sci

Computational function

- SCI/modules/scicos_blocks/src/c/matz_conj.c

See also

- Matrix_pal - Matrix operation palette (xcos Palette)

Authors

Fady NASSIF - INRIA

Name

RICC — Riccati Equation

Block Screenshot



Contents

- Riccati Equation
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Matrix operation palette

Description

This block computes the solution of Riccati equation using different method and for both case continuous and discrete.

The Riccati equation in continuous time is:

$$A^*X + X^*A + C - X^*D^*X = 0$$

The Riccati equation in discrete time is:

$$A^*X^*(\text{inv}(\text{In}-D))^*A - X + C = 0$$

where A is an NxN matrix, it is the first input of the block, C and D are two NxN symmetric matrices and are respectively the second and third input of the RICC block. X represent the output of the block, it is also a NxN matrix.

The user can choose between two methods of computation. For the continuous time he can use even the Schur method or the matrix sign function approach method, by setting the Model parameter to 1 or 2. For the discrete time, the models are the Schur method and the inverse free spectral decomposition method.

Dialog box

Set RICC Block

Type (1=Cont 2=Disc)

Model(1=Schr 2=sign(cont) inv(disc))

- **Type (1=Cont 2=Disc)**

For continuous time signal set this parameter to 1. For discrete input time set it to 2.

Properties : Type 'vec' of size 1.

- **Model(1=Schr 2=sign(cont) inv(disc))**

To use the Shur method in computation set this parameter to 1. To use matrix sign function approach in continuous case or the inverse free spectral decomposition method in discrete case set this parameter to 2.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-1] / type 1
 - port 2 : size [-1,-1] / type 1
 - port 3 : size [-1,-1] / type 1
- **regular outputs:**
 - port 1 : size [-1,-1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ricc_m*

Interfacing function

- `SCI/modules/scicos_blocks/macros/MatrixOp/RICC.sci`

Computational function

- `SCI/modules/scicos_blocks/src/c/ricc_m.c`

See also

- `Matrix_pal` - Matrix operation palette (xcos Palette)

Authors

Fady NASSIF - INRIA

Name

ROOTCOEF — Coefficient computation

Block Screenshot



Contents

- Coefficient computation
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Matrix operation palette

Description

This block computes the coefficients of a polynomial given its root values.

Dialog box



- **Datatype(1=real double 2=Complex)**

This block can only support double inputs values. These values can be real or complex.

Properties : Type 'vec' of size 1.

- **input row size**

The input row size.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-2,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *root_coef*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/ROOTCOEF.sci

Computational function

- SCI/modules/scicos_blocks/src/c/root_coef.c
- SCI/modules/scicos_blocks/src/c/rootz_coef.c

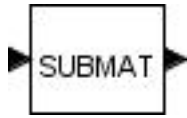
Authors

Fady NASSIF - INRIA

Name

SUBMAT — Sub-matrix extraction

Block Screenshot



Contents

- Sub-matrix extraction
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Matrix operation palette

Description

This block outputs a sub matrix of the input matrix. The output matrix will be defining by using the parameters of this block.

Dialog box

Set SUBMAT Block	
Datatype (1- real double 2-Complex)	1
Starting Row Index	1
Ending Row Index	1
Starting Column Index	1
Ending Column Index	1
Input Dimensions	[1,1]
DismissOK	

- **Datatype (1=real double 2=Complex)**

Type of the output matrix. It can be double or complex.

Properties : Type 'vec' of size 1.

- **Starting Row Index**

The first row of the submatrix.

Properties : Type 'vec' of size 1.

- **Ending Row Index**

The last row of the Submatrix.

Properties : Type 'vec' of size 1.

- **Starting Column Index**

The first column of the submatrix.

Properties : Type 'vec' of size 1.

- **Ending Column Index**

The last row of the submatrix.

Properties : Type 'vec' of size 1.

- **Input Dimension**

The Matrix input dimensions.

Properties : Type 'vec' of size 2.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *submat*

Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/SUBMAT.sci

Computational function

- SCI/modules/scicos_blocks/src/c/submat.c
- SCI/modules/scicos_blocks/src/c/submatz.c

Authors

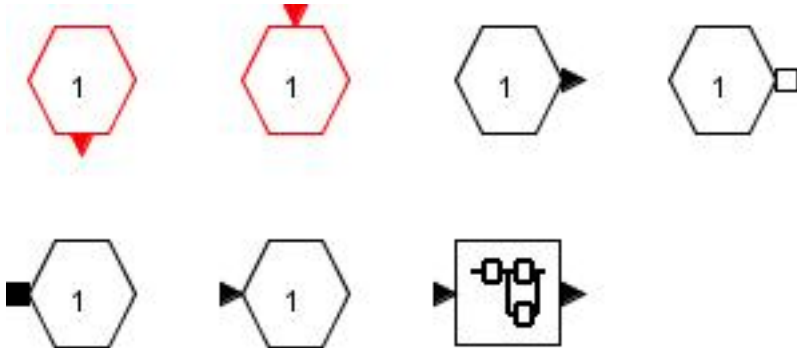
Fady NASSIF - INRIA

14. Port & Subsystem palette

Name

Portaction_pal — Port & Subsystem palette

Block Screenshot



Module

- xcOS

Description

The Port & Subsystem palette includes blocks for creating subsystems.

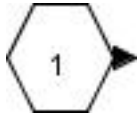
Blocks

- CLKINV_f — Input activation port Event-Select
- CLKOUTV_f — Output activation port Event-Select
- IN_f - Input Port
- INIMPL_f — Input implicit port
- OUTIMPL_f — Output implicit port
- OUT_f - Output Port
- SUPER_f — Super block

Name

IN_f — Input Port

Block Screenshot



Contents

- Input Port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

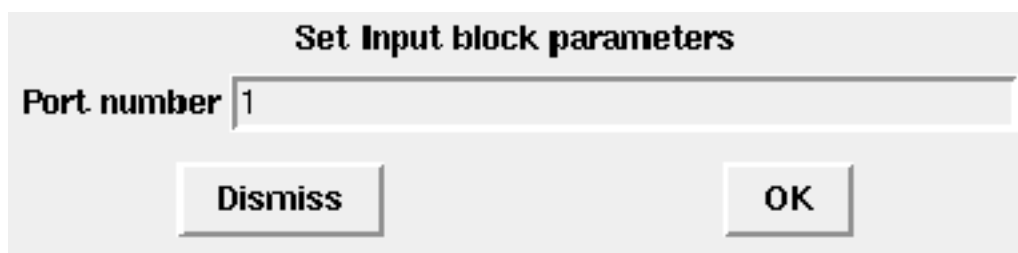
Palette

- Port & Subsystem palette

Description

This block must only be used inside xcoss Super Blocks to represent a regular input port. The input size is determined by the context. In a Super Block, regular input ports must be numbered from 1 to the number of regular input ports.

Dialog box



- **Port number**
an integer defining the port number.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,-2] / type -1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *input*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/IN_f.sci

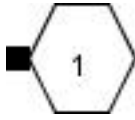
Authors

Ramine Nikoukhah - INRIA

Name

OUTIMPL_f — Output implicit port

Block Screenshot



Contents

- Output implicit port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

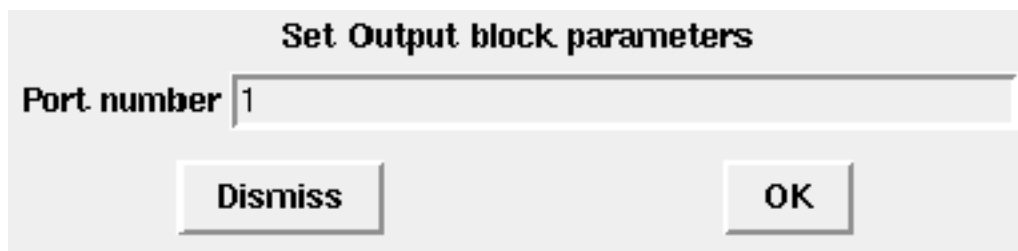
Palette

- Port & Subsystem palette

Description

Outport blocks are the links from a system to a destination outside the system.

Dialog box



- **Port number**
Specify the port number of the Outport block.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *outimpl*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/OUTIMPL_f.sci

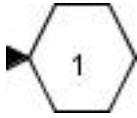
Authors

Ramine Nikoukhah - INRIA

Name

OUT_f — Output Port

Block Screenshot



Contents

- Output Port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

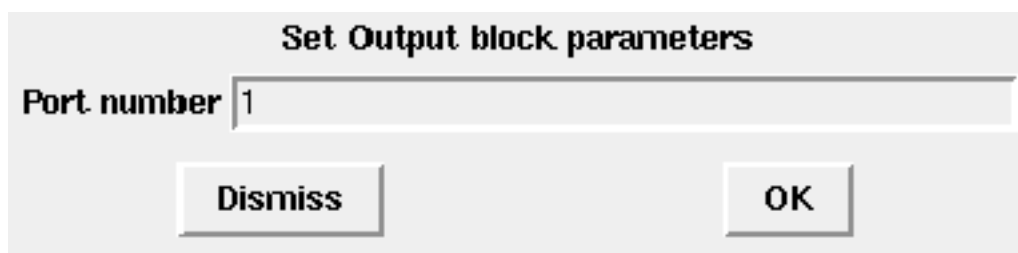
Palette

- Port & Subsystem palette

Description

This block must only be used inside Xcos Super Blocks to represent a regular output port. In a Super Block, regular output ports must be numbered from 1 to the number of regular output ports. Size of the output is determined by the compiler according to the connected blocks port sizes.

Dialog box



- **Port number**
an integer defining the port number.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type -1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *output*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/OUT_f.sci

Authors

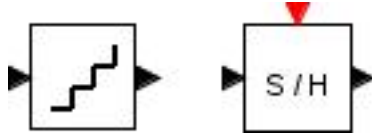
Ramine Nikoukhah - INRIA

15. Signal processing palette

Name

Signalprocessing_pal — Signal processing palette

Block Screenshot



Module

- xcOS

Description

The signal processing palette contains blocks designed specifically for signal processing applications.

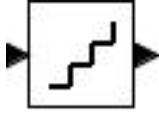
Blocks

- MCLOCK_f
- MFCLCK_f
- QUANT_f — Quantization
- SAMPHOLD_m — Sample and hold

Name

QUANT_f — Quantization

Block Screenshot



Contents

- Quantization
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

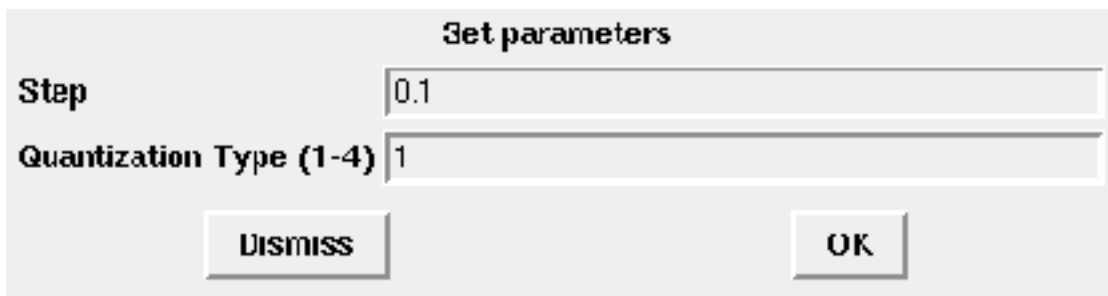
Palette

- Signal processing palette

Description

This block outputs the quantization of the input according to a choice of methods:

Dialog box



The dialog box is titled "Set parameters". It contains two input fields: "Step" with the value "0.1" and "Quantization Type (1-4)" with the value "1". At the bottom, there are two buttons: "Dismiss" and "OK".

- **Step**
scalar, Quantization step
Properties : Type 'vec' of size 1.
- **Quantization Type**
scalar with possible values 1,2,3 or 4 Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *qzrnd*

Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/QUANT_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/qzrnd.f
- SCI/modules/scicos_blocks/src/fortran/qztrn.f
- SCI/modules/scicos_blocks/src/fortran/qzflr.f
- SCI/modules/scicos_blocks/src/fortran/qzcel.f

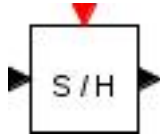
Authors

Ramine Nikoukhah - INRIA

Name

SAMPHOLD_m — Sample and hold

Block Screenshot



Contents

- Sample and hold
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

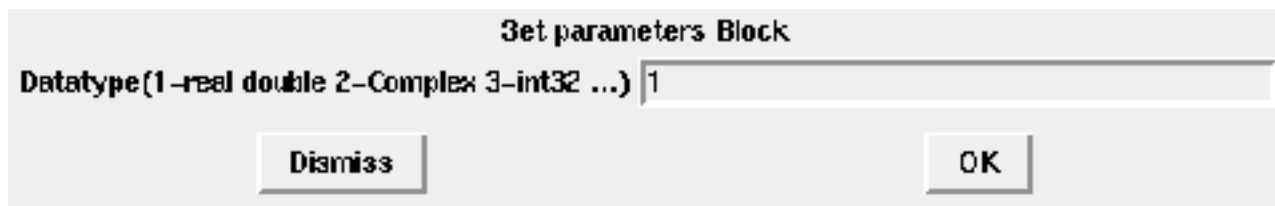
Palette

- Signal Processing palette

Description

Each time an input event is received block copy its input on the output and hold it until input event.
For periodic Sample and hold, event input must be generated by a **Clock**.

Dialog box



- **Datatype(1=real double 2=Complex 3=int32 ...)**
Output datatype. This block can support all data types.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *samphold4_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SAMPHOLD_m.sci

Computational function

- SCI/modules/scicos_blocks/src/c/samphold4_m.c (Type 4)

Authors

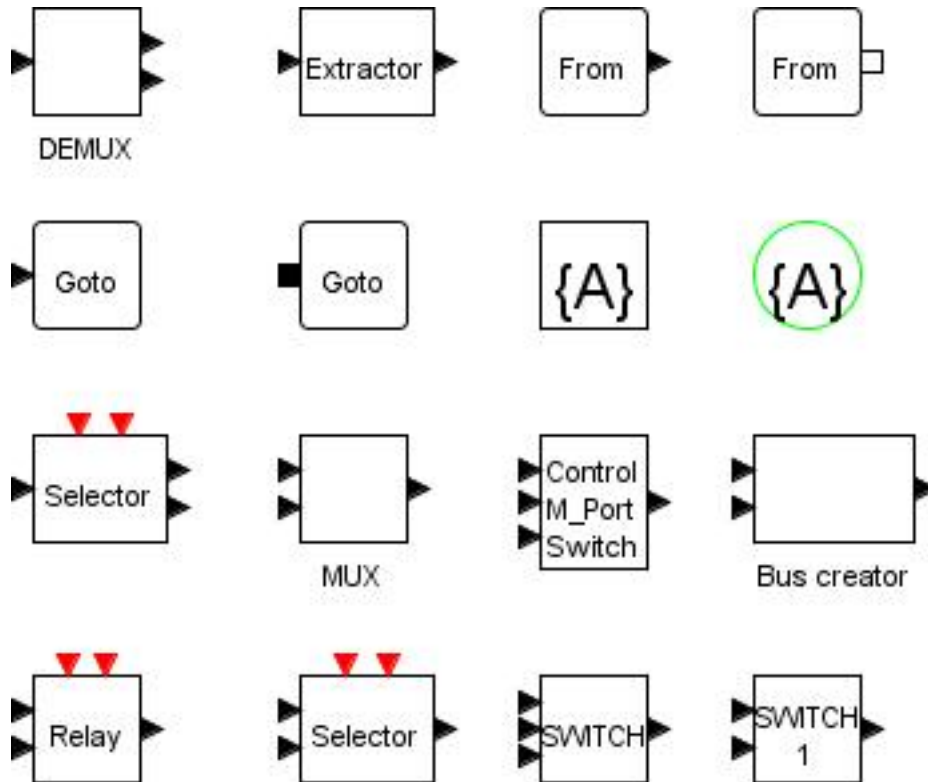
Fady NASSIF - INRIA

16. Signal routing palette

Name

Signalrouting_pal — Signal routing palette

Block Screenshot



Module

- xcOS

Description

The Signal routing palette includes blocks that transport signals from one point in a block diagram to another.

Blocks

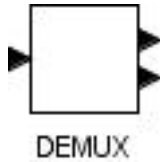
- DEMUX — Demultiplexer
- EXTRACTOR — Extractor
- FROM — FROM Receives data from a corresponding GOTO
- FROMMO — Receives data from a corresponding GOTOMO
- GOTO — GOTO Pass block input to From block
- GOTOMO — Pass block input to FROMMO block
- GotoTagVisibility — Define Scope of GOTO tag visibility
- GotoTagVisibilityMO — Define Scope of GOTOMO tag visibility
- ISELECT_m — Iselect

- MUX — Multiplexer
- M_SWITCH — Multi-port switch
- NRMSOM_f — Merge data
- RELAY_f — Relay
- SELECT_m — Select
- SWITCH2_m — Switch2
- SWITCH_f — Switch

Name

DEMUX — Demultiplexer

Block Screenshot



Contents

- Demultiplexer
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

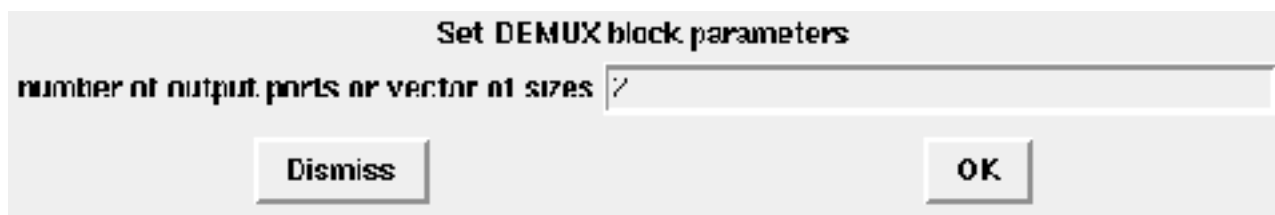
Palette

- Signal routing palette

Description

Given a vector valued input this block splits inputs over vector valued outputs. So , where are numbered from top to bottom. Input and Output port sizes are determined by the context.

Dialog box



- **number of output ports or vector of sizes**

positive integer less than or equal to 8 .

Properties : Type 'vec' of size -1

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [0,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-2,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *multiplex*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/DEMUX.sci

Computational function

- SCI/modules/scicos_blocks/src/c/multiplex.c (Type 4)

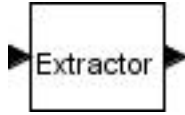
Authors

Ramine Nikoukhah - INRIA

Name

EXTRACTOR — Extractor

Block Screenshot



Contents

- Extractor
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

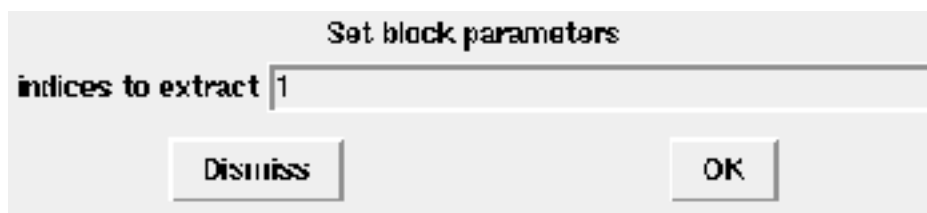
Palette

- Signal routing palette

Description

Extracts or select a regular input from a multiple regular input.

Dialog box



- **indices to extract**
a regular input to be extracted from the multiple regular inputs.
Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extractor*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/EXTRACTOR.sci

Computational function

- SCI/modules/scicos_blocks/src/c/extractor.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

FROM — FROM Receives data from a corresponding GOTO

Block Screenshot



Contents

- FROM Receives data from a corresponding GOTO
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

Palette

- Signal routing palette

Description

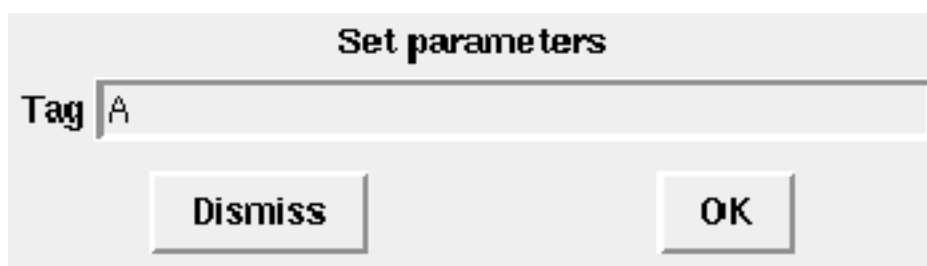
The main role of the GOTO/FROM blocks is to transport signals from a block to another block without connecting them physically. The FROM block transports its received data (from the corresponding GOTO) to its output. Multiples FROM blocks can receive data from one GOTO, although a GOTO can send data to multiple FROM.

The GOTO and FROM blocks are connected by the tag parameter.

For information on the visibility and limitation of these blocks please refer to the GOTO documentation.

This block can support all the data types.

Dialog box



- **Tag**

The tag of the GOTO block passing the signal to this FROM block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,-2] / type -1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *from*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/FROM.sci

See also

- GOTO - GOTO Pass block input to From block
- GotoTagVisibility - Define Scope of GOTO tag visibility

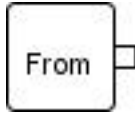
Authors

Fady NASSIF - INRIA

Name

FROMMO — Receives data from a corresponding GOTOMO

Block Screenshot



Contents

- Receives data from a corresponding GOTOMO
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

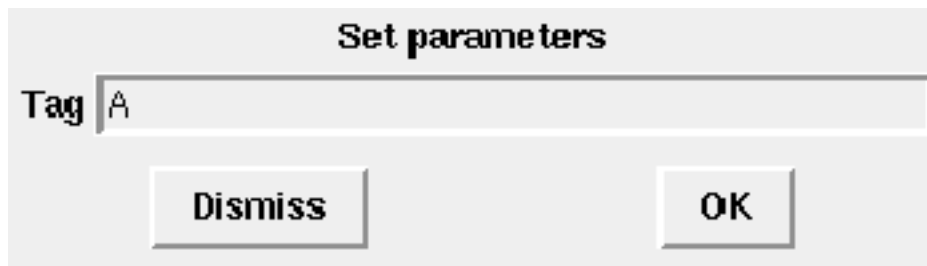
Palette

- Signal Routing palette

Description

This block is used to connect Modelica blocks. For more information on how it works please refer to the documentation of the FROM block by clicking on the link in the "See also" field.

Dialog box



- **Tag**

The tag of the GOTOMO block passing the signal to this FROMMO block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,-2] / type -1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *frommo*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/FROMMO.sci

See also

- FROM - FROM Receives data from a corresponding GOTO (xcos Block)

Authors

Fady NASSIF - INRIA

Name

GOTO — GOTO Pass block input to From block

Block Screenshot



Contents

- GOTO Pass block input to From block
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

Palette

- Signal routing palette

Description

The main role of the GOTO/FROM blocks is to transport signals from a block to another block without connecting them physically. The GOTO block transports its input data to its corresponding FROM block. A simple GOTO block can send data to multiple FROM, although a FROM can receive data from only one GOTO.

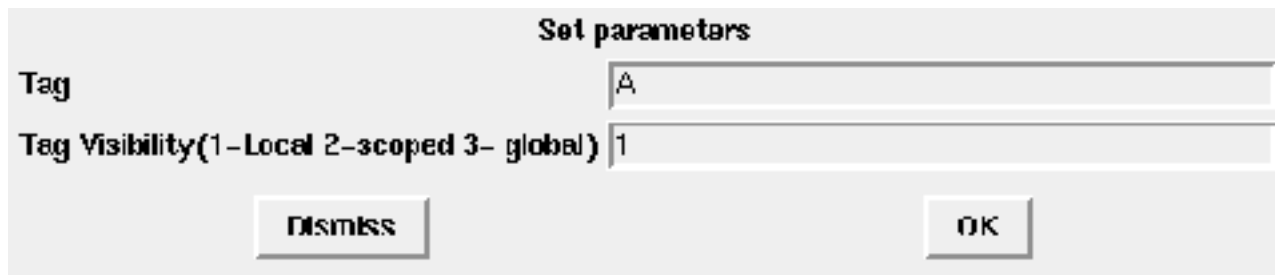
The GOTO and FROM blocks are connected by the tag parameter.

The "Tag Visibility" parameter indicates if the location of the FROM block is limited:

- Local: means that the corresponding FROM of that GOTO must be in the same subsystem.
- Scoped: means that the corresponding FROM of that GOTO must be in the same subsystem or in any subsystem below the GotoTagVisibility block in the model hierarchy.
- Global: means that the corresponding FROM of that GOTO can be anywhere in the model.

This block can support all the data types.

Dialog box



The dialog box is titled "Set parameters". It has two input fields. The first field is labeled "Tag" and contains the text "A". The second field is labeled "Tag Visibility(1-Local 2-scoped 3- global)" and contains the number "1". Below the input fields are two buttons: "Dismiss" on the left and "OK" on the right.

- **Tag**

This parameter identifies the Goto block whose scope is defined in this block.

Properties : Type 'str' of size -1.

- **Tag Visibility(1=Local 2=scoped 3= global)**

This parameter identifies the visibility of the block. It can be local(1), scoped(2) or global(3).

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 :** size [-1,-2] / type -1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *goto*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GOTO.sci

See also

- FROM - FROM Receives data from a corresponding GOTO
- GotoTagVisibility - Define Scope of GOTO tag visibility

Authors

Fady NASSIF - INRIA

Name

GOTOMO — Pass block input to FROMMO block

Block Screenshot



Contents

- Pass block input to FROMMO block
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

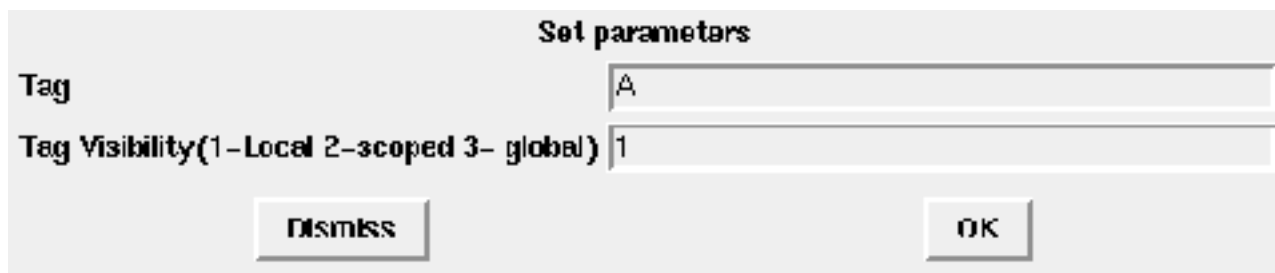
Palette

- Signal Routing palette

Description

This block is used to connect Modelica blocks. For more information on how it works please refer to the documentation of the GOTO block by clicking on the link in the "See also" field.

Dialog box



Set parameters

Tag

Tag Visibility(1-Local 2-scoped 3- global)

- **Tag**
This parameter identifies the Goto block whose scope is defined in this block.
Properties : Type 'str' of size -1.
- **Tag Visibility(1=Local 2=scoped 3= global)**
This parameter identifies the visibility of the block. It can be local(1), scoped(2) or global(3).

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gotomo*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GOTOMO.sci

See also

- GOTO - GOTO Pass block input to From block (xcos Block)

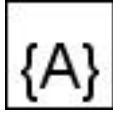
Authors

Fady NASSIF - INRIA

Name

GotoTagVisibility — Define Scope of GOTO tag visibility

Block Screenshot



Contents

- Define Scope of GOTO tag visibility
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

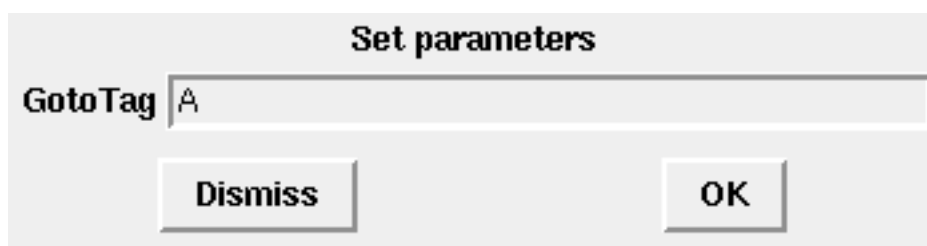
Palette

- Signal routing palette

Description

This block defines the accessibility of the GOTO block when it is configure as "scoped". The FROM block corresponding to that GOTO must be in the same subsystem of the GotoTagVisibility or in subsystems below it in the model hierarchy.

Dialog box



- **GotoTag**

The Goto block tag whose visibility is defined by the location of this block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gototagvisibility*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GotoTagVisibility.sci

See also

- GOTO - GOTO Pass block input to From block
- FROM - FROM Receives data from a corresponding GOTO

Authors

Fady NASSIF - INRIA

Name

GotoTagVisibilityMO — Define Scope of GOTOMO tag visibility

Block Screenshot



Contents

- Define Scope of GOTOMO tag visibility
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “See also”
 - the section called “Authors”

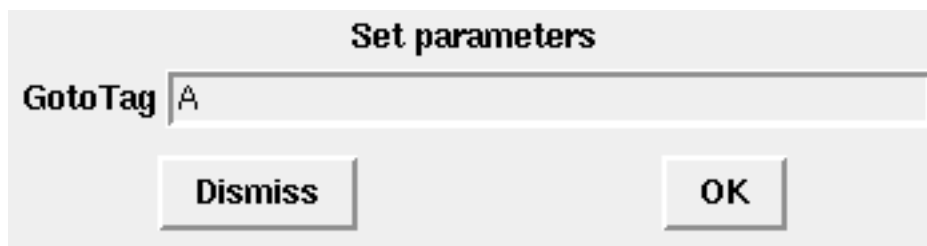
Palette

- Signal Routing palette

Description

This block is used in case of Modelica connection. For more information on how it works please refer to the GotoTagVisibility block by clicking on the link in the "See also" field.

Dialog box



- **GotoTag**

The Goto block tag whose visibility is defined by the location of this block.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gototagvisibilitymo*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GotoTagVisibilityMO.sci

See also

- GotoTagVisibility - Define Scope of GOTO tag visibility (xcos Block)

Authors

Fady NASSIF - INRIA

Name

ISELECT_m — Iselect

Block Screenshot



Contents

- Iselect
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

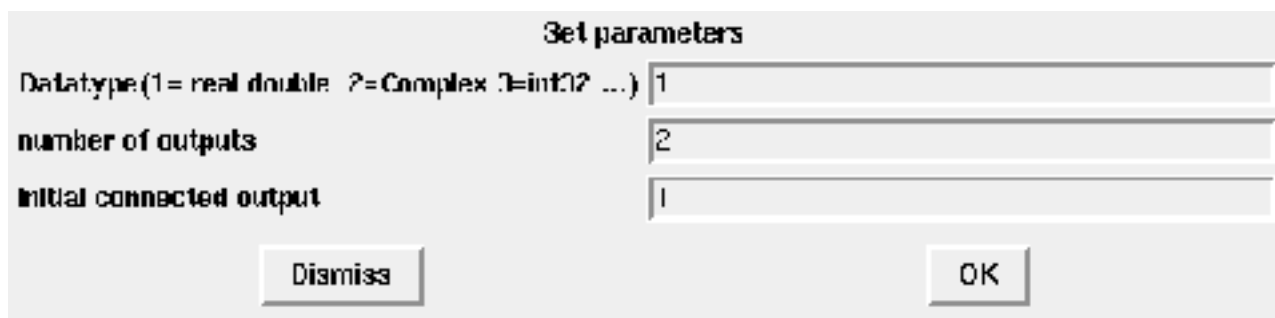
Palette

- Signal routing palette

Description

Select signals from an incoming events. This block has one regular input port.

Dialog box



- **Datatype(1= real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1

- **number of outputs**

a scalar. Number of regular and event outputs.

Properties : Type 'vec' of size 1

- **initial connected output**

an integer. It must be between 1 and the number of inputs.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *selector_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/ISELECT_m.sci

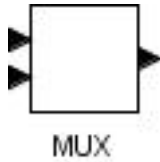
Computational function

- SCI/modules/scicos_blocks/src/c/selector_m.c (Type 4)

Name

MUX — Multiplexer

Block Screenshot



Contents

- Multiplexer
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

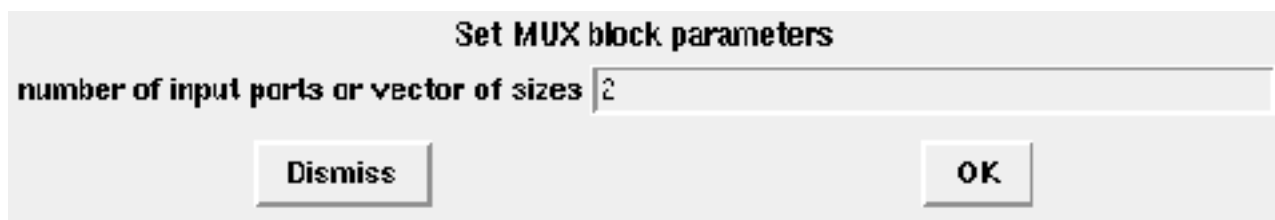
Palette

- Signal routing palette

Description

Given vector valued inputs this block merges inputs in an single output vector. So , where are numbered from top to bottom. Input and Output port sizes are determined by the context.

Dialog box



- **number of input ports or vector of sizes**
integer greater than or equal to 1 and less than 8.
Properties : Type 'vec' of size -1.

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-2,1] / type 1
- **regular outputs:**
 - port 1 : size [0,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *multiplex*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/MUX.sci

Computational function

- SCI/modules/scicos_blocks/src/c/multiplex.c (Type 4)

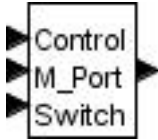
Authors

Ramine Nikoukhah - INRIA

Name

M_SWITCH — Multi-port switch

Block Screenshot



Contents

- Multi-port switch
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

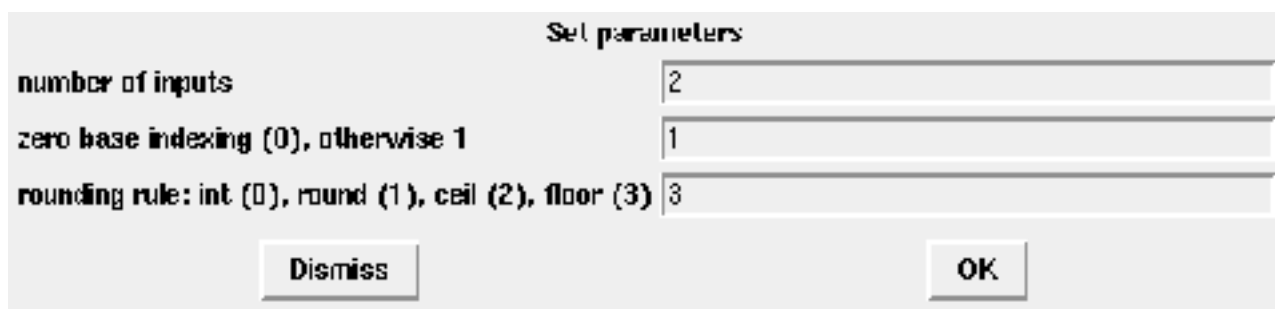
Palette

- Signal routing palette

Description

The Multi-Port Switch block chooses between a number of inputs. The first (top) input is called the control input, while the rest of the inputs are called data inputs. The value of the control input determines which data input is passed through to the output port.

Dialog box



Set parameters	
number of inputs	2
zero base indexing (0), otherwise 1	1
rounding rule: int (0), round (1), ceil (2), floor (3)	3

Dismiss OK

- **number of inputs**
Specify the number of data inputs to the block.
Properties : Type 'vec' of size 1
- **zero base indexing**

If selected, the block uses zero-based indexing. Otherwise, the block uses one-based indexing.

Properties : Type 'vec' of size 1

- **rounding rule:** int

Select the rounding mode for the output.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [-1,1] / type 0
 - port 3 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mswitch*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/M_SWITCH.sci

Computational function

- SCI/modules/scicos_blocks/src/c/mswitch.c (Type 4)

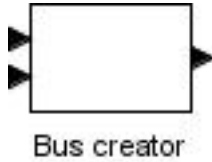
Authors

Ramine Nikoukhah - INRIA

Name

NRMSOM_f — Merge data

Block Screenshot



Contents

- Merge data
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

Palette

- Signal routing palette

Description

The Merge block combines its inputs into a single output line whose value at any time is equal to the most recently computed output of its driving blocks. You can specify any number of inputs by setting the block's Number of inputs parameter.

Dialog box



- **number of inputs**

The number of input ports to be merged.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *junk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/NRMSOM_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

RELAY_f — Relay

Block Screenshot



Contents

- Relay
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Signal routing palette

Description

This block routes one of the regular inputs to the unique regular output. the choice of which input is to be routed is done, initially by the "initial connected input" parameter. Then, every time an input event arrives on the i-th input event port, the i-th regular input port is routed to the regular output.

Dialog box

Set parameters

number of inputs

initial connected input

Dismiss **OK**

- **number of inputs**
a scalar. Number of regular and event inputs.
Properties : Type 'vec' of size 1

- **initial connected input**

an integer. It must be between 1 and the number of inputs.

Properties : Type 'vec' of size 1

Default properties

- **always active:** yes
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *relay*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/RELAY_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/relay.c (Type 2)

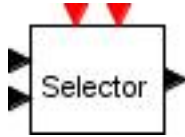
Authors

Ramine Nikoukhah - INRIA

Name

SELECT_m — Select

Block Screenshot



Contents

- Select
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

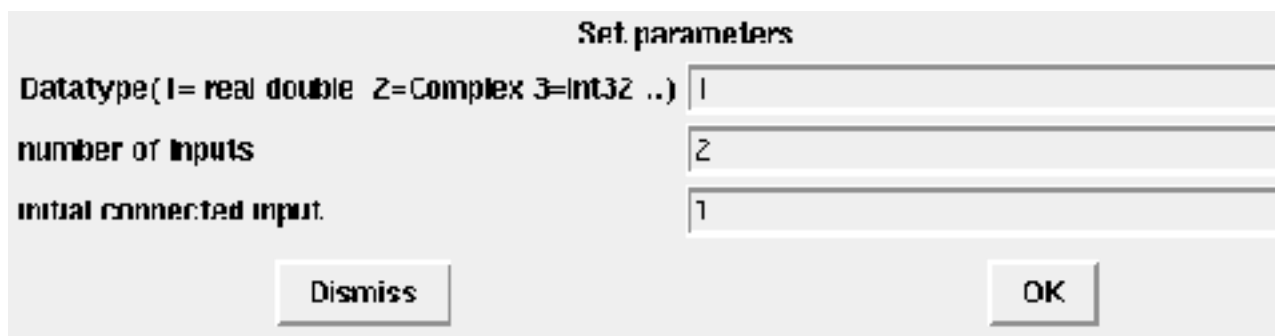
Palette

- Signal routing palette

Description

A Selector block accepts either vector or matrix signals as input. Set the Input Type parameter to the type of signal (vector or matrix) that the block should accept in your model. The parameter dialog box and the block's appearance change to reflect the type of input that you select. The way the block determines the elements to select differs slightly, depending on the type of input.

Dialog box



- **Datatype(1= real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported". Properties : Type 'vec' of size 1

- **number of inputs**

a scalar. Number of regular and event inputs.

Properties : Type 'vec' of size 1

- **initial connected input**

an integer. It must be between 1 and the number of inputs.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *selector_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SELECT_m.sci

Computational function

- SCI/modules/scicos_blocks/src/c/selector_m.c (Type 4)

Authors

- **Fady NASSIF** INRIA
- **Ramine Nikoukhah** INRIA

Name

SWITCH2_m — Switch2

Block Screenshot



Contents

- Switch2
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Signal routing palette

Description

The Switch block passes through the first (top) input or the third (bottom) input based on the value of the second (middle) input. The first and third inputs are called data inputs. The second input is called the control input. You select the conditions under which the first input is passed with the Criteria for passing first input parameter. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in the Criteria for passing first input parameter, then the first input is passed. Otherwise, the third input is passed.

Dialog box

Set parameters	
Datatype (1=real double 2=complex 3=int32 ...)	1
pass first input if: $u2 \geq a$ (0), $u2 > a$ (1), $u2 \neq a$ (2)	0
threshold a	0
use zero crossing: yes (1), no (0)	1
Dismiss OK	

- **Datatype(1= real double 2=Complex)**

a scalar. Give the datatype of the inputs/output.

Properties : Type 'vec' of size 1

-



pass first input if: u2 >=a

Select the conditions under which the first input is passed. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in this parameter, then the first input is passed. Otherwise, the third input is passed.

Properties : Type 'vec' of size 1.

- **threshold a**

Assign the switch threshold that determines which input is passed to the output.

Properties : Type 'vec' of size 1.

- **use zero crossing: yes**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
 - port 2 : size [1,1] / type 0
 - port 3 : size [-1,-2] / type 1
- **regular outputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *switch2_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SWITCH2_m.sci

Computational function

- SCI/modules/scicos_blocks/src/c/switch2_m.c (Type 4)

Authors

- **Fady NASSIF** INRIA
- **Ramine Nikoukhah** INRIA

Name

SWITCH_f — Switch

Block Screenshot



Contents

- Switch
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Signal routing palette

Description

This is a Manual Switch block. It selects one of its inputs to pass through to the output. The selected input is propagated to the output, while the unselected inputs are discarded.

Dialog box

Set switch parameters

number of inputs 2

connected input. 1

Dismiss OK

- **number of inputs**
Specify the number of data inputs to the block.
Properties : Type 'vec' of size 1
- **connected input**
an integer. It must be between 1 and the number of inputs.

Properties : Type 'vec' of size 1

Default properties

- **always active:** yes
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
 - port 2 : size [-1,1] / type 1
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *switchn*

Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SWITCH_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/switchn.c (Type 2)

Authors

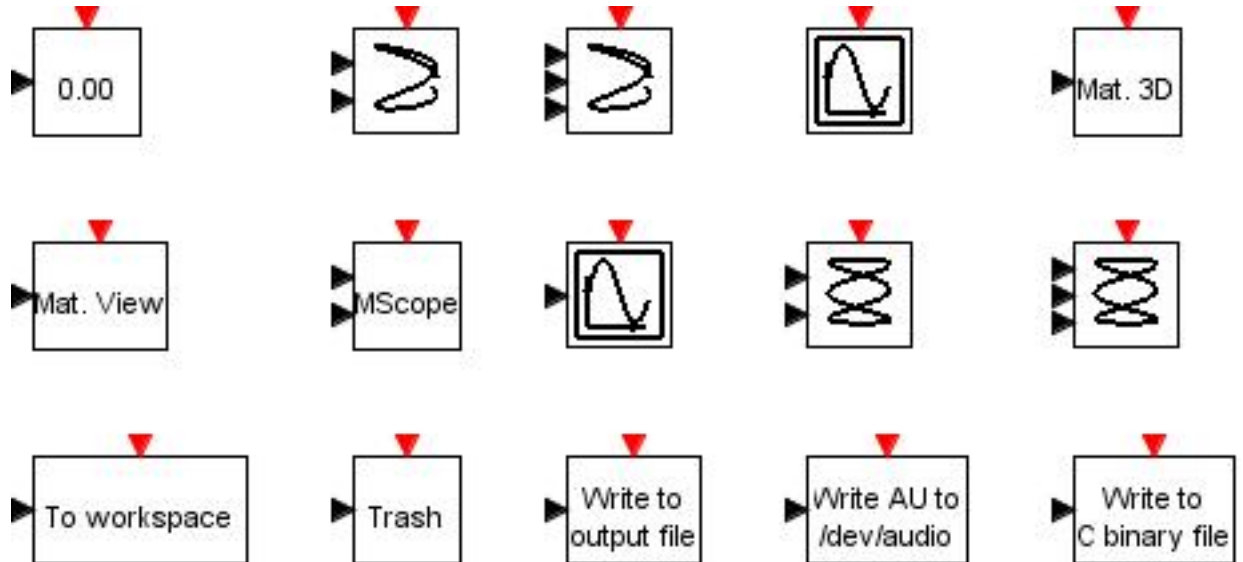
Ramine Nikoukhah - INRIA

17. Sinks palette

Name

Sinks_pal — Sinks palette

Block Screenshot



Module

- xcOS

Description

In Sinks palette, you can find a variety of blocks used to display (Scope) and write data during simulation and also some output ports used in superblocks.

The blocks of that palette doesn't have regular output ports.

Blocks

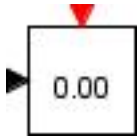
- AFFICH_m - Display
- CANIMXY - $y=f(x)$ animated viewer
- CANIMXY3D - $z=f(x,y)$ animated viewer
- CFSCOPE - Floating point scope
- CLKOUTV_f - Output activation port
- CMAT3D - Matrix z values 3D viewer
- CMATVIEW - Matrix Colormapped viewer
- CMSCOPE - Multi display scope
- CSCCOPE - Single Display Scope
- CSCOPXY - $y=f(x)$ permanent viewer
- CSCOPXY3D - $z=f(x,y)$ permanent viewer
- ENDBLK - END block viewer

- END_c - END_c block viewer
- HALT_f - Halt block viewer
- OUTIMPL_f - Output implicit port viewer
- OUT_f - Output port viewer
- TOWS_c - Data to Scilab workspace
- TRASH_f — Trash block
- WFILE_f — Write to file
- WRITEAU_f — Write AU sound file
- Write binary data

Name

AFFICH_m — Display

Block Screenshot



Contents

- Display
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Sinks palette

Description

This block displays the value of its unique input inside the block (in the diagram) during simulation.

Dialog box

	Set parameters
Input Size	[1,1]
Font number	1
Font size	1
Color	1
Total number of digits	5
Number of rational part digits	1
Block inherits (1) or not (0)	0

Dismiss

OK

- **Input Size**

Explicitly define the size of the input port.

Properties : Type 'mat' of size [1,2].

- **Font number**

integer, the selected font number (see xset).

Properties : Type 'vec' of size 1

- **Font size**

integer, the selected font size (set xset)

Properties : Type 'vec' of size 1

- **Color**

integer, the selected color for the text (see xset)

Properties : Type 'vec' of size 1

- **Total number of digits**

an integer greater than 3, the maximum number of digits used to represent the number (sign, integer part and rational part)

Properties : Type 'vec' of size 1

- **Number of rational part digits**

n integer greater than or equal 0, the number of digits used to represent the rational part

Properties : Type 'vec' of size 1

- **Block inherits**

Options to choose event inheritance from regular input or from explicit event input (0).

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *affich2*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/AFFICH_m.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/affich2.f (Type 0)

Name

CANIMXY — $y=f(x)$ animated viewer

Block Screenshot



Contents

- $y=f(x)$ animated viewer
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks palette

Description

This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port. When a point is drawn on screen it stays until the buffer length is reached. This scope is useful to make simple animations.

Dialog box

Set Scope parameters	
Number of Curves	1
color (>0) or mark (<0)	-4
line or mark size	1
Output window number (1 for automatic)	1
Output window position	0
Output window sizes	0
Xmin	-15
Xmax	15
Ymin	15
Ymax	15
Buffer size	2
Dismiss OK	

- **Number of Curves**

Set the number of curves.

Properties : Type 'vec' of size 1

- **color**

≥ 0 < 0

an integer. It is the color number (≥ 0) or marker type (< 0) used to draw the evolution of the input port signal. See **exset()** for color (dash type) definitions.

Properties : Type 'vec' of size 1

- **line or mark size**

an integer.

Properties : Type 'vec' of size 1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Properties : Type 'vec' of size -1

- **Xmin**

Minimum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Xmax**

Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymin**

Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymax**

Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Buffer size**

An integer value. In order to minimize the number of graphics outputs, data may be buffered.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *canimxy*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CANIMXY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/canimxy.c (Type 4)

See also

- CANIMXY3D - $z=f(x,y)$ animated viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CANIMXY3D — $z=f(x,y)$ animated viewer

Block Screenshot



Contents

- $z=f(x,y)$ animated viewer
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks palette

Description

This block realizes the visualization of the evolution of the three regular input signals by drawing the third input as a function of the two others at instants of events on the event input port. When a point is drawn on screen it stays until the buffer length is reached. This scope is useful to make simple animations.

Dialog box

Set Scope parameters	
Number of curves	1
color (>0) or mark (<0)	1 2 3 4 5 6 7 13
line or mark size	1 1 1 1 1 1 1
Output window number (1 for automatic)	1
Output window position	0
Output window sizes	0
Xmin and Xmax	-15 15
Ymin and Ymax	-15 15
Zmin and Zmax	15 15
Alpha and Theta	50 200
Buffer size	2
Dismiss OK	

- **Number of Curves**

Set the number of curves.

Properties : Type 'vec' of size -1

- **color**

≥ 0 < 0

an integer. It is the color number () or marker type () used to draw the evolution of the input port signal. See **set()** for color (dash type) definitions.

Properties : Type 'vec' of size -1

- **line or mark size**

an integer.

Properties : Type 'vec' of size -1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

Properties : Type 'vec' of size -1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Properties : Type 'vec' of size -1

- **Xmin and Xmax**

Minimum and Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Ymin and Ymax**

Minimum and Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Zmin and Zmax**

Minimum and Maximum values of the third input; used to set up the Z-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Alpha and Theta**

Set Alpha and Theta for the 3D view.

Properties : Type 'vec' of size -1

- **Buffer size**

An integer value. In order to minimize the number of graphics outputs, data may be buffered.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
 - port 3 : size [1,1] / type 1

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *canimxy3d*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CANIMXY3D.sci

Computational function

- SCI/modules/scicos_blocks/src/c/canimxy3d.c (Type 4)

See also

- CANIMXY - $y=f(x)$ animated viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CFSCOPE — Floating point scope

Block Screenshot



Contents

- Floating point scope
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sinks palette

Description

This is a floating scope block.

Dialog box

Set Scope parameters	
Color (>0) or mark (<0) vector (8 entries)	1 3 5 7 9 11 13 15
Output window number (-1 for automatic)	-1
Output window position	[]
Output window sizes	[600;400]
Ymin	-15
Ymax	15
Refresh period	30
Buffer size	7
Links to view	1
Dismiss OK	

- **Color**

> 0 < 0

a vector of integers. The i-th element is the color number () or dash type () used to draw the evolution of the i-th input port signal. See **plot2d** for color (dash type) definitions.

Properties : Type 'vec' of size 8

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used (**1**), Xcos define the output window number.

Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Ymin**

Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Ymax**

Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Refresh period**

Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

Properties : Type 'vec' of size

- **Buffer size**

To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

Properties : Type 'vec' of size 1

- **Links to view**

This parameter allows you to display the output of specified link.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cfscope*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CFSCOPE.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cfscope.c (Type 4)

Authors

- **Ramine Nikoukhah** INRIA

- **Benoit Bayol**

Name

CMAT3D — Matrix z values 3D viewer

Block Screenshot



Contents

- Matrix z values 3D viewer
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks palette

Description

CMAT3D is a scope that shows matrix values as z values on a xy grid.

Dialog box

Set Scope parameters	
Bounds Vector X (1 for standard)	1
Bounds Vector Y (-1 for standard)	-1
ColorMap	jetcolormap(25)
Zmin	0
Zmax	100
DismissOK	

- **Bounds Vector X (-1 for standard)**

If you let -1 the x ticks would be between 0 and 1 else you can put your own vector.

Properties : Type 'vec' of size -1.

- **Bounds Vector Y (-1 for standard)**

If you let -1 the x ticks would be between 0 and 1 else you can put your own vector.

Properties : Type 'vec' of size -1.

- **ColorMap**

The colormap is a range color linked to the window output of the scope. You can put a jetcolormap or hotcolormap or graycolormap or your own (see colormap help).



Properties : Must be a mx3 matrix and m = 3

- **Zmin**

Minimum value in Z values

Properties : Type 'vec' of size 1.

- **Zmax**

Maximum values in Z values

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmat3d*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMAT3D.sci

Computational function

- `SCI/modules/scicos_blocks/src/c/cmat3d.c` (Type 4)

See also

- `CMATVIEW` - Matrix Colormapped viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CMATVIEW — Matrix Colormapped viewer

Block Screenshot



Contents

- Matrix Colormapped viewer
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

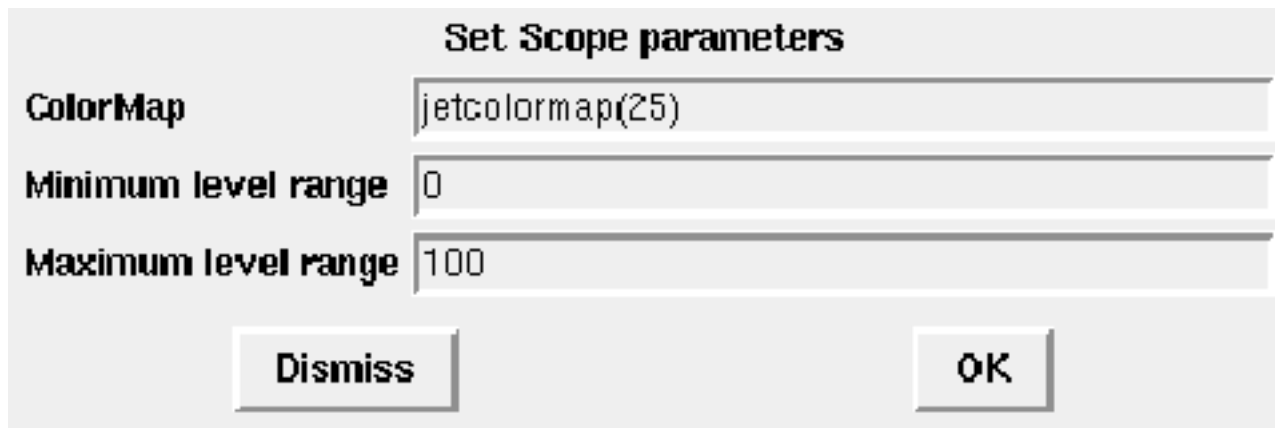
Palette

- Sinks palette

Description

CMATVIEW is a scope that shows matrix values on a colormapped grid.

Dialog box



- ColorMap

The colormap is a range color linked to the window output of the scope. You can put a jetcolormap or hotcolormap or graycolormap or your own (see colormap help).



Properties : Must be a mx3 matrix and m = 3

- **Minimum level range**

The minimum level range is the minimum value who comes in the regular input port. It would be linked to the 'cold value' of the colormap.

Properties : A scalar

- **Maximum level range**

The maximum level range is the maximum value who comes in the regular input port. It would be linked to the 'hot value' of the colormap.

Properties : A scalar

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,-2] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmatview*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMATVIEW.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cmatview.c (Type 4)

See also

- CMAT3D - Matrix z values 3D viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CMSCOPE — Multi display scope

Block Screenshot



Contents

- Multi display scope
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sinks palette

Description

When you start a simulation, Xcos open Scope windows. The Scope block displays its input with respect to simulation time. The Scope block can have multiple axes (one per port); all axes have a common time range with independent y-axes. The Scope allows you to adjust the amount of time and the range of input values displayed.

Dialog box

Set Scope parameters	
Input ports sizes	1 1
Drawing colors (>0) or mark (<0)	1 3 5 7 9 11 13 15
Output window number (-1 for automatic)	-1
Output window position	[]
Output window sizes	[]
Ymin vector	-1 -5
Ymax vector	1 5
Refresh period	30 30
Buffer size	20
Accept inherited events D/M	0
Name of Scope (label&id)	
Dismiss OK	

- **Input ports sizes**

It allows multiple input ports.

Properties : Type 'vec' of size -1

- **Drawing colors**

> 0 < 0

a vector of integers. The i-th element is the color number () or dash type () used to draw the evolution of the i-th input port signal. See **plot2d** for color (dash type) definitions.

Properties : Type 'vec' of size -1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used (**1**), Xcos define the output window number.

Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size size -1

- **Ymin vector**

Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size size(%1, '*')

- **Ymax vector**

Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size size(%1, '*')

- **Refresh period**

Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

Properties : Type 'vec' of size size(%1, '*')

- **Buffer size**

To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

Properties : Type 'vec' of size 1

- **Accept herited events 0/1**

if **0CSCOPE_f** draws a new point only when an event occurs on its event input port. if **1CSCOPE_f** draws a new point when an event occurs on its event input port and when it's regular input changes due to an event on an other upstream block (herited events).

Properties : Type 'vec' of size 1

- **Name of Scope**

Name/label of the block.

Properties : Type 'str' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmscope*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMSCOPE.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cmscope.c (Type 4)

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CSCOPE — Single Display Scope

Block Screenshot



Contents

- Single Display Scope
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks palette

Description

The Scope block displays its input with respect to simulation time. Both axes have a common range. The Scope allows you to adjust the amount of time and the range of input values displayed.

Dialog box

Set Scope parameters	
Color (>0) or mark (<0) vector (8 entries)	1 3 5 7 9 11 -3 15
Output window number (-1 for automatic)	-1
Output window position	[]
Output window sizes	[600;400]
Ymin	-15
Ymax	15
Refresh period	30
Buffer size	20
Accept inherited events DM	0
Name of Scope (label&id)	
Dismiss OK	

- **Color**

$\begin{matrix} > 0 & & < 0 \end{matrix}$

a vector of integers. The i-th element is the color number () or dash type () used to draw the evolution of the i-th input port signal. See **plot2d** for color (dash type) definitions.

Properties : Type 'vec' of size 8

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used (**1**), Scicos define the output window number.

Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size 1

- **Output window sizes**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Ymin**

Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymax**

Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Refresh period**

Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

Properties : Type 'vec' of size 1

- **Buffer size**

To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

Properties : Type 'vec' of size 1

- **Accept herited events 0/1**

if 0 **CSCOPE_f** draws a new point only when an event occurs on its event input port. If 1 **CSCOPE_f** draws a new point when an event occurs on its event input port and when it's regular input changes due to an event on an other upstream block (herited events).

Properties : Type 'vec' of size 1

- **Name of Scope**

Name/label of the block.

Properties : Type 'str' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscope*

Interfacing function

- `SCI/modules/scicos_blocks/macros/Sinks/CSCOPE.sci`

Computational function

- `SCI/modules/scicos_blocks/src/c/cscope.c` (Type 4)

See also

- `CMSCOPE` - Multi display scope

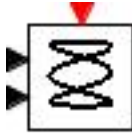
Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CSCOPXY — $y=f(x)$ permanent viewer

Block Screenshot



Contents

- $y=f(x)$ permanent viewer
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “See also”
 - the section called “Authors”

Palette

- Sinks palette

Description

This block realizes the visualization of the evolution of the two regular inputs signals by drawing the second input as a function of the first at instants of events on the event input port. When a point is drawn on screen it stays until the simulation is finished.

Dialog box

Set Scope parameters	
Number of Curves	1
color (>0) or mark (<0)	4
line or mark size	1
Output window number (1 for automatic)	1
Output window position	[0]
Output window sizes	[600;400]
Xmin	-15
Xmax	15
Ymin	15
Ymax	15
Buffer size	2
Dismiss OK	

- **Number of Curves**

Set the number of curves. Properties : Type 'vec' of size 1

- **color**

> 0 < 0

an integer. It is the color number () or dash type () used to draw the evolution of the input port signal. See **plot2d** for color (dash type) definitions.

Properties : Type 'vec' of size 1

- **line or mark size**

an integer.

Properties : Type 'vec' of size 1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Properties : Type 'vec' of size -1

- **Xmin**

Minimum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type " of size

- **Xmax**

Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymin**

Minimum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymax**

Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Buffer size**

To improve efficiency it is possible to buffer the input data. The drawing is only done after each Buffer size call to the block.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscopy*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CSCOPXY.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cscopy.c (Type 4)

See also

- CSCOPXY3D - $z=f(x,y)$ permanent viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

CSCOPXY3D — $z=f(x,y)$ permanent viewer

Block Screenshot



Contents

- $z=f(x,y)$ permanent viewer
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks palette

Description

This block realizes the visualization of the evolution of the three regular inputs signals by drawing the third input as a function of the two others at instants of events on the event input port. When a point is drawn on screen it stays until the simulation is finished.

Dialog box

Set Scope parameters	
Number of curves	1
color (>0) or mark (<0)	1 2 3 4 5 6 7 13
Line or Mark Size	1 1 1 1 1 1 1
Output window number (1 for automatic)	1
Output window position	0
Output window sizes	[600;400]
Xmin and Xmax	-15 15
Ymin and Ymax	-15 15
Zmin and Zmax	15 15
Alpha and Theta	50 200
Buffer size	2
Dismiss OK	

- **Number of Curves**

Set the number of curves.

Properties : Type 'vec' of size -1

- **color**

≥ 0 < 0

an integer. It is the color number () or marker type () used to draw the evolution of the input port signal. See **set()** for color (dash type) definitions.

Properties : Type 'vec' of size -1

- **line or mark size**

an integer.

Properties : Type 'vec' of size -1

- **Output window number**

The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

Properties : Type 'vec' of size -1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Properties : Type 'vec' of size -1

- **Xmin and Xmax**

Minimum and Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Ymin and Ymax**

Minimum and Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Zmin and Zmax**

Minimum and Maximum values of the third input; used to set up the Z-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Alpha and Theta**

Set Alpha and Theta for the 3D view.

Properties : Type 'vec' of size -1

- **Buffer size**

An integer value. In order to minimize the number of graphics outputs, data may be buffered.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
 - port 3 : size [1,1] / type 1

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscopy3d*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CSCOPXY3D.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cscopy3d.c (Type 4)

See also

- CSCOPXY - $y=f(x)$ permanent viewer

Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

Name

ENDBLK — END block

Block Screenshot



Contents

- END block
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

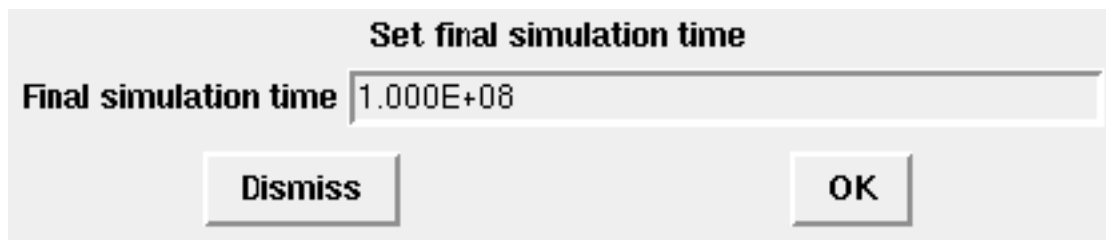
Palette

- Sinks - Sinks palette

Description

That block can be used to set the final time of the simulation. When that block is truly parametrized then the simulator will jump to the 'final integration time' defined in the Setup item of the simulate Menu from the time defined by the parameter 'Final simulation time' of the dialog box. That parameter can be a numerical value or a symbolic variable defined in the scicos context.

Dialog box



- **Final simulation time**

Set the final time of the simulation.

When simulator reaches that value then the current time will jump to the final integration time.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/ENDBLK.sci

Compiled Super Block content



See also

- END_c - END_c block (Scicos Block)

Authors

Alan Layec - INRIA

Name

END_c — END_c block

Block Screenshot



Contents

- END_c block
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sinks - Sinks palette

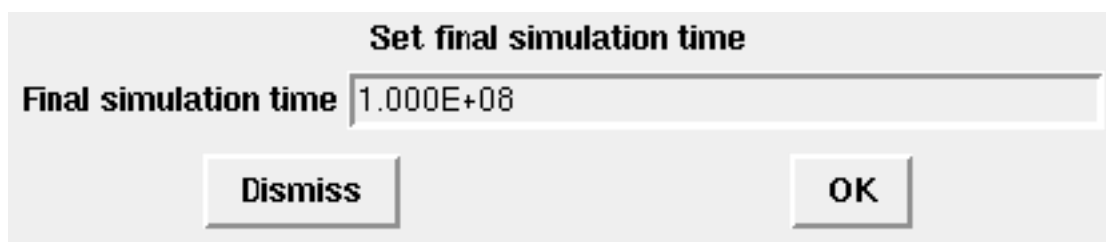
Description

That block increases the current time to the final integration time of the simulation when it is activated :

$$T_f = T_{cur}$$

with **T_{cur}** the activation date of the block and **T_f** the final integration time defined in the in the Setup item of the simulate Menu (scs_m.props.tf).

Dialog box



- Final simulation time

That parameter is a date for an initial output event. By using a feed back from the event output port to the event input port, then that block can himself end the simulation at the time defined by this parameter.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *scicosexit*

Interfacing function

- SCI/modules/scicos_blocks/macros/Events/END_c.sci

Computational function

- SCI/modules/scicos_blocks/src/c/scicosexit.c (Type 4)

See also

- ENDBLK - END block (Scicos Block)

Authors

Alan Layec - INRIA

Name

TOWS_c — Data to Scilab workspace

Block Screenshot



Contents

- Data to Scilab workspace
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sinks palette

Description

That block is used to put simulated data in the scilab workspace.

Each sampling time, both dates and values of input are recorded.

Dialog box

Set Scicos buffer block	
Size of buffer	128
Scilab variable name	A
Inherit (no:0, yes:1)	0
DismissOK	

- **Size of buffer**

Set the size of the input buffer. That gives the total number of samples recorded during the simulation.

That buffer is a circulate buffer.

Properties : Type 'vec' of size 1.

- **Scilab variable name**

Set the name of the Scilab variable. This must be a valid variable name.

The simulation must be finished to retrieve that variable in the Scilab workspace.

Properties : Type 'str' of size 1.

- **Inherit (no:0, yes:1)**

Options to choose event inheritance from regular input or from explicit event input (0).

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type -1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tows_c*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/TOWS_c.sci

Computational function

- SCI/modules/scicos_blocks/src/c/tows_c.c (Type 4)

Authors

Alan Layec - INRIA

Name

TRASH_f — Trash block

Block Screenshot



Contents

- Trash block
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- Sinks palette

Description

That block is an end-block. It do nothing.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *trash*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/TRASH_f.sci

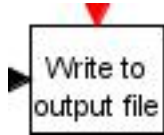
Computational function

- SCI/modules/scicos_blocks/src/fortran/trash.f (Type 0)

Name

WFILE_f — Write to file

Block Screenshot



Contents

- Write to file
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sinks palette

Description

This block allows user to save data in a file, in formatted and binary mode. Each call to the block corresponds to a record in the file. Each record has the following form: where is the value of time when block is called and is the i th input value.

This block does not manage UTF filename.

The pair block is .

Dialog box

Set WFILE block parameters	
Input size	1
Output file name	foo
Output Format	(7(e10.3,1x))
Buffer size	2
Dismiss OK	

- **Input size**

a scalar. This fixes the input size.

Properties : Type 'vec' of size 1.

- **Output file name**

a character string defining the path of the file.

Properties : Type 'str' of size 1.

- **Output Format**

a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must begin by a left parenthesis and end by a right parenthesis. exam-

(e10.3)

ple: .

Properties : Type 'str' of size 1.

- **Buffer size**

To improve efficiency it is possible to buffer the input data. Write on the file is only done after each **Buffer size** call to the block.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writef*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WFILE_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/writef.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

WRITEAU_f — Write AU sound file

Block Screenshot



Contents

- Write AU sound file
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

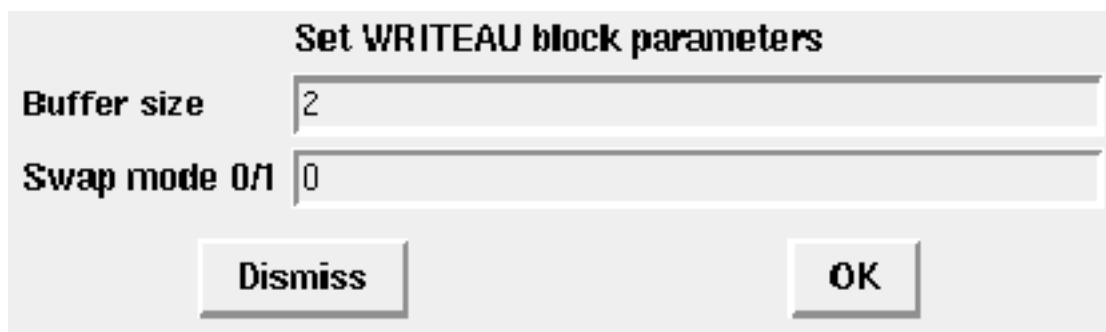
Palette

- Sinks palette

Description

This block writes a sound file specified by the string *.au file. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are clipped prior to writing. auwrite supports multichannel data for 8-bit mu-law and 8- and 16-bit linear formats.

Dialog box



- **Buffer size**

To improve efficiency it is possible to buffer the input data. read on the file is only done after each Buffer size call to the block.

Properties : Type 'vec' of size 1

- **Swap mode 0/1**

With **Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swapped if necessary to match the IEEE format of the processor. If **Swap mode=0** then automatic bytes swap is disabled.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writeau*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WRITEAU_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/writeau.c (Type 2)

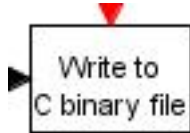
Authors

Ramine Nikoukhah - INRIA

Name

WRITEC_f — Write binary data

Block Screenshot



Contents

- Write binary data
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Sinks palette

Description

This block allows user to write data in a C binary file.

Dialog box

Set WRITEC block parameters	
Input size	1
Output file name	foo
Output Format	c
Buffer size	2
Swap mode 0/1	0
DismissOK	

- Input size

a scalar, the size of the input.

Properties : Type 'vec' of size 1

- **Output file name**

a character string defining the output file name.

Properties : Type 'str' of size 1

- **Output Format**

a character string defining the format to use.

Properties : Type 'str' of size 1

- **Buffer size**

To improve efficiency it is possible to buffer the input data. read on the file is only done after each **Buffer size** call to the block.

Properties : Type 'vec' of size 1

- **Swap mode 0/1**

With **Swap mode=1** the file is supposed to be coded in ``little endian IEEE format" and data are swapped if necessary to match the IEEE format of the processor. If **Swap mode=0** then automatic bytes swap is disabled.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - **port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writec*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WRITEC_f.sci

Computational function

- `SCI/modules/scicos_blocks/src/c/wrotec.c` (Type 2)

Authors

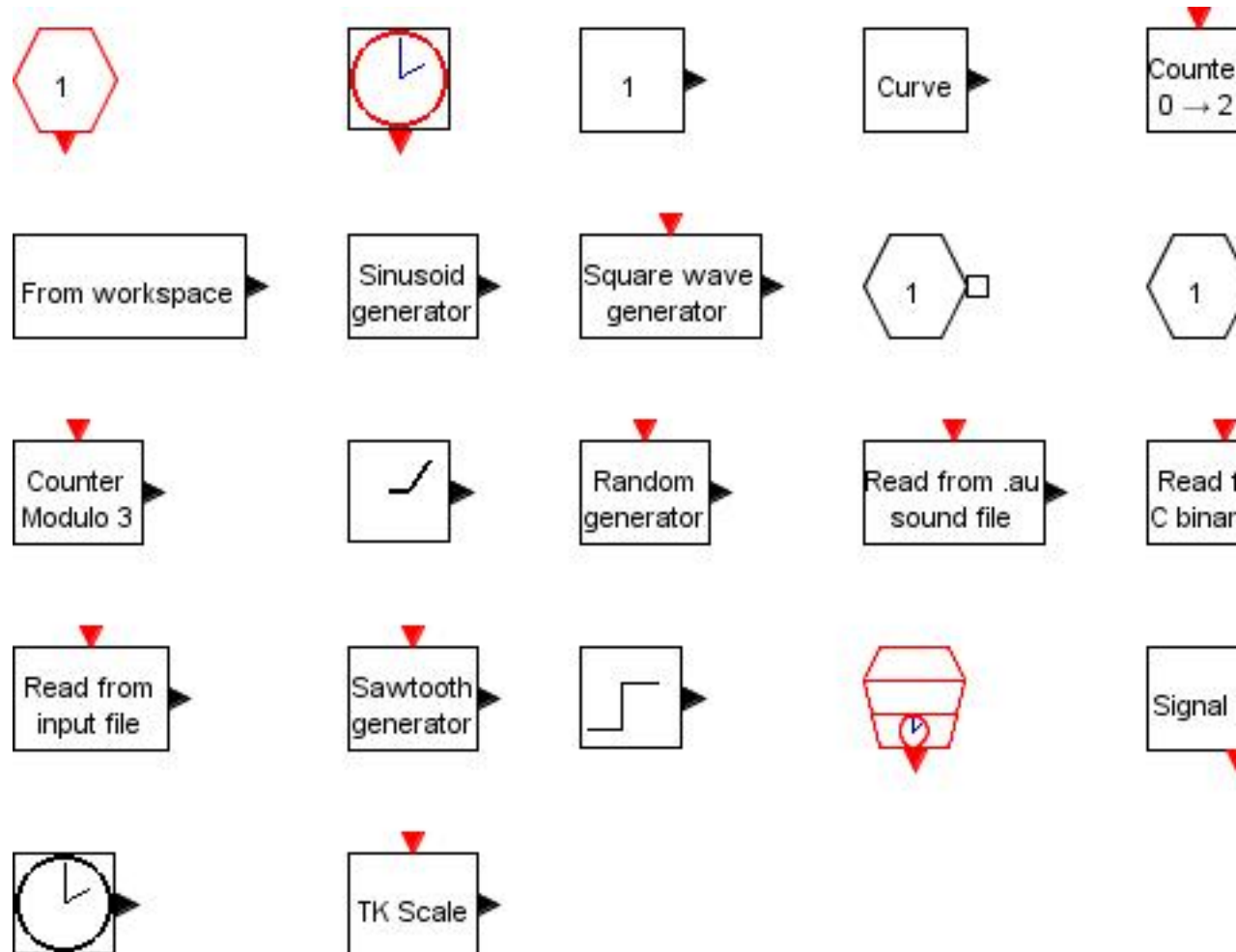
Ramine Nikoukhah - INRIA

18. Sources palette

Name

Sources_pal — Sources palette

Block Screenshot



Module

- xcoss

Description

Most of blocks of the Source palette can be viewed as data generators. That palette also contains blocks to read data from files and input ports used in superblocs.

The blocks of that palette doesn't have regular input ports.

Blocks

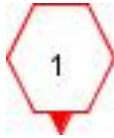
- CLKINV_f - Input activation port
- CLOCK_c - Activation clock
- CONST_m - Constant
- Counter - Counter

- CURV_f - Curve
- FROMWSB - Data from Scilab workspace to Xcos
- GENSIN_f - Sin generator
- GENSQR_f - Square wave generator
- IN_f - Input Port
- INIMPL_f - Input implicit port
- Modulo_Count - Modulo counter
- RAMP - Ramp
- RAND_m - Random generator
- READAU_f - Read AU sound file
- READC_f - Read binary data
- RFILE_f - Read from file
- SampleCLK - Sample Time Clock
- SAWTOOTH_f - Sawtooth generator
- Sigbuilder - Signal creator/generator
- STEP_FUNCTION - Step function generator
- TIME_f - Time
- TKSCALE - Adjust constant value with a tk widget

Name

CLKINV_f — Input activation port

Block Screenshot



Contents

- Input activation port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

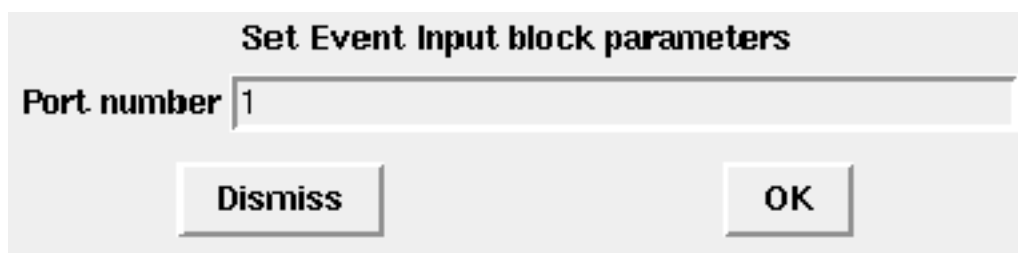
Palette

- Sources palette

Description

This block must only be used inside Xcos Super Blocks to represent an event input port. In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

Dialog box



- **Port number**
an integer defining the port number.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *input*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CLKINV_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

CLOCK_c — Activation clock

Block Screenshot



Contents

- Activation clock
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

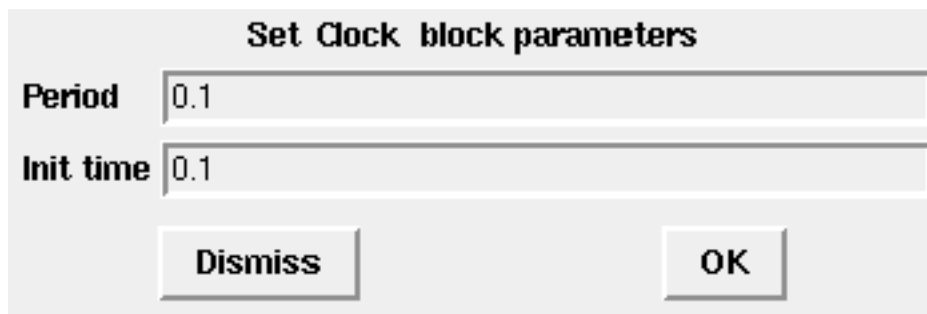
- Sources palette

Description

This block is a Super Block constructed by feeding back the output of the block into its input event port.

The unique output of this block generates a regular train of events that's scheduled by the dialog parameter **Period**.

Dialog box



Set Clock block parameters

Period

Init time

- **Period**
scalar.
One over the frequency of the clock.

Period is the time that separates two output events.

Properties : Type 'vec' of size 1.

- **Init time**

scalar.

Starting date.

If negative the clock never starts.

Properties : Type 'vec' of size 1.

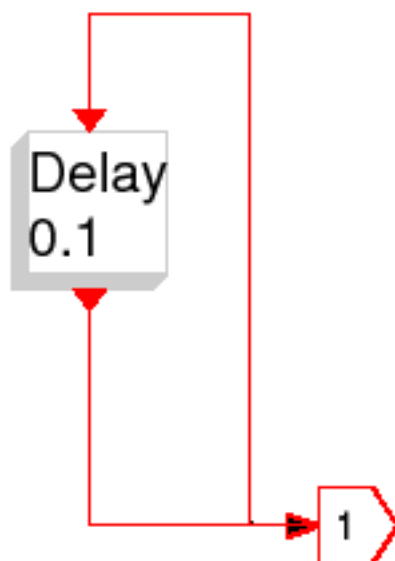
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CLOCK_c.sci

Compiled Super Block content



See also

- EVTDLY_c - Event delay

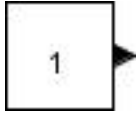
Authors

Alan Layec - INRIA

Name

CONST_m — Constant

Block Screenshot



Contents

- Constant
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This block is a constant value generator.

Dialog box



- **Constant**

A matrix of any type.

The size of the matrix gives the size of the regular output port.

The constant(i,j) value is the component(i,j) value of the output port.

From this value the block inherits its data type.

Properties : Type 'mat' of size [-1,-2].

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cstblk4*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CONST_m.sci

Computational function

- SCI/modules/scicos_blocks/src/c/cstblk4.c (Type 4)

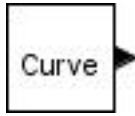
Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

Name

CURV_f — Curve

Block Screenshot



Contents

- Curve
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sources palette

Description

This block defines a tabulated function of time. Between mesh points block performs a linear interpolation. Outside tabulation block outputs last tabulated value. User may define the tabulation of the function using a curve editor.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - **port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *intplt*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CURV_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/intplt.f (Type 0)

See also

- Sigbuilder - Signal creator/generator

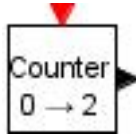
Authors

Ramine Nikoukhah - INRIA

Name

Counter — Counter

Block Screenshot



Contents

- Counter
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “See also”
- the section called “Authors”

Palette

- Sources palette

Description

This block counts from minimal to maximal or from maximal to minimal depending on the choice of the rule.

Dialog box

Set Counter block parameters

Minimum	0
Maximum	2
Rule (1=Increment 2=Decrement)	1

Dismiss

OK

- **Minimum**
The lowest number of the counter.

Properties : Type 'vec' of size 1.

- **Maximum**

The highest number of the counter.

Properties : Type 'vec' of size 1.

- **Rule (1=Increment 2=Decrement)**

The rule of counting. If it is 1 then the counter counts from the lower number to the higher number. the count in this case is increasing. otherwise, if the rule is equal to 2 the counter will decrease and it will count from the higher number to the lower one.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *counter*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Counter.sci

Computational function

- SCI/modules/scicos_blocks/src/c/counter.c (Type 4)

See also

- Modulo_Count - Modulo counter (xcos Block)

Authors

Fady NASSIF - INRIA

Name

FROMWSB — Data from Scilab workspace to Xcos

Block Screenshot



Contents

- Data from Scilab workspace to Xcos
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “See also”
- the section called “Authors”

Palette

- Sources palette

Description

That block is used to get data defined in the Scilab workspace and use them in Xcos. Data should have "time" and "values" fields.

Dialog box

A dialog box titled 'Set From_Workspace block parameters'. It contains four input fields with labels: 'Variable name' (value: 'v'), 'Interpolation Method' (value: '1'), 'Enable zero crossing(0:No, 1:Yes)?' (value: '1'), and 'Output at end(0:Zero, 1:Hold, 2:Repeat)' (value: '0'). At the bottom, there are two buttons: 'Dismiss' and 'OK'.

- **Variable name:** This variable is defined in Scilab and should be a structure with two fields, i.e., a "time" field of size (Nx1) and a "values" field of size (NxM). "time" is a column vector of size

Nx1 and "values" is a matrix of size "N*M". "time" field can only be of Real type, whereas the "values" field can be , , , , , and .

- **Interpolation method:** Variables read by Xcos are data values read at discrete instants given by the time field. This option causes the block to interpolate at time steps for which no corresponding workspace data exists. There are four interpolation methods available.

- **0: "Zero order method".** This method generates a piecewise constant signal. i.e., for

$$t_i \leq t < t_{i+1} \quad y(t) = y_i$$

, . This method is available for all data types.

- **1: "Linear method".** This method generates a piecewise linear signal, i.e., for

$$t_i \leq t < t_{i+1} \quad y(t) = y_i + (t - t_i) \frac{y_{i+1} - y_i}{t_{i+1} - t_i}$$

, . For data types other than double and complex, the linear interpolation can be used, but the final output will be computed by casting interpolation result into the original data type.

- **2:"NATURAL method".** This cubic spline is computed by using the following conditions (con-

$$x_1, \dots, x_n$$

sidering N points): . This method is only available for Real and complex data types.

- **3:"NOT_A_KNOT method".** The cubic spline is computed by using the following conditions

$$x_1, \dots, x_n$$

(considering N points): . This method is only available for Real and complex data types.

- **Enable zero crossing(0:No, 1:Yes)?:** Enables zero crossing detection. When and interpolation methods are chosen, the output signal will be discontinuous at data time instants. These possible discontinuities may cause problem for the numerical solver. In order to perform a reliable numerical integration, the zero crossing option is used. If output of the block affects data used by the numerical solver, at discontinuous points, a discrete event is generated and the numerical solver is cold

$$t_1 \quad t_n$$

restarted. The discrete event is also generated at the and for other interpolating methods.

- **Output at end(0:Zero, 1:Hold, 2:Repeat):** This option is for selecting method for generating output after the last time point for which data is available from the workspace.

- **0 ("Zero"):** The output is set to zero.
- **1 ("Hold"):** The output is hold.
- **2 ("Repeat"):** The output is repeated from workspace.

Default properties

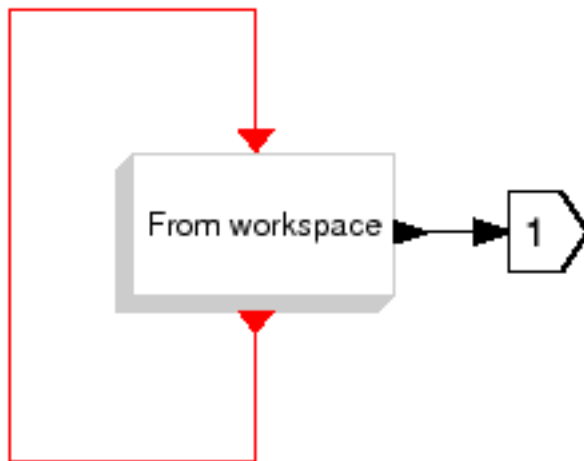
- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

- port 1 : size [-1,1] / type 1
- number/sizes of activation inputs: 0
- number/sizes of activation outputs: 0
- continuous-time state: no
- discrete-time state: no
- object discrete-time state: no
- name of computational function: *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/FROMWSB.sci

Compiled Super Block content



See also

- TOWS_c - Data to Scilab worspace

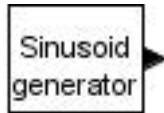
Authors

Masoud Najafi - INRIA

Name

GENSIN_f — Sin generator

Block Screenshot



Contents

- Sin generator
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

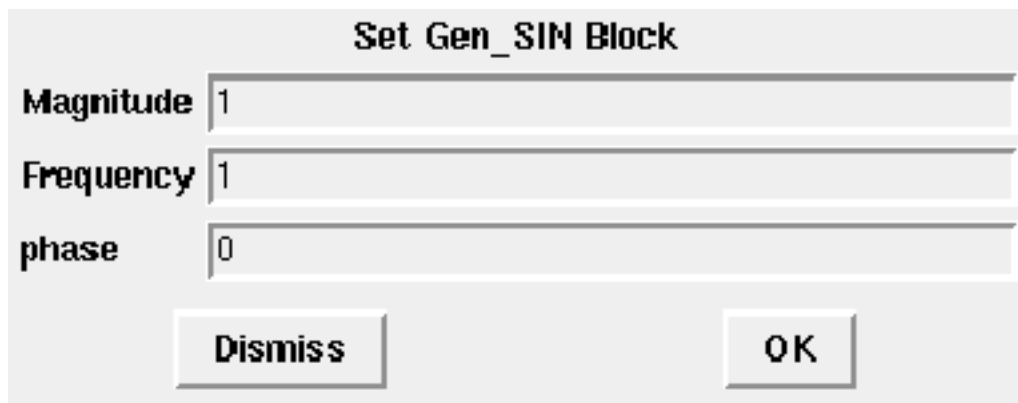
Palette

- Sources palette

Description

This block is a sine wave generator:

Dialog box



The dialog box is titled "Set Gen_SIN Block". It contains three input fields: "Magnitude" with the value "1", "Frequency" with the value "1", and "phase" with the value "0". At the bottom, there are two buttons: "Dismiss" and "OK".

- **Magnitude**
a scalar. The magnitude M.
Properties : Type 'vec' of size 1.

- **Frequency**

a scalar. The frequency F.

Properties : Type 'vec' of size 1.

- **phase**

a scalar. The phase P.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - **port 1 :** size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gensin*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/GENSIN_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/gensin.f (Type 0)

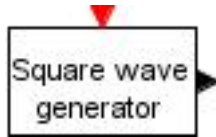
Authors

Ramine Nikoukhah - INRIA

Name

GENSQR_f — Square wave generator

Block Screenshot



Contents

- Square wave generator
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This block is a square wave generator: output takes values $-M$ and M . Every time an event is received on the input event port, the output switches from $-M$ to M , or M to $-M$.

Dialog box



- **Amplitude**

a scalar M .

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *gensqr*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/GENSQR_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/gensqr.f (Type 0)

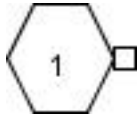
Authors

Ramine Nikoukhah - INRIA

Name

INIMPL_f — Input implicit port

Block Screenshot



Contents

- Input implicit port
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

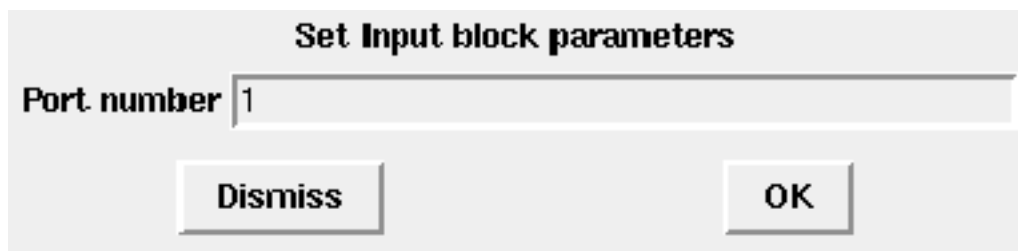
Palette

- Sources palette

Description

Inport blocks are the links from outside a system into the system.

Dialog box



- **Port number**
Specify the port number of the Inport block.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *inimpl*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/INIMPL_f.sci

Authors

Ramine Nikoukhah - INRIA

Name

Modulo_Count — Modulo counter

Block Screenshot



Contents

- Modulo counter
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

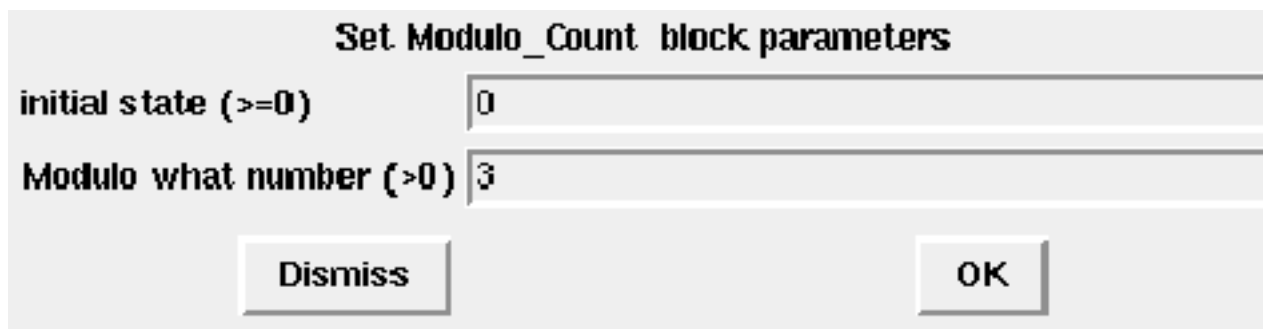
Palette

- Sources palette

Description

It is a discrete block. The block outputs a periodic scalar signal having a waveform that the user specifies.

Dialog box



The dialog box is titled "Set. Modulo_Count block parameters". It contains two input fields: "initial state (>=0)" with the value "0" and "Modulo what number (>0)" with the value "3". At the bottom, there are two buttons: "Dismiss" and "OK".

- **initial state**
A scalar initial discrete state.
Properties : Type 'vec' of size 1
- **Modulo what number**

Number of required discrete signals.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - **port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *modulo_count*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Modulo_Count.sci

Computational function

- SCI/modules/scicos_blocks/src/c/modulo_count.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

RAMP — Ramp

Block Screenshot



Contents

- Ramp
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Sources palette

Description

The Ramp block generates a signal that starts at a specified time and value and changes by a specified rate. The block's **Slope** , **Start time** and **Initial output** parameters determine the characteristics of the output signal. All must have the consistent dimensions after scalar expansion.

Dialog box

Set ramp parameters

Slope 0

Start time 0

Initial output 0

Dismiss OK

- **Slope**

The rate of change of the generated signal.

Properties : Type 'vec' of size 1.

- **Start time**

The time at which the signal begins to be generated.

Properties : Type 'vec' of size 1.

- **Initial output**

The initial value of the signal.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** yes
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ramp*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RAMP.sci

Computational function

- SCI/modules/scicos_blocks/src/c/ramp.c (Type 4)

Authors

Ramine Nikoukhah - INRIA

Name

RAND_m — Random generator

Block Screenshot



Contents

- Random generator
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This block is a random wave generator: each output component takes piecewise constant random values. Every time an event is received on the input event port, the outputs take new independent random values.

Output port size is given by the size of **A** and **B** matrices.

Dialog box

Set Random generator block parameters
flag = 0 : Uniform distribution A is min and A+B max
flag = 1 : Normal distribution A is mean and B deviation

A and B must be matrix with equal sizes

Datatype(1=real double 2=complex)

flag

A

B

SEED

- **Datatype(1=real double 2=complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **flag**

0 or 1.

0 for uniform distribution on [A,A+B].

1 for normal distribution.

Properties : Type 'vec' of size 1.

- **A**

matrix

Properties : Type 'mat' of size [-1,-2].

- **B**

matrix

Properties : Type 'mat' of size [-1,-2].

- **seed**

matrix

Seed value for a sequence of random number.

First number is for the real part and the second for the imaginary part.

Properties : Type 'mat' of size [1,2].

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - **port 1 :** size [1,1] / **type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *rndblk_m*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RAND_m.sci

Computational function

- SCI/modules/scicos_blocks/src/c/rndblk_m.c
- SCI/modules/scicos_blocks/src/c/rndblkz_m.c

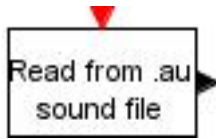
Authors

- **Fady Nassif** INRIA
- **Ramine Nikoukhah** INRIA

Name

READAU_f — Read AU sound file

Block Screenshot



Contents

- Read AU sound file
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

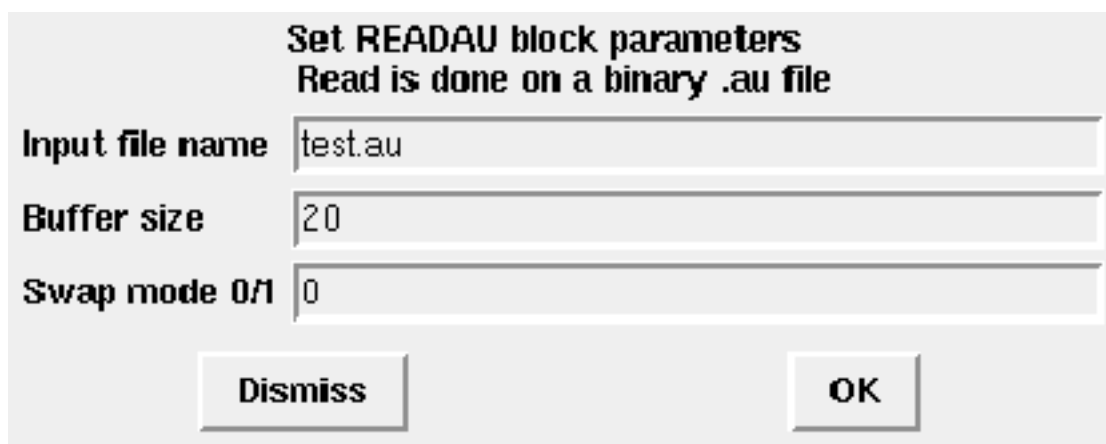
- Sources palette

Description

Loads a sound file specified by the string *.au file, returning the sampled data in y. The .au extension is appended if no extension is given. Amplitude values are in the range [-1,+1]. auread supports multichannel data in the following formats:

- 8-bit mu-law
- 8-, 16-, and 32-bit linear
- Floating-point

Dialog box

A dialog box titled "Set READAU block parameters" with the subtitle "Read is done on a binary .au file". It contains three input fields: "Input file name" with the value "test.au", "Buffer size" with the value "20", and "Swap mode 0/1" with the value "0". At the bottom, there are two buttons: "Dismiss" and "OK".

Set READAU block parameters	
Read is done on a binary .au file	
Input file name	test.au
Buffer size	20
Swap mode 0/1	0
DismissOK	

- **Input file name**

a character string defining the path of the file.

Properties : Type 'str' of size 1

- **Buffer size**

To improve efficiency it is possible to buffer the input data. Read on the file is only done after each Buffer size call to the block.

Properties : Type 'vec' of size 1

- **Swap mode 0/1**

With **Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swapped if necessary to match the IEEE format of the processor. If **Swap mode=0** then automatic bytes swap is disabled.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readau*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/READAU_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/readau.c (Type 2)

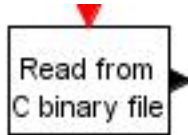
Authors

Ramine Nikoukhah - INRIA

Name

READC_f — Read binary data

Block Screenshot



Contents

- Read binary data
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This block allows user to read data in a C file. **Output record selection** and **Time record Selection** allows the user to select data among file records. Each call to the block advance one record in the file.

Dialog box

Set READC block parameters
Read is done on a binary file

Time record selection	
Outputs record selection	1
Input file name	foo
Input Format	d
Record size	1
Buffer size (in records)	20
Initial record index	1
Swap mode 0/1	0

- **Time record selection**

an empty matrix or a positive integer.

If an integer i is given the i th element of the read record is assumed to be the date of the output event.

If empty no output event exists.

Properties : Type 'vec' of size -1

- **Outputs record selection**

a vector of positive integer.

$[k_1, \dots, k_n]$ k_i , The k_i th element of the read record gives the value of i th output.

Properties : Type 'vec' of size -1

- **Input file name**

a character string defining the path of the file.

Properties : Type 'str' of size 1

- **Input Format**

a character string defining the format to use.

Properties : Type 'str' of size 1

- **Record size**

The file is supposed to be formed by a sequence of data with same format.

These data are organized in a sequence of record each of them containing Record size data.

Properties : Type 'vec' of size 1

- **Buffer size**

To improve efficiency it is possible to buffer the input data. Read on the file is only done after each Buffer size call to the block.

Properties : Type 'vec' of size 1

- **Initial record index**

a scalar. This fixes the first record of the file to use.

Properties : Type 'vec' of size 1

- **Swap mode 0/1**

With **Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swapped if necessary to match the IEEE format of the processor. If **Swap mode=0** then automatic bytes swap is disabled.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readc*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/READC_f.sci

Computational function

- SCI/modules/scicos_blocks/src/c/readc.c (Type 2)

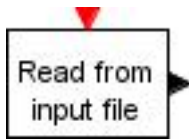
Authors

Ramine Nikoukhah - INRIA

Name

RFILE_f — Read from file

Block Screenshot



Contents

- Read from file
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Sources

Description

This block allows user to read data in a file, in formatted or binary mode. **Output record selection** and **Time record selection** allows the user to select data among file records. Each call to the block advance one record in the file.

The pair block is .

Dialog box

A dialog box titled 'Set RFILE block parameters'. Below the title, it says 'Read is done on' followed by two bullet points: '- a binary file if no format given' and '- a formatted file if a format (fortran type) is given'. There are five input fields: 'Time record selection' with a dropdown arrow, 'Outputs record selection' with the value '1', 'Input file name' with the value 'f11', 'Input Format' with the value '(7(e-0.0, "X"))', and 'Buffer size' with the value '2'. At the bottom, there are two buttons: 'Dismiss' and 'OK'.

- **Time record selection**

an empty matrix or a positive integer.

If an integer i is given the i th element of the read record is assumed to be the date of the output event. If empty no output event exists.

Properties : Type 'vec' of size -1.

- **Outputs record selection**

a vector of positive integer.

$$[k_1, \dots, k_n]$$

. The k_i th element of the read record gives the value of i th output.

Properties : Type 'vec' of size -1.

- **Input file name**

a character string defining the path of the file.

Properties : Type 'str' of size 1.

- **Input Format**

a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must began by a left parenthesis and end by a right parenthesis. exam-

$(e10.3)$

ple: .

Properties : Type 'str' of size 1.

- **Buffer size**

To improve efficiency it is possible to buffer the input data. read on the file is only done after each **Buffer size** call to the block.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readf*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RFILE_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/readf.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

SAWTOOTH_f — Sawtooth generator

Block Screenshot



Contents

- Sawtooth generator
 - the section called “Palette”
 - the section called “Description”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- Sources palette

Description

This block is a sawtooth wave generator: output is from to where and denote the times of two successive input events.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *sawtth*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/SAWTOOTH_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/sawtth.f (Type 0)

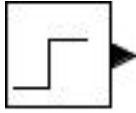
Authors

Ramine Nikoukhah - INRIA

Name

STEP_FUNCTION — Step function generator

Block Screenshot



Contents

- Step function generator
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “Authors”

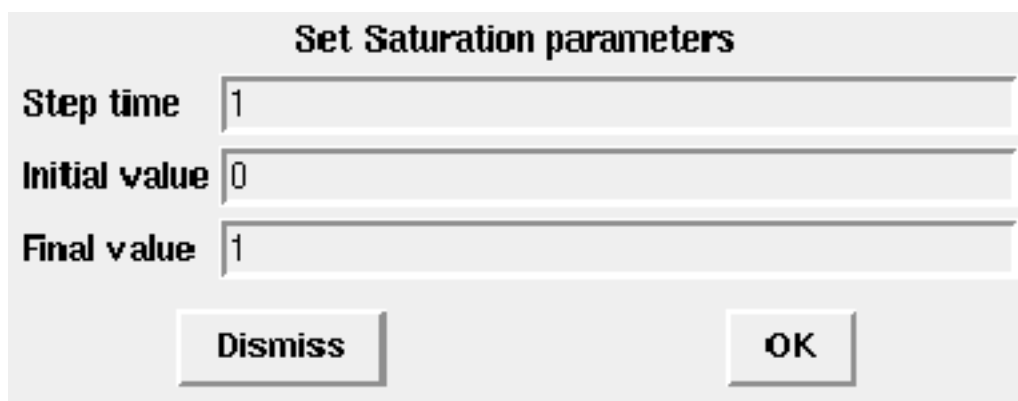
Palette

- Sources palette

Description

The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the Step time parameter value, the block's output is the Initial value parameter value. For simulation time greater than or equal to the Step time, the output is the Final value parameter value.

Dialog box



Set Saturation parameters	
Step time	1
Initial value	0
Final value	1
Dismiss OK	

- **Step time**

The time, in seconds, when the output jumps from the Initial value parameter to the Final value parameter.

Properties : Type 'vec' of size 1.

- **Initial value**

The block output until the simulation time reaches the Step time parameter.

Properties : Type 'vec' of size -1.

- **Final value**

The block output when the simulation time reaches and exceeds the Step time parameter.

Properties : Type 'vec' of size -1.

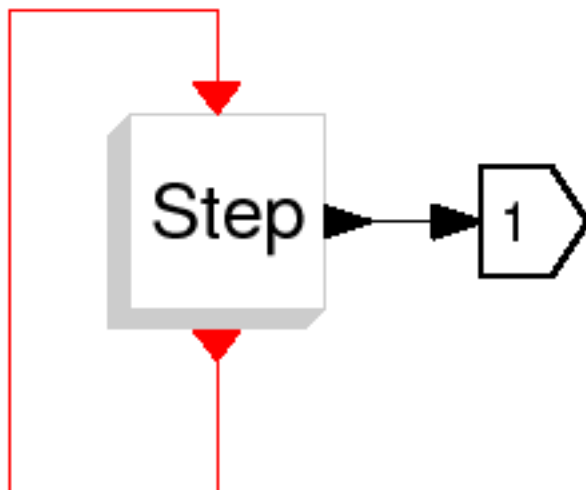
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/STEP_FUNCTION.sci

Compiled Super Block content



Authors

Ramine Nikoukhah - INRIA

Name

SampleCLK — Sample Time Clock

Block Screenshot



Contents

- Sample Time Clock
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “See also”
- the section called “Authors”

Palette

- Sources palette

Description

The difference between the SampleCLK and the CLOCK_c is that all the SampleCLK blocks in our diagram are synchronous. The synchronism is done due to two different methods of computation in the compilation phase.

The first method consists of computing a clock that is faster than all the SampleCLK connected to a counter which activate the event select block.

The clock is calculated due to the following rule.

If all the blocks have the same offset then the frequency of the clock is the gcd of the sample time, and the offset of the clock is equal to the offset.

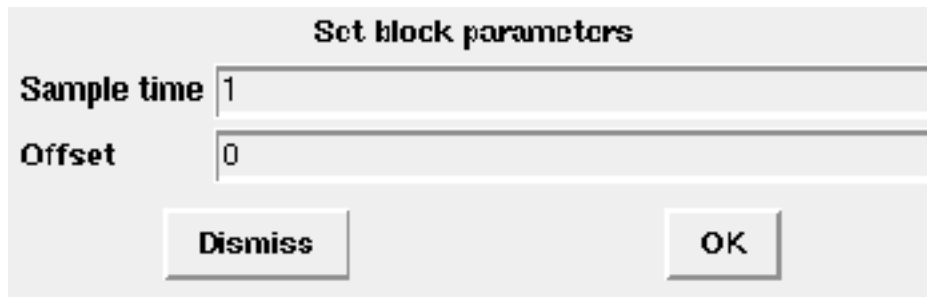
If the offsets are different, then the frequency of the clock is the gcd of the sample time and the offset, and the offset of the clock is equal to 0.

The Counter counts from one to the least common multiple of the sample time (lcm).

The number of outputs of the ESELECT_f block is equal to the lcm.

The second method uses the Multifrequency block it generates events only for specific time. Events in this method are not periodically generated as in the first one.

Dialog box



- **Sample time**
The Sample time value.
Properties : Type 'vec' of size 1.
- **Offset**
The offset value.
Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sampleclk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/SampleCLK.sci

See also

- CLOCK_c - Activation clock
- Counter - Counter
- ESELECT_f - Synchronous block Event-Select
- M_freq - Multiple Frequencies

Authors

Fady NASSIF - INRIA

Name

Sigbuilder — Signal creator/generator

Block Screenshot



Contents

- Signal creator/generator
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Compiled Super Block content
- the section called “Authors”

Palette

- Sources palette

Description

The Signal Builder block is a superblock containing a block whose output event port is connected to its input event port. This event feedback gives the possibility to generate events at discontinuous point of the signal. The generated events automatically restart the numerical solver and avoids numerical problems. The generated event is also made available to the user for possible use. Remind that if higher interpolation methods are used, the events are generated only at the beginning and at the end of the signal.

Dialog box

Spline data	
Spline Method (0..7)	3
x	[0,1,2]
y	[10,20,-30]
Periodic signal(y/n)?	y
Launch graphic window(y/n)?	n
DismissOK	

The parameters of Sigbuilder block is the same as that of block.

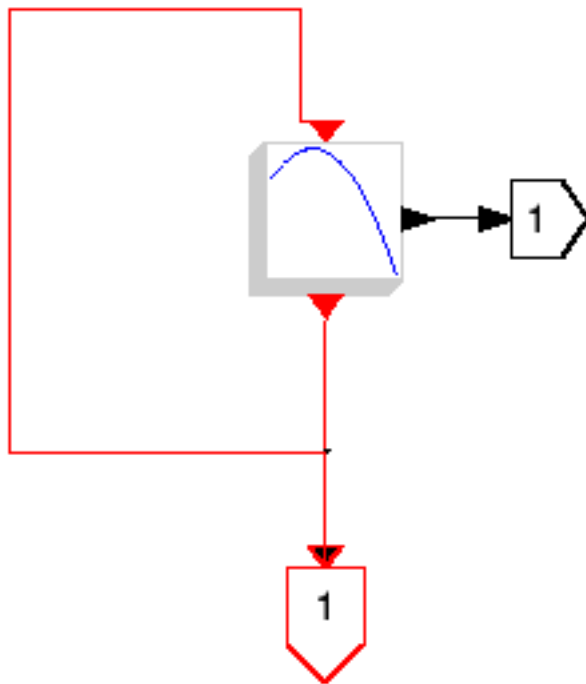
Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [-1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Sigbuilder.sci

Compiled Super Block content



Authors

- Masoud Najafi, INRIA

Name

TIME_f — Time

Block Screenshot



Contents

- Time
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This block is a time generator.

The unique regular output is the current time.

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *timblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/TIME_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/timblk.f (Type 0)

Authors

Ramine Nikoukhah - INRIA

Name

TKSCALE — Adjust constant value with a tk widget

Block Screenshot



Contents

- Adjust constant value with a tk widget
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Sources palette

Description

This source block generates a piecewise constant signal the value of which can be set interactively using a TK widget in the course of the simulation. The output value is the value set by the slider (an integer between Min value and Max value) divided by the Normalization factor. Increasing proportionally all three block parameters does not change the output range, but it does increase precision.

Dialog box

A screenshot of a dialog box titled 'Set scale block parameters'. It contains three input fields: 'Min value' with the value '-10', 'Max value' with the value '10', and 'Normalization' with the value '1'. At the bottom, there are two buttons: 'Dismiss' and 'OK'.

- Min value

An integer specifying the min value in the range of the scale.

Properties : Type 'vec' of size 1.

- **Max value**

An integer specifying the max value in the range of the scale.

Properties : Type 'vec' of size 1.

- **Normalization**

The output of the block is the integer value specified by the slider (an integer between Min value and the Max value) divided by this Normalization factor.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tkscaleblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/TKSCALE.sci

Computational function

- SCI/modules/scicos/macros/scicos_scicos/tkscaleblk.sci (Type 5)

Authors

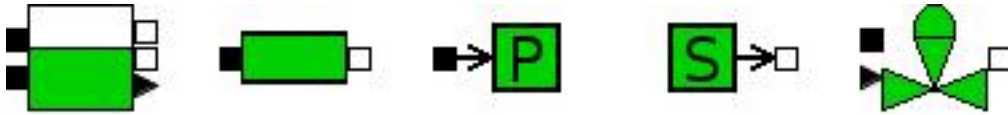
Ramine Nikoukhah - INRIA

19. Thermohydraulics palette

Name

ThermoHydraulics_pal — Thermal-Hydraulics toolbox

Block Screenshot



Module

- xCOS

Description

Thermal-Hydraulics toolbox contains some basic thermal-hydraulic components such as pressure source, pipe, control valves, etc.

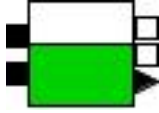
Blocks

- Bache - Thermal-hydraulic tank (reservoir)
- PerteDP - Thermal-hydraulic pipe
- PuitsP - Thermal-hydraulic drain (well)
- SourceP - Thermal-hydraulic constant pressure source
- VanneReglante - Thermal-hydraulic control valve

Name

Bache — Thermal-hydraulic tank (reservoir)

Block Screenshot



Contents

- Thermal-hydraulic tank (reservoir)
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Thermal-Hydraulics palette

Description

The **Bache** block represents a thermal-hydraulic tank or reservoir. This block has two inlets and two outlets whose altitudes can be changed by the user. Conventionally, for input ports (black ports) flow direction is positive when fluid flows into the tank. On the other hand, for output ports (white ports) flow direction is positive when fluid flows out of the tank. The user can set the surface area of the tank, the initial temperature and initial level of the fluid in the tank. If an input or output port is left unused, it should be blocked by a **stopper** block.

Dialog box

Parametres de la bache	
Pression dans le ciel de la bache : Patm (Pa)	101300
Section de la bache : A (m2)	1
Altitude du piquage d'entrée 1: ze1 (m)	40
Altitude du piquage d'entrée 2: ze2 (m)	0
Altitude du piquage de sortie 1: zs1 (m)	40
Altitude du piquage de sortie 2: zs2 (m)	0
Altitude initiale du fluide : z0 (m)	0
Température initiale du fluide : T0 (K)	290
Si >0, masse volumique imposée du fluide : p_rho (kg/m3)	1
Dismiss OK	

- **Pression dans le ciel de la bache : Patm (Pa)**

Atmospheric pressure inside the tank.

Properties : Type 'vec' of size -1.

- **Section de la bache : A (m2)**

Surface area of the tank.

Properties : Type 'vec' of size -1.

- **Altitude du piquage d'entrée 1: ze1 (m)**

Altitude of the first input port

Properties : Type 'vec' of size -1.

- **Altitude du piquage d'entrée 2: ze2 (m)**

Altitude of the second input port

Properties : Type 'vec' of size -1.

- **Altitude du piquage de sortie 1: zs1 (m)**

Altitude of the first output port.

Properties : Type 'vec' of size -1.

- **Altitude du piquage de sortie 2: zs2 (m)**

Altitude of the second output port.

Properties : Type 'vec' of size -1.

- **Altitude initiale du fluide : z0 (m)**
Initial fluid level in the tank.
Properties : Type 'vec' of size -1.
- **Température initiale du fluide : T0 (K)**
Temperature of fluid in the tank
Properties : Type 'vec' of size -1.
- **Si 0, masse volumique imposée du fluide : p_rho (kg/m3)**
Density of fluid
Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**
 - **Modelica variable name : 'Ce1'**
Implicit **variable**.
 - **Modelica variable name : 'Ce2'**
Implicit **variable**.
- **Outputs :**
 - **Modelica variable name : 'Cs1'**
Implicit **variable**.
 - **Modelica variable name : 'Cs2'**
Implicit **variable**.
 - **Modelica variable name : 'yNiveau'**
Explicit **variable**.
- **Parameters :**
 - **Modelica parameter name : 'Patm'**
Default value : 101300
Is a state variable : no.
 - **Modelica parameter name : 'A'**
Default value : 1
Is a state variable : no.
 - **Modelica parameter name : 'ze1'**
Default value : 40
Is a state variable : no.

- **Modelica parameter name :** 'ze2'
Default value : 0
Is a state variable : no.
- **Modelica parameter name :** 'zs1'
Default value : 40
Is a state variable : no.
- **Modelica parameter name :** 'zs2'
Default value : 0
Is a state variable : no.
- **Modelica parameter name :** 'z0'
Default value : 30
Is a state variable : no.
- **Modelica parameter name :** 'T0'
Default value : 290
Is a state variable : no.
- **Modelica parameter name :** 'p_rho'
Default value : 0
Is a state variable : no.
- **File name of the model :** Bache

Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/Bache.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Bache.mo

Name

PerteDP — Thermal-hydraulic pipe

Block Screenshot



Contents

- Thermal-hydraulic pipe
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Thermal-Hydraulics palette

Description

A **PertDP** block represents a hydraulic pipe with linear resistance losses. This component represents a hydraulic resistance and pressure loss is directly proportional to the flow rate. Conventionally, the flow direction is the positive when fluid flows from the black port to the white port. The pressure loss is obtained

with

$$P_{loss} = P_{black} - P_{white}$$

. The key parameters of this block are the pipes' length, the pipe's diameter, inlet and outlet altitudes, and some other thermal-hydraulic coefficients.

Dialog box

Paramètres du tuyau	
Longueur du tube : L (m)	10
Diamètre interne du tube : D (m)	0.2
Coefficient de perte de charge-frottement(S.U) : lambda	1111
Altitude entrée tuyauterie : z1 (m)	0
Altitude sortie tuyauterie : z2 (m)	0
Si 0, masse volumique imposée fu fluide : p_rho (kg/m3)	11

Dismiss OK

- **Longueur du tube : L (m)**
Length of pipe
Properties : Type 'vec' of size -1.
- **Diamètre interne du tube : D (m)**
Pipe diameter
Properties : Type 'vec' of size -1.
- **Coefficient de perte de charge-frottement(S.U) : lambda**
Coefficient of thermohydraluc resistance
Properties : Type 'vec' of size -1.
- **Altitude entrée tuyauterie : z1 (m)**
Altitude of the first port (z1)
Properties : Type 'vec' of size -1.
- **Altitude sortie tuyauterie : z2 (m)**
Altitude of the second port (z2)
Properties : Type 'vec' of size -1.
- **Si 0, masse volumique imposée fu fluide : p_rho (kg/m3)**
Fluid density
Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**
 - **Modelica variable name :** 'C1'
- **Implicit variable.**

- **Outputs :**
 - **Modelica variable name :** 'C2'
Implicit variable.
- **Parameters :**
 - **Modelica parameter name :** 'L'
Default value : 10
Is a state variable : no.
 - **Modelica parameter name :** 'D'
Default value : 0.2
Is a state variable : no.
 - **Modelica parameter name :** 'lambda'
Default value : 0.03
Is a state variable : no.
 - **Modelica parameter name :** 'z1'
Default value : 0
Is a state variable : no.
 - **Modelica parameter name :** 'z2'
Default value : 0
Is a state variable : no.
 - **Modelica parameter name :** 'p_rho'
Default value : 0
Is a state variable : no.
- **File name of the model :** PerteDP

Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/PerteDP.sci

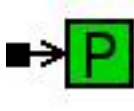
Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/PerteDP.mo

Name

PuitsP — Thermal-hydraulic drain (well)

Block Screenshot



Contents

- Thermal-hydraulic drain (well)
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
- Modelica model

Palette

- Thermal-Hydraulics palette

Description

This thermal-hydraulic component represents a thermal-hydraulic constant pressure drain (well). This block is specified with its pressure and temperature. Conventionally, the flow direction is positive when fluid flows into the block.

Dialog box

Paramètres du puit	
Pression de la source : P0 (Pa)	00000
Temperature de la source : T0 (K)	290
Enthalpie spécifique de la source : H0 (J/kg)	00000
1:température fixée - 2:enthalpie fixée : option_temperature	-
DismissOK	

- **Pression de la source : P0 (Pa)**

Pressure of the thermohydraulic source.

Properties : Type 'vec' of size -1.
- **Temperature de la source : T0 (K)**

Temperature of the thermohydraulic source.

Properties : Type 'vec' of size -1.

- **Enthalpie spécifique de la source : H0 (J/kg)**

Specific Enthalpy of the thermohydraulic source.

Properties : Type 'vec' of size -1.

- **1:température fixée - 2:enthalpie fixée : option_temperature**

Temperature option. 1: fixed temperature - 2: fixed enthalpy

Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**

- **Modelica variable name : 'C'**

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name : 'P0'**

Default value : 100000

Is a state variable : no.

- **Modelica parameter name : 'T0'**

Default value : 290

Is a state variable : no.

- **Modelica parameter name : 'H0'**

Default value : 100000

Is a state variable : no.

- **Modelica parameter name : 'option_temperature'**

Default value : 1

Is a state variable : no.

- **File name of the model :** Puits

Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/PuitsP.sci

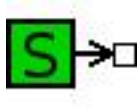
Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Puits.mo

Name

SourceP — Thermal-hydraulic constant pressure source

Block Screenshot



Contents

- Thermal-hydraulic constant pressure source
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Thermal-Hydraulics palette

Description

This thermal-hydraulic component represents a thermal-hydraulic constant pressure supply. This block is specified with its output pressure and temperature. Conventionally, the flow direction is positive when the fluid flows out of the block.

Dialog box

Paramètres du puit	
Pression de la source : P0 (Pa)	300000
Temperature de la source : T0 (K)	290
Enthalpie spécifique de la source : H0 (J/kg)	00000
1:température fixée - 2:enthalpie fixée : option_temperature	-
Dismiss OK	

- **Pression de la source : P0 (Pa)**
Pressure of the thermohydraulic source
Properties : Type 'vec' of size -1.
- **Temperature de la source : T0 (K)**

Temperature of the thermohydraulic source

Properties : Type 'vec' of size -1.

- **Enthalpie spécifique de la source : H0 (J/kg)**

Specific enthalpie of the thermohydraulic source

Properties : Type 'vec' of size -1.

- **1:température fixée - 2:enthalpie fixée : option_temperature**

Temperature option. 1: fixed temperature - 2: fixed enthalpy

Properties : Type 'vec' of size -1.

Default properties

- **Outputs :**

- **Modelica variable name : 'C'**

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name : 'P0'**

Default value : 300000

Is a state variable : no.

- **Modelica parameter name : 'T0'**

Default value : 290

Is a state variable : no.

- **Modelica parameter name : 'H0'**

Default value : 100000

Is a state variable : no.

- **Modelica parameter name : 'option_temperature'**

Default value : 1

Is a state variable : no.

- **File name of the model :** Source

Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/SourceP.sci

Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Source.mo

Name

VanneReglante — Thermal-hydraulic control valve

Block Screenshot



Contents

- Thermal-hydraulic control valve
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- Modelica model

Palette

- Thermal-Hydraulics palette

Description

The **VanneReglante** block represents a variable orifice control valve. The flow rate through the valve is proportional to the valve opening, ,

$$\Delta P \times h \times |h| = k \times Q \times |Q|$$

where h is the valve opening, ΔP is the pressure difference, and Q is the flow rate. This model is only used for the laminar flow regimes. k is a constant depending on the valve geometry and mass density of fluid.

Dialog box

Paramètres de la vanne réglante

Cvmax 8005.42

p_rho 0

Dismiss OK

- Cvmax

Cvmax (maximum opening of the valve)

Properties : Type 'vec' of size -1.

- **p_rho**

Fluid density

Properties : Type 'vec' of size -1.

Default properties

- **Inputs :**

- **Modelica variable name :** 'C1'

Implicit **variable**.

- **Modelica variable name :** 'Ouv'

Explicit **variable**.

- **Outputs :**

- **Modelica variable name :** 'C2'

Implicit **variable**.

- **Parameters :**

- **Modelica parameter name :** 'Cvmax'

Default value : 8005.42

Is a state variable : no.

- **Modelica parameter name :** 'p_rho'

Default value : 0

Is a state variable : no.

- **File name of the model :** VanneReglante

Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/VanneReglante.sci

Modelica model

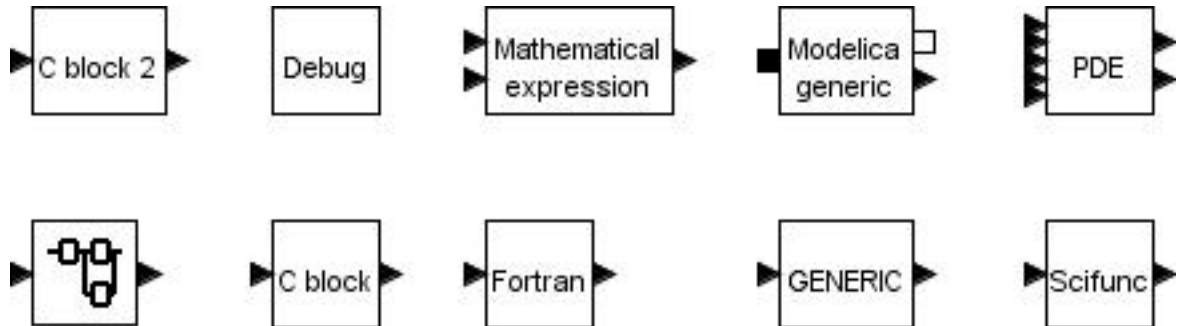
- SCI/modules/scicos_blocks/macros/Hydraulics/VanneReglante.mo

20. User defined functions palette

Name

Userdefinedfunctions_pal — User defined functions palette

Block Screenshot



Module

- xcOS

Description

The user defined function contains blocks that allow you to model the component behaviour. The output is expressed as a function of the input.

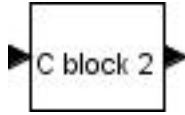
Blocks

- c_block — C file function
- CBLOCK — New C
- EXPRESSION — Mathematical expression
- fortran_block — Fortran
- generic_block3 — Generic block
- MBLOCK — Modelica generic block
- PDE — 1D PDE block
- scifunc_block_m — Scilab function block
- SUPER_f — Super block
- TEXT_f — Text

Name

CBLOCK — New C

Block Screenshot



Contents

- New C
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- User defined functions palette

Description

This block creates skeleton of the C-computing function. It also creates library file and object files.

Dialog box

3et C-Block2 block parameters	
simulation function	toto
is block implicit? (y,n)	n
input ports sizes	1
output ports sizes	1
input event ports sizes	0
output events ports sizes	0
initial continuous state	0
number of zero crossing surfaces	0
initial discrete state	0
Real parameters vector	0
Integer parameters vector	0
initial firing vector (<0 for no firing)	0
direct feedthrough (y or n)	y
time dependence (y or n)	n
Dismiss OK	

- **simulation function**

Name of the function to be generated.

Properties : Type 'str' of size 1

- **is block implicit?**

If yes (y) is selected, it calls implicit solver (dasrt) else (n) it calls explicit solver, lsodar.

Properties : Type 'str' of size 1

- **input ports sizes**

Number of regular input ports

Properties : Type 'vec' of size -1

- **output ports sizes**

Number of regular output ports

Properties : Type 'vec' of size -1

- **input event ports sizes**

Number of event input ports

Properties : Type 'vec' of size -1

- **output events ports sizes**

Number of event output ports

Properties : Type 'vec' of size -1

- **initial continuous state**

Initial Conditions

Properties : Type 'vec' of size -1

- **number of zero crossing surfaces**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

- **initial discrete state**

Initial conditions of the discrete states.

Properties : Type 'vec' of size -1

- **Real parameters vector**

Real Parameter vector that the function accepts.

Properties : Type 'vec' of size -1

- **Integer parameters vector**

Integer Parameter vector that the function accepts.

Properties : Type 'vec' of size -1

- **initial firing vector**

A vector. Size of this vector corresponds to the number of event outputs. The value of the i^{th} entry specifies the time of the preprogrammed event firing on the i^{th} output event port. If less than zero, no event is preprogrammed.

Properties : Type 'vec' of size 'sum(%6)'

- **direct feedthrough**

The input to the block at the current time determine the output of the block at the current time. This forces the input to feed through to the output, as if the system were operating at steady-state.

Properties : Type 'str' of size 1

- **time dependence**

Create a signal that specifies the time dependence.

Properties : Type 'str' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/CBLOCK.sci

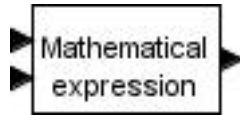
Authors

Ramine Nikoukhah - INRIA

Name

EXPRESSION — Mathematical expression

Block Screenshot



Contents

- Mathematical expression
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- User defined functions palette

Description

The Expression block applies the specified Scilab functions to its input.

Dialog box

Give a scalar scilab expression using inputs u1, u2,...
If only one input, input is vector [u1,u2,...] (max 8)
ex: (dd*u1+sin(u2)>0)*u3
Note that here dd must be defined in context

number of inputs	2
scilab expression	(u1>0)*sin(u2)^2
use zero-crossing (0: no, 1 yes)	1

- **number of inputs**

Block input can be a scalar or vector.

Properties : Type 'vec' of size 1

- **scilab expression**

The Scilab expression applied to the input.

Properties : Type 'vec' of size 1

- **use zero-crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evaluate_expr*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/EXPRESSION.sci

Computational function

- SCI/modules/scicos_blocks/src/c/evaluate_expr.c (Type 4)

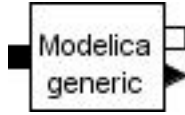
Authors

Ramine Nikoukhah - INRIA

Name

MBLOCK — Modelica generic block

Block Screenshot



Contents

- Modelica generic block
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

Palette

- User Defined function palette

Description

The block "MBlock" provides an easy way to build a xcos block whose behavior is specified by a Modelica program. Using this block, the user will be able to write and compile Modelica programs in xcos without creating any interfacing function. The associated Modelica program of this block can be either given in a file or written in the window opened by the block. In order to link this block to other xcos blocks that may be other Modelica blocks, the types of block ports' as well as their associated variables should be specified.

Dialog box

Set Modelica generic block parameters

Input variables:	"u1"
Input variables types:	"I"
Output variables:	["y1";"y2"]
Output variables types:	["I";"E"]
Parameters in Modelica:	["R";"L"]
Parameters properties:	[0;0]
Function name:	generic

Dismiss OK

- **Input variables**

In this field, the ports connected to the left hand side of the block are defined. If the port is an explicit port, it will be an input port. In this case, the variable should be declared in the Modelica program as Real. If the port is an implicit port, the variable designating this port should be a "connector". Remind that for implicit port, the notion of input and output does not exist and specifying an implicit variable in this field is just placing the port at the left hand side of the block.

- **Input variables types**

In this field, the type of ports are specified, i.e., 'I' for implicit ports and 'E' for explicit ports. The size of the vector of "input variables" and the vector of "input_vector_type" should be equal.

- **Output variables**

Similar to the input variables vector, the explicit output variables and implicit variables which are displayed at the right hand side of the block are specified in this field.

- **Output variables types**

The type of variables given in the Output variable vector are specified, i.e., 'I' for implicit ports and 'E' for explicit ports.

- **Parameters in Modelica**

The values of parameters declared in the Modelica program can be overloaded. To overload a parameter value, the name of parameters are given in this field and their corresponding values are given in the "parameter values" fields that are displayed in the second dialog box.

- **Parameters properties**

The type of the Modelica parameters. For that time being, one can parametrize three types of Modelica variable :

- **0** : the parameter is set to be a **Modelica parameter** variable (scalar or vector).

- **1** : the parameter is set to be an **initial condition of Modelica state** variable (scalar or vector).
- **2** : the parameter is set to be an **initial condition of Modelica state** variable with the property **fixed=true** (scalar or vector).

- **Function name**

The Modelica class name is specified in this field. If the Modelica class name is specified without any path or extension, an interactive window is opened and the user can write or edit the Modelica program. This window is opened each time the user clicks on the block. If the Modelica class name is specified with path and '.mo' extension, the compiler looks for the file and if it is found, the file will be compiled, otherwise a window is opened and the user can write the Modelica program. This Modelica file will be saved with the given filename in the specified path. The next time, only input/output characteristics of the block can be changed, and the Modelica file should be edited with another text editor.

- **Parameter values**

The value of Modelica parameters are given in the "Set parameters values" dialog box. These values that can be scalar or vector, can also be defined in the xcoss context. In order to access the xcoss context, click on the "Diagram" menu then click on the "Context" submenu. For instance, here is an example of overloading of parameters in a Modelica program. Parameters vector = ['Speed';'Position';'Length']

Parameters properties vector = [0;2;1] Speed value = [12.0]

Position value = [0.0 ; 0.1 ; POS]

Length value = [13.0 ; 12.1]

Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
 - port 2 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *generic*

Interfacing function

- `SCI/modules/scicos_blocks/macros/Misc/MBLOCK.sci`

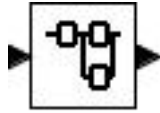
Authors

M. Najafi/A. Layec - INRIA

Name

SUPER_f — Super block

Block Screenshot



Contents

- Super block
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

Palette

- User defined functions palette

Description

This block opens up a new Scicos window for editing a new block diagram. This diagram describes the internal functions of the super block.

Super block inputs and outputs (regular or event) are designated by special (input or output) blocks.

Regular input blocks must be numbered from 1 to the number of regular input ports. Regular input ports of the super block are numbered from the top of the block shape to the bottom. Regular output ports must be numbered from 1 to the number of regular output ports. Regular output ports of the super block are numbered from the top of the block shape to the bottom. Event input blocks must be numbered from 1 to the number of event input ports. Event input ports of the super block are numbered from the left of the block shape to the right. Event output ports must be numbered from 1 to the number of event output ports. Event output ports of the super block are numbered from the left of the block shape to the right.

Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**

- port 1 : size [1,1] / type 1

- number/sizes of activation inputs: 0
- number/sizes of activation outputs: 0
- continuous-time state: no
- discrete-time state: no
- object discrete-time state: no
- name of computational function: *super*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/SUPER_f.sci

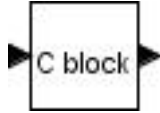
Authors

Ramine Nikoukhah - INRIA

Name

c_block — C language

Block Screenshot



Contents

- C language
- the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Authors”

Palette

- User defined functions palette

Description

This block creates skeleton of the C computing function. Also it creates library file and object files.

Dialog box

A dialog box titled "Set C_block parameters". It contains four input fields: "input ports sizes" with the value "1", "output port sizes" with the value "1", "System parameters vector" with an empty array "[]", and "function name" with the value "toto". At the bottom, there are two buttons: "Dismiss" and "OK".

Set C_block parameters	
input ports sizes	1
output port sizes	1
System parameters vector	[]
function name	toto
DismissOK	

- **input ports sizes**
Number of regular input ports.
Properties : Type 'vec' of size -1.

- **output port sizes**

Number of regular output ports.

Properties : Type 'vec' of size -1.

- **System parameters vector**

Number of parameters that this function accepts.

Properties : Type 'vec' of size -1.

- **function name**

Name of the function to be generated.

Properties : Type 'str' of size -1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/c_block.sci

Authors

Ramine Nikoukhah - INRIA

Name

fortran_block — Fortran

Block Screenshot



Contents

- Fortran
- • the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Authors”

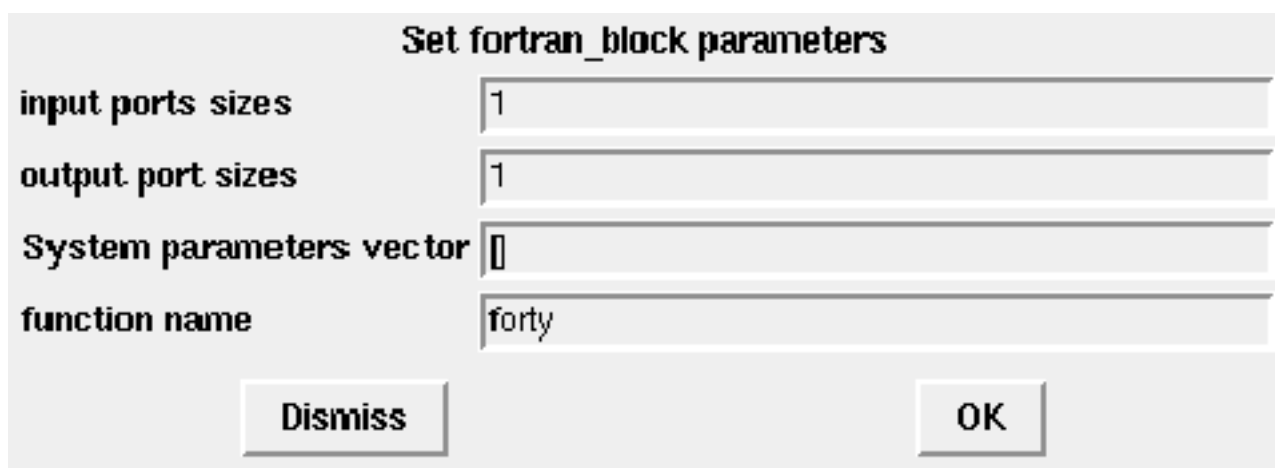
Palette

- User defined functions palette

Description

This block creates skeleton of the FORTRAN computing function. Also it creates library file and object files.

Dialog box



Set fortran_block parameters	
input ports sizes	1
output port sizes	1
System parameters vector	[]
function name	forty
DismissOK	

- **input ports sizes**

Number of regular input ports.

Properties : Type 'vec' of size -1.

- **output port sizes**

Number of regular output ports.

Properties : Type 'vec' of size -1.

- **System parameters vector**

Number of parameters that this function accepts.

Properties : Type 'vec' of size -1.

- **function name**

Name of the function to be generated.

Properties : Type 'vec' of size -1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/fortran_block.sci

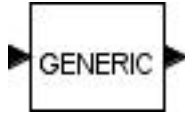
Authors

Ramine Nikoukhah - INRIA

Name

generic_block3 — Generic block

Block Screenshot



Contents

- Generic block
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”
 - the section called “Authors”

Palette

- User defined functions palette

Description

The block provides a generic interfacing function but the computational function needs to be defined separately, either as a Scilab function or a Fortran or a C function. Besides the name of the function, user should specify information such as the type, whether or not the block contains a direct feed-through term. The function realising computational functions of generic blocks of a Scicos diagram must be saved along with the diagram and loaded or dynamically linked before simulation.

Dialog box

Set GENERIC block parameters	
Simulation function	sinblk
Function type (0,1,2,...)	4
Input ports sizes	[1,1]
Input ports type	1
Output port sizes	[1,1]
Output ports type	1
Input event ports sizes	[]
Output events ports sizes	[]
Initial continuous state	[]
Initial discrete state	[]
Initial object state	[]
Real parameters vector	[]
Integer parameters vector	[]
Object parameters list	[]
Number of modes	0
Number of zero crossings	0
Initial firing vector (<0 for no firing)	[]
Direct feedthrough (y or n)	y
Time dependence (y or n)	n
Dismiss OK	

- **Simulation function**

Name of the function to be loaded.

Properties : Type 'str' of size 1

- **Function type**

Type of the computational function supported by Scicos.

Properties : Type 'vec' of size 1

- **Input ports sizes**

Number of regular input ports.

Properties : Type 'mat' of size [-1,2]

- **Input ports type**

Set the datatype of the regular input ports.

Properties : Type 'vec' of size -1

- **Iutput port sizes**

Number of regular input ports.

Properties : Type 'mat' of size [-1,2]

- **Output ports type**

Set the datatype of the regular output ports.

Properties : Type 'vec' of size -1

- **Input event ports sizes**

a vector of ones, size of event input ports. The size of the vector gives the number of event input ports.

Properties : Type 'vec' of size -1

- **Output events ports sizes**

a vector of ones, size of event output ports. The size of the vector gives the number of of event output ports.

Properties : Type 'vec' of size -1

- **Initial continuous state**

A column vector of Initial State Conditions.

Properties : Type 'vec' of size -1

- **Initial discrete state**

A column vector Initial discrete Conditions.

Properties : Type 'vec' of size -1

- **Initial object state**

A Scilab list that defined the initial object state (oz).

Properties : Type 'lis' of size -1

- **Real parameters vector**

column vector. Any parameters used in the block can be defined here as a column vector.

Properties : Type 'vec' of size -1

- **Integer parameters vector**

column vector. Any integer parameters used in the block can be defined here as a column vector.

Properties : Type 'vec' of size -1

- **Object parameters list**

A Scilab list that defined the list of the Object parameters (opar).

Properties : Type 'lis' of size -1

- **Number of modes**

Number of Right hand side functions in the system.

Properties : Type 'vec' of size 1

- **Number of zero_crossings**

No. of zero-crossings

Properties : Type 'vec' of size 1

- **Initial firing vector**

vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.

Properties : Type 'vec' of size sum(%6)

- **Direct feedthrough**

character "y" or "n", specifies if block has a direct input to output feedthrough.

Properties : Type 'vec' of size 1

- **Time dependence**

Time dependence : character "y" or "n", specifies if block output depends explicitly on time.

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sinblk*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/generic_block3.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/sinblk.f (Type 4)

Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

Name

scifunc_block_m — Scilab function block

Block Screenshot



Contents

- Scilab function block
 - the section called “Palette”
 - the section called “Description”
 - the section called “Dialog box”
 - the section called “Default properties”
 - the section called “Interfacing function”
 - the section called “Computational function”

Palette

- User defined functions palette

Description

This block can realize any type of Scicos block. The function of the block is defined interactively using dialogue boxes and in Scilab language. During simulation, these instructions are interpreted by Scilab; the simulation of diagrams that include these types of blocks is slower. For more information see Scicos reference manual.

Dialog box

Set scifunc_block parameters
only regular blocks supported

input ports sizes	[1,1]
output port sizes	[1,1]
input event ports sizes	[]
output events ports sizes	[]
initial continuous state	[]
initial discrete state	[]
System parameters vector	[]
initial timing vector (<0 for no timing)	[]
is block always active (0:no, 1:yes)	0

Dismiss **OK**

- **input ports sizes**
a scalar. Number of regular input ports
Properties : Type 'vec' of size -1
- **output port sizes**
a scalar. Number of regular output ports
Properties : Type 'vec' of size -1
- **input event ports sizes**
a scalar. Number of input event ports
Properties : Type 'vec' of size -1
- **output events ports sizes**
a scalar. Number of output event ports
Properties : Type 'vec' of size -1
- **initial continuous state**
a column vector.
Properties : Type 'vec' of size -1
- **initial discrete state**
a column vector.

Properties : Type 'vec' of size -1

- **System parameters vector**

a string: c or d (**CBB** or **DBB**), other types are not supported.

Properties : Type 'vec' of size -1

- **initial firing vector**

vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.

Properties : Type 'vec' of size sum(%4)

- **is block always active**

other dialogues are opened consecutively where used may input Scilab code associated with the computations needed (block initialization, outputs, continuous and discrete state, output events date, block ending)

Properties : Type 'vec' of size 1

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **regular outputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *scifunc*

Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/scifunc_block_m.sci

Computational function

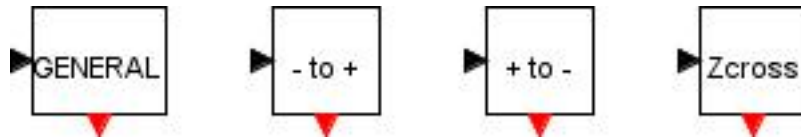
- SCI/modules/scicos/src/fortran/scifunc.f (Type 3)

21. Zero crossing detection palette

Name

Zerocrossingdetection_pal — Zero crossing detection palette

Block Screenshot



Module

- xcOS

Description

Zero crossing detection blocks are used to detect values crossing of state variables during the simulation. These blocks use the solvers (ODE or DAE) to do that operation.

Blocks

- CLINDUMMY_f — Dummy
- GENERAL_f — Zero crossing
- NEGTOPOS_f - Threshold negative to positive
- POSTONEG_f - Threshold positive to negative
- ZCROSS_f - Threshold detection at zero

Name

GENERAL_f — GENERAL_f title

Block Screenshot



Contents

- GENERAL_f title
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”

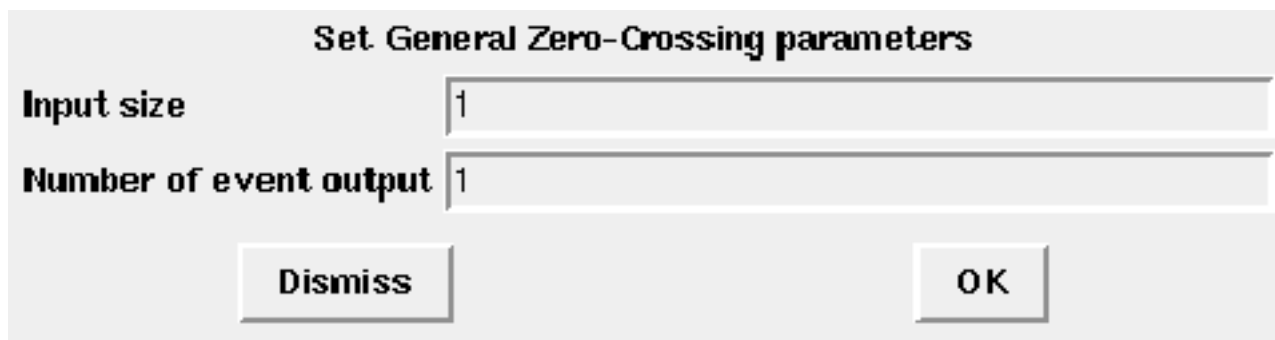
Palette

- Zero crossing detection palette

Description

Add here a paragraph of the function description

Dialog box



- **Input size**
The parameter description 1.
Properties : Type 'vec' of size 1.
- **Number of event output**
The parameter description 2.

Properties : Type 'vec' of size 1.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - **port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/GENERAL_f.sci

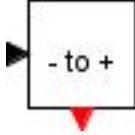
Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

Name

NEGTOPOS_f — Threshold negative to positive

Block Screenshot



Contents

- Threshold negative to positive
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Zero crossing detection palette

Description

An output event is generated when the unique input crosses zero with a positive slope.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

Interfacing function

- `SCI/modules/scicos_blocks/macros/Threshold/NEGTPOPOS_f.sci`

Computational function

- `SCI/modules/scicos_blocks/src/fortran/zcross.f` (Type 1)

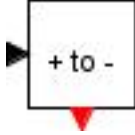
Authors

Ramine Nikoukhah - INRIA

Name

POSTONEG_f — Threshold positive to negative

Block Screenshot



Contents

- Threshold positive to negative
- • the section called “Palette”
- the section called “Description”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

Palette

- Zero crossing detection palette

Description

An output event is generated when the unique input crosses zero with a negative slope.

Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/POSTONEG_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

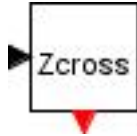
Authors

Ramine Nikoukhah - INRIA

Name

ZCROSS_f — Threshold detection at zero

Block Screenshot



Contents

- Threshold detection at zero
- • the section called “Palette”
- the section called “Description”
- the section called “Dialog box”
- the section called “Default properties”
- the section called “Interfacing function”
- the section called “Computational function”
- the section called “Authors”

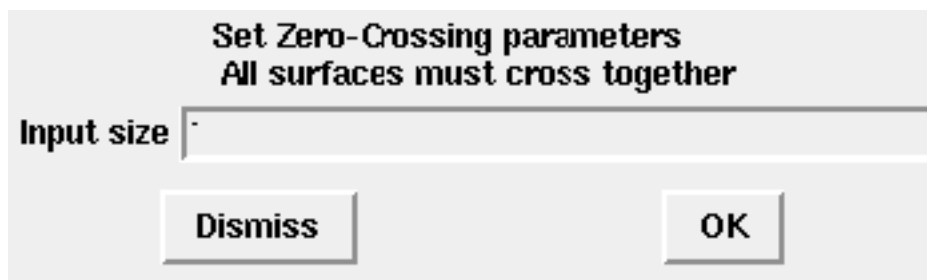
Palette

- Zero crossing detection palette

Description

An output event is generated when all inputs (if more than one) cross zero simultaneously.

Dialog box



- **Input size**
a positive integer.
Property : Type 'vec' of size 1.

Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**
 - port 1 : size [1,1] / type 1
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/ZCROSS_f.sci

Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

Authors

Ramine Nikoukhah - INRIA

Chapter 8. Programming xcos Blocks

1. C Computational Functions

Name

C_macros — Utilities C macros

Contents

- C_macros - Utilities C macros
- • the section called “Module”
- the section called “Description”
- the section called “Inputs/outputs”
- the section called “Events”
- the section called “Parameters”
- the section called “States and work”
- the section called “Zero crossing surfaces and modes”
- the section called “Authors”

Module

- xcoss

Description

The following C macros are available by including the file in a C computational function.

Inputs/outputs

Macro	Description
GetNin(blk)	Get number of regular input port.
GetInPortRows(blk,x)	Get number of rows (first dimension) of regular input port number x.
GetInPortCols(blk,x)	Get number of columns (second dimension) of regular input port number x.
GetInPortSize(blk,x,y)	Get regular input port size number x. (y=1 for the first dimension, y=2 for the second dimension)
GetInType(blk,x)	Get type of regular input port number x.
GetInPortPtrs(blk,x)	Get regular input port pointer of port number x.
GetRealInPortPtrs(blk,x)	Get pointer of real part of regular input port number x.
GetImagInPortPtrs(blk,x)	Get pointer of imaginary part of regular input port number x.
Getint8InPortPtrs(blk,x)	Get pointer of int8 typed regular input port number x.
Getint16InPortPtrs(blk,x)	Get pointer of int16 typed regular input port number x.
Getint32InPortPtrs(blk,x)	Get pointer of int32 typed regular input port number x.

Getuint8InPortPtrs(blk,x)	Get pointer of uint8 typed regular input port number x.
Getuint16InPortPtrs(blk,x)	Get pointer of uint16 typed regular input port number x.
Getuint32InPortPtrs(blk,x)	Get pointer of uint32 typed regular input port number x.
GetSizeOfIn(blk,x)	Get the sizeof of the regular input port number x.
GetNout(blk)	Get number of regular output port.
GetOutPortRows(blk,x)	Get number of rows (first dimension) of regular output port number x.
GetOutPortCols(blk,x)	Get number of columns (second dimension) of regular output port number x.
GetOutPortSize(blk,x,y)	Get regular output port size number x. (y=1 for the first dimension, y=2 for the second dimension)
GetOutType(blk,x)	Get type of regular output port number x.
GetOutPortPtrs(blk,x)	Get regular output port pointer of port number x.
GetRealOutPortPtrs(blk,x)	Get pointer of real part of regular output port number x.
GetImagOutPortPtrs(blk,x)	Get pointer of imaginary part of regular output port number x.
Getint8OutPortPtrs(blk,x)	Get pointer of int8 typed regular output port number x.
Getint16OutPortPtrs(blk,x)	Get pointer of int16 typed regular output port number x.
Getint32OutPortPtrs(blk,x)	Get pointer of int32 typed regular output port number x.
Getuint8OutPortPtrs(blk,x)	Get pointer of uint8 typed regular output port number x.
Getuint16OutPortPtrs(blk,x)	Get pointer of uint16 typed regular output port number x.
Getuint32OutPortPtrs(blk,x)	Get pointer of uint32 typed regular output port number x.
GetSizeOfOut(blk,x)	Get the sizeof of the regular output port number x.

Events

Macro	Description
GetNevIn(blk)	Get the input event number.
GetNevOut(blk)	Get number of event output port.
GetNevOutPtrs(blk)	Get pointer of event output register.

Parameters

Macro	Description
GetNipar(blk)	Get number of integer parameters.

GetIparPtrs(blk)	Get pointer of the integer parameters register
GetNrpar(blk)	Get number of real parameters.
GetRparPtrs(blk)	Get pointer of the real parameters register.
GetNopar(blk)	Get number of object parameters.
GetOparType(blk,x)	Get type of object parameters number x.
GetOparSize(blk,x,y)	Get size of object parameters number x. (y=1 for the first dimension, y=2 for the second dimension)
GetOparPtrs(blk,x)	Get pointer of object parameters number x.
GetRealOparPtrs(blk,x)	Get pointer of real object parameters number x.
GetImagOparPtrs(blk,x)	Get pointer of imaginary part of object parameters number x.
Getint8OparPtrs(blk,x)	Get pointer of int8 typed object parameters number x.
Getint16OparPtrs(blk,x)	Get pointer of int16 typed object parameters number x.
Getint32OparPtrs(blk,x)	Get pointer of int32 typed object parameters number x.
Getuint8OparPtrs(blk,x)	Get pointer of uint8 typed object parameters number x.
Getuint16OparPtrs(blk,x)	Get pointer of uint16 typed object parameters number x.
Getuint32OparPtrs(blk,x)	Get pointer of uint32 typed object parameters number x.
GetSizeOfOpar(blk,x)	Get the sizeof of the object parameters number x.

States and work

Macro	Description
GetNstate(blk)	Get number of continuous state.
GetState(blk)	Get pointer of the continuous state register.
GetDstate(blk)	Get number of discrete state.
GetNdstate(blk)	Get pointer of the discrete state register.
GetNoz(blk)	Get number of object state.
GetOzType(blk,x)	Get type of object state number x.
GetOzSize(blk,x,y)	Get size of object state number x. (y=1 for the first dimension, y=2 for the second dimension)
GetOzPtrs(blk,x)	Get pointer of object state number x.
GetRealOzPtrs(blk,x)	Get pointer of real object state number x.
GetImagOzPtrs(blk,x)	Get pointer of imaginary part of object state number x.
Getint8OzPtrs(blk,x)	Get pointer of int8 typed object state number x.
Getint16OzPtrs(blk,x)	Get pointer of int16 typed object state number x.
Getint32OzPtrs(blk,x)	Get pointer of int32 typed object state number x.
Getuint8OzPtrs(blk,x)	Get pointer of uint8 typed object state number x.

Getuint16OzPtrs(blk,x)	Get pointer of uint16 typed object state number x.
Getuint32OzPtrs(blk,x)	Get pointer of uint32 typed object state number x.
GetSizeOfOz(blk,x)	Get the sizeof of the object state number x.
GetWorkPtrs(blk)(blk)	Get the pointer of the Work array.

Zero crossing surfaces and modes

Macro	Description
GetNg(blk)	Get number of zero crossing surface.
GetGPtrs(blk)	Get pointer of the zero crossing register.
GetNmode(blk)	Get number of modes.
GetModePtrs(blk)	Get pointer of the mode register.

Authors

Alan Layec - INRIA

Name

C_struct — C Block structure of a computational function

Contents

- C_struct - C Block structure of a computational function
- • the section called “Module”
- the section called “Description”
- the section called “Inputs/outputs”
- the section called “Events”
- the section called “Parameters”
- the section called “States and work”
- the section called “Zero crossing surfaces and modes”
- the section called “Miscellaneous”
- the section called “Authors”

Module

- xcos

Description

The C structure of a xcos block defines all the fields to handle data provided by the simulator such inputs/outputs, parameters, states, ...

That structure of type is defined in the file , and user must include that header in each computational functions in the form :

```
#include <varargs.h>
```

```
void computational_function(scicos_block *block, , int flags);
```

```
scicos_block *block, , int flags ;
```

The fields, that can be either C pointers or directly data, are then accessible via the `scicos_block` structure : `block->field`

This access is a approach and most of users should prefer the approach for facilities purpose.

In the current version of xcos, the structure is defined :

```
/* scicos_block structure definition */
typedef struct {
    int nevprt;
    voidg funpt ;
    int type;
    int scsptr;
    int nz;
    double *z;
    int noz;
```

```

int *ozsz;
int *oztyp;
void **ozptr;
int nx;
double *x;
double *xd;
double *res;
int nin;
int *insz;
void **inptr;
int nout;
int *outsz;
void **outptr;
int nevout;
double *evout;
int nrpar;
double *rpar;
int nipar;
int *ipar;
int nopar;
int *oparsz;
int *opartyp;
void **oparptr;
int ng;
double *g;
int ztyp;
int *jroot;
char *label;
void **work;
int nmode;
int *mode;
} scicos_block;

```

Inputs/outputs

- **block->nin** : Integer that gives the number of regular input ports of the block.

One can't override the index when reading sizes of input ports in the array and the index when reading data in the array with a C computational function.

The number of regular input ports can also be got by the use of the C macros .

- **block->insz** : An array of integers of size that respectively gives the first dimensions, the second dimensions and the type of data driven by regular input ports.

Note that this array of size differs from the array and to provide full compatibility with blocks that only use a single dimension.

Suppose that you have a block with three inputs : the first is an int32 matrix of size 3,2, the second a single complex number (matrix of size 1,1) and the last a real matrix of size 4,1.

In thescicos_model of such a block, the inputs will be defined : and the corresponding field at C computational function level will be coded as : Do the difference here in the type numbers defined at the (2,1,3) and the type numbers defined at the (84,11,10). The following table gives the correspondance for all xcos type:

- **block->inptr** : An array of pointers of size nin,1 that allow to directly access to the data contained in the regular input matrices.

Suppose the previous example (block with three inputs : an int32 matrix of size [3,2], a complex scalar and a real matrix of size [4,1]).

contains three pointers, and should be viewed as arrays contained the data for the int32, the real and the complex matrices : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

```
, ,
, ,
, ,
, ,
```

to have the appropriate pointer of the data to handle and

```
, ,
, ,
,
```

to handle number, dimensions and type of regular input ports. ().

For the previous example that gives : Finally note that the regular input port registers are only accessible for reading.

- **block->nout** : Integer that gives the number of regular output ports of the block.

One can't override the index when reading sizes of output ports in the array and the index when reading data in the array with a C computational function.

The number of regular output ports can also be got by the use of the C macros .

- **block->outsz** : An array of integers of size that respectively gives the first dimensions, the second dimensions and the type of data driven by regular output ports.

Note that this array of size differs from the array and to provide full compatibilty with blocks that only use a single dimension.

Suppose that you have a block with two outputs : the first is an int32 matrix of size 3,2, the second a single complex number (matrix of size 1,1) and the last a real matrix of size 4,1.

In thescicos_model of such a block, the outputs will be defined : and the corresponding field at C computational function level will be coded as : Do the difference here in the type numbers defined at the (2,1,3) and the type numbers defined at the (84,11,10) and please report to the previous table to have the correspondence for all xcos type.

- **block->outptr** : An array of pointers of size nout,1 that allow to directly acces to the data contained in the regular output matrices.

Suppose the previous example (block with three outputs : an int32 matrix of size [3,2], a complex scalar and a real matrix of size [4,1]).

contains three pointers, and should be viewed as arrays contained the data for the int32, the real and the complex matrices : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

```
, ,
```

, ,

, ,

, ,

to have the appropriate pointer of the data to handle and

, ,

, ,

,

to handle number, dimensions and type of regular output ports. (). For the previous example that gives : Finally note that the regular output port registers must be only written for =1.

Events

- **block->nevprt** : Integer that gives the event input port number by which the block has been activated. This number is a binary coding. For i.e, if block have two event inputs ports, can take the value 1 if the block has been called by its first event input port, the value 2 if it has been called by the second event input port and 3 if it is called by the same event on both input port 1 and 2.

Note that can be -1 if the block is internally called.

One can also retrieve this number by using the C macros .

- **block->nevout** : Integer that gives the number of event output ports of the block (also called the length of the output event register).

One can't override the index when setting value of events in the output event register .

The number of event output ports can also be got by the use of the C macro .

- **block->evout** : Array of double of size nevout,1 corresponding to the output event register. That register is used to program date of events during the simulation.

When setting values in that array, you must understand that you give a delay relative to the current time of simulator :

where t_{event} is the date of the programmed event, t_{cur} is the current time in the simulator

and T_{delay} the value that must be informed in the output event register. For i.e, suppose that you want generate an event with the first event output port, 1ms after each calls of the block, then you'll use :

Note that every events generated from output event register will be asynchronous with event coming from event input port (even if you set).

The event output register must be only written for =3.

Parameters

- **block->nrpar** : Integer that gives the length of the real parameter register.

One can't override the index when reading value of real parameters in the register .

The total number of real parameters can also be got by the use of the C macro .

- **block->rpar** : Array of double of size nrpar,1 corresponding to the real parameter register. That register is used to pass real parameters coming from the scilab/xcos environment to your block model.

The C type of that array is (or C xcos type).

Suppose that you have defined the following real parameters in thescicos_model of a block : you can retrieve the previous data in the C computational function with : You can also use the C macro to get the pointer of the real parameter register. For i.e., if we define the following in an interfacing function of a xcos block : in the corresponding C computational function of that block, we'll use : Note that real parameters register is only accessible for reading.

- **block->nipar** : Integer that gives the length of the integer parameter register.

One can't override the index when reading value of integer parameters in the register .

The total number of integer parameters can also be got by the use of the C macro .

- **block->ipar** : Array of int of size nipar,1 corresponding to the integer parameter register. That register is used to pass integer parameters coming from the scilab/xcos environment to your block model.

The C type of that array is (or C xcos type).

Suppose that you have defined the following integer parameters in thescicos_model of a block : you can retrieve the previous data in the C computational function with : You can also use the C macro to get the pointer of the real parameter register.

Most of time in the xcos C block libraries, the integer register is used to parametrize the length of real parameters. For i.e. if you define the following in a block : the array of real parameters (parametrized by ipar) can be retrieved in the correspondig C computational function with : Note that integer parameters register is only accessible for reading.

- **block->nopar** : Integer that gives the number of the object parameters.

One can't override the index when accessing data in the arrays , and in a C computational function.

This value is also accessible via the C macro .

- **block->oparsz** : An array of integer of size nopar,2 that contains the dimensions of matrices of object parameters.

The first column is for the first dimension and the second for the second dimension. For i.e. if we want the dimensions of the last object parameters, we'll use the instructions : The dimensions of object parameters can be get with the following C macro : with an integer that gives the index of the object parameter, .

- **block->opartyp** : An array of integer of size nopar,1 that contains the type of matrices of object parameters.

The following table gives the correspondence for xcos type expressed in Scilab number, in C number and also corresponding C pointers and C macros used for : The type of object parameter can also be got by the use of the C macro . For i.e, if we want the C number type of the first object parameter, we'll use the following C instructions:

- **block->oparptr** : An array of pointers of size nopar,1 that allow to directly access to the data contained in the object parameter.

Suppose that you have defined in the editor a block with the following **opar** field in scicos_model : Then we have two object parameters, one is an 32-bit integer matrix with two rows and two columns and the second is a vector of complex numbers that can be understood as a matrix of size 1,3.

At the C computational function level, the instructions `*,*,*` will respectively return the values 2,1,2,3 and the instructions `*,*,*` the values 11 and 84.

will contain then two pointers, and should be viewed as arrays containing data of object parameter as shown in the following figure : For i.e., to directly access to the data, the user can use these instructions : One can also use the set of C macros :

```

, ,
, ,
, ,
,

```

to have the appropriate pointer of the data to handle ().

For the previous example that gives : Note that object parameters register is only accessible for reading.

States and work

- **block->nx** : Integer that gives the length of the continuous state register.

One can't override the index when reading or writing data in the array `*,*,*` or with a C computational function.

- **block->x** : Array of double of size nx,1 corresponding to the continuous state register.

That gives the result of the computation of the state derivative.

A value of a continuous state is readable (for i.e the first state) with the C instructions : Note that on `=4`, user can write some initial conditions in that register.

The pointer of that array can also be retrieved via the C macro `*,*,*`.

- **block->xd** : Array of double of size nx,1 corresponding to the derivative of the continuous state register.

When systems are explicitly given in terms of Ordinary Differential Equations (ODE), it can be explicitly expressed or implicitly used in the residual vector when systems are expressed in terms of Differential Algebraic Equations (DAE).

Both systems must be programmed with `*,*,*`.

For i.e the Lorentz attractor written as an ODE system with three state variables, of the form :

will be defined :

- **block->res** : Array of double of size nx,1 corresponding to Differential Algebraic Equation (DAE) residual.

It is used to write the vector of systems that have the following form :

For i.e the Lorentz attractor written as a DAE system with three state variables, will be defined :

- **block->nz** : Integer that gives the length of the discrete state register.

One can't override the index when reading data in the array with a C computational function.

This value is also accessible via the C macros .

- **block->z** : Array of double of size nz,1 corresponding to the discrete state register.

A value of a discrete state is directly readable (for i.e the second state) with the C instructions : Note that the state register should be only written for =4 and =2.

The pointer of that array can also be retrieve via the C macro .

- **block->noz** : Integer that gives the number of the discrete object states.

One can't override the index when accessing data in the arrays , and in a C computational function.

This value is also accessible via the C macro .

- **block->ozsz** : An array of integer of size noz,2 that contains the dimensions of matrices of discrete object states.

The first column is for the first dimension and the second for the second dimension. For i.e. if we want the dimensions of the last object state, we'll use the instructions : The dimensions of object discrete states can be get with the following C macro : with an integer that gives the index of the discrete object state, .

- **block->oztyp** : An array of integer of size noz,1 that contains the type of matrices of discrete object states.

The following table gives the correspondence table for xcos type expressed in Scilab number, in C number and also corresponding C pointers and C macros used for : The type of discrete object state can also be got by the use of the C macro . For i.e, if we want the C number type of the first discrete object state, we'll use the following C instructions:

- **block->ozptr** : An array of pointers of size noz,1 that allow to directly acces to the data contained in the discrete object state.

Suppose that you have defined in the editor a block with the following **godstate** field inscicos_model : Then we have two discrete object states, one is an 32-bit integer matrix with two rows and two columns and the second is a vector of complex numbers that can be understand as a matrix of size 1,3.

At the C computational function level, the instructions , , , will respectively return the values 2,1,2,3 and the instructions , the values 11 and 84.

will contain then two pointers, and should be viewed as arrays contained data of discrete object state as shown in the following figure : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

, ,

, ,

, ,

,

to have the appropriate pointer of the data to handle ().

For the previous example that gives : Finally note that the discrete objects state should be only written for =4 and =2.

- **block->work** : A free pointer to set a working array for the block.

The work pointer must be firstly allocated when = 4 and finally be free in the = 5.

Then a basic life cycle of that pointer in a C computational function should be : Note that if a block use a pointer, it will be called with =2 even if the block don't use discrete states.

The pointer of that array can also be retrieve via the C macro .

Zero crossing surfaces and modes

- **block->ng** : Integer that gives the number of zero crossing surface of the block.

One can't override the index when reading/writing data in the array with a C computational function.

The number of zero crossing surface can also be got by the use of the C macro .

- **block->g** : Array of double of size ng,1 corresponding to the zero crossing surface register.

That register is used to detect zero crossing of state variable during time domain integration.

Note that it is accessible for writting for = 9.

The pointer of that array can also be retrieve via the C macro .

- **block->nmode** : Integer that gives the number of mode of the block.

One can't override the index when reading/writing data in the array with a C computational function.

The number of mode can also be got by the use of the C macro .

- **block->mode** : Array of integer of size nmode,1 corresponding to the mode register.

That register is used to set the mode of state variable during time domain integration.

It is typically accessible for writting for = 9.

The pointer of that array can also be retrieve via the C macro .

Miscallaneous

- **block->type** : Integer that gives the type of the computational function. For C blocks, this number is equal to 4.
- **block->label** : Strings array that allows to retrieve the label of the block.

Authors

Alan Layec - INRIA

Clément David - DIGITEO

Name

C_utils — Utilities C functions

Contents

- C_utils - Utilities C functions
- • the section called “Module”
- the section called “Description”
- the section called “Authors”

Module

- xcos

Description

The header provides some utilities functions to interact with the simulator in the C computational functions.

- **void do_cold_restart();**

This function forces the solver to do a cold restart. It should be used in situations where the block creates a non smooth signal. Note that in most situations, non smooth situations are detected by zero-crossings and this function is not needed. This block is used in very exceptional situations.

- **int get_phase_simulation();**

That function returns an integer which says if the simulator is realizing time domain integration. It can returns :

- **1** : The simulator is on a discrete activation time.
- **2** : The simulator is realizing a continuous time domain integration.

- **double get_scicos_time();**

That function returns the current time of simulator.

- **int get_block_number();**

That function returns an integer : the block index in the compiled structure. Each block in the simulated diagram have a single index, and blocks are numbered from 1 to nblk (the total number of blocks in the compiled structure).

- **void set_block_error(int);**

Function to set a specific error during the simulation for the current block. If it is used, then after the execution of the computational function of the block, the simulator will end and will return an error message associated to the number given in the integer argument.

The following calls are allowed :

- **set_block_error(-1);** : the block has been called with input out of its domain,
- **set_block_error(-2);** : singularity in a block,
- **set_block_error(-3);** : block produces an internal error,

- **set_block_error(-16);** : cannot allocate memory in block.

- **void end_scicos_sim();**

A very specific function to set the current time of the simulator to the final time integration.

Only expert user should use this function.

- **void set_pointer_xproperty(int* pointer);**

This function set a vector of integer to inform the type (algebraic or differential) of the continuous state variables of the block.

- **void * scicos_malloc(size_t);**

That function must be used to do allocation of scicos pointers inside a C computational function and in particular for =4 for the work pointer .

- **void scicos_free(void *p);**

That function must be used to free scicos pointers inside a C computational function and in particular for =5 for the work pointer .

Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

2. Scilab Computational Functions

Name

sci_struct — Scicos block structure of a scilab computational function

Contents

- sci_struct - Scicos block structure of a scilab computational function
- • the section called “Module”
- the section called “Description”
- the section called “Inputs/outputs”
- the section called “Events”
- the section called “Parameters”
- the section called “States”
- the section called “Zero crossing surfaces and modes”
- the section called “Miscellaneous”
- the section called “Authors”

Module

- xcos

Description

A Scicos computational function of type 5 can be realized by the use of a Scilab function. That function doesn't really differs from all other scilab function : one can use all functions and instructions of the scilab language inside that function to do the computation.

Such a function must be written in a file with extension .sci, must be loaded inside scilab by the common loading scilab function (, , , ...) and must have two right hand side arguments and one left hand side argument, as the following calling sequence :

When the simulator is calling such a computational function, it build a scilab structure (in the previous exemple this is the named rhs/lhs arguments) from his own internal C representation of a block structure (see for more details about the C structure of scicos blocks).

That scilab structure is a scilab typed list variable that have the following fields :

Each fields are then accessible inside the scilab computational function by the use of :

Inputs/outputs

- **block.nin** : a scalar that gives the number of regular input ports. This is a read only data.
- **block.insz** : a vector of size , that gives the dimensions and types of the regular input ports.
 - : are the first dimensions.
 - : are the second dimensions.
 - : are the type of data (C coding).

This is a read only data.

- **block.inptr** : a list of size that enclosed typed matrices for regular input ports. Each element correspond to only one regular input port. Then i-th matrix of the block.inptr list will have the dimensions [,] and the type .

The data type that can be provided by regular input ports are :

- **1** : matrix of real numbers,
- **2** : matrix of complex numbers,
- **3** : matrix of int32 numbers,
- **4** : matrix of int16 numbers,
- **5** : matrix of int8 numbers,
- **6** : matrix of uint32 numbers,
- **7** : matrix of uint16 numbers,
- **8** : matrix of uint8 numbers.

This is a read only data.

- **block.nout** : a scalar that gives the number of regular output ports. This is a read only data.
- **block.outsz** : a vector of size , that gives the dimensions and types of the regular output ports.
 - : are the first dimensions.
 - : are the second dimensions.
 - : are the type of data (C coding).

This is a read only data.

- **block.outptr** : a list of size that enclosed typed matrices for regular output ports. Each element correspond to only one regular output port. Then i-th matrix of the block.outptr list will have the dimensions [,] and the type .

The data type that can be provided by regular output ports are :

- **1** : matrix of real numbers,
- **2** : matrix of complex numbers,
- **3** : matrix of int32 numbers,
- **4** : matrix of int16 numbers,
- **5** : matrix of int8 numbers,
- **6** : matrix of uint32 numbers,
- **7** : matrix of uint16 numbers,
- **8** : matrix of uint8 numbers.

Values of regular output ports will be saved in theC structure of the block only for =6 and =1.

Events

- **block.nevp** : a scalar given the event input port number (binary coding) which have activated the block. This is a read only data.
- **block.nevout** : a scalar given the number of output event port of the block. This is a read only data.
- **block.evout** : a vector of size corresponding to the register of output event. Values of output event register will be saved in theC structure of the block only for =3.

Parameters

- **block.nrpar** : a scalar given the number of real parameters. This is a read only data.
- **block.rpar** : a vector of size corresponding to the real parameter register. This is a read only data.
- **block.nipar** : a scalar given the number of integer parameters. This is a read only data.
- **block.ipar** : a vector of size corresponding to the integer parameter register. This is a read only data.
- **block.nopar** : a scalar given the number of object parameters. This is a read only data.
- **block.oparsz** : a matrix of size , that respectively gives the first and the second dimension of object parameters. This is a read only data.
- **block.opartyp** : a vector of size given the C coding type of data. This is a read only data.
- **block.opar** : a list of size given the values of object parameters. Each element of can be either a typed matrix or a list. Only matrix that encloses numbers of type real, complex, int32, int16, int8, uint32, uint16 and uint8 are allowed, all other types of scilab data will be enclosed in a sub-list. This is a read only data.

States

- **block.nz** : a scalar given the number of discrete state for the block. This is a read only data.
- **block.z** : a vector of size corresponding to the discrete state register. Values of discrete state register will be saved in theC structure of the block only for =4, =6, =2 and =5.
- **block.noz** : a scalar that gives the number of discrete object state. This is a read only data.
- **block.ozsz** : a matrix of size , that respectively gives the first and the second dimension of discrete object state. This is a read only data.
- **block.oztyp** : a vector of size given the C coding type of data.
- **block.oz** : a list of size given the values of discrete object states. Each element of can be either a typed matrix or a list. Only matrix that encloses numbers of type real, complex, int32, int16, int8, uint32, uint16 and uint8 are allowed, all other types of scilab data will be enclosed in a sub-list. Values of discrete object state will be saved in theC structure of the block only for =4, =6, =2 and =5.
- **block.nx** : a scalar given the number of continuous state for the block. This is a read only data.
- **block.x** : a vector of size given the value of the continuous state register. Values of the continuous state register will be saved in theC structure of the block only for =4, =6 and =2.

- **block.xd** : a vector of size given the value of the derivative continuous state register. Values of the derivative continuous state register will be saved in theC structure of the block only for =4, =6, =0 and =2.
- **block.res** : a vector of size corresponding to the Differential Algebraic Equation (DAE) residual. Values of that register will be saved in theC structure of the block only for =0, and =10.

Zero crossing surfaces and modes

- **block.ng** : a scalar given the number of zero crossing surfaces for the block. This is a read only data.
- **block.g** : a vector of size corresponding to the zero crossing register. Values of that register will be saved in theC structure of the block only for =9.
- **block.nmode** : a scalar given the number of mode for the block. This is a read only data.
- **block.mode** : a vector of size that corresponds to the mode register. Values of that register will be saved in theC structure of the block only for =9, with =1.

Miscellaneous

- **block.type** : a scalar given the type of the block. This is a read only data.
- **block.label** : a string given the label of the block. This is a read only data.

Authors

Alan Layec - INRIA

3. Utilities Functions

Name

curblock — Return the current called xcos block during the simulation

Module

- xcos

```
blk=curblock()
```

Parameters

- **blk** : the current block number in the compiled structure.

Authors

Ramine Nikoukhah - INRIA

Name

getblocklabel — Get the label of a scicos block

Module

- xcos

```
[label]=getblocklabel(blk)
```

Parameters

- **blk** : Integer parameter. Set the index of a block (in the compiled structure).
- **label** : String parameter. Gives the string of the label of the block numbered blk.

Name

getscicosvars — Supervisor utility function

```
[myvar]=getscicosvars(str) [myvar]=getscicosvars([str1;str2;...])
```

Module

- xcos

Description

That utility function is used to retrieve working arrays of Scicos simulator and compiler during simulation.

It can be used inside a Scilab block to get information of all type of blocks. That function is very useful to debug diagrams and to do prototypes of simulations.

```
[myvar]=getscicosvars(str)
[myvar]=getscicosvars([str1;str2;...])
```

Parameters

- **str, str1, str2, ...** : That parameter can be a string or a matrix of string. The following entries are allowed :

"x" : get the continuous state register.

"nx" : get the length of the continuous state register.

"xptr" : get the pointers register of the continuous state register.

"zcptr" : get the pointers register of the zero-crossing surfaces register.

"z" : get the discrete state register.

"nz" : get the length of the continuous state register.

"zptr" : get the pointers register of the discrete state register.

"noz" : get the number of elements of the discrete object state list.

"oz" : get the discrete object state list.

"ozsz" : get the size of the elements of the discrete object state list.

"oztyp" : get the type of the elements of the discrete object state list.

"ozptr" : get the pointers register of the discrete object state list.

"rpar" : get the real parameter register.

"rpptr" : get the pointers register of the real parameter register.

"ipar" : get the integer parameter register.

"ipptr" : get the pointers register of the integer parameter register.

"opar" : get the object parameter list.

"oparsz" : get the size of the elements of the object parameter list.

"opartyp" : get the type of the elements of the object parameter list.

"opptr" : get the pointers register of the object parameter list.

"outtb" : get the output register.

"inpptr" : get the pointers register of the number of regular input ports.

"outptr" : get the pointers register of the number of regular output ports.

"inplnk" : get the pointers register of the links connected to regular input ports.

"outlnk" : get the pointers register of the links connected to regular output ports.

"subs" : not used

"tevt" : get the current date register of the agenda.

"evtspt" : get the current event register of the agenda.

"pointi" : get the next event to be activated.

"iord" : get the vector of blocks activated at the start of the simulation.

"oord" : get the vector of blocks whose outputs affects computation of continuous state derivatives.

"zord" : get the vector of blocks whose outputs affects computation of zero-crossing surfaces.

"funtyp" : get the vector of type of computational functions.

"ztyp" : get the pointers vector for blocks which use zero-crossing surfaces.

"cord" : get the vector of blocks whose outputs evolve continuously.

"ordclk" : get the matrix associated to blocks activated by output activation ports.

"clkptr" : get the pointers vector for output activation ports.

"ordptr" : get the pointers vector to ordclk designating the part of ordclk corresponding to a given activation.

"critev" : get the vector of the critical events.

"mod" : get the vector pointers of block modes.

"nmod" : get the length of the vector pointers of block modes.

"iz" : get the register that store pointers of block- > work.

"izptr" : get the pointers vector of the register that store C pointers of block- > work.

"nblk" : get the number of block.

"outbptr" : get the register that store C pointers of outtb.

"outbpsz" : get the register that store the size of the elements of outtb.

"outbtyp" : get the register that store the type of the elements of outtb.

"nlbk" : get the number of output.

"ncord" : get the number of blocks whose outputs evolve continuously.

"nordptr" : get the number of blocks whose outputs evolve by activation.

"iwa" : n.d.

"blocks" : get a scilab list that contains all block structures contains in the diagram.

"ng" : get length of the zero-crossing surfaces register.

"g" : get the zero-crossing surfaces register.

"t0" : get the current time of the simulation.

"tf" : get the final time of the simulation.

"Atol" : get the integrator absolute tolerance for the numerical solver.

"rtol" : get the integrator relative tolerance for the numerical solver.

"ttol" : get the tolerance on time of the simulator.

"deltat" : get the maximum integration time interval.

"hmax" : get the maximum step size for the numerical solver.

"nelem" : get the number of elements in outtb.

"outtb_elem" : get the vector of the number of elements in outtb.

- **myvar** : That output parameter can be an int32 matrix, a double matrix or a Tlist. This is given by the input parameter.

See Also

- **DEBUG_SCICOS** - Debug block (Scicos Block)

Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

Name

phase_simulation — Get the current simulation phase

```
[psim]=phase_simulation()
```

Module

- xcos

Description

That function says if the Scicos simulator is realizing the time domain integration.

```
[psim]=phase_simulation()
```

Parameters

- **psim** : get the current phase of the simulation
- **1** : The simulator is on a discrete activation time.
- **2** : The simulator is realizing a continuous time domain integration.

Name

pointer_xproperty — Get the type of a continuous time state variable

```
[xprop]=pointer_xproperty
```

Module

- xcos

Description

This function returns a vector that informs the type (algebraic or differential) of the continuous state variables of a block.

```
[xprop]=pointer_xproperty
```

Parameters

- **xprop**

The value gives the type of the states :

- -1 : an algebraic state.
- 1 : a differential state.

See Also

- set_xproperty - Set the type of a continuous time state variable (Scilab Function)

Name

scicos_time — Returns the current time during simulation

Module

- xcos

```
t=scicos_time()
```

Parameters

- **t** : that is the current simulated time returned in real number.

Name

set_blockerror — set the block error number

```
set_blockerror(n)
```

Module

- xcos

Description

Function to set a specific error during the simulation for the current block. If it is used, then after the execution of the computational function of the block, the simulator will end and will return an error message associated to the number given in argument.

```
set_blockerror(n)
```

Parameters

- **n** : an error number. The following calls are allowed :
 - **set_blockerror(-1)**
the block has been called with input out of its domain
 - **set_blockerror(-2)**
singularity in a block
 - **set_blockerror(-3)**
block produces an internal error
 - **set_blockerror(-16)**
cannot allocate memory in block

Authors

Alan Layec - INRIA

Name

set_xproperty — Set the type of a continuous time state variable

```
set_xproperty(xprop)
```

Module

- xcos

Description

This function set a vector to inform the type (algebraic or differential) of the continuous state variables of a block.

```
set_xproperty(xprop)
```

Parameters

- **xprop**

The value gives the type of the states :

- -1 : an algebraic state.
- 1 : a differential state.

See Also

- pointer_xproperty - Get the type of a continuous time state variable (Scilab Function)

Chapter 9. Scilab Data Structures

1. Blocks

Name

scicos_block — Define a block structure

Module

- xcos

block

Basic structure that define a xcos block.

That structure includes fields graphics, model, gui and doc.

Size : 5.

Type : scilab list.

- **graphics**

Scilab object including graphical information concerning the features of the block.

Size : 14.

Type : scilab list.

- **model**

Scilab list that contains the features of the block used for the compilation.

Size : 23.

Type : Scilab list.

- **gui**

The name of the Scilab GUI function associated with the block.

Size : 1.

Type : string.

- **doc**

Field used for documentation of the block Size : 1.

Type : string.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_block.sci

Name

scicos_graphics — Define a graphics structure

Module

- `xcos`

graphics

Scilab object including graphical information concerning the features of the block.

Size : 14.

Type : scilab list.

- **orig**

Vector $[x_0, y_0]$, where x_0 is the x coordinate of the block origin and y_0 is the y coordinate of the block origin.

$[x_0, y_0]$ is the coordinate of down-left point of the block shape.

Size : 2.

Type : row vector of real.

- **sz**

Vector $[w, h]$, where w is the block width and h the block height.

Size : 2.

Type : row vector of real.

- **flip**

Set the block orientation. If true the input ports are on the left of the box and output ports are on the right.

If false the input ports are on the right of the box and output ports are on the left.

Size : 1.

Type : boolean.

- **theta**

Set the angle of the Scicos object.

This value is in degree and is included in $[-360, 360]$.

Size : 1.

Type : real.

- **exprs**

Strings including formal expressions used in the dialog box of the block.

Size : number of formal expressions.

Type : column vector of strings.

- **pin**

Vector. pin(i) is the number of the link connected to the ith regular input port (counting from one), or 0 if this port is not connected.

Size : number of regular input ports.

Type : column vector of integers.

- **pout**

Vector. pout(i) is the number of the link connected to the ith regular output port (counting from one), or 0 if this port is not connected.

Size : number of regular output ports.

Type : column vector of integers.

- **pein**

Vector. pein(i) is the number of the link connected to the ith event input port (counting from one), or 0 if this port is not connected.

Size : number of events input ports.

Type : column vector of integers.

- **peout**

Vector. peout(i) is the number of the link connected to the ith event output port (counting from one), or 0 if this port is not connected.

Size : number of events output ports.

Type : column vector of integers.

- **gr_i**

Strings including Scilab graphics expressions for customizing the block graphical aspect. This field may be set with Icon sub_menu.

Size : -.

Type : column vector of strings.

- **id**

A string including an identification for the block. The string is displayed under the block in the diagram.

Size : 1.

Type : string.

- **in_implicit**

A vector of strings including 'E' or 'I'.

'E' and 'I' stand respectively for explicit and implicit port, and this vector indicates the nature of each input port. For regular blocks (not implicit), this vector is empty or contains only "E".

Size : nul or number of regular input ports.

Type : column vector of strings.

- **out_implicit**

A vector of strings including 'E' or 'I'.

'E' and 'I' stand respectively for explicit and implicit port, and this vector indicates the nature of each output port. For regular blocks (not implicit), this vector is empty or contains only "E".

Size : nul or number of regular output ports.

Type : column vector of strings.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_graphics.sci

Name

scicos_model — Define a model structure

Module

- xcOS

model

Scilab list that contains the features of the block used for the compilation.

Size : 23.

Type : Scilab list.

- **sim**

A list containing two elements. The first element is a string containing the name of the computational function (C, Fortran, or Scilab). The second element is an integer specifying the type of the computational function. Currently type 4 and 5 are used, but older types continue to work to ensure backward compatibility.

For some older case, sim can be a single string and that means that the type is supposed to be 0.

Size : 2.

Type : Scilab list.

- **in**

A vector specifying the number and size of the first dimension of regular input ports indexed from top to bottom of the block. If no input port exist in==[].

The size can be negative, equal to zero or positive :

- If a size is less than zero, the compiler will try to find the appropriate size.
- If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector
- If a size is greater than zero, then the size is explicitly given.

Size : number of regular input ports.

Type : column vector of integer numbers.

- **in2**

A vector specifying the second dimension of regular input ports indexed from top to bottom of the block.

in with in2 formed then the regular input sizes matrix.

For compatibility, this dimension can stay empty ([]).

That means that the dimensions of input ports will be in,1 The size can be negative, equal to zero or positive :

- If a size is less than zero, the compiler will try to find the appropriate size.
- If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector.
- If a size is greater than zero, then the size is explicitly given.

Size : number of regular input ports.

Type : column vector of integer numbers.

- **intyp**

A vector specifying the types of regular input ports.

Its sizes is equal to the sizes of in.

The types of regular input ports can be :

- 1 real matrix,
- 2 complex matrix,
- 3 int32 matrix,
- 4 int16 matrix,
- 5 int8 matrix,
- 6 uint32 matrix,
- 7 uint16 matrix,
- 8 uint8 matrix.

Size : number of regular input ports.

Type : column vector of integer numbers.

- **out**

A vector specifying the number and size of the first dimension of regular output ports indexed from top to bottom of the block. If no output port exist out==[].

The size can be negative, equal to zero or positive :

- If a size is less than zero, the compiler will try to find the appropriate size.
- If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector
- If a size is greater than zero, then the size is explicitly given.

Size : number of regular output ports.

Type : column vector of integer numbers.

- **out2**

A vector specifying the second dimension of regular output ports indexed from top to bottom of the block.

out with out2 formed then the regular output sizes matrix.

For compatibility, this dimension can stay empty ([]). That means that the dimensions of output ports will be out,1 That dimension can be negative, equal to zero or positive :

- If a size is less than zero, the compiler will try to find the appropriate size.
- If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector.
- If a size is greater than zero, then the size is explicitly given.

Size : number of regular output ports.

Type : column vector of integer numbers.

- **outtyp**

A vector specifying the types of regular output ports.

Its sizes is equal to the sizes of out.

The types of regular output ports can be :

- 1 real matrix,
- 2 complex matrix,
- 3 int32 matrix,
- 4 int16 matrix,
- 5 int8 matrix,
- 6 uint32 matrix,
- 7 uint16 matrix,
- 8 uint8 matrix.

Size : number of regular output ports.

Type : column vector of integer numbers.

- **evtin**

A vector specifying the number and sizes of activation inputs. Currently activation ports can be only of size one.

If no event input port exists evtin must be equal to [].

Size : number of input event ports.

Type : column vector of integer numbers.

- **evtout**

A vector specifying the number and sizes of activation outputs.

Currently activation ports can be only of size one.

If no event output port exists evtout must be equal to [].

Size : number of output event ports.

Type : column vector of integer numbers.

- **state**

Vector containing initial values of continuous-time state.

Must be [] if no continuous state.

Size : number of continuous-time state.

Type : column vector of real numbers.

- **dstate**

Vector containing initial values of discrete-time state.

Must be [] if no discrete state.

Size : number of discrete-time state.

Type : column vector of real numbers.

- **odstate**

List containing initial values of objects state.

Must be list() if no objects state.

Objects state can be any types of scilab variable.

In the computational function case of type 4 (C blocks) only elements containing matrix of real, complex, int32, int16, int8, uint32, uit16 and uint8 will be correctly provided for readind/writing.

Size : number of objects state.

Type : scilab list of scilab objects.

- **rpar**

The vector of floating point block parameters.

Must be [] if no floating point parameters.

Size : number of real parameters.

Type : column vector of real numbers.

- **ipar**

The vector of integer block parameters.

Must be [] if no integer parameters.

Size : number of integer parameters.

Type : column vector of integer numbers.

- **opar**

List of objects block parameters. Must be list() if no objects parameters.

Objects parameters can be any types of scilab variable.

In the computational function case of type 4 (C blocks) only elements containing matrix of real, complex, int32, int16, int8, uint32, uit16 and uint8 will be correctly provided for reading.

Size : number of objetcs parameters.

Type : list of scilab object.

- **blocktype**

Character that can be set to 'c' or 'd' indifferently for standard blocks. 'x' is used if we want to force the computational function to be called during the simulation phase even if the block does not contribute to computation of the state derivative.

'l', 'm' and 's' are reserved. Not to be used.

Size : 1.

Type : Character.

- **firing**

Vector of initial event firing times of size equal to the number of activation output ports (see evout). It contains output initial event dates (Events generated before any input event arises). Negative values stands for no initial event on the corresponding port. Size : number of output event ports.

Type : column vector of real numbers.

- **dep_ut**

Boolean vector [dep_u, dep_t].

- **dep_u**

true if block is always active.

(output depends continuously of the time)

- **dep_t**

true if block has direct feed-through, i.e., at least one of the outputs depends directly (not through the states) on one of the inputs. In other words, when the computational function is called with flag 1, the value of an input is used to compute the output.

Size : 2.

Type : Boolean vector.

- **label**

String that defines a label. It can be used to identify a block in order to access or modify its parameters during simulation.

Size : 1.

Type : string.

- **nzcross**

Number of zero-crossing surfaces.

Size : Number of zero-crossing surfaces.

Type : column vector of integer numbers.

- **nmode**

Length of the mode register. Note that this gives the size of the vector mode and not the total number of modes in which a block can operate in. Suppose a block has 3 modes and each mode can take two values, then the block can have up to $2^3=8$ modes.

Size : Number of modes.

Type : column vector of integer numbers.

- **equations**

Used in case of implicit blocks.

Data structure of type modelica which contains modelica code description if any. That list contains four entries :

- **model**

a string given the name of the file that contains the modelica function.

- **inputs**

a column vector of strings that contains the names of the modelica variables used as inputs.

- **outputs**

a column vector of strings that contains the names of the modelica variables used as outputs.

- **parameters**

a list with two entries. The first is a vector of strings for the name of modelica variable names used as parameters and the second entries is a list that contains the value of parameters.

Names of modelica states can also be informed with parameters. In that case a third entry is used to do the difference between parameters and states.

For i,e : `mo.parameters=list(['C','v'],list(C,v),[0,1])` means that 'C' is a parameter(0) of value C, and 'v' is a state(1) with initial value v.

Size : 5.

Type : scilab list.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_model.sci

2. Compilation/Simulation

Name

scicos_cpr — Compiled Scicos structure

Module

- `xcos`

cpr

The Scilab object `cpr` contains the result of the compilation.

That structure includes fields `state`, `sim`, `cor` and `corinv`.

Size : 5.

Type : scilab list.

- **state**

Scilab typed list of type `xcs`. It contains all the states of the model, that is, everything than can evolve during the simulation.

`state` contains `x`, `z`, `oz`, `iz`, `tevts`, `evtspt`, `pointi` and `outtb`.

Size : 9.

Type : scilab tlist.

- **sim**

Scilab typed list of type `scs`. It contains static arrays coming from the result of the compilation. That arrays does not evolve during the simulation.

Size : 33.

Type : scilab tlist.

- **cor**

It is a list with same recursive structure as `scs_m`. Each leaf contains the index of associated block in `sim` data structure. Size : number of objects in `scs_m`.

Type : scilab list.

- **corinv**

`corinv(i)` is the path of `i` th block defined in `sim` data structure in the `scs_m` data structure.

Size : number of blocks in the compiled structre.

Type : scilab list.

File content

- `SCI/modules/scicos/macros/scicos_scicos/scicos_cpr.sci`

Name

scicos_sim — Define a sim structure

Module

- xcos

sim

Scilab typed list of type scs. It contains static arrays coming from the result of the compilation. That arrays does not evolve during the simulation.

Size : 33.

Type : scilab tlist.

- **funcs**

A list containing names of the computational functions or scilab functions.

Size : number of blocks.

Type : list of strings and/or scilab function.

- **xptr**

A vector pointer to the continuous time state register x. The continuous-time state of block i is state.x(sim.xptr(i):sim.xptr(i+1)-1).

Size : number of blocks + 1.

Type : column vector of integers.

- **zptr**

A vector pointer to the discrete time state register z. The discrete-time state of block i is state.z(sim.zptr(i):sim.zptr(i+1)-1).

Size : number of blocks + 1.

Type : column vector of integers.

- **ozptr**

A vector pointer to the object discrete state register oz. The object discrete state of block i is state.oz(sim.ozptr(i):sim.ozptr(i+1)-1).

Size : number of blocks + 1.

Type : column vector of integers.

- **zcptr**

A vector pointer to the zero-crossing surfaces.

register. That vector gives by block the used number of the zero-crossing.

Size : number of blocks + 1.

Type : column vector of integers.

- **inpptr**

$(\text{sim.inpptr}(i+1)-\text{sim.inpptr}(i))$ gives the number of regular input ports of the i block.

$\text{inpptr}(i)$ points to the beginning of i th block inputs within the indirection table inplnk .

Size : number of blocks + 1.

Type : column vector of integers.

- **outptr**

$(\text{sim.outptr}(i+1)-\text{sim.outptr}(i))$ gives the number of regular ouyput ports of the i block.

$\text{outptr}(i)$ points to the beginning of i th block outputs within the indirection table outlnk .

Size : number of blocks + 1.

Type : column vector of integers.

- **inplnk**

$(\text{cpr.sim.inplnk}(\text{cpr.sim.inpptr}(i)-1+j))$ is the index of the link connected to the j th input port of the i th block where j goes from 1 to $(\text{cpr.sim.inpptr}(i+1)-\text{cpr.sim.inpptr}(i))$.

Size : total number of regular input port.

Type : column vector of integers.

- **outlnk**

$(\text{cpr.sim.outlnk}(\text{cpr.sim.outptr}(i)-1+j))$ is the index of the link connected to the j th output port of the i th block where j goes from 1 to $(\text{cpr.sim.outptr}(i+1)-\text{cpr.sim.outptr}(i))$.

Size : total number of regular output port.

Type : column vector of integers.

- **rpar**

Vector of real parameters that is obtained by concatenating the real parameters registers of all the blocks.

Size : total number of real parameters.

Type : column vector of real numbers.

- **rpptr**

A vector pointer to the real parameters register rpar . The real parameters of block i are $\text{sim.rpar}(\text{sim.rpptr}(i):\text{sim.rpptr}(i+1)-1)$.

Size : number of blocks + 1.

Type : column vector of integer.

- **ipar**

Vector of integer parameters that is obtained by concatenating the integer parameters registers of all the blocks.

Size : total number of integer parameters.

Type : column vector of integer.

- **ipptr**

A vector pointer to the integer parameters register ipar. The integer parameters of block i are `sim.ipar(sim.ipptr(i):sim.ipptr(i+1)-1)`.

Size : number of blocks + 1.

Type : column vector of real numbers.

- **opar**

List of object parameters that is obtained by concatenating the list of object parameters of all the blocks.

Size : total number of object parameters.

Type : list of scilab objects.

- **opptr**

A vector pointer to the object parameters list opar. The object parameters of block i are `sim.opar(sim.opptr(i):sim.opptr(i+1)-1)`.

Size : number of blocks + 1.

Type : column vector of integers.

- **clkptr**

A vector pointer to output activation ports.

`(cpr.sim.clkptr(i):cpr.sim.clkptr(i+1)-1)` gives the number of output event ports of the block i.

Size : number of blocks + 1.

Type : column vector of integers.

- **ordptr**

A vector pointer to ordclk designating the part of ordclk corresponding to a given activation.

`(cpr.sim.ordptr(i):cpr.sim.ordptr(i+1)-1)` points to the region within ordclk indicates the number of blocks activated by the output event ports numbered i.

Size : number of sources of activation + 1.

Type : column vector of integers.

- **execlk**

Unused.

Size : - Type : matrix of real.

- **ordclk**

A matrix associated to blocks activated by output activation ports. The first column contains the block number, and the second, the event code by which the block should be called.

Size : total number of blocks summed by source of activations.

Type : matrix of integers.

- **cord**

A matrix associated to always active blocks.

The first column contains the block number, and the second, the event code by which the block should be called.

Size : ncord.

Type : matrix of integers.

- **oord**

Subset of cord. Blocks of that matrix have outputs which affect computation of continuous state derivatives.

Size : noord.

Type : matrix of integers.

- **zord**

Subset of zord. Blocks of that matrix have outputs which affect computation of zero-crossing surfaces.

Size : nzord.

Type : matrix of integers.

- **critev**

A vector of size equal to the number of activations and containing zeros and ones. The value one indicates that the activation is critical in the sense that the continuous-time solver must be cold restarted.

Size : number of source of activation.

Type : column vector of integers.

- **nb**

Number of blocks. Note that the number of blocks may differ from the original number of blocks in the diagram because c_pass2 may duplicate some conditional blocks.

Size : 1.

Type : integer.

- **ztyp**

A vector of size equal to the number of blocks.

A 1 entry indicates that the block may have zero-crossings, even if it doesn't in the context of the diagram. Usually not used by the simulator.

Size : number of blocks.

Type : column vector of integers.

- **nblk**

Not used. Set to nb.

Size : 1 Type : integer.

- **ndcblk**

Not used.

Size : - Type : integer.

- **subscr**

Not used.

Size : 0 Type : empty real.

- **funtyp**

A vector of size equal to the number of blocks indicating the type of the computational function of the block. Block type can be 0 through 5.

Currently only type 4 (C language) and type 5 (Scilab language) computational functions should be used. But older blocks can also be used.

Size : number of blocks.

Type : column vector of integer.

- **iord**

A matrix associated to blocks that must be activated at the start of the simulation. This includes blocks inheriting from constant blocks and always active blocks.

Size : niord.

Type : matrix of integers.

- **labels**

A string vector of size equal to the number of blocks containing block labels.

Size : numbers of blocks.

Type : column vector of strings.

- **modptr**

A vector pointer to the block modes.

Size : number of blocks + 1.

Type : column vector of integer.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_sim.sci

Name

scicos_state — Define a state structure

Module

- **xcos**

state

Scilab typed list of type xcs. It contains all the states of the model, that is, everything than can evolve during the simulation.

state contains x, z, oz, iz, tevts, evtspt, pointi and outtb.

Size : 9.

Type : scilab tlist.

- **x**

The continuous-time state register, which is obtained by concatenating the continuous-time states of all the blocks.

Size : total of all the size of continuous-time state registers.

Type : column vector of real numbers.

- **z**

The discrete-time state register, which is obtained by concatenating the discrete-time states of all the blocks.

Size : total of all the size of discrete-time state registers.

Type : column vector of real number.

- **oz**

The list of the object discrete-time state, which is obtained by concatenating the object discrete-time states of all the blocks.

Size : total of all the size of object state.

Type : list of scilab object.

- **iz**

Vector of size equal to the number of blocks.

That vector is used to store pointers of the working state register (work). If a block needs to allocate memory at initialization (flag 4), the associated pointer is saved here.

Size : number of blocks.

Type : column vector of real numbers.

- **tevts**

Vector of size equal to the number of activation sources. It contains the scheduled times for programmed activations in `evtspt`.

Size : number of activation sources.

Type : column vector of integers.

- **evtspt**

Vector of size equal to the number of activation sources. It is an event scheduler.

Size : number of activation sources.

Type : column vector of integers.

- **pointi**

The number of the next programmed event.

Size : 1.

Type : integer.

- **outtb**

Scilab list containing all output registers of blocks. Each element of that list contains typed matrix-based data.

Size : number of regular output ports.

Type : list of scilab matrix.

File content

- `SCI/modules/scicos/macros/scicos_scicos/scicos_state.sci`

3. Diagram

Name

scicos_diagram — Define a scs_m structure

Module

- xcos

diagram

Size : 4.

Type : scilab list.

- **props**

Diagram properties.

This entry contains various informations such some main diagram initials values.

This variable is a tlist of type and contains wpar, title, tol, tf, context, options and doc.

Size : 11.

Type : Scilab tlist of type .

- **objs**

List of objects included in the Scicos diagram.

The objects used in scicos are block ,link and Text.

The objects can also be deleted object data structure.

Deleted object data structure is marked list('Deleted').

Size : total number of objects in the diagram.

Type : Scilab tlist of type , or Text.

- **version**

A string that gives the version of the Scicos diagram.

This is used to provide compatibility with old diagram.

Note that you can get the current version of Scicos by using the entry 'About scicos' in the help menu or by using the function `get_scicos_version()`.

Size : 1.

Type : String.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_diagram.sci

Name

scicos_params — Define a param structure

Module

- **xcos**

params

Size : 11.

Type : scilab list.

- **wpar**

This vector is not currently used.

It may be used in the future to code window sizes of the editor.

Size : 6.

Type : column vector or real.

- **title**

Vector of character strings, where the first one is the diagram title and default name of save file name, and the second one is the path of the directory of the file name.

Size : 2.

Type : row vector of strings.

- **tol**

A vector containing simulation parameters including various tolerances used by the solver:

- **atol**

Integrator absolute tolerance for the numerical solver.

- **rtol**

Integrator relative tolerance for the numerical solver.

- **ttol**

Tolerance on time.

If an integration period is less than ttol, the numerical solver is not called.

- **deltat**

Maximum integration time interval.

If an integration period is larger than deltat, the numerical solver is called more than once in such a way that for each call the integration period remains below deltat

- **scale**

Real-time scaling; the value 0 corresponds to no real-time scaling.

It associates a Scicos simulation time to the real time in seconds.

A value of 1 means that each Scicos unit of time corresponds to one second.

- **solver**

Choice of numerical solver.

The value 0 implies an ODE solver and 100 implies a DAE solver.

- **hmax**

Maximum step size for the numerical solver.

0 means no limit.

Size : 7.

Type : column vector of real.

- **tf**

Final time simulation.

The simulation stops at this time.

The default value is 100000.

Size : 1.

Type : real.

- **context**

A vector of strings containing Scilab instructions defining Scilab variables to be used inside block's dialog box as symbolic parameters.

All valid Scilab instructions can be used and also comments.

Size : number of lines of the context.

Type : column vector of strings.

- **void1**

unused field.

Size : -.

Type : -.

- **options**

Scilab object of type scsopt defining graphical properties of the editor such as background color and link color.

The fields are the following:

- **3D**

A list with two entries. The first one is a boolean indicating whether or not blocks should have 3D aspect.

The second entry indicates the color in the current colormap to be used to create the 3D effect.

The default is 33 which corresponds to gray added by Scicos to the standard colormap, which contains 32 colors.

The default value is `list(%t,33)`.

- **Background**

Vector with two entries: background and foreground colors.

The default value is `[8,1]`.

- **link**

Default link colors for regular and activation links.

These colors are used only at link construction.

Changing them does not affect already constructed links.

The default value is `[1,5]`, which corresponds to black and red if the standard Scilab colormap is used.

- **ID**

A list of two vectors including font number and sizes.

The default value is `[5,1],[4,1]`.

- **Cmap**

An $n,3$ matrix containing RGB values of colors to be added to the colormap.

The default value is, `[0.8,0.8,0.8]` i.e., the color gray.

Size : 6.

Type : scilab tlist of type scsopt.

- **void2**

unused field.

Size : -.

Type : -.

- **void3**

unused field.

Size : -.

Type : -.

- **doc**

User defined diagram documentation structure.

Size : 1.

Type : Strings.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_params.sci

4. Links

Name

scicos_link — Define a link structure

Module

- **xcos**

link

Size : 8.

Type : scilab list.

- **xx**

Vector of x coordinates of the link path.

A link is defined as a polyline line.

Size : number of points of the link.

Type : column vector of real numbers.

- **yy**

Vector of y coordinates of the link path.

A link is defined as a polyline line.

Size : number of points of the link.

Type : column vector of real numbers.

- **id**

Character string, the link identification.

Size : 1.

Type : string.

- **thick**

Vector of size two defining line thickness.

Size : 2.

Type : row vector of integers.

- **ct**

The first entry of this vector designates the color, and the second, the nature of the link. The second entry is 1 for a regular link, -1 for an activation link, and 2 for an implicit link.

Size : 2.

Type : row vector of integers.

- **from**

Vector of size three including the block number, port number, and port type (0 for output, 1 for input) at the origin of the link. Note that the third entry may be 1 if the link is implicit; otherwise it is zero.

Size : 3.

Type : row vector of integers.

- **to**

Vector of size three including the block number, port number, and port type (0 for output, 1 for input) at the destination of the link. Note that the third entry may be 1 if the link is implicit; otherwise it is zero.

Size : 3.

Type : row vector of integers.

File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_link.sci

Chapter 10. Scilab Utilities Functions

Name

buildouttb — Build of the sublist %cpr.state.outtb

```
[outtb]=buildouttb(lnksz, lnktyp)
```

Module

- xcoss

Description

Build an initialized outtb list.

```
[outtb]=buildouttb(lnksz, lnktyp)
```

Parameters

- **outtb** : a list of size n.
- **lnksz** : That parameter gives the size of Scilab object in outtb.
This matrix of integer or real numbers can have a n,2 or 2,n size.
- **lnktyp** : That parameters gives the type of Scilab object in outtb :
1 : double
2 : complex
3 : int32
4 : int16
5 : int8
6 : uint32
7 : uint16
8 : uint8
else : double
This matrix of integer or real numbers can have a n,1 or 1,n size.

Authors

Alan Layec - INRIA

Name

create_palette — Palette generator

```
[routines,IntFunc] = create_palette(Path)
```

Module

- xcOS

Description

This function generates a palette if Path is a string indicating the directory in which the interfacing functions of the blocks are to be found. If Path is not present or if it is %, then standard scicos palettes are regenerated. If %f, then only IntFunc (list of interfacing functions) is returned. List of routines is empty in this case.

```
[routines,IntFunc] = create_palette(Path)
```

Parameters

- **Path** : a string given the directory path that contains Scicos blocks interfacing functions.
- **routines** : a vector of strings that contains names of computational functions used in scicos blocks.
- **IntFunc** : a vector of strings that contains names of interfacing functions used in scicos blocks.

File content

- SCI/modules/scicos/macros/scicos_utils/create_palette.sci

Name

get_scicos_version — Get the current Scicos version

```
scicos_ver = get_scicos_version
```

Module

- xcOS

Description

This function is used to know the current version number of Scicos.

```
scicos_ver = get_scicos_version
```

Parameters

- **scicos_ver** : a string given the current number version of Scicos.

Example

```
//Get the scicos version  
get_scicos_version()
```

File content

- SCI/modules/scicos/macros/scicos_utils/get_scicos_version.sci

Authors

Alan Layec - INRIA

Name

scicos_debug — Set the level of the Scicos debugging

```
scicos_debug(level) level=scicos_debug()
```

Module

- xcOS

Description

This Scilab function is used to set the debug level of a Scicos simulation.

One can use it in the "Calc" mode of the Scicos editor or as an instruction in a Scilab block or in an interfacing function.

```
scicos_debug(level)  
level=scicos_debug()
```

Parameters

- **level** : set/get the current level of the Scicos simulation debugging.
 - **0** : no debugging.
 - **1** : light debugging information printout.
 - **2** : more information printout and execution of Debug Block if any in diagram.
 - **3** : silent debugging mode (no information printout) and execution of Debug Block if any in diagram.

See Also

- DEBUG_SCICOS - Debug block (Scicos Block)

Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

Name

var2vec — Transform a scilab variable in a vector of double

```
[vec]=var2vec(var)
```

Module

- xcOS

Description

var2vec / vec2var functions are used inside the interfacing functions of Scilab blocks to give the possibility to the user to handle Scilab objects with the real parameter (rpar) and with the discrete state register (z).

```
[vec]=var2vec(var)
```

Parameters

- **var** : Input parameter. Can be any types of Scilab objects.
- **vec** : Output parameter. A vector of real numbers.

Example

```
-->a=list("cos",[1.1,2])
a =
a(1)
cos
a(2)
1.1      2.
-->b=var2vec(a)
b =
4.244-314
1.273-313
8.488-314
2.122-314
4.941-324
8.488-314
5.093-313
2.122-314
2.122-314
9.881-324
1.1
2.
```

See Also

- vec2var - Transform a vector of double in a scilab variable (Scilab Function)

Name

vec2var — Transform a vector of double in a scilab variable

```
[var]=vec2var(vec)
```

Module

- xcOS

Description

var2vec / vec2var functions are used inside the interfacing functions of Scilab blocks to give the possibility to the user to handle Scilab objects with the register of real parameters (rpar) and with the discrete state register (z).

```
[var]=vec2var(vec)
```

Parameters

- **vec** : Input parameter. A vector of real numbers.
- **var** : Output parameter. Can be any types of Scilab objects.

Example

```
-->a=list("cos",[1.1,2])
a =
a(1)
cos
a(2)
1.1    2.
-->b=var2vec(a)
b =
4.244-314
1.273-313
8.488-314
2.122-314
4.941-324
8.488-314
5.093-313
2.122-314
2.122-314
9.881-324
1.1
2.
-->c=vec2var(b)
c =
c(1)
cos
c(2)
1.1    2.
```

See Also

- `var2vec` - Transform a scilab variable in a vector of double (Scilab Function)

Name

xcos — Block diagram editor and GUI for the hybrid simulator

```
xcos(filename)
xcos(scs_m_list)
```

Module

- xcos

Description

Xcos is a visual editor for constructing models of hybrid dynamical systems.

Invoking Xcos with no argument opens up an empty Xcos window. Models can then be assembled, loaded, saved, compiled, simulated, using GUI of Xcos. Xcos serves as an interface to the various block diagram compilers and the hybrid simulator scicosim.

```
xcos('mydiagram.xcos')
```

Parameters

- **filename** : a character string containing the path of the diagram file (.cos, .cosf or .xcos extension). If no input argument is used, an empty diagram is opened (default name Untitled).
- **scs_m_list** : a Xcos diagram structure after edition.

Example

```
// Open a new diagram
xcos();
xcos

// Load a diagram
xcos SCI/scicos/demos/bounce.cosf;

// Load a structure
load('mondiagr.cos');
xcos(scs_m);
```

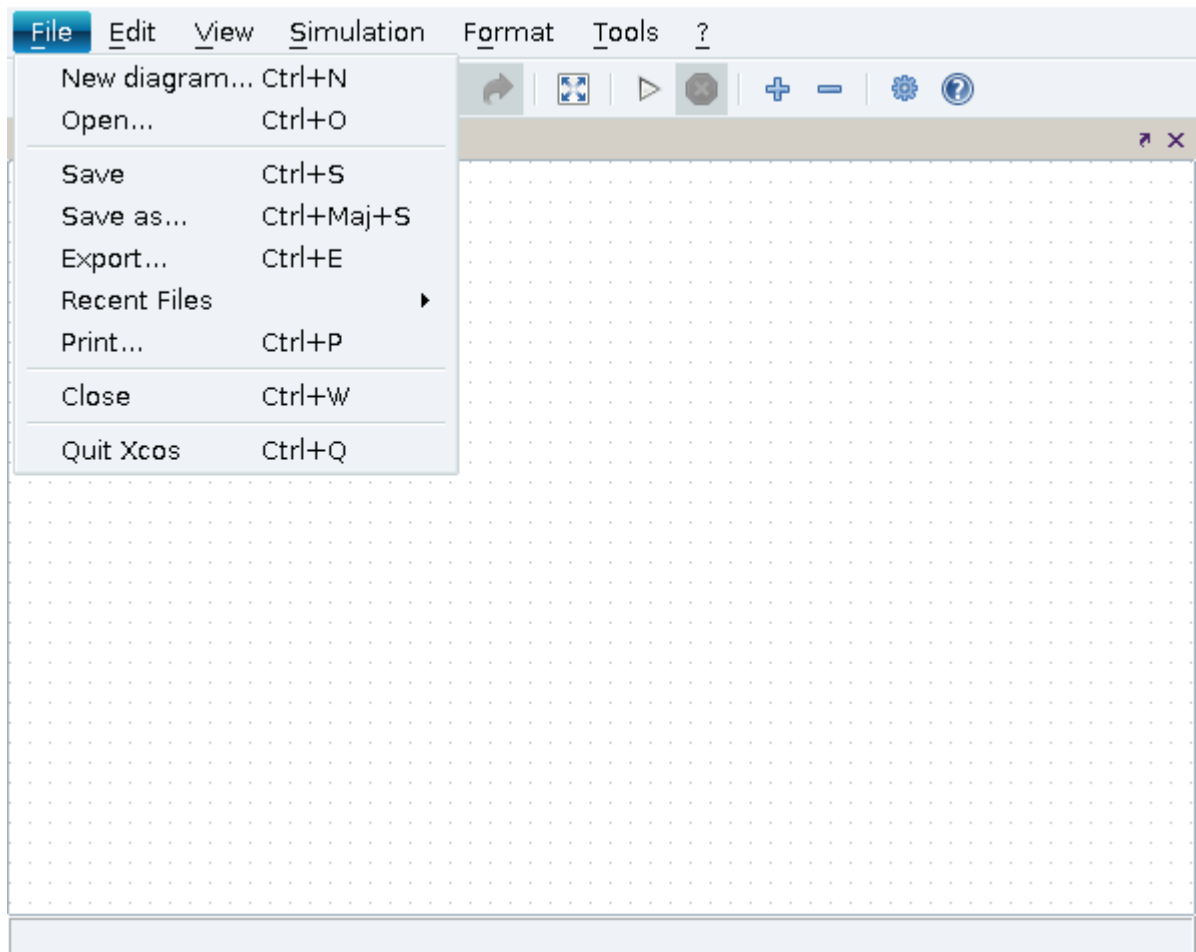
See Also

- scicosim - Scicos (batch) simulation function (Scilab Function)
- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)
- Menu entries

Name

Menu_entries — Editor menu entries

File menu



- **File:New**

Clicking on the New menu item loads an empty diagram in the active editor xcos window. If the previous content of the window is not saved, it will be lost. With this menu, you can open a new diagram or a new palette.

- **File:Open (Ctrl+o)**

Select the Open menu item to load an ASCII or binary file containing a saved block diagram or palette. A dialog box allows user choosing the file.

- **File:Save (Ctrl+s)**

Select the save menu item to save the block diagram in a binary file already selected by a previous select the Save As menu item. If you select this menu item and you have never clicked on the Save As menu item, the diagram is saved in the current directory as "window_name".cos where "window_name" is the name of the window appearing on top of the window (usually Untitled or Super Block). The .cos binary files are machine independent.

- **File:Save As (Ctrl+Shift+s)**

Select the Save As menu item to save the block diagram or palette in a file. A dialog box allows choosing the file which must have a .cos or .cosf extension. The diagram takes the name of the file

(without the extension). If extension is ".cosf" an ASCII formatted save is performed instead of binary save. Formatted save is slower than regular save.

- **File:Export (Ctrl+e)**

This menu is used to export a figure of the current xcos diagram. The export can be done directly in postscript format or done first in a graphic window to export in a second step in all the format that scilab can provide.

- **File:Recent Files**

Via this menu, you have a quick access to the recent opened files.

- **File:Print (Ctrl+p)**

Print the current diagram onto a printer.

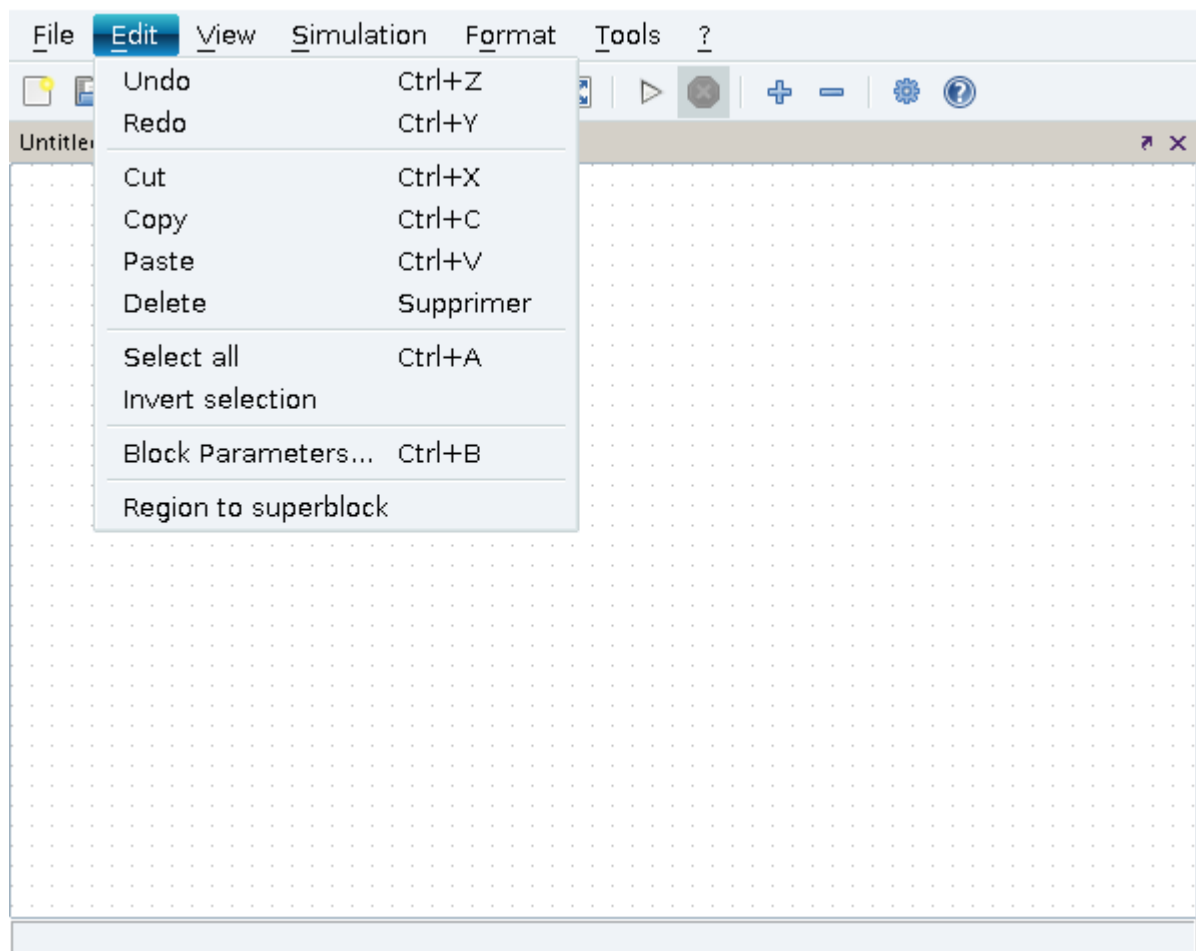
- **File:Close (Ctrl+w)**

If several diagram are opened, the Close action closes the current diagram. If only one diagram is opened, the Close action will close xcos and closes the viewport and palettes windows if these windows are opened.

- **File:Quit (Ctrl+q)**

If several diagram are opened, the Quit action will close xcos and closes the viewport and palettes windows if these windows are opened. It will close all the opened diagram.

Edit menu



- **Edit:Undo (Ctrl+z)**

Select the Undo menu item to undo the last edit operation.

- **Edit:Redo (Ctrl+y)**

Select the Redo menu item to redo the last undo edit operation.

- **Edit:Cut (Ctrl+x)**

Cut is used to remove the selected object from the diagram and keep a copy in the clipboard if the object is a block.

- **Edit:Copy (Ctrl+c)**

Copy is used to place a copy of the selected object in the clipboard if the object is a block.

- **Edit:Paste (Ctrl+v)**

Paste places the object in the Clipboard in the diagram.

- **Edit>Delete (Delete)**

To delete blocks or a links, select first the Delete menu item, then click successively on the selected objects (with left button). When you delete a block all links connected to it are deleted as well.

- **Edit:Select all (Ctrl+a)**

Select all the blocks in the current diagram.

- **Edit:Invert selection**

Invert the current selection.

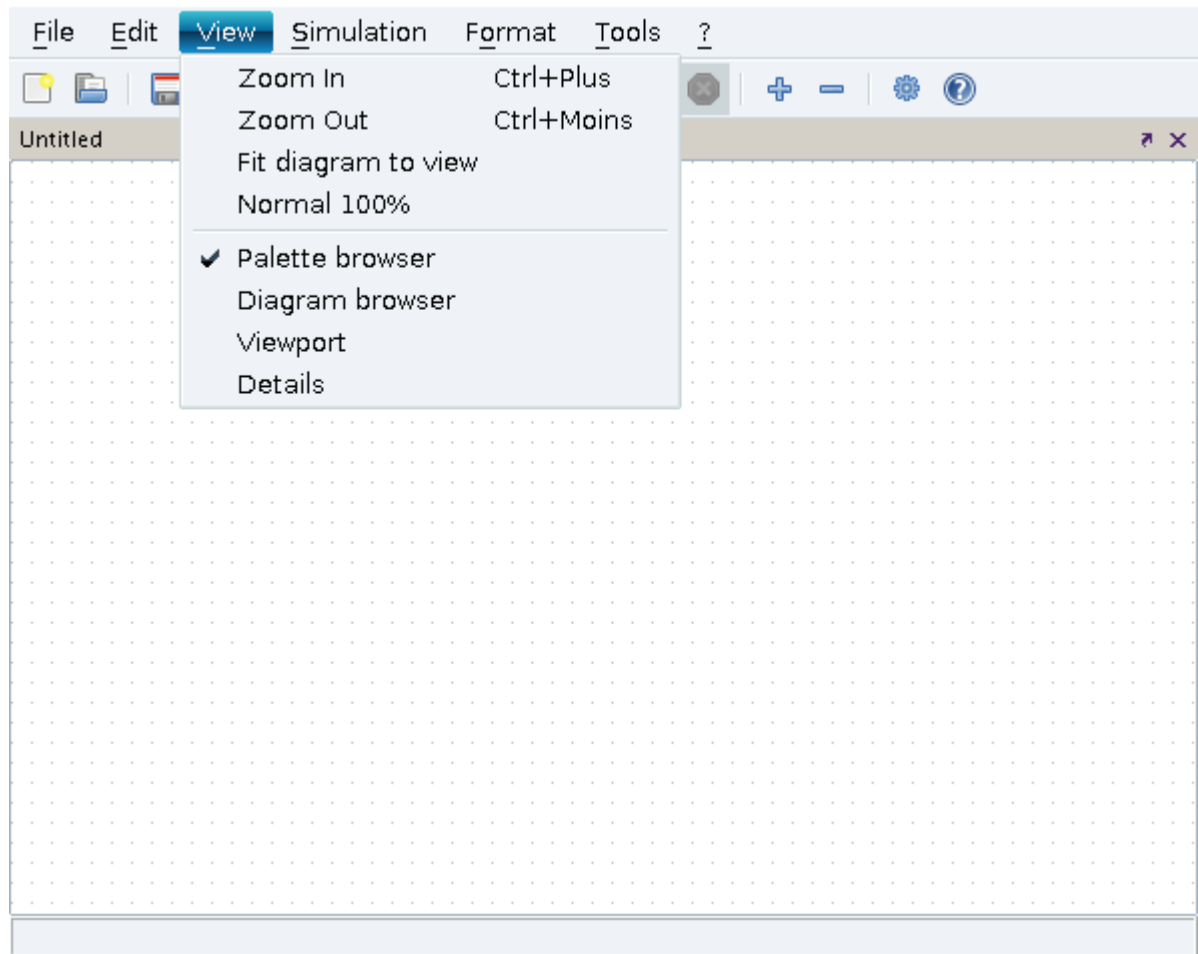
- **Edit:Block Parameters (Ctrl+b)**

Open the block configuration window for the current selected block.

- **Edit:Region to superblock**

Convert a selection of blocks into a superblock.

View menu



- **View:Zoom in (Ctrl+plus)**

When you select this menu item the diagram is zoomed in by a factor of 10%.

- **View:Zoom out (Ctrl+minus)**

When you select this menu item the diagram is zoomed out by a factor of 10%.

- **View:Fit diagram to view**

When you select this menu item the diagram is fit to the size of the current window.

- **View:Normal 100%**

Resize the work area so as the diagram fits onto this work area.

- **View:Palette browser**

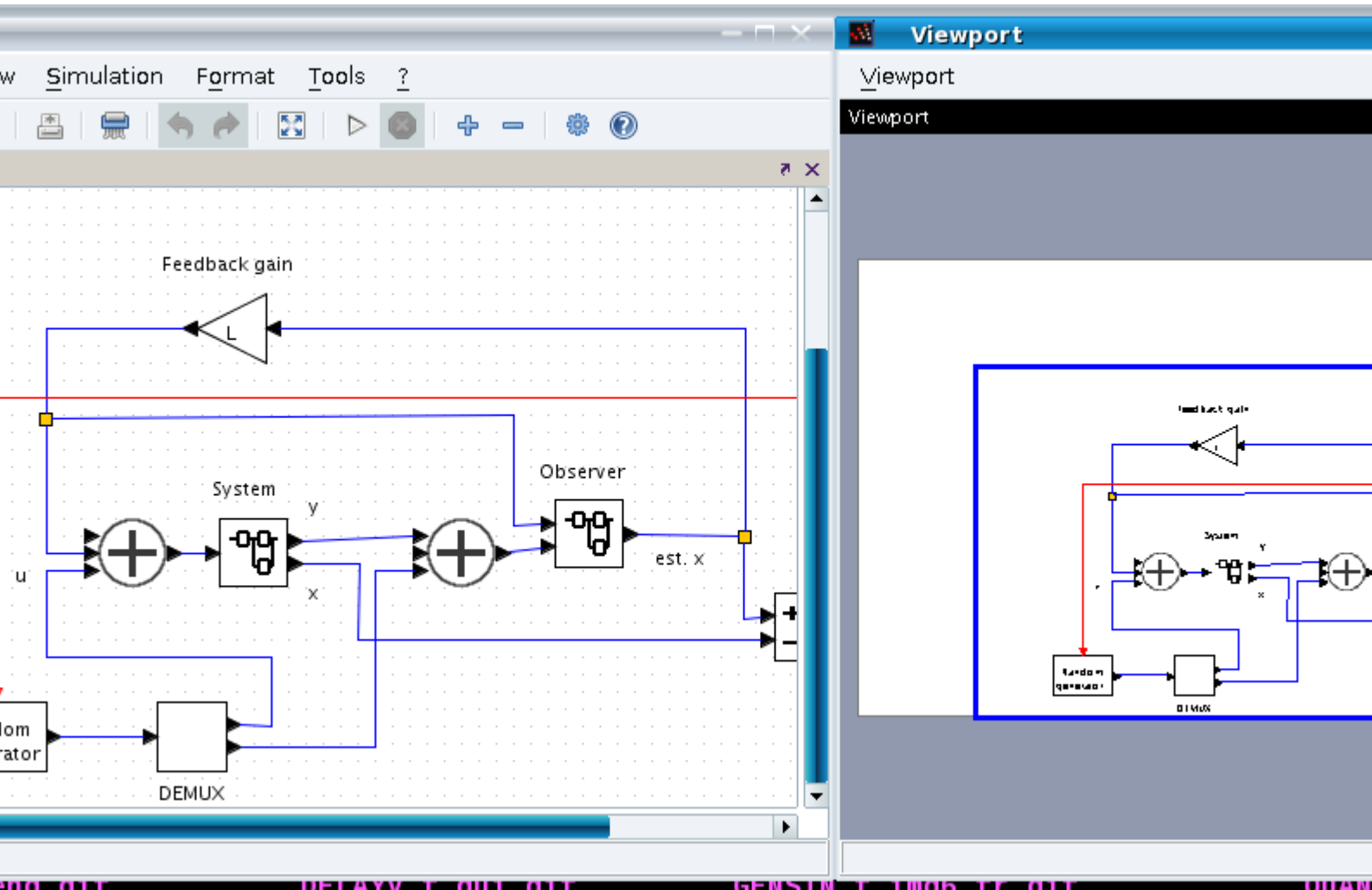
Open the palette browser.

- **View:Diagram browser**

Displays a window which lists all the blocks of a diagram and print some informations related to the scs_m structure of the blocks.

- **View:Viewport**

Display the viewport. The viewport is an image of the current diagram. With the viewport, you can move the working area onto a piece of the diagram. You can zoom and unzoom part of a diagram.

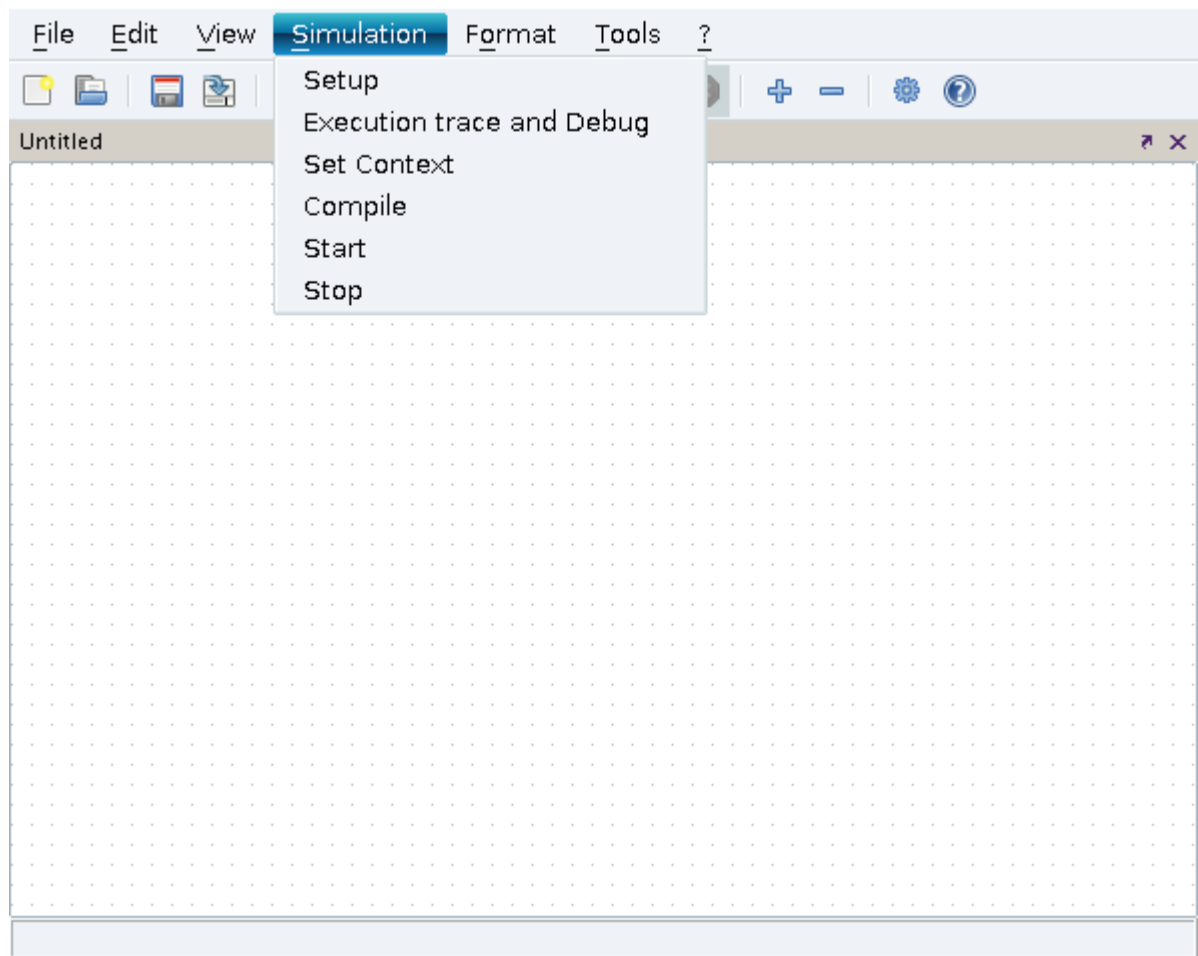


Above, you have an example of the viewport which is used to zoom on a part of a diagram, and on the right, the xcos window displays the zoomed part of the diagram.

- **View:Details**

Displays a window which lists all the selected blocks of a diagram and print some informations related to the scs_m structure of these blocks.

Simulation menu



- **Simulation:Setup**

In the main Xcos window, clicking on the Setup menu item invokes a dialog box that allows you to change integration parameters:

- Final integration time (integration ends at this time, it always starts from 0)
- Real time scaling (forces real time simulation by setting xcos unit of time to 1 second)
- Absolute and relative error tolerances (solver properties)
- Time tolerance (the smallest time interval for which the ode solver is used to update continuous states)
- Max integration time interval (the maximum time interval for each call to solver, it must be reduced if the error message "too many calls" is encountered)
- Solver (choose the numerical solver to be used), Max step size (max time step taken by solver)

- **Execution trace and Debug**

Set Xcos in debug mode. Allows diagram debugging.

- **Simulation:Set Context**

When you select this menu item you obtain a dialog to enter Scilab instructions for defining symbolic Xcos parameters used in block definitions or to do whatever you want. These instructions will

be evaluated each time the diagram is loaded. If you change the value of a symbolic Xcos parameters in the context, all the blocks are updated (Eval is performed).

- **Simulate:Compile**

Select the Compile menu item to compile the block diagram. This menu item need never be used since compilation is performed automatically, if necessary, before the beginning of every simulation (Run menu item). Normally, a new compilation is not needed if only system parameters and internal states are modified. In some cases however these modifications are not correctly updated and a manual compilation may be needed before a Restart or a Continue. Please report if you encounter such a case.

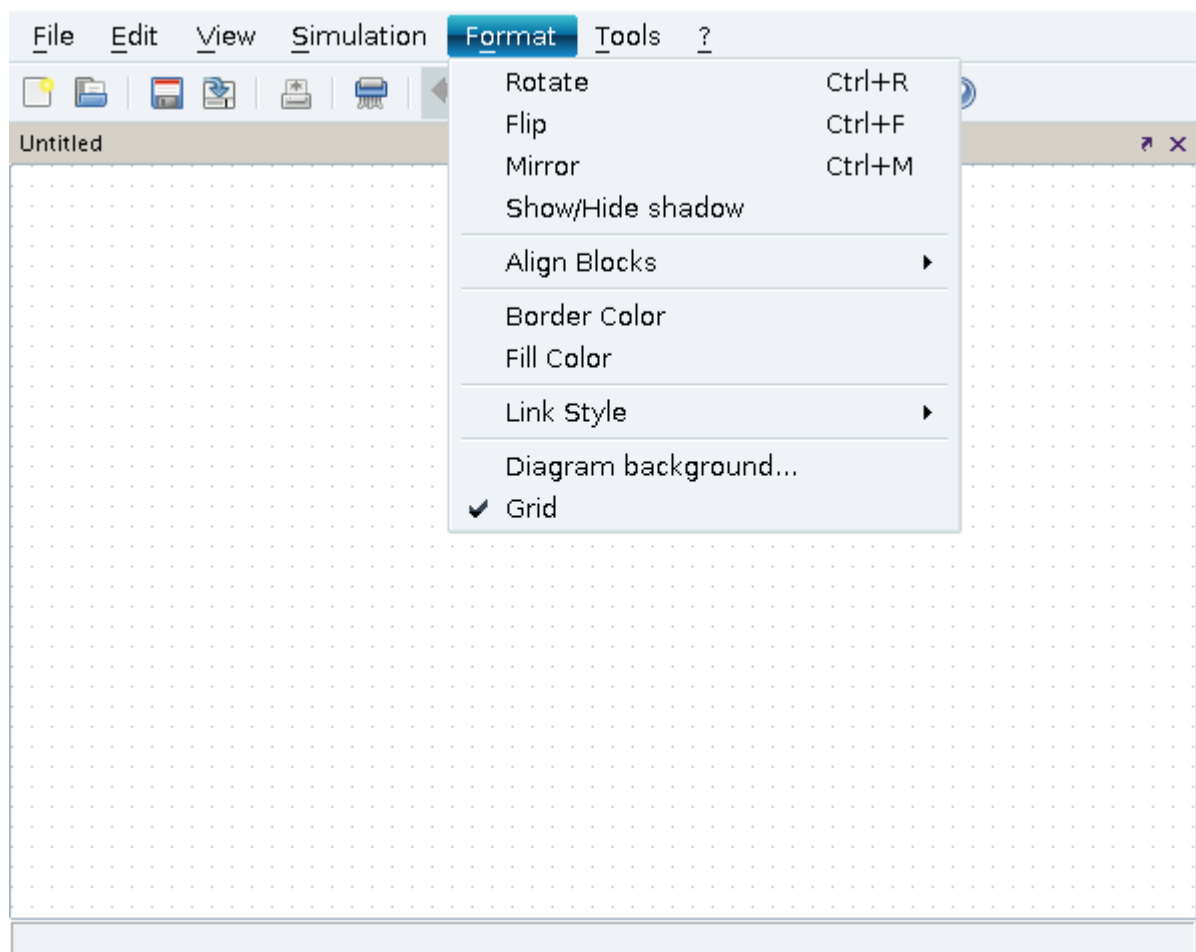
- **Simulate:start**

Select the Run menu item to start the simulation. If the system has already been simulated, a dialog box appears where you can choose to Continue, Restart or End the simulation.

- **Simulation:Stop**

You may interrupt the simulation by clicking on the "stop" button, change any of the block parameters and continue the simulation with the new values.

Format menu



- **Format:Rotate (Ctrl+r)**

Rotate allows to turn a block on the Left. Each time the block is turned left, his angle is decreased of 45 degrees. If no blocks or many blocks are selected, this is the block under the mouse pointer which turns.

- **Format:Flip (Ctrl+f)**

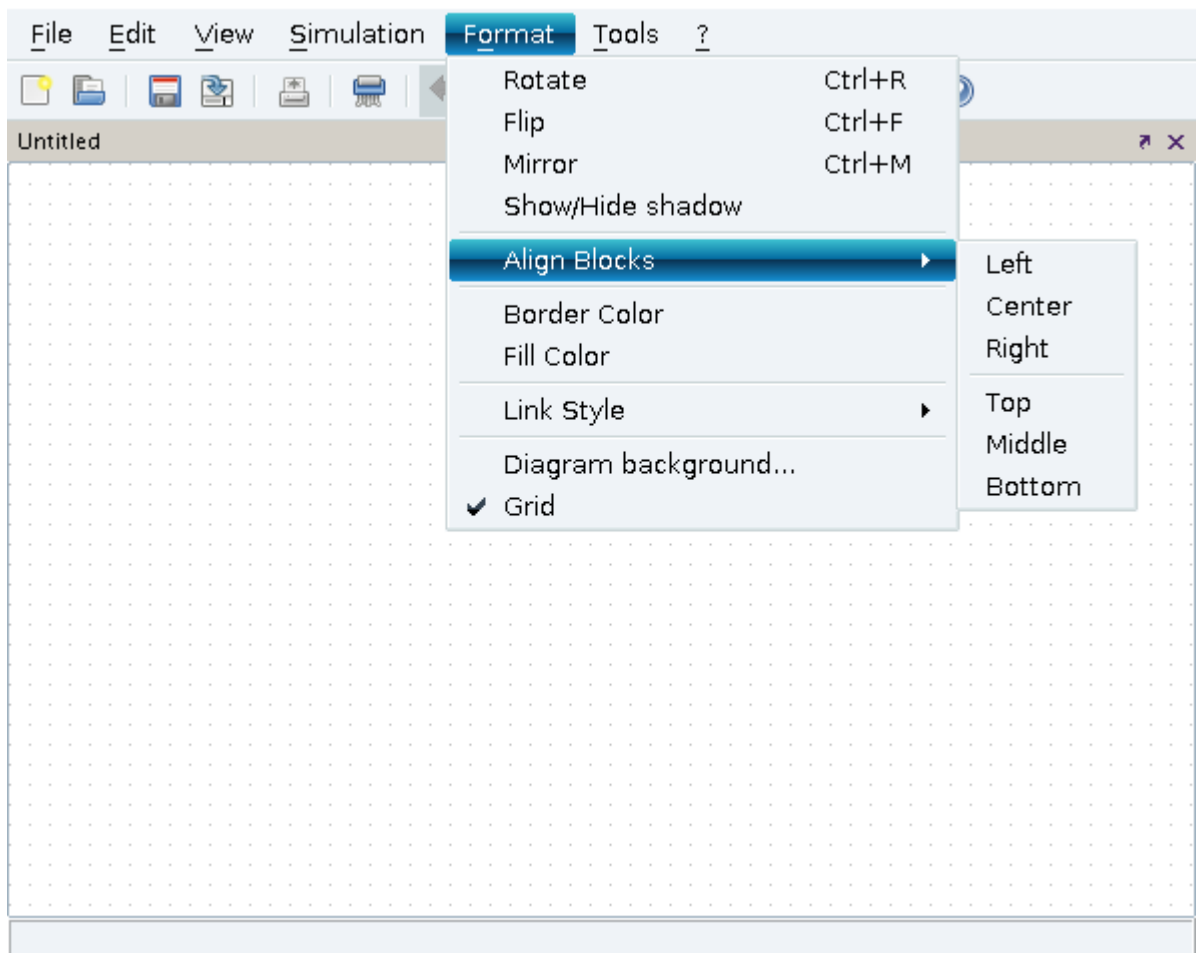
To reverse the positions of the (regular) inputs and outputs of a block placed on its sides, select the Flip menu item first and then click on the selected block. This does not affect the order, nor the position of the input and output event ports which are numbered from left to right.

- **Format:Show/Hide shadow**

This menu allows to select 3D shape for selected blocks and associated parameters.

- **Format:Align Blocks**

When you select several blocks, it is possible to align them vertically (Top, Bottom and Middle) and horizontally (Left, Right, Center).



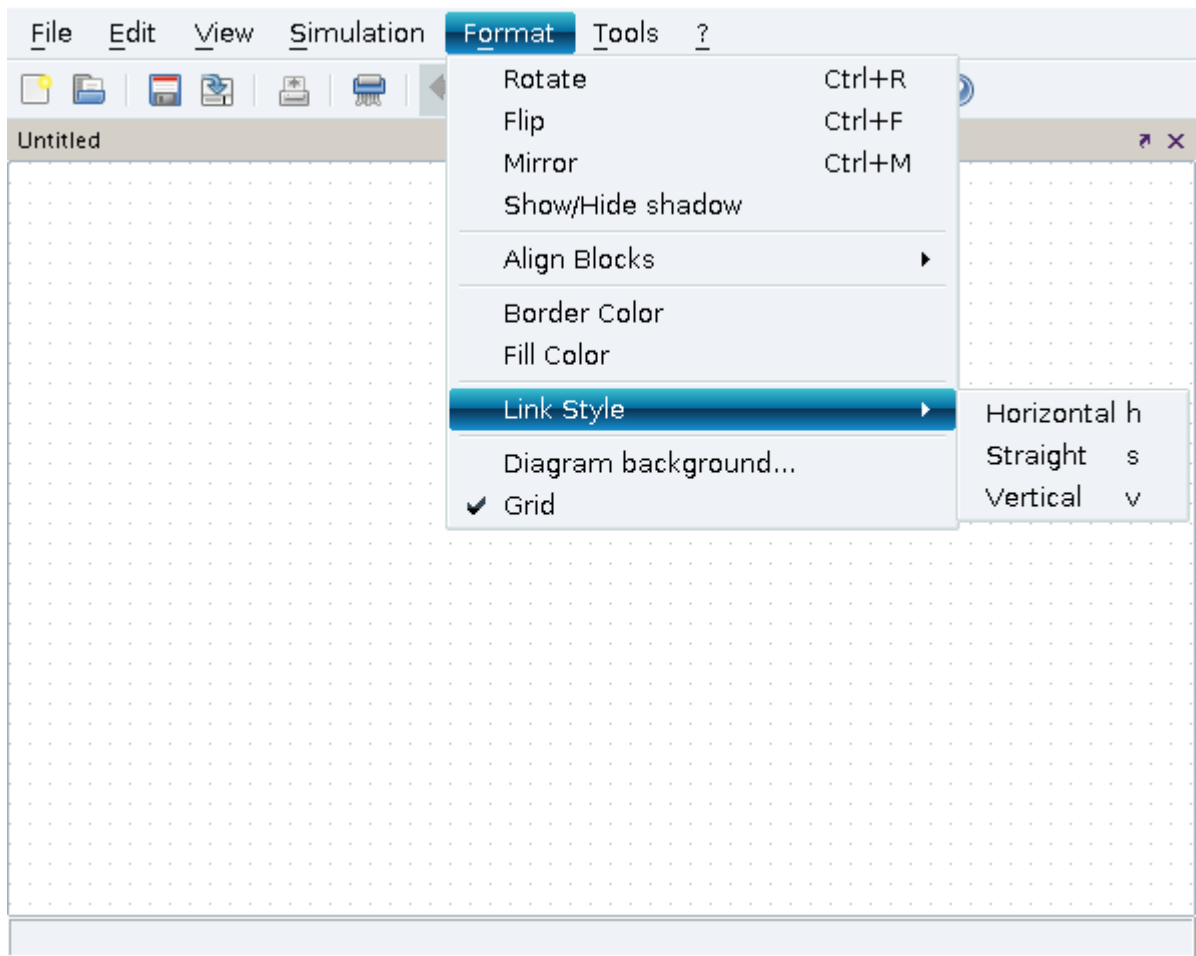
- **Format: Border Color**

This menu allows to change the border color.

- **Format:Line Color**

This menu allows to change the line color.

- **Format:Link Style**



This menu allows to the the style of the link:

- Horizontal
- Straight
- Vertical
- **Format:Diagram background**

This menu allows to change the background color.

- **Format:Grid**

This menu allows to activate / deactivate the grid. Using a grid, it is easier to place a block on the working area.

Tools menu

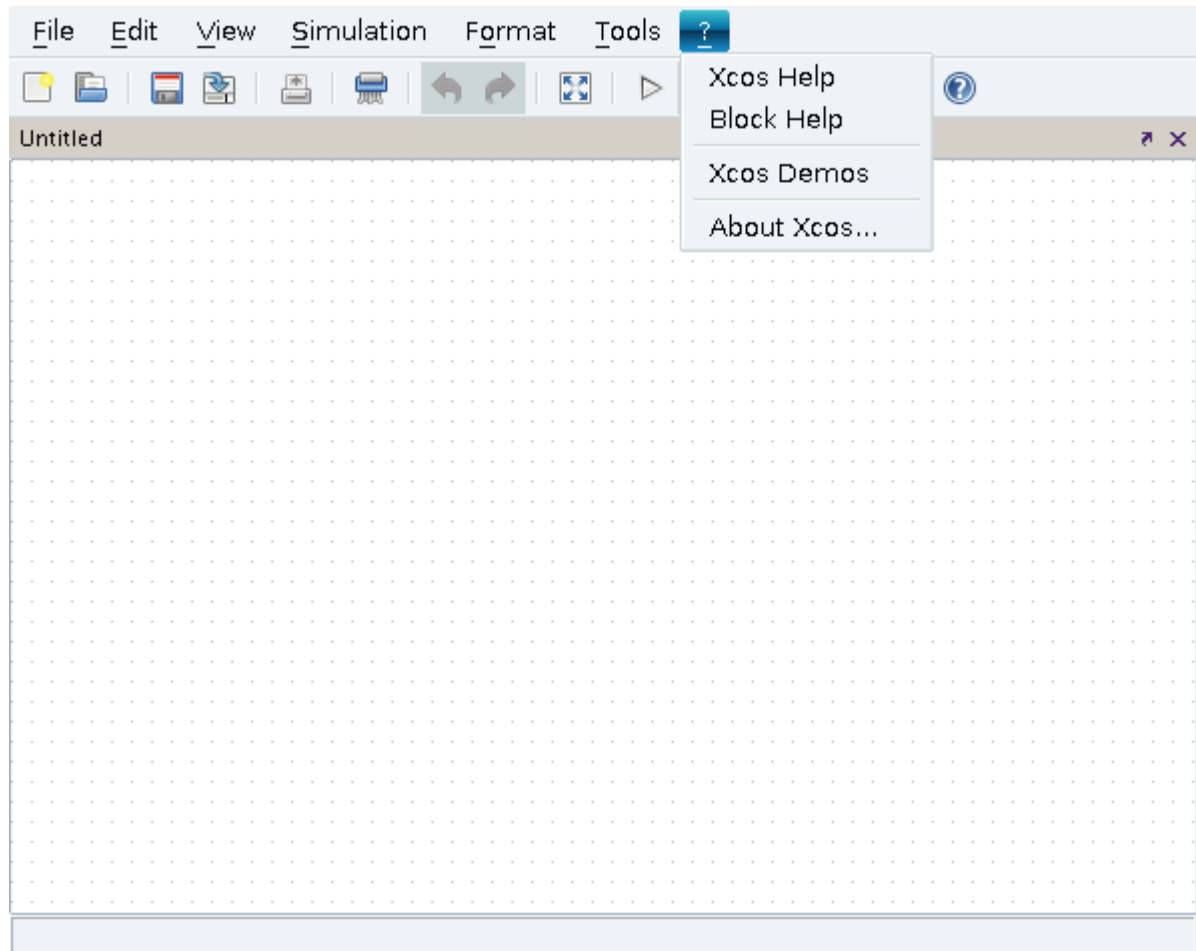
Code generation

- **Tools:Code generation**

This menu allows to generate the simulation code associated with a discrete time Super Block. The code generation is obtained simply by selecting this menu and then the desired Super Block. If the Super Block satisfies the required conditions, a dialog box pops up to ask for a block name, a directory where to put the generated files and for optional libraries requested by the linker. Given this information the code is generated, compiled and linked with Scilab. The Super Block is auto-

matically replaced by a new block which implements the generated code. It is then possible to run the modified diagram. The code for standalone use is also generated.

Help menu



- **Help:Xcos Help**

This menu opens the help browser.

- **Help:Block Help**

To get help on a Xcos object (a block), select first an object (a block) and then click on this menuHelp menu item and then on the selected object or menu item.

- **Help:Xcos Demos**

The Demos menu allows to open some examples of Xcos diagram.

- **Help:About Xcos**

About xcos item display the current version of Xcos and gives some useful informations.

Part LVI. Text editor

Name

`edit_error` — opens in scilab editor the source of the last recorded error

```
answ = edit_error(clearerror)
```

Parameters

`clearerror`

boolean - if true the error condition is cleared, if false it is kept (as in `lasterror`)

`answ`

a string stating which source file is open (or why no file was open)

Description

This function opens in scilab editor the source of the function which caused the last recorded error.

This function works only for functions which are defined in libraries, i.e. not for internal functions, nor with functions defined online, nor loaded with individual `exec` or `getd`. This is since Scilab presently retains only the path to libraries and not to individual function sources.

Correspondance between the function name `foo` and function filename `foo.sci` is tacitly assumed.

Examples

```
acosh abc  
edit_error
```

See Also

`lasterror` , `errclear`

Authors

Enrico Segre

Allan CORNET

Name

Editor — Embedded Scilab text editor

```
editor()  
editor(file)  
editor([file1, file2])  
editor(file, line_number)  
editor([file1, file2], [line_number1, line_number2])
```

Parameters

`file`

a string, the file we want to open.

`[file1, file2]`

a matrix of string, files we want to open.

`line_number`

an integer, number of the line we want to highlight at the opening of the file.

`[line_number1, line_number2]`

a matrix of integer, each opened file will have it's corresponding line highlighted.

Description

`Editor` is an embedded Scilab text editor.

It can be started with a fresh text buffer pressing the "Editor" button on top of the main Scilab window, or from Scilab command line with the instruction `editor()`, or it can open specific files if invoked with any of the calling sequences above (whithout any parameters, it opens editor with a blank file).

The same invocation adds further files to an already opened Editor.

Keyboard shortcuts are defined for most possible editing actions and reported by the menu entries.

`editor` can be started in the following ways :

- By the menu Applications. Choose Applications => Editor
- From the command line:
 - `editor()`
 - `editor(file)`
 - `editor([file1, file2])`
 - `editor(file, line_number)`
 - `editor([file1, file2], [line_number1, line_number2])`

Menus and Shortcuts

- Menu File

Commande	Shortcut	Description
New...	<CTRL-N>	Open a new file
Open...	<CTRL-O>	Open an existing file
Recent Files		Display files recently opened
Save	<CTRL-S>	Save a file
Save as...	<CTRL-MAJ-S>	Save a file as
Page Setup		Setup page for printing
Print Preview	<CTRL-MAJ-P>	Open a print preview window
Print...	<CTRL-P>	Print a file
Close	<CTRL-W>	Close a file
Quit	<CTRL-Q>	Close Editor

- Menu Edit

Commande	Shortcut	Description
Undo	<CTRL-Z>	Undo action
Redo	<CTRL-Y>	Redo action
Cut	<CTRL-X>	Cut the selection
Copy	<CTRL-C>	Copy the selection
Paste	<CTRL-V>	Paste the selection
Select All	<CTRL-A>	Select the entire document
Delete		Delete the selection
Comment Selection	<CTRL-D>	Comment selected lines
Uncomment Selection	<CTRL-MAJ-D>	Uncomment selected lines
Tabify Selection	<TAB>	Tabify selected lines
Untabify Selection	<MAJ-TAB>	Untabify selected lines
Indent	<CTRL-I>	Indent selected lines

- Menu Search

Commande	Shortcut	Description
Find/Replace	<CTRL-F>	Find and/or Replace an element
Goto line	<CTRL-G>	Goto line

- Menu View

Commande	Shortcut	Description
Show/Hide Toolbar		Option to show or hide the toolbar
Highlight current line	<CTRL-J>	Highlight the current line
Line Numbers	<CTRL-B>	Display document's line numbers
Set Colors...		Color settings for documents
Set Fonts...		Font settings for documents
Reset default font		Reset default font settings for documents

- Menu Document

Commande	Shortcut	Description
Syntaxe Type		Syntaxe type settings (default type is Scilab)
Encoding		Encoding settings (default type is UTF-8 Encoding)
Colorize		Colorize the document
Auto Indent		Activate the automatic indentation

- Menu Execute

Commande	Shortcut	Description
Load Into Scilab	<CTRL-L>	Load the entire document into the Scilab console
Evaluate Selection		Load the selection into the Scilab console
Execute Into Scilab	<CTRL-E>	If the file exist, execute the content of the file

Remarks

Document :

The default text colourisation is the Scilab's syntaxe colourisation.

The auto-indent mode indent a line according to Scilab's syntaxe (after a return action).

Bugs:

You can reported bugs in the Bugzilla <http://bugzilla.scilab.org>

There are still a few bugs that we are trying to fix, details can be found in the given link by filtering entries with the "Editor" element.

Examples

```
// editor without parameters
editor();

// editor with a file name
editor('SCI/modules/time/macros/datenum.sci');

// editor with a matrix of files name
editor(['SCI/modules/time/macros/datenum.sci', 'SCI/modules/time/macros/datevec.

// editor with a file name and the line number to highlight
editor('SCI/modules/time/macros/datenum.sci', 5);

// editor with a matrix of files name and the corresponding matrix of lines to l
// the files name matrix and the lines to highlight matrix should have the same
editor(['SCI/modules/time/macros/datenum.sci', 'SCI/modules/time/macros/datevec.
```

Author

Sylvestre KOUMAR

Part LVII. API Scilab

Table of Contents

11. Scilab Gateway API	3050
1. How to	3050
CheckColumn	3072
CheckDimProp	3073
CheckDims	3074
CheckLength	3075
CheckLhs	3076
CheckRhs	3077
CheckRow	3078
CheckSameDims	3079
CheckScalar	3081
CheckSquare	3082
CheckVector	3083
CreateListVarFrom	3084
CreateListVarFromPtr	3087
CreateVar	3090
FindOpt	3092
FirstOpt	3094
GetListRhsVar	3096
GetRhsVar	3098
GetType	3100
IsOpt	3101
Lhs	3103
LhsVar	3104
NumOpt	3106
OverLoad	3108
Rhs	3110
Scierror	3111
Scilab C Types	3112
get_optionals	3114
istk	3116
sci_types	3117
sciprint	3120
stk	3121
12. list_management	3122
Boolean reading (Scilab gateway)	3123
Boolean writing (Scilab gateway)	3135
Boolean sparse reading (Scilab gateway)	3139
Boolean sparse writing (Scilab gateway)	3151
Create List (Scilab gateway)	3155
Double reading (Scilab gateway)	3159
Double writing (Scilab gateway)	3171
Get child item (Scilab gateway)	3175
Item Number (Scilab gateway)	3178
Integer reading (Scilab gateway)	3181
Integer writing (Scilab gateway)	3193
Pointer reading (Scilab gateway)	3198
Pointer writing (Scilab gateway)	3210
Polynomial reading (Scilab gateway)	3214
Polynomial writing (Scilab gateway)	3226
Sparse reading (Scilab gateway)	3230
Sparse writing (Scilab gateway)	3242
String reading (Scilab gateway)	3246
String writing (Scilab gateway)	3258

Chapter 11. Scilab Gateway API

1. How to

Name

Calling a scilab function (macros) from a C gateway — Calling a scilab function (macros) from a C interface

Calling a scilab function (macros) from a C interface

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

When you write a interface with scilab, you can need to call another function directly from your function. You can pass a pointer on scilab function to your function.

The source files of this example are stored in the directory `examples/call_scifunction`.

How to use this example:

```
exec call_scifunction.sce; // launch scilab
v = call_scifunc(30,12,scilabfoo);
```

you pass a pointer on a scilab function (scilabfoo macro) to your function 'call_scifunc'.

C2F(scifunction) calls another scilab function (here scilabfoo).

You need to indicate :

- position of the first element (Rhs) in the Scilab memory
- pointer on scilab function
- number of Lhs (output of scilab function called)
- number of Rhs (input of scilab function called)

The script `call_scifunction.sce` used to build and load the C interface into Scilab is the following:

```
files=['sci_call_scifunc.c']; // Defines the list of file
ilib_build('callscifunc',['call_scifunc','sci_call_scifunc'],files,[]); // Build
exec loader.sce;

function r = scilabfoo(x,y)
    r = x + y;
endfunction

v = call_scifunc(30,12,scilabfoo);
disp('result : ' + string(v));

v = call_scifunc(300,120,scilabfoo);
disp('result : ' + string(v));

ulink(); // unload the dynamic library
```

The file `sci_call_scifunc.c` is the following:

```
#include "stack-c.h"
#include "Scierror.h"
#include "localization.h"
```

```

int sci_call_scifunc(char *fname)
{
    int m1      = 0, n1      = 0, l1      = 0;
    int m2      = 0, n2      = 0, l2      = 0;
    int m3      = 0, n3      = 0, l3      = 0;
    int rm1     = 0, rn1     = 0, rl1     = 0;
    int m_out   = 1, n_out   = 1, l_out   = 0;
    double v1   = 0, v2   = 0, r   = 0;
    int positionFirstElementOnStackForScilabFunction = 0;
    int numberOfRhsOnScilabFunction = 0;
    int numberOfLhsOnScilabFunction = 0;
    int pointerOnScilabFunction      = 0;

    CheckRhs(3,3);
    CheckLhs(1,1);

    if (GetType(1) != sci_matrix)
    {
        Scierror(999,_"%s: Wrong type for input argument #%d: A real expected.\n");
        return 0;
    }

    GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m1, &n1, &l1);
    if ( (m1 == n1) && (n1 == 1) )
    {
        v1 = *stk(l1);
    }
    else
    {
        Scierror(999,_"%s: Wrong size for input argument #%d: A scalar expected.\n");
        return 0;
    }

    if (GetType(2) != sci_matrix)
    {
        Scierror(999,_"%s: Wrong type for input argument #%d: A real expected.\n");
        return 0;
    }

    GetRhsVar(2, MATRIX_OF_DOUBLE_DATATYPE, &m2, &n2, &l2);
    if ( (m2 == n2) && (n2 == 1) )
    {
        v2 = *stk(l2);
    }
    else
    {
        Scierror(999,_"%s: Wrong size for input argument #%d: A scalar expected.\n");
        return 0;
    }

    if (GetType(3) != sci_c_function)
    {
        Scierror(999,_"%s: Wrong type for input argument #%d: A scilab function expected.\n");
        return 0;
    }

    // get pointer on external function (here scilabfoo)

```

```
GetRhsVar(3, EXTERNAL_DATATYPE, &m3, &n3, &l3);

// r = scilabfoo(x, y)
// rhs eq. 2
// lhs eq. 1

// Position first element in the Scilab memory to use by Scilab Function
// v = call_scifunc(300,120,scilabfoo);
// On stack : 300 is on Top position (1)
// 120 second position
// scilabfoo third position
// we want to pass 300 & 120 to scilab Function
// First position is here : 1

positionFirstElementOnStackForScilabFunction = 1;
numberOfRhsOnScilabFunction = 2;
numberOfLhsOnScilabFunction = 1;
pointerOnScilabFunction      = l3;

// r = scilabfoo(x, y)
// Scifunction call a scilab function
Scifunction(&positionFirstElementOnStackForScilabFunction,
            &pointerOnScilabFunction,
            &numberOfLhsOnScilabFunction,
            &numberOfRhsOnScilabFunction);

// result r is now on position positionFirstElementOnStackForScilabFunction o
GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &rm1, &rn1, &rl1);
r = *stk(rl1);

CreateVar(Rhs+1, MATRIX_OF_DOUBLE_DATATYPE, &m_out, &n_out, &l_out);
*stk(l_out) = r;

LhsVar(1) = Rhs + 1;

return 0;
}
```

The main function in this C file is Scifunction. It allows to call a Scilab function inside a C interface.

See Also

CheckLhs, CheckRhs, stk, LhsVar, GetType, Scierror, Rhs, Lhs, sci_types

Name

How to access a matrix — How to access a matrix using the C gateway functions

Description

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

The goal is to get a matrix of doubles send to a function written in C.

For this, we will wrote a C gateway function in which we will retrieve the matrix, we will perform some simple steps in this C function:

- First, we will get an access to the matrix in the Scilab memory
- We will perform some simple operations on the matrix (in this example, we will multiply by 2 each elements of the matrix)
- We will return the result to Scilab

This example is available in the directory SCI/modules/core/examples/ex1.

The C function

```
#include <stack-c.h>

int sci_multiply_by_two(char * fname)
{
    int m_in_var, n_in_var, l_in_var;
    int m_out_var, n_out_var, l_out_var;
    int i_row, j_col;
    double * pMatrix = NULL;

    // First, access to the input variable (a matrix of doubles)
    GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_in_var, &n_in_var, &l_in_var);

    // Create the returned variable (a matrix of doubles)
    m_out_var = m_in_var;
    n_out_var = n_in_var;
    CreateVar(2, MATRIX_OF_DOUBLE_DATATYPE, &m_out_var, &n_out_var, &l_out_var);

    pMatrix = stk(l_in_var);

    // Perform some simple operations on the matrix
    for(i_row=0; i_row<m_in_var; i_row++)
    {
        for(j_col=0; j_col<n_in_var; j_col++)
        {
            pMatrix[i_row + j_col * m_out_var] = 2 * pMatrix[i_row + j_col * m_in];
        }
    }

    // Return the output variable
    LhsVar(1) = 2;

    return 0;
}
```

This file must be saved as "multiply_by_two.c".

The main thing to highlight is that, to build a C gateway function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C gateway functions.

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name = 'lib_multiply_by_two';

files = ['multiply_by_two.c'];

libs = [];

table = ['multiply_by_two', 'sci_multiply_by_two'];

ldflags = "";
cflags = "";
fflags = "";

ilib_build(ilib_name, table, files, libs, 'MakeLib', ldflags, cflags, fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's multiply_by_two.c), what will be the name of the shared library (here, it's lib_multiply_by_two) and which C symbol will be linked to a Scilab function (here, we will link the sci_multiply_by_two C symbol to the Scilab function "multiply_by_two").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

Testing our new function

We now write a simple example to test our new function.

```
A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15];  
  
B = multiply_by_two(A);  
  
disp(B);
```

The script must be saved as "test.sce".

Let's run our scripts and see what is the result:

```
-->exec builder.sce;  
Generate a gateway file  
Generate a loader file  
Generate a Makefile  
ilib_gen_Make: Copy compilation files (Makefile*, libtool...) to TMPDIR  
ilib_gen_Make: Copy multiply_by_two.c to TMPDIR  
ilib_gen_Make: Copy lib_multiply_by_two.c to TMPDIR  
ilib_gen_Make: Modification of the Makefile in TMPDIR.  
Running the makefile  
  
-->exec loader.sce;  
Shared archive loaded.  
Link done.  
  
-->exec test.sce;  
  
      2.      4.      6.      8.      10.  
     12.     14.     16.     18.     20.  
     22.     24.     26.     28.     30.  
-->
```

This simple function has produced a new matrix which corresponds to the matrix transmitted as an input argument and for which each element of the matrix has been multiplied by 2.

Rebuilding a gateway function

Let's imagine that our gateway function has already been build and we would like to make some changes in our function (multiply by 3 instead of 2).

How do we perform such a changes without restarting Scilab ?

First, we need to list all the dynamic libraries which has been loaded into Scilab. The can be done using the link('show') function:

```
-->link('show')  
Number of entry points 1.  
Shared libraries :  
[ 0 ] : 1 libraries.  
Entry point lib_multiply_by_two in shared library 0.  
ans =  
  
      0.
```

Here, in our current Scilab session, only 1 dynamic library has been loaded. This library has a reference number. For our library, it's "0". Now that we know the reference number of our library, we are able to:

- unload this library (using the function `ulink(0)` - 0 is the reference number of our library)
- perform some modification in the source code of our C gateway function (replace multiply by 2 by 3)
- rebuild the C gateway function (`exec builder.sce;`)
- load the modified C gateway function into scilab (`exec loader.sce;`)

This is what is done in the following example:

```
-->ulink(0)

-->exec builder.sce;
Generate a gateway file
Generate a loader file
Generate a Makefile
ilib_gen_Make: Copy compilation files (Makefile*, libtool...) to TMPDIR
ilib_gen_Make: Copy multiply_by_two.c to TMPDIR
ilib_gen_Make: Copy lib_multiply_by_two.c to TMPDIR
ilib_gen_Make: Modification of the Makefile in TMPDIR.
Running the makefile

-->exec loader.sce;
Shared archive loaded.
Link done.

-->exec test.sce;

      3.      6.      9.     12.     15.
     18.     21.     24.     27.     30.
     33.     36.     39.     42.     45.
```

See Also

`GetRhsVar`, `Scilab C Types`, `CreateVar`, `LhsVar`, `stk`, `ilib_build`, `link`, `ulink`

Name

How to check parameters — how to check parameter send to an interface using the C gateway functions

Description

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

The goal is to get a set of parameters via a set of C gateway functions and then to perform some checks in the C function.

This example is available in the directory core/examples/check_properties.

The C function

```
#include <stack-c.h>
#include <sciprint.h>

int sci_check_properties_1(char * fname)
{
    int m1, n1, l1;
    int m2, n2, l2;
    int m3, n3, l3;
    int m4, n4, l4;
    int m5, n5, l5;

    CheckRhs(5,5);
    CheckLhs(0,1) ;

    //////////////////////////////////////
    // Getting first argument //
    //////////////////////////////////////

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckVector(1,m1,n1); // Check that first argument is a vector
    CheckLength(1,m1*n1,4); // Check vector length

    //////////////////////////////////////
    // Getting second argument //
    //////////////////////////////////////

    GetRhsVar(2, "d", &m2, &n2, &l2);

    CheckRow(2,m2,n2); // Checks that second argument is a row vector
                      // CheckColumn can also be used

    CheckDimProp(1,2, m1 * n1 != n2); // Check compatibility between arg 1 and a

    //////////////////////////////////////
    // Getting third argument //
    //////////////////////////////////////

    GetRhsVar(3, "d", &m3, &n3, &l3);

    CheckSameDims(1,3,m1,n1,m3,n3); // Checks that arg 1 and arg3 have same dimen
```



```

////////////////////////////////////
// Getting fourth argument //
////////////////////////////////////

GetRhsVar(4, "d", &m4, &n4, &l4);

CheckScalar(4, m4, n4); // arg 4 must be scalar

////////////////////////////////////
// Getting fourth argument //
////////////////////////////////////

GetRhsVar(5, "d", &m5, &n5, &l5);

CheckSquare(5, m5, n5); // square matrix
CheckDims(5, m5, m5, 5, 5); // check dimensions

LhsVar(1)=0;

return 0;
}

// We must be careful on the scilab name function (8 chars max).

int sci_check_properties_2(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1, 1);
    CheckLhs(0, 1);

    switch(VarType(1))
    {
        case 1:
            GetRhsVar(1, "d", &m1, &n1, &l1);
            sciprint("1 is a scalar matrix\n");
            break;
        case 10:
            GetRhsVar(1, "c", &m1, &n1, &l1);
            sciprint("1 is a string\n");
            break;
        case 5:
            sciprint("1 is a sparse trying to overload\n");
            OverLoad(1);
    }

    LhsVar(1) = 0;

    return 0;
}

```

This file must be saved as "check_properties.c".

The main thing to highlight is that, to build a C gateway function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C gateway functions. We also need to include sciprint.h because we use the sciprint function.

To be able to build and link such a C function to Scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name = 'lib_check_properties';

files = ['check_properties.c'];

libs = [];

table = ['check_properties_1', 'sci_check_properties_1'; ...
        'chprop2',           'sci_check_properties_2'];

// We must be careful when we choose a scilab function name in case of overload.
// We Scilab name function must be 8 char max.

ldflags = "";
cflags = "";
fflags = "";

ilib_build(ilib_name, table, files, libs, 'Makelib', ldflags, cflags, fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's check_properties.c), what will be the name of the shared library (here, it's lib_check_properties) and which C symbol will be linked to a Scilab function (here, we will link the sci_check_properties_1 C symbol to the Scilab function "check_properties_1").

For the other C function, we must be careful on the name of the Scilab function we will choose. Because this function will be overloading, the current overloading process of Scilab works only on Scilab primitives (Scilab function wrote in C) which have a name which is maximum 8 char wide.

For this function, we will link the sci_check_properties_2 C symbol to the Scilab function "chprop2").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

Testing our new function

We now write a simple example to test our new functions.

```
// checks arguments compatibility

check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],90,rand(5,5))

// first argument can have different types

chprop2([1,2,2]);
chprop2('foo');

// overload case

deff('[ ]=%sp_chprop2(sp)','disp(''sparse overloaded'')');
chprop2(sparse([1,2,3]));

// tests which give an error message with check_properties_1

try
    check_properties_1([1;2;3;4]',[3,4,5,6],[6;7;8;9],90,rand(5,5))
catch
    disp(lasterror());
end

try
    check_properties_1([1;2;3;4],[3,4,5,6]',[6;7;8;9],90,rand(5,5))
catch
    disp(lasterror());
end

try
    check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9]',90,rand(5,5))
catch
    disp(lasterror());
end

try
    check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],[ ],rand(5,5))
catch
    disp(lasterror());
end

try
    check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],90,rand(4,4))
catch
    disp(lasterror());
end
```

The script must be saved as "check_properties.sce".

Let's run our scripts and see what is the result:

```
-->exec builder.sce;
    Génère un fichier gateway
    Génère un fichier loader
    Génère un Makefile : Makelib
    Exécute le makefile
    Compilation de check_properties.c
```

```
Construction de la bibliothèque partagée (soyez patient)

-->exec loader.sce;
Bibliothèque partagée chargée.
Link done.

-->exec check_properties.sce;
1 is a scalar matrix
1 is a string
1 is a sparse trying to overload

sparse overloaded

check_properties_1 : Les paramètres first et third a des dimensions incompatib
check_properties_1: second paramètre devrait être un vecteur ligne
check_properties_1 : Les paramètres first et third a des dimensions incompatib
check_properties_1: fourth paramètre devrait être un scalaire
check_properties_1 : Argument numéro 5 n'a pas les bonnes dimensions (4,4), (
```

See Also

GetRhsVar, CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, OverLoad, ilib_build

Name

How to create and access a list — How to create and access a list using the C gateway functions

Description

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

The goal is to get a [mt]list, to get some elements stored in that this and to send a new [mt]list using a function written in C.

For this, we will wrote two C gateway function in which we will retrieve the [mt]list, and we will create a new [mt]list.

The C function - access to a [mt]list

This example is available in SCI/modules/core/example/print_list.

Let's initialize a mlist in Scilab.

```
A = mlist(['mytype', 'var1', 'var2'], 'a string', [1 2; 3 4]);
```

This mlist is of type 'mytype' (typeof(A)=='mytype') and it has 2 elements:

- A('var1') which is equal to 'a string'
- A('var2') which is equal to [1 2; 3 4]

We now create a C function called sci_print_list which will print the elements stored in the list.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_print_list(char * fname)
{
    int m_list_in, n_list_in, l_list_in;
    int m_type,      n_type;
    int m_var1,      n_var1,      l_var1;
    int m_var2,      n_var2,      l_var2;
    char ** LabelList = NULL;

    CheckRhs(1,1); // We accept only 1 parameter

    GetRhsVar(1,"m",&m_list_in,&n_list_in,&l_list_in); // Get a mlist

    // Get the type and the name of the variables (the first element of the mlist)
    GetListRhsVar(1,1,"S",&m_type,&n_type,&LabelList);

    if (strcmp(LabelList[0],"mytype")!=0)
    {
        sciprint("error, you must ship a mlist or type mytype\n");
        return 0;
    }

    // Get the first variable (a string)
    GetListRhsVar(1,2,"c",&m_var1,&n_var1,&l_var1);
```

```

sciprint("var1 = %s\n",cstk(l_var1));

// Get the second variable (a double matrix)
GetListRhsVar(1,3,"d",&m_var2,&n_var2,&l_var2);
sciprint("var2 = [%f %f %f %f]\n", *stk(l_var2+0),
                                         *stk(l_var2+1),
                                         *stk(l_var2+2),
                                         *stk(l_var2+3));

return 0;
}

```

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function. The C file is available in the example directory. It is named `print_list.c`.

The builder script

```

// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name = 'lib_print_list';

files = ['print_list.c'];

libs = [];

table=['print_list', 'sci_print_list'];

ldflags = "";
cflags = "";
fflags = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
</programlisting>

<para>This file must be saved as "builder.sce".</para>

<para>This script will tell Scilab which files must be compiled (here,
it's print_list.c), what will be the name of the shared library (here,
it's lib_print_list) and which C symbol will be linked to a Scilab
function (here, we will link the sci_print_list C symbol to the Scilab
function "print_list").</para>

<para>To build this function, we just need to to:</para>

<programlisting role = "">#!/CDATA[
exec builder.sce;

```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce. Let's test our new function:

```
exec builder.sce;
exec loader.sce;

A = mlist(['mytype', 'var1', 'var2'], 'a string', [1 2; 3 4]);

print_list(A);
```

The C function - creation of a [mt]list

This example is available in SCI/modules/core/example/create_list.

We now write a simple example to test our new function to create a [mt]list.

```
A = create_list();

disp(A);
```

First, let's write the C function:

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
    int m_list_out, n_list_out;
    int m_var1,      n_var1,      l_var1,  l_list_var1;
    int m_var2,      n_var2,      l_var2,  l_list_var2;
    int m_mlist,     n_mlist,     l_mlist;

    // The labels of our mlist
    static const char * ListLabels [] = {"mylist", "var1", "var2"};

    // First, we create the variables using a classical way
    // The size of the Scilab variables
    m_var1 = 1; n_var1 = strlen("a string")+1; // a null terminated string
    m_var2 = 2; n_var2 = 2; // A 2x2 double matrix
    m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

    // Creation of the Scilab variables
    // A('var1')
    CreateVar(1, "c", &m_var1, &n_var1, &l_var1);
    // A('var2')
    CreateVar(2, "d", &m_var2, &n_var2, &l_var2);
    // A
    CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

    // We store values in the create variables
    // The matrix will be stored in A('var2')
    *stk(l_var2+0) = 1;
    *stk(l_var2+1) = 2;
    *stk(l_var2+2) = 3;
    *stk(l_var2+3) = 4;
```

```

// The string will be stored in A('var1')
strncpy(cstk(l_var1),"a string\0",n_var1);

m_list_out = 3; n_list_out = 1;

// now, affect the variable to the mlist
// The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...
CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
// The value stored in A('var1') (it corresponds to A = ..., 'a string', ...
CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
// The value stored in A('var2') (it corresponds to A = ..., [1 2,3 4]);
CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

// We return only the mlist which has been created at position 3
LhsVar(1) = 3;

return 0;
}

```

Some important comments related to the `CreateVar(Pos,"m",&m, &n, &l)` function. When called on a mlist, only the `m` parameter is taken in account, the `n` parameter is not used. So, be careful:

```

m_list = 3; n_list = 1;
CreateVar(1, "m", &m_list, &n_list, &l_list);

```

creates a mlist with 3 elements but:

```

m_list = 1; n_list = 3;
CreateVar(1, "m", &m_list, &n_list, &l_list);

```

creates a mlist with only 1 element !

Another important thing: when we create a list element using `CreateListVarFrom`, it is not recommended to access the created variable using, for example, `stk(l_list_var2)` because `CreateListVarFrom` performs type transformation on the list variables.

The builder script

```

// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name = 'lib_create_list';

files = ['create_list.c'];

libs = [];

table = ['create_list', 'sci_create_list'];

ldflags = "";

```



```
cflags = "";  
fflags = "";  
  
ilib_build(ilib_name, table, files, libs, 'MakeLib', ldflags, cflags, fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's create_list.c), what will be the name of the shared library (here, it's lib_create_list) and which C symbol will be linked to a Scilab function (here, we will link the sci_create_list C symbol to the Scilab function "create_list").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce. Let's test our new function:

```
exec builder.sce;  
exec loader.sce;  
  
A = create_list();  
  
disp(typeof(A))  
disp(getfield(1,A))  
disp(A('var1'))  
disp(A('var2'))
```

See Also

GetRhsVar, GetListRhsVar, CreateVar, CreateListVarFrom, CreateListVarFromPtr, LhsVar, stk, ilib_build

Name

How to deal with optional parameters — how to deal with optional parameters send to an interface using the C gateway functions

Description

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

The goal is to get a set of optional parameters via a C gateway and then to perform some checks in the C function (number of optional parameters, does an optional parameters exists, etc.).

This example is available in the directory `core/examples/optional_parameters`

The C function

```
#include <stack-c.h>

int ex2c(double * a, int * ma, int * na,
         double * b, int * mb, int * nb)
{
    int i;

    for(i=0;i<(*ma)*(*na);i++) a[i] = 2*a[i];
    for(i=0;i<(*mb)*(*nb);i++) b[i] = 3*b[i];

    return(0);
}

int sci_optional_parameters(char * fname)
{
    int m1,n1,l1;

    // optional names must be stored in alphabetical order in opts
    static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                             {-1,"v2","d",0,0,0},
                             {-1,NULL,NULL,0,0}};

    int minrhs = 1, maxrhs = 1;
    int minlhs = 1, maxlhs = 3;
    int nopt, iopos, res;
    char buffer_name[csiz]; // csiz used for character coding

    nopt = NumOpt();

    CheckRhs(minrhs,maxrhs+nopt);
    CheckLhs(minlhs,maxlhs);

    // first non optional argument
    GetRhsVar( 1, "c", &m1, &n1, &l1);

    if (get_optionals(fname,opts)==0) return 0;

    // default values if optional arguments are not given: v1=[99] and v2=[3]

    sciprint("number of optional parameters = %d\n", NumOpt());
    sciprint("first optional parameters = %d\n", FirstOpt());
}
```

```

sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

iopos = Rhs;

if (opts[0].position==-1)
{
    iopos++;
    opts[0].position = iopos;
    opts[0].m = 1; opts[0].n = 1;
    opts[0].type = "d";
    CreateVar(opts[0].position, opts[0].type, &opts[0].m, &opts[0].n, &opts[0].l);
    *stk(opts[0].l) = 99.0;
}

if (opts[1].position==-1)
{
    iopos++ ;
    opts[1].position = iopos;
    opts[1].m = 1; opts[1].n = 1;
    opts[1].type = "d";
    CreateVar(opts[1].position, opts[1].type, &opts[1].m, &opts[1].n, &opts[1].l);
    *stk(opts[1].l) = 3;
}

ex2c(stk(opts[0].l),&opts[0].m,&opts[0].n,
      stk(opts[1].l),&opts[1].m,&opts[1].n);

// return the first argument (unchanged ) then v1 and v2

LhsVar(1) = 1;
LhsVar(2) = opts[0].position;
LhsVar(3) = opts[1].position;

return 0;
}

```

This file must be saved as "optional_parameters.c".

The main thing to highlight is that, to build a C interface function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C interface functions. We also need to include sciprint.h because we use the sciprint function.

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

The builder script

```
// This is the builder.sce
```

```
// must be run from this directory

lines(0);

ilib_name = 'lib_optional_parameters';

files = ['optional_parameters.c'];

libs = [];

table = ['optional_parameters', 'sci_optional_parameters'];

ldflags = "";
cflags = "";
fflags = "";

ilib_build(ilib_name, table, files, libs, 'MakeLib', ldflags, cflags, fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's optional_parameters.c), what will be the name of the shared library (here, it's lib_optional_parameters) and which C symbol will be linked to a Scilab function (here, we will link the sci_optional_parameters C symbol to the Scilab function "optional_parameters").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

Testing our new function

We now write a simple example to test our new functions.

```
// Example with optional argument specified with the 'arg=value syntax'
// [a,b,c] = ex12c(x1, [v1 = arg1, v2 = arg2]), arg1 default value 99
//                                           arg2 default value 3
// only v1 and v2 are recognized as optional argument names
// the return value are a = x1, b = 2*v2, c = 3*v2

[a,b,c] = optional_parameters('test');
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));

[a,b,c] = optional_parameters('test',v1=[10,20]);
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));

[a,b,c] = optional_parameters('test',v1=[10,20],v2=8);
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));
```

```
[a,b,c] = optional_parameters('test',v2=8,v1=[10]);  
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));
```

The script must be saved as "optional_parameters.sce".

Let's run our scripts and see what is the result:

```
-->;exec builder.sce;  
  Génère un fichier gateway  
  Génère un fichier loader  
  Génère un Makefile : Makelib  
  Exécute le makefile  
  Compilation de optional_parameters.c  
  Construction de la bibliothèque partagée (soyez patient)  
  
-->;exec loader.sce;  
Bibliothèque partagée chargée.  
Link done.  
  
-->;exec optional_parameters.sce;  
number of optional parameters = 0  
first optional parameters = 2  
FindOpt(v1) = 0  
FindOpt(v2) = 0  
  
a = test b = 198 c = 9  
number of optional parameters = 1  
first optional parameters = 2  
FindOpt(v1) = 2  
FindOpt(v2) = 0  
parameter 2 is optional: v1  
  
!a = test b = 20 c = 9 a = test b = 40 c = 9 !  
number of optional parameters = 2  
first optional parameters = 2  
FindOpt(v1) = 2  
FindOpt(v2) = 3  
parameter 2 is optional: v1  
parameter 3 is optional: v2  
  
!a = test b = 20 c = 24 a = test b = 40 c = 24 !  
number of optional parameters = 2  
first optional parameters = 2  
FindOpt(v1) = 3  
FindOpt(v2) = 2  
parameter 2 is optional: v2  
parameter 3 is optional: v1  
  
a = test b = 20 c = 24
```

See Also

GetRhsVar, CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, OverLoad, ilib_build

Name

CheckColumn — C interface function which checks if a parameter send to the C function is a column vector or not

```
CheckColumn(StackPos,m_var,n_var)
```

Parameters

StackPos

the position in the Scilab memory of the argument for which we want to perform the check (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory

n_var

the number of columns of the parameter at position StackPos in the Scilab memory

Description

C interface function which checks if a parameter send to the C function is a column vector or not. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckColumn(1,m1,n1); // Check that first argument is a column vector

    return 0;
}
```

See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CheckDimProp — C interface function which checks the compatibility between 2 arguments send to the C function

```
CheckColumn(StackPos_var1, StackPos_var2, expression)
```

Parameters

StackPos_var1

the position in the Scilab memory of the first parameter for which we want to perform the check (input parameter)

StackPos_var2

the position in the Scilab memory of the second variable for which we want to perform the check (input parameter)

expression

a boolean expression which represent the size check we want to perform

Description

C interface function which checks the compatibility between 2 arguments send to the C function. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;
    int m2, n2, l2;

    CheckRhs(2,2);

    GetRhsVar(1, "d", &m1, &n1, &l1);
    GetRhsVar(2, "d", &m2, &n2, &l2);

    // Check compatibility between arg 1 and arg 2. We want m1*n1 == n2
    CheckDimProp(1,2, m1 * n1 != n2);

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CheckDims — C interface function which checks if a parameter send to the C function has the required dimensions

```
CheckDims(StackPos,m_var,n_var,m_required,n_required)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to know the type (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory (input parameter)

n_var

the number of columns of the parameter at position StackPos in the Scilab memory (input parameter)

m_required

the required number of lines (input parameter)

n_required

the required number of columns (input parameter)

Description

C interface function which checks if a parameter send to the C function has the required dimensions. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckDims(1,m1,n1,1,4); // Check that argument is a 1x4 matrix

    return 0;
}
```

See Also

CheckColumn, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CheckLength — C interface function which checks the length of a vector send as a parameter to the C function

```
CheckLength(StackPos,m_var,m_required)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to check (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory (input parameter)

m_required

the required number of lines (input parameter)

Description

C interface function which checks the length of a vector send as a parameter to the C function. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckLength(1,m1*n1,4); // Check that vector has 4 elements

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDim-Prop, CheckSameDims, CheckSquare, How to check parameters

Name

CheckLhs — C macro which checks the number of output arguments present in the calling Scilab function.

```
CheckLhs(nb_min_params, nb_max_params)
```

Parameters

nb_min_params

the minimum number of output arguments which must be present in the calling Scilab function

nb_max_params

the maximum number of output arguments which must be present in the calling Scilab function

Description

C macro which checks the number of output arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

If the number of arguments is not between nb_min_params and nb_max_params, we quit the C interface (`return 0;`) and an error is returned in the Scilab console.

Since CheckLhs is doing a `return 0;` within the gateway function, it is important to call this macro before any memory allocation in order to avoid any memory leak.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C gateway function checks for a number of output arguments which must be between 2 and 3.

```
#include <stack-c.h>

int sci_mychecklhs(char * fname)
{
    CheckLhs(2,3);

    return 0;
}
```

Now, some functions testing this interface:

```
[A,B] = mychecklhs(); // OK, 2 output arguments
[A,B,C] = mychecklhs(); // OK, 3 output arguments
[A] = mychecklhs(); // ERROR, 1 output argument
[A,B,C,D] = mychecklhs(); // ERROR, 4 output arguments
```

See Also

CheckRhs

Name

CheckRhs — C macro which checks the number of input arguments present in the calling Scilab function.

```
CheckRhs(nb_min_params, nb_max_params)
```

Parameters

nb_min_params

The minimum number of input arguments which must be present in the calling Scilab function

nb_max_params

the maximum number of input arguments which must be present in the calling Scilab function

Description

C macro which checks the number of input arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

If the number of input arguments is not between nb_min_params and nb_max_params, we quit the C interface (`return 0;`) and an error is returned in the Scilab console.

Since CheckRhs is doing a `return 0;` within the gateway function, it is important to call this macro before any memory allocation in order to avoid any memory leak.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C gateway function checks for a number of input arguments which must be between 2 and 3.

```
#include <stack-c.h>

int sci_mycheckrhs(char * fname)
{
    CheckRhs(2,3);

    return 0;
}
```

Now, some functions testing this interface:

```
mycheckrhs(A,B); // OK, 2 output arguments
mycheckrhs(A,B,C); // OK, 3 output arguments
mycheckrhs(A); // ERROR, 1 output argument
mycheckrhs(A,B,C,D); // ERROR, 4 output arguments
```

See Also

CheckLhs

Name

CheckRow — C interface function which checks if a parameter send to the C function is a row vector or not

```
CheckRow(StackPos,m_var,n_var)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to know the type (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory

n_var

the number of columns of the parameter at position StackPos in the Scilab memory

Description

C interface function which checks if a parameter send to the C function is a row vector or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckRow(1,m1,n1); // Check that first argument is a row vector

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CheckSameDims — C interface function which checks if two parameters send to the C function have the same size

```
CheckSameDims(StackPos_var1, StackPos_var2, m_var1, n_var1, m_var2, n_var2)
```

Parameters

StackPos_var1

the position in the Scilab memory of the first parameter for which we want to perform the check (input parameter)

StackPos_var2

the position in the Scilab memory of the second parameter for which we want to perform the check (input parameter)

m_var1

the number of lines of the parameter at position StackPos_var1 in the Scilab memory

n_var1

the number of columns of the parameter at position StackPos_var1 in the Scilab memory

m_var2

the number of lines of the parameter at position StackPos_var2 in the Scilab memory

n_var2

the number of columns of the parameter at position StackPos_var2 in the Scilab memory

Description

C interface function which checks if two parameters send to the C function have the same size. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;
    int m2, n2, l2;

    CheckRhs(2,2);

    GetRhsVar(1, "d", &m1, &n1, &l1);
    GetRhsVar(2, "d", &m2, &n2, &l2);

    CheckSameDims(1,2,m1,n1,m2,n2); // Check that both vectors have the same size
```

```
    return 0;  
}
```

See Also

[CheckColumn](#), [CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDim-Prop](#), [CheckLength](#), [CheckSquare](#), [How to check parameters](#)

Name

CheckScalar — C interface function which checks if a parameter send to the C function is a scalar or not

```
CheckScalar(StackPos,m_var,n_var)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory

n_var

the number of columns of the parameter at position StackPos in the Scilab memory

Description

C interface function which checks if a parameter send to the C function is a scalar or not. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckScalar(1,m1,n1); // Check that first argument is a scalar

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckRow, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CheckSquare — C interface function which checks if a parameter send to the C function is a square matrix or not

```
CheckSquare(StackPos,m_var,n_var)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory

n_var

the number of columns of the parameter at position StackPos in the Scilab memory

Description

C interface function which checks if a parameter send to the C function is a square matrix or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckSquare(1,m1,n1); // Check that first argument is a square matrix

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDim-Prop, CheckLength, CheckSameDims, How to check parameters

Name

CheckVector — C interface function which checks if a parameter send to the C function is a vector (column or row) or not

```
CheckVector(StackPos,m_var,n_var)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var

the number of lines of the parameter at position StackPos in the Scilab memory

n_var

the number of columns of the parameter at position StackPos in the Scilab memory

Description

C interface function which checks if a parameter send to the C function is a vector (column or row) or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
    int m1, n1, l1;

    CheckRhs(1,1);

    GetRhsVar(1, "d", &m1, &n1, &l1);

    CheckVector(1,m1,n1); // Check that first argument is a vector (column or row)

    return 0;
}
```

See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

CreateListVarFrom — a C interface function which allows to create a new Scilab parameter in a [mt]list

```
CreateListVarFrom(StackPos,Type, &m_rows, &n_cols, &l_stack_list_pos, &l_stack_pos)
```

Parameters

StackPos

the rank of the parameter to be created (input parameter)

Type

the Scilab type of the parameter to be created (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows

the number of lines of the matrix to be created (input parameter)

n_cols

the number of columns of the matrix to be created (input parameter)

l_stack_list_pos

the position in the Scilab memory of the created variable in the list(output parameter)

l_stack_pos

the position in the Scilab memory of the created variable (input parameter)

Description

A C interface function which allows to create a new Scilab variable in a [mt]list

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0.
Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
    int m_list_out, n_list_out;
    int m_var1,      n_var1,      l_var1,  l_list_var1;
    int m_var2,      n_var2,      l_var2,  l_list_var2;
    int m_mlist,     n_mlist,     l_mlist;

    // The labels of our mlist
    static const char * ListLabels [] = {"mylist","var1","var2"};

    // First, we create the variables using a classical way
    // The size of the Scilab variables
    m_var1 = 1; n_var1 = strlen("a string")+1; // a null terminated string
    m_var2 = 2; n_var2 = 2; // A 2x2 double matrix
    m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

    // Creation of the Scilab variables
    // A('var1')
    CreateVar(1, "c", &m_var1, &n_var1, &l_var1);
    // A('var2')
    CreateVar(2, "d", &m_var2, &n_var2, &l_var2);
    // A
    CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

    // We store values in the create variables
    // The matrix will be stored in A('var2')
    *stk(l_var2+0) = 1;
    *stk(l_var2+1) = 2;
    *stk(l_var2+2) = 3;
    *stk(l_var2+3) = 4;

    // The string will be stored in A('var1')
    strncpy(cstk(l_var1),"a string\0",n_var1);

    m_list_out = 3; n_list_out = 1;

    // now, affect the variable to the mlist
    // The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...)
    CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
    // The value stored in A('var1') (it corresponds to A = ..., 'a string', ...)
    CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
    // The value stored in A('var2') (it corresponds to A = ..., [1 2, 3 4]);
    CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

    // We return only the mlist which has been created at position 3
    LhsVar(1) = 3;

    return 0;
}
```

This example is available in `SCI/modules/core/example/create_list`.

See Also

Scilab C Type, `istk`, `LhsVar`, `CreateVar`

Name

CreateListVarFromPtr — a C interface function which allows to create a new Scilab parameter from a pointer in a [mt]list

```
CreateListVarFrom(StackPos, Type, &m_rows, &n_cols, &l_stack_list_pos, void * P
```

Parameters

StackPos

the rank of the parameter to be created (input parameter)

Type

the Scilab type of the parameter to be created (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows

the number of lines of the matrix to be created (input parameter)

n_cols

the number of columns of the matrix to be created (input parameter)

l_stack_list_pos

the position in the Scilab memory of the created parameter in the list (output parameter)

Pointer

the pointer to the data area (input parameter)

Description

A C interface function which allows to create a new Scilab parameter from a pointer in a [mt]list

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
    int m_list_out, n_list_out;
    int m_var1,      n_var1,      l_var1,  l_list_var1;
    int m_var2,      n_var2,      l_var2,  l_list_var2;
    int m_mlist,     n_mlist,     l_mlist;

    // The labels of our mlist
    static const char * ListLabels [] = {"mylist","var1","var2"};

    // First, we create the variables using a classical way
    // The size of the Scilab variables
    m_var1 = 1; n_var1 = strlen("a string")+1; // a null terminated string
    m_var2 = 2; n_var2 = 2; // A 2x2 double matrix
    m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

    // Creation of the Scilab variables
    // A('var1')
    CreateVar(1, "c", &m_var1, &n_var1, &l_var1);
    // A('var2')
    CreateVar(2, "d", &m_var2, &n_var2, &l_var2);
    // A
    CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

    // We store values in the create variables
    // The matrix will be stored in A('var2')
    *stk(l_var2+0) = 1;
    *stk(l_var2+1) = 2;
    *stk(l_var2+2) = 3;
    *stk(l_var2+3) = 4;

    // The string will be stored in A('var1')
    strncpy(cstk(l_var1),"a string\0",n_var1);

    m_list_out = 3; n_list_out = 1;

    // now, affect the variable to the mlist
    // The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...
    CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
    // The value stored in A('var1') (it corresponds to A = ..., 'a string', ...
    CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
    // The value stored in A('var2') (it corresponds to A = ..., [1 2, 3 4]);
    CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

    // We return only the mlist which has been created at position 3
    LhsVar(1) = 3;

    return 0;
}
```

This example is available in `SCI/modules/core/example/create_list`.

See Also

Scilab C Type, `istk`, `LhsVar`, `CreateVar`

Name

CreateVar — a C gateway function which allows to create a new Scilab parameter

```
CreateVar(StackPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

Parameters

StackPos

The rank of the parameter to be created (input argument)

Type

The Scilab C Type of the parameter to be created (input argument).

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows

the number of lines of the matrix to be created (input argument)

n_cols

the number of columns of the matrix to be created (input argument)

l_stack_pos

the position in the Scilab memory of the created parameter (output argument)

Description

A C gateway function which allows to create a new Scilab parameter

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
    int m_row, n_col, l_pos;

    m_row = 1; n_col = 1; // We create a scalar
    CreateVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_row, &n_col, &l_pos);

    *istk(l_pos) = 1;

    LhsVar(1) = 1;

    return 0;
}
```

See Also

Scilab C Type, istk, LhsVar

Name

FindOpt — C gateway function find the position of an optional argument given its name

```
Pos FindOpt(varname, opts)
```

Parameters

varname

the name of the optional parameter

opts

a C list of optional parameters

```
typedef struct rhs_opts__ {  
    int position ; // stack position : -1 if not present  
    char *name; // the name of the variable  
    char *type; // a Scilab type (like "d") representing the type of the variable  
    int m,n; // the size of the variable  
    unsigned long int l; // a pointer to the Scilab stack  
} rhs_opts;
```

Pos

the rank of the optional parameter if it has been found in the parameters sent to the C function, 0 otherwise.

Description

A C gateway function which find the position of an optional argument given its name. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>  
  
int sci_optional_parameters(char * fname)  
{  
    int m1,n1,l1;  
  
    // optional names must be stored in alphabetical order in opts  
    static rhs_opts opts[] = {{-1,"v1","d",0,0,0},  
                              {-1,"v2","d",0,0,0},  
                              {-1,NULL,NULL,0,0}};  
  
    int minrhs = 1, maxrhs = 1;  
    int minlhs = 1, maxlhs = 3;  
    int nopt, iopos, res;  
    char buffer_name[csiz]; // csiz used for character coding
```

```
nopt = NumOpt();

CheckRhs(minrhs,maxrhs+nopt);
CheckLhs(minlhs,maxlhs);

// first non optional argument
GetRhsVar( 1, "c", &m1, &n1, &l1);

if (get_optionals(fname,opts)==0) return 0;

sciprint("number of optional parameters = %d\n", NumOpt());
sciprint("first optional parameters = %d\n", FirstOpt());
sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

return 0;
}
```

See Also

[CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDimProp](#), [CheckLength](#), [CheckSameDims](#), [CheckSquare](#), [How to check parameters](#)

Name

FirstOpt — C gateway function which returns the position of the first optional parameter

```
Pos FirstOpt()
```

Parameters

Pos

the position of the first optional parameter, Rhs + 1 if no optional parameters have been given to the function

Description

A C gateway function which returns the position of the first optional parameter. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
    int m1,n1,l1;

    // optional names must be stored in alphabetical order in opts
    static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                             {-1,"v2","d",0,0,0},
                             {-1,NULL,NULL,0,0}};

    int minrhs = 1, maxrhs = 1;
    int minlhs = 1, maxlhs = 3;
    int nopt, iopos, res;
    char buffer_name[csiz]; // csiz used for character coding

    nopt = NumOpt();

    CheckRhs(minrhs,maxrhs+nopt);
    CheckLhs(minlhs,maxlhs);

    // first non optional argument
    GetRhsVar( 1, "c", &m1, &n1, &l1);

    if (get_optionals(fname,opts)==0) return 0;

    sciprint("number of optional parameters = %d\n", NumOpt());
    sciprint("first optional parameters = %d\n", FirstOpt());
    sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
    sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));
```

```
if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

return 0;
}
```

See Also

[CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDimProp](#), [CheckLength](#), [CheckSameDims](#), [CheckSquare](#), [How to check parameters](#)

Name

GetListRhsVar — a C gateway function which allows to access a parameter stored in a [mt]list transmitted to a Scilab function

```
GetListRhsVar(StackPos, ListPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

Parameters

StackPos

the rank of the [mt]list to be accessed (input parameter)

ListPos

the rank in the list of the parameter to be accessed (input parameter)

Type

the Scilab type of the parameter to be accessed (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows

the number of lines of the accessed parameter (output parameter)

n_cols

the number of columns of the accessed parameter (output parameter)

l_stack_pos

the position on the stack of the accessed parameter (output parameter)

Description

A C gateway function which allows to access a parameter stored in a [mt]list transmitted to a Scilab function

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the function has one input parameter. It gets a mlist and the prints some informations related to the content of the mlist.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_print_list(char * fname)
{
    int m_list_in, n_list_in, l_list_in;
    int m_type,    n_type;
    int m_var1,    n_var1,    l_var1;
    int m_var2,    n_var2,    l_var2;
    char ** LabelList = NULL;

    CheckRhs(1,1); // We accept only 1 parameter

    GetRhsVar(1,"m",&m_list_in,&n_list_in,&l_list_in); // Get a mlist

    // Get the type and the name of the variables (the first element of the mlist
    GetListRhsVar(1,1,"S",&m_type,&n_type,&LabelList);

    if (strcmp(LabelList[0],"mytype")!=0)
    {
        sciprint("error, you must ship a mlist or type mytype\n");
        return 0;
    }

    // Get the first variable (a string)
    GetListRhsVar(1,2,"c",&m_var1,&n_var1,&l_var1);
    sciprint("var1 = %s\n",cstk(l_var1));

    // Get the second variable (a double matrix)
    GetListRhsVar(1,3,"d",&m_var2,&n_var2,&l_var2);
    sciprint("var2 = [%f %f %f %f]\n",*stk(l_var2+0),
                                                *stk(l_var2+1),
                                                *stk(l_var2+2),
                                                *stk(l_var2+3));

    return 0;
}
```

This example is available in SCI/modules/core/example/print_list.

See Also

Scilab C Type, istk, LhsVar

Name

GetRhsVar — a C gateway function which allows to access an argument transmitted to a Scilab function

```
GetRhsVar(StackPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

Parameters

StackPos

The rank of the variable to be accessed (input argument)

Type

The Scilab C Type of the parameter to be accessed (input argument).

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows

the number of lines of the accessed parameter (output argument)

n_cols

the number of columns of the accessed parameter (output argument)

l_stack_pos

the position in the Scilab memory of the accessed parameter (output argument)

Description

A C gateway function which allows to access a argument transmitted to a Scilab function

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the function has two input arguments:

- the number of columns (first argument)
- the number of lines (second argument)

The goal of this function is to create a matrix of integers equal to 1.

```
#include <stack-c.h>
#include <string.h>

int sci_myones(char * fname)
{
    int m_param_1, n_param_1, l_param_1;
    int m_param_2, n_param_2, l_param_2;
    int m_out_row, n_out_col, l_out_pos;
    int i;
    int * pOutPos = NULL;

    GetRhsVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_param_1, &n_param_1, &l_param_1);
    GetRhsVar(2, MATRIX_OF_INTEGER_DATATYPE, &m_param_2, &n_param_2, &l_param_2);

    // We create a matrix of ints equal to 1
    m_out_row = *istk(l_param_1); // The first dimension of the matrix to be created
                                // is stored in the first input parameter
    n_out_col = *istk(l_param_2); // The second dimension of the matrix to be created
                                // is stored in the second input parameter

    CreateVar(3, MATRIX_OF_INTEGER_DATATYPE, &m_out_row, &n_out_col, &l_out_pos);

    pOutPos = istk(l_out_pos); // Get a pointer to the area allocated by CreateVar
    for(i=0; i<m_out_row*n_out_col; i++) pOutPos[i] = 1;

    // A concise way to write the preceding line of code:
    // for(i=0; i<m_out_row*n_out_col; i++) *istk(l_out_pos+i) = 1;

    LhsVar(1) = 3; // We return the 3rd Scilab variable of our gateway

    return 0;
}
```

See Also

Scilab C Type, istk, LhsVar

Name

GetType — C gateway function which returns the type of a parameter in the Scilab memory

```
SciType GetType(StackPos)
```

Parameters

StackPos

the position on the Scilab memory of the parameter for which we want to know the type (input argument)

SciType

the type (defined in the `sci_types` enum which you can find in `stack-c.h`) of the parameter at position `StackPos` in the Scilab memory

Description

GetType is a C gateway function which returns the type of a parameter in the Scilab memory. You must include `stack-c.h` to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C gateway function takes one input argument and prints the integer corresponding to the type of the variable sent as argument in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mygettype(char * fname)
{
    sciprint("The type of the first argument is %d\n", GetType(1));

    return 0;
}
```

See Also

`sciprint`, `sci_types`

Name

IsOpt — C gateway function which checks if a parameter is optional and returns the name of the parameter

```
Res IsOpt(Pos,buffer_name)
```

Parameters

Pos

the position in the Scilab memory of the parameter to be checked (input parameter)

buffer_name

an array of char (of size csiz which correspond to the maximum length of a Scilab variable) in which the name of the parameter will be copied if the position Pos is optional (output parameter)

Res

1 if Pos has an optional parameter, 0 otherwise

Description

C gateway function which checks if a parameter is optional and returns the name of the parameter. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
    int m1,n1,l1;

    // optional names must be stored in alphabetical order in opts
    static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                             {-1,"v2","d",0,0,0},
                             {-1,NULL,NULL,0,0}};

    int minrhs = 1, maxrhs = 1;
    int minlhs = 1, maxlhs = 3;
    int nopt, iopos, res;
    char buffer_name[csiz]; // csiz used for character coding

    nopt = NumOpt();

    CheckRhs(minrhs,maxrhs+nopt);
    CheckLhs(minlhs,maxlhs);

    // first non optional argument
    GetRhsVar( 1, "c", &m1, &n1, &l1);
```

```
if (get_optionals(fname,opts)==0) return 0;

sciprint("number of optional parameters = %d\n", NumOpt());
sciprint("first optional parameters = %d\n", FirstOpt());
sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

return 0;
}
```

See Also

[CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDimProp](#), [CheckLength](#), [CheckSameDims](#), [CheckSquare](#), [How to check parameters](#)

Name

Lhs — A C gateway function which provides the number of output arguments present in the calling Scilab function

```
nb_params Lhs
```

Parameters

nb_params
the number of output arguments present in the calling Scilab function

Description

Lhs provides a C gateway function which provides the number of output arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

Note: Lhs means Left Hand Side.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C gateway function can take several output arguments and prints in the Scilab console the integer corresponding to the number of output arguments detected in the calling Scilab function.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mylhs(char * fname)
{
    sciprint("The number of output arguments is %d\n", Lhs);

    return 0;
}
```

See Also

sciprint, Rhs

Name

LhsVar — a C gateway function which specifies which parameters created inside the C gateway will be returned as an output argument into Scilab.

```
LhsVar(RankPos) = RankVar;
```

Parameters

RankPos

as integer providing the rank of the output argument

RankVar

the rank of the parameter created inside the C gateway to be returned as an Scilab output argument

Description

A C gateway function which specifies which variables created inside the C interface will be returned as an output argument into Scilab.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

This example takes a matrix of doubles as input and returns:

- the number of lines (first output argument)
- the number of rows (second output argument)

We create an intermediate Scilab parameter which will handle an integer but will neither be used nor returned as an output argument.

TODO: insert an example in the Scilab language

```
#include <stack-c.h>

int sci_mysizedouble(char * fname)
{
    int m_in_row,          n_in_col,          l_in_pos;
    int m_out_lines_row,   n_out_lines_col,   l_out_lines_pos;
    int m_out_columns_row, n_out_columns_col, l_out_columns_pos;
    int m_nop,             n_nop,             l_nop;

    GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_in_row, &n_in_col, &l_in_pos);

    m_out_lines_row   = 1; n_out_lines_col   = 1; // We create a scalar
    m_out_columns_row = 1; n_out_columns_col = 1; // We create a scalar
    m_nop              = 1; n_nop              = 1; // We create a scalar

    CreateVar(2, MATRIX_OF_INTEGER_DATATYPE, &m_out_lines_row, &n_out_lines_col,
    CreateVar(3, MATRIX_OF_INTEGER_DATATYPE, &m_nop, &n_nop,
    CreateVar(4, MATRIX_OF_INTEGER_DATATYPE, &m_out_columns_row, &n_out_columns_col,

    *istk(l_out_lines_pos) = m_in_row; // the out_lines_pos parameter handles t
    *istk(l_nop)           = 1; // store a mere value, but will neither be used
```

```
*istk(l_out_columns_pos) = n_in_col; // the out_columns_pos parameter handles  
  
LhsVar(1) = 2; // We set the parameter 2 as an output argument  
LhsVar(2) = 4; // We set the parameter 4 as an output argument  
  
return 0;  
}
```

See Also

Scilab C Type, istk, CreateVar, GetRhsVar

Name

NumOpt — C gateway function which returns the number of optional parameters sent to a C function

```
Res = NumOpt()
```

Parameters

Res
the number of optional parameters detected

Description

A C gateway function which returns the number of optional parameters sent to a C function. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
    int m1,n1,l1;

    // optional names must be stored in alphabetical order in opts
    static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                             {-1,"v2","d",0,0,0},
                             {-1,NULL,NULL,0,0}};

    int minrhs = 1, maxrhs = 1;
    int minlhs = 1, maxlhs = 3;
    int nopt, iopos, res;
    char buffer_name[csiz]; // csiz used for character coding

    nopt = NumOpt();

    CheckRhs(minrhs,maxrhs+nopt);
    CheckLhs(minlhs,maxlhs);

    // first non optional argument
    GetRhsVar( 1, "c", &m1, &n1, &l1);

    if (get_optionals(fname,opts)==0) return 0;

    sciprint("number of optional parameters = %d\n", NumOpt());
    sciprint("first optional parameters = %d\n", FirstOpt());
    sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
    sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

    if (IsOpt(1,buffer_name))
```



```
    sciprint("parameter 1 is optional: %s\n", buffer_name);  
    if (IsOpt(2,buffer_name))  
        sciprint("parameter 2 is optional: %s\n", buffer_name);  
    if (IsOpt(3,buffer_name))  
        sciprint("parameter 3 is optional: %s\n", buffer_name);  
  
    return 0;  
}
```

See Also

[CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDimProp](#), [CheckLength](#), [CheckSameDims](#), [CheckSquare](#), [How to check parameters](#)

Name

OverLoad — C gateway function which tells Scilab to look for another overloaded function

```
OverLoad(StackPos)
```

Parameters

StackPos

the position in the Scilab memory of the parameter for which we want to take into account for the overloading process (input argument)

Description

A C gateway function which tells Scilab to look for another overloaded function. Scilab then appends to the name of the function a prefix (like %sp_ if the parameter taken into account is sparse) and look for the overloaded function. You must include stack-c.h to benefit from this function.

Be careful with the Scilab name of the function. Indeed, the current overloading process of Scilab works only on Scilab primitives (Scilab function wrote in C) which must have a Scilab name which is maximum 8 char wide.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_check_properties_2(char * fname)
{
    int m1,n1,l1;

    CheckRhs(1,1);
    CheckLhs(0,1) ;

    switch(VarType(1))
    {
        case sci_matrix:
            GetRhsVar(1, "d", &m1, &n1, &l1);
            sciprint("1 is a scalar matrix\n");
            break;
        case sci_strings:
            GetRhsVar(1, "c", &m1, &n1, &l1);
            sciprint("1 is a string\n");
            break;
        case sci_sparse:
            sciprint("1 is a sparse trying to overload\n");
            OverLoad(1);
    }

    LhsVar(1) = 0;
```

```
    return 0;
}
</programlisting>

<para>The builder.sce script look like this:</para>

<programlisting role = "example"><![CDATA[
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_check_properties';

files = ['check_properties.c'];

libs  = [];

table =['chprop2', 'sci_check_properties_2'];

// We must be careful when we choose a scilab function name in case of overload.
// We Scilab name function must be 8 char max.

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

And now, an example of use of this new function:

```
chprop2([1,2,2]);
chprop2('foo');

// overload case

deff('[ ]=%sp_chprop2(sp)','disp(''sparse overloaded'')');
chprop2(sparse([1,2,3]));
```

See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

Name

Rhs — A C gateway function which provides the number of input arguments present in the calling Scilab function

```
nb_params Rhs
```

Parameters

nb_params

the number of input arguments present in the calling Scilab function

Description

A C gateway function which provides the number of input arguments present in the calling Scilab function. You must include `stack-c.h` to benefit from this function.

Note: Rhs means Right Hand Side.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C interface function can take several input arguments and prints in the Scilab console the integer corresponding to the number of input arguments detected in the calling Scilab function.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_myrhs(char * fname)
{
    sciprint("The number of input parameters is %d\n", Rhs);

    return 0;
}
```

See Also

sciprint, Lhs

Name

Scierror — C gateway function which displays an error message to the user (same profil as the printf function) and returns an integer value specifying an error level

```
void Scierror(error_level, format, value_1, ..., value_n)
```

Parameters

error_level

an integer value specifying an error level

format

a char* string. Specifies a character string combining literal characters with conversion specifications.

value_i

Specifies the data to be converted according to the format parameter.

returns

If the operation is successfull, this function returns the number of characters printed (not including the trailing '\0' used to end output to strings).

If an error occurred, a negative value is returned.

Description

Scierror is a C gateway function which displays an error message to the user (same profil as the printf function) and returns an integer value specifying an error level. You must include Scierror.h to benefit from this function. This header is provided in the output_stream module (this directory should be included by default).

Examples

In this example, the C gateway function prints an error message and returns the error level 133.

```
#include <stack-c.h>
#include <Scierror.h>

int sci_myscierror(char * fname)
{
    Scierror(133, "An error has occured: %d\n", 1);

    return 0;
}
```

See Also

printf_conversion, printf, sciprint

Name

Scilab C Types — the C types available in a C gateway

Description

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

The string/char * datatype in the Scilab memory:

- STRING_DATATYPE
- "c"

The string/char ** datatype in the Scilab memory:

- MATRIX_OF_STRING_DATATYPE
- "S"

A matrix of double * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_DOUBLE_DATATYPE
- "d"

A matrix of rational * if the size of the matrix is 1,1, it is a single value

- MATRIX_OF_RATIONAL_DATATYPE
- "r"

A matrix of integer * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE
- "I"

A matrix of 'little' integer * 'little' because in reality, this int is a complex with the imaginary * part set to 0 * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_INTEGER_DATATYPE
- "i"

A matrix of boolean * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_BOOLEAN_DATATYPE
- "b"

A matrix of complex * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_COMPLEX_DATATYPE
- "z"

A sparse matrix * if the size of the matrix is 1,1, it is a single value

- SPARSE_MATRIX_DATATYPE
- "s"

A list:

- LIST_DATATYPE
- "l"

A typed list:

- TYPED_LIST_DATATYPE
- "t"

A Matrix oriented typed list * mlist object are very similar to tlist objects. But * if M is an mlist, for any index i which is not a field name, * M(i) is not the ith field of the list but is interpreted as * the ith entry of M seen as a vector. * This is the only difference between mlist and tlist:

- MATRIX_ORIENTED_TYPED_LIST_DATATYPE
- "m"

The scilab pointer datatype in the Scilab memory:

- SCILAB_POINTER_DATATYPE
- "p"

The scilab graphic handle datatype in the Scilab memory:

- GRAPHICAL_HANDLE_DATATYPE
- "h"

An "external" is a function or routine which is used as an argument * of some high-level primitives (such as ode, optim, schur...):

- EXTERNAL_DATATYPE
- "f"

A matrix of polynomial coeff * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_POLYNOMIAL_DATATYPE
- "x"

See Also

mlist, list, tlist, CreateVar, GetRhsVar

Name

get_optionals — C gateway function which initialize the list of optional parameters

```
res get_optionals(fname, opts)
```

Parameters

fname

the name passed to the C gateway. The name of the calling function (of type char *) (input parameter)

opts

a C list of optional parameters

```
typedef struct rhs_opts__ {  
    int position ; // stack position : -1 if not present  
    char *name; // the name of the variable  
    char *type; // a Scilab type (like "d") representing the type of the variable  
    int m,n; // the size of the variable  
    unsigned long int l; // a pointer to the Scilab stack  
} rhs_opts;
```

res

if no optional parameters has been sent, the res = 0. Otherwise, res = 1.

Description

A C gateway function which initialize the list of optional parameters. You must include stack-c.h to benefit from this function.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>  
  
int sci_optional_parameters(char * fname)  
{  
    int m1,n1,l1;  
  
    // optional names must be stored in alphabetical order in opts  
    static rhs_opts opts[] = {{-1,"v1","d",0,0,0},  
                               {-1,"v2","d",0,0,0},  
                               {-1,NULL,NULL,0,0}};  
  
    int minrhs = 1, maxrhs = 1;  
    int minlhs = 1, maxlhs = 3;  
    int nopt, iopos, res;  
    char buffer_name[csiz]; // csiz used for character coding
```



```
nopt = NumOpt();

CheckRhs(minrhs,maxrhs+nopt);
CheckLhs(minlhs,maxlhs);

// first non optional argument
GetRhsVar( 1, "c", &m1, &n1, &l1);

if (get_optionals(fname,opts)==0) return 0;

sciprint("number of optional parameters = %d\n", NumOpt());
sciprint("first optional parameters = %d\n", FirstOpt());
sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

return 0;
}
```

See Also

[CheckDims](#), [CheckRow](#), [CheckScalar](#), [CheckVector](#), [CheckOverLoad](#), [CheckDimProp](#), [CheckLength](#), [CheckSameDims](#), [CheckSquare](#), [How to check parameters](#)

Name

istk — Return a pointer to an integer to access data stored at a given position in the Scilab memory

```
istk(l_stack_pos)
```

Parameters

`l_stack_pos`

the position in the Scilab memory of the parameter to be accessed (input argument)

Description

This C gateway function returns a pointer to an integer to access data stored at a given position in the Scilab memory.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
    int m_row, n_col, l_pos;

    m_row = 1; n_col = 1; // We create a scalar
    CreateVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_row, &n_col, &l_pos);

    *istk(l_pos) = 1; // We store the value 1 in the memory allocated for output
                    // parameter 1

    LhsVar(1) = 1;

    return 0;
}
```

See Also

Scilab C Type, CreateVar, LhsVar

Name

`sci_types` — a C enumeration which defines the types available for a variable

Description

A C enumeration which defines the types available for a variable. You must include `stack-c.h` to benefit from this type definition.

The list of available types is the following:

- 1 - `sci_matrix`: a matrix of doubles
- 2 - `sci_poly`: a polynomials matrix
- 4 - `sci_boolean`: a boolean matrix
- 5 - `sci_sparse`: a sparse matrix
- 6 - `sci_boolean_sparse`: a sparse boolean matrix
- 7 - `sci_matlab_sparse`: a sparse matlab matrix
- 8 - `sci_ints`: a matrix of integers
- 9 - `sci_handles`: a graphical handle
- 10 - `sci_strings`: a matrix of strings
- 11 - `sci_u_function`: an uncompiled Scilab function
- 13 - `sci_c_function`: a compiled Scilab function
- 14 - `sci_lib`: a library of Scilab functions
- 15 - `sci_list`: a Scilab list
- 16 - `sci_tlist`: a Scilab tlist
- 17 - `sci_mlist`: a Scilab mlist
- 128 - `sci_pointer` (was: `sci_lufact_pointer` before Scilab 5.2): a pointer

Integers or enumeration types can be used to check the type of the variables. Using the enumeration type is recommended because of the explicit meaning of the value of the enumeration type.

If this function is used, it is probable that `GetType` will also be used.

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

In this example, the C gateway function takes one argument. Through a switch case structure, we display the type of the variable sent as a parameter.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mysci_typesrhs(char * fname)
{
    switch(GetType(1))
    {
        case sci_matrix:
```

```
        sciprint("A matrix of doubles\n");
        break;
    case sci_poly:
        sciprint("A matrix of polynomials\n");
        break;
    case sci_boolean:
        sciprint("A matrix of booleans\n");
        break;
    case sci_sparse:
        sciprint("A sparse matrix of doubles\n");
        break;
    case sci_boolean_sparse:
        sciprint("A sparse matrix of booleans\n");
        break;
    case sci_matlab_sparse:
        sciprint("A sparse matlab matrix\n");
        break;
    case sci_ints:
        sciprint("A matrix of integers\n");
        break;
    case sci_handles:
        sciprint("A graphic handle\n");
        break;
    case sci_strings:
        sciprint("A matrix of strings\n");
        break;
    case sci_u_function:
        sciprint("An uncompiled Scilab function\n");
        break;
    case sci_c_function:
        sciprint("A compiled Scilab function\n");
        break;
    case sci_lib:
        sciprint("A library of Scilab functions\n");
        break;
    case sci_list:
        sciprint("A Scilab list\n");
        break;
    case sci_tlist:
        sciprint("A Scilab tlist\n");
        break;
    case sci_mlist:
        sciprint("A Scilab mlist\n");
        break;
    case sci_pointer:
    case sci_lufact_pointer: /* Renamed in version 5.2 */
        sciprint("A pointer\n");
        break;
    default:
        sciprint("Unknown type !\n"); // Should never happen
}

return 0;
}
```

See Also

sciprint, GetType

Name

sciprint — A C gateway function which displays standard messages to the user (same profil as the C printf function)

```
void sciprint(format,value_1,...,value_n)
```

Parameters

format

a char* string. Specifies a character string combining literal characters with conversion specifications.

value_i

Specifies the data to be converted according to the format parameter (%s, %d, ...).

Description

This C gateway function provides the capabilities to display messages to the Scilab user. Basically; it emulates the C language printf function. You must include sciprint.h to benefit from this function. This header is provided in the output_stream module (this directory should be included by default).

Note that if you want to trigger an error, the function Scierror is more appropriate.

Examples

In this example, the C gateway function prints several messages illustrating the use of the sciprint function in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mysciprint(char * fname)
{
    sciprint("printing an integer: %d\n", 1);
    sciprint("printing a double:   %f\n", 2.1);
    sciprint("printing a string:   %s\n", "test");

    return 0;
}
```

See Also

printf_conversion, printf, Scierror

Name

stk — Return a pointer to a double to access data stored at a given position in the Scilab memory

```
stk(l_stack_pos)
```

Parameters

l_stack_pos

the position in the Scilab memory of the variable to be accessed (input argument)

Description

This C gateway function returns a pointer to a double to access data stored at a given position in the Scilab memory .

WARNING: This API is deprecated from Scilab 5.2.0 and is going to be removed with Scilab 6.0. Please use API Scilab (the new Scilab API).

Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
    int m_row, n_col, l_pos;

    m_row = 1; n_col = 1; // We create a scalar
    CreateVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_row, &n_col, &l_pos);

    *stk(l_pos) = 1.0; // We store the value 1.0 in the area allocated by CreateV

    LhsVar(1) = 1; // We set the parameter 1 created by CreateVar as an output pa
                  // of the gateway function

    return 0;
}
```

See Also

Scilab C Type, CreateVar, LhsVar

Chapter 12. list_management

Name

Boolean reading (Scilab gateway) — How to read matrix of boolean in a list.

Input argument profile:

```
SciErr getMatrixOfBooleanInList(void* _pvCtx, int* _piParent, int _iItemPos, in
```

Named variable profile:

```
SciErr readMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piPar
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piBool`

Return address of data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of boolean in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
```

```

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
        case sci_matrix :
            iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_poly :
            iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean :
            iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_sparse :
            iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean_sparse :
            iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_ints :
            iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_strings :

```

```

    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem      = 0;
    int* piChild   = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        iTab++;
        iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
        iTab--;
    }
}

```

```

return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    { //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen      = 0;
    int iRows     = 0;
    int iCols     = 0;

    char pstVar[16];
    int* piCoeff  = NULL;
    double** pdblReal = NULL;

```

```

double** pdblImg    = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(_iItemPos == 0)
{
    //not in list
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff    = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {

```

```

    printError(&sciErr, 0);
    return 0;
}

piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
}

if(isVarComplex(pvApiCtx, _piAddr))
{
    sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows = 0;
    int iCols = 0;

    int* piBool = NULL;

```

```

    if(_iItemPos == 0)
    {
        sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
    }
    else
    {
        sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean (%d x %d)\n", iRows, iCols);
    return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        /*Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
        }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))
            {
                sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
            }
            else
            {
                sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
            }
        }
    }

    insert_indent();
    sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);

```

```

    return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData    = NULL;
    short* psData   = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        switch(iPrec)
        {

```



```

case SCI_INT8 :
    sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
    break;
default :
    return 1;
}
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    default :
        return 1;
    }
}
}

if(sciErr.iErr)
{

```

```

    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen      = NULL;

    char **pstData  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }

        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo

```

```

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

```

```
}  
  
insert_indent();  
sciprint("Pointer : 0x%08X\n", pvPtr);  
  
return 0;  
}
```

Scilab test script

```
function read_all()  
d = [1,2,3;4,5,6;7,8,9];common_read(d);  
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4  
b = [%t,%f;%t,%f;%f,%t];common_read(b);  
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r  
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);  
i8 = int8([1,2,3]);common_read(i8);  
ui32 = uint32([3;2;1]);common_read(ui32);  
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);  
if with_module('umfpack') then  
    Cp = taucs_chfact(sp);  
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);  
else  
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));  
end  
common_read(l)  
endfunction  
read_all;
```

Name

Boolean writing (Scilab gateway) — How to add matrix of boolean in a list.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

Named variable profile:

```
SciErr createMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piP
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_piBool`

Address of data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

- `_iRows`
Number of rows of the new variable.
- `_iCols`
Number of columns of the new variable.
- `_piBool`
Returns address of data array (size: `_iRows * _iCols`).
- `SciErr`
Error structure where is stored errors messages history and first error number.

Description

This help describes how to add matrix of boolean in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
    double pdblPoly5[]    = {1,2,3,4,5};
    double pdblPoly6[]    = {-6,-5,-4,-3,-2,-1};
    double *pdblPoly[]    = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
    int piCoef[]          = {1,3,5,2,4,6};
    int piNbItemRow[]     = {1,2,1};
    int piColPos[]        = {8,4,7,2};
    double pdblSReal[]    = {1,2,3,4};
    double pdblSImg[]     = {4,3,2,1};
    int piBool[]          = {1,0,1,0,1,0,1,0,1};
    double* pdblDataPtr   = NULL;

    sciErr = createList(pvApiCtx, 1, 8, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
```

```

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piCoeffs);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, piData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBooleans);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, piData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piData);

```

```
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}
```

Scilab test script

```
size_ref      = 8;
type_ref      = ["constant","string","int16","polynomial", "sparse", "boolean", ""];
dim_ref       = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```


Name

Boolean sparse reading (Scilab gateway) — How to read boolean sparse in a list.

Input argument profile:

```
SciErr getBooleanSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos
```

Named variable profile:

```
SciErr readBooleanSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _p
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piNbItem`

Return number of non zero value.

`_piNbItemRow`

Return number of item in each rows (size: `_iRows`).

`_piColPos`

Return column position for each item (size: `_iNbItem`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read boolean sparse in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}
```

```

}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
        case sci_matrix :
            iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_poly :
            iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean :
            iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_sparse :
            iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean_sparse :

```

```

    iRet = get_bsparsed_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_ints :
    iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_strings :
    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem     = 0;
    int* piChild  = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

        iTab++;
        iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
        iTab--;
    }

    return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))
            {
                sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
            else
            {
                sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
        }
    }

    if(sciErr.iErr)
    {
        {
            printError(&sciErr, 0);
            return 0;
        }
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen      = 0;

```

```

int iRows      = 0;
int iCols      = 0;

char pstVar[16];
int* piCoeff    = NULL;
double** pdblReal = NULL;
double** pdblImg  = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(_iItemPos == 0)
{
    //not in list
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
    else
    {
        sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{

```

```

SciErr sciErr;
int iRows      = 0;
int iCols      = 0;

int* piBool     = NULL;

if(_iItemPos == 0)
{
    sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
}
else
{
    sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Boolean (%d x %d)\n", iRows, iCols);
return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        /*Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
        }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }
}

```

```

        sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

insert_indent();
sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow   = NULL;
    int* piColPos  = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData   = NULL;
    short* psData  = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData  = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {

```



```

    printError(&sciErr, 0);
    return 0;
}

switch(iPrec)
{
case SCI_INT8 :
    sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
    break;
default :
    return 1;
}
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
        break;
    default :

```

```

    return 1;
}
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }

        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);

```

```

    return 0;
}
}
else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }
}

```

```
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Pointer : 0x%08X\n", pvPtr);

return 0;
}
```

Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2, -1, 0, 0, 0; -1, 2, -1, 0, 0; 0, -1, 2, -1, 0; 0, 0, -1, 2, -1; 0, 0, 0, -1, 2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

Name

Boolean sparse writing (Scilab gateway) — How to add boolean sparse matrix in a list.

Input argument profile:

```
SciErr createBooleanSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createBooleanSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int*
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_iNbItem`

Number of non zero itmes in the sparse.

`_piNbItemRow`

Number of item in each rows (size: `_iRows`).

`_piColPos`

Column position for each item (size: `_iNbItem`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add boolean sparse matrix in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1, 3, 5, 2, 4, 6};
    double pdblData2[]    = {6, 4, 2, 5, 3, 1};
```

```

char *pstData[]      = {"may", "be", "the", "with", "puffin", "you"};
short psData[]       = {1, 4, 2, 5, 3, 6};
double pdblPoly1[]   = {1};
double pdblPoly2[]   = {-2, -1};
double pdblPoly3[]   = {1, 2, 3};
double pdblPoly4[]   = {-4, -3, -2, -1};
double pdblPoly5[]   = {1, 2, 3, 4, 5};
double pdblPoly6[]   = {-6, -5, -4, -3, -2, -1};
double *pdblPoly[]   = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
int piCoef[]         = {1, 3, 5, 2, 4, 6};
int piNbItemRow[]    = {1, 2, 1};
int piColPos[]       = {8, 4, 7, 2};
double pdblSReal[]   = {1, 2, 3, 4};
double pdblSImg[]    = {4, 3, 2, 1};
int piBool[]         = {1, 0, 1, 0, 1, 0, 1, 0, 1};
double* pdblDataPtr  = NULL;

sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2, 0);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, pdblPoly);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, pdblSReal, pdblSImg, piBool);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```

```

}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, .
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}

```

Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", ""];
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```


Name

Create List (Scilab gateway) — How to get create a list in Scilab memory.

Input argument profile:

```
SciErr createList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
SciErr createMList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
SciErr createTList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
```

Named variable profile:

```
SciErr createNamedList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAdd
SciErr createNamedTList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAd
SciErr createNamedMList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAd
```

Parameters

_pvCtx
Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
Position in the Scilab memory where you want to put the variable.

_pstName
Name of the variable for "named" functions.

_iNbItem
Number of items in the list.

_piAddress
Return address of the list.

SciErr
Error structure where is stored errors messages history and first error number.

Description

This help describes how to get the address of a list in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
```

```

double pdblPoly5[]      = {1,2,3,4,5};
double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly4};
int piCoef[]            = {1,3,5,2,4,6};
int piNbItemRow[]       = {1,2,1};
int piColPos[]          = {8,4,7,2};
double pdblSReal[]      = {1,2,3,4};
double pdblSImg[]       = {4,3,2,1};
int piBool[]            = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr     = NULL;

sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, RhS + 1, piAddr, 1, 3, 2, piCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, RhS + 1, piAddr, 2, 2, 3, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, RhS + 1, piAddr, 3, 2, 3, psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, RhS + 1, piAddr, 4, "x", 3, 2, piCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, RhS + 1, piAddr, 5, 3, 10, piCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, RhS + 1, piAddr, 6, 3, 3, piBool);
if(sciErr.iErr)
{
    printError(&sciErr, 0);

```

```

    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}

```

Scilab test script

```

size_ref      = 8;
type_ref      = ["constant", "string", "int16", "polynomial", "sparse", "boolean", ""];
dim_ref       = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

```

```
l = list_createlist();  
if size(l) <> size_ref then error("failed"), end  
for i = 1 : size_ref  
    if typeof(l(i)) <> type_ref(i) then error("failed"), end  
    if size(l(i)) <> dim_ref(i) then error("failed"), end  
end
```

Name

Double reading (Scilab gateway) — How to read matrix of double in a list.

Input argument profile:

```
SciErr getMatrixOfDoubleInList(void* _pvCtx, int* _piParent, int _iItemPos, int
```

```
SciErr getComplexMatrixOfDoubleInList(void* _pvCtx, int* _piParent, int _iItemPos,
```

Named variable profile:

```
SciErr readMatrixOfDoubleInNamedList(void* _pvCtx, char* _pstName, int* _piParent,
```

```
SciErr readComplexMatrixOfDoubleInNamedList(void* _pvCtx, char* _pstName, int* _piParent,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_pdblReal`

Return address of real part data array (size: `_iRows * _iCols`).

`_pdblImg`

Return address of imaginary part data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of double in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}
```

```

}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
    case sci_matrix :
        iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_poly :
        iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_boolean :
        iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_sparse :
        iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_boolean_sparse :

```

```

    iRet = get_bsparsed_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_ints :
    iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_strings :
    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem     = 0;
    int* piChild  = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

        iTab++;
        iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
        iTab--;
    }

    return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))
            {
                sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
            else
            {
                sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
        }
    }

    if(sciErr.iErr)
    {
        {
            printError(&sciErr, 0);
            return 0;
        }
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen      = 0;

```



```

int iRows      = 0;
int iCols      = 0;

char pstVar[16];
int* piCoeff    = NULL;
double** pdblReal = NULL;
double** pdblImg  = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(_iItemPos == 0)
{
    //not in list
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
    else
    {
        sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{

```

```

SciErr sciErr;
int iRows      = 0;
int iCols      = 0;

int* piBool     = NULL;

if(_iItemPos == 0)
{
    sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
}
else
{
    sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Boolean (%d x %d)\n", iRows, iCols);
return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        /*Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
        }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }
}

```

```

        sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

insert_indent();
sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow   = NULL;
    int* piColPos  = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData   = NULL;
    short* psData  = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData  = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {

```

```

    printError(&sciErr, 0);
    return 0;
}

switch(iPrec)
{
case SCI_INT8 :
    sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
    break;
default :
    return 1;
}
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
case SCI_INT8 :
    sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
    break;
default :

```

```

    return 1;
}
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }

        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);

```

```

    return 0;
}
}
else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }
}

```

```
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Pointer : 0x%08X\n", pvPtr);

return 0;
}
```

Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2, -1, 0, 0, 0; -1, 2, -1, 0, 0; 0, -1, 2, -1, 0; 0, 0, -1, 2, -1; 0, 0, 0, -1, 2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```


Name

Double writing (Scilab gateway) — How to add matrix of double in a list.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

Named variable profile:

```
SciErr createMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piP
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_pdblReal`

Address of real data array (size: `_iCols * _iRows`).

`_pdblImg`

Address of imaginary data array (size: `_iCols * _iRows`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfDoubleInList(void* _pvCtx, int _iVar, int* _piParent, int _
```

```
SciErr allocComplexMatrixOfDoubleInList(void* _pvCtx, int _iVar, int* _piParent
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_pdblReal`

Return address of real data array (size: `_iCols * _iRows`).

`_pdblImg`

Return address of imaginary data array (size: `_iCols * _iRows`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add matrix of double in a list.

Two types of functions can be used to write in the memory of Scilab.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
    double pdblPoly5[]    = {1,2,3,4,5};
    double pdblPoly6[]    = {-6,-5,-4,-3,-2,-1};
    double *pdblPoly[]    = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
    int piCoef[]           = {1,3,5,2,4,6};
    int piNbItemRow[]     = {1,2,1};
    int piColPos[]         = {8,4,7,2};
    double pdblSReal[]    = {1,2,3,4};
    double pdblSImg[]     = {4,3,2,1};
    int piBool[]           = {1,0,1,0,1,0,1,0,1};
    double* pdblDataPtr    = NULL;

    sciErr = createList(pvApiCtx, 1, 8, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
```

```

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD

```

```
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0] = 1;
pdblDataPtr[1] = 2;
pdblDataPtr[2] = 3;
pdblDataPtr[3] = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}
```

Scilab test script

```
size_ref = 8;
type_ref = ["constant", "string", "int16", "polynomial", "sparse", "boolean", ""];
dim_ref = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

Name

Get child item (Scilab gateway) — How to get the address of a list child.

```
SciErr getListItemAddress(void* _pvCtx, int* _piAddress, int _iItemNum, int** _piItemAddress, SciErr sciErr);
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the list.

`_iItemNum`

Item number.

`_piItemAddress`

Return address of the item.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to get the address of a list child.

Gateway Source

```
int get_list_info(int* _piAddress);
void insert_indent(void);

static int iLocalTab = 0;

int common_list(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_list_info(piAddr);

    LhsVar(1) = 0;
    return 0;
}

int get_list_info(int* _piAddress)
{
    SciErr sciErr;
```

```

int i      = 0;
int iRet   = 0;
int iItem  = 0;

//get list item number, failed if variable is not a kind of list
sciErr = getListItemNumber(pvApiCtx, _piAddress, &iItem);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    sciprint("This variable is not a list");
    return 0;
}

sciprint("List (%d items) -> address : 0x%08X) : \n", iItem, _piAddress);
for(i = 0 ; i < iItem ; i++)
{
    int iType      = 0;
    int* piAddrChild = NULL;
    sciErr = getListItemAddress(pvApiCtx, _piAddress, i + 1, &piAddrChild);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = getVarType(pvApiCtx, piAddrChild, &iType);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(iType == sci_list || iType == sci_tlist || iType == sci_mlist)
    {
        insert_indent();
        sciprint("Child %d -> ", i + 1);
        iLocalTab++;
        iRet = get_list_info(piAddrChild);
        iLocalTab--;
        if(iRet)
        {
            return 1;
        }
    }
    else
    {
        insert_indent();
        sciprint("Child %d -> address : 0x%08X\n", i + 1, piAddrChild);
    }
}
return 0;
}

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iLocalTab ; i++)
    {

```

```
        sciprint("\t");  
    }  
}
```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l5 = list(l1,l2,l3);  
l4 = list(l5, list(l5,l5));  
l6 = uint16([1000,2000,3000]);  
l = list(l1,l2,l3,l6,l4,l5);  
common_list(l)
```

Name

Item Number (Scilab gateway) — How to get the number of items in a list (list, mlist, tlist).

```
SciErr getListItemNumber(void* _pvCtx, int* _piAddress, int* _piNbItem)
```

Parameters

_pvCtx

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress

Address of the list.

_piNbItem

Return the number of items.

SciErr

Error structure where is stored errors messages history and first error number.

Description

This help describes how to get the number of items in a list (list, mlist, tlist).

Gateway Source

```
int get_list_info(int* _piAddress);
void insert_indent(void);

static int iLocalTab = 0;

int common_list(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr      = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_list_info(piAddr);

    LhsVar(1) = 0;
    return 0;
}

int get_list_info(int* _piAddress)
{
    SciErr sciErr;
    int i      = 0;
    int iRet    = 0;
```



```

int iItem    = 0;

//get list item number, failed if variable is not a kind of list
sciErr = getListItemNumber(pvApiCtx, _piAddress, &iItem);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    sciprint("This variable is not a list");
    return 0;
}

sciprint("List (%d items) -> address : 0x%08X) : \n", iItem, _piAddress);
for(i = 0 ; i < iItem ; i++)
{
    int iType          = 0;
    int* piAddrChild   = NULL;
    sciErr = getListItemAddress(pvApiCtx, _piAddress, i + 1, &piAddrChild);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = getVarType(pvApiCtx, piAddrChild, &iType);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(iType == sci_list || iType == sci_tlist || iType == sci_mlist)
    {
        insert_indent();
        sciprint("Child %d -> ", i + 1);
        iLocalTab++;
        iRet = get_list_info(piAddrChild);
        iLocalTab--;
        if(iRet)
        {
            return 1;
        }
    }
    else
    {
        insert_indent();
        sciprint("Child %d -> address : 0x%08X\n", i + 1, piAddrChild);
    }
}
return 0;
}

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iLocalTab ; i++)
    {
        sciprint("\t");
    }
}

```

```
}
```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l5 = list(l1,l2,l3);  
l4 = list(l5, list(l5,l5));  
l6 = uint16([1000,2000,3000]);  
l = list(l1,l2,l3,l6,l4,l5);  
common_list(l)
```

Name

Integer reading (Scilab gateway) — How to read matrix of integer in a list.

Input argument profile:

Signed integer :

```
SciErr getMatrixOfInteger8InList(void* _pvCtx, int* _piParent, int _iItemPos, i
```

```
SciErr getMatrixOfInteger16InList(void* _pvCtx, int* _piParent, int _iItemPos, i
```

```
SciErr getMatrixOfInteger32InList(void* _pvCtx, int* _piParent, int _iItemPos, i
```

Unsigned integer :

```
SciErr getMatrixOfUnsignedInteger8InList(void* _pvCtx, int* _piParent, int _iIt
```

```
SciErr getMatrixOfUnsignedInteger16InList(void* _pvCtx, int* _piParent, int _iI
```

```
SciErr getMatrixOfUnsignedInteger32InList(void* _pvCtx, int* _piParent, int _iI
```

Named variable profile:

Signed integer :

```
SciErr readMatrixOfIntger8InNamedList(void* _pvCtx, char* _pstName, int* _piPar
```

```
SciErr readMatrixOfIntger16InNamedList(void* _pvCtx, char* _pstName, int* _piPa
```

```
SciErr readMatrixOfIntger32InNamedList(void* _pvCtx, char* _pstName, int* _piPa
```

Unsigned integer :

```
SciErr readMatrixOfUnsignedInteger8InNamedList(void* _pvCtx, char* _pstName, in
```

```
SciErr readMatrixOfUnsignedInteger16InNamedList(void* _pvCtx, char* _pstName, i
```

```
SciErr readMatrixOfUnsignedInteger32InNamedList(void* _pvCtx, char* _pstName, i
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_pcData`, `_pucData`, `_psData`, `_pusData`, `_piData`, `_puiData`

Return address of data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of integer in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
```

```

int iType    = 0;

sciErr = getVarType(pvApiCtx, _piAddr, &iType);
switch(iType)
{
case sci_matrix :
    iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_poly :
    iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_boolean :
    iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_sparse :
    iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_boolean_sparse :
    iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_ints :
    iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_strings :
    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet    = 0;
    int iItem    = 0;

```

```

int* piChild    = NULL;

sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciprint("(%d)\n", iItem);
for(i = 0 ; i < iItem ; i++)
{
    sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    iTab++;
    iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
    iTab--;
}

return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))
            {
                sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
            else
            {
                sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
        }
    }
}

```

```

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen          = 0;
    int iRows         = 0;
    int iCols         = 0;

    char pstVar[16];
    int* piCoeff      = NULL;
    double** pdblReal = NULL;
    double** pdblImg  = NULL;

    sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(_iItemPos == 0)
    {
        //not in list
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
        pdblImg  = (double**)malloc(sizeof(double*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
            pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
        }
    }
}

```

```

if(isVarComplex(pvApiCtx, _piAddr))
{
    sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff,
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
}

if(isVarComplex(pvApiCtx, _piAddr))
{
    sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
}
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```



```

}

insert_indent();
sciprint("Poly  (%d x %d), varname : \'%s\'\\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    int* piBool    = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
    }
    else
    {
        sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
    }

    if(sciErr.iErr)
    {
        {
            printError(&sciErr, 0);
            return 0;
        }
    }

    insert_indent();
    sciprint("Boolean (%d x %d)\\n", iRows, iCols);
    return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)

```

```

{ //Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
    sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow);
  }
  else
  {
    sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow);
  }
}
else
{
  if(isVarComplex(pvApiCtx, _piAddr))
  {
    sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow);
  }
  else
  {
    sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow);
  }
}

insert_indent();
sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
  SciErr sciErr;
  int iRows      = 0;
  int iCols      = 0;
  int iItem      = 0;

  int* piNbRow   = NULL;
  int* piColPos  = NULL;

  if(_iItemPos == 0)
  { //Not in list
    sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
  }
  else
  {
    sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
  }

  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  insert_indent();
  sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
  return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)

```

```

{
    SciErr sciErr;
    int iPrec          = 0;
    int iRows          = 0;
    int iCols          = 0;

    char* pcData        = NULL;
    short* psData       = NULL;
    int* piData         = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        switch(iPrec)
        {
            case SCI_INT8 :
                sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
                break;
            case SCI_INT16 :
                sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
                break;
            case SCI_INT32 :
                sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
                break;
            case SCI_UINT8 :
                sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
                break;
            case SCI_UINT16 :
                sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
                break;
            case SCI_UINT32 :
                sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puiData);
                break;
            default :
                return 1;
        }
    }
    else
    {
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        switch(iPrec)
        {
            case SCI_INT8 :

```

```

    sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    break;
default :
    return 1;
}
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

piLen = (int*)malloc(sizeof(int) * iRows * iCols);
sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
}

sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
}
else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```

```

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Pointer : 0x%08X\n", pvPtr);

    return 0;
}

```

Scilab test script

```

function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;

```

Name

Integer writing (Scilab gateway) — How to add matrix of integer in a list.

Create from existing data.

Input argument profile:

Signed integer :

```
SciErr createMatrixOfInteger8InList(void* _pvCtx, int _iVar, int* _piParent, in
```

```
SciErr createMatrixOfInteger16InList(void* _pvCtx, int _iVar, int* _piParent, i
```

```
SciErr createMatrixOfInteger32InList(void* _pvCtx, int _iVar, int* _piParent, i
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8InList(void* _pvCtx, int _iVar, int* _piPa
```

```
SciErr createMatrixOfUnsignedInteger16InList(void* _pvCtx, int _iVar, int* _piP
```

```
SciErr createMatrixOfUnsignedInteger32InList(void* _pvCtx, int _iVar, int* _piP
```

Named variable profile:

Signed integer :

```
SciErr createMatrixOfInteger8InNamedList(void* _pvCtx, char* _pstName, int* _pi
```

```
SciErr createMatrixOfInteger16InNamedList(void* _pvCtx, char* _pstName, int* _p
```

```
SciErr createMatrixOfInteger32InNamedList(void* _pvCtx, char* _pstName, int* _p
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8InNamedList(void* _pvCtx, char* _pstName, .
```

```
SciErr createMatrixOfUnsignedInteger16InNamedList(void* _pvCtx, char* _pstName,
```

```
SciErr createMatrixOfUnsignedInteger32InNamedList(void* _pvCtx, char* _pstName,
```

Parameters

pvCtx

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

iVar

Position in the Scilab memory where you want to put the variable.

pstName

Name of the variable for "named" functions.

piParent

Address of the parent of the new item.

iItemPos

Position of the new item in the list.

iRows

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32`

Address of data array (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

Signed integer :

```
SciErr allocMatrixOfInteger8InList(void* _pvCtx, int _iVar, int* _piParent, int
```

```
SciErr allocMatrixOfInteger16InList(void* _pvCtx, int _iVar, int* _piParent, in
```

```
SciErr allocMatrixOfInteger32InList(void* _pvCtx, int _iVar, int* _piParent, in
```

Unsigned integer :

```
SciErr allocMatrixOfUnsignedInteger8InList(void* _pvCtx, int _iVar, int* _piPar
```

```
SciErr allocMatrixOfUnsignedInteger16InList(void* _pvCtx, int _iVar, int* _piPa
```

```
SciErr allocMatrixOfUnsignedInteger32InList(void* _pvCtx, int _iVar, int* _piPa
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32`

Return address of data array (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add matrix of integer in a list.

Two types of functions can be used to write in the memory of Scilab.

Gateway Source

```

int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
    double pdblPoly5[]    = {1,2,3,4,5};
    double pdblPoly6[]    = {-6,-5,-4,-3,-2,-1};
    double *pdblPoly[]    = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
    int piCoef[]           = {1,3,5,2,4,6};
    int piNbItemRow[]     = {1,2,1};
    int piColPos[]        = {8,4,7,2};
    double pdblSReal[]    = {1,2,3,4};
    double pdblSImg[]     = {4,3,2,1};
    int piBool[]          = {1,0,1,0,1,0,1,0,1};
    double* pdblDataPtr   = NULL;

    sciErr = createList(pvApiCtx, 1, 8, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psData);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

```

```

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)

```

```
{  
    printError(&sciErr, 0);  
    return 0;  
}  
  
LhsVar(1) = 1;  
return 0;  
}
```

Scilab test script

```
size_ref    = 8;  
type_ref    = ["constant", "string", "int16", "polynomial", "sparse", "boolean", "  
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);  
  
l = list_createlist();  
if size(l) <> size_ref then error("failed"), end  
for i = 1 : size_ref  
    if typeof(l(i)) <> type_ref(i) then error("failed"), end  
    if size(l(i)) <> dim_ref(i) then error("failed"), end  
end
```

Name

Pointer reading (Scilab gateway) — How to read pointer in a list.

Input argument profile:

```
SciErr getPointerInList(void* _pvCtx, int* _piParent, int _iItemPos, void** _pvPtr)
```

Named variable profile:

```
SciErr readPointerInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int _iItemPos)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_pvPtr`

Return pointer on data.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read pointer in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
```

```

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparsed_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
        case sci_matrix :
            iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_poly :
            iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean :
            iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_sparse :
            iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean_sparse :
            iRet = get_bsparsed_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_ints :
            iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_strings :
            iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
    }
}

```

```

case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem     = 0;
    int* piChild  = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        iTab++;
        iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
        iTab--;
    }

    return 0;;
}

```

```

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen      = 0;
    int iRows     = 0;
    int iCols     = 0;

    char pstVar[16];
    int* piCoeff  = NULL;
    double** pdblReal = NULL;
    double** pdblImg  = NULL;

```

```

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(_iItemPos == 0)
{
    //not in list
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

```



```

}

piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
}

if(isVarComplex(pvApiCtx, _piAddr))
{
    sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows = 0;
    int iCols = 0;

    int* piBool = NULL;

    if(_iItemPos == 0)
    {

```

```

    sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
}
else
{
    sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Boolean (%d x %d)\n", iRows, iCols);
return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows          = 0;
    int iCols          = 0;
    int iItem          = 0;

    int* piNbRow       = NULL;
    int* piColPos      = NULL;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }

    insert_indent();
    sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

```

```

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData    = NULL;
    short* psData    = NULL;
    int* piData      = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData = NULL;

    if(_iItemPos == 0)
    { //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        switch(iPrec)
        {
            case SCI_INT8 :
                sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);

```

```

        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
        break;
    default :
        return 1;
    }
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
        break;
    default :
        return 1;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```

```

}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }

        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
        if(sciErr.iErr)
        {

```

```

    printError(&sciErr, 0);
    return 0;
}

piLen = (int*)malloc(sizeof(int) * iRows * iCols);
sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));/*+ 1 for null terminator*/
}

sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

```

```
insert_indent();
sciprint("Pointer : 0x%08X\n", pvPtr);

return 0;
}
```

Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2, -1, 0, 0, 0; -1, 2, -1, 0, 0; 0, -1, 2, -1, 0; 0, 0, -1, 2, -1; 0, 0, 0, -1, 2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

Name

Pointer writing (Scilab gateway) — How to add pointer in a list.

Input argument profile:

```
SciErr createPointerInList(void* _pvCtx, int _iVar, int* _piParent, int _iItemP
```

Named variable profile:

```
SciErr createPointerInNamedList(void* _pvCtx, char* _pstName, int* _piParent, i
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_pvPtr`

Address of the pointer.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add pointer in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
    double pdblPoly5[]    = {1,2,3,4,5};
    double pdblPoly6[]    = {-6,-5,-4,-3,-2,-1};
    double *pdblPoly[]    = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
    int piCoef[]          = {1,3,5,2,4,6};
```



```

int piNbItemRow[]      = {1,2,1};
int piColPos[]         = {8,4,7,2};
double pdblSReal[]     = {1,2,3,4};
double pdblSImg[]      = {4,3,2,1};
int piBool[]           = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr    = NULL;

sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10,

```

```

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}

```

Scilab test script

```

size_ref      = 8;
type_ref      = ["constant", "string", "int16", "polynomial", "sparse", "boolean", "
dim_ref       = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref

```

```
    if typeof(l(i)) <> type_ref(i) then error("failed"), end  
    if size(l(i)) <> dim_ref(i) then error("failed"), end  
end
```

Name

Polynomial reading (Scilab gateway) — How to read matrix of polynomial in a list.

Input argument profile:

```
SciErr getMatrixOfPolyInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _piRows,
```

```
SciErr getComplexMatrixOfPolyInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _piCols,
```

Named variable profile:

```
SciErr readMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int* _piRows,
```

```
SciErr readComplexMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int* _piCols,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piNbCoef`

Return number of coefficient for each polynomial. (must be allocated)

`_pdblReal`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`, must be allocated)

`_pdblImg`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`, must be allocated)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of polynomial in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
```

```

int i = 0;
for(i = 0 ; i < iTab ; i++)
{
    sciprint("\t");
}
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
        case sci_matrix :
            iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_poly :
            iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean :
            iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);

```

```

        break;
    case sci_sparse :
        iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_boolean_sparse :
        iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_ints :
        iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_strings :
        iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_list :
        insert_indent();
        sciprint("List ");
        iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_tlist :
        insert_indent();
        sciprint("TList ");
        iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_mlist :
        insert_indent();
        sciprint("MList ");
        iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    case sci_pointer :
        iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
        break;
    default :
        insert_indent();
        sciprint("Unknown type\n");
        return 1;
    }
    return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem     = 0;
    int* piChild  = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);

```

```

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    iTab++;
    iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
    iTab--;
}

return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))
            {
                sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
            else
            {
                sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
            }
        }
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

```

```

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen          = 0;
    int iRows         = 0;
    int iCols         = 0;

    char pstVar[16];
    int* piCoeff      = NULL;
    double** pdblReal  = NULL;
    double** pdblImg   = NULL;

    sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(_iItemPos == 0)
    {
        //not in list
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piCoeff      = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pdblReal      = (double**)malloc(sizeof(double*) * iRows * iCols);
        pdblImg        = (double**)malloc(sizeof(double*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
            pdblImg[i]   = (double*)malloc(sizeof(double) * piCoeff[i]);
        }

        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
            if(sciErr.iErr)
            {
                printError(&sciErr, 0);
                return 0;
            }
        }
        else
        {
            sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
            if(sciErr.iErr)

```



```

    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
    else
    {
        sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);

```

```

return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    int* piBool     = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
    }
    else
    {
        sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean (%d x %d)\n", iRows, iCols);
    return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        /*Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
        }
    }
    else
    {
        {
            if(isVarComplex(pvApiCtx, _piAddr))

```

```

    {
        sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
    else
    {
        sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

insert_indent();
sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow   = NULL;
    int* piColPos  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData   = NULL;
    short* psData  = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData  = NULL;

```

```

if(_iItemPos == 0)
{
    //Not in list
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
        break;
    default :
        return 1;
    }
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
        break;
    default :
        return 1;
    }
}

```

```

        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piLen);
        break;
    default :
        return 1;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }
    }
}

```

```

    sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {

```

```
    sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
}
else
{
    sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Pointer : 0x%08X\n", pvPtr);

return 0;
}
```

Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2, -1, 0, 0, 0; -1, 2, -1, 0, 0; 0, -1, 2, -1, 0; 0, 0, -1, 2, -1; 0, 0, 0, -1, 2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

Name

Polynomial writing (Scilab gateway) — How to add matrix of polynomial in a list.

Input argument profile:

```
SciErr createMatrixOfPolyInList(void* _pvCtx, int _iVar, int* _piParent, int _i
```

```
SciErr createComplexMatrixOfPolyInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _piParent,
```

```
SciErr createComplexMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int*
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_pstVarName`

Variable name in polynomials (Scilab5: 4 characters max).

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_piNbCoef`

Number of coefficient for each polynomial.

`_pdblReal`

Address of array of double* with real part of coefficient (size: `_iCols * _iRows`)

`_pdblImg`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add matrix of polynomial in a list.

Gateway Source

```

int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1,3,5,2,4,6};
    double pdblData2[]    = {6,4,2,5,3,1};
    char *pstData[]       = {"may","be","the","with","puffin","you"};
    short psData[]        = {1,4,2,5,3,6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2,-1};
    double pdblPoly3[]    = {1,2,3};
    double pdblPoly4[]    = {-4,-3,-2,-1};
    double pdblPoly5[]    = {1,2,3,4,5};
    double pdblPoly6[]    = {-6,-5,-4,-3,-2,-1};
    double *pdblPoly[]    = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
    int piCoef[]          = {1,3,5,2,4,6};
    int piNbItemRow[]     = {1,2,1};
    int piColPos[]        = {8,4,7,2};
    double pdblSReal[]    = {1,2,3,4};
    double pdblSImg[]     = {4,3,2,1};
    int piBool[]          = {1,0,1,0,1,0,1,0,1};
    double* pdblDataPtr   = NULL;

    sciErr = createList(pvApiCtx, 1, 8, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psData);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

```

```
sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
pdblDataPtr[3]   = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
```

```
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}
```

Scilab test script

```
size_ref    = 8;
type_ref    = ["constant", "string", "int16", "polynomial", "sparse", "boolean", ""];
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

Name

Sparse reading (Scilab gateway) — How to read sparse in a list.

Input argument profile:

```
SciErr getSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _piRows,
```

```
SciErr getComplexSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _piCols,
```

Named variable profile:

```
SciErr readSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int* _piRows,
```

```
SciErr readComplexSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int* _piCols,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piNbItem`

Return number of non zero value.

`_piNbItemRow`

Return number of item in each rows (size: `_iRows`).

`_piColPos`

Return column position for each item (size: `_iNbItem`).

`_pdblReal`

Return array on real part data (size: `_iNbItem`).

`_pdblImg`

Return array on imaginary part data (size: `_iNbItem`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read sparse in a list.

Gateway Source

```

static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparsed_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *piAddr    = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)

```

```

{
case sci_matrix :
    iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_poly :
    iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_boolean :
    iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_sparse :
    iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_boolean_sparse :
    iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_ints :
    iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_strings :
    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem     = 0;
    int* piChild  = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)

```

```

{
    printError(&sciErr, 0);
    return 0;
}

sciprint("(%d)\n", iItem);
for(i = 0 ; i < iItem ; i++)
{
    sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    iTab++;
    iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
    iTab--;
}

return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    { //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
        else
        {
            sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
    }
}

```

```

    return 0;
}

insert_indent();
sciprint("Double (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen          = 0;
    int iRows         = 0;
    int iCols         = 0;

    char pstVar[16];
    int* piCoeff      = NULL;
    double** pdblReal = NULL;
    double** pdblImg  = NULL;

    sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(_iItemPos == 0)
    {
        //not in list
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
        pdblImg  = (double**)malloc(sizeof(double*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
            pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
        }

        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff,
            if(sciErr.iErr)

```



```

    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
else
{
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg      = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
    }
    else
    {
        sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

```

```

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    int* piBool     = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
    }
    else
    {
        sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piBool);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Boolean (%d x %d)\n", iRows, iCols);
    return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
    }
    else
    {
        sciErr = getMatrixOfSparseInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Sparse (%d x %d)\n", iRows, iCols);
    return 0;
}

```

```

    }
    else
    {
        sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow)
    }
}
else
{
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow)
        }
        else
        {
            sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow)
        }
    }

    insert_indent();
    sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow   = NULL;
    int* piColPos  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos)
    }
    else
    {
        {
            sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos)
        }

        if(sciErr.iErr)
        {
            {
                printError(&sciErr, 0);
                return 0;
            }
        }

        insert_indent();
        sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
        return 0;;
    }
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;

```

```

int iCols                = 0;

char* pcData             = NULL;
short* psData            = NULL;
int* piData              = NULL;
unsigned char* pucData   = NULL;
unsigned short* pusData  = NULL;
unsigned int* puiData    = NULL;

if(_iItemPos == 0)
{
    //Not in list
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puiData);
        break;
    default :
        return 1;
    }
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puiData);
        break;
    default :
        return 1;
    }
}

```

```

        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
        break;
    default :
        return 1;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
if(iPrec > 10)
{
    sciprint("Unsigned ");
}

sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen      = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {

```

```

    printError(&sciErr, 0);
    return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
}

sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
}
else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;

```

```

}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    sciprint("Pointer : 0x%08X\n", pvPtr);

    return 0;
}

```

Scilab test script

```

function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2, -1, 0, 0, 0; -1, 2, -1, 0, 0; 0, -1, 2, -1, 0; 0, 0, -1, 2, -1; 0, 0, 0, -1, 2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;

```

Name

Sparse writing (Scilab gateway) — How to add sparse matrix in a list.

Create from existing data.

Input argument profile:

```
SciErr createSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent, int _i
```

```
SciErr createComplexSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _piParent,
```

```
SciErr createComplexSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_iNbItem`

Number of non zero itmes in the sparse.

`_piNbItemRow`

Number of item in each rows (size: `_iRows`).

`_piColPos`

Column position for each item (size: `_iNbItem`).

`_pdblReal`

Address of real data array (size: `_iNbItem`).

`_pdblImg`

Address of imaginary data array (size: `_iNbItem`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add sparse matrix in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
  SciErr sciErr;
  int *piAddr          = NULL;
  int* piChild         = NULL;
  double pdblData1[]   = {1,3,5,2,4,6};
  double pdblData2[]   = {6,4,2,5,3,1};
  char *pstData[]      = {"may","be","the","with","puffin","you"};
  short psData[]       = {1,4,2,5,3,6};
  double pdblPoly1[]   = {1};
  double pdblPoly2[]   = {-2,-1};
  double pdblPoly3[]   = {1,2,3};
  double pdblPoly4[]   = {-4,-3,-2,-1};
  double pdblPoly5[]   = {1,2,3,4,5};
  double pdblPoly6[]   = {-6,-5,-4,-3,-2,-1};
  double *pdblPoly[]   = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly6};
  int piCoef[]         = {1,3,5,2,4,6};
  int piNbItemRow[]    = {1,2,1};
  int piColPos[]       = {8,4,7,2};
  double pdblSReal[]   = {1,2,3,4};
  double pdblSImg[]    = {4,3,2,1};
  int piBool[]         = {1,0,1,0,1,0,1,0,1};
  double* pdblDataPtr  = NULL;

  sciErr = createList(pvApiCtx, 1, 8, &piAddr);
  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2, pdblData1, pdblData2);
  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData);
  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psData);
  if(sciErr.iErr)
  {
```

```
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr      = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]   = 1;
pdblDataPtr[1]   = 2;
pdblDataPtr[2]   = 3;
```

```
pdblDataPtr[3] = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}
```

Scilab test script

```
size_ref    = 8;
type_ref    = ["constant", "string", "int16", "polynomial", "sparse", "boolean", ""];
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

Name

String reading (Scilab gateway) — How to read matrix of string in a list.

Input argument profile:

```
SciErr getMatrixOfStringInList(void* _pvCtx, int* _piParent, int _iItemPos, int
```

Named variable profile:

```
SciErr readMatrixOfStringInNamedList(void* _pvCtx, char* _pstName, int* _piParent,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piParent`

Address of the parent of the new item.

`_pstName`

Name of the variable for "named" functions.

`_iItemPos`

Position of the new item in the list.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piLength`

Address of array of strings length (must be allocated size: `_piRows * _piCols`)

`_pstStrings`

Address of array of `char*` (must be allocated size: `_piRows * _piCols`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of string in a list.

Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iTab ; i++)
    {
        sciprint("\t");
    }
}
```

```

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparsed_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int iItem      = 0;
    int iRet       = 0;
    int *_piAddr   = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_info(1, NULL, piAddr, 0);

    LhsVar(1) = 0;
    return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRet      = 0;
    int iType     = 0;

    sciErr = getVarType(pvApiCtx, _piAddr, &iType);
    switch(iType)
    {
        case sci_matrix :
            iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_poly :
            iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean :
            iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_sparse :
            iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_boolean_sparse :
            iRet = get_bsparsed_info(_iRhs, _piParent, _piAddr, _iItemPos);
            break;
        case sci_ints :

```

```

    iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_strings :
    iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_list :
    insert_indent();
    sciprint("List ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_tlist :
    insert_indent();
    sciprint("TList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_mlist :
    insert_indent();
    sciprint("MList ");
    iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
case sci_pointer :
    iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
    break;
default :
    insert_indent();
    sciprint("Unknown type\n");
    return 1;
}
return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRet      = 0;
    int iItem      = 0;
    int* piChild   = NULL;

    sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    sciprint("(%d)\n", iItem);
    for(i = 0 ; i < iItem ; i++)
    {
        sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        iTab++;
        iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
    }
}

```

```

    iTab--;
}

return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;

    double* pdblReal    = NULL;
    double* pdblImg     = NULL;

    if(_iItemPos == 0)
    {
        //not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pdblReal);
        }
    }

    if(sciErr.iErr)
    {
        {
            printError(&sciErr, 0);
            return 0;
        }
    }

    insert_indent();
    sciprint("Double (%d x %d)\n", iRows, iCols);

    return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iLen      = 0;
    int iRows     = 0;
    int iCols     = 0;

```

```

char pstVar[16];
int* piCoeff      = NULL;
double** pdblReal  = NULL;
double** pdblImg   = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(_iItemPos == 0)
{
    //not in list
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff      = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal      = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg        = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i]   = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
    else
    {
        sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblReal, pdblImg);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}
else
{

```



```

    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
    pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
        pdblImg[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
    }

    if(isVarComplex(pvApiCtx, _piAddr))
    {
        sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
    else
    {
        sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Poly (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
free(pdblReal);
free(pdblImg);
free(piCoeff);
return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows = 0;
    int iCols = 0;

```

```

int* piBool      = NULL;

if(_iItemPos == 0)
{
    sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
}
else
{
    sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Boolean (%d x %d)\n", iRows, iCols);
return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow    = NULL;
    int* piColPos   = NULL;

    double* pdblReal = NULL;
    double* pdblImg  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }
    else
    {
        if(isVarComplex(pvApiCtx, _piAddr))
        {
            sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos, &pdblReal, &pdblImg);
        }
        else
        {
            sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
        }
    }
}

```

```

    insert_indent();
    sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;

    int* piNbRow   = NULL;
    int* piColPos  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }
    else
    {
        sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &iItem, &piNbRow, &piColPos);
    }

    if(sciErr.iErr)
    {
        {
            printError(&sciErr, 0);
            return 0;
        }
    }

    insert_indent();
    sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
    return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int iPrec      = 0;
    int iRows      = 0;
    int iCols      = 0;

    char* pcData    = NULL;
    short* psData   = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData  = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
        if(sciErr.iErr)
        {
            {
                printError(&sciErr, 0);
                return 0;
            }
        }
    }

```

```

switch(iPrec)
{
case SCI_INT8 :
    sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
    break;
case SCI_INT16 :
    sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
    break;
case SCI_INT32 :
    sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
    break;
case SCI_UINT8 :
    sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pucData);
    break;
case SCI_UINT16 :
    sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pusData);
    break;
case SCI_UINT32 :
    sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &puData);
    break;
default :
    return 1;
}
}
else
{
    sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    switch(iPrec)
    {
    case SCI_INT8 :
        sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pcData);
        break;
    case SCI_INT16 :
        sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &psData);
        break;
    case SCI_INT32 :
        sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &piData);
        break;
    case SCI_UINT8 :
        sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pucData);
        break;
    case SCI_UINT16 :
        sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &pusData);
        break;
    case SCI_UINT32 :
        sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols, &puData);
        break;
    default :
        return 1;
    }
}
}

```

```

    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    insert_indent();
    if(iPrec > 10)
    {
        sciprint("Unsigned ");
    }

    sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

    return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    int i;
    int iRows      = 0;
    int iCols      = 0;
    int* piLen     = NULL;

    char **pstData = NULL;

    if(_iItemPos == 0)
    {
        //Not in list
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        piLen = (int*)malloc(sizeof(int) * iRows * iCols);
        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
        for(i = 0 ; i < iRows * iCols ; i++)
        {
            pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
        }

        sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
    }
}

```

```

else
{
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    piLen = (int*)malloc(sizeof(int) * iRows * iCols);
    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1)); //+ 1 for null term
    }

    sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
    SciErr sciErr;
    void* pvPtr = NULL;

    if(_iItemPos == 0)
    {
        sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
    }
    else
    {
        sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
    }

    if(sciErr.iErr)

```

```
{
    printError(&sciErr, 0);
    return 0;
}

insert_indent();
sciprint("Pointer : 0x%08X\n", pvPtr);

return 0;
}
```

Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with", "you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

Name

String writing (Scilab gateway) — How to add matrix of string in a list.

Input argument profile:

```
SciErr createMatrixOfStringInList(void* _pvCtx, int _iVar, int* _piParent, int _iItemPos, int _iRows, int _iCols, char* _pstStrings, SciErr _sciErr)
```

Named variable profile:

```
SciErr createMatrixOfStringInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int _iItemPos, int _iRows, int _iCols, char* _pstStrings, SciErr _sciErr)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piParent`

Address of the parent of the new item.

`_iItemPos`

Position of the new item in the list.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_pstStrings`

Address of array of char* (size: `_iCols * _iRows`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to add matrix of string in a list.

Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr          = NULL;
    int* piChild          = NULL;
    double pdblData1[]    = {1, 3, 5, 2, 4, 6};
    double pdblData2[]    = {6, 4, 2, 5, 3, 1};
    char *pstData[]       = {"may", "be", "the", "with", "puffin", "you"};
    short psData[]        = {1, 4, 2, 5, 3, 6};
    double pdblPoly1[]    = {1};
    double pdblPoly2[]    = {-2, -1};
    double pdblPoly3[]    = {1, 2, 3};
```



```

double pdblPoly4[]      = {-4, -3, -2, -1};
double pdblPoly5[]      = {1, 2, 3, 4, 5};
double pdblPoly6[]      = {-6, -5, -4, -3, -2, -1};
double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPoly4};
int piCoef[]            = {1, 3, 5, 2, 4, 6};
int piNbItemRow[]       = {1, 2, 1};
int piColPos[]          = {8, 4, 7, 2};
double pdblSReal[]      = {1, 2, 3, 4};
double pdblSImg[]       = {4, 3, 2, 1};
int piBool[]            = {1, 0, 1, 0, 1, 0, 1, 0, 1};
double* pdblDataPtr     = NULL;

sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2, piCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pdblPoly);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, piNbItemRow);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, pdblSReal, pdblSImg);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBool);
if(sciErr.iErr)
{

```

```

    printError(&sciErr, 0);
    return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblDataPtr = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0] = 1;
pdblDataPtr[1] = 2;
pdblDataPtr[2] = 3;
pdblDataPtr[3] = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = 1;
return 0;
}

```

Scilab test script

```

size_ref = 8;
type_ref = ["constant", "string", "int16", "polynomial", "sparse", "boolean", ""]

```

```
dim_ref      = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);  
l = list_createlist();  
if size(l) <> size_ref then error("failed"), end  
for i = 1 : size_ref  
    if typeof(l(i)) <> type_ref(i) then error("failed"), end  
    if size(l(i)) <> dim_ref(i) then error("failed"), end  
end
```

Name

api_scilab — api_scilab is the Scilab interface to read/write data from/to Scilab memory

Description

In the previous versions of Scilab, there was no clear or easy way to extend Scilab or to use it as an embedded application.

Since Scilab 5.2.0, these issues have been tackled by API Scilab. Its provides a new consistent, documented and easy way API.

This API provides functions to read/write data from/to Scilab memory.

It provides many advantages:

- Management of all Scilab data types
- Consistency over all data types
- Error management
- Fully documented
- Fully tested by unitary tests
- Straight and named access (See the call_scilab API) to variables.
- And it will be maintained over future versions of Scilab

Usually, this API is used to extend Scilab capabilities but can be used in other contexts

- **Libraires or C/C++ functions:** Many librairies have been developped on the free/Open source or proprietary markets. This library can be loaded and used in a high level language like Scilab. API Scilab provides the capabilities to interact with such libraries.
- **Scilab can be used as computing engine** from a third party software. This feature is called call_scilab when Scilab is used from C/C++ code or javasci when used from Java. Access to variable is done through their names (named variable).

Note that old APIs (stackX.h) will not be available after Scilab 6.0 (included).

See Also

Compile and run with call_scilab, Matrix Management, Boolean Management, Complex Management, String Management, Call_Scilab

Authors

Sylvestre Ledru

Name

Boolean reading (Scilab gateway) — How to read matrix of boolean.

Input argument profile:

```
SciErr getMatrixOfBoolean(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols, int* _piBool)
```

Named variable profile:

```
SciErr readNamedMatrixOfBoolean(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols, int* _piBool)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piBool`

Return address of data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read matrix of boolean.

Gateway Source

```
int read_write_boolean(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    //first variable info : real matrix of double
    int iRows    = 0;
    int iCols    = 0;
    int *piAddr  = NULL;
    int* piBool  = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);
}
```

```
//get variable address of the first input argument
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//get size and data from Scilab memory
sciErr = getMatrixOfBoolean(pvApiCtx, piAddr, &iRows, &iCols, &piBool);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//Do something with data
for(i = 0 ; i < iRows * iCols ; i++)
{
    piBool[i] = piBool[i] == 0 ? 1 : 0;
}

sciErr = createMatrixOfBoolean(pvApiCtx, Rhs + 1, iRows, iCols, piBool);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = Rhs + 1;
return 0;
}
```

Scilab test script

```
a = [%t, %f, %t ; %f, %t, %f ; %t, %f, %t];
a_ref = [%f, %t, %f ; %t, %f, %t ; %f, %t, %f];
b = read_write_boolean(a);

if or(b <> a_ref) then error("failed"), end
```

Name

Boolean writing (Scilab gateway) — How to write matrices of boolean.

Input argument profile:

```
SciErr createMatrixOfBoolean(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _piBool)
```

Named variable profile:

```
SciErr createNamedMatrixOfBoolean(void* _pvCtx, char* _pstName, int _iRows, int _iCols, int _piBool)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows of the variable.

`_piCols`

Return number of columns of the variable.

`_piBool`

Return address of data array (size: `_iRows * _iCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfBoolean(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _piBool)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_iRows`

Number of rows of the new variable.

`_iCols`

Numbers of columns of the new variable.

`_piBool`

Returns address of real data array (size: `_iCols * _iRows`).

SciErr

Error structure where is stored errors messages history and first error number.

Description

This help describes how to write matrix of boolean.

Gateway Source

```
int read_write_boolean(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    //first variable info : real matrix of double
    int iRows    = 0;
    int iCols    = 0;
    int *piAddr   = NULL;
    int* piBool   = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);

    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getMatrixOfBoolean(pvApiCtx, piAddr, &iRows, &iCols, &piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //Do something with data
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        piBool[i] = piBool[i] == 0 ? 1 : 0;
    }

    sciErr = createMatrixOfBoolean(pvApiCtx, Rhs + 1, iRows, iCols, piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
}
```



```
}
```

Scilab test script

```
a = [%t, %f, %t ; %f, %t, %f ; %t, %f, %t];  
a_ref = [%f, %t, %f ; %t, %f, %t ; %f, %t, %f];  
b = read_write_boolean(a);  
  
if or(b <> a_ref) then error("failed"), end
```

Name

Boolean sparse reading (Scilab gateway) — How to read boolean sparse in a gateway.

Input argument profile:

```
SciErr getBooleanSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int*
```

Named variable profile:

```
SciErr readNamedBooleanSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows,
```

Parameters

_pvCtx

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress

Address of the Scilab variable.

_pstName

Name of the variable for "named" functions.

_piRows

Return number of rows of the variable.

_piCols

Return number of columns of the variable.

_piNbItem

Return number of non zero value.

_piNbItemRow

Return number of item in each rows (size: _iRows).

_piColPos

Return column position for each item (size: _iNbItem).

SciErr

Error structure where is stored errors messages history and first error number.

Description

This help describes how to read boolean sparse.

Gateway Source

```
int read_write_bsparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i                = 0;
    int j                = 0;
    int k                = 0;

    //first variable info : real matrix of double
    int iRows            = 0;
    int iCols            = 0;
    int *piAddr          = NULL;
    int iNbItem          = 0;
```

```

int* piNbItemRow      = NULL;
int* piColPos         = NULL;

int iCol              = 0;
int iNewCol           = 0;

int iNewItem          = 0;
int* piNewRow         = NULL;
int* piNewCol         = NULL;

//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

//get variable address of the first input argument
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//get size and data from Scilab memory
sciErr = getBooleanSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem,
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//Do something with data
//convert %T -> %F and %F -> %T

iNewItem = (iRows * iCols) - iNbItem;
piNewRow = (int*)MALLOC(sizeof(int) * iRows);
piNewCol = (int*)MALLOC(sizeof(int) * iNewItem);

for(i = 0 ; i < iRows ; i++)
{
    piNewRow[i] = iCols - piNbItemRow[i];
    for(j = 0 ; j < iCols ; j++)
    {
        int iFind = 0;
        for(k = 0 ; k < piNbItemRow[i] ; k++)
        {
            if(piColPos[iCol + k] == (j + 1))
            {
                iFind = 1;
                break;
            }
        }
        if(iFind == 0)
        {
            piNewCol[iNewCol++] = (j + 1);
        }
    }
}

```

```
        iCol += piNbItemRow[i];
    }

    sciErr = createBooleanSparseMatrix(pvApiCtx, Rhs + 1, iRows, iCols, iNewIter);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
}
```

Scilab test script

```
a = sparse([%t, %f, %t ; %f, %t, %f ; %t, %f, %t]);
a_ref = sparse([%f, %t, %f ; %t, %f, %t ; %f, %t, %f]);

b = read_write_bsparsed(a);
if or(b <> a_ref) then error("failed");end
```

Name

Boolean sparse writing (Scilab gateway) — How to add boolean sparse matrix in a gateway.

Input argument profile:

```
SciErr createBooleanSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

Named variable profile:

```
SciErr createNamedBooleanSparseMatrix(void* _pvCtx, char* _pstName, int _iRows,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_iNbItem`

Number of non zero itmes in the sparse.

`_piNbItemRow`

Number of item in each rows (size: `_iRows`).

`_piColPos`

Column position for each item (size: `_iNbItem`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocBooleanSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_iNbItem`
Number of non zero itmes in the sparse.

`_piNbItemRow`
Number of item in each rows (size: `_iRows`).

`_piColPos`
Column position for each item (size: `_iNbItem`).

`SciErr`
Error structure where is stored errors messages history and first error number.

Description

This help describes how to add boolean sparse matrix in a list.

Gateway Source

```
int read_write_bsparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i                = 0;
    int j                = 0;
    int k                = 0;

    //first variable info : real matrix of double
    int iRows            = 0;
    int iCols            = 0;
    int *piAddr          = NULL;
    int iNbItem          = 0;
    int* piNbItemRow     = NULL;
    int* piColPos        = NULL;

    int iCol              = 0;
    int iNewCol          = 0;

    int iNewItem          = 0;
    int* piNewRow        = NULL;
    int* piNewCol        = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);

    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getBooleanSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem,
    if(sciErr.iErr)
```

```

{
    printError(&sciErr, 0);
    return 0;
}

//Do something with data
//convert %T -> %F and %F -> %T

iNewItem = (iRows * iCols) - iNbItem;
piNewRow = (int*)MALLOC(sizeof(int) * iRows);
piNewCol = (int*)MALLOC(sizeof(int) * iNewItem);

for(i = 0 ; i < iRows ; i++)
{
    piNewRow[i] = iCols - piNbItemRow[i];
    for(j = 0 ; j < iCols ; j++)
    {
        int iFind = 0;
        for(k = 0 ; k < piNbItemRow[i] ; k++)
        {
            if(piColPos[iCol + k] == (j + 1))
            {
                iFind = 1;
                break;
            }
        }
        if(iFind == 0)
        {
            piNewCol[iNewCol++] = (j + 1);
        }
    }
    iCol += piNbItemRow[i];
}

sciErr = createBooleanSparseMatrix(pvApiCtx, Rhs + 1, iRows, iCols, iNewItem);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = Rhs + 1;
return 0;
}

```

Scilab test script

```

a = sparse([%t, %f, %t ; %f, %t, %f ; %t, %f, %t]);
a_ref = sparse([%f, %t, %f ; %t, %f, %t ; %f, %t, %f]);

b = read_write_bsparsed(a);
if or(b <> a_ref) then error("failed");end

```

Name

Variable Reference (Scilab gateway) — How to get the address of an argument or a variable in a gateway.

Input argument profile:

```
SciErr getVarAddressFromPosition(void* _pvCtx, int _iVar, int** _piAddress)
```

Named variable profile:

```
SciErr getVarAddressFromName(void* _pvCtx, char* _pstName, int** _piAddress)
```

Parameters

_pvCtx

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_iVar

Position of the argument in the function call.

_pstName

Scilab variable name.

_piAddress

Return address of the Scilab variable.

SciErr

Error structure where is stored errors messages history and first error number.

Description

This function retrieves the address of an argument in a gateway.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
}
```



```

    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
            {
                char pstSigned[]    = "signed";
                char pstUnsigned[]  = "unsigned";
            }
    }
}

```

```

        char* pstSign      = pstSigned;

        int iPrec          = 0;
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
    case sci_strings :
        sciprint("strings\n");
        break;
    case sci_list :
        sciprint("list\n");
        break;
    case sci_tlist :
        sciprint("tlist\n");
        break;
    case sci_mlist :
        sciprint("mlist\n");
        break;
    default :
        sciprint("Not manage by this function\n");
        return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {
        return sciErr;
    }
    sciprint("%d", iItem);
}

```

```
    }  
    return sciErr;  
}
```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l4 = uint16([1000,2000,3000]);  
l5 = list(l1,l2,l3);  
l = list(l1,l2,l3,l4,l5);  
common_function(l(1:$))
```

Name

Variable dimension (Scilab gateway) — How to get the dimensions of an argument or a variable stored as matrix.

Input argument profile:

```
SciErr getVarDimension(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

```
SciErr getNamedVarDimension(void* _pvCtx, char *_pstName, int* _piRows, int* _piCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

The address of the variable.

`_pstName`

Scilab variable name.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to get the dimensions of a variable in a gateway.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }
}
```

```

//1 for true, 0 for false
iBool = sciErr.iErr == 0 ? 1 : 0;
sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
//assign allocated variables to Lhs position
LhsVar(1) = 1;

return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :

```

```

    {
        char pstSigned[]    = "signed";
        char pstUnsigned[]  = "unsigned";
        char* pstSign       = pstSigned;

        int iPrec           = 0;
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
case sci_strings :
    sciprint("strings\n");
    break;
case sci_list :
    sciprint("list\n");
    break;
case sci_tlist :
    sciprint("tlist\n");
    break;
case sci_mlist :
    sciprint("mlist\n");
    break;
default :
    sciprint("Not manage by this function\n");
    return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {

```

```
        return sciErr;  
    }  
    sciprint("%d", iItem);  
}  
return sciErr;  
}
```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l4 = uint16([1000,2000,3000]);  
l5 = list(l1,l2,l3);  
l = list(l1,l2,l3,l4,l5);  
common_function(l(1:$))
```

Name

Variable Type (Scilab gateway) — How to get the type of an argument or a variable within a gateway.

Input argument profile:

```
SciErr getVarType(void* _pvCtx, int* _piAddress, int* _piType)
```

Named variable profile:

```
SciErr getNamedVarType(void* _pvCtx, char* _pstName, int* _piType)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

The address of the variable

`_pstName`

Scilab variable name.

`_piType`

Scilab type of the variable (sci_matrix, sci_strings, sci_ints, ...).

Note that the list of the different variable types is available as an enum in stack-c.h.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to retrieve the Scilab type of a variable in a gateway.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
```



```

        iBool = sciErr.iErr == 0 ? 1 : 0;
        sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }
        //assign allocated variables to Lhs position
        LhsVar(1) = 1;

        return 0;
    }

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[] = "signed";

```

```

        char pstUnsigned[] = "unsigned";
        char* pstSign      = pstSigned;

        int iPrec          = 0;
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
case sci_strings :
    sciprint("strings\n");
    break;
case sci_list :
    sciprint("list\n");
    break;
case sci_tlist :
    sciprint("tlist\n");
    break;
case sci_mlist :
    sciprint("mlist\n");
    break;
default :
    sciprint("Not manage by this function\n");
    return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {
        return sciErr;
    }
}

```

```
        sciprint("%d", iItem);  
    }  
    return sciErr;  
}
```

Scilab test script

```
l1 = [1, 2*%i, 3;%i, 2, 3*%i];  
l2 = ["may", "the"; "puffin", "be"; "with", "you"];  
l3 = int8([1, 2, 3]);  
l4 = uint16([1000, 2000, 3000]);  
l5 = list(l1, l2, l3);  
l = list(l1, l2, l3, l4, l5);  
common_function(l(1:$))
```

Name

Variable Complexity (Scilab gateway) — How to get the argument or variable complexity.

Input argument profile:

```
int isVarComplex(void* _pvCtx, int* _piAddress)
```

Named variable profile:

```
int isNamedVarComplex(void* _pvCtx, char *_pstName)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the variable

`_pstName`

Scilab variable name.

Returned value

0 for real variables and 1 for complex variables.

Description

This help describes how to retrieve the variable complexity.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
```

```
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
            {
                char pstSigned[]    = "signed";
                char pstUnsigned[]  = "unsigned";
                char* pstSign       = pstSigned;

                int iPrec           = 0;
            }
            break;
    }
}
```

```
        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
    case sci_strings :
        sciprint("strings\n");
        break;
    case sci_list :
        sciprint("list\n");
        break;
    case sci_tlist :
        sciprint("tlist\n");
        break;
    case sci_mlist :
        sciprint("mlist\n");
        break;
    default :
        sciprint("Not manage by this function\n");
        return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {
        return sciErr;
    }
    sciprint("%d", iItem);
}
return sciErr;
}
```

Scilab test script

```
l1 = [1, 2*%i, 3;%i, 2, 3*%i];  
l2 = ["may", "the"; "puffin", "be"; "with", "you"];  
l3 = int8([1, 2, 3]);  
l4 = uint16([1000, 2000, 3000]);  
l5 = list(l1, l2, l3);  
l = list(l1, l2, l3, l4, l5);  
common_function(l(1:$))
```

Name

Matrix Type (Scilab gateway) — How to know if an argument or a variable is stored as a matrix.

Input argument profile:

```
int isVarMatrixType(void* _pvCtx, int* _piAddress)
```

Named variable profile:

```
int isNamedVarMatrixType(void* _pvCtx, char *_pstName)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the variable

Returned value

1 if the variable is stored as matrix otherwise 0.

Description

This help describes how to know if a variable is stored as a matrix.

In some cases (exemple: list), a variable is not stored as a standard Scilab matrix. This function provides a way to handle both cases.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
```



```

        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
            {
                char pstSigned[]    = "signed";
                char pstUnsigned[]  = "unsigned";
                char* pstSign      = pstSigned;

                int iPrec          = 0;

```

```

        sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
case sci_strings :
    sciprint("strings\n");
    break;
case sci_list :
    sciprint("list\n");
    break;
case sci_tlist :
    sciprint("tlist\n");
    break;
case sci_mlist :
    sciprint("mlist\n");
    break;
default :
    sciprint("Not manage by this function\n");
    return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {
        return sciErr;
    }
    sciprint("%d", iItem);
}
return sciErr;
}

```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l4 = uint16([1000,2000,3000]);  
l5 = list(l1,l2,l3);  
l = list(l1,l2,l3,l4,l5);  
common_function(l(1:$))
```

Name

Double reading (Scilab gateway) — How to read matrices of double in a gateway.

Input argument profile:

```
SciErr getMatrixOfDouble(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getComplexMatrixOfDouble(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

```
SciErr readNamedMatrixOfDouble(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedComplexMatrixOfDouble(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`_pdblReal`

Return address of real data array (size: `_iCols * _iRows`).

`_pdblImg`

Return address of imaginary data array (size: `_iCols * _iRows`).

This argument does not exist with `getMatrixOfDouble` and `readNamedMatrixOfDouble`.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of doubles can be handled through the Scilab API.

Gateway Source

```
int read_double(char *fname,unsigned long fname_len)
{
    int i;
    //first variable info : real matrix of double
    int iType    = 0;
    int iRows    = 0;
    int iCols    = 0;
    int iComplex = 0;
    int *piAddr  = NULL;
    double* pdblReal = NULL;
```

```

double* pdblImg = NULL;

SciErr sciErr;
//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

/*****
*   First variable   *
*****/

//get variable address of the first input argument
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//check type
sciErr = getVarType(pvApiCtx, piAddr, &iType);
if(sciErr.iErr || iType != sci_matrix)
{
    printError(&sciErr, 0);
    return 0;
}

//get complexity
iComplex = isVarComplex(pvApiCtx, piAddr);

//check complexity
if(iComplex)
{
    //get size and data from Scilab memory
    sciErr = getComplexMatrixOfDouble(pvApiCtx, piAddr, &iRows, &iCols, &pdblReal
}
else
{
    //get size and data from Scilab memory
    sciErr = getMatrixOfDouble(pvApiCtx, piAddr, &iRows, &iCols, &pdblReal);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//Do something with data
//if variable is complex, switch real part and imaginary part otherwise multip

if(iComplex)
{
    sciErr = createComplexMatrixOfDouble(pvApiCtx, Rhs + 1, iRows, iCols, pdblImg
}
else
{

```

```

for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = pdblReal[i] * -1;
}
sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, iRows, iCols, pdblReal);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = Rhs + 1;
return 0;
}

```

Scilab test script

```

a = [ 0 1 2 3; ..
      4 5 6 7; ..
      8 9 10 11];
b = [ 23*%i,      1+22*%i,      2+21*%i,      3+20*%i,      4+19*%i,      5+18*%i; ..
      6+17*%i,      7+16*%i,      8+15*%i,      9+14*%i,      10+13*%i,      11+12*%i; ..
      12+11*%i,      13+10*%i,      14+9*%i,      15+8*%i,      16+7*%i,      17+6*%i; ..
      18+5*%i,      19+4*%i,      20+3*%i,      21+2*%i,      22+1*%i,      23];

a2 = read_double(a);
b2 = read_double(b);

if or(a2 <> a * -1) then error("failed"), end
if or(b2 <> (imag(b) + real(b) * %i)) then error("failed"), end

```

Name

Double writing (Scilab gateway) — How to write matrices of doubles in a gateway.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

```
SciErr createComplexMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

Named variable profile:

```
SciErr createNamedMatrixOfDouble(void* _pvCtx, char* _pstName, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

```
SciErr createNamedComplexMatrixOfDouble(void* _pvCtx, char* _pstName, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_iRows`

Number of rows of the new variable.

`_iCols`

Numbers of columns of the new variable.

`_pdblReal`

Address of real data array (size: `_iCols * _iRows`).

`_pdblImg`

Address of imaginary data array (size: `_iCols * _iRows`). This argument does not exist with `createMatrixOfDouble` and `createNamedMatrixOfDouble`.

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

```
SciErr allocComplexMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, double* _pdblReal, double* _pdblImg)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable.

_iRows
Number of rows of the new variable.

_iCols
Numbers of columns of the new variable.

_pdblReal
Returns address of real data array (size: **_iCols** * **_iRows**).

_pdblImg
Returns address of imaginary data array (size: **_iCols** * **_iRows**). This argument does not exist with `allocMatrixOfDouble`.

SciErr
Error structure where is stored errors messages history and first error number.

Gateway Source

```
int write_double(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i,j;
    //first variable info : real matrix of double 3 x 4
    int iRows1    = 3;
    int iCols1    = 4;
    double* pdblReal1 = NULL;

    //second variable info : complex matrix of double 4 x 6
    int iRows2    = 4;
    int iCols2    = 6;
    double* pdblReal2 = NULL;
    double* pdblImg2 = NULL;

    /*****
    *    First variable    *
    *****/
    //alloc array of data in OS memory
    pdblReal1 = (double*)malloc(sizeof(double) * iRows1 * iCols1);

    //fill array with incremental values
    //[ 0  1  2  3
    // 4  5  6  7
    // 8  9 10 11]
    for(i = 0 ; i < iRows1 ; i++)
    {
        for(j = 0 ; j < iCols1 ; j++)
        {
            pdblReal1[i + iRows1 * j] = i * iCols1 + j;
        }
    }
    //can be written in a single loop
    //for(i = 0 ; i < iRows1 * iCols1; i++)
    //{
    //    pdblReal1[i] = i;
    //}
```



```

//create a variable from a existing data array
sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, iRows1, iCols1, pdblReal1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//after creation, we can free memory.
free(pdblReal1);

/*****
*    Second variable    *
*****/
//reserve space in scilab memory and fill it
sciErr = allocComplexMatrixOfDouble(pvApiCtx, Rhs + 2, iRows2, iCols2, &pdblReal2, &pdblImg2);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//fill array with incremental values for real part and decremental for imaginary part
//[ 23i      1+22i      2+21i      3+20i      4+19i      5+18i
// 6+17i      7+16i      8+15i      9+14i      10+13i      11+12i
// 12+11i     13+10i      14+9i      15+8i      16+7i      17+6i
// 18+5i      19+4i      20+3i      21+2i      22+1i      23 ]
for(i = 0 ; i < iRows2 ; i++)
{
    for(j = 0 ; j < iCols2 ; j++)
    {
        pdblReal2[i + iRows2 * j] = i * iCols2 + j;
        pdblImg2 [i + iRows2 * j] = (iRows2 * iCols2 - 1) - (i * iCols2 + j);
    }
}
//can be written in a single loop
//for(i = 0 ; i < iRows2 * iCols2; i++)
//{
//    pdblReal2[i] = i;
//    pdblImg2 [i] = (iRows2 * iCols2 - 1) - i;
//}

// !\ DO NOT FREE MEMORY, in this case, it's the Scilab memory

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
LhsVar(2) = Rhs + 2;
return 0;
}

```

Scilab test script

```

a_ref = [ 0 1 2 3; ..
          4 5 6 7; ..
          8 9 10 11];

```

```
b_ref = [ 23*i, 1+22*i, 2+21*i, 3+20*i, 4+19*i, 5+18*i,
          6+17*i, 7+16*i, 8+15*i, 9+14*i, 10+13*i, 11+12*i,
          12+11*i, 13+10*i, 14+9*i, 15+8*i, 16+7*i, 17+6*i,
          18+5*i, 19+4*i, 20+3*i, 21+2*i, 22+1*i, 23];
[a,b] = write_double();
if or(a <> a_ref) then error("failed");end
if or(b <> b_ref) then error("failed");end
```

Name

Integer Precision (Scilab gateway) — How to get precision of an integer matrix.

```
SciErr getMatrixOfIntegerPrecision(void* _pvCtx, int* _piAddress, int* _piPreci
```

```
SciErr getNamedMatrixOfIntegerPrecision(void* _pvCtx, char* _pstName, int* _piP
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the variable.

`_pstName`

Name of the variable for "named" functions.

`_piPrecision`

Return precision of an integer variable. (SCI_INT8, SCI_UINT8, SCI_INT16, ...)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to get precision of an integer matrix.

Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
    }
}
```

```

        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr    = NULL;
    int iType      = 0;
    int iRows      = 0;
    int iCols      = 0;
    int iItem      = 0;
    int iComplex   = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
            {
                char pstSigned[]    = "signed";
                char pstUnsigned[]  = "unsigned";
                char* pstSign      = pstSigned;

                int iPrec          = 0;
                sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            }
            break;
    }
}

```

```

        if(sciErr.iErr)
        {
            return sciErr;
        }

        if(iPrec > 10)
        {
            pstSign = pstUnsigned;
        }

        sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
    }
    break;
case sci_strings :
    sciprint("strings\n");
    break;
case sci_list :
    sciprint("list\n");
    break;
case sci_tlist :
    sciprint("tlist\n");
    break;
case sci_mlist :
    sciprint("mlist\n");
    break;
default :
    sciprint("Not manage by this function\n");
    return sciErr;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciprint("\tComplex: Yes\n");
}

sciprint("\tDimensions: ");
if(isVarMatrixType(pvApiCtx, piAddr))
{
    sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("%d x %d", iRows, iCols);
}
else
{
    sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
    if(sciErr.iErr)
    {
        return sciErr;
    }
    sciprint("%d", iItem);
}
return sciErr;
}

```

Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];  
l2 = ["may","the";"puffin","be";"with","you"];  
l3 = int8([1,2,3]);  
l4 = uint16([1000,2000,3000]);  
l5 = list(l1,l2,l3);  
l = list(l1,l2,l3,l4,l5);  
common_function(l(1:$))
```

Name

Integer reading (Scilab gateway) — How to read matrices of integer in a gateway.

Input argument profile:

Signed integer :

```
SciErr getMatrixOfInteger8(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getMatrixOfInteger16(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getMatrixOfInteger32(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Unsigned integer :

```
SciErr getMatrixOfUnsignedInteger8(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getMatrixOfUnsignedInteger16(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getMatrixOfUnsignedInteger32(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

Signed integer :

```
SciErr readNamedMatrixOfInteger8(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedMatrixOfInteger16(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedMatrixOfInteger32(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

Unsigned integer :

```
SciErr readNamedMatrixOfUnsignedInteger8(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedMatrixOfUnsignedInteger16(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedMatrixOfUnsignedInteger32(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`_pcData8, _pucData8, _psData16, _pusData16, _piData32, _puiData32`

Returns address of array (size: `_piRows * _piCols`).

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrices of integer can be handled through the Scilab API.

Gateway Source

```
void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pVD)

int read_integer(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    //output variable info
    int iRows8      = 0;
    int iCols8      = 0;
    int iRows16     = 0;
    int iCols16     = 0;
    int iRows32     = 0;
    int iCols32     = 0;
    int iRowsu8     = 0;
    int iColsu8     = 0;
    int iRowsu16    = 0;
    int iColsu16    = 0;
    int iRowsu32    = 0;
    int iColsu32    = 0;
    int iPrec       = 0;

    int* piAddr8    = NULL;
    int* piAddr16   = NULL;
    int* piAddr32   = NULL;
    int* piAddru8   = NULL;
    int* piAddru16  = NULL;
    int* piAddru32  = NULL;

    char* pcData    = NULL;
    short* psData   = NULL;
    int* piData     = NULL;
    unsigned char* pucData = NULL;
    unsigned short* pusData = NULL;
    unsigned int* puiData = NULL;

    char* pcDataOut  = NULL;
    short* psDataOut = NULL;
    int* piDataOut   = NULL;
    unsigned char* pucDataOut = NULL;
    unsigned short* pusDataOut = NULL;
    unsigned int* puiDataOut = NULL;

    //check input/ouput arguments count
    CheckRhs(6,6);
    CheckLhs(6,6);

    //get varialbe address
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr8);
    if(sciErr.iErr)
    {
```



```
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddr8);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 3, &piAddr16);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 4, &piAddr16);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 5, &piAddr32);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 6, &piAddr32);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr8, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT8)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr8, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT8)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr16, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT16)
```

```
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr16, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT16)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr32, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT32)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr32, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT32)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger8(pvApiCtx, piAddr8, &iRows8, &iCols8, &pcData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, piAddr8, &iRowsu8, &iColsu8, &pcData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger16(pvApiCtx, piAddr16, &iRows16, &iCols16, &psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, piAddr16, &iRowsu16, &iColsu16, &psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
```

```
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger32(pvApiCtx, piAddr32, &iRows32, &iCols32, &piData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, piAddr32, &iRowsu32, &iColsu32, &piData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc and fill new variable
pcDataOut = (char*)create_output(2, 1, iRows8, iCols8, (void*)pcData);
pucDataOut = (unsigned char*)create_output(4, 1, iRowsu8, iColsu8, (void*)pucData);
psDataOut = (short*)create_output(8, 2, iRows16, iCols16, (void*)psData);
pusDataOut = (unsigned short*)create_output(16, 2, iRowsu16, iColsu16, (void*)pusData);
piDataOut = (int*)create_output(32, 4, iRows32, iCols32, (void*)piData);
puiDataOut = (unsigned int*)create_output(64, 4, iRowsu32, iColsu32, (void*)puiData);

//create new variable
sciErr = createMatrixOfInteger8(pvApiCtx, Rhc + 1, iRows8, iCols8, pcDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger8(pvApiCtx, Rhc + 2, iRowsu8, iColsu8, pucDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfInteger16(pvApiCtx, Rhc + 3, iRows16, iCols16, psDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger16(pvApiCtx, Rhc + 4, iRowsu16, iColsu16, pusDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
```

```

//create new variable
sciErr = createMatrixOfInteger32(pvApiCtx, Rhs + 5, iRows32, iCols32, piDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger32(pvApiCtx, Rhs + 6, iRowsu32, iColsu32, piDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
LhsVar(2) = Rhs + 2;
LhsVar(3) = Rhs + 3;
LhsVar(4) = Rhs + 4;
LhsVar(5) = Rhs + 5;
LhsVar(6) = Rhs + 6;
return 0;
}

void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvDataIn)
{
    int i = 0;
    void* pvDataOut = (void*)malloc(_iSize * _iRows * _iCols);
    for(i = 0 ; i < _iRows * _iCols ; i++)
    {
        int iVal = 0;
        memcpy(&iVal, (char*)_pvDataIn + i * _iSize, _iSize);
        iVal *= _iCoeff;
        memcpy((char*)pvDataOut + i * _iSize, &iVal, _iSize);
    }
    return pvDataOut;
}

```

Scilab test script

```

a8 = int8([ 1 -2 3 -4 5; ..
           -6 7 -8 9 -10; ..
           11 -12 13 -14 15]);

au8 = uint8([ 1 2 3 4 5; ..
             6 7 8 9 10; ..
             11 12 13 14 15]);

a16 = int16([ 1 -2 3 -4 5; ..
             -6 7 -8 9 -10; ..
             11 -12 13 -14 15]);

```

```
au16 = uint16([ 1  2  3  4  5; ..
               6  7  8  9 10; ..
               11 12 13 14 15]);

a32 = int32([ 1  -2  3  -4  5; ..
             -6  7  -8  9  -10; ..
             11 -12 13 -14 15]);

au32 = uint32([ 1  2  3  4  5; ..
               6  7  8  9 10; ..
               11 12 13 14 15]);

[c8, cu8, c16, cu16, c32, cu32] = read_integer(a8, au8, a16, au16, a32, au32);

if or(c8 <> a8 * 2) then error("failed"), end
if or(cu8 <> au8 * 4) then error("failed"), end
if or(c16 <> a16 * 8) then error("failed"), end
if or(cu16 <> au16 * 16) then error("failed"), end
if or(c32 <> a32 * 32) then error("failed"), end
if or(cu32 <> au32 * 64) then error("failed"), end
```

Name

Integer writing (Scilab gateway) — How to write matrices of integers in a gateway.

Create from existing data.

Input argument profile:

Signed integer :

```
SciErr createMatrixOfInteger8(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData8)
```

```
SciErr createMatrixOfInteger16(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData16)
```

```
SciErr createMatrixOfInteger32(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData32)
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData8)
```

```
SciErr createMatrixOfUnsignedInteger16(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData16)
```

```
SciErr createMatrixOfUnsignedInteger32(void* _pvCtx, int _iVar, int _iRows, int _iCols, void* _pData32)
```

Named variable profile:

Signed integer :

```
SciErr createNamedMatrixOfInteger8(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData8)
```

```
SciErr createNamedMatrixOfInteger16(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData16)
```

```
SciErr createNamedMatrixOfInteger32(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData32)
```

Unsigned integer :

```
SciErr createNamedMatrixOfUnsignedInteger8(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData8)
```

```
SciErr createNamedMatrixOfUnsignedInteger16(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData16)
```

```
SciErr createNamedMatrixOfUnsignedInteger32(void* _pvCtx, char* _pstName, int _iRows, int _iCols, void* _pData32)
```

Parameters

_pvCtx

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar

Position in the Scilab memory where you want to put the variable

_pstName

Name of the variable for "named" functions.

_iRows

Number of rows of the new variable

_iCols

Numbers of columns of the new variable

_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32

Address of data array (size: $\_iCols * \_iRows$)

SciErr

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

Signed integer :

```
SciErr allocMatrixOfInteger8(void* _pvCtx, int _iVar, int _iRows, int _iCols, c
```

```
SciErr allocMatrixOfInteger16(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

```
SciErr allocMatrixOfInteger32(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

Unsigned integer :

```
SciErr allocMatrixOfUnsignedInteger8(void* _pvCtx, int _iVar, int _iRows, int _
```

```
SciErr allocMatrixOfUnsignedInteger16(void* _pvCtx, int _iVar, int _iRows, int _
```

```
SciErr allocMatrixOfUnsignedInteger32(void* _pvCtx, int _iVar, int _iRows, int _
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable

`_iRows`

Number of rows of the new variable

`_iCols`

Numbers of columns of the new variable

`_pcData8`, `_psData16`, `_piData32`, `_pucData8`, `_pusData16`, `_puiData32`

Returns address of data array (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of integers can be handled through the Scilab API.

Two types of functions can be used to write in the memory of Scilab.

Gateway Source

```
void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvD

int read_integer(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    //output variable info
    int iRows8      = 0;
    int iCols8      = 0;
    int iRows16     = 0;
    int iCols16     = 0;
    int iRows32     = 0;
    int iCols32     = 0;
    int iRowsu8     = 0;
    int iColsu8     = 0;
```

```
int iRowsu16    = 0;
int iColsu16    = 0;
int iRowsu32    = 0;
int iColsu32    = 0;
int iPrec       = 0;

int* piAddr8     = NULL;
int* piAddr16    = NULL;
int* piAddr32    = NULL;
int* piAddru8     = NULL;
int* piAddru16   = NULL;
int* piAddru32   = NULL;

char* pcData     = NULL;
short* psData    = NULL;
int* piData      = NULL;
unsigned char* pucData = NULL;
unsigned short* pusData = NULL;
unsigned int* puiData = NULL;

char* pcDataOut  = NULL;
short* psDataOut = NULL;
int* piDataOut   = NULL;
unsigned char* pucDataOut = NULL;
unsigned short* pusDataOut = NULL;
unsigned int* puiDataOut = NULL;

//check input/output arguments count
CheckRhs(6,6);
CheckLhs(6,6);

//get variable address
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr8);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddru8);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 3, &piAddr16);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 4, &piAddru16);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}
```



```
}

sciErr = getVarAddressFromPosition(pvApiCtx, 5, &piAddr32);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 6, &piAddru32);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr8, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT8)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru8, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT8)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr16, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT16)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru16, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT16)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr32, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT32)
{
    printError(&sciErr, 0);
    return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru32, &iPrec);
```

```
if(sciErr.iErr || iPrec != SCI_UINT32)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger8(pvApiCtx, piAddr8, &iRows8, &iCols8, &pcData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, piAddr8u, &iRowsu8, &iColsu8, &pucuData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger16(pvApiCtx, piAddr16, &iRows16, &iCols16, &psData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, piAddr16u, &iRowsu16, &iColsu16, &pusData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger32(pvApiCtx, piAddr32, &iRows32, &iCols32, &piData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, piAddr32u, &iRowsu32, &iColsu32, &puDiData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc and fill new variable
pcDataOut = (char*)create_output(2, 1, iRows8, iCols8, (void*)pcData);
pucuDataOut = (unsigned char*)create_output(4, 1, iRowsu8, iColsu8, (void*)pucuData);
psDataOut = (short*)create_output(8, 2, iRows16, iCols16, (void*)psData);
```

```
pusDataOut = (unsigned short*)create_output(16, 2, iRowsu16, iColsu16, (void*)
piDataOut = (int*)create_output(32, 4, iRowsu32, iColsu32, (void*)piData);
puiDataOut = (unsigned int*)create_output(64, 4, iRowsu32, iColsu32, (void*)p

//create new variable
sciErr = createMatrixOfInteger8(pvApiCtx, Rhs + 1, iRowsu8, iColsu8, pcDataOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger8(pvApiCtx, Rhs + 2, iRowsu8, iColsu8, p
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfInteger16(pvApiCtx, Rhs + 3, iRowsu16, iColsu16, psDataOu
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger16(pvApiCtx, Rhs + 4, iRowsu16, iColsu16
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfInteger32(pvApiCtx, Rhs + 5, iRowsu32, iColsu32, piDataOu
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger32(pvApiCtx, Rhs + 6, iRowsu32, iColsu32
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
LhsVar(2) = Rhs + 2;
LhsVar(3) = Rhs + 3;
LhsVar(4) = Rhs + 4;
LhsVar(5) = Rhs + 5;
```

```

    LhsVar(6) = Rhs + 6;
    return 0;
}

void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvD
{
    int i = 0;
    void* pvDataOut = (void*)malloc(_iSize * _iRows * _iCols);
    for(i = 0 ; i < _iRows * _iCols ; i++)
    {
        int iVal = 0;
        memcpy(&iVal, (char*)_pvDataIn + i * _iSize, _iSize);
        iVal *= _iCoeff;
        memcpy((char*)pvDataOut + i * _iSize, &iVal, _iSize);
    }
    return pvDataOut;
}

```

Scilab test script

```

a8 = int8([ 1 -2 3 -4 5; ..
           -6 7 -8 9 -10; ..
           11 -12 13 -14 15]);

au8 = uint8([ 1 2 3 4 5; ..
             6 7 8 9 10; ..
             11 12 13 14 15]);

a16 = int16([ 1 -2 3 -4 5; ..
            -6 7 -8 9 -10; ..
            11 -12 13 -14 15]);

au16 = uint16([ 1 2 3 4 5; ..
              6 7 8 9 10; ..
              11 12 13 14 15]);

a32 = int32([ 1 -2 3 -4 5; ..
            -6 7 -8 9 -10; ..
            11 -12 13 -14 15]);

au32 = uint32([ 1 2 3 4 5; ..
              6 7 8 9 10; ..
              11 12 13 14 15]);

[c8, cu8, c16, cu16, c32, cu32] = read_integer(a8, au8, a16, au16, a32, au32);

if or(c8 <> a8 * 2) then error("failed"), end
if or(cu8 <> au8 * 4) then error("failed"), end
if or(c16 <> a16 * 8) then error("failed"), end
if or(cu16 <> au16 * 16) then error("failed"), end
if or(c32 <> a32 * 32) then error("failed"), end
if or(cu32 <> au32 * 64) then error("failed"), end

```

Name

Pointer reading (Scilab gateway) — How to read pointer in a gateway.

Input argument profile:

```
SciErr createNamedPointer(void* _pvCtx, char* _pstName, void* _pvPtr)
```

Named variable profile:

```
SciErr readNamedPointer(void* _pvCtx, char* _pstName, void** _pvPtr)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the Scilab variable.

`_pvPtr`

Return address of pointer.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how pointer can be handled through the Scilab API.

Gateway Source

```
int read_pointer(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    CheckRhs(0,1);
    CheckLhs(1,1);

    if(Rhs == 0)
    { //create mode
        double* pdblData = (double*)malloc(sizeof(double) * 2 * 2);
        pdblData[0] = 1;
        pdblData[1] = 3;
        pdblData[2] = 2;
        pdblData[3] = 4;

        sciErr = createPointer(pvApiCtx, Rhs + 1, (void*)pdblData);
    }
    else if(Rhs == 1)
    { //read mode
        int iType = 0;
        int* piAddr = NULL;
        void* pvPtr = NULL;
        double* pdblData = NULL;

        sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
```

```
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciErr = getPointer(pvApiCtx, piAddr, &pvPtr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pdblData = (double*)pvPtr;

sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, 2, 2, pdblData);
}
else
{
    return 0;
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

LhsVar(1) = Rhs + 1;
return 0;
}
```

Scilab test script

```
b_ref = [1,2;3,4];
a = read_pointer();
b = read_pointer(a);
if or(b <> b_ref) then error("failed"), end
```

Name

Pointer writing (Scilab gateway) — How to write pointer in a gateway.

Create from existing data.

Input argument profile:

```
SciErr getPointer(void* _pvCtx, int* _piAddress, void** _pvPtr)
```

Named variable profile:

```
SciErr createNamedPointer(void* _pvCtx, char* _pstName, void** _pvPtr)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_piAddress`

Address of the Scilab variable.

`_pvPtr`

Address of pointer.

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

```
SciErr allocPointer(void* _pvCtx, int _iVar, void** _pvPtr)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pvPtr`

Return address of pointer.

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how pointer can be handled through the Scilab API.

Two types of functions can be used to write in the memory of Scilab.

Gateway Source

```
int read_pointer(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
```

```
CheckRhs(0,1);
CheckLhs(1,1);

if(Rhs == 0)
{
  //create mode
  double* pdblData = (double*)malloc(sizeof(double) * 2 * 2);
  pdblData[0] = 1;
  pdblData[1] = 3;
  pdblData[2] = 2;
  pdblData[3] = 4;

  sciErr = createPointer(pvApiCtx, Rhs + 1, (void*)pdblData);
}
else if(Rhs == 1)
{
  //read mode
  int iType = 0;
  int* piAddr = NULL;
  void* pvPtr = NULL;
  double* pdblData = NULL;

  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  sciErr = getPointer(pvApiCtx, piAddr, &pvPtr);
  if(sciErr.iErr)
  {
    printError(&sciErr, 0);
    return 0;
  }

  pdblData = (double*)pvPtr;

  sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, 2, 2, pdblData);
}
else
{
  return 0;
}

if(sciErr.iErr)
{
  printError(&sciErr, 0);
  return 0;
}

LhsVar(1) = Rhs + 1;
return 0;
}
```

Scilab test script

```
b_ref = [1,2;3,4];  
a = read_pointer();  
b = read_pointer(a);  
if or(b <> b_ref) then error("failed"), end
```

Name

Polynomial Symbolic Variable (Scilab gateway) — How to get the symbolic variable name.

Input argument profile:

```
SciErr getPolyVariableName(void* _pvCtx, int* _piAddress, char* _pstVarName, int
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the variable.

`_pstVarName`

Return the symbolic variable name

`_piVarNameLen`

Return the length of `_pstVarName`

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how to get the symbolic variable name.

Gateway Source

```
int read_poly(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i,j;
    //variable info
    int iRows = 0;
    int iCols = 0;
    int iVarLen = 0;
    int* piAddr = NULL;
    int* piNbCoef = NULL;
    double** pdblReal = NULL;
    double** pdblImg = NULL;
    char* pstVarname = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    if(isVarComplex(pvApiCtx, piAddr) == FALSE)
    {
```

```

//Error
return 0;
}

//get variable name length
sciErr = getPolyVariableName(pvApiCtx, piAddr, NULL, &iVarLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc buff to receive variable name
pstVarname = (char*)malloc(sizeof(char) * (iVarLen + 1)); //1 for null terminator
//get variable name
sciErr = getPolyVariableName(pvApiCtx, piAddr, pstVarname, &iVarLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//First call: retrieve dimension
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL, 1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc array of coefficient
piNbCoef = (int*)malloc(sizeof(int) * iRows * iCols);
//Second call: retrieve coefficient
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, NULL, 2);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc arrays of data
pdblReal = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
    pdblImg[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
}

//Third call: retrieve data
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, pdblReal, pdblImg, 3);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```

```
//Do something with Data

//Invert polynomials in the matrix and invert coefficients

for(i = 0 ; i < (iRows * iCols) / 2 ; i++)
{
    int iPos1    = iRows * iCols - 1 - i;
    double* pdblSave = NULL;
    int iNbCoefSave = 0;

    //switch array of coefficient
    pdblSave = pdblReal[i];
    pdblReal[i] = pdblReal[iPos1];
    pdblReal[iPos1] = pdblSave;

    pdblSave = pdblImg[i];
    pdblImg[i] = pdblImg[iPos1];
    pdblImg[iPos1] = pdblSave;

    //switch number of coefficient
    iNbCoefSave = piNbCoef[i];
    piNbCoef[i] = piNbCoef[iPos1];
    piNbCoef[iPos1] = iNbCoefSave;
}

//switch coefficient
for(i = 0 ; i < iRows * iCols ; i++)
{
    for(j = 0 ; j < piNbCoef[i] / 2 ; j++)
    {
        int iPos2    = piNbCoef[i] - 1 - j;
        double dblVal = pdblReal[i][j];
        pdblReal[i][j] = pdblReal[i][iPos2];
        pdblReal[i][iPos2] = dblVal;

        dblVal = pdblImg[i][j];
        pdblImg[i][j] = pdblImg[i][iPos2];
        pdblImg[i][iPos2] = dblVal;
    }
}

sciErr = createComplexMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarname, iRows, iCols);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//free OS memory
free(pstVarname);
free(piNbCoef);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}
```

```

free(pdblReal);
free(pdblImg);

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
return 0;
}

```

Scilab test script

```

coeff1 = [ ..
29*%i, 22*%i, 16*%i, 11*%i, 7*%i, 30, 23, 17, 12, 8, -31*%i, -24*%i, -18*%i, -13*%i, -9*%i, 32
4*%i, 2*%i, %i, 22, 16, 5, -3, 0, -23*%i, -17*%i, -6*%i, 0, 0, 24, 18, 0, 0, 0, -25*%i, -19*%i, 0, 0
11, 7, 4, 2, 1, -12*%i, -8*%i, -5*%i, 3*%i, 0, 13, 9, 6, 0, 0, -14*%i, -10*%i, 0, 0, 0, 15, 0, 0, 0, 0,

x = poly(0, "x");
p1 = 1;
p2 = 2 * x + 3 * %i;
p3 = 4 * x**2 - 5 * %i * x + 6;
p4 = 7 * x**3 - 8 * %i * x**2 + 9 * x - 10 * %i;
p5 = 11 * x**4 - 12 * %i * x**3 + 13 * x**2 - 14 * %i * x + 15;
p6 = 16 * x**5 - 17 * %i * x**4 + 18 * x**3 - 19 * %i * x**2 + 20 * x - 21 * %i;
p7 = 22 * x**6 - 23 * %i * x**5 + 24 * x**4 - 25 * %i * x**3 + 26 * x**2 - 27 * %i;
p8 = %i;
p9 = 2 * %i * x - 3;
p10 = 4 * %i * x**2 + 5 * x - 6 * %i;
p11 = 7 * %i * x**3 + 8 * x**2 - 9 * %i * x + 10;
p12 = 11 * %i * x**4 + 12 * x**3 - 13 * %i * x**2 + 14 * x - 15 * %i;
p13 = 16 * %i * x**5 + 17 * x**4 - 18 * %i * x**3 + 19 * x**2 - 20 * %i * x + 21 * %i;
p14 = 22 * %i * x**6 + 23 * x**5 - 24 * %i * x**4 + 25 * x**3 - 26 * %i * x**2 + 27 * %i;
p15 = 29 * %i * x**7 + 30 * x**6 - 31 * %i * x**5 + 32 * x**4 - 33 * %i * x**3 + 34 * x**2 - 35 * %i * x + 36 * %i;
p = [p1, p2, p3, p4, p5 ; p6, p7, p8, p9 ,p10 ; p11, p12, p13, p14, p15];

p1 = read_poly(p);
coeff2 = coeff(p1);
if or(coeff2 <> coeff1) then error("failed"), end

```

Name

Polynomial reading (Scilab gateway) — How to read matrices of polynomials in a gateway.

Input argument profile:

```
SciErr getMatrixOfPoly(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)  
SciErr getComplexMatrixOfPoly(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

```
SciErr readNamedMatrixOfPoly(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)  
SciErr readNamedComplexMatrixOfPoly(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`_piNbCoef`

Return number of coefficient for each polynomial. (must be allocated)

`_pdblReal`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`, must be allocated)

`_pdblImg`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`, must be allocated)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of polynomials can be handled through the Scilab API.

Gateway Source

```
int read_poly(char *fname,unsigned long fname_len)  
{  
    SciErr sciErr;  
    int i,j;  
    //variable info  
    int iRows    = 0;  
    int iCols    = 0;
```

```

int iVarLen    = 0;
int* piAddr    = NULL;
int* piNbCoef  = NULL;
double** pdblReal = NULL;
double** pdblImg = NULL;
char* pstVarname = NULL;

//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(isVarComplex(pvApiCtx, piAddr) == FALSE)
{
    //Error
    return 0;
}

//get variable name length
sciErr = getPolyVariableName(pvApiCtx, piAddr, NULL, &iVarLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc buff to receive variable name
pstVarname = (char*)malloc(sizeof(char) * (iVarLen + 1)); //1 for null terminat
//get variable name
sciErr = getPolyVariableName(pvApiCtx, piAddr, pstVarname, &iVarLen);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//First call: retrieve dimension
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL, 1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//alloc array of coefficient
piNbCoef = (int*)malloc(sizeof(int) * iRows * iCols);
//Second call: retrieve coefficient
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, NULL);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

```

```

}

//alloc arrays of data
pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pdblReal[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
    pdblImg[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
}

//Third call: retrieve data
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, pd
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//Do something with Data

//Invert polynomials in the matrix and invert coefficients

for(i = 0 ; i < (iRows * iCols) / 2 ; i++)
{
    int iPos1    = iRows * iCols - 1 - i;
    double* pdblSave = NULL;
    int iNbCoefSave = 0;

    //switch array of coefficient
    pdblSave    = pdblReal[i];
    pdblReal[i] = pdblReal[iPos1];
    pdblReal[iPos1] = pdblSave;

    pdblSave    = pdblImg[i];
    pdblImg[i] = pdblImg[iPos1];
    pdblImg[iPos1] = pdblSave;

    //switch number of coefficient
    iNbCoefSave = piNbCoef[i];
    piNbCoef[i] = piNbCoef[iPos1];
    piNbCoef[iPos1] = iNbCoefSave;
}

//switch coefficient
for(i = 0 ; i < iRows * iCols ; i++)
{
    for(j = 0 ; j < piNbCoef[i] / 2 ; j++)
    {
        int iPos2    = piNbCoef[i] - 1 - j;
        double dblVal = pdblReal[i][j];
        pdblReal[i][j] = pdblReal[i][iPos2];
        pdblReal[i][iPos2] = dblVal;

        dblVal    = pdblImg[i][j];
        pdblImg[i][j] = pdblImg[i][iPos2];
        pdblImg[i][iPos2] = dblVal;
    }
}

```



```

    }
}

sciErr = createComplexMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarname, 1);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//free OS memory
free(pstVarname);
free(piNbCoef);

for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pdblReal[i]);
    free(pdblImg[i]);
}

free(pdblReal);
free(pdblImg);

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
return 0;
}

```

```

coeff1 = [ ..
29*i, 22*i, 16*i, 11*i, 7*i, 30, 23, 17, 12, 8, -31*i, -24*i, -18*i, -13*
4*i, 2*i, i, 22, 16, 5, -3, 0, -23*i, -17*i, -6*i, 0, 0, 24, 18, 0, 0, 0, -25*%
11, 7, 4, 2, 1, -12*i, -8*i, -5*i, 3*i, 0, 13, 9, 6, 0, 0, -14*i, -10*i, 0, 0, 0,
x = poly(0, "x");
p1 = 1;
p2 = 2 * x + 3 * %i;
p3 = 4 * x**2 - 5 * %i * x + 6;
p4 = 7 * x**3 - 8 * %i * x**2 + 9 * x - 10 * %i;
p5 = 11 * x**4 - 12 * %i * x**3 + 13 * x**2 - 14 * %i * x + 15;
p6 = 16 * x**5 - 17 * %i * x**4 + 18 * x**3 - 19 * %i * x**2 + 20 * x - 21 * %i;
p7 = 22 * x**6 - 23 * %i * x**5 + 24 * x**4 - 25 * %i * x**3 + 26 * x - 27 * %i;
p8 = %i;
p9 = 2 * %i * x - 3;
p10 = 4 * %i * x**2 + 5 * x - 6 * %i;
p11 = 7 * %i * x**3 + 8 * x**2 - 9 * %i * x + 10;
p12 = 11 * %i * x**4 + 12 * x**3 - 13 * %i * x**2 + 14 * x - 15 * %i;
p13 = 16 * %i * x**5 + 17 * x**4 - 18 * %i * x**3 + 19 * x**2 - 20 * %i * x + 21;
p14 = 22 * %i * x**6 + 23 * x**5 - 24 * %i * x**4 + 25 * x**3 - 26 * %i * x**2 + 27 * x - 28 * %i;
p15 = 29 * %i * x**7 + 30 * x**6 - 31 * %i * x**5 + 32 * x**4 - 33 * %i * x**3 + 34 * x**2 - 35 * %i * x + 36;
p = [p1, p2, p3, p4, p5 ; p6, p7, p8, p9 ,p10 ; p11, p12, p13, p14,
p1 = read_poly(p);
coeff2 = coeff(p1);
if or(coeff2 <> coeff1) then error("failed"), end

```



Name

Polynomial writing (Scilab gateway) — How to write matrices of polynomials in a gateway.

Input argument profile:

```
SciErr createMatrixOfPoly(void* _pvCtx, int _iVar, char* _pstVarName, int _iRow  
SciErr createMatrixOfPoly(void* _pvCtx, int _iVar, char* _pstVarName, int _iRow
```

Named variable profile:

```
SciErr createNamedMatrixOfPoly(void* _pvCtx, char* _pstName, char* _pstVarName,  
SciErr createNamedComplexMatrixOfPoly(void* _pvCtx, char* _pstName, char* _pstV
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable

`_pstName`

Name of the variable for "named" functions.

`_pstVarName`

Variable name in polynomials (Scilab5: 4 characters max)

`_iRows`

Number of rows of the new variable

`_iCols`

Numbers of columns of the new variable

`_piNbCoef`

Number of coefficient for each polynomial.

`_pdblReal`

Address of array of double* with real part of coefficient (size: `_iCols * _iRows`)

`_pdblImg`

Address of array of double* with imaginary part of coefficient (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of polynomials can be handled through the Scilab API.

Gateway Source

```
int write_poly(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
```

```

//output variable info : polinomial matrix 2 x 4
//[ x + 2          x^2 - 4x + 5    4x^3 - 14x^2 + 18 ;
// 2x^3 - 12x^2 + 64      1          8x^5 + 32x^3]
int iRows          = 2;
int iCols          = 3;

//varname "x"
char pstVarName[2] = {"x"};

//coefficient array
int piNbCoef[6]    = {2,4,3,1,4,6};

//data array
double *pdblReal[6] = {0};
double pdblPoly0[2] = {2,1};
double pdblPoly1[4] = {64,0,-12,2};
double pdblPoly2[3] = {5,-4,1};
double pdblPoly3[1] = {1};
double pdblPoly4[4] = {18,0,-14,4};
double pdblPoly5[6] = {0,0,0,32,0,8};

pdblReal[0]        = pdblPoly0;
pdblReal[1]        = pdblPoly1;
pdblReal[2]        = pdblPoly2;
pdblReal[3]        = pdblPoly3;
pdblReal[4]        = pdblPoly4;
pdblReal[5]        = pdblPoly5;

sciErr = createMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarName, iRows, iCols, piNbCoef);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
return 0;
}

```

Scilab test script

```

p_ref = [2 5 18 1 -4 0 0 1 -14 0 0 4 0 0 0 0 0 0;64 1 0 0 0 0 -12 0 0 2 0 32 0 0 0 0 0];
l = list();
a = write_poly();
p = coeff(a);
if or(p <> p_ref) then error("failed"), end

```

Name

Sparse matrix reading (Scilab gateway) — How to read sparse matrix in a gateway.

Input argument profile:

```
SciErr getSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getComplexSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

```
SciErr readNamedSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

```
SciErr readNamedComplexSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows, int* _piCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`_piNbItem`

Return number of non zero value.

`_piNbItemRow`

Return number of item in each rows (size: `_iRows`).

`_piColPos`

Return column position for each item (size: `_iNbItem`).

`_pdblReal`

Return address of real data array (size: `_iCols * _iRows`)

`_pdblImg`

Return address of imaginary data array (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how sparse matrix can be handled through the Scilab API.

Gateway Source

```
int read_sparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
```

```

int i,j,k;
int* piAddr = NULL;

int iRows = 0;
int iCols = 0;
int iNbItem = 0;
int* piNbItemRow = NULL;
int* piColPos = NULL;

double* pdblReal = NULL;
double* pdblImg = NULL;

CheckRhs(1,1);

sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

if(isVarComplex(pvApiCtx, piAddr))
{
    sciErr = getComplexSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem, &piNbItemRow, &piColPos);
}
else
{
    sciErr = getSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem, &piNbItemRow, &piColPos);
}

if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

sciprint("Sparse %d item(s)\n", iNbItem);

k = 0;
for(i = 0 ; i < iRows ; i++)
{
    for(j = 0 ; j < piNbItemRow[i] ; j++)
    {
        sciprint("(%d,%d) = %f", i+1, piColPos[k], pdblReal[k]);
        if(isVarComplex(pvApiCtx, piAddr))
        {
            sciprint(" %+fi", pdblImg[k]);
        }
        sciprint("\n");
        k++;
    }
}

//assign allocated variables to Lhs position
LhsVar(1) = 0;
return 0;
}

```

Scilab test script

```
sp=sparse([1,2;4,5;3,10],[1 + 2*i,2 - 3*i,-3 + 4*i]);  
read_sparse(sp);
```

Name

Sparse writing (Scilab gateway) — How to write sparse matrix in a gateway.

Create from existing data.

Input argument profile:

```
SciErr createSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```

```
SciErr createComplexSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```

Named variable profile:

```
SciErr createNamedSparseMatrix(void* _pvCtx, char* _pstName, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```

```
SciErr createNamedComplexSparseMatrix(void* _pvCtx, char* _pstName, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

`_iVar`

Position in the Scilab memory where you want to put the variable.

`_pstName`

Name of the variable for "named" functions.

`_iRows`

Number of rows of the new variable.

`_iCols`

Number of columns of the new variable.

`_iNbItem`

Number of non zero itmes in the sparse.

`_piNbItemRow`

Number of item in each rows (size: `_iRows`).

`_piColPos`

Column position for each item (size: `_iNbItem`).

`_pdblReal`

Address of real data array (size: `_iNbItem`).

`_pdblImg`

Address of imaginary data array (size: `_iNbItem`).

This argument does not exist with `createSparseMatrix` and `createNamedSparseMatrix`.

`SciErr`

Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```

```
SciErr allocComplexSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _iNbItem, int _piNbItemRow, int _piColPos, void* _pdblReal, void* _pdblImg)
```


Parameters

- `_pvCtx`
Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.
- `_iVar`
Position in the Scilab memory where you want to put the variable.
- `_iRows`
Number of rows of the new variable.
- `_iCols`
Number of columns of the new variable.
- `_iNbItem`
Number of non zero itmes in the sparse.
- `_piNbItemRow`
Return address of number of item in each rows (size: `_iRows`).
- `_piColPos`
Return address of column position for each item (size: `_iNbItem`).
- `_pdblReal`
Address of real data array (size: `_iNbItem`).
- `_pdblImg`
Address of imaginary data array (size: `_iNbItem`).
- This argument does not exist with `allocSparseMatrix`.
- `SciErr`
Error structure where is stored errors messages history and first error number.

Description

This help describes how to add sparse matrix.

Two types of functions can be used to write in the memory of Scilab.

Gateway Source

```
int write_sparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;

    int piNbItemRow[] = {1,2,1};
    int piColPos[] = {8,4,7,2};
    double pdblSReal[] = {1,2,3,4};
    double pdblSImg[] = {4,3,2,1};
    int iNbItem = 4;

    sciErr = createComplexSparseMatrix(pvApiCtx, Rhs + 1, 3, 10, iNbItem, piNbItemRow, piColPos, pdblSReal, pdblSImg);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
}
```

```
}  
  
LhsVar(1) = 1;  
return 0;  
}
```

Scilab test script

```
sp_ref = sparse([1,8;2,4;2,7;3,2],[1+4*i,2+3*i,3+2*i,4+i], [3,10]);  
sp = write_sparse();  
if or(sp <> sp_ref) then error("failed"), end
```

Name

String reading (Scilab gateway) — How to read matrices of strings in a gateway.

Input argument profile:

```
SciErr getMatrixOfString(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

```
SciErr getMatrixOfWideString(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols)
```

Named variable profile:

```
SciErr createNamedMatrixOfString(void* _pvCtx, char* _pstName, int _iRows, int _iCols)
```

```
SciErr createNamedMatrixOfWideString(void* _pvCtx, char* _pstName, int _iRows, int _iCols)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_piAddress`

Address of the Scilab variable.

`_pstName`

Name of the variable for "named" functions.

`_piRows`

Return number of rows.

`_piCols`

Return number of columns.

`_piLength`

Address of array of strings length (must be allocated size: `_piRows * _piCols`)

`_pstStrings`

Address of array of `char*` (must be allocated size: `_piRows * _piCols`)

`_pwstStrings`

Address of array of `wchar_t*` (must be allocated size: `_piRows * _piCols`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of strings can be handled through the Scilab API.

Gateway Source

```
int read_string(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i,j;
    int iLen = 0;
    //variable info
    int iRows = 0;
    int iCols = 0;
    int* piAddr = NULL;
```

```

int* piLen = NULL;
char** pstData = NULL;

//output variable
int iRowsOut = 1;
int iColsOut = 1;
char* pstOut = NULL;

//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

//get variable address
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//fisrt call to retrieve dimensions
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

piLen = (int*)malloc(sizeof(int) * iRows * iCols);
//second call to retrieve length of each string
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, piLen, NULL);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
    pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term.
}
//third call to retrieve data
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, piLen, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//computer length of all strings
for(i = 0 ; i < iRows * iCols ; i++)
{
    iLen += piLen[i];
}

//alloc output variable
pstOut = (char*)malloc(sizeof(char) * (iLen + iRows * iCols));

```

```

//initialize string to 0x00
memset(pstOut, 0x00, sizeof(char) * (iLen + iRows * iCols));

//concat input strings in output string
for(i = 0 ; i < iRows ; i++)
{
    for(j = 0 ; j < iCols ; j++)
    {
        int iCurLen = strlen(pstOut);
        if(iCurLen)
        {
            strcat(pstOut, " ");
        }
        strcpy(pstOut + strlen(pstOut), pstData[j * iRows + i]);
    }
}

//create new variable
sciErr = createMatrixOfString(pvApiCtx, Rhs + 1, iRowsOut, iColsOut, &pstOut);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//free memory
free(piLen);
for(i = 0 ; i < iRows * iCols ; i++)
{
    free(pstData[i]);
}
free(pstData);
free(pstOut);

LhsVar(1) = Rhs + 1;
return 0;
}

```

Scilab test script

```

a_ref = ["may the puffin be with you"];
a = ["may", "the", "puffin"; "be","with","you"];
b = read_string(a);
if a_ref <> b then error("failed"), end

```

Name

String writing (Scilab gateway) — How to write matrices of string in a gateway.

Input argument profile:

```
SciErr createMatrixOfString(void* _pvCtx, int _iVar, int _iRows, int _iCols, char** _pstStrings)
```

```
SciErr createMatrixOfWideString(void* _pvCtx, int _iVar, int _iRows, int _iCols, char** _pstStrings)
```

Named variable profile:

```
SciErr createNamedMatrixOfString(void* _pvCtx, char* _pstName, int _iRows, int _iCols, char** _pstStrings)
```

```
SciErr createNamedMatrixOfWideString(void* _pvCtx, char* _pstName, int _iRows, int _iCols, char** _pstStrings)
```

Parameters

`_pvCtx`

Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

`_iVar`

Position in the Scilab memory where you want to put the variable

`_pstName`

Name of the variable for "named" functions.

`_iRows`

Number of rows of the new variable

`_iCols`

Numbers of columns of the new variable

`_pstStrings`

Address of array of char* (size: `_iCols * _iRows`)

`SciErr`

Error structure where is stored errors messages history and first error number.

Description

This help describes how matrix of strings can be handled through the Scilab API.

Gateway Source

```
int write_string(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    //variable info : matrix of string 2 x 3
    int iRows = 2;
    int iCols = 3;
    char** pstData = NULL;

    //data to put in the new variable
    char string11[] = "may";
    char string21[] = "be";
```

```
char string12[] = "the";
char string22[] = "with";
char string13[] = "puffin";
char string23[] = "you";

//alloc new array
pstData = (char**)malloc(sizeof(char*) * iRows * iCols);

//copy data address to the "main" array
pstData[0] = string11;
pstData[1] = string21;
pstData[2] = string12;
pstData[3] = string22;
pstData[4] = string13;
pstData[5] = string23;

//create the variable
sciErr = createMatrixOfString(pvApiCtx, Rhs + 1, iRows, iCols, pstData);
if(sciErr.iErr)
{
    printError(&sciErr, 0);
    return 0;
}

//free container
free(pstData);

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
return 0;
}
```

Scilab test script

```
a_ref = "may the puffin be with you ";
b = [];
a = write_string();
for i=1:size(a,"r")
    for j=1:size(a,"c")
        b = b + a(i,j);
        b = b + " ";
    end
end
if b <> a_ref then error("failed"), end
```

Part LVIII. Online help management

Name

`add_help_chapter` — Add an entry in the helps list

```
ok = add_help_chapter(title,path[,mode])
```

Parameters

`title`

String array, Help chapters title

`path`

String array, Help chapters path

`mode`

A boolean, if TRUE, the chapter is considered as belongs to a internal modules, otherwise, the chapter is added as external module. It's an optional input argument and its default value is %F.

`ok`

a vector of booleans. The same size as the `path` variable. If the chapter page number `i` was not existing in the help system then `ok(i)` is set to %t.

Description

This function adds a new entry in the helps list. If the path is already present in the help list, `add_help_chapter` skips the add. The function checks if the directory exist.

See Also

`help` , `del_help_chapter`

Authors

Serge Steer , INRIA

Name

`apropos` — searches keywords in Scilab help

```
apropos(key)
apropos(regex)
```

Parameters

`key`

character string. give the sequence of characters to be found

`regex`

character string. give the regular expression to be found (only with "Scilab Browser")

Description

`apropos(key)` looks for Scilab help files containing keywords `key` in their short description section.

`apropos(regex)` looks for Scilab help files containing regular expression `regex` in their short description section.

Examples

```
apropos('ode')
apropos ode
apropos "list of"
apropos "sin.*hyperbolic"
apropos "^ab" //search help beginning the two characters "ab"
apropos "quadratic.*solver"
```

See Also

`help` , `man`

Name

`del_help_chapter` — Delete an entry in the helps list

```
del_help_chapter(title[,mode])
```

Parameters

`title`

String array, Help chapters title

`mode`

A boolean, if TRUE, the chapter is considered as belongs to a internal modules, otherwise, the chapter is added as external module. It's an optional input argument and its default value is %F.

Description

This function deletes an entry in the helps list.

See Also

`help` , `add_help_chapter`

Name

help — on-line help command

```
help(key)
help
```

Parameters

key
character string. Gives the help page to be found

Description

help without argument displays the hypertext page containing the list of help chapters.

help(key) displays the Scilab help file associated with the given key. If no help file is found, help(key) automatically call apropos(key).

See man for more explanation on how to write new help pages .

See Also

apropos , man

Name

`help_from_sci` — Generate help files and demo files from the head comments section of a .sci source file.

```
help_from_sci() // generate an empty function template
help_from_sci(funname,helpdir) // generate helpdir/funname.xml from funname.sci
help_from_sci(dirname,helpdir) // process dirname/*.sci and create helpdir/*.xm
help_from_sci(dirname,helpdir,helpdir) // as above but also creating helpdir/*.
[helptxt,demotxt]=help_from_sci(funname) // return funname.xml and funname.dem.
```

Parameters

`funname:`

the name of a single .sci source file to be processed.

`dirname:`

directory name where all .sci files will be processed.

`helpdir:`

optional path where the .xml help file will be created.

`demodir:`

optional path where .dem.sce demo files will be created based on code from the Examples section.

`helptxt:`

returns the XML help code if helpdir is empty, or the path to the new .xml file.

`demotxt:`

returns the demo code if demodir is empty, or the path to the new .dem.sc file.

Description

`help_from_sci` is a revised version of the `help_skeleton` function. Its objective is to generate .xml help files based on the head comments section of .sci source files. Optionally .dem.sce demo files can be generated based on code from the Examples section in the head comments section of .sci files.

In order for `help_from_sci` to format the .xml file properly the head comments section should comply with some simple formatting rules.

The first comment line following the function definition should contain a short description of the function.

The remaining comments are formatted according to the following (optional) headlines: "Calling Sequence", "Parameters", "Description", "Examples", "See also", "Used functions", "Authors" and "Bibliography".

The following guidelines should be used when writing the source code comments:

- **Calling Sequence** - one example pr. line.
- **Parameters** - separate parameter name and description by a ":". Keep the description of each parameter on the same line.
- **Description** - formatting of the text can be done using XML commands. Adding an empty comment line in the Description section is interpreted as the start of a new paragraph.
- **See also** - list one function name pr line.

- **Authors** - write one author on each line following the Authors headline. Use ";" to separate the authors name from any add additional information.
- **Bibliography** - write one reference pr line following the References headline.

Remark

If your .sci file have some Non-ASCII characters, help_from_sci can fail to generate .xml help file associated. In this case, please save as .sci file with UTF-8 (NO-BOM) file encoding.

Examples

```
help_from_sci() // Open an empty source code template in the scilab editor.
// Save this template as test_fun.sci in the current directory before running
// the next example commands.

help_from_sci('test_fun') // return the xml skeleton as a text string

help_from_sci('test_fun','.') // create the xml help file in the current dir

// create both a xml help file and a demo file in the current directory.
help_from_sci('test_fun','.', '.')

// From a toolbox root directory a typical calling sequence would be:
// help_from_sci('macros','help\en_US','demos')
// This command would process all .sci files in the macros directory
// and use the head comments section to update all .xml help files in the
// help\en_US directory an rebuild the .dem.sce files in the demos\ directory.
```

See also

help, help_skeleton, xmltohtml

Authors

T. Pettersen
torbjorn.pettersen@broadpark.no

Name

help_skeleton — build the skeleton of the xml help file associated to a Scilab function

```
txt = help_skeleton(funname [,path [, language]])
```

Parameters

funname

character string : the name of the function

path

character string : the path where the file will be create if required. If this argument is not given the skeleton is returned as a string.

language

character string :with possible value "fr_FR" or "en_US" the defaultis "en_US"

txt

the XML code or the complete xml file path

Description

`txt = help_skeleton(funname)` generates a vector of strings containing the skeleton of the XML code describing the help of the function `funname`.

`fullpath = help_skeleton(funname,dirpath)` generates the XML code describing the help of the function `funname` in a file named `funname.xml` in the directory specified by the path `dirpath`. In this case the function returns the file path.

Examples

```
function [y,z]=foo(a,b),y=a+b,z=1,endfunction
p=help_skeleton('foo',TMPDIR)
if (isdef('editor') | (funptr('editor')<>0)) then
    editor(p);
end
```

See Also

help

Authors

Serge Steer, INRIA

Name

man — on line help XML file description format

Description

The on line help source files are written in XML.

Source files (with extension .xml) can be found in the `<SCIDIR>/man/<language>/*` directories. The file name is usually associated to a keyword (corresponding to a function name most of the cases) it describes.

A few words about XML

An XML file resembles to an HTML file but with both a more rigid and free syntax. Free because you may build your own tags: the set of tags together with its rules must be described somewhere, generally in another file (`<SCIDIR>/man/manrev.dtd` for scilab), and rigid because, once the tags and rules are defined (which are called the Definition Type Document: DTD), you must respect its (in particular to every open tags `<MY_TAG>` must correspond a closed `</MY_TAG>`).

The DTD `manrev.dtd` is written in SGML and precises the exact syntax required by a scilab XML help page. So if you know this language you may read this file. The following annotated example (see the next section) shows you some possibilities offered by this DTD and may be enough to write simple help pages.

Once an XML page is written and conforms to the DTD, it may be transformed in HTML to be read by your favorite browser or by the tcltk scilab browser (see section browser choice in this page). The XML -> HTML translation is controlled by a set of rules written in the (XML) file `<SCIDIR>/man/language/html.xsl`. Those rules are currently more or less restricted to fit the tcltk scilab browser features (which may display correctly only basic HTML): if you use a real HTML browser and want a better appearance you have to modify this file.

How to write a simple xml scilab help page: the lazy way

If one want to write the xml file associated to a new scilab function he or she may use the Scilab function `help_skeleton` to produce the skeleton of the xml file. In most cases the user will not be required to know xml syntax.

How to write a simple xml scilab help page: an example

Here is a simple annotated XML scilab help page which describes an hypothetical foo scilab function. In the following, the XML file is displayed in a `type writer` font and cut-out in several parts, each part being preceded by some associated explanations. The entire XML file `foo.xml` is in the `<SCIDIR>/man/eng/utility` directory and the result may be displayed by clicking on foo. (you may find others examples in the `<SCIDIR>/examples/man-examples-xml` directory). **Finally** note that some tag pairs `<TAG>`, `</TAG>` have been renamed here `<ATAG>`, `</ATAG>`. This is because some scilab scripts which do some work on or from the xml files don't verify if a tag is inside a `VERBATIM` entry.

The 3 first lines of the file are mandatory, the second precises the path to the DTD file and the third, formed by the `<MAN>` tag, begin the hierarchical description (the file must finish with the `</MAN>` tag). The 4 followings entries : `LANGUAGE`, `TITLE`, `TYPE` and `DATE`, are also mandatory (in this order) the text corresponding to `<TYPE>` being generally 'Scilab function' (most of the cases) but may be simply 'Scilab keyword' or 'Scilab data type', ..., depending of what explains the help page.


```
<!DOCTYPE MAN SYSTEM "<SCIDIR>/man/manrev.dtd">
<MAN>
  <LANGUAGE>eng</LANGUAGE>
  <TITLE>foo</TITLE>
  <TYPE>Scilab function</TYPE>
  <DATE>$LastChangedDate$</DATE>
```

The first of these 2 following entries (**SHORT_DESCRIPTION**) is mandatory and important since the words of the short description text, are used by the `apropos` command to search help pages from a keyword: the short description is used to build the `whatis.html` file corresponding to your toolbox and the `apropos` keyword command looks in all the `whatis` files and then proposes the links to every page containing the word `keyword` in its short description (in fact the actual associated tags are `<SHORT_DESCRIPTION>` and `</SHORT_DESCRIPTION>` and not `<AShort_Description>` and `</AShort_Description>`). The next entry (**CALLING_SEQUENCE**) must be used if you describe a function (but is not strictly mandatory). If your function have several calling sequences use several **CALLING_SEQUENCE_ITEM** entries.

```
<AShort_Description name="foo">foo short description</AShort_Description>
<Calling_Sequence>
  <Calling_Sequence_Item>[y] = foo(x)</Calling_Sequence_Item>
</Calling_Sequence>
```

The following entry (**PARAM**) is not strictly mandatory but is the good one to describe each parameters (input and output) in case of a function.

```
<PARAM>
  <PARAM_Indent>
    <PARAM_Item>
      <PARAM_Name>x</PARAM_Name>
      <PARAM_Description>
        <SP>: what may be x</SP>
      </PARAM_Description>
    </PARAM_Item>
    <PARAM_Item>
      <PARAM_Name>y</PARAM_Name>
      <PARAM_Description>
        <SP>: what may be y</SP>
      </PARAM_Description>
    </PARAM_Item>
  </PARAM_Indent>
</PARAM>
```

The **DESCRIPTION** entry is perhaps the most significant one (but not strictly mandatory) and may be more sophisticated than in this example (for instance you may have **DESCRIPTION_ITEM** sub-entries). Here you see how to write several paragraphs (each one enclosed between the `<P>` and `</P>` tags), how to emphasis a variable or a function name (by enclosing it between the `<VERB>` and `</VERB>` tags), how to emphasis a part of text (`<EM>` or `<BD>` and `<TT>` to put it in a type writer font)), and finally, how to put a link onto another help page (in fact the actual associated tags are `<LINK>` and `</LINK>` and not `<ALINK>` and `</ALINK>`).

```
<DESCRIPTION>
  <P>
    A first paragraph which explains what computes the foo function.
    If you want to emphasis a parameter name then you use the follow
    tag <VERB>x</VERB>, if you want to emphasis a part of text
    <EM>inclose it inside theses tags</EM> and use theses ones
    <BD>to have a bold font</BD> and finally <TT>for a type writer
  </P>
  <P>
    A second paragraph... Here is an example of a link to another p
    <ALINK>man</ALINK>.
  </P>
</DESCRIPTION>
```

Here is how to write your own entry, for instance to describe some outside remarks and/or notes about your wonderful function.

```
<SECTION label='Notes'>
  <P>
    Here is a list of notes :
  </P>
  <ITEM label='first'><SP>blablabla...</SP></ITEM>
  <ITEM label='second'><SP>toto is the french foo...</SP></ITEM>
</SECTION>
```

An important entry is the **EXAMPLE** one which is reserved to show scilab uses of your function (begin with simple ones !). Note that you must close this entry with `]]></EXAMPLE>` and not like here with `}}></EXAMPLE>` (once again this is a bad trick to avoid some interpretation problems).

```
<EXAMPLE><![CDATA[
    deff("y=foo(x)","y=x"); // define the foo function as the identity
    foo("toto")
]]></EXAMPLE>
```

This last part explains how to put the links onto others related help pages (as said before the good tags are in fact `<LINK>` and `</LINK>` and not `<ALINK>` and `</ALINK>`) and finally how to reveal your name if you want (use one **AUTHOR_ITEM** entry by author). Perhaps it is a good idea to put an email adress if you look for bug reports !

```
<SEE_ALSO>
  <SEE_ALSO_ITEM> <ALINK>man</ALINK> </SEE_ALSO_ITEM>
  <SEE_ALSO_ITEM> <ALINK>apropos</ALINK> </SEE_ALSO_ITEM>
</SEE_ALSO>

<AUTHOR>
```

```
<AUTHOR_ITEM>B. P.</AUTHOR_ITEM>
</AUTHOR>
</MAN>
```

How to create an help chapter

Create a directory and write down a set of xml files build as described above. Then start Scilab and execute `xmltohtml(dir)`, where `dir` is a character string giving the path of the directory (see `xmltohtml` for more details) .

How to make Scilab know a new help chapter

This can be done by the function `add_help_chapter`.

Examples

```
function y=foo(a,b,c),y=a+2*b+c,endfunction
path=help_skeleton('foo',TMPDIR)
if (isdef('editor') | (funptr('editor')<>0)) then
    editor(path);
end
```

See Also

`apropos` , `help` , `help_skeleton`

Name

manedit — editing a manual item

```
manedit(manitem)
```

Parameters

manitem
character string (usually, name of a function)

Description

`edit(manitem)` opens the xml file associated to `manitem` in the scilab editor.

If there is no xml file associated with `manitem` and `manitem` is the name of a Scilab function, editor opens with the skeleton of the xml file produced by `help_skeleton`. This file is located in `TMPDIR`.

Examples

```
manedit('manedit')

function [x,y,z]=foo123(a,b),
x=a+b,y=a-b,z=a==b
endfunction
manedit foo123
```

See Also

`help` , `help_skeleton`

Name

`%helps` — Variable defining the path of help directories

Description

The global variable `%helps` is an $N \times 2$ matrix of strings. The k th row of `%helps`, `%helps(k, :)` represents the k th chapter of the manual and is made of two strings:

`%helps(k, 1)` is the absolute pathname for a help directory.

`%helps(k, 2)` is a title for this help directory. For instance, for $k=2$, we have the graphics chapter `%helps(2, :)`.

The variable `%helps` is defined in the Scilab startup file `SCI+"/scilab.start"`.

To add a new help directory, the user should add a row to the variable `%helps`. (One row for each directory).

For instance, `%helps=[%helps; "Path-Of-My-Help-Dir", "My-Title"]`; enables the Scilab help browser to look for help manual items in the directory with pathname "Path-Of-My-Help-Dir".

"My-Title" is then the title of a new help chapter.

A valid help directory must contain:

1- A set of `.html` files (e.g. `item1.html`, `item2.html` etc). The `.html` files are usually built from XML files.

2- A `whatis.html` file, which must have a special format. Each row of the `whatis` must be as follows:

```
<BR><A HREF="item.html">item</A> - quick description
```

`item` is the item of the help, i.e. the command `help item` displays the contents of the file `item.html`.

The command `apropos keyword` returns the row(s) of all the `whatis.html` file(s) in which the keyword appears.

On Linux platforms Scilab provides a Makefile for transforming `.xml` pages into `.html` pages (see `SCIDIR/examples/man-examples`).

See Also

`apropos`, `help`, `man`

Name

xmltochm — converts xml Scilab help files to Microsoft Compressed HTML format (Windows)

```
xmltochm(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs

vector of strings: a set of directory paths for which html manuals are to be generated or []

titles

vector of strings: titles associated to directory paths or []

dir_language

vector of strings: languages associated to directory paths or []

default_language

vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into chm files.

Microsoft HTML Help Downloads (Windows) [[http://msdn.microsoft.com/en-us/library/ms669985\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms669985(VS.85).aspx)]

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// =====
my_french_help_dir = my_module_path+'/help/fr_FR';
my_french_help_title = 'Example 1 [fr_FR]';
res = xmltochm(my_french_help_dir,my_french_help_title,'fr_FR');
if MSDOS then
    dos('start ' + res);
end
```

```
// Build the english help
// =====
my_english_help_dir  = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
res = xmltochm(my_english_help_dir,my_english_help_title,'en_US');
if MSDOS then
  dos('start ' + res);
end

// Build the chinese help
// =====
my_chinese_help_dir  = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
res = xmltochm(my_chinese_help_dir,my_chinese_help_title,'zh_TW');
if MSDOS then
  dos('start ' + res);
end
```

See Also

[help](#) , [add_help_chapter](#)

Name

xmltohtml — converts xml Scilab help files to HTML format

```
xmltohtml(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs

vector of strings: a set of directory paths for which html manuals are to be generated or []

titles

vector of strings: titles associated to directory paths or []

dir_language

vector of strings: languages associated to directory paths or []

default_language

vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into HTML files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// =====
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_html_dir    = xmltohtml(my_french_help_dir,my_french_help_title,'fr_FR');

// Build the english help
// =====
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
my_english_html_dir   = xmltohtml(my_english_help_dir,my_english_help_title,'en_US');
```



```
// Build the chinese help
// =====
my_chinese_help_dir  = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_html_dir  = xmltohtml(my_chinese_help_dir,my_chinese_help_title,'zh');
```

See Also

help , add_help_chapter

Name

xmltojar — converts xml Scilab help files to javaHelp format

```
xmltojar(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs

vector of strings: a set of directory paths for which html manuals are to be generated or []

titles

vector of strings: titles associated to directory paths or []

dir_language

vector of strings: languages associated to directory paths or []

default_language

vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into jar files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// =====
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
xmltojar(my_french_help_dir,my_french_help_title,'fr_FR');

// Build the english help
// =====
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
xmltojar(my_english_help_dir,my_english_help_title,'en_US');
```

```
// Build the chinese help
// =====
my_chinese_help_dir  = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
xmltojar(my_chinese_help_dir,my_chinese_help_title,'zh_TW');

// Add french, english or chinese help chapters
// =====

if getlanguage() == 'fr_FR' then
    add_help_chapter(my_french_help_title,my_module_path+"/jar");

elseif getlanguage() == 'zh_TW' then
    add_help_chapter(my_chinese_help_title,my_module_path+"/jar");

else
    add_help_chapter(my_english_help_title,my_module_path+"/jar");
end

// See the result in the help browser
// =====
help();

// Del french and english help chapters
// =====
if getlanguage() == 'fr_FR' then
    del_help_chapter(my_french_help_title);
else
    del_help_chapter(my_english_help_title);
end
```

See Also

help , add_help_chapter

Name

xmltopdf — converts xml Scilab help files to pdf format

```
xmltopdf(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs

vector of strings: a set of directory paths for which pdf manuals are to be generated or []

titles

vector of strings: titles associated to directory paths or []

dir_language

vector of strings: languages associated to directory paths or []

default_language

vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into pdf files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// =====
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_pdf         = xmltopdf(my_french_help_dir,my_french_help_title,'fr_FR')

// Build the english help
// =====
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
my_english_pdf        = xmltopdf(my_english_help_dir,my_english_help_title,'en_US')
```

```
// Build the chinese help
// =====
my_chinese_help_dir  = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_pdf       = xmlltopdf(my_chinese_help_dir,my_chinese_help_title,'zh_')
```

See Also

help , add_help_chapter

Name

xmltops — converts xml Scilab help files to postscript format

```
xmltops(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs

vector of strings: a set of directory paths for which postscript manuals are to be generated or []

titles

vector of strings: titles associated to directory paths or []

dir_language

vector of strings: languages associated to directory paths or []

default_language

vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into ps files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// =====
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_ps          = xmltops(my_french_help_dir,my_french_help_title,'fr_FR')

// Build the english help
// =====
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [EN_US]';
my_english_ps         = xmltops(my_english_help_dir,my_english_help_title,'en_US')
```

```
// Build the chinese help
// =====
my_chinese_help_dir  = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_ps        = xmlltops(my_chinese_help_dir,my_chinese_help_title,'zh_TW');
```

See Also

help , add_help_chapter