Freefem+: Tutorial

F. Hecht, O. Pironneau

December 5, 2000

Contents

1 file = a_tutorial.edp

A remark to start: notice that in this file the program is put between an opening brace { and an ending brace }.
The effect is to force syntax checking of the entire program before starting its execution. Freefem+ is basically an interpreter but a statement between braces is a compound instruction so it is scanned as only one instruction. Thus this forces freefem+ to "compile" the program rather than interpret it.

Consider the problem

$$-\Delta v = 1 \text{ in } \Omega = \{(x,y) \in R^2 : x^2 + y^2 \leq 1\}, \quad v = 0 \text{ on } \Gamma = \partial\Omega.$$

The problem is solved by the finite element method, namely:
Find $u \in V$ the space of continuous piecewise linear functions on a triangulation of Ω which are zero on the boundary $\partial\Omega$ such that

$$\int_\Omega \nabla u \cdot \nabla w = \int_\Omega w \quad \forall w \in V$$

The first thing to do is to prepare the mesh (i.e. the triangulation) ; that is done by first defining the border (the unit circle) and then call the mesh generator (buildmesn) with the right orientation of the border (by definition Ω is on the left side of the oriented Γ).

```
border a(t=0,2*pi){ x = cos(t); y = sin(t)};
mesh disk = buildmesh(a(50));
```

Next `freefem+` will solve the PDE discretized by FEM with the following instruction

```
solve(u)  {
   pde(u)  -laplace(u) = 1;
   on(a)  u=0;
  };
```

Next we can check that the result is correct. Here we display the result first and then display the error field and compute the l^2 error and the H^1 error

```
plot (u);
```

```
plot(u-(1-x^2-y^2)/4);
print("error L2=", sqrt(int()(u-(1-x^2-y^2)/4)^2));
print("error H10=", sqrt(int()(dx(u)-x/2)^2
                    + int()(dy(u)-y/2)^2));
```

For better results we can use mesh adaptation. This module constructs a mesh which fits best a function of V, so u is the main argument of `adaptmesh`. Note that adaptmesh "improves" a mesh, so it requires also the name of a mesh for argument. Therefore mesh adaptation is done in `freefem+` by

```
mesh disk1 = adaptmesh  (disk,u);
```

where `disk1` is a new mesh adapted to u.
To check that this mesh is better we solve the problem again and compute the errors. Notice the improvement!

2 file = adapt.edp

Here we use more systematically the mesh adaptation to track the singularity at an obtuse angle of the domain.
The domain is L-shaped and defined by a set of connecting segments labeled a, b, c, d, e, f.

```
border a(t=0,1){x=t;y=0};
border b(t=0,0.5){x=1;y=t};
border c(t=0,0.5){x=1-t;y=0.5};
border d(t=0.5,1){x=0.5;y=t};
border e(t=0.5,1){x=1-t;y=1};
border f(t=0,1){x=0;y=1-t};
mesh th= buildmesh ("th", a(6) + b(4)
                + c(4) +d(4) + e(4) + f(6));
savemesh("th.msh");
```

Here `buildmesh` has an extra parameter, the character chain "th". Its effect is to create a postscript file named "th.ps" containing the triangulation `th` displayed during the execution of the program. This feature is general to all commands creating screen displays:*those freefem commands which generate a screen display accept a optional character chain as first parameter which ,if present, then produces the creation of a (color) postcript file of the displayed picture*
Note that `savemesh` refers to the current mesh, therefore both keywords below generate a file named `th.msh` but

```
savemesh("th.msh");  // saves current mesh in freefem format
savemesh("th.msh",sh);   //  saves mesh sh in freefem format
```

Now we are going to solve the Laplace equation with Dirichlet boundary conditions 4 times on finer and finer meshes, something like

```
err := 0.1;
for i := 1 to 4  do
{
    solve(u)  {
        pde(u)  -laplace(u) = 1;
        on(a,b,c,d,e,f)  u=0;
    };
    err:=err/2;
    mesh th = adaptmesh  (th,u)err=err;
}
```

after each solve a new mesh adapted to u is computed. To speed up the adaptation we change by hand a default parameter of `adaptmesh: err`, which specifies the required precision, is divided by two at every iteration.

In practice the program is more complex for two reasons

- We must use a dynamic name for files if we want to keep track of all iterations. This is done with the concatenation operator $\sim$. for instance

```
for i := 1 to 4  do
    savemesh("th"~i~".msh",th);
```

saves mesh th four times in files `th1.msh`,`th2.msh`,`th3.msh`, `th3.msh`.

- There many default parameters which can be redefined either throughout the rest of the program or locally within adaptmesh. Here is the list below together with their default value
- In freefem as a whole
- `wait=1` if false (=0) no mouse click are necessary to close the graphics
- `verbosity=1` controls the output (highest is verbose)
- in adaptmesh only
- `abserror=1` if not true then relative error is used

- `nbjacoby=1` number of Jacobi iterations used to smooth the metric.
- `inquire` for queries on the mesh in the display (need to press "f"(forward) to exit the inquiry mode)
- `nbvx=9000` max number of vertices that buildmesh is allowed to generate
- `omega=1` Jacobi surrelaxation parameter
- `ratio=1.8` max allowed ratio of length of two opposite edges (by a vertex)
- `nbsmooth=3` number of times nodes are moved at their optimal position in its Voronoi polygon.
- `splitpbedge=1` if true any internal edge which happens to have its two vertices on the border will be split.
- `maxsubdiv=10` max number of time a triangle is divided.
- `rescaling=1` the function with respect to which the mesh is adapted is rescaled to be between 0 and 1
- `keepbackvertices=1` if true will try to keep as many vertices of the previous mesh as possible. ;
- `cutoff=1e-6` if no abserror then the metric is divided by cutoff + the abs value of the function (useful for boundary layers)
- `anisomax=1e6` the max aspect ratio of triangles
- `err=0.01` relative error level of a posteriori error wished for the function.
- `hmin=0` smallest edge size
- `hmax = diam(`Ω`/3` largest edge size
- `errg=0.01` relative error between the discrete border and the continuous one
- `ismetric=1` if =3 the metric is given by hand so 3 functions defining a symmetric matrix field are needed if =0 then a function is given and its hessian is computed to define a metric, if =1 then isotropic mesh size is given directly at every point through a function.

3 file = blackScholes.edp

3.1 Two-dimensional Black-Scholes equation

In mathematical finance, an option on two assets is modeled by a Black-Scholes equations in two space variables, (see for example Wilmott's book : a student introduction to mathematical finance, Cambridge University Press).

$$\begin{aligned}\partial_t u &+ \frac{(\sigma_1 x_1)^2}{2}\frac{\partial^2 u}{\partial x_1^2} + \frac{(\sigma_2 x_2)^2}{2}\frac{\partial^2 u}{\partial x_2^2} \\ &+ \rho x_1 x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2} + rS_1\frac{\partial u}{\partial x_1} + rS_2\frac{\partial u}{\partial x_2} - rP = 0 \end{aligned} \tag{1}$$

which is to be integrated in $(0,T)\times R^+\times R^+$ subject to, in the case of a put

$$u(x_1,x_2,T) = (K - \max(x_1,x_2))^+. \tag{2}$$

Boundary conditions for this problem may not be so easy to device.
As in the one dimensional case the PDE contains boundary conditions on the axis $x_1 = 0$ and on the axis $x_2 = 0$, namely two one dimensional Black-Scholes equations driven respectively by the data $u(0,+\infty,T)$ and $u(+\infty,0,T)$. These will be automatically accounted for because they are embedded in the PDE. So if we do nothing in the variational form (i.e. if we take a Neuman boundary condition at these two axis in the strong form) there will be no disturbance to these.
At infinity in one of the variable, as in 1D, it makes sense to match the final condition:

$$u(x_1,x_2,t) \approx (K - \max(x_1,x_2))^+ e^{r(T-t)} \quad \text{when} \quad |x| \to \infty. \tag{3}$$

For an American put we will also have the constraint

$$u(x_1,x_2,t) \geq (K - \max(x_1,x_2))^+ e^{r(T-t)}. \tag{4}$$

We take

$$\sigma_1 = 0.3, \quad \sigma_2 = 0.3, \quad \rho = 0.3, \quad r = 0.05, \quad K = 40, \quad T = 0.5 \tag{5}$$

An implicit Euler scheme with projection is used and a mesh adaptation is done every 10 time steps. The first order terms are treated by the Characteristic Galerkin method, which, roughly, approximates

$$\frac{\partial u}{\partial t} + a_1\frac{\partial u}{\partial x} + a_2\frac{\partial u}{\partial y} \approx \frac{1}{\delta t}(u^{n+1}(x) - u^n(x - \vec{a}\delta t)) \tag{6}$$

The listing of the freefem program is given below. The program is self-explanatory and gives all the numerical values needed to reproduce the test.

```
m:=20; L:=80; LL:=80;
border aa(t=0,L){x=t;y=0};
border bb(t=0,LL){x=L;y=t};
border cc(t=L,0){x=t ;y=LL};
border dd(t=LL,0){x = 0; y = t};
mesh th =buildmesh(aa(m)+bb(m)+cc(m)+dd(m));
sigmax:=0.3; sigmay:=0.3; rho:=0.3; r:=0.05;
```

```
K=40; dt:=0.02;
f = max(K-max(x,y),0);
femp1(th) u=f;
femp1(th) xveloc = -x*r+x*sigmax^2+x*rho*sigmax*sigmay/2;
femp1(th) yveloc = -y*r+y*sigmay^2+y*rho*sigmax*sigmay/2;
j:=0;
for n=0 to 0.5/dt do
{
       solve(th,u) with AA(j){
           pde(u) u*(r+1/dt)
     - dxx(u)*(x*sigmax)^2/2 -dyy(u)*(y*sigmay)^2/2
     - dxy(u)*rho*sigmax*sigmay*x*y/2
     - dyx(u)*rho*sigmax*sigmay*x*y
     = convect(th,xveloc,yveloc,dt,u)/dt;
   on(aa,dd) dnu(u)=0;
   on(bb,cc) u = f;
};
u = max(u,f); plot("uf",th, u-f);
if(j==10) then {
    mesh th = adaptmesh("th",th,u);
femp1(th) xveloc = -x*r+x*sigmax^2+x*rho*sigmax*sigmay/2;
femp1(th) yveloc = -y*r+y*sigmay^2+y*rho*sigmax*sigmay/2;
femp1(th) u=u;
        wait:=0;   j:=-1;
        }; j=j+1;
};
```

For more details see the article in the file BlackScholEastWest.pdf

4 file = cavity.edp

See also section 3.3 of the documentation of freefem+

The driven cavity flow problem is solved first at zero Reynolds number (Stokes flow) and then at Reynolds 100. The velocity pressure formulation is used first and then the calculation is repeated with the stream function vorticity formulation.

Stokes flow is modeled by

$$-\Delta u + \nabla p = 0, \quad \nabla \cdot u = 0 \quad \text{in } \Omega$$

and the boundary conditions for the driven cavity problem is $u \cdot n = 0$ and $u \cdot s = 1$ on the top boundary and zero elsewhere.
The mesh is constructed by

```
wait:=0;
border a(t=0,1){ x=t; y=0};  //  the unit square
border b(t=0,1){ x=1; y=t};
border c(t=1,0){ x=t; y=1};
border d(t=1,0){ x=0; y=t};
n:=20;
mesh th= buildmesh(a(n)+b(n)+c(n)+d(n));
```

Notice that we set `wait:=0` to avoid clicking in the graph window all the time
The Stokes problem is implemented as a system solve for the velocity (u, v) and the pressure p:

```
solve(u,v,p){
    pde(u)  - laplace(u) + dx(p) = 0;
        on(a,b,d) u =0;
        on(c) u = 1;
    pde(v) - laplace(v) + dy(p) = 0 ;
        on(a,b,c,d) v=0;
    pde(p) p*0.001- laplace(p)*0.001 + dx(u)+dy(v) = 0;
        on(a,b,c,d) dnu(p)=0;
    };
```

. Each PDE has its own boundary conditions. There is some arbitrary decision there which will effect the condition of the linear system. Basically Dirichlet data should be associated to the corresponding PDE so that the penalization is done on the diagonal of the matrix of the underlying discrete linear system.
Notice the term `p*0.001- laplace(p)*0.001` which is a regularization term, necessary here because the finite element P1-P1 for velocity-pressure does not satisfy the LBB condition.

Next the streamlines are computed by finding ψ such that rot$\psi = u$ or better, $-\Delta\psi =$rot$u$,

```
solve(psi){ pde(psi) -laplace(psi) = dy(u)-dx(v);
            on(a,b,c,d) psi=0};
```

Now the Navier-Stokes equations are solved

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0$$

with the same boundary conditions and initial conditions $u = 0$. This is implement by using the convection operator `convect` for the term $\frac{\partial u}{\partial t} + u \cdot \nabla u$, giving a discretization in time

$$\frac{1}{\delta t}(u^{n+1} - u^n o X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \quad \nabla \cdot u^{n+1} = 0$$

The term,$u^n o X^n(x) \approx u^n(x - u^n(x)\delta t)$ will be computed by the operator "convect", so we obtain

```
nu:=0.01; dt :=0.1;
for i=0 to 20 do
{
solve(u,v,p) with B(i){
    pde(u)  u/dt- laplace(u)*nu + dx(p) = convect(u,v,dt,u)/dt;
        on(a,b,d) u =0;
        on(c) u = 1;
    pde(v) v/dt- laplace(v)*nu + dy(p) =  convect(u,v,dt,v)/dt;
        on(a,b,c,d) v=0;
    pde(p) p*0.1*dt - laplace(p)*0.1*dt + dx(u)+dy(v) = 0;
        on(a,b,c,d) dnu(p)=0;
    };
};
```

Notice that the matrices are reused (keyword `with`)

Now the same problem can be solved by the Stream function with vorticity $\omega = \text{rot} u$, as

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega - \nu \Delta \omega = 0$$

giving

```
femp1 psi = 0;                    //  stream function
femp1 om = 0;                     //  vorticity
for i=0 to 20 do
{
    femp1 u = dy(psi);            //  velocity
    femp1 v = -dx(psi);
    solve(psi,om) with D(i){
        pde(psi)  om -laplace(psi) = 0;
        on(a,b,d) dnu(psi)=0;
        on(c) dnu(psi) = 1;
        pde(om) om/dt - laplace(om)*nu = convect(u,v,dt,om)/dt;
        on(a,b,c,d) dnu(om) + psi*1e8 = 0;  //  a trick to im-
pose psi = 0
    };
plot(om);
};
```

5 file = convect.edp

See section 2.4 of the documentation of freefem+

Freefem+ implements the Characteristic-Galerkin method for convection operators. Recall that the equation

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u = f$$

can be discretized as

$$\frac{Du}{Dt} = f \quad \text{i.e.} \quad \frac{du}{dt}(X(t),t) = f(X(t),t) \quad \text{where} \quad \frac{dX}{dt}(t) = \vec{v}(X(t),t)$$

and where D is the total derivative operator. So a good scheme is

$$\frac{1}{\delta t}(u^{m+1}(x) - u^m(X^m(x))) = f^m(x)$$

where $X^m(x)$ is an approximation of the solution at $t = m\delta t$ of

$$\frac{dX}{dt}(t) = \vec{v}(X(t),t), \quad X((m+1)\delta t) = x.$$

In freefem+ this is implemented by

```
for i=1 to 1/dt do
{    u = convect(v,vx,vy,dt)*dt + f*dt;
     v = u;
 }
```

where $\vec{v} = (vx, vy)^T$.

It is strange that one cannot write

```
u = convect(u,vx,vy,dt)*dt + f*dt;
```

but this is because "convect" is a **nonlocal** operator. To compute the value $u[i]$ of u at vertex i we need the values of u at all neighboring vertices. Now equalities between functions in freefem are implemented by a do loop so that $u[i]$ is computed and stored thereby overwriting the old value of $u[i]$.

5.1 The Rotating Hill

The rotating hill problem is the convection of a hump type initial condition by a solid rotation velocity; this at all time the computed solution should be equal to the initial condition rotated around the origin.

```
border a(t=0,2*pi){ x := cos(t); y := sin(t)};
mesh th = buildmesh(a(50));
femp1 v = exp(-10*((x-0.3)^2 +(y-0.3)^2));
dt := 0.17;
femp1 u1 = y;
femp1 u2 = -x;
for i=0 to 10 do {
      convect(u1,u2,dt,f,v);
      v=f;
};
```

Note that this form of the operator convect is not the same as the one used before. Here the interpolation is more precise.

6 file = fictitiousdomain.edp

This example is somewhat similar to the one in "jump.edp" in that it solves a problem involving a boundary which is not part of the triangulation.
Here we solve a Neumann problem

$$u - \Delta u = 0 \text{ in } \Omega \quad \frac{\partial u}{\partial n} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot n$$

which has an analytical solution $u = x + y$. The domain is a circle but it is embedded in a greater square and the fictitious domain method is used, namely:
Find $u \in H_0^1(D)$ such that

$$\int_D ((1_\Omega + \epsilon)(uw + \nabla u \cdot \nabla w)) = \int_\Gamma (n_x + n_y)w$$

where ϵ is a regularization parameter.
This gives

```
border a(t=-1,1){ x=t; y=-1};    //  the unit square
border b(t=-1,1){ x=1; y=t};
```

```
border c(t=1,-1){ x=t; y=1};
border d(t=1,-1){ x=-1; y=t};
border circle(t=0,2*pi) { x=cos(t)/2;y=sin(t)/2;}

Chi= (x*x+y*y)<0.25;

for i:=1 to 3 do {
  n = i*10;
  mesh Ch= buildmesh(circle(pi*n/5));
  mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)); // +circle(pi*n));
  varsolve(Th) A(U,V) with {
     A= int()( (Chi+0.001)*(U*V + dx(U)*dx(V)+dy(U)*dy(V) ) )
      +   on(Th,a,b,c,d)(V)(U=0)
      -   int(Ch,circle)(2*(x+y)*V)
  };
  append("err.txt",n,sqrt(int()(Chi*(U-x-y)^2)));
};
save("Ufd.wrl",U); // 3D plot viewed with Int Explorer/Netscape
plot(Chi*(U-x-y)); // error
```

Note that "append" writes at the end of the file if ever it contains something. It is easy after that to call `gnuplot` to display in log-log scale the error decrease in terms of h (i.e. 1/n).

7 file = fitmesh.edp

This illustrates the flexibility of "adaptmesh".
The domain is a circle with a circular hole

```
border a(t=0,pi*2){ x = cos(t); y = sin(t)};
border b(t=0,2*pi){ x = 0.3 + 0.3*cos(t);  y = 0.3*sin(t) };
mesh th = buildmesh(a(40) + b(-20)) ;
```

Two "crazy" functions are chosen:

```
sy = (10*x*x*x+y*y*y) + atan(100*(sin(5*y)-2*x));
s = (x*x*x+10*y*y*y) + atan(10*(sin(5*x)-4*y));
```

Then the mesh is adapted to the two functions in 4 iterations:

```
for i= 1 to 4 do
{
```

```
      mesh th = adaptmesh (th,s,sy) verbosity=4,
          err=0.01, hmax=2, hmin=0.00005, nbvx=10000, omega=1.8,
nbsmooth=2,
          splitpbedge=0., maxsubdiv=5 ;
      plot(s());
      plot(sy());
  };
```

The parameters of adaptmesh have been described in section 1. What is important here is to notice that the adpativity is requested with respect to *two* functions.

8 file = fluidstruct.edp

This is a variation of the example described in section 11.5 of freefem+ documentation.

A beam sits on a box full of fluid rotating because the left vertical side has velocity one. The beam is bent by its own weight, but the pressure of the fluid modifies the bending.
The bending displacement of the beam is given by (uu,vv) solution of

```
border a(t=2,0)  { x=0; y=t };          //   left beam
border b(t=0,10) { x=t; y=0 };          //   bottom of beam
border c(t=0,2)  { x=10; y=t };         //   rigth beam
border d(t=0,10) { x=10-t; y=2 };       //   top beam

E := 21.5;
sigma := 0.29;
mu := E/(2*(1+sigma));
lambda := E*sigma/((1+sigma)*(1-2*sigma));
gravity := -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
varsolve(th,0) bb(uu,w,vv,s) with {
    e11 = dx(uu);   e22 = dy(vv);   e12 = (dx(vv)+dy(uu))/2;
    w11 = dx(w);    w22 = dy(s);    w12 = (dx(s)+dy(w))/2;
        bb = int()( 2*mu*(e11*w11+e12*w12+e22*w22)
                + lambda*(e11+e22)*(w11+w22)/2 -gravity*s)
                + on(a,c)(w)(uu=0) + on(a,c)(s)(vv=0)
     };
```

Then Stokes equations for fluids at low speed is solved in the box below the beam but the beam has deformed the box (see border h):

```
   border e(t=0,10) { x=t; y=-10 };       //   bottom
   border f(t=0,10) { x=10; y=-10+t };    //   right
   border g(t=0,10) { x=0; y=-t };        //   left
   border h(t=0,10) { x=t; y=vv(t,0)*( t>=0.001 )*(t <= 9.999);};
   mesh sh = buildmesh(h(-20)+f(15)+e(15)+g(15));

      solve(U,V,P){
       pde(U)  - laplace(U) + dx(P) = 0; on(e,f,g,h) U =0;
       pde(V)  - laplace(V) + dy(P) = 0 ; on(e,f,h) V=0; on(g) V=(-
y)*(10+y)/25;
       pde(P)  P*0.001- laplace(P)*0.001 + dx(U)+dy(V) = 0;
           on(e,f,g,h) dnu(P)=0;
       };
      P = P - int(b)(P)/10;
```

Now the beam will feel the stress constraint from the fluid:

```
   femp1  sigma11(x+uu,y+vv) = (2*dx(U)-P);
   femp1  sigma22(x+uu,y+vv) = (2*dy(V)-P);
   femp1  sigma12(x+uu,y+vv) = (dx(V)+dy(U));
```

which comes as a boundary condition to the PDE of the beam:

```
   varsolve(1) bb(uu,w,vv,s) with {
         bb = int()(  2*mu*(e11*w11+e12*w12+e22*w22)
            + lambda*(e11+e22)*(w11+w22)/2 -gravity*s )
            - coef*int(b)(sigma11 *nrmlx*w + sigma22*nrmly*s
            + sigma12*(nrmly*w + nrmlx*s))
            + on(a,c)(w)(uu=0) + on(a,c)(s)(vv=0)};
      err = sqrt(int()( (uu-uu1)^2 + (vv-vv1)^2 ));
```

Notice that the matrix generated by bb is reused. Finally we deform the beam

```
      mesh th = movemesh(th, x+uu, y+vv);
```

9 file = jump.edp

Here we wish to solve the problem ($[f]|_\gamma$ denotes the jump of f across γ)

$$-\Delta u = f \text{ in } \Omega \quad [u] = g \text{ on } \gamma \quad [\frac{\partial u}{\partial n} = 0] \text{ on } \gamma \quad u = 0 \text{ on } \Gamma$$

where $\Gamma = \partial\Omega$ and γ is a closed curve strictly inside Ω. The variational formulation of this problem is

$$\int_\Omega \nabla u \cdot \nabla w = \int_\Omega fw - \int_\gamma w \quad \forall w \in H_0^1(\Omega)$$

In this example, $f = 1, g = 1$, Ω is the unit square and γ is the circle of radius 0.5 centered at the origin, in the middle of the square.
In many applications it is not possible to locate γ beforehand and so it is not part of the triangulation (it could move while the background mesh should stay fixed).
Freefem can handle both cases. First when γ is part of the triangulation:

```
border a(t=-1,1){ x=t; y=-1};    //  the unit square
border b(t=-1,1){ x=1; y=t};
border c(t=1,-1){ x=t; y=1};
border d(t=1,-1){ x=-1; y=t};
border circle(t=0,2*pi) { x=cos(t)/2;y=sin(t)/2;}
n = 40;
Chi= (x*x+y*y)<0.25;

//  The circle is part of the mesh
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n)+circle(pi*n));
varsolve(Th) A(U,V) with {
 A=       int()( dx(U)*dx(V)+dy(U)*dy(V) - Chi*V)
      +   on(Th,a,b,c,d)(V)(U=0)
      +   int(circle)( nrmlx*dx(V) + nrmly*dy(V) )
};
```

Then when it is not part of the triangulation

```
//  Ch is needed only to fix nb of points on Circle
mesh Ch= buildmesh(circle(pi*n/5));
mesh Th= buildmesh(a(n)+b(n)+c(n)+d(n));
varsolve(Th) A(U,V) with {
 A=       int()( dx(U)*dx(V)+dy(U)*dy(V) - Chi*V)
      +   on(Th,a,b,c,d)(V)(U=0)
      +   int(Ch,circle)( nrmlx*dx(V) + nrmly*dy(V) )
};
```

However not that there is some restriction, namely that the circle must be divided into segments for integration purpose and the only way to do that is to build a triangulation with it. The algorithm is not optimal and so it is slow. In both cases the normal is the outer normal with respect to the orientation of γ.

We can check if mesh adaptation works as follows:

```
mesh Th= adaptmesh(Th,U);
varsolve(Th) A(U,V) with {
 A=       int()( dx(U)*dx(V)+dy(U)*dy(V) - Chi*V)
      +   on(Th,a,b,c,d)(V)(U=0)
      +   int(Ch,circle)( nrmlx*dx(V) + nrmly*dy(V) )
};
plot(U);
save("U.wrl",U);
```

Notice the last command. It builds a 3D plot of u which can be viewed in all angles with a vrml engine such as present in recent versions of Internet Explorer or Netscape (else download the plugin from Cosmo for instance).

10 file = naca.edp

A NACA0012 airfoil is considered, at the center of a large circle:

```
border a(t=0,2*pi) { x=5*cos(t);  y=5*sin(t) };  //  approximates
infinity

border b(t=0,2){                                    // Profil S1
                  if((t<0.001)or(t>1.999)){x=0; y=0.} else
                  if(t<=1) then {x=t;
                                  y=0.17735*sqrt(t)-0.075597*t
-0.212836*(t^2)+0.17363*(t^3)-0.062534*(t^4);}
                           else if(t>1.0001){x=2-t;
                                  y=-(0.17735*sqrt(2-t)-0.075597*(2-
t)
 -0.212836*((2-t)^2)+0.17363*((2-t)^3)-0.06254*((2-t)^4))}
 };
mesh th = buildmesh(a(30)+b(70));
```

It is tricky to get the geometry right because of the round off errors. Here we had to consider the leading edge separately otherwise the profile wasn't closing properly.

Next we solve potential flow for two values of the circulation, 0 and 1. First zero:

```
solve (psi0) with A(0){
    pde(psi0) -laplace(psi0) = 0;
    on(a)psi0=y-0.1*x;    //  10 percent lift
    on(b)psi0= 0;
};
```

then one:

```
solve(psi1) with A(1){
    pde(psi1) -laplace(psi1)= 0;
    on(a) psi1 = 0;
    on(b) psi1 = 1;
};
```

The general solution is then psi0 + beta psi1 and beta is determined by writing that the pressure is continuous at the trailing edge (x,y)=(1,0):

```
beta := psi0(0.99,0.01)+psi0(0.99,-0.01);
beta := -beta / (psi1(0.99,0.01)+ psi1(0.99,-0.01)-2);
```

The pressure cp is computed by

```
femp1 psi = beta*psi1+psi0;
plot(psi);
femp1 cp = -dx(psi)^2 - dy(psi)^2;
plot(cp);
```

Notice that some precision is lost on cp due to the fact that it is obtained from the derivatives of the stream function ψ and so it should be a P0 function, but the plot routine of freefem does not plot P0 functions.

11 file = optdes.edp

An optimal shape design is solved. The solution is know: it is a rectangle. The state equation of the system is potential flow p and at the solution it should be $p = x$. So we set to

$$\min_{\Omega}\{\int_D (p-x)^2 \ : \ -\Delta p = 0 \text{ in } \Omega, \quad + b.c.\}$$

The boundary conditions are Dirichlet ($p = x$) on the vertical boundaries and homogeneous Neumann on the horizontal (or quasi-horizontal) walls. The set D is the part of Ω where $y < 0.25$. The initial geometry Ω is a rectangle with a hump:

```
border h(t=0,3){x=t;y=0.25};  // 0
border a(t=1,0){x=t; y=0.5}; // 1
border b(t=3,2){x=t;y=0.5};
border e(t=0,3){x=t;y=0};
border c(t=0,0.25){x=3;y=t}; // 3
border d(t=0.5,1){x=3;y=0.5*t};
border f(t=0.5,0.25){x=0;y=t};
border g(t=0.25,0){x=0;y=t};
border i(t=2,1){x=t;y=0.5+(t-1)*(2-t)}; // 2
mesh th=buildmesh(a(20)+b(20)+c(10)+d(10)
            +e(60)+f(10)+g(10)+h(60)+i(20));
```

The state equation is solved by

```
solve(p) {
 pde(p)  p*0.0001 - laplace(p) = 1;
 on(c,d,f,g)  p=x;
};
```

there is a small regularization because the rectangle is long and there are too many walls with Neumann conditions.
The gradient of the function with respect to shape deformation must be computed. This requires an adjoint q:

```
solve(q){
 pde(q) q *0.0001 - laplace(q) = 20*(y<0.25)*(p-x);
 on(c,d,f,g)  q = 0;
};
```

and then , with respect to vertical variations of boundary Ω, the gradient of the functional which is minimized is $J' = n_x \nabla p \cdot \nabla q$. So in principle iterations which move the top boundary proportionally to J' should work. But then oscillations appear and so a smoother is needed. It is implemented as follows:

```
solve(v){
pde(v) -laplace(v) = 0;
on(a,b,e,c,d,f,g) v =0;
on(i) v = nrmlx*(dx(p)*dx(q)+dy(p)*dy(q));
};

solve(v)
```

```
{ pde(v) -laplace(v) = v;
    on(a,b,e,c,d,f,g) v =0;
};
```

And now we move the top boundary proportionally to v:

```
mesh th = movemesh(th,x,y - 0.4*(y>0.25)*v*(y-0.25));
};
```

12 file = optshape.edp

The object of this exercise is to recover a shape from the value of the PDE in a domain Z. The shape is a disk but its center (xcent,ycent) is not known. The disk moves in a domain which is the union of a rectangle and a circle, given by

```
wait:=1;    xcent:=2.5; ycent:=0.5; r:=0.2; L=4;
border a(t=0,1){x=t;y=1-t};
border a1(t=1,L){x=t;y=0};
border b(t=0,1){x=L;y=t};
border c(t=L,0){x=t ;y=1};
border e(t=0, pi/2){ x= cos(t); y = sin(t)};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t)};
border f(t=0,2*pi){ x =xcent+r*cos(t); y=ycent+r*sin(t);};
n:=1;
mesh sh = buildmesh(a1(15*n) + b(5*n) + c(20*n) +e1(25*n)+f(-15*n));
```

By solving the PDE for a given position of the disk we find a target function u_d:

```
D = (x+xcent)^2+(y+ycent)^2<=0.3;
solve(ud) with B(0)
     {  pde(ud) -laplace(ud)=10*D;
        on(e1,f)ud=0;on(b)ud=1; };
```

As can be seen the PDE is a Laplace equation with mixed Dirichlet-Neumann data and source term in D. So the problem now is to recover $(xcent, ycent)$ by minimizing the norm of $u - u_d$ in Z, the complementary of the unit disk in the circular part of the domain. The problem is solved by a gradient method. It is complicated by the fact that we start with a course mesh and make it finer if the gradient method stalls.

```
    C =(x^2+y^2 <= 1);
   ibb:=0; xcent:=1.; ycent:=0.3; eps:=0.1; Noptim:= 7; opteps :=
1; cout := 1;

   for ioptim:=0 to Noptim do
    {
       if(cout < opteps)then{ opteps := opteps/100; n := n+1};
       mesh th = buildmesh(a1(7*n) + b(2*n) + c(10*n) +e1(12*n)+f(-
7*n));
       D1 = ((x+xcent)^2+(y+ycent)^2<=0.3); C1 = (x^2+y^2 <= 1);

       solve(u)
        {  pde(u) -laplace(u)=10*D1; on(e1,f)u=0; on(b)u=1 };

       ibb:=ibb+1; plot("ub"~ibb,u);
       cout := int(th)((x^2+y^2>1)*(u-ud)^2) ;
       coutb:= int(f)(u^2+(dx(u)*nrmlx)^2+(dy(u)*nrmly)^2);

   //  don't compute if last iteration
       if(ioptim <= Noptim-1) then {
           solve(th,p)
               {   pde(p) -laplace(p)=2*(u-ud); on(e1,f)p=0; };

   dxcent = -int(f)(((u-ud)^2+dx(p)*dx(u)+dy(p)*dy(u))*(x-xcent)/0.09);
   dycent = -int(f)(((u-ud)^2+dx(p)*dx(u)+dy(p)*dy(u))*(y-ycent)/0.09);
           xcent:=xcent-dxcent;  //  gradient method
           ycent:=ycent-dycent;
      };
    };
```

This program was written in cooperation with B. Mohammadi. For further details on optimal shape design the reader is referred to our book *Applied Optimal Shape Design* Oxford University Press, 2000.

13 file = readmesh.edp

Freefem can read and write files which can be reused once read but the names of the borders are lost and they have to be replaced by the number which corresponds to their order of appearance in the program, unless the number is forced by the keyword "label".

```
   border floor(t=0,1){ x=t; y=0};  //  the unit square
   border right(t=0,1){ x=1; y=t; label=5};
   border ceiling(t=1,0){ x=t; y=1; label=5};
```

```
border left(t=1,0){ x=0; y=t; label=5};
n:=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh("toto.am_fmt"); //  format "formated Marrocco"
savemesh("toto.Th"); // format database "bamg"
savemesh("toto.dbg"); // format debug
savemesh("toto.msh");  // format freefem
mesh th2 = readmesh("toto.msh");
save("f.txt",f);
read("f.txt",g);
plot(g);
solve(u){
  pde(u) u+laplace(u) = g;
  on(1) u=0;
  on(5) dnu(u)=1;
};
plot (th2,u);
```

There are many formats of mesh files available for communications with other tools such as emc2, modulef..., the suffix gives the chosen type. More details can be found in the article by F. Hecht "bamg : a bidimentional anisotropic mesh generator" (INRIA report 1999) available in the freefem web page.
Note also the wrong sign in the Laplace equation, but freefem can handle it as long as it is not a resonance mode (i.e. the matrix of the linear system should be non-singular).

14 file = region.edp

This example explains the definition and manipulation of *region*, i.e. subdomains of the whole domain.

Consider this L-shaped domain with 3 diagonals as internal boundaries, defining 4 subdomains:

```
border a(t=0,1){x=t;y=0};
border b(t=0,0.5){x=1;y=t};
border c(t=0,0.5){x=1-t;y=0.5};
border d(t=0.5,1){x=0.5;y=t};
border e(t=0.5,1){x=1-t;y=1};
border f(t=0,1){x=0;y=1-t};
//   internal boundary
border i1(t=0,0.5){x=t;y=1-t};
border i2(t=0,0.5){x=t;y=t};
border i3(t=0,0.5){x=1-t;y=t};
mesh th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) +
    f(6)+i1(6)+i2(6)+i3(6));
```

```
plotp0(region);
```

"region" is a keyword of freefem+ which is in fact a function and can be accessed by `region(x,y)`. The value returned is the number of the subdomain. This number is defined by "buildmesh" which scans while building the mesh all its connected component. So to get the number of a region containing a particular point one does:

```
nupper:=region(0.4,0.9);
nlower:=region(0.9,0.1);
print(nlower,nupper);
```

This is particularly useful to define discontinuous functions such as might occur when one part of the domain is copper and the other one is iron, for example.
We this in mind we proceed to solve a Laplace equation with discontinuous coefficients (ν is 1, 6 and 11 below).

```
nu=1+5*(region==nlower) + 10*(region==nupper);
plotp0(nu);
solve(u){ pde(u) -laplace(u)*nu = 1; on(a,b,c,d,e,f) u=0};
plot(u);
```

15 file = schwarz.edp

To solve

$$-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,$$

the Schwarz algorithm for domain decomposition runs like this

$$-\Delta u_1^{m+1} = f \text{ in } \Omega_1 \quad u_1^{m+1}|_{\Gamma_1} = u_2^m$$

$$-\Delta u_2^{m+1} = f \text{ in } \Omega_2 \quad u_2^{m+1}|_{\Gamma_2} = u_1^m$$

where Γ_i is the boundary of Ω_i and on the condition that $\Omega_1 \cap \Omega_2 \neq \emptyset$ and that u_i are zero at iteration 1.

Here we take Ω_1 to be a quadrangle, Ω_2 a disk and we apply the algorithm starting from zero.

```
border a(t=1,2){x=t;y=0};
border b(t=0,1){x=2;y=t};
border c(t=2,0){x=t ;y=1};
border d(t=1,0){x = 1-t; y = t};
border e(t=0, pi/2){ x= cos(t); y = sin(t)};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t)};
n:=4;
mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh ( e(5*n) + e1(25*n) );

femp1(th) u = 0;
for i=0 to 10 do
{
    solve(TH,U) with A(i){pde(U) -laplace(U) = 1;
              on(e) U = u;    on(e1) U = 0  };

    solve(th,u) with B(i){pde(u)    -laplace(u) = 1;
              on(d) u = U; on(a,b,c) u = 0};

    plot(TH,U,th,u);
};
```

Notice that the PDEs have each their own mesh and that the matrices are stored factorized and reused. Notice also how both graphics are superposed.

16 file = subroutine.edp

There is a (very) limited possibility for avoiding code repetition in freefem. For example, if we declare

```
border a(t=0, 2*pi){ x := cos(t); y := sin(t)};
mesh th = buildmesh(a(70));
femp1 v = sin(x);

subroutine cout(ro){
    v = 2*v+ro;
    cout = int()(v^2*cos(x));
};
```

Then to avoid code duplication, we can write for instance

```
print(cout(0),cout(1), cout(2));
```

17 file = turekstep.edp

See also paragraph 10.2 in the documentation of freefem.

The Navier-Stokes equations

$$\partial_t \vec{u} + \vec{u} \cdot \nabla \vec{u} - \nu \Delta \vec{u} + \nabla p = 0, \quad \nabla \cdot \vec{u} = 0$$

are approximated in time by

$$\frac{1}{\delta t}(u^{n+1} - u^n o X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \quad \nabla \cdot u^{n+1} = 0$$

The term,$u^n o X^n(x) \approx u^n(x - u^n(x)\delta t)$ will be computed by the operator "convect", so we obtain

```
nu:=0.01; dt :=0.1;
for i=0 to 20 do
{
solve(u,v,p) with B(i){
    pde(u)  u/dt- laplace(u)*nu + dx(p) = convect(u,v,dt,u)/dt;
        on(a,b,d) u =0;
        on(c) u = 1;
    pde(v) v/dt- laplace(v)*nu + dy(p) =  convect(u,v,dt,v)/dt;
        on(a,b,c,d) v=0;
    pde(p) p*0.1*dt - laplace(p)*0.1*dt + dx(u)+dy(v) = 0;
        on(a,b,c,d) dnu(p)=0;
    };
};
```

Notice that the first time solve occurs it has B(0) so the matrix is built and factorized. The second time it has B(1) so the matrix is reused.

The flow in an expanding pipe is studied. It is again the Navier-Stokes The Reynolds number based on the size of the step an the mean inflow is Re = 200. The projection algorithm is used as in Rannacher-Turek (featflow).
The boundary conditions at the outflow boundary has been chosen so as to give optimal performance, among the one allowed by the physics of the problem.

18 file = verifs.edp

This file and the 2 others that follow compare the computed solution with the analytical solution in odd cases where all the options of the solver are used. There are several examples. Let us take one with exact solution $ue = sin(x+y)$. First we define a number of constants

```
visc := 2.2;   //  coef of PDE, taken as constants
dis  := 3.3;
rob  :=  1.1;
aa   :=  1.3;
vxx  :=  1;
vyy  :=  2;
vxy  :=  3;
vyx  :=  4;
nnx() = -x;   //  component of normal to border b
nny() = -y;
```

Then we compute the solution, its derivative, its normal derivative

```
ue()= sin(x+y);                    //  analytical solution
dxue() = cos(x+y);                   //  x derivative of ue
dnuue()=dxue()*(visc*(nnx+nny) + (vxy+vxx)*nnx + (vyx+vyy)*nny);
```

And now we compute the data of the pde so that it gives ue for solution:

```
neu() = ue()*rob+dnuue();
rhs() = dis*ue + ue * (2 * visc + vxx + vyy +vxy + vyx) + aa *
cos(x+y);
```

And then solve the PDE

```
solve(th1,u) {
  pde(u) u*dis + dx(u)*aa - laplace(u)*visc
    -dxx(u)*vxx-dyy(u)*vyy-dxy(u)*vxy-dyx(u)*vyx = rhs;
  on(a)  u=ue;
  on(b) dnu(u) + u*rob = neu
  };
```

and finally plot the result and the error

```
plot(u);
plot(th1,u-ue);
```

```
print( int(th1)((u-ue)^2));
```

The same can be sone on a finer mesh to check that the error decreases.

19 file = verifss.edp

Same thing as file verifs.edp but for a 2-system

20 file = verifvs.edp

Same as above but when using `varsolve`.

Index