4 Visualización Gráfica de Datos

La visualización gráfica de datos constituye una disciplina propia dentro del universo de las Ciencias e Ingenierías y en particular en la Ciencia de Datos. Esta práctica ha marcado hitos importantes a lo largo de la historia en la analítica de datos. Desde el gráfico más sencillo hasta el más complejo y refinado, todos ofrecen alto valor tanto al analista, durante su proceso de ciencia de datos, como al usuario final, al cual estamos comunicando una historia basada en datos.

¿Por qué es tan importante la visualización gráfica de los datos? En la Ciencia de Datos existen muchos tipos diferentes de datos para analizar. Una forma de clasificación de los datos atiende al nivel de estructuración lógica que éstos tienen. Por ejemplo, los datos en formatos similares a hojas de cálculo -aquellos datos que se estructuran en forma de filas y columnas- son datos con una estructura bien definida -o datos estructurados- Sin embargo, aquellos datos como los 140 caracteres de un Feed de X (Twitter) se consideran datos sin estructura -o desestructurados-.

En medio de estos dos extremos se encuentra toda una gama de grises, que va desde los ficheros delimitados por caracteres especiales (comas, puntos y comas, espacios, etc.) hasta las imágenes o los videos de Youtube. Es evidente que las imágenes y los vídeos solamente cobran sentido humano una vez representadas visualmente. De nada serviría (para un humano) que presentaremos una imagen como una matriz de números que representan una combinación de colores RGB (Red, Green, Blue). En el caso de los datos estructurados, su representación gráfica es necesaria en todas las etapas del proceso de análisis, desde la etapa exploratoria, hasta la presentación final de resultados.

La visualización gráfica de los datos tiene un papel fundamental en todos los estadios del análisis de datos. Existen múltiples aproximaciones sobre cómo realizar un proceso de análisis de datos de forma correcta y completa.

La visualización de datos está en el núcleo del proceso. Es una herramienta básica para el analista o científico de datos que -mediante un proceso iterativo- va transformando y componiendo un modelo lógico de los datos. Apoyándose en la visualización, el analista va descubriendo los secretos enterrados en los datos. La visualización permite de forma rápida:

- Descartar aquellos datos poco representativos o erróneos.

- Identificar aquellas variables que dependen unas de otras y por lo tanto contienen información redundante
- Realizar cortes a los datos para poder observarlos desde diferentes perspectivas.
- Finalmente, comprobar que aquellos modelos, tendencias, predicciones y agrupaciones que hemos aplicado sobre los datos, nos devuelven el resultado esperado.

Herramientas para el análisis visual de datos Tan importante es la visualización gráfica de los datos en todos los ámbitos de la ciencia, ingeniería, negocios, banca, medio ambiente, etc. que existen multitud de herramientas para diseñar, desarrollar y comunicar la visualización gráfica de los datos. Estas herramientas cubren un amplio espectro del público objetivo, desde desarrolladores de Software, hasta científicos de datos pasando por periodistas y profesionales de la comunicación.

Para desarrolladores de Software, existen cientos de librerías y paquetes de Software que contienen miles de tipos de visualizaciones. Los desarrolladores tan solo tienen que cargar estas librerías en sus respectivos entornos de trabajo de programación y parametrizar el tipo de gráfico que deseen generar. El desarrollador tan solo ha de indicar los datos de origen que desea representar, el tipo de gráfico (líneas, barras, etc.) y la parametrización de dicho gráfico (escalas, colores, etiquetas, etc.).

El científico de datos acostumbra a trabajar con un entorno de trabajo de análisis concreto que, normalmente, incluye todos los componentes, entre ellos su motor de análisis visual de los datos. Los entornos más populares, hoy en día, para la ciencia de datos son R y Python, y ambos incluyen librerías nativas para la analítica visual. Quizás la librería más popular y potente en R sea ggplot2, mientras que en Python, matplotlib y Plotly son de las más populares.

Para comunicadores profesionales o personal no técnico de las distintas áreas de negocio (Marketing, Recursos Humanos, Producción, etc.) que necesitan tomar decisiones basadas en datos, existen herramientas -que no son únicamente herramientas de Visual Analytics- con funcionalidades para generar representaciones gráficas de los datos. Herramientas modernas de Business Intelligence de autoservicio como MS Excel, MS Power BI, Qlik,

Tableau, etc. son estupendas herramientas para comunicar los datos sin necesidad de disponer de competencias en programación o codificación.

En definitiva, las herramientas de visualización permiten a todos estos profesionales acceder a los datos de una manera más ágil y sencilla. En un universo donde la cantidad de datos útiles a analizar no deja de crecer, cada vez son más necesarias este tipo de herramientas, que facilitan la obtención de valor procedente de los datos y, con ello, la toma de decisiones relativas al presente y al futuro de nuestro negocio o actividad. Se ponen a disposición los fuentes de los gráficos mostrados a continuación y otros más en la liga: Graficos

4.1 Python

El lenguaje Python permite realizar una amplia variedad de visualizaciones de datos en 2D, 3D y animaciones. La misma se realiza mediante la ayuda de módulos (como math o numpy) que están destinados a brindar herramientas de visualización.

En la lista que sigue se presentan algunos de los módulos más usados. El objetivo no es aprenderlos, sino saber que existen varias opciones y disminuir un poco la confusión que se puede producir al principio al buscar ayuda y bibliografía:

- Matplotlib
- Seaborn
- Plotly
- Bokeh
- Altair
- Pygal
- Pandas
- Plotnine

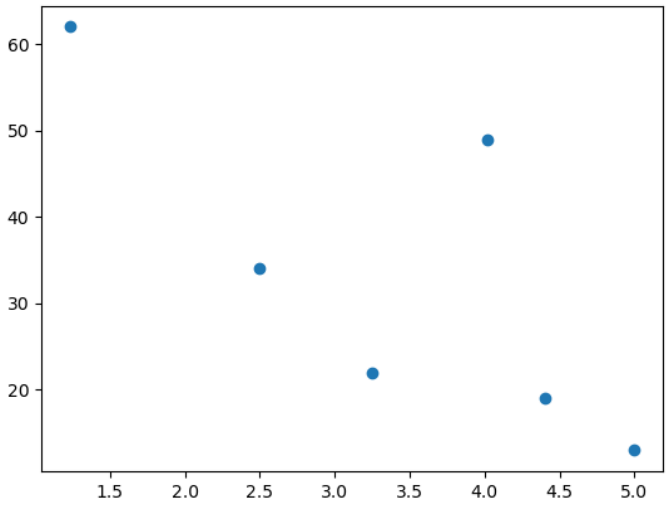
La elección de una de estas herramientas para hacer un gráfico depende del contexto, para qué se quiere hacer el gráfico, dónde se va a mostrar y a partir

de qué datos. Así, por ejemplo Plotly, Bokeh y Altair devuelven gráficos en HTML para ser mostrados en páginas Web, Pygal genera gráficos vectoriales y Pandas gráfica datos guardados en cierto tipo especial de estructura.

Por ejemplo:

```
import matplotlib.pyplot as plt
precio = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
ventas_por_dia = [34, 62, 49, 22, 13, 19]
plt.scatter(precio, ventas_por_dia)
plt.savefig('Grafica.pdf')
plt.savefig('Grafica.png')
plt.show()
```

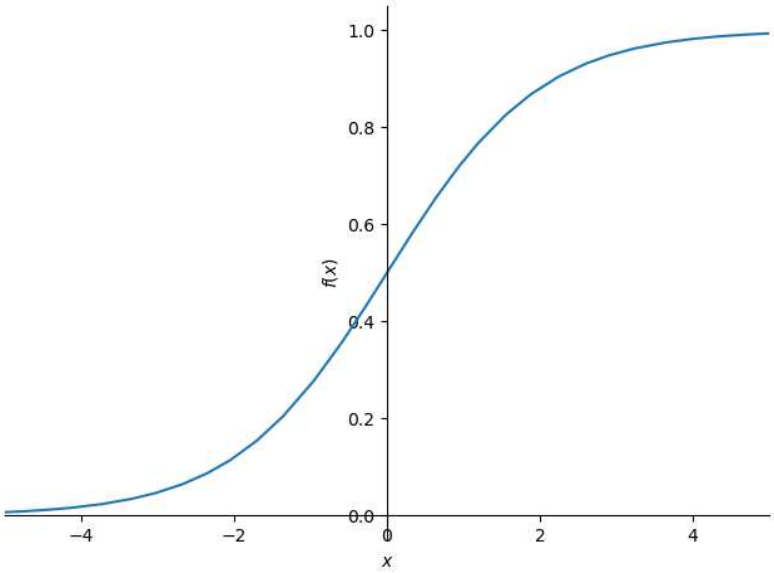
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejemplo:

```
import sympy
graf = sympy.plot("1/(1+exp(-x))", xlim=(-5,5));
graf.save('Grafica.pdf')
graf.save('Grafica.png')
```

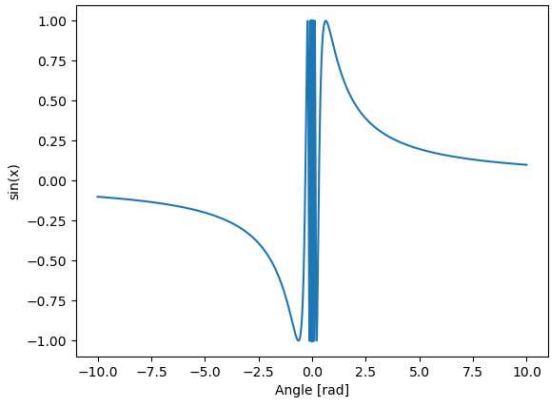
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejmplo:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 100000)
y = np.sin(1 / x)
# se graban los datos
np.savetxt('valores.txt'.(x,y))
# se leen los datos
xx,yy = np.loadtxt('valores.txt')
# se grafica
plt.figure()
plt.plot(xx, yy)
plt.xlabel("Angle [rad]")
plt.ylabel("sin(x)")
plt.axis("tight")
plt.show()
```

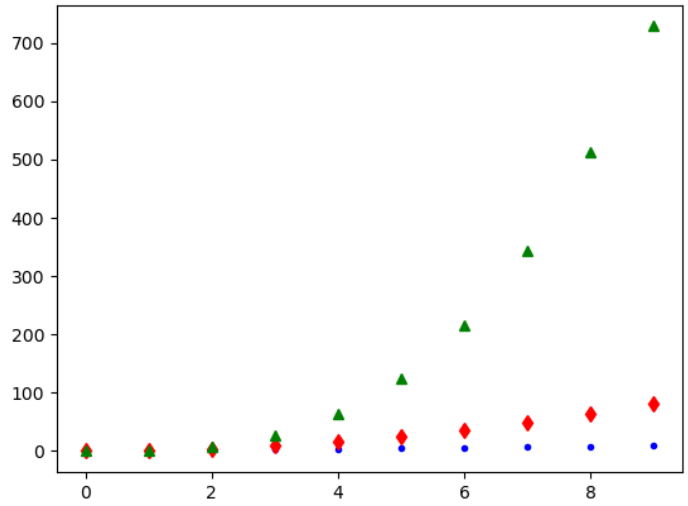
genera el siguiente gráfico (además de grabar y leer los valores de las variables x e y):



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = arange(10.) # array de floats, de 0.0 a 9.0
x2 = x**2             # definimos el array x2
x3 = x**3             # definimos el array x3
# dibujamos tres curvas en el mismo gráfico y figura
plot(x, x, 'b.', x, x2, 'rd', x, x3, 'g^')
show()
```

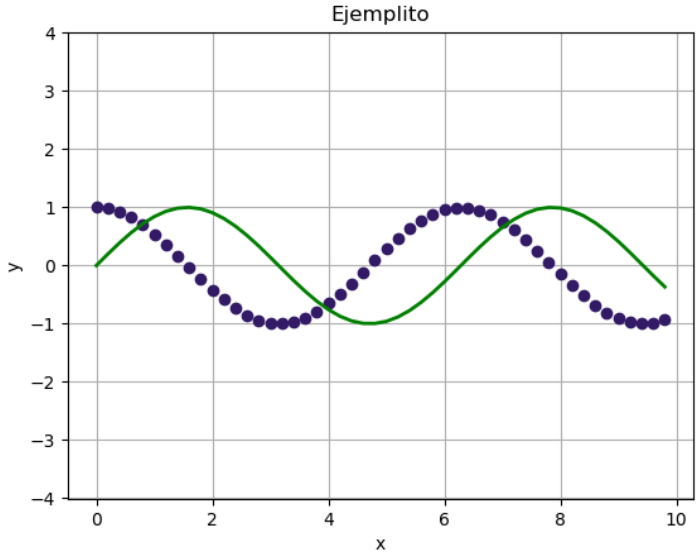
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,10,0.2)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x,y1,'o',linewidth=3,color=(0.2,0.1,0.4))
plt.plot(x,y2,'-',linewidth=2,color='g')
plt.grid()
plt.axis('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ejemplito')
plt.show()
```

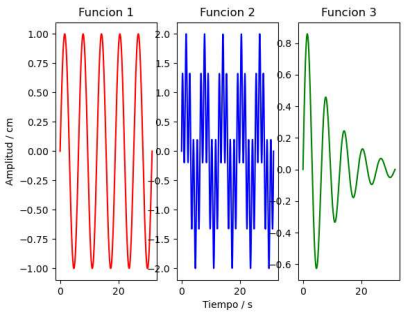
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = linspace(0, 10*pi, 800) # array de valores a representar
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
subplot(131)
p1, = plot(x,f1(x),'r-')
ylabel('Amplitud / cm')
title('Funcion 1')
subplot(132)
p2, = plot(x,f2(x),'b-')
xlabel('Tiempo / s')
title('Funcion 2')
subplot(133)
p3, = plot(x, f3(x),'g-')
title('Funcion 3')
show()
```

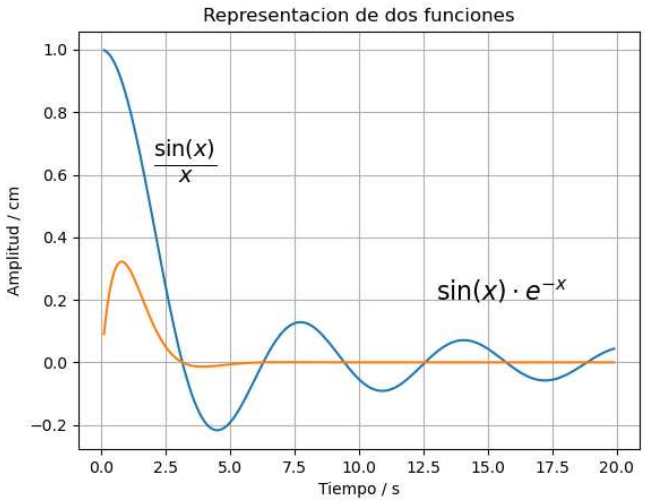
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
t = arange(0.1, 20, 0.1)
y1 = sin(t)/t
y2 = sin(t)*exp(-t)
p1, p2 = plot(t, y1, t, y2)
texto1 = text(2, 0.6, r'$\frac{\sin(x)}{x}$', fontsize=20)
texto2 = text(13, 0.2, r'$\sin(x) \cdot e^{-x}$', fontsize=16)
grid()
title('Representacion de dos funciones')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
show()
```

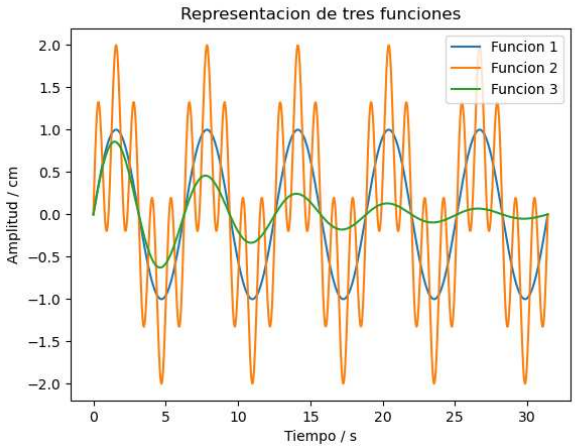
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
x = linspace(0, 10*pi, 800)
p1, p2, p3 = plot(x, f1(x), x, f2(x), x, f3(x))
legend(('Funcion 1', 'Funcion 2', 'Funcion 3'),
prop = {'size': 10}, loc='upper right')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
title('Representacion de tres funciones')
figure(figsize=(12, 5))
show()
```

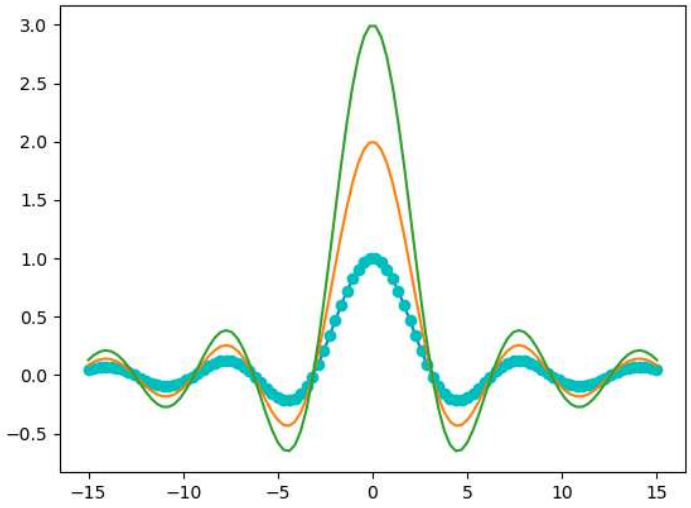
genera el siguiente gráfico:



Otro ejemplo:

```
import pylab
import numpy
x = numpy.linspace(-15, 15, 100)
y = numpy.sin(x) / x
pylab.plot(x, y)
pylab.plot(x, y, "co")
pylab.plot(x, 2 * y, x, 3 * y)
pylab.show()
```

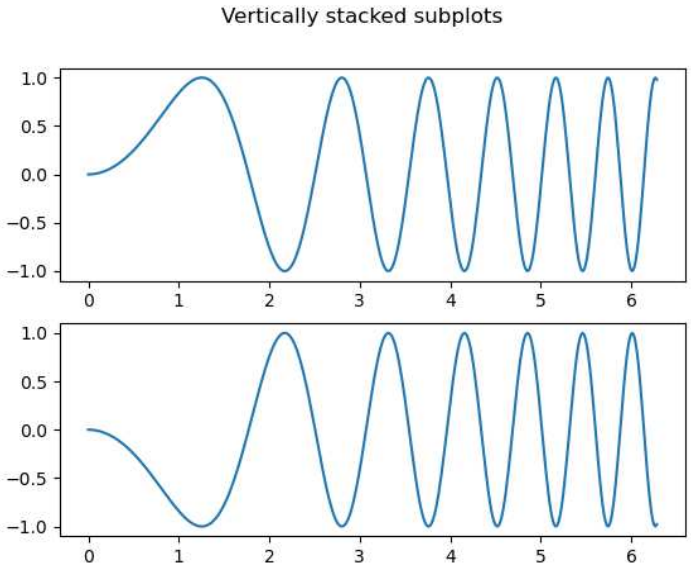
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
def multiple_plots():
    # Some example data to display
    x = np.linspace(0, 2 * np.pi, 400)
    y = np.sin(x ** 2)
    fig, axs =plt.subplots(2)
    fig.suptitle('Vertically stacked subplots')
    axs[0].plot(x, y)
    axs[1].plot(x, -y)
    plt.show( )
multiple_plots()
```

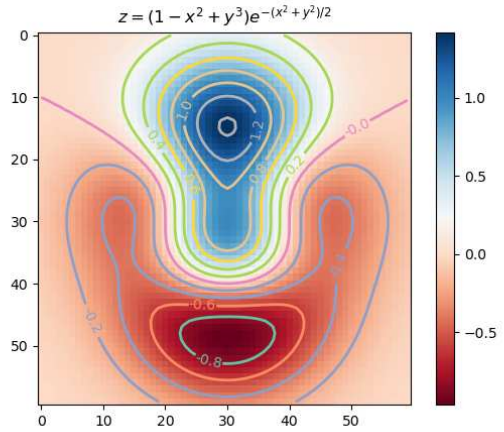
genera el siguiente gráfico:



Otro ejemplo:

```
from numpy import exp, arange
from pylab import meshgrid, cm, imshow, contour, clabel,
colorbar, axis, title, show
def z_func(x, y):
  return (1 - (x ** 2 + y ** 3)) * exp(-(x ** 2 + y ** 2) / 2)
x = arange(-3.0, 3.0, 0.1)
y = arange(-3.0, 3.0, 0.1)
X, Y = meshgrid(x, y)
Z = z_func(X, Y)
im = imshow(Z, cmap=cm.RdBu)
cset = contour(Z, arange(-1, 1.5, 0.2), linewidths=2, cmap=cm.Set2)
clabel(cset, inline=True, fmt="%1.1f", fontsize=10)
colorbar(im)
title("$z=(1-x^2+y^3) e^{-(x^2+y^2)/2}$")
show()
```

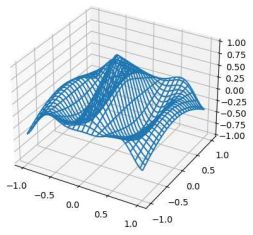
genera el siguiente gráfico:



Otro ejemplo:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
import time
def generate(X, Y, phi):
    R = 1 - np.sqrt(X ** 2 + Y ** 2)
    return np.cos(2 * np.pi * X + phi) * R
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
xs = np.linspace(-1, 1, 50)
ys = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(xs, ys)
ax.set_zlim(-1, 1)
wframe = None
tstart = time.time()
for phi in np.linspace(0, 180.0 / np.pi, 100):
    if wframe:
    ax.collections.remove(wframe)
    Z = generate(X, Y, phi)
    wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)
    plt.pause(0.001)
```

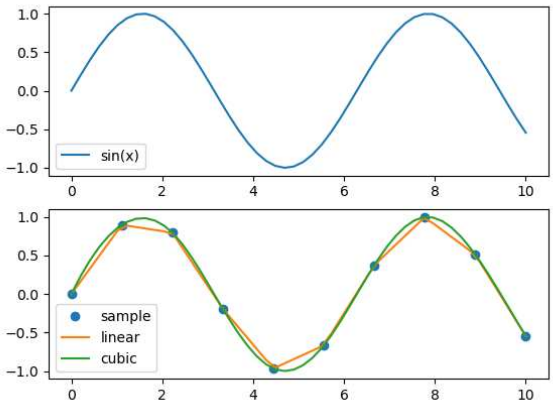
genera el siguiente animación:



Otro ejemplo:

```
import scipy.interpolate as sp
import numpy
import pylab
xx = numpy.linspace(0, 10, 50)
yy = numpy.sin(xx)
x = numpy.linspace(0, 10, 10)
y = numpy.sin(x)
fl = sp.interp1d(x, y,kind='linear')
fc = sp.interp1d(x, y,kind='cubic')
xnew = numpy.linspace(0, 10, 50)
pylab.subplot(211)
pylab.plot(xx, yy)
pylab.legend(['sin(x)'], loc='best')
pylab.subplot(212)
pylab.plot(x, y, 'o', xnew, fl(xnew), xnew, fc(xnew))
pylab.legend(['sample', 'linear', 'cubic'], loc='lower left')
pylab.show()
```

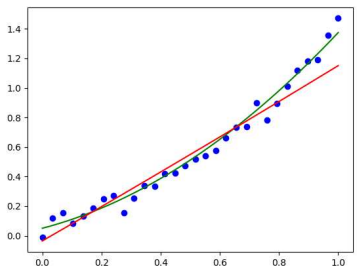
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
from numpy import *
from numpy.random import normal
from scipy.optimize import fmin
# parametric function, x independent variable, c parameters
fp = lambda c, x: c[0]+c[1]*x+c[2]*x*x
real_p = rand(3)
# error function to minimize
e = lambda p, x, y: (abs((fp(p,x)-y))).sum()
# generating data with noise
n = 30
x = linspace(0,1,n)
y = fp(real_p,x) + normal(0,0.05,n)
# fitting the data with fmin
p0 = rand(3) # initial parameter value
p = fmin(e, p0, args=(x,y))
print ('estimater parameters: ', p)
print ('real parameters: ', real_p)
xx = linspace(0,1,n*3)
plot(x,y,'bo', xx,fp(real_p,xx),'g', xx, fp(p,xx),'r')
show()
```

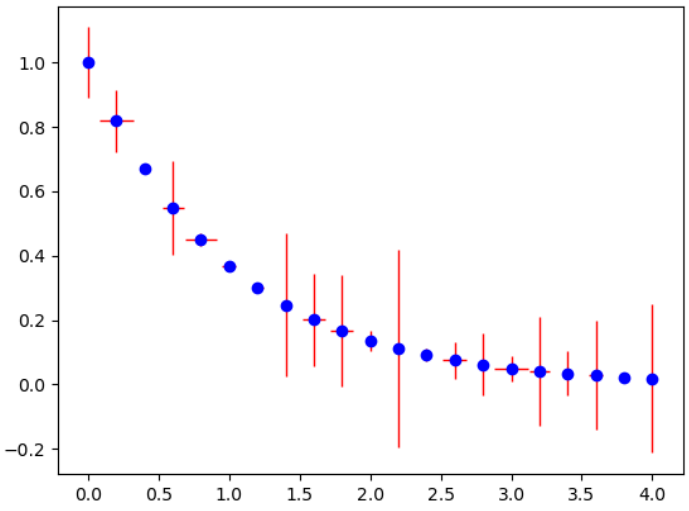
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as npr
x=np.linspace(0,4,21)
y=np.exp(-x)
xe=np.abs(0.08*npr.randn(len(x)))
ye=np.abs(0.1*npr.randn(len(y)))
plt.errorbar(x,y,fmt='bo',lw=2,xerr=xe,yerr=ye,ecolor='r',elinewidth=1)
plt.show()
```

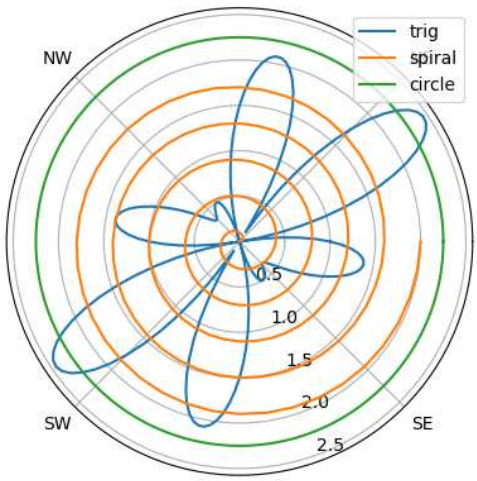
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
theta=np.linspace(0,2*np.pi,201)
r1=np.abs(np.cos(5.0*theta) - 1.5*np.sin(3.0*theta))
r2=theta/np.pi
r3=2.25*np.ones_like(theta)
plt.polar(theta, r1,label='trig')
plt.polar(5*theta, r2,label='spiral')
plt.polar(theta, r3,label='circle')
plt.thetagrids(np.arange(45,360,90), ('NE','NW','SW','SE'))
plt.rgrids((0.5,1.0,1.5,2.0,2.5),angle=290)
plt.legend(loc='best')
plt.show()
```

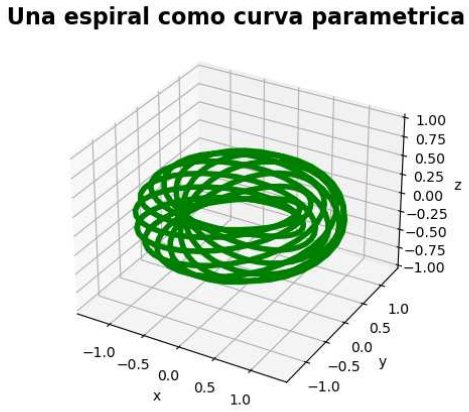
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
theta=np.linspace(0,2*np.pi,401)
a, m, n = 0.3, 11, 9
x=(1+a*np.cos(n*theta))*np.cos(m*theta)
y=(1+a*np.cos(n*theta))*np.sin(m*theta)
z=a*np.sin(n*theta)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot(x,y,z,'g',linewidth=4)
ax.set_zlim3d(-1.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Una espiral como curva parametrica', weight='bold',size=16)
plt.show()
```

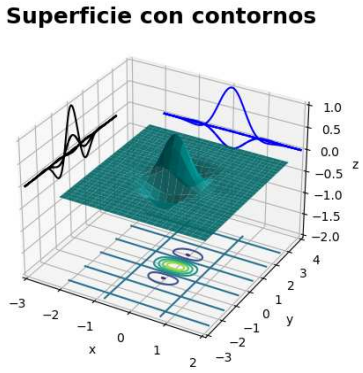
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
xx, yy=np.mgrid[-2:2:81j, -3:3:91j]
zz=np.exp(-2*xx**2-yy**2)*np.cos(2*xx)*np.cos(3*yy)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(xx,yy,zz,rstride=4,cstride=3,color='c',alpha=0.9)
ax.contour3D(xx,yy,zz,zdir='x',offset=-3.0,colors='black')
ax.contour3D(xx,yy,zz,zdir='y',offset=4.0,colors='blue')
ax.contour3D(xx,yy,zz,zdir='z',offset=-2.0)
ax.set_xlim3d(-3.0,2.0)
ax.set_ylim3d(-3.0,4.0)
ax.set_zlim3d(-2.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Superficie con contornos',weight='bold',size=18)
plt.show()
```

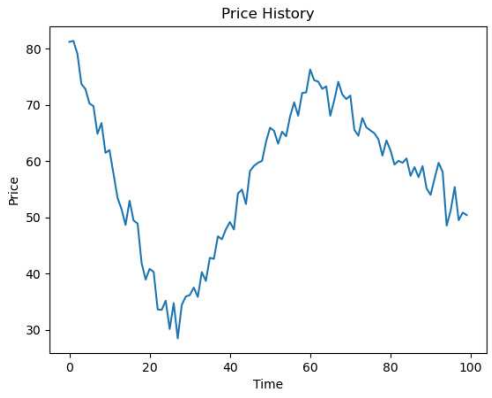
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
prices = np.full(100, fill_value=np.nan)
prices[[0, 25, 60, -1]] = [80.0, 30.0, 75.0, 50.0]
x = np.arange(len(prices))
is_valid = ~np.isnan(prices)
prices = np.interp(x=x, xp=x[is_valid], fp=prices[is_valid])
prices += np.random.randn(len(prices)) * 2
mn = np.argmin(prices)
mx = mn + np.argmax(prices[mn:])
kwargs = {"markersize": 12, "linestyle": ""}
fig, ax = plt.subplots()
ax.plot(prices)
ax.set_title("Price History")
ax.set_xlabel("Time")
ax.set_ylabel("Price")
ax.plot(mn, prices[mn], color="green", **kwargs)
ax.plot(mx, prices[mx], color="red", **kwargs)
plt.show()
```

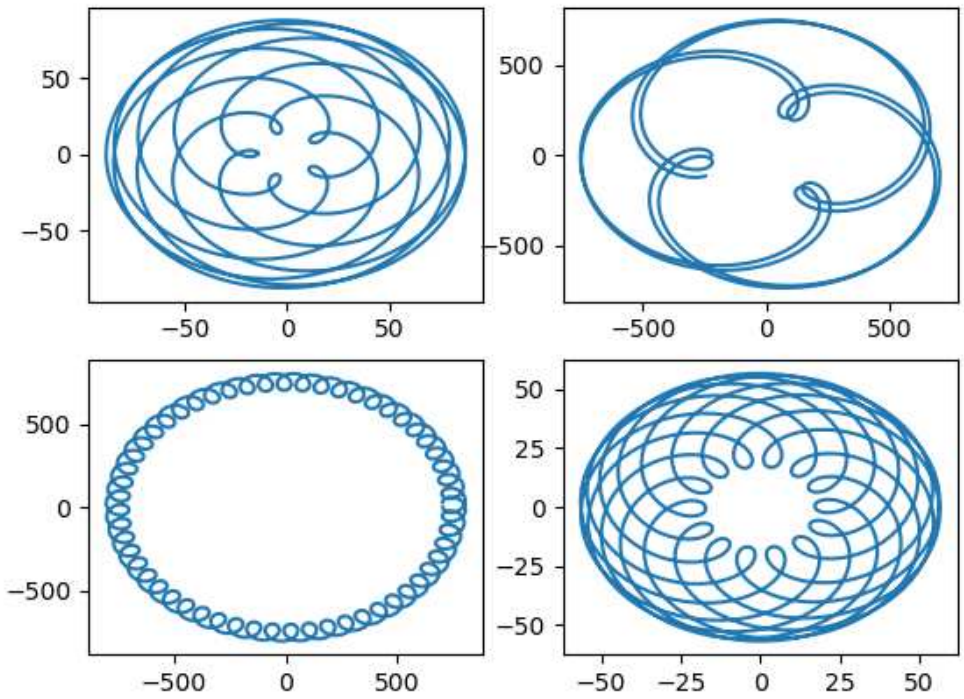
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plot
from numpy import linspace, sin, cos, pi
period = {
    'mercury' : 87.96926,
    'venus' : 224.7008,
    'earth' : 365.25636,
    'mars' : 686.97959,
    'ceres' : 1680.22107,
    'jupiter' : 4332.8201,
    'saturn' : 10755.699,
    'uranus' : 20687.153,
    'neptune' : 60190.03
}
dist = lambda T : T**(2/3) # Kepler
def plot_orbit(ax, planet1, planet2, periods=10):
    T1 = period[planet1]
    T2 = period[planet2]
    d1 = dist(T1)
    d2 = dist(T2)
    theta = linspace(0, 2*pi*periods, 1000)
    x = d1*cos(T2*theta/T1) - d2*cos(theta)
    y = d1*sin(T2*theta/T1) - d2*sin(theta)
    ax.plot(x, y)
fig=plot.figure()
ax = fig.add_subplot(2,2,1)
plot_orbit(ax, "venus", "earth", 8)
ax = fig.add_subplot(2,2,2)
plot_orbit(ax, "jupiter", "saturn", 4)
ax = fig.add_subplot(2,2,3)
plot_orbit(ax, "uranus", "earth", 57)
ax = fig.add_subplot(2,2,4)
plot_orbit(ax, "mercury", "venus", 9)
plot.show()
```

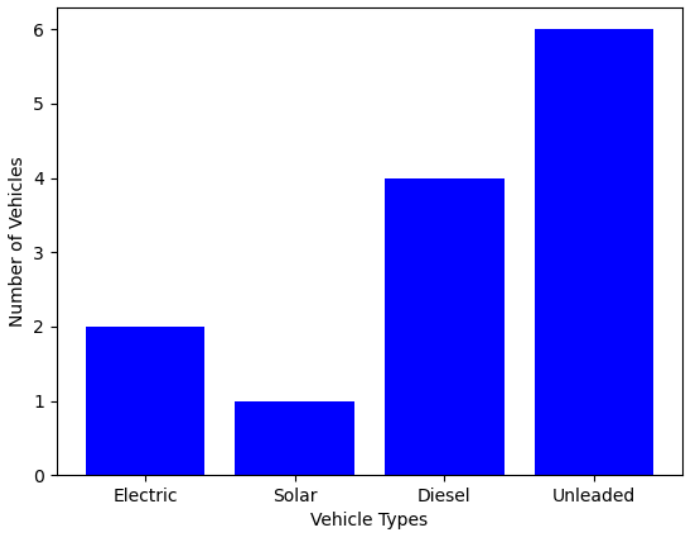
genera el siguiente gráfico:


50
0
−50
−50
0
50
500
0
−500
−500
0
500
500
0
−500
−500
0
500
50
25
0
−25
−50
−50
−25
0
25
50

Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

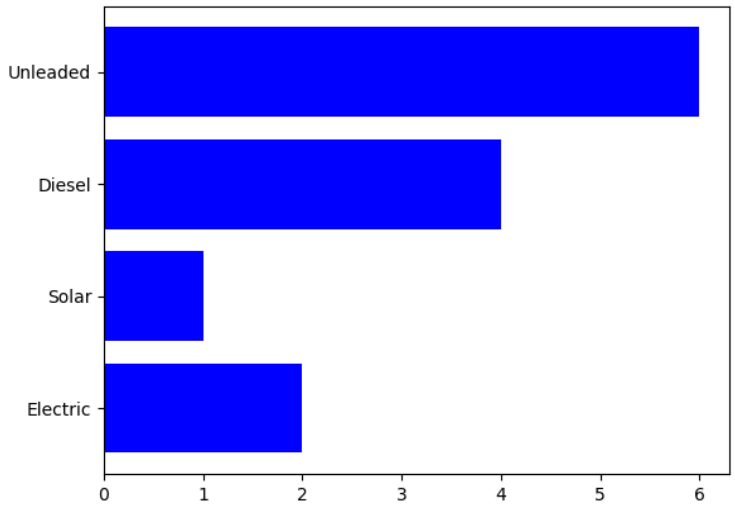
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_charth(numbers, labels, pos):
    plt.barh(pos, numbers, color="blue")
    plt.yticks(ticks=pos, labels=labels)
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_charth(numbers, labels, pos)
```

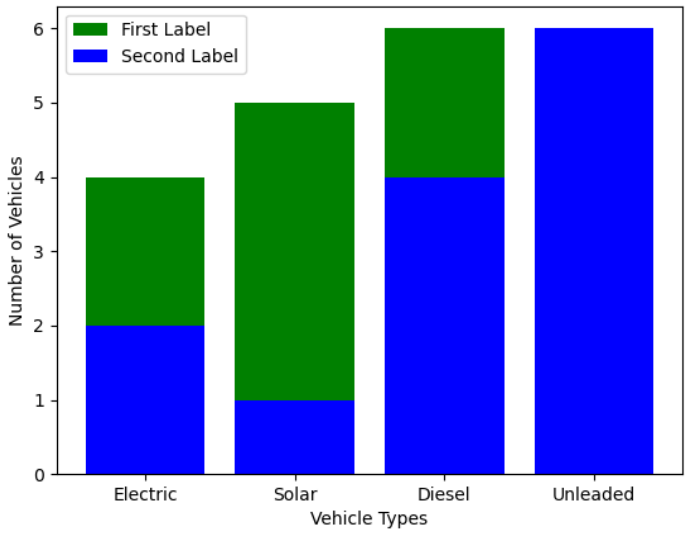
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, [4, 5, 6, 3], color="green")
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.legend(["First Label", "Second Label"], loc="upper
left")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

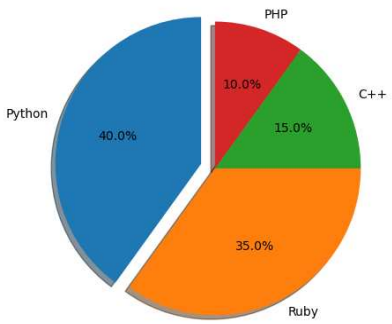
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def pie_chart():
    numbers = [40, 35, 15, 10]
    labels = ["Python", "Ruby", "C++", "PHP"]
# Explode the first slice (Python)
    explode = (0.1, 0, 0, 0)
    fig1, ax1 = plt.subplots()
    ax1.pie(
        numbers,
        explode=explode,
        labels=labels,
        shadow=True,
        startangle=90,
        autopct="%1.1f%%",
    )
    ax1.axis("equal")
    plt.show()
pie_chart()
```

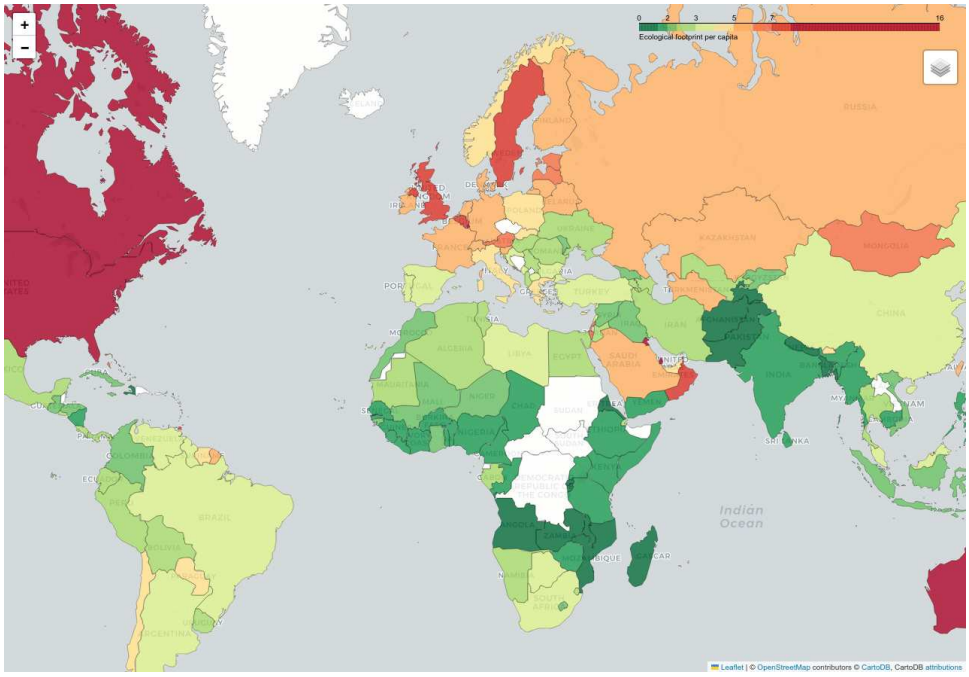
genera el siguiente gráfico:



Otro ejemplo:

```
import folium
import pandas as pd
eco_footprints = pd.read_csv("footprint.csv")
max_eco_footprint = eco_footprints["Ecological footprint"].max()
political_countries_url = (
    "http://geojson.xyz/naturalearth-3.3.0/ne_50m_admin_0_countries.geojson"
)
m = folium.Map(location=(30, 10), zoom_start=3, tiles="cartodb
positron")
folium.Choropleth(
    geo_data=political_countries_url,
    data=eco_footprints,
    columns=("Country/region", "Ecological footprint"),
    key_on="feature.properties.name",
    bins=(0, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, max_eco_footprint),
    fill_color="RdYlGn_r",
    fill_opacity=0.8,
    line_opacity=0.3,
    nan_fill_color="white",
    legend_name="Ecological footprint per capita",
    name="Countries by ecological footprint per capita",
).add_to(m)
folium.LayerControl().add_to(m)
m.save("M7.html")
```

genera el siguiente gráfico:


Ecological footprint per capita
Indian Ocean
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

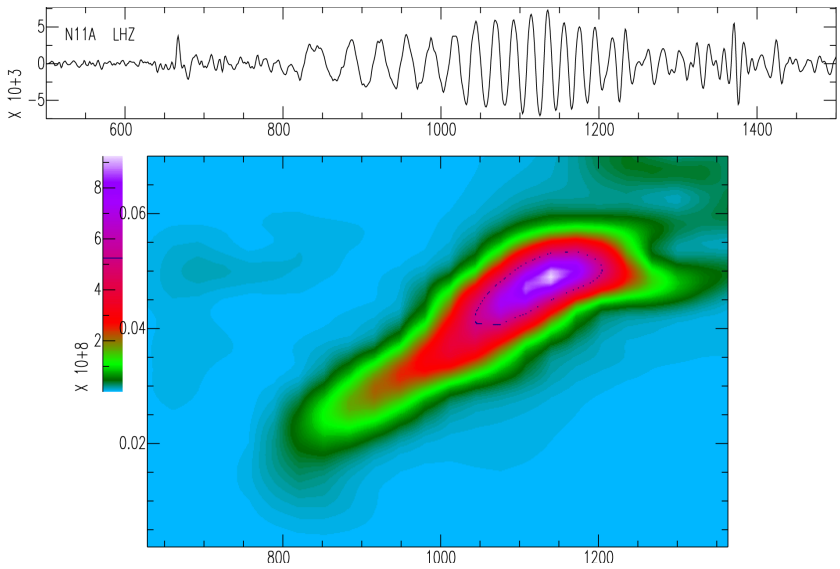
Existe una gran cantidad herramientas para generar gráficos 2D, 3D y videos que se pueden usar desde lenguajes de programación, en particular para Python hay una gran variedad, entre las utilidades destacan:

- https://mpmath.org/
- https://plotly.com/python/
- https://github.com/szabolcsdombi/zengl
- https://towardsdatascience.com/interactive-animated-visualization-db91d1c858ad
- https://docs.enthought.com/mayavi/mayavi/
- https://docs.bokeh.org/en/latest/index.html
- https://matplotlib.org/
- https://pyopengl.sourceforge.net/
- https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- https://pythonplot.com/
- http://www.pyqtgraph.org/
- http://seaborn.pydata.org/
- https://github.com/sepandhaghighi/samila
- https://github.com/paulcbogdan/NiChord
- https://pypi.org/project/Shapely/
- https://tech.marksblogg.com/pretty-maps-in-python.html
- https://www.manim.community/
- https://plotnine.readthedocs.io/en/stable/
- https://www.pythonguis.com/tutorials/pyqt6-plotting-pyqtgraph/
- https://pbpython.com/visualization-tools-1.html

- https://github.com/moshi4/pyCirclize
- https://mathspp.com/blog/til/xkcd-plots
- https://github.com/ResidentMario/geoplot

4.2 SAC

SAC (Código de análisis sísmico) es un programa interactivo de propósito general diseñado para el estudio de señales secuenciales, especialmente datos de series temporales. Se ha puesto énfasis en las herramientas de análisis utilizadas por los sismólogos investigadores en el estudio detallado de eventos sísmicos. Las capacidades de análisis incluyen operaciones aritméticas generales, transformadas de Fourier, tres técnicas de estimación espectral, filtrado IIR y FIR, apilamiento de señales, diezmado, interpolación, correlación y selección de fase sísmica. SAC también contiene una amplia capacidad gráfica.

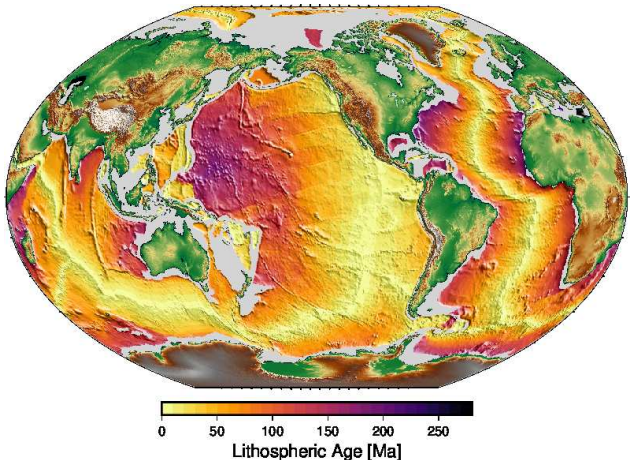


SAC fue desarrollado en el Laboratorio Nacional Lawrence Livermore y tiene derechos de autor de la Universidad de California. Actualmente es desarrollado y mantenido por un pequeño grupo de desarrolladores que trabajan en cooperación con IRIS. Las versiones binarias están disponibles para Intel Mac y Linux, pero SAC se puede construir a partir del código fuente para otros sistemas operativos de computadora. El código fuente está escrito en C.

4.3 Sistema de Información Geográfica

Un sistema de información geográfica (SIG) es un marco para recopilar, gestionar y analizar datos. Adaptado en la ciencia de la geografía, SIG integra muchos tipos de datos. Analiza la ubicación espacial y organiza capas de información en visualizaciones utilizando mapas y escenas en 3D. Con esta capacidad única, SIG revela conocimientos más profundos sobre los datos, como patrones, relaciones y situaciones, lo que ayuda a los usuarios a tomar decisiones más inteligentes.

Uno de los problemas más complejos de resolver cuando se trabaja con SIG, es la obtención de salidas cartográficas de suficiente calidad. No se trata de obtener mapas como los que producen los organismos cartográficos, sino tan sólo de disponer de un sistema lo suficientemente flexible como para crear un mapa que une los objetivos de representación y simplificación, así como los criterios estéticos, que se esperan de un mapa que va a publicarse, como tal o dentro de un trabajo científico.



Las salidas gráficas de un SIG se caracterizan por su inmediatez y por ser una representación para el usuario de los modelos de datos que utiliza el programa, asumiendo a este usuario los suficientes conocimientos de SIG, del área de estudio y de las variables representadas, como para que baste un representación cruda. Sin embargo un mapa es para uso de terceros a los que no debemos suponer ningún conocimiento a priori y que, por tanto, van a necesitar ayuda para interpretar el mapa (escalas, leyendas, etc.). Cons-

cientes de este problema, los desarrolladores de SIG han implementado una serie de herramientas de maquetación de mapas.

Los sistema de información geográfica debieran permitir:

- Crear, compartir y utilizar mapas inteligentes
- Compilar información geográfica
- Crear y administrar bases de datos geográficas
- Resolver problemas con el análisis espacial
- Crear aplicaciones basadas en mapas
- Dar a conocer y compartir información mediante la geografía y la visualización

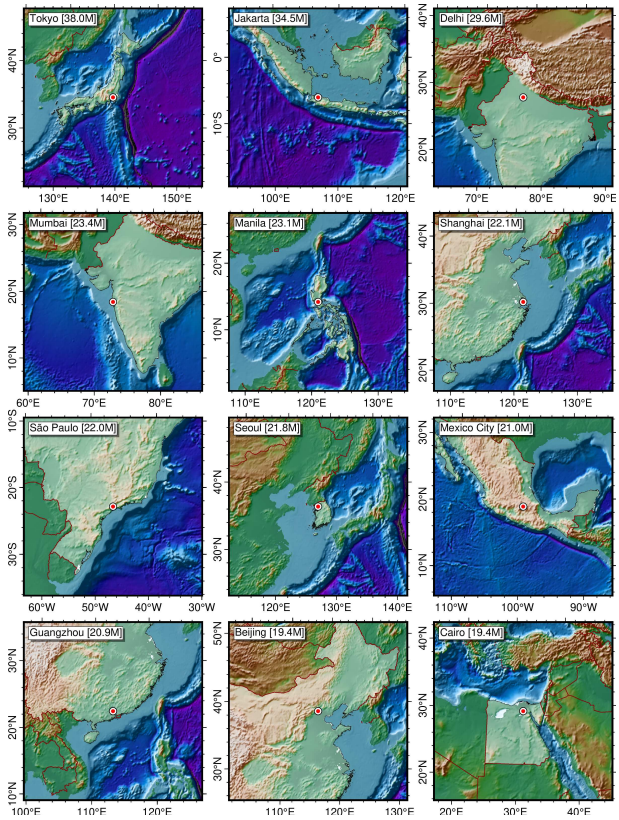
Existe una gran cantidad herramientas para que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- https://geopy.readthedocs.io/en/stable/

GMT GMT (Generic Mapping Tools) pone a disposición del usuario un conjunto de módulos orientados a la producción de cartografía a partir de datos codificados según los modelos lógicos (Raster, vectorial, lista de puntos) habituales en los SIG. Los datos raster se almacenan por defecto en formato propio (GMT/ netCDF) pero también se admiten ficheros binarios con datos en coma flotante, enteros, bytes o bits. Los datos vectoriales se almacenan como simples pares de coordenadas, no se incluye por tanto información topológica pero de hecho no tiene sentido incluirla si el resultado que espera obtenerse es un mapa para imprimir en papel.

El manejo de estos módulos en línea de comandos y la posibilidad de combinarlos (entre sí y con otras herramientas LINUX/UNIX) así como el elevado número de opciones de cada uno de ellos, convierte GMT en un entorno de maquetación de mapas extremadamente flexible. La contrapartida de esta flexibilidad es un lenguaje que puede llegar a ser bastante complejo y hermético.

Además de los módulos estrictamente destinados a la generación de mapas, GMT dispone de módulos de álgebra de mapas y de producción de



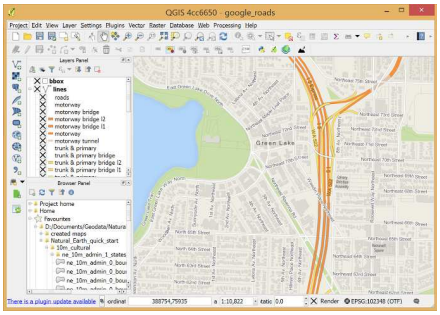
gráficos más generales (histogramas, etc.). GMT tiene capacidad de trabajar con lenguajes como: MATLAB/Octave, Julia o Python.

instalación en linux:

```
# apt install gmt gmt-gshhg-full
```

QGIS QGIS es un Sistema de Información Geográfica (SIG) de Código Abierto licenciado bajo GNU - General Public License . QGIS es un proyecto oficial de Open Source Geospatial Foundation (OSGeo). Corre sobre Linux, Unix, Mac OSX, Windows y Android y soporta numerosos formatos y funcionalidades de datos vector, datos ráster y bases de datos.

ArcGIS ArcGIS es un completo sistema que permite recopilar, organizar, administrar, analizar, compartir y distribuir información geográfica. Es una

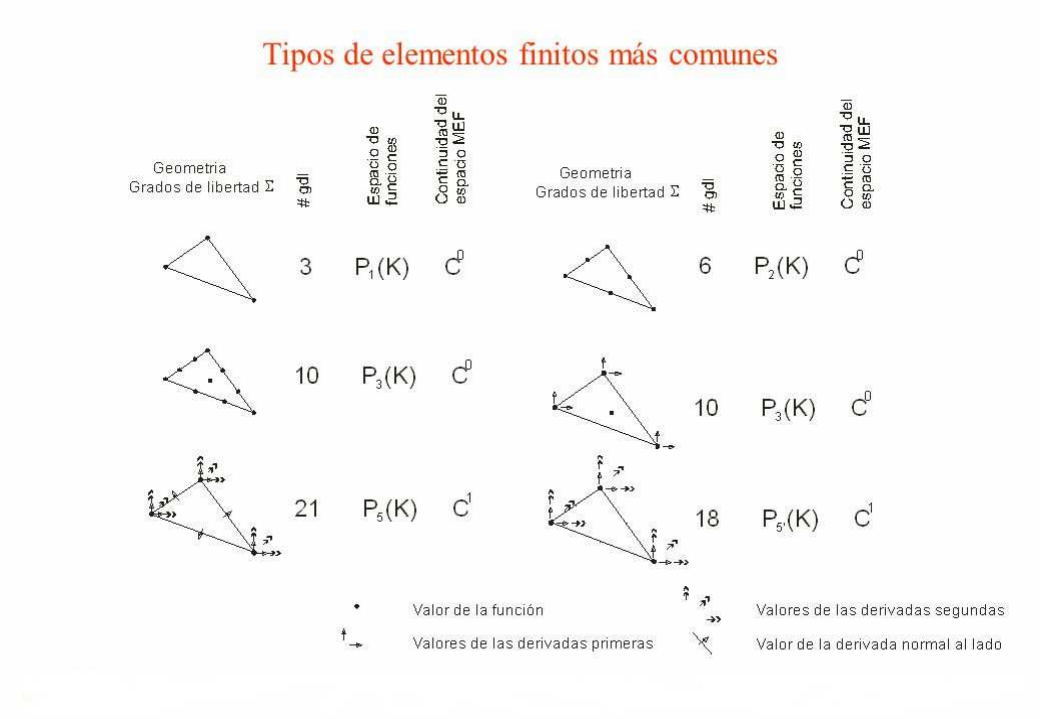


plataforma para crear y utilizar sistemas de información geográfica (SIG), ArcGIS es utilizada por personas de todo el mundo para poner el conocimiento geográfico al servicio de los sectores del gobierno, la empresa, la ciencia, la educación y los medios. ArcGIS permite publicar la información geográfica para que esté accesible para cualquier usuario. El sistema está disponible en cualquier lugar a través de navegadores Web, dispositivos móviles como Smartphones y equipos de escritorio.

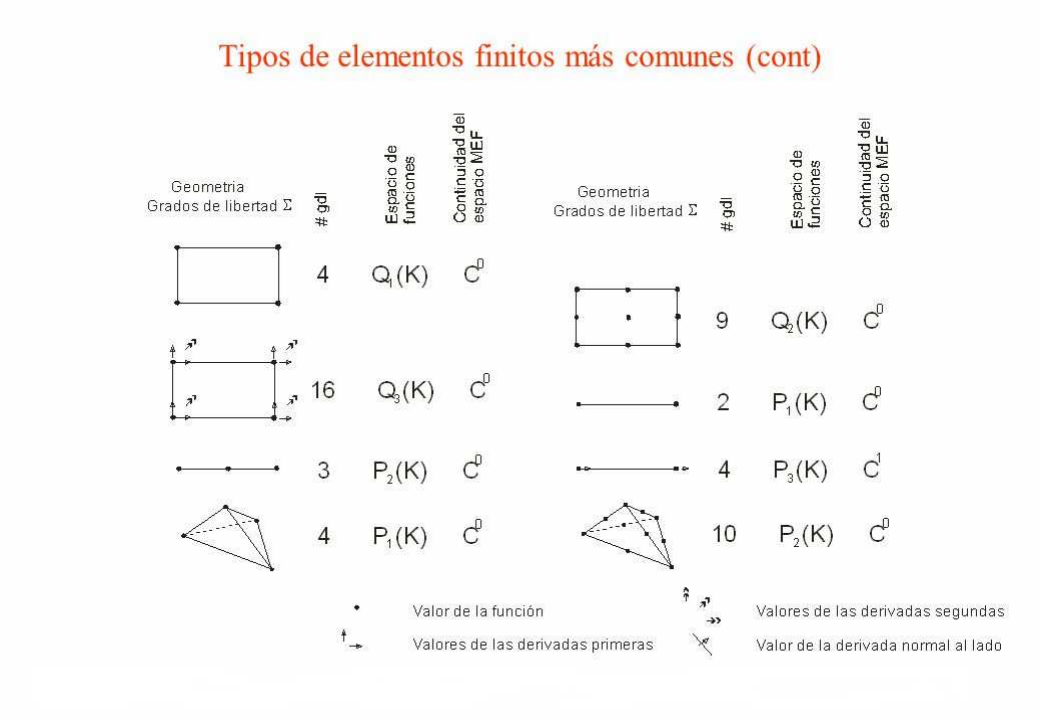
4.4 Generar Mallas con GMSH

GMSH es un generador para mallar tanto sus superficies como sus volúmenes para Elementos Finitos de código abierto con un motor integrado tipo CAD y un post-procesador. Su diseño tiene como fin proveer una herramienta rápida, ligera y amigable con un editor paramétrico y una gran capacidad de visualización avanzada. GMSH está organizado en cuatro módulos: geometría, malla, motor de cálculo, y post-proceso. La introducción de datos e instrucciones a cualquiera de los módulos puede hacerse ya sea de manera interactiva usando la interfaz gráfica, mediante archivos de texto ASCII empleando el

lenguaje propio de GMSH ó a través de la interfaz de programación de aplicaciones C, C++, Python, Julia, Fortran, etc.



Mallado. Se define como mallado la discretización de una línea, superficie o volúmen en porciones de tamaño finito. Las porciones, además de tener un tamaño característico varias veces menor que el espacio discretizado, serán entidades geométricas elementales como los triángulos o cuadriláteros en dos dimensiones o los tetraedros o prismas en tres dimensiones. Esta discretización en estructuras más elementales es esencial para la resolución de ecuaciones en derivadas parciales en dominios arbitrarios por el método de volúmenes finitos (FVM) o el método de elementos finitos (FEM).



El proceso de mallado más habitual es el de discretizar sucesivamente entidades de mayor dimensión como si de un problema de contorno se tratara. Por ejemplo, si queremos mallar un cubo primero definiremos los puntos de control en cada una de sus doce aristas. Luego discretizaremos en porciones elementales cada uno de sus seis lados y finalmente discretizaremos el volúmen. Para entender mejor este proceso, y con anterioridad a entender la sintaxis de los archivos GMSH presentamos este ejemplo de mallado de un cubo con tetraedros. El primer paso es definir los puntos básicos de la geometría.

Mallas estructuradas Una malla estructurada es una malla que puede ser transformada, mediante una transformación biyectiva, a una cuadrícula. Esto significa que el problema podría ser resuelto en una malla uniforme y rectangular y devuelto a la geometría original sólo conociendo el jacobiano de la transformación porque este jacobiano se puede invertir.

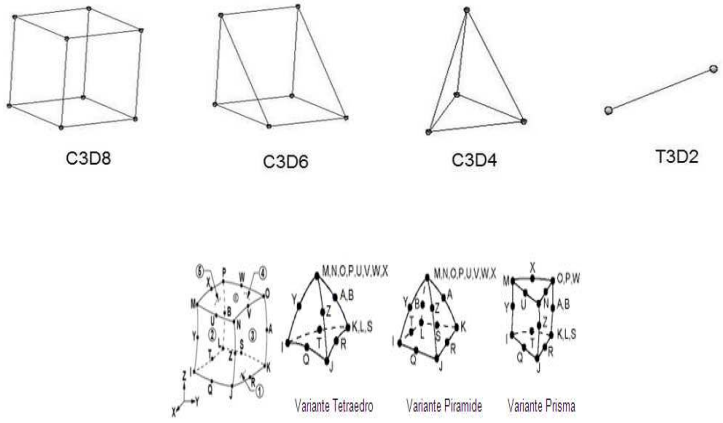


Fig. 3. Geometría del elemento finito sólido estructural

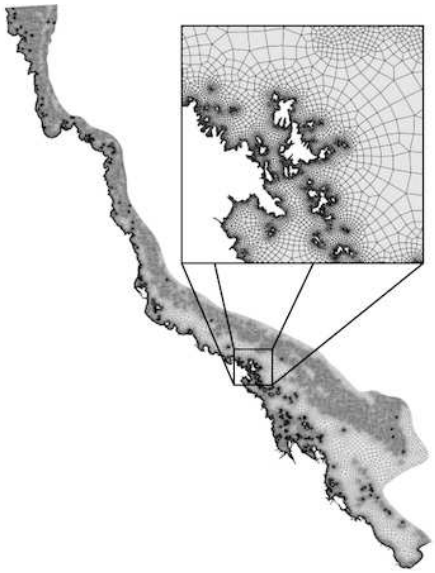
Mallas no estructuradas Las mallas no estructuradas parten de distribuciones de puntos que cumplen una serie de propiedades dadas, la mayoría de ellas relacionadas con una distribución lo más uniforme posible dentro del contorno. Son adecuadas para geometrías irregulares donde es muy difícil generar una malla estructurada.

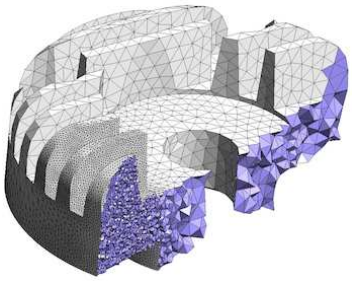
GMSH utiliza un algoritmo de creación de malla basado en la triangulación Delaunay. Dado un contorno se demuestra que sólo existe una triangulación óptima en la que, para cada triángulo formado por tres puntos de la malla, no exista ningún punto dentro de la circunferencia que pasa por los tres puntos.

Si bien esta propiedad cierra el problema de la triangulación dados los puntos de la malla, en la mayoría de los casos sólo dispondremos del contorno. GMSH es capaz de encontrar una distribución no estructurada de puntos cuya triangulación resultante tiene suficiente calidad.

Discretización y calidad de malla. Una de las diferencias esenciales entre las mallas estructuradas y no estructuradas es el proceso de refinado. En algunos problemas la malla utilizada no produce suficiente precisión y es necesario rehacerla para aumentarla. En las mallas estructuradas el aumento de precisión no suele suponer un problema más allá de generar demasiados grados de libertad. Sin embargo en las mallas no estructuradas algunos algoritmos de refinado automático pueden funcionar mal.

Uno de los algoritmos más utilizados es el de partir cada uno de los

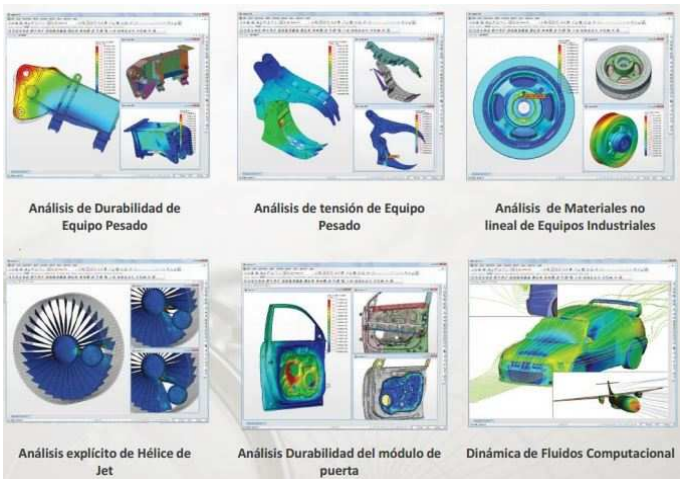




triángulos o tetraedros en dos o más elementos. Este algoritmo recursivo puede refinar puntos arbitrarios en el espacio produciendo una pérdida de precisión al introducir gradientes espurios debidos a una mala definición de la geometría.

GMSH se puede usar en conjunto con FreeFem++ que es un software libre que sirve para la resolución numérica de problemas 2D y 3D utilizando el método de elementos finitos. ¿Qué problemas puedes resolver? De mecánica de fluidos, difusión de calor, elasticidad, electromagnetismo, etc. Todos aquellos que estén modelizados matemáticamente a través de una ecuación en derivadas parciales (EDP), por ejemplo:

https://www.um.es/freefem/ff++/pmwiki.php?n=Main.Elasticidad3D



Existe una gran cantidad herramientas para generar mallas en 2D, 3D que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- https://www.salome-platform.org/
- https://www.paraview.org/
- https://www.meshlab.net/
- https://wias-berlin.de/software/index.jsp?id=TetGen&lang=1
- http://alice.loria.fr/software/geogram/doc/html/index.html