Capítulo 7

Gráficas de Datos

Una de las grandes ventajas de aprender programación en Python a diferencia de Fortran o C/C++, es que Python tiene paquetes para generar figuras de alta calidad (aunque yo recomiendo aprender igual Fortran o C/C++). El poder de los paquetes de graficas en Python es mucho mayor a lo que se puede presentar en este curso y es su trabajo aprender más de lo que se muestra acá.

Lo primero, es tener claro que una figura debe tener toda la información necesaria para que cualquier persona que la vea lo pueda enender. Por eso, cada eje debe estar descrito, y si se tienen más de un tipo de datos, se debe explicar que significa cada uno. Para la presentación de resultados científicos y especialmente en geociencias, es importante generar figuras que muestren la información de manera correcta, veraz y con buena resolución. No es aceptable entregar figuras sin ejes explicados, o escalas si es necesario.

Por último, aunque se muestran algunos ejemplos de figuras en 3D, éstas en muchos casos (aunque bonitas) no son útiles para presentar la información. Las páginas de revistas o la pantalla del computador es 2D, por lo que no es fácil ver la tercera dimensión. A veces es mejor un mapa de contornos, que mostrar la topografía en 3D en una figura.

El paquete que se utilizará es `matplotlib` y específicamente `pyplot`. Como referencia se puede encontrar en `https://matplotlib.org`

7.1. Gráficas 1D/2D

En el primer ejemplo, se muestra una figura con varios arreglos de datos y la forma de graficarlos en una sola figura. En esto se puede ver la forma de generar diferentes tipos de líneas, la leyenda y poner los límites para uno de los ejes. Finalmente, como se puede guardar la figura a un formato definido, que en este caso es un `.png`. Otros formatos incluyen `.ps`, `.eps`, `.pdf`, `.svg`, etc. E los ejemplos de abajo, los comandos `plt.savefig` y `plt.show()` no se muestran para evitar ocupar demasiado espacio. Los códigos completos (en la página del curso) si muestran todos los comandos.

```
# plot_lines.py
# Example of simple line plotting

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10., 10., num=20)
y1 = x*0 + 1
y2 = 0.2*x + 0.1
y3 = 0.05*x**2 + 0.1*x - 0.1
y4 = -0.002*x**3 + 0.01*x**2 - 0.05*x + 0.1
y5 = +1e-4*x**4 +0.002*x**3 + 0.01*x**2 - 0.05*x + 0.1

plt.plot(x,y1,marker='^',linestyle=' ',label='y1')
plt.plot(x,y2,':',label='y2')
plt.plot(x,y3,'--',label='y3')
plt.plot(x,y4,'-.',label='y4')
plt.plot(x,y5,'-',label='y5')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('X vs Y')
plt.ylim((-3, 8))
plt.legend()
plt.savefig('chap6_fig1.png')
plt.show()
```

con resultado en la figura 7.1.

Los formatos de las líneas y de los `markers` es muy variada y está por fuera del objetivo del curso. Una explicación más precisa se puede encontrar en la documentación de `matplotlib`. El espesor de la línea, el relleno de los símbolos, etc. se puede cambiar.

Como se muestra en el ejemplo anterior y en la Figura 7.1, se muestra un número de comandos para generar la figura (`plt.plot`) con escala lineal-lineal, aunque también se puede usar `semilogx` o `semilogy` o `loglog`. Se puede adicionar texto a cada eje, título de la figura, leyenda de los símbolos y finalmente guardar la figura y mostrar la figura en pantalla.

En el siguiente ejemplo se muestra el uso de figuras para mostrar datos reales, con barras de error. La idea es mostrar una serie de datos (que pueden representar datos tomados en campo) en función de la posición (`x`). La Figura 7.2 muestra dos paneles (ver comando `plt.subplots`), uno con los datos brutos (datos con errores aleatorios) y una curva que representa el modelo que representa los datos. Sin embargo, en la vida real cada dato tiene una incertidumbre que muchas veces se representa con una barra de error (por ejemplo mostrando el $\pm\sigma$). Esto se puede mostrar con el comando `plt.errorbar(x,y,yerr=yerr)`.

El código `plot_errorbar.py` muestra otra forma de trabajar con los módu-

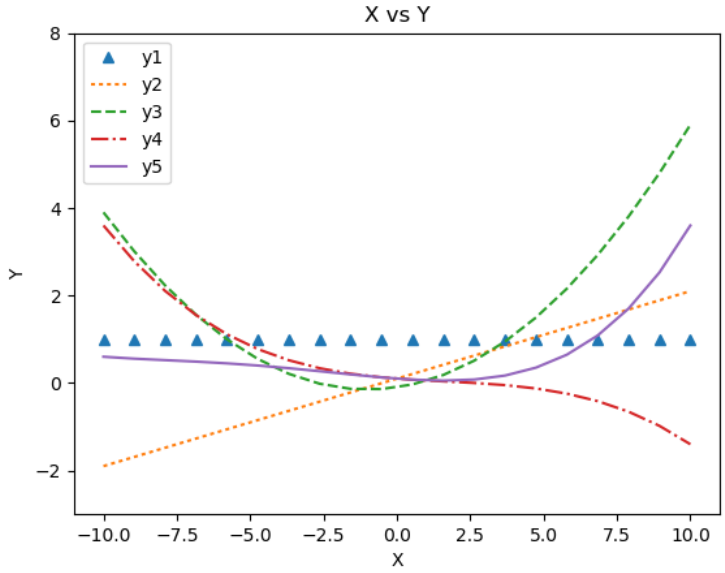

Figura 7.1: Resultado de correr `plot_lines.py`.

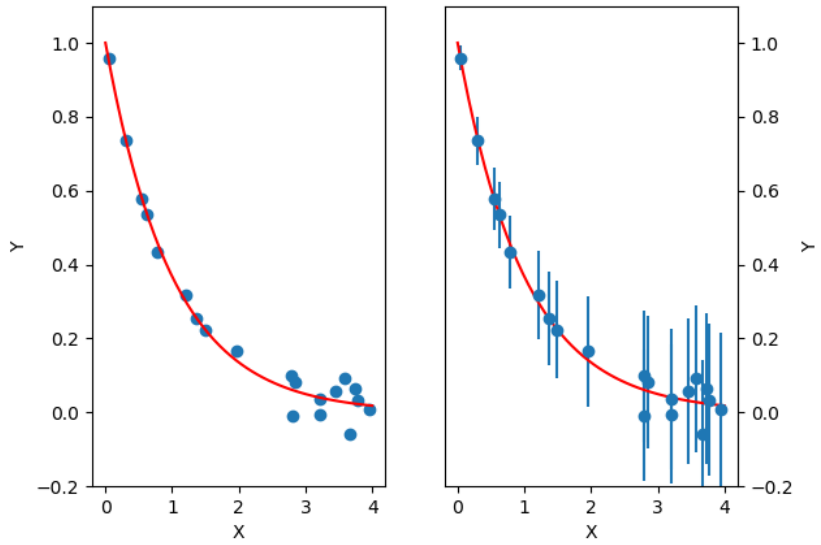

Figura 7.2: Resultado de correr `plot_errorbar.py`.

los de gráficas de Python. Note que primero se genera una figura con subpaneles `plt.subplots`, donde se genera una variable `axs` que se usa para generar las figuras de cada subpanel, usando `ax1 = axs[0]`, para el panel 1, y `ax2 = axs[1]` para el segundo panel.

```
# plot_errorbar.py
# Simple code to plot some data with
# error bars, and synthetic curve

import numpy as np
import matplotlib.pyplot as plt

N    = 20
x    = np.random.rand(N)*4
x    = np.sort(x)
sig  = 0.01 + 0.1*np.sqrt(x)
yerr = np.random.normal(0,0.3,size=N)*sig
y    = np.exp(-x) + yerr

x0 = np.linspace(0.0, 4.0)
y0   = np.exp(-x0)

# Plot the data
f, axs = plt.subplots(1, 2)

ax1 = axs[0]
ax1.scatter(x, y)
ax1.plot(x0,y0,color='r')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_ylim([-0.2 , 1.1])

ax2 = axs[1]
ax2.errorbar(x, y,yerr=sig,fmt='o')
ax2.plot(x0,y0,color='r')
ax2.set_xlabel('X')
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")
ax2.set_ylabel('Y')
ax2.set_ylim([-0.2 , 1.1])
```

Para definir el título de las figuras, los ejes, etc, se usa otra forma de comandos para definirlos, como se ve en

```
ax2.set_xlabel('X')
ax2.yaxis.tick_right()
```

```
ax2.yaxis.set_label_position("right")
ax2.set_ylabel('Y')
ax2.set_ylim([-0.2 , 1.1])
```

donde incluso se puede poner la posición del eje `Y` al lado derecho, para hacer la figura más legible.

Finalmente un ejemplo para mostrar una distribución de una serie de datos, que muestra por ejemplo si los datos tienen una distribución normal (gaussiana) y de otro tipo. La Figura 7.3 muestra un histograma de los datos y una cruva de una distribución normal.

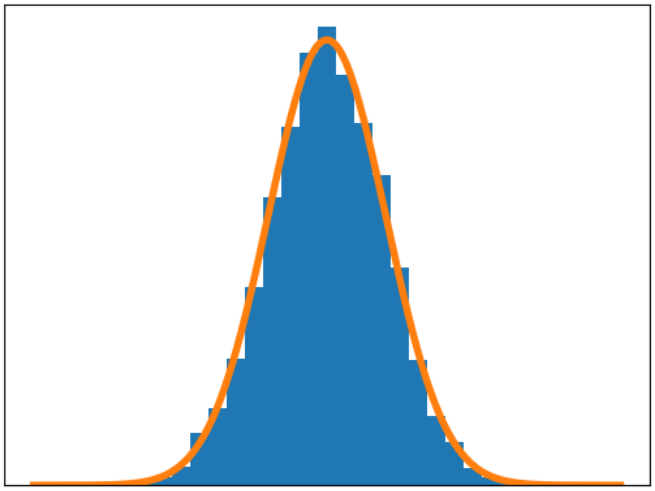

Figura 7.3: Resultado de correr `plot_histogram.py`.

El comando en este caso es `hist(X,bins=25, normed=True)` que muestra el histograma de los datos dividida en 25 *bins* para el rango de los datos. Se han removido los valores de los ejes.

```
# plot_histogram.py
# A simple histogram examples

import matplotlib.pyplot as plt
import numpy as np

rd = np.random.RandomState()
X = rd.randn(10000)
```

```
fig, ax = plt.subplots()
ax.hist(X, bins=25, normed=True)
x = np.linspace(-5, 5, 1000)
ax.plot(x, 1 / np.sqrt(2*np.pi) * np.exp(-(x**2)/2), linewidth=4)
ax.set_xticks([])
ax.set_yticks([])
...
```

7.2. Gráficas 2D/3D

En Geociencias, muchas veces se toman datos en la superficie de la Tierra y se quieren mostrar en mapa o en sección cruzada o en figura 3D. Sin embargo, es importante tener en cuenta que las figuras en 3D muchas veces no son muy útiles para mostrar los datos. En los siguientes ejemplos muestro 3-4 formas de presentar unos datos `z` tomados en la posición `x` y `y` donde `z` puede representar por ejemplo altura, alguna anomalía geofísica, contenido de SiO_2, etc.

La figura 7.4 muestra una figura tipo `scatter` donde cada punto representa un dato tomado en campo con tres valores `x y z`. La figura muestra la posición (`x,y`) y en color el valor correspondiente a `z`.

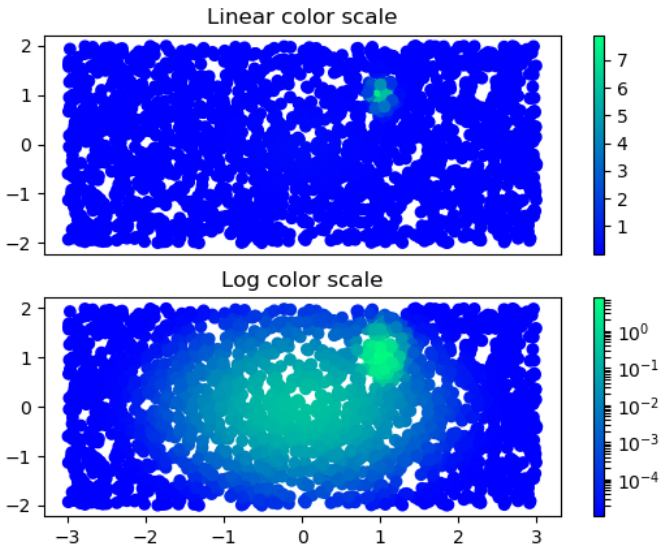

Figura 7.4: Resultado de correr `plot_scatter.py`.

La figura muestra la suma de dos montículos gaussianos, uno con un pico

muy alto. Si la escala de colores de `z` es lineal, la figura muestra sólo un pico. En cambio, si se usa una escala de colores logarítmica (`norm=cm.LogNorm()`), si se puede observar mucho mejor la tendencia de los dos montículos.

```
# plot_scatter.py

import matplotlib.colors as cm
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as ml

N = 2000
x = (np.random.rand(N)-0.5)*6
y = (np.random.rand(N)-0.5)*4

z = (ml.bivariate_normal(x, y, 0.1, 0.2, 1.0, 1.0)
     + 0.5 * ml.bivariate_normal(x, y, 0.5, 0.5, 0.0, 0.0))
z = z+1e-5

plt.subplot(212)
plt.scatter(x, y, c=z, norm=cm.LogNorm(), cmap='winter')
plt.colorbar()
plt.title('Log color scale')

plt.subplot(211)
plt.scatter(x, y, c=z, cmap='winter')
plt.colorbar()
frame1 = plt.gca()
frame1.axes.get_xaxis().set_ticks([])
plt.title('Linear color scale')
```

El código `plot_scatter.py` muestra el uso de `plt.scatter`, que en realidad es una forma de mostrar unos datos con tres dimensiones en plano 2D. Python tiene la opción para hacer figuras tipo `scatter` en 3D también, aunque su uso no lo recomiendo, porque no muestra los datos con facilidad.

Como último ejemplo, se muestra el uso de contornos o superficies en 2D/3D. Las figuras 7.5 y 7.6 muestran los mismos datos de la figura 7.4 pero que fueron tomados en una grilla constante y no aleatoria. Aunque esto no es lo que uno esperaría obtener en campo (es difícil hacer una grilla perfecta), se requeriría utilizar códigos para interpolación que se puede obtener en varios paquetes de Python (pero que está más allá de lo que se busca en este capítulo).

En la primera figura (7.5) se muestra los contornos para la variable `z`, con curvas de nivel marcadas, o con curvas de nivel rellenas con colores proporcionales a la altura. Note que en este caso se muestra `z` en escala logarítmica.

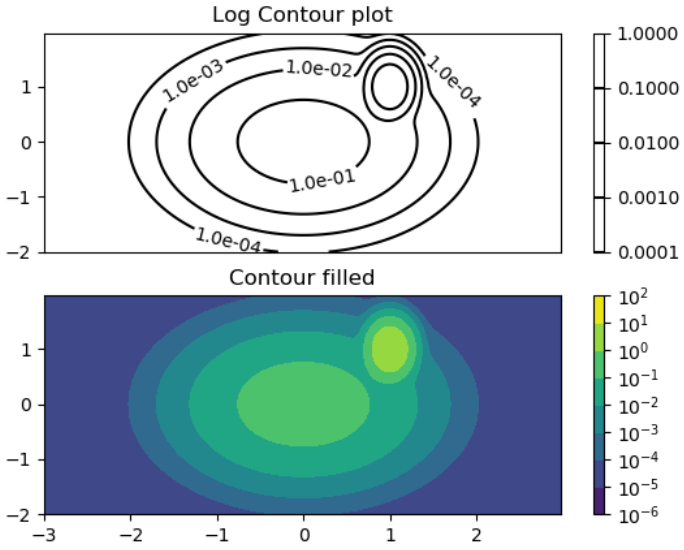

Figura 7.5: Figuras de contorno del programa `plot_contour.py`.

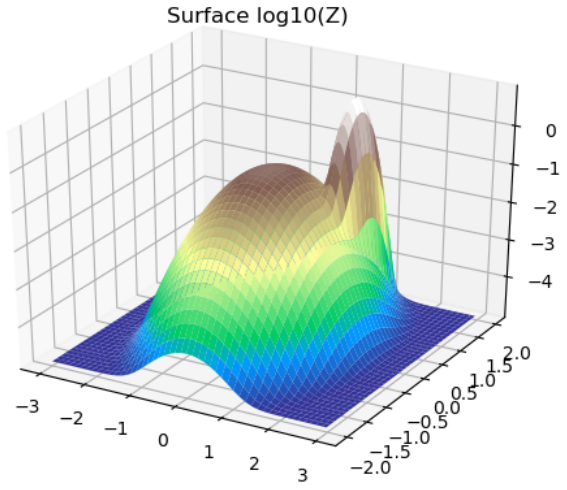

Figura 7.6: Figuras de superficie en 3D del programa `plot_contour.py`.

En la segunda figura (7.6) se muestran los mismos datos pero ahora en 3D, con colores para representar esa altura.

Para terminar, se recomienda visitar `https://matplotlib.org/examples/index.html` donde se muestran muchos ejemplos de cómo generar figuras con Python. En este curso no vamos a revisar todo, y se espera que Ud. aprenda y genere figuras de calidad.

```
# plot_contour.py

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.colors as cm
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as ml

delta = 0.025
x     = np.arange(-3.0, 3.0,delta)
y     = np.arange(-2.0, 2.0, delta)
X, Y  = np.meshgrid(x,y)

z = (ml.bivariate_normal(X, Y, 0.1, 0.2, 1.0, 1.0)
     + 0.5 * ml.bivariate_normal(X, Y, 0.5, 0.5, 0.0, 0.0))
z = z+1e-5

plt.figure(1)
plt.subplot(212)
CS = plt.contourf(X, Y, z,norm=cm.LogNorm())
plt.colorbar()
plt.title('Contour filled')

plt.subplot(211)
CS = plt.contour(X, Y, z,norm=cm.LogNorm(),colors='k')
plt.clabel(CS,inline=1,fontsize=10,fmt='%4.1e')
frame1 = plt.gca()
frame1.axes.get_xaxis().set_ticks([])
plt.colorbar()
plt.title('Log Contour plot')

fig  = plt.figure()
ax   = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, np.log10(z))
plt.title('Surface log10(Z)')
```

Problemas

7.1. Descargar el archivo `chap6_data.dat` y generar archivos formato `.png`

a) Graficar columna 1, contra columnas 2 y 3, en figuras separadas.

b) Graficar columna 1, contra columnas 2 y 3, en la misma figura.

c) Graficar columnas 4 contra 5, como nube de puntos, de color rojo, y columnas 4 y 6 como nube de puntos de otro color. Interpretar lo que se observa.

7.2. Utilizando la ecuación del movimiento parabólico,

$$\begin{aligned} x &= v_0 * t * cos(\theta) \\ y &= v_0 * t * sin(\theta) - \frac{1}{2} g * t^2 \end{aligned}$$

donde t es el tiempo, v_0 es la velocidad inicial, g es la aceleración de la gravedad (9.8 m/s^2) y (x, y) la posición en el tiempo t, y θ es el ángulo de disparo con respecto a la horizontal.

Genere dos figuras de dos disparos con diferente ángulo, mostrando en linea punteada el movimiento parabólico de cada uno y en forma de puntos (o triángulos) la posición (x, y) cada tiempo t (por ejemplo cada segundo). Es decir, no graficar *todos* los puntos, solo cada 5 o 10 puntos.