EBook Gratis

APRENDIZAJE matplotlib

Free unaffiliated eBook created from
Stack Overflow contributors.

#matplotlib

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: matplotlib

It is an unofficial and free matplotlib ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official matplotlib.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con matplotlib

Observaciones

Visión general

matplotlib es una biblioteca de trazado para Python. Proporciona API orientadas a objetos para incrustar gráficos en aplicaciones. Es similar a MATLAB en capacidad y sintaxis.

Fue escrito originalmente por JDHunter y se está desarrollando activamente. Se distribuye bajo una licencia BSD-Style.

Versiones

Versión	Versiones de Python compatibles	Observaciones	Fecha de lanzamiento
1.3.1	2.6, 2.7, 3.x	Versión estable más antigua	2013-10-10
1.4.3	2.6, 2.7, 3.x	Versión estable anterior	2015-07-14
1.5.3	2.7, 3.x	Versión estable actual	2016-01-11
2.x	2.7, 3.x	Última versión de desarrollo	2016-07-25

Examples

Instalación y configuración

Existen varias formas de instalar matplotlib, algunas de las cuales dependerán del sistema que esté utilizando. Si tiene suerte, podrá usar un administrador de paquetes para instalar fácilmente el módulo matplotlib y sus dependencias.

Windows

En las máquinas con Windows puede intentar usar el administrador de paquetes pip para instalar matplotlib. Consulte aquí para obtener información sobre la configuración de pip en un entorno Windows.

OS X

Se recomienda que utilice el administrador de paquetes pip para instalar matplotlib. Si necesita instalar algunas de las bibliotecas que no son de Python en su sistema (por ejemplo, `libfreetype`), considere usar Homebrew .

Si no puede usar pip por cualquier motivo, intente instalar desde la fuente .

Linux

Lo ideal sería utilizar el administrador de paquetes del sistema o pip para instalar matplotlib, ya sea instalando el paquete `python-matplotlib` o ejecutando `pip install matplotlib` .

Si esto no es posible (por ejemplo, no tiene privilegios de sudo en la máquina que está usando), entonces puede instalar desde la fuente usando la opción `--user` : `python setup.py install --user` . Normalmente, esto instalará matplotlib en `~/.local` .

Debian / Ubuntu

```
sudo apt-get install python-matplotlib
```

Fedora / Red Hat

```
sudo yum install python-matplotlib
```

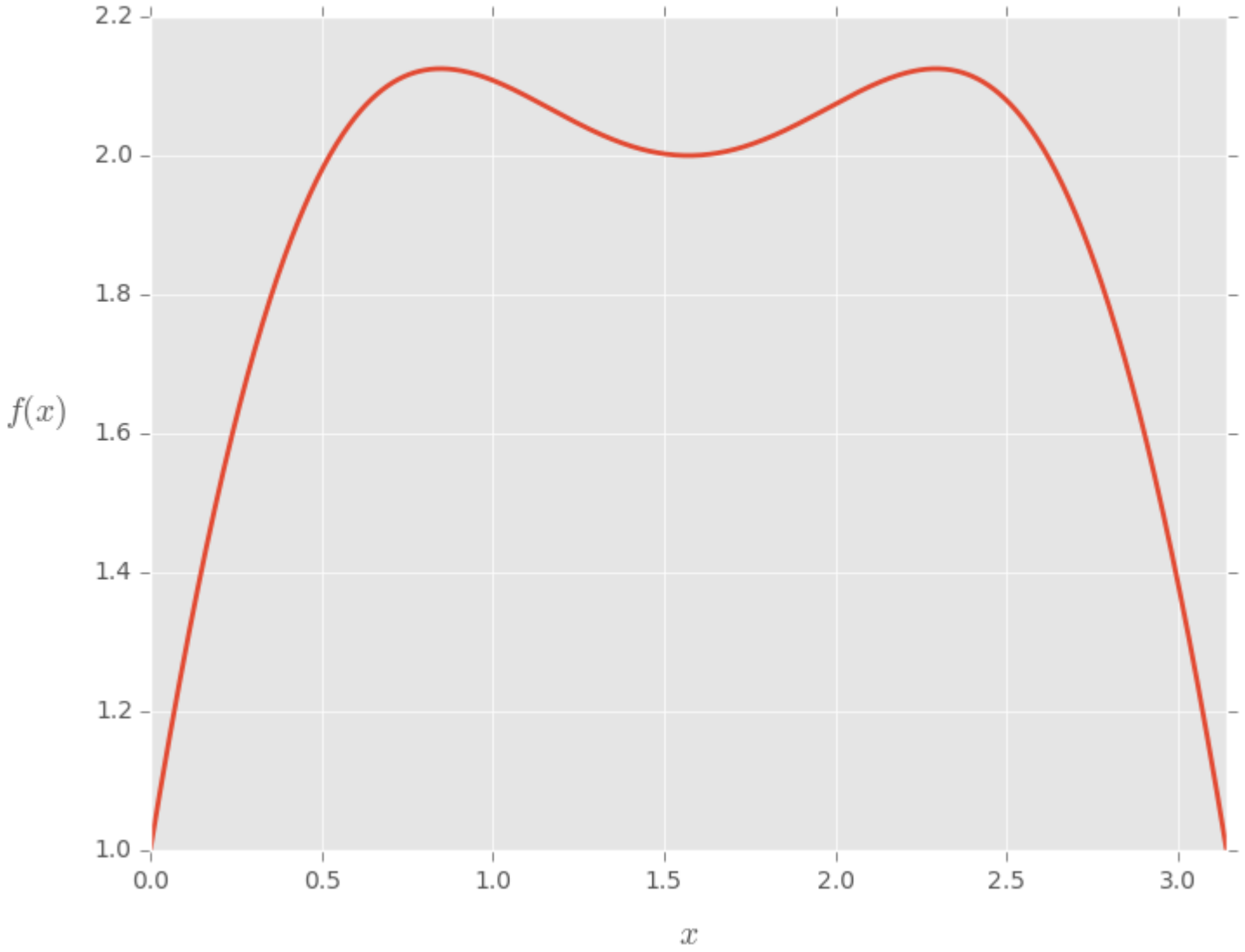
Solución de problemas

Consulte el sitio web de matplotlib para obtener consejos sobre cómo reparar un matplotlib roto.

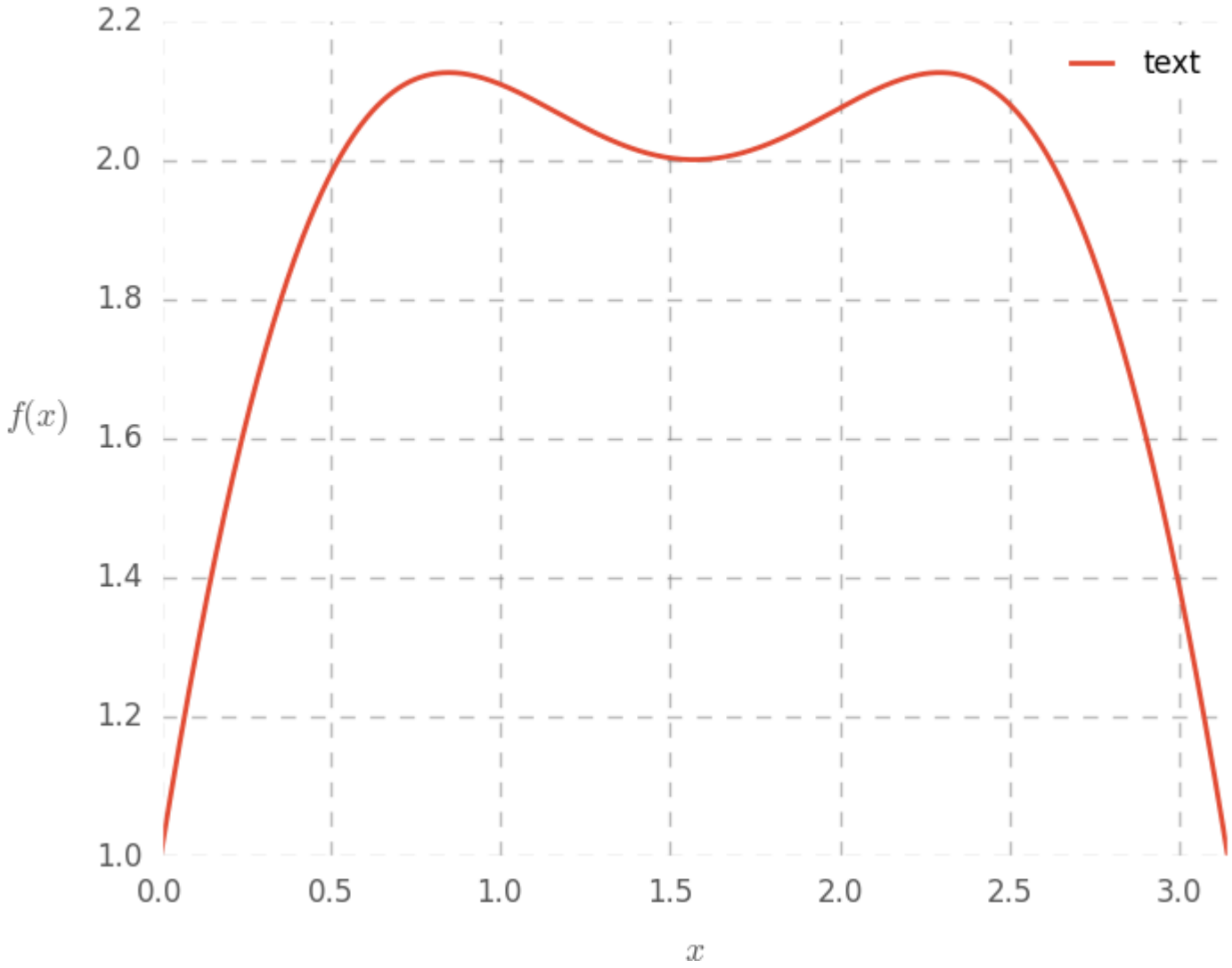
Personalizando un gráfico de matplotlib

```
import pylab as plt
import numpy as np

plt.style.use('ggplot')

fig = plt.figure(1)
ax = plt.gca()

# make some testing data
x = np.linspace( 0, np.pi, 1000 )
test_f = lambda x: np.sin(x)*3 + np.cos(2*x)

# plot the test data
ax.plot( x, test_f(x) , lw = 2)

# set the axis labels
```

```
ax.set_xlabel(r'$x$', fontsize=14, labelpad=10)
ax.set_ylabel(r'$f(x)$', fontsize=14, labelpad=25, rotation=0)

# set axis limits
ax.set_xlim(0,np.pi)

plt.draw()
```




```
# Customize the plot
ax.grid(1, ls='--', color='#777777', alpha=0.5, lw=1)
ax.tick_params(labelsize=12, length=0)
ax.set_axis_bgcolor('w')
# add a legend
leg = plt.legend( ['text'], loc=1 )
fr = leg.get_frame()
fr.set_facecolor('w')
fr.set_alpha(.7)
plt.draw()
```



Sintaxis imperativa vs orientada a objetos

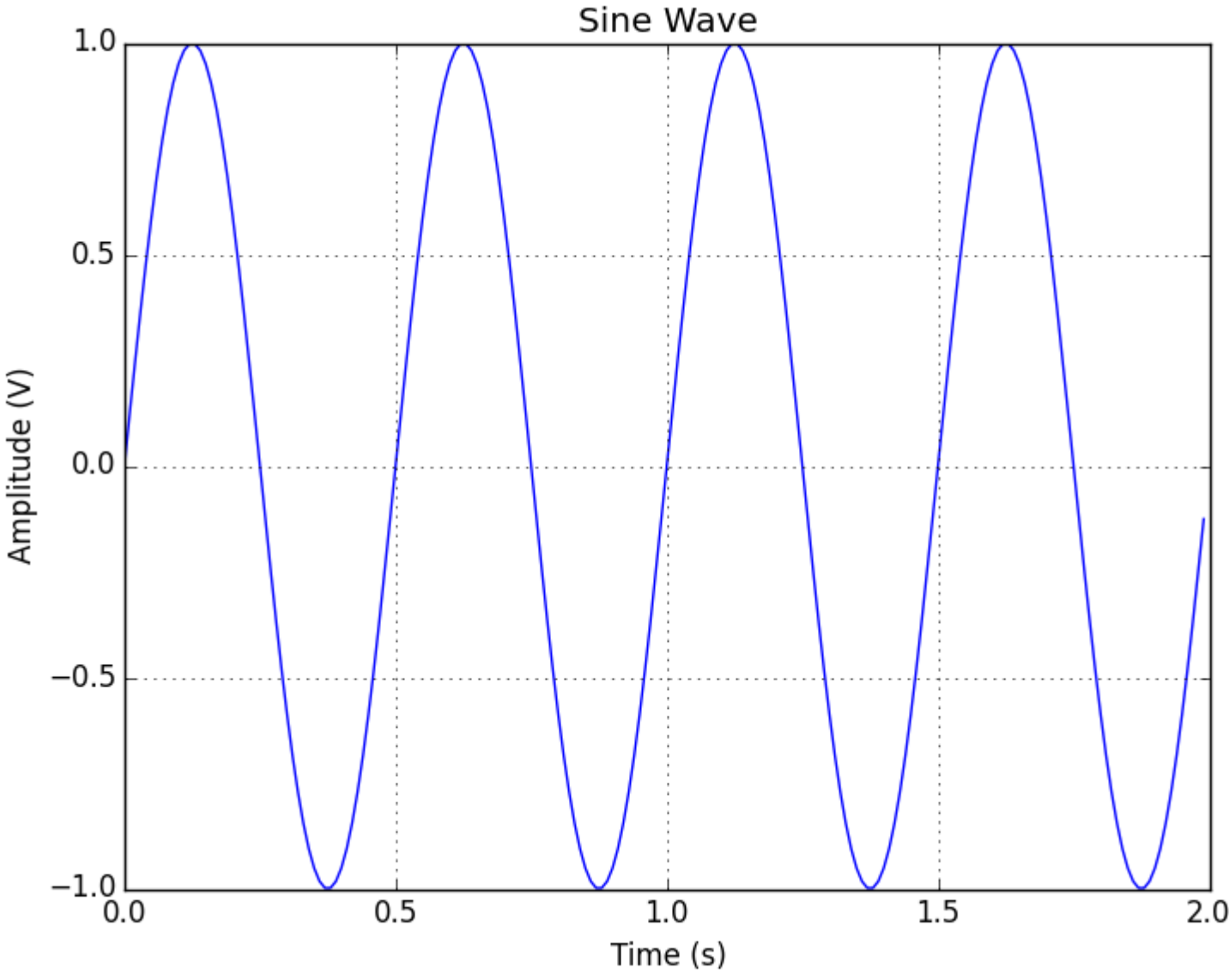
Matplotlib admite sintaxis tanto orientada a objetos como imperativa para el trazado. La sintaxis imperativa está diseñada intencionalmente para estar muy cerca de la sintaxis de Matlab.

La sintaxis imperativa (a veces llamada sintaxis 'máquina de estado') emite una serie de comandos que actúan sobre la figura o el eje más reciente (como Matlab). La sintaxis orientada a objetos, por otra parte, actúa explícitamente sobre los objetos (figura, eje, etc.) de interés. Un punto clave en el zen de Python afirma que explícito es mejor que implícito, por lo que la sintaxis orientada a objetos es más pirónica. Sin embargo, la sintaxis imperativa es conveniente para los nuevos conversos de Matlab y para escribir pequeños guiones de argumento "desechables". A continuación se muestra un ejemplo de los dos estilos diferentes.

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0, 2, 0.01)
y = np.sin(4 * np.pi * t)

# Imperative syntax
```

```
plt.figure(1)
plt.clf()
plt.plot(t, y)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (V)')
plt.title('Sine Wave')
plt.grid(True)

# Object oriented syntax
fig = plt.figure(2)
fig.clf()
ax = fig.add_subplot(1,1,1)
ax.plot(t, y)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Amplitude (V)')
ax.set_title('Sine Wave')
ax.grid(True)
```

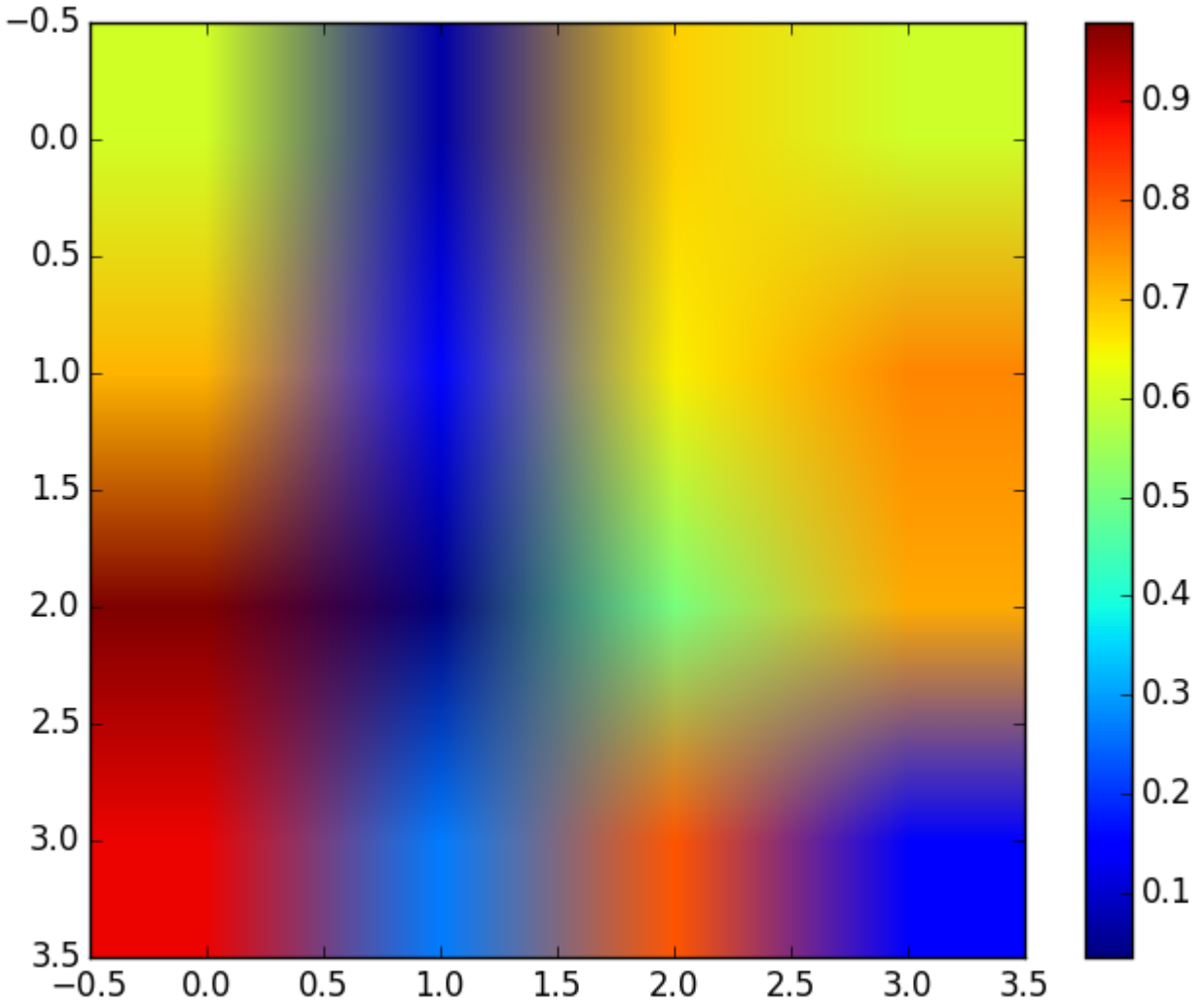
Ambos ejemplos producen la misma gráfica que se muestra a continuación.



Arreglos bidimensionales (2D)

Mostrar una matriz bidimensional (2D) en los ejes.

```
import numpy as np
from matplotlib.pyplot import imshow, show, colorbar

image = np.random.rand(4,4)
imshow(image)
colorbar()
show()
```



Lea Empezando con matplotlib en línea: https://riptutorial.com/es/matplotlib/topic/881/empezando-con-matplotlib

Capítulo 2: Animaciones y tramas interactivas.

Introducción

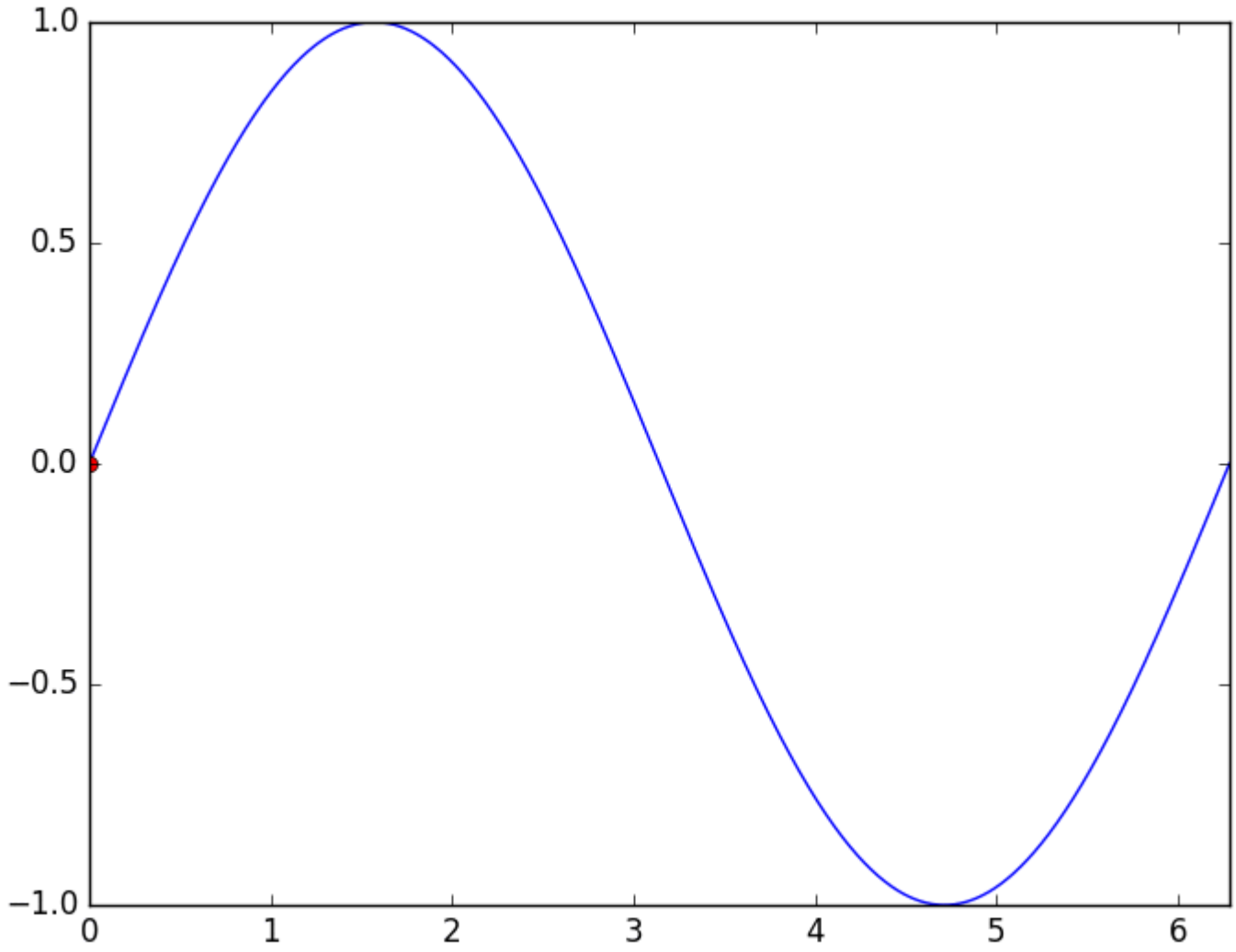
Con python matplotlib puedes hacer correctamente gráficos animados.

Examples

Animación básica con función de animación.

El paquete matplotlib.animation ofrece algunas clases para crear animaciones. `FuncAnimation` crea animaciones llamando repetidamente a una función. Aquí usamos una función `animate()` que cambia las coordenadas de un punto en el gráfico de una función sinusoidal.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
s = np.sin(t)
l = plt.plot(t, s)

ax = plt.axis([0,TWOPI,-1,1])

redDot, = plt.plot([0], [np.sin(0)], 'ro')

def animate(i):
    redDot.set_data(i, np.sin(i))
    return redDot,

# create animation using the animate() function
myAnimation = animation.FuncAnimation(fig, animate, frames=np.arange(0.0, TWOPI, 0.1), \
                                      interval=10, blit=True, repeat=True)

plt.show()
```



Guarda la animación en gif

En este ejemplo se utiliza el `save` método para guardar una `Animation` de objetos mediante ImageMagick.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import rcParams

# make sure the full paths for ImageMagick and ffmpeg are configured
rcParams['animation.convert_path'] = r'C:\Program Files\ImageMagick\convert'
rcParams['animation.ffmpeg_path'] = r'C:\Program Files\ffmpeg\bin\ffmpeg.exe'

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
s = np.sin(t)
l = plt.plot(t, s)
```

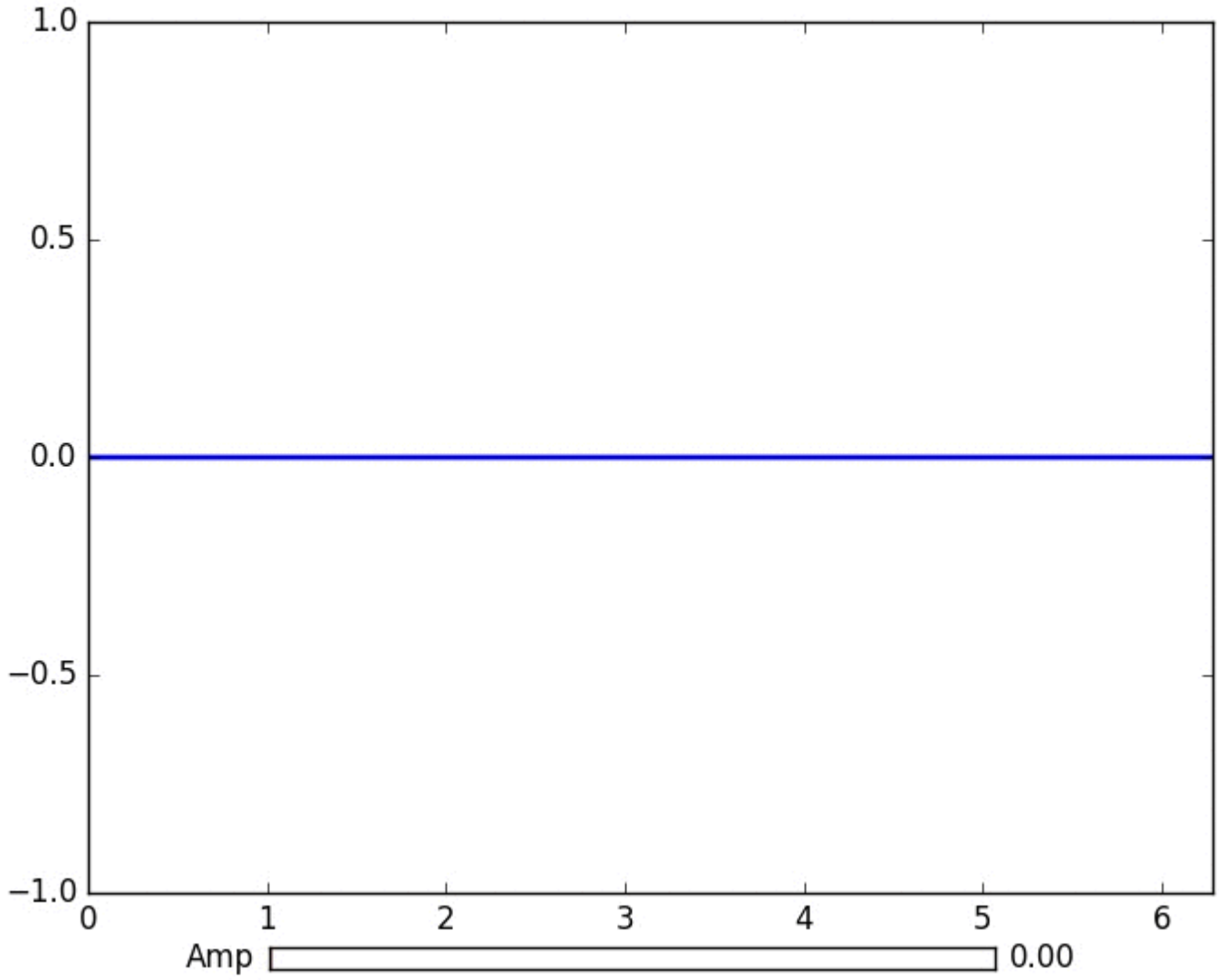
```
ax = plt.axis([0,TWOPI,-1,1])

redDot, = plt.plot([0], [np.sin(0)], 'ro')

def animate(i):
    redDot.set_data(i, np.sin(i))
    return redDot,

# create animation using the animate() function with no repeat
myAnimation = animation.FuncAnimation(fig, animate, frames=np.arange(0.0, TWOPI, 0.1), \
                                      interval=10, blit=True, repeat=False)

# save animation at 30 frames per second
myAnimation.save('myAnimation.gif', writer='imagemagick', fps=30)
```

Controles interactivos con matplotlib.widgets

Para interactuar con parcelas, Matplotlib ofrece widgets neutros de GUI. Los widgets requieren un objeto `matplotlib.axes.Axes` .

Aquí hay una demostración del widget deslizante que detalla la amplitud de una curva sinusoidal. La función de actualización es activada por el evento `on_changed()` del control deslizante.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Slider

TWOPI = 2*np.pi

fig, ax = plt.subplots()

t = np.arange(0.0, TWOPI, 0.001)
initial_amp = .5
s = initial_amp*np.sin(t)
l, = plt.plot(t, s, lw=2)

ax = plt.axis([0,TWOPI,-1,1])

axamp = plt.axes([0.25, .03, 0.50, 0.02])
# Slider
samp = Slider(axamp, 'Amp', 0, 1, valinit=initial_amp)

def update(val):
    # amp is the current value of the slider
    amp = samp.val
    # update curve
    l.set_ydata(amp*np.sin(t))
    # redraw canvas while idle
    fig.canvas.draw_idle()

# call update function on slider value change
samp.on_changed(update)

plt.show()
```



Otros widgets disponibles:

- AxesWidget
- Botón
- Botones de control
- Cursor
- ElipseSelector
- Lazo
- LassoSelector
- LockDraw
- MultiCursor
- Botones de radio
- RectangleSelector
- SpanSelector
- SubplotTool
- Manijas de herramientas

Trazar datos en vivo de la tubería con matplotlib

Esto puede ser útil cuando desea visualizar los datos entrantes en tiempo real. Estos datos podrían, por ejemplo, provenir de un microcontrolador que muestrea continuamente una señal analógica.

En este ejemplo, obtendremos nuestros datos de una canalización con nombre (también conocida como fifo). Para este ejemplo, los datos en la tubería deben ser números separados por caracteres de nueva línea, pero puede adaptarlos a su gusto.

Ejemplo de datos:

```
100
123.5
1589
```

Más información sobre tuberías con nombre.

También utilizaremos el tipo de datos deque, de las colecciones de la biblioteca estándar. Un objeto deque funciona bastante como una lista. Pero con un objeto deque es bastante fácil agregarle algo mientras se mantiene el objeto deque en una longitud fija. Esto nos permite mantener el eje x en una longitud fija en lugar de siempre crecer y aplastar la gráfica. Más información sobre objetos deque.

Elegir el backend correcto es vital para el rendimiento. Compruebe qué componentes internos funcionan en su sistema operativo y elija uno rápido. Para mí, solo qt4agg y el backend predeterminado funcionaron, pero el predeterminado fue demasiado lento. Más información sobre backends en matplotlib.

Este ejemplo se basa en el ejemplo matplotlib de trazar datos aleatorios .

Ninguno de los caracteres en este código está destinado a ser eliminado.

```
import matplotlib
import collections
#selecting the right backend, change qt4agg to your desired backend
matplotlib.use('qt4agg')
import matplotlib.pyplot as plt
import matplotlib.animation as animation

#command to open the pipe
datapipe = open('path to your pipe','r')

#amount of data to be displayed at once, this is the size of the x axis
#increasing this amount also makes plotting slightly slower
data_amount = 1000

#set the size of the deque object
datalist = collections.deque([0]*data_amount,data_amount)

#configure the graph itself
fig, ax = plt.subplots()
line, = ax.plot([0,]*data_amount)

#size of the y axis is set here
ax.set_ylim(0,256)
```

```
def update(data):
        line.set_ydata(data)
        return line,

def data_gen():
    while True:
        """
        We read two data points in at once, to improve speed
        You can read more at once to increase speed
        Or you can read just one at a time for improved animation smoothness
        data from the pipe comes in as a string,
        and is seperated with a newline character,
        which is why we use respectively eval and rstrip.
        """
        datalist.append(eval((datapipe.readline()).rstrip('\n')))
        datalist.append(eval((datapipe.readline()).rstrip('\n')))
        yield datalist

ani = animation.FuncAnimation(fig,update,data_gen,interval=0, blit=True)
plt.show()
```

Si su trama comienza a demorarse después de un tiempo, intente agregar más datos de datalist.append, para que se lean más líneas en cada fotograma. O elige un backend más rápido si puedes.

Esto funcionó con datos de 150Hz de una tubería en mi 1.7ghz i3 4005u.

Lea Animaciones y tramas interactivas. en línea:
https://riptutorial.com/es/matplotlib/topic/6983/animaciones-y-tramas-interactivas-

Capítulo 3: Cerrar una ventana de figura

Sintaxis

- plt.close () # cierra la figura activa actual
- plt.close (fig) # cierra la figura con el mango 'fig'
- plt.close (num) # cierra el número de cifra 'num'
- plt.close (nombre) # cierra la figura con la etiqueta 'nombre'
- plt.close ('all') # cierra todas las cifras

Examples

Cerrando la figura activa actual usando pyplot

La interfaz pyplot para `matplotlib` podría ser la forma más sencilla de cerrar una figura.

```
import matplotlib.pyplot as plt
plt.plot([0, 1], [0, 1])
plt.close()
```

Cerrar una figura específica usando plt.close ()

Una figura específica se puede cerrar manteniendo su asa.

```
import matplotlib.pyplot as plt

fig1 = plt.figure() # create first figure
plt.plot([0, 1], [0, 1])

fig2 = plt.figure() # create second figure
plt.plot([0, 1], [0, 1])

plt.close(fig1) # close first figure although second one is active
```

Lea Cerrar una ventana de figura en línea: https://riptutorial.com/es/matplotlib/topic/6628/cerrar-una-ventana-de-figura

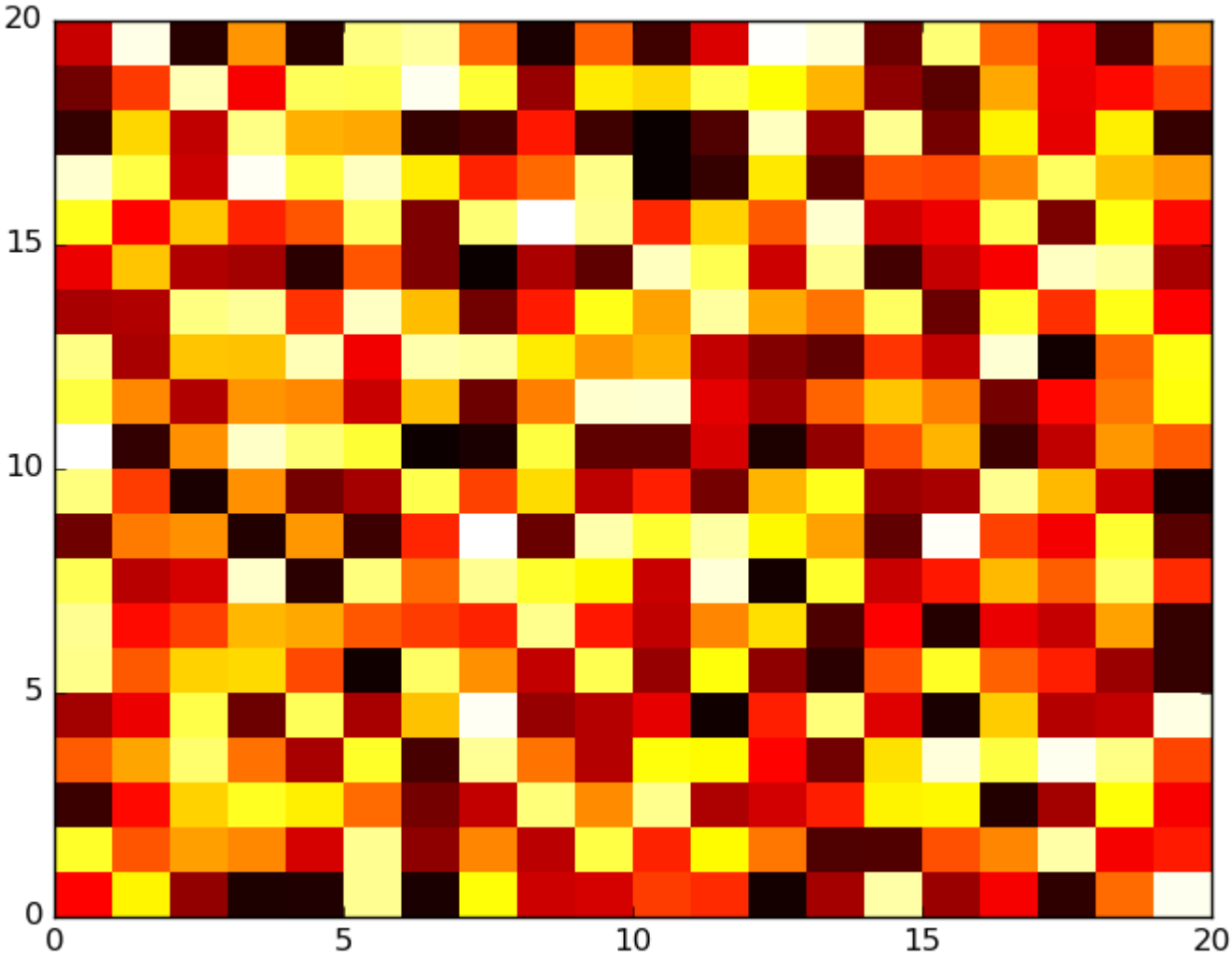
Capítulo 4: Colormaps

Examples

Uso básico

El uso de `pcolormesh` `contourf` integrados es tan simple como pasar el nombre del mapa de colores requerido (como se indica en la referencia de `pcolormesh` `contourf`) a la función de trazado (como `pcolormesh` o `contourf`) que lo espera, generalmente en la forma de un argumento de palabra clave `cmap` :

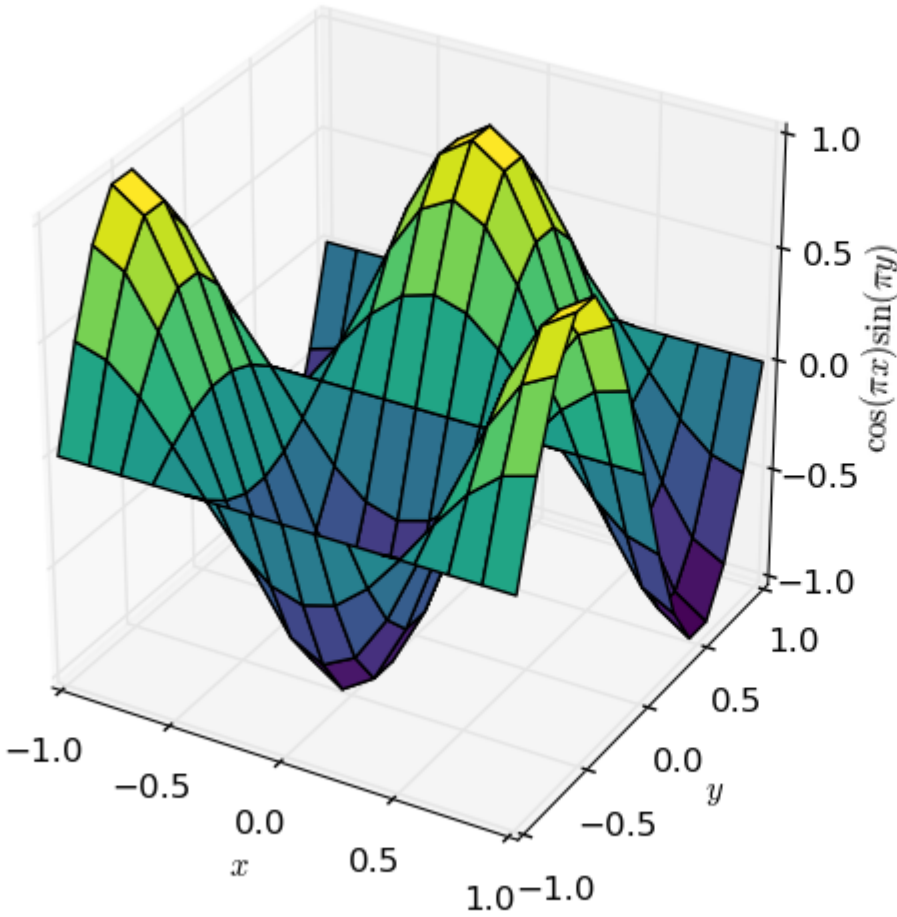
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()
plt.pcolormesh(np.random.rand(20,20),cmap='hot')
plt.show()
```



Los mapas de colores son especialmente útiles para visualizar datos tridimensionales en gráficos bidimensionales, pero un buen mapa de colores también puede hacer que un gráfico tridimensional adecuado sea mucho más claro:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import LinearLocator

# generate example data
import numpy as np
x,y = np.meshgrid(np.linspace(-1,1,15),np.linspace(-1,1,15))
z = np.cos(x*np.pi)*np.sin(y*np.pi)

# actual plotting example
fig = plt.figure()
ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(x,y,z,rstride=1,cstride=1,cmap='viridis')
ax2 = fig.add_subplot(122)
cf = ax2.contourf(x,y,z,51,vmin=-1,vmax=1,cmap='viridis')
cbar = fig.colorbar(cf)
cbar.locator = LinearLocator(numticks=11)
cbar.update_ticks()
```

```
for ax in {ax1, ax2}:
    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$y$')
    ax.set_xlim([-1,1])
    ax.set_ylim([-1,1])
    ax.set_aspect('equal')

ax1.set_zlim([-1,1])
ax1.set_zlabel(r'$\cos(\pi x) \sin(\p    i y)$')

plt.show()
```



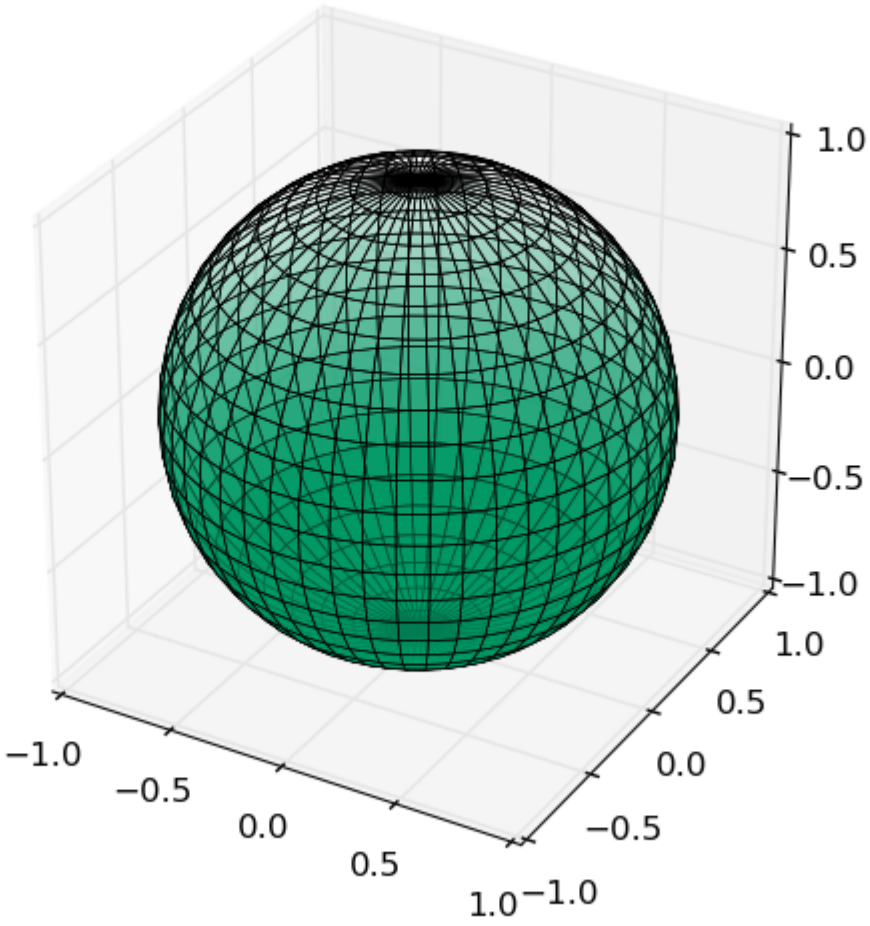
Usando colormaps personalizados

Además de los mapas de `'_r'` incorporados definidos en la referencia de mapas de colores (y sus mapas invertidos, con `'_r'` anexada a su nombre), también se pueden definir mapas de `'_r'` personalizados. La clave es el módulo `matplotlib.cm` .

El siguiente ejemplo define un mapa de `cm.register_cmap` muy simple que utiliza `cm.register_cmap` , que contiene un solo color, con la opacidad (valor alfa) del color que se interpola entre totalmente

opaco y completamente transparente en el rango de datos. Tenga en cuenta que las líneas importantes desde el punto de vista del mapa de colores son la importación de `cm` , la llamada a `register_cmap` y el paso del mapa de `plot_surface` a `plot_surface` .

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm

# generate data for sphere
from numpy import pi,meshgrid,linspace,sin,cos
th,ph = meshgrid(linspace(0,pi,25),linspace(0,2*pi,51))
x,y,z = sin(th)*cos(ph),sin(th)*sin(ph),cos(th)

# define custom colormap with fixed colour and alpha gradient
# use simple linear interpolation in the entire scale
cm.register_cmap(name='alpha_gradient',
                 data={'red':   [(0.,0,0),
                                 (1.,0,0)],

                       'green': [(0.,0.6,0.6),
                                 (1.,0.6,0.6)],

                       'blue':  [(0.,0.4,0.4),
                                 (1.,0.4,0.4)],

                       'alpha': [(0.,1,1),
                                 (1.,0,0)]})

# plot sphere with custom colormap; constrain mapping to between |z|=0.7 for enhanced effect
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z,cmap='alpha_gradient',vmin=-
0.7,vmax=0.7,rstride=1,cstride=1,linewidth=0.5,edgecolor='b')
ax.set_xlim([-1,1])
ax.set_ylim([-1,1])
ax.set_zlim([-1,1])
ax.set_aspect('equal')

plt.show()
```



En escenarios más complicados, uno puede definir una lista de valores R / G / B (/ A) en los que matplotlib interpola linealmente para determinar los colores utilizados en los gráficos correspondientes.

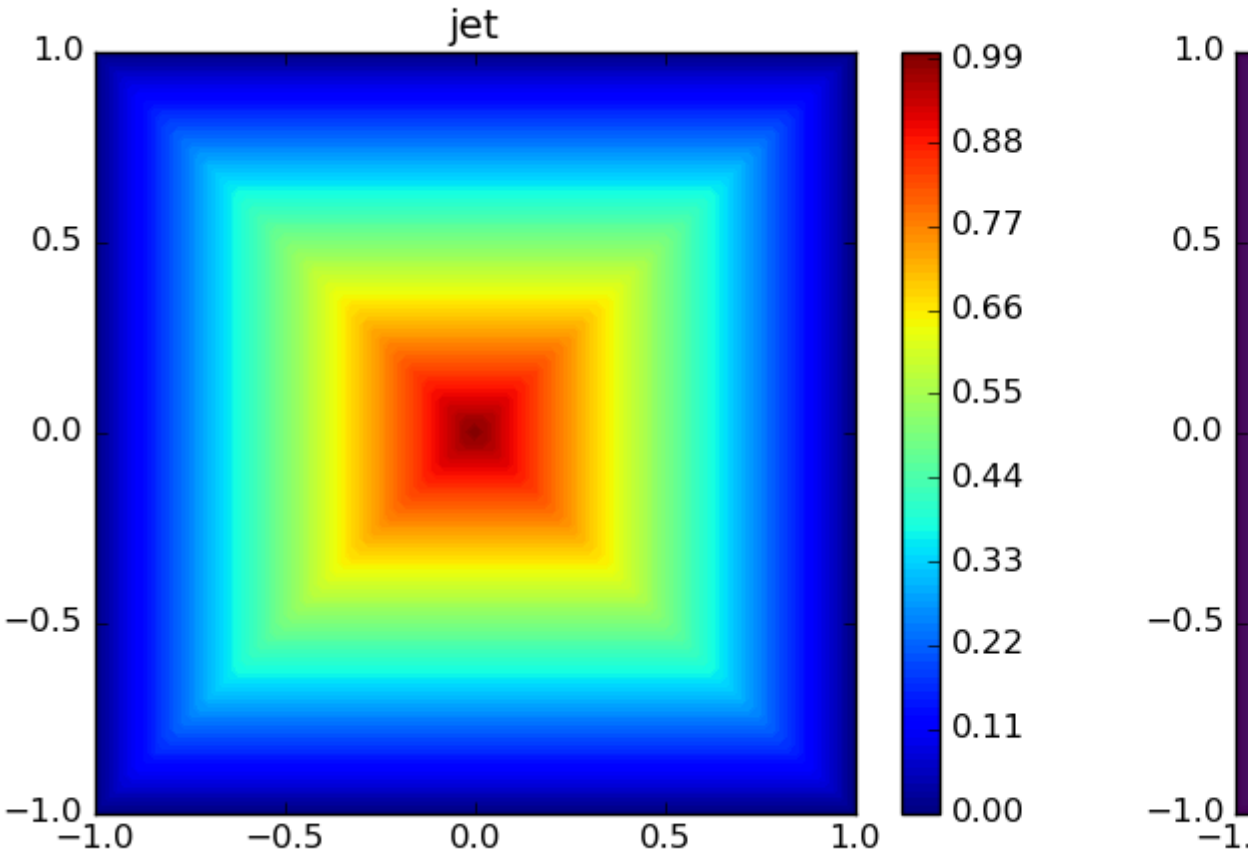
Colormaps perceptualmente uniformes

El mapa de colores predeterminado original de MATLAB (reemplazado en la versión R2014b) llamado `jet` es ubicuo debido a su alto contraste y familiaridad (y fue el valor predeterminado de matplotlib por razones de compatibilidad). A pesar de su popularidad, los mapas de colores tradicionales a menudo tienen deficiencias cuando se trata de representar datos con precisión. El cambio percibido en estos mapas de colores no corresponde a cambios en los datos; y una conversión del mapa de colores a escala de grises (por ejemplo, al imprimir una figura utilizando una impresora en blanco y negro) puede causar la pérdida de información.

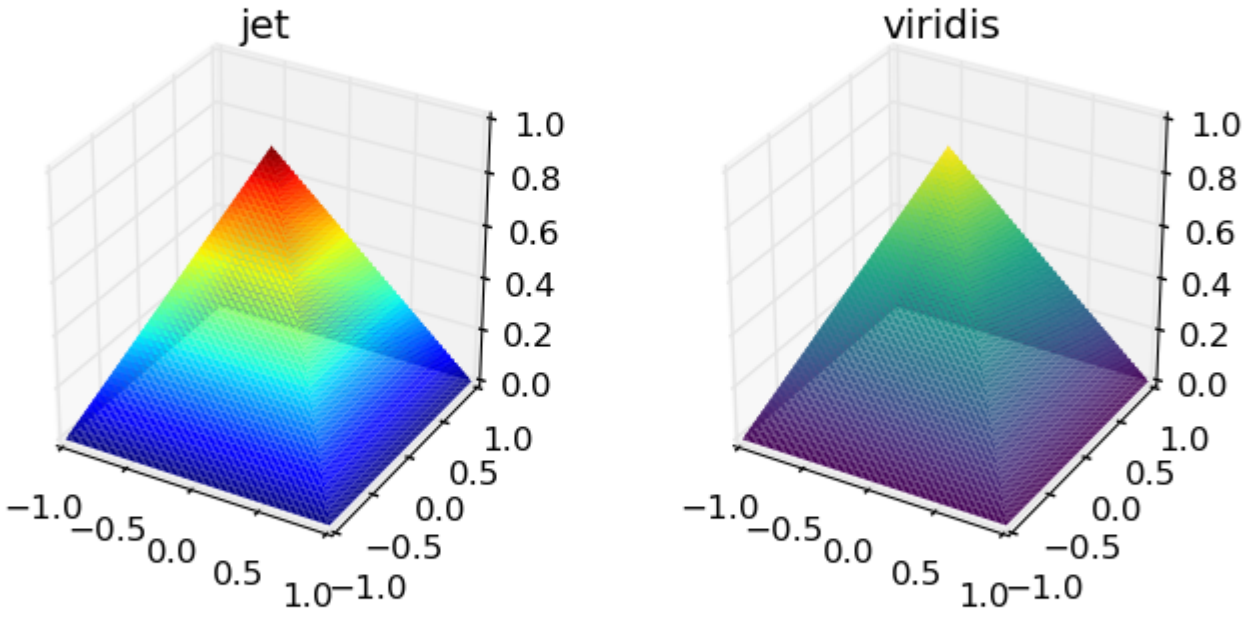
Se han introducido mapas de colores uniformes para que la visualización de los datos sea lo más precisa y accesible posible. Matplotlib introdujo cuatro nuevos mapas de color perceptualmente uniformes en la versión 1.5, con uno de ellos (llamado `viridis`) como predeterminado de la versión 2.0. Estos cuatro mapas de color (`viridis` , `inferno` , `plasma` y `magma`) son óptimos desde el

punto de vista de la percepción, y deben usarse para la visualización de datos por defecto, a menos que existan buenas razones para no hacerlo. Estos mapas de colores introducen el menor sesgo posible (al no crear funciones donde no hay ninguna para empezar), y son adecuados para una audiencia con una percepción de color reducida.

Como ejemplo para distorsionar visualmente los datos, considere las siguientes dos gráficas de vista superior de objetos similares a pirámides:



¿Cuál de los dos es una pirámide adecuada? La respuesta es, por supuesto, que ambos lo son, pero esto dista mucho de ser obvio de la trama utilizando el mapa de colores de `jet` :



Esta característica está en el núcleo de la uniformidad perceptiva.

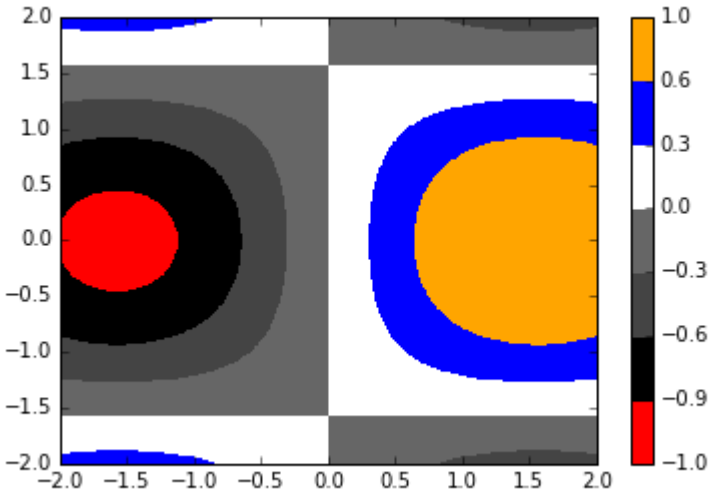
Mapa de colores discreto personalizado

Si tiene rangos predefinidos y desea usar colores específicos para esos rangos, puede declarar un mapa de colores personalizado. Por ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.colors

x = np.linspace(-2,2,500)
y = np.linspace(-2,2,500)
XX, YY = np.meshgrid(x, y)
Z = np.sin(XX) * np.cos(YY)

cmap = colors.ListedColormap(['red', '#000000','#444444', '#666666', '#ffffff', 'blue',
'orange'])
boundaries = [-1, -0.9, -0.6, -0.3, 0, 0.3, 0.6, 1]
norm = colors.BoundaryNorm(boundaries, cmap.N, clip=True)
```

```
plt.pcolormesh(x,y,Z, cmap=cmap, norm=norm)
plt.colorbar()
plt.show()
```

Produce



El color *i* se utilizará para los valores entre el límite *i* y *i + 1* . Los colores se pueden especificar por nombres (`'red'` , `'green'`), códigos HTML (`'#ffaa44'` , `'#441188'`) o tuplas RGB (`(0.2, 0.9, 0.45)`).

Lea Colormaps en línea: https://riptutorial.com/es/matplotlib/topic/3385/colormaps

Capítulo 5: Figuras y objetos de ejes

Examples

Creando una figura

La figura contiene todos los elementos de la trama. La forma principal de crear una figura en `matplotlib` es usar `pyplot` .

```
import matplotlib.pyplot as plt
fig = plt.figure()
```

Opcionalmente, puede proporcionar un número, que puede usar para acceder a una figura creada anteriormente. Si no se proporciona un número, la ID de la última figura creada se incrementará y se usará en su lugar; Las cifras se indexan a partir de 1, no 0.

```
import matplotlib.pyplot as plt
fig = plt.figure()
fig == plt.figure(1)  # True
```

En lugar de un número, las cifras también se pueden identificar por una cadena. Si utiliza un backend interactivo, esto también establecerá el título de la ventana.

```
import matplotlib.pyplot as plt
fig = plt.figure('image')
```

Elegir la figura usar

```
plt.figure(fig.number) # or
plt.figure(1)
```

Creando unos ejes

Hay dos formas principales de crear un eje en matplotlib: usar pyplot, o usar la API orientada a objetos.

Usando pyplot:

```
import matplotlib.pyplot as plt

ax = plt.subplot(3, 2, 1)  # 3 rows, 2 columns, the first subplot
```

Usando la API orientada a objetos:

```
import matplotlib.pyplot as plt

fig = plt.figure()
```

```
ax = fig.add_subplot(3, 2, 1)
```

La función de conveniencia `plt.subplots()` se puede usar para producir una figura y una colección de subparcelas en un comando:

```
import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(ncols=2, nrows=1)  # 1 row, 2 columns
```

Lea Figuras y objetos de ejes en línea: https://riptutorial.com/es/matplotlib/topic/2307/figuras-y-objetos-de-ejes

Capítulo 6: Gráficas de caja

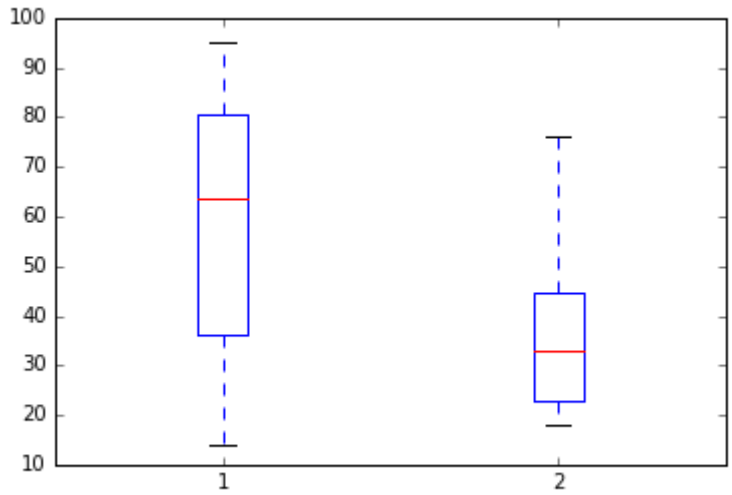
Examples

Cuadros de caja básicos

Los diagramas de caja son diagramas descriptivos que ayudan a comparar la distribución de diferentes series de datos. Son *descriptivos* porque muestran medidas (por ejemplo, la *mediana*) que no asumen una distribución de probabilidad subyacente.

El ejemplo más básico de un diagrama de caja en matplotlib se puede lograr simplemente pasando los datos como una lista de listas:

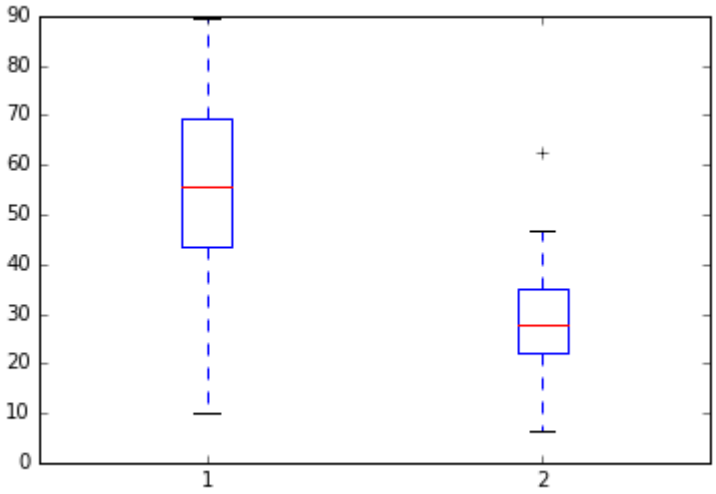
```
import matplotlib as plt

dataline1 = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
dataline2 = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
data = [ dataline1, dataline2 ]

plt.boxplot( data )
```



Sin embargo, es una práctica común utilizar matrices `numpy` como parámetros para los gráficos, ya que a menudo son el resultado de cálculos anteriores. Esto puede hacerse de la siguiente manera:

```
import numpy as np
import matplotlib as plt

np.random.seed(123)
dataline1 = np.random.normal( loc=50, scale=20, size=18 )
dataline2 = np.random.normal( loc=30, scale=10, size=18 )
data = np.stack( [ dataline1, dataline2 ], axis=1 )
```

```
plt.boxplot( data )
```



Lea Gráficas de caja en línea: https://riptutorial.com/es/matplotlib/topic/6086/graficas-de-caja

Capítulo 7: Gráficas de caja

Examples

Función boxplot

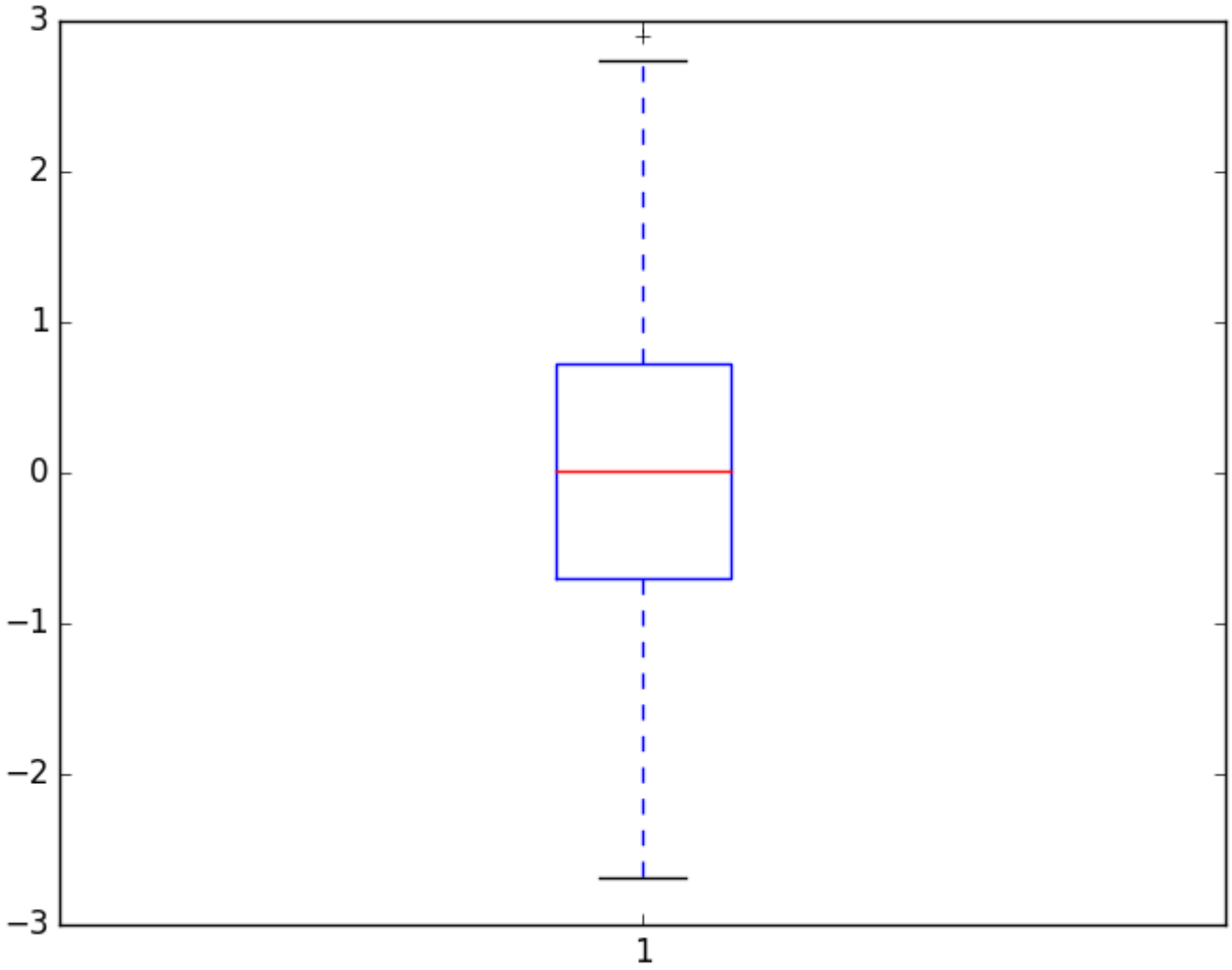
Matplotlib tiene su propia implementación de boxplot . Los aspectos relevantes de esta función es que, de forma predeterminada, el diagrama de caja muestra la mediana (percentil 50%) con una línea roja. La caja representa Q1 y Q3 (percentiles 25 y 75), y los bigotes dan una idea del rango de los datos (posiblemente en Q1 - 1.5 *IQR; Q3 + 1.5* IQR; siendo IQR el rango intercuartil, pero esto carece de confirmación). También tenga en cuenta que las muestras que se encuentran más allá de este rango se muestran como marcadores (se denominan volantes).

> **NOTA:** No todas las implementaciones de *boxplot* siguen las mismas reglas. Quizás el diagrama de caja de caja más común utiliza los bigotes para representar el mínimo y el máximo (haciendo que los volantes no existan). Observe también que esta trama es a veces llamado *diagrama de cajas y bigotes* y el *diagrama de caja y bigotes.*

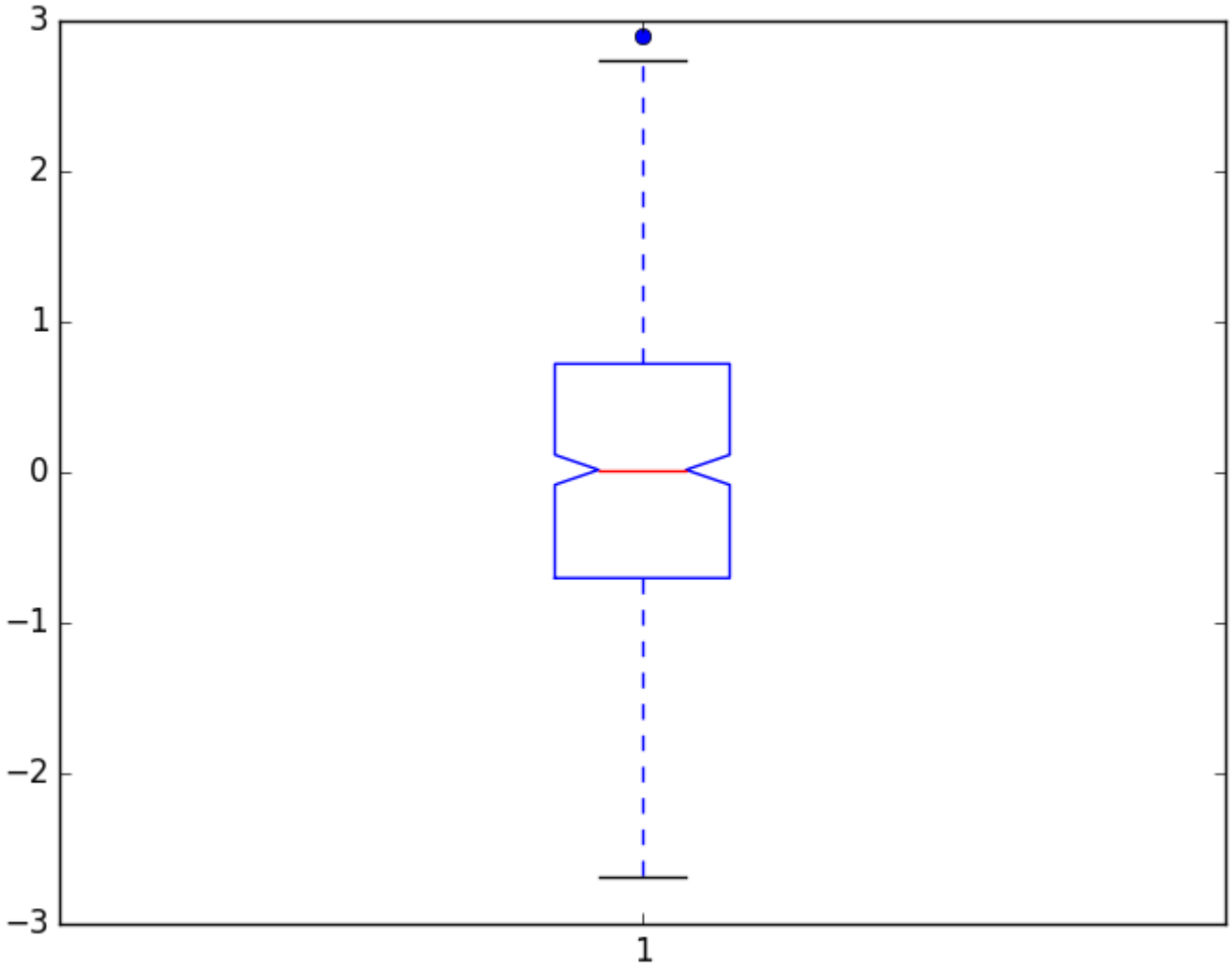
La siguiente receta muestra algunas de las cosas que puede hacer con la implementación matplotlib actual de boxplot:

```
import matplotlib.pyplot as plt
import numpy as np

X1 = np.random.normal(0, 1, 500)
X2 = np.random.normal(0.3, 1, 500)

# The most simple boxplot
plt.boxplot(X1)
plt.show()

# Changing some of its features
plt.boxplot(X1, notch=True, sym="o") # Use sym="" to shown no fliers; also showfliers=False
plt.show()

# Showing multiple boxplots on the same window
plt.boxplot((X1, X2), notch=True, sym="o", labels=["Set 1", "Set 2"])
plt.show()

# Hidding features of the boxplot
plt.boxplot(X2, notch=False, showfliers=False, showbox=False, showcaps=False, positions=[4],
labels=["Set 2"])
plt.show()

# Advanced customization of the boxplot
line_props = dict(color="r", alpha=0.3)
bbox_props = dict(color="g", alpha=0.9, linestyle="dashdot")
flier_props = dict(marker="o", markersize=17)
plt.boxplot(X1, notch=True, whiskerprops=line_props, boxprops=bbox_props,
flierprops=flier_props)
plt.show()
```

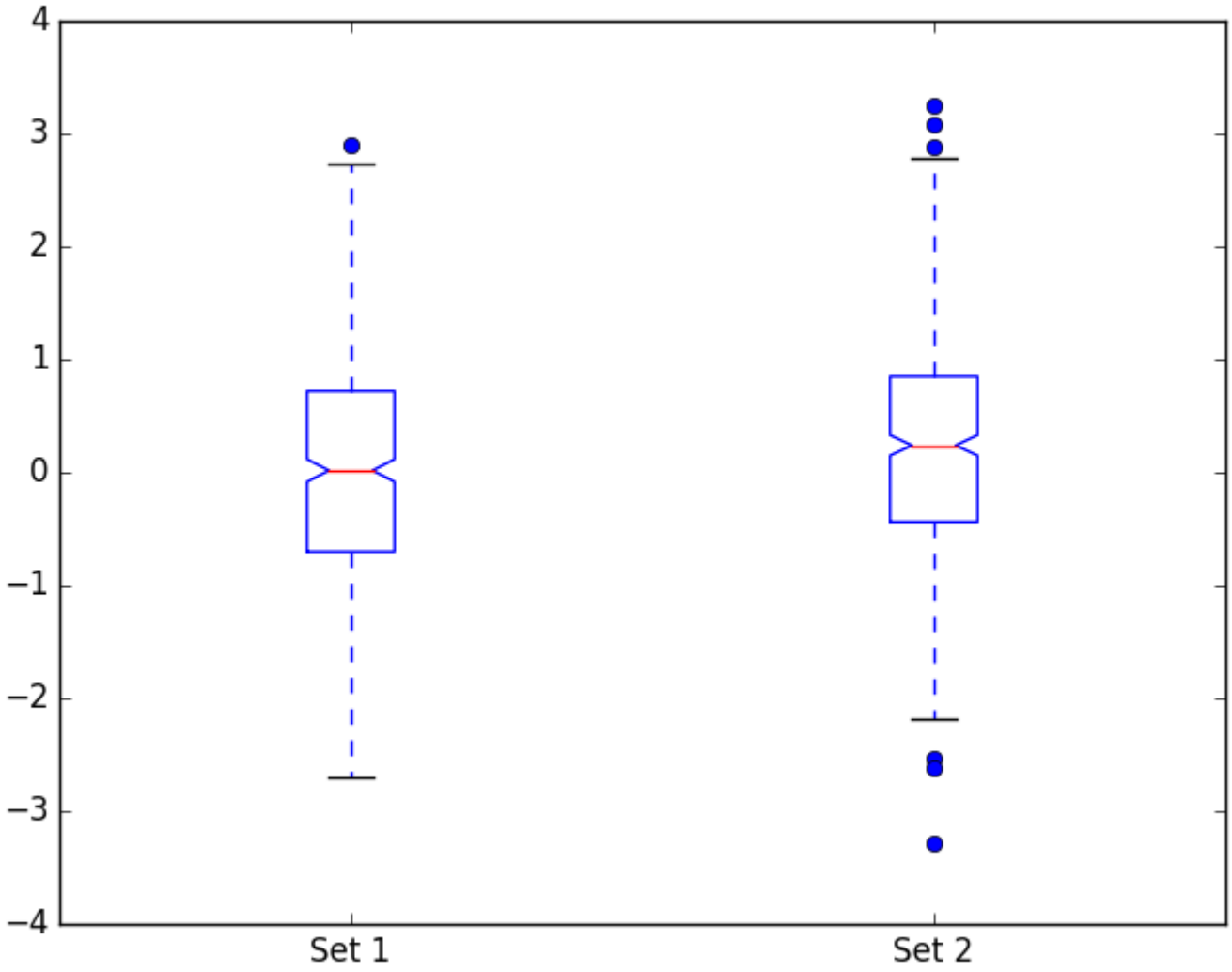
Este resultado en las siguientes parcelas:



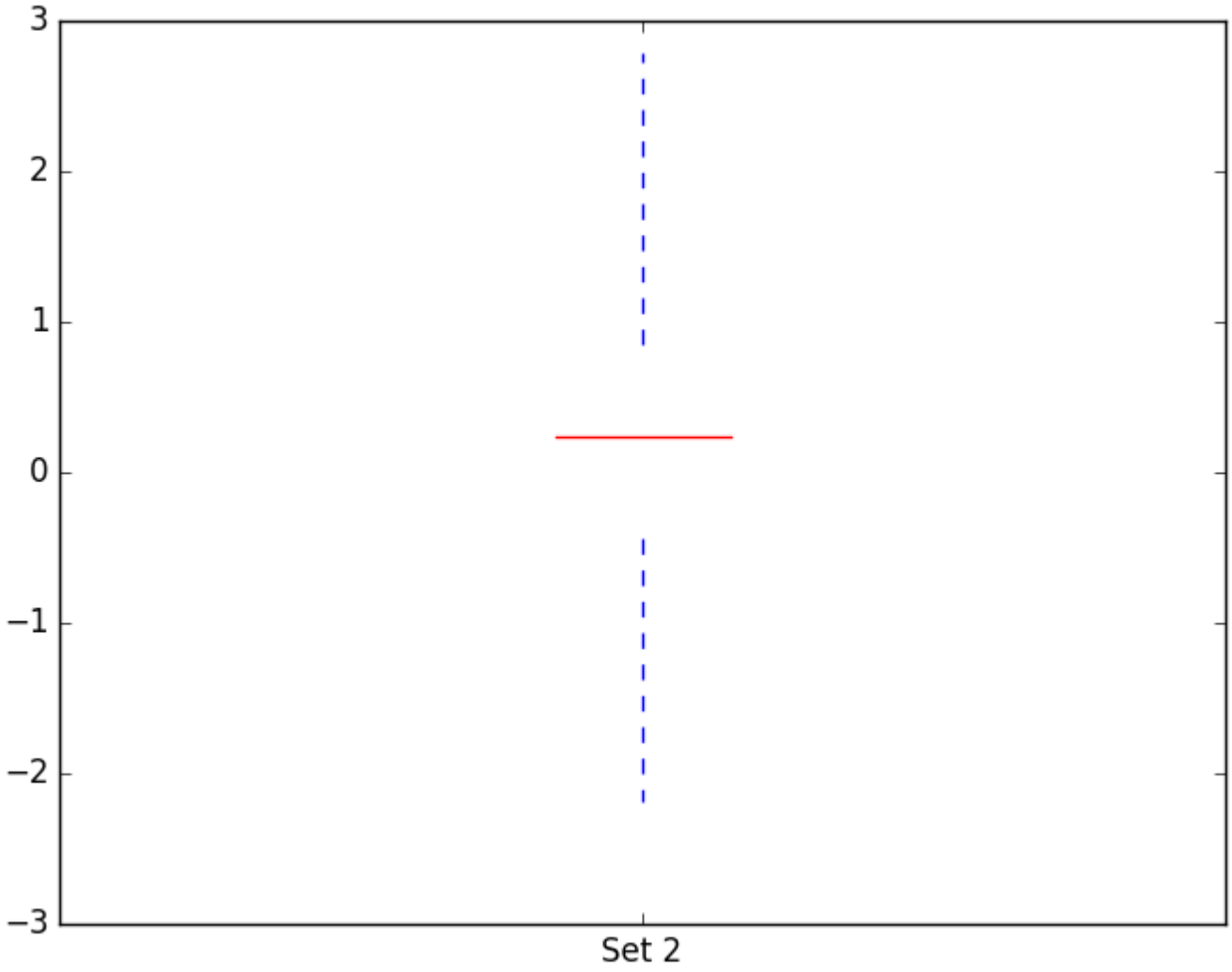
1. *Predeterminado matplotlib boxplot*



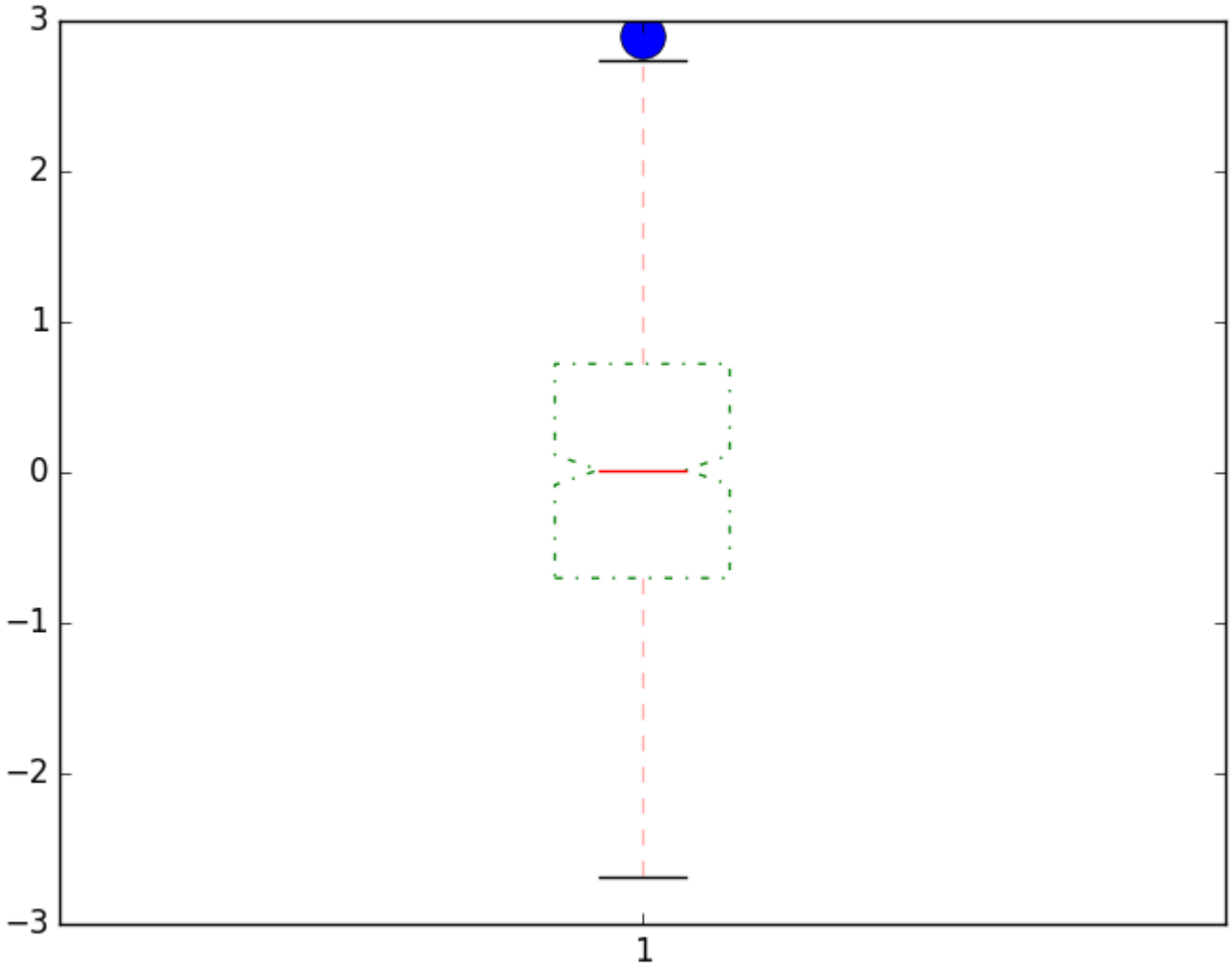
2. *Cambiando algunas características del diagrama de caja usando argumentos de función*



3. *Cuadro de caja múltiple en la misma ventana de gráfico*



4. *Ocultando algunas características del boxplot.*



5. *Personalización avanzada de un boxplot usando accesorios*

Si tiene la intención de hacer algún tipo de personalización avanzada de su diagrama de caja debe saber que los apoyos de los diccionarios a construir (por ejemplo):

```
line_props = dict(color="r", alpha=0.3)
bbox_props = dict(color="g", alpha=0.9, linestyle="dashdot")
flier_props = dict(marker="o", markersize=17)
plt.boxplot(X1, notch=True, whiskerprops=line_props, boxprops=bbox_props,
flierprops=flier_props)
plt.show()
```

... referirse principalmente (si no todos) a los objetos de Line2D . Esto significa que solo los argumentos disponibles en esa clase son modificables. Notará la existencia de palabras clave como `whiskerprops` , `boxprops` , `flierprops` y `capprops` . Estos son los elementos que necesita para proporcionar un diccionario de accesorios para personalizarlo aún más.

> NOTA: una mayor personalización del diagrama de caja con esta implementación puede resultar difícil. En algunos casos, el uso de otros elementos de matplotlib como

parches para crear los propios gráficos de caja puede ser ventajoso (cambios considerables en el elemento de caja, por ejemplo).

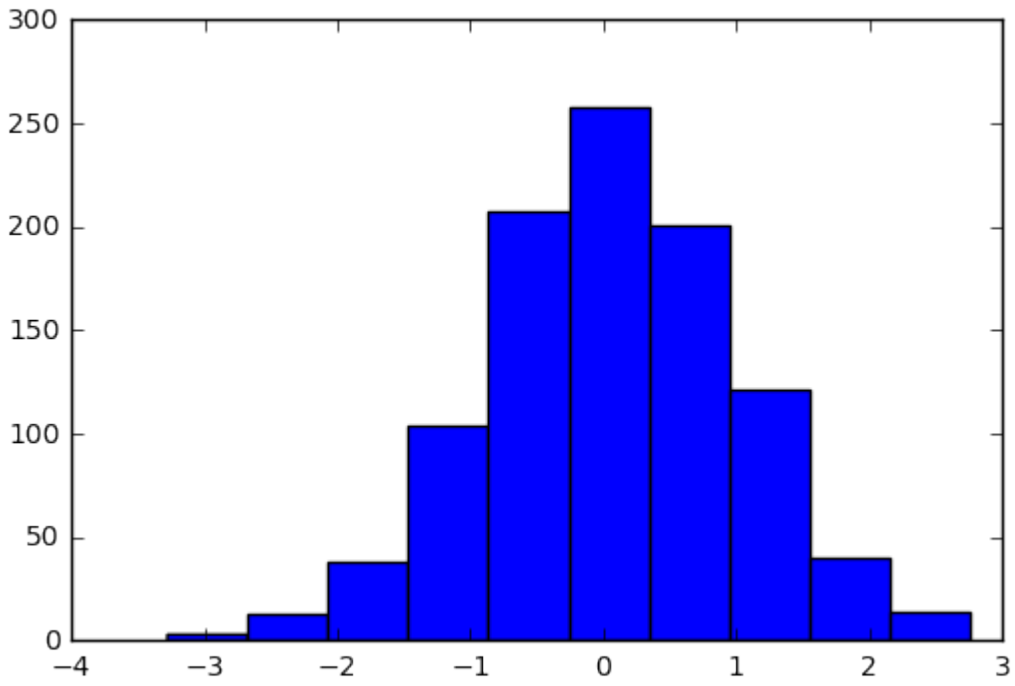
Lea Gráficas de caja en línea: https://riptutorial.com/es/matplotlib/topic/6368/graficas-de-caja

Capítulo 8: Histograma

Examples

Histograma simple

```
import matplotlib.pyplot as plt
import numpy as np

# generate 1000 data points with normal distribution
data = np.random.randn(1000)

plt.hist(data)

plt.show()
```



Lea Histograma en línea: https://riptutorial.com/es/matplotlib/topic/7329/histograma

Capítulo 9: Integración con TeX / LaTeX

Observaciones

- El soporte de LaTeX de Matplotlib requiere una instalación de LaTeX que funcione, dvipng (que puede incluirse con su instalación de LaTeX) y Ghostscript (se recomienda GPL Ghostscript 8.60 o posterior).
- El soporte pgf de Matplotlib requiere una instalación reciente de LaTeX que incluya los paquetes TikZ / PGF (como TeXLive), preferiblemente con XeLaTeX o LuaLaTeX instalado.

Examples

Insertando fórmulas TeX en parcelas

Las fórmulas TeX se pueden insertar en la gráfica usando la función `rc`

```
import matplotlib.pyplot as plt
plt.rc(usetex = True)
```

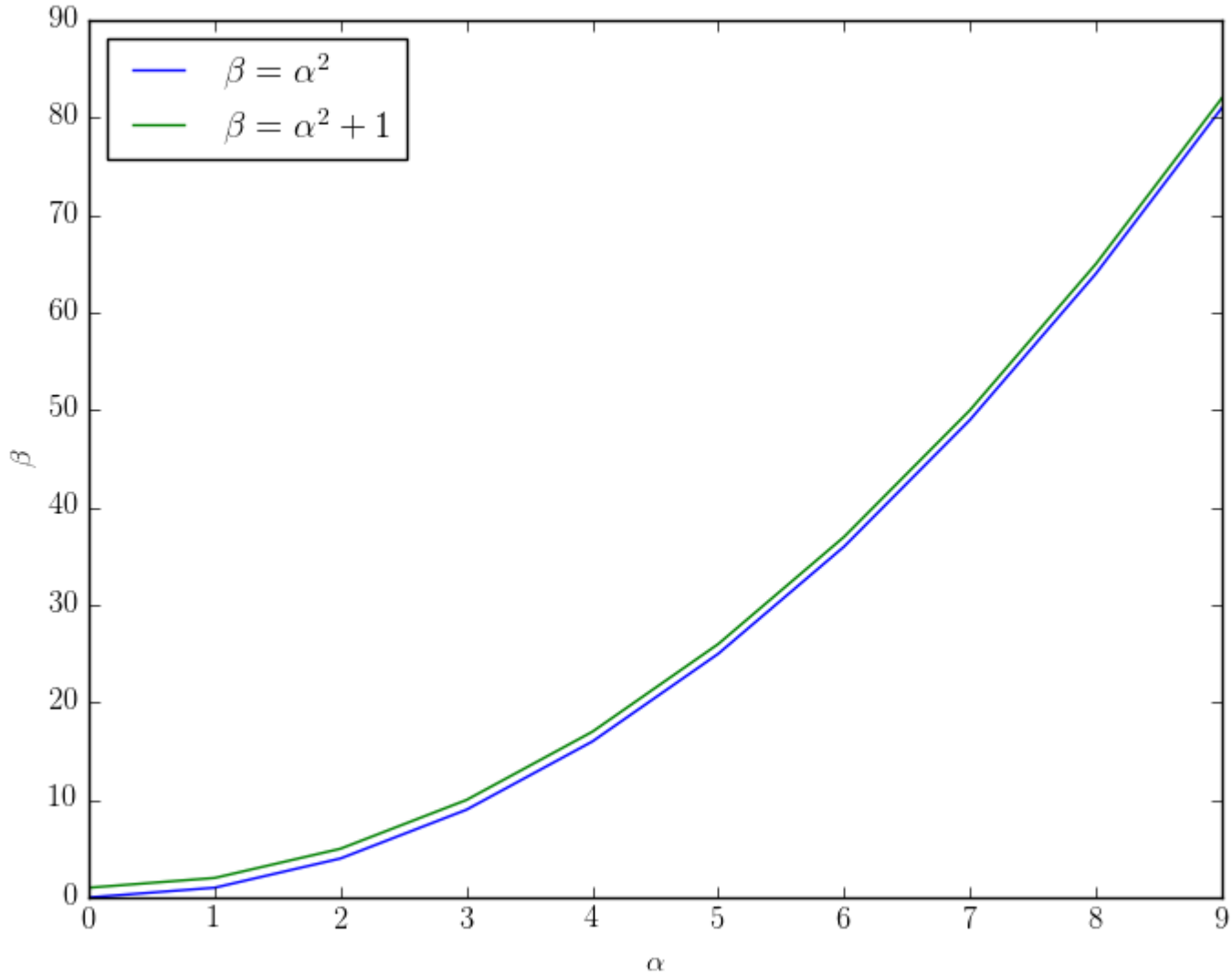
o accediendo a los `rcParams` :

```
import matplotlib.pyplot as plt
params = {'tex.usetex': True}
plt.rcParams.update(params)
```

TeX utiliza la barra invertida `\` para comandos y símbolos, que puede entrar en conflicto con caracteres especiales en las cadenas de Python. Para utilizar barras diagonales literales en una cadena de Python, deben ser evadidas o incorporadas en una cadena en bruto:

```
plt.xlabel('\\alpha')
plt.xlabel(r'\alpha')
```

La siguiente parcela



puede ser producido por el codigo

```
import matplotlib.pyplot as plt
plt.rc(usetex = True)
x = range(0,10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label = r'$\beta=\alpha^2$')
plt.plot(x, z, label = r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.show()
```

Las ecuaciones mostradas (como `$$...$$` o `\begin{equation}...\end{equation}`) no son compatibles. Sin embargo, el estilo matemático mostrado es posible con `\displaystyle` .

Para cargar paquetes de látex use el argumento `tex.latex.preamble` :

```
params = {'text.latex.preamble' : [r'\usepackage{siunitx}', r'\usepackage{amsmath}']}
plt.rcParams.update(params)
```

Tenga en cuenta, sin embargo, la advertencia en el archivo matplotlibrc de ejemplo :

```
#text.latex.preamble : # IMPROPER USE OF THIS FEATURE WILL LEAD TO LATEX FAILURES
                       # AND IS THEREFORE UNSUPPORTED. PLEASE DO NOT ASK FOR HELP
                       # IF THIS FEATURE DOES NOT DO WHAT YOU EXPECT IT TO.
                       # preamble is a comma separated list of LaTeX statements
                       # that are included in the LaTeX document preamble.
                       # An example:
                       # text.latex.preamble : \usepackage{bm},\usepackage{euler}
                       # The following packages are always loaded with usetex, so
                       # beware of package collisions: color, geometry, graphicx,
                       # type1cm, textcomp. Adobe Postscript (PSSNFS) font packages
                       # may also be loaded, depending on your font settings
```

Guardando y exportando parcelas que utilizan TeX.

Para incluir las parcelas creadas con matplotlib en documentos TeX, deben guardarse como archivos `pdf` o `eps` . De esta manera, cualquier texto en la trama (incluidas las fórmulas TeX) se representa como texto en el documento final.

```
import matplotlib.pyplot as plt
plt.rc(usetex=True)
x = range(0, 10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label=r'$\beta=\alpha^2$')
plt.plot(x, z, label=r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.savefig('my_pdf_plot.pdf')  # Saving plot to pdf file
plt.savefig('my_eps_plot.eps')  # Saving plot to eps file
```

Los diagramas en matplotlib se pueden exportar a código TeX utilizando el paquete de macros `pgf` para mostrar gráficos.

```
import matplotlib.pyplot as plt
plt.rc(usetex=True)
x = range(0, 10)
y = [t**2 for t in x]
z = [t**2+1 for t in x]
plt.plot(x, y, label=r'$\beta=\alpha^2$')
plt.plot(x, z, label=r'$\beta=\alpha^2+1$')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.legend(loc=0)
plt.savefig('my_pgf_plot.pgf')
```

Utilice el comando `rc` para cambiar el motor TeX utilizado

```
plt.rc('pgf', texsystem='pdflatex')  # or luatex, xelatex...
```

Para incluir la figura `.pgf` , escriba en su documento LaTeX

```
\usepackage{pgf}
\input{my_pgf_plot.pgf}
```

Lea Integración con TeX / LaTeX en línea:
https://riptutorial.com/es/matplotlib/topic/2962/integracion-con-tex---latex

Capítulo 10: Leyendas

Examples

Leyenda simple

Suponga que tiene varias líneas en la misma trama, cada una de un color diferente, y desea hacer una leyenda para decir qué representa cada línea. Puede hacer esto pasando una etiqueta a cada una de las líneas cuando llame a `plot()` , por ejemplo, la siguiente línea se etiquetará como *"Mi línea 1"* .

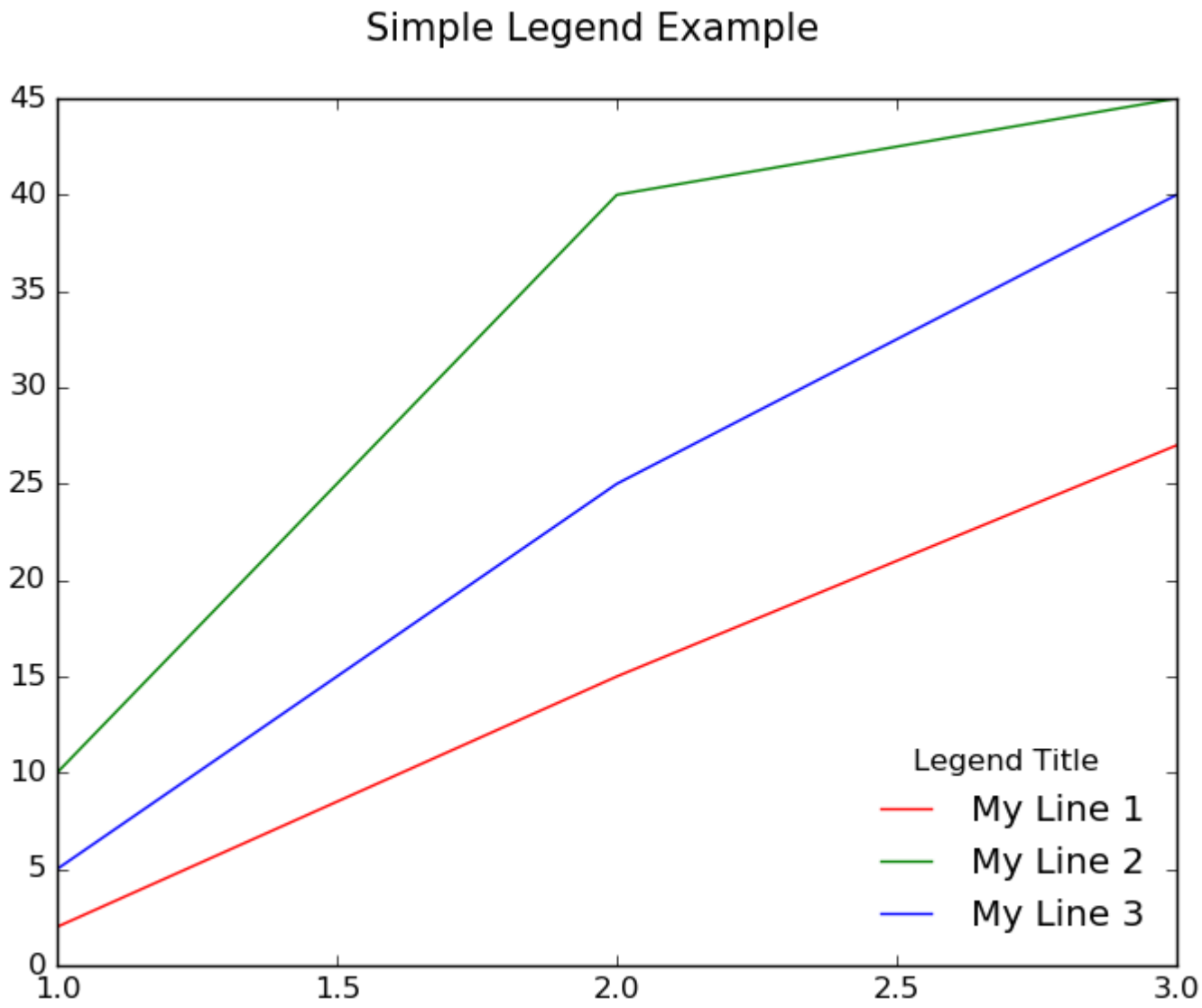
```
ax.plot(x, y1, color="red", label="My Line 1")
```

Esto especifica el texto que aparecerá en la leyenda para esa línea. Ahora para hacer visible la leyenda real, podemos llamar `ax.legend()`

Por defecto, creará una leyenda dentro de un cuadro en la esquina superior derecha de la parcela. Puede pasar argumentos a `legend()` para personalizarlo. Por ejemplo, podemos colocarlo en la esquina inferior derecha, sin un cuadro de marco que lo rodea, y crear un título para la leyenda llamando a lo siguiente:

```
ax.legend(loc="lower right", title="Legend Title", frameon=False)
```

A continuación se muestra un ejemplo:

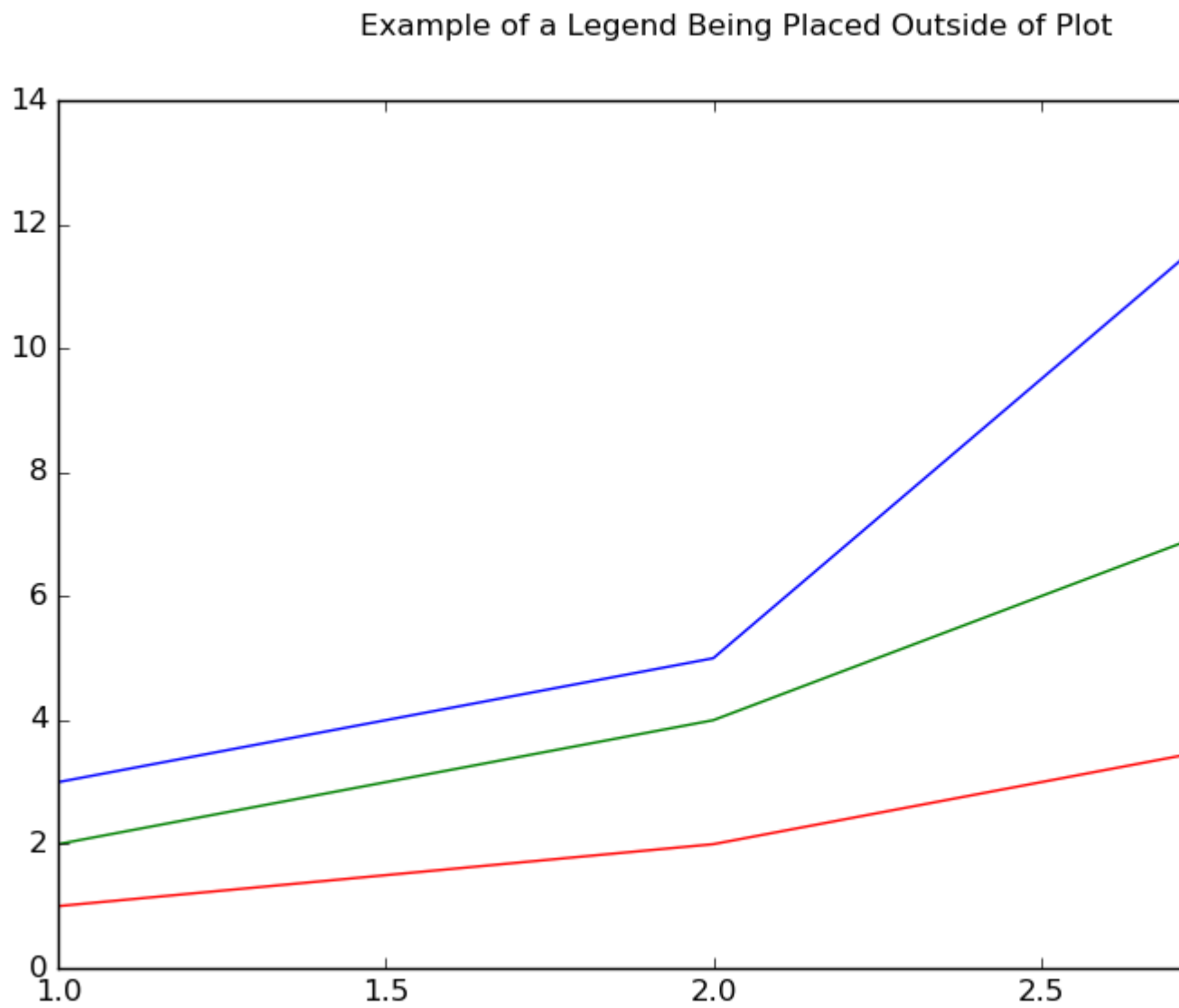



```
import matplotlib.pyplot as plt

# The data
x =  [1, 2, 3]
y1 = [2,  15, 27]
y2 = [10, 40, 45]
y3 = [5,  25, 40]

# Initialize the figure and axes
fig, ax = plt.subplots(1, figsize=(8, 6))

# Set the title for the figure
fig.suptitle('Simple Legend Example ', fontsize=15)

# Draw all the lines in the same plot, assigning a label for each one to be
# shown in the legend
ax.plot(x, y1, color="red", label="My Line 1")
ax.plot(x, y2, color="green", label="My Line 2")
ax.plot(x, y3, color="blue", label="My Line 3")

# Add a legend with title, position it on the lower right (loc) with no box framing (frameon)
ax.legend(loc="lower right", title="Legend Title", frameon=False)
```

```
# Show the plot
plt.show()
```

Leyenda colocada fuera de la trama

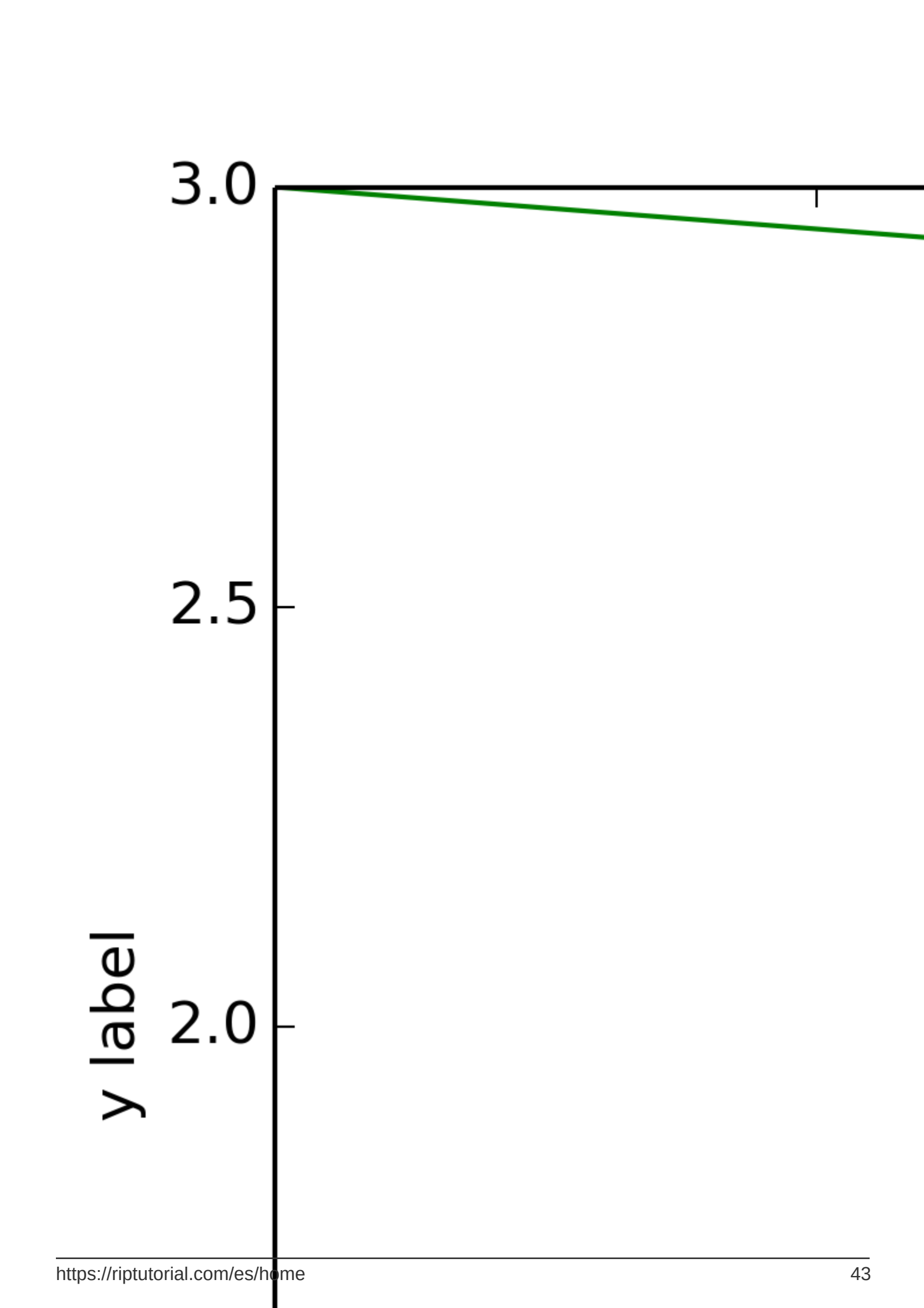
A veces es necesario o deseable colocar la leyenda fuera de la trama. El siguiente código muestra cómo hacerlo.




```
import matplotlib.pylab as plt
fig, ax = plt.subplots(1, 1, figsize=(10,6)) # make the figure with the size 10 x 6 inches
fig.suptitle('Example of a Legend Being Placed Outside of Plot')

# The data
x =  [1, 2, 3]
y1 = [1, 2, 4]
y2 = [2, 4, 8]
y3 = [3, 5, 14]

# Labels to use for each line
line_labels = ["Item A", "Item B", "Item C"]
```

```
# Create the lines, assigning different colors for each one.
# Also store the created line objects
l1 = ax.plot(x, y1, color="red")[0]
l2 = ax.plot(x, y2, color="green")[0]
l3 = ax.plot(x, y3, color="blue")[0]

fig.legend([l1, l2, l3],              # List of the line objects
           labels= line_labels,       # The labels for each line
           loc="center right",        # Position of the legend
           borderaxespad=0.1,         # Add little spacing around the legend box
           title="Legend Title")      # Title for the legend

# Adjust the scaling factor to fit your legend text completely outside the plot
# (smaller value results in more space being made for the legend)
plt.subplots_adjust(right=0.85)

plt.show()
```

Otra forma de colocar la leyenda fuera de la trama es usar `bbox_to_anchor` + `bbox_extra_artists` + `bbox_inches='tight'` , como se muestra en el siguiente ejemplo:


3.0
2.5
2.0
y label

lugar de crear una leyenda en el nivel de los *ejes* (lo que creará una leyenda independiente para cada subparcela). Esto se logra llamando a `fig.legend()` como se puede ver en el código para el siguiente código.

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(10,4))
fig.suptitle('Example of a Single Legend Shared Across Multiple Subplots')

# The data
x =  [1, 2, 3]
y1 = [1, 2, 3]
y2 = [3, 1, 3]
y3 = [1, 3, 1]
y4 = [2, 2, 3]

# Labels to use in the legend for each line
line_labels = ["Line A", "Line B", "Line C", "Line D"]

# Create the sub-plots, assigning a different color for each line.
# Also store the line objects created
l1 = ax1.plot(x, y1, color="red")[0]
l2 = ax2.plot(x, y2, color="green")[0]
l3 = ax3.plot(x, y3, color="blue")[0]
l4 = ax3.plot(x, y4, color="orange")[0] # A second line in the third subplot

# Create the legend
fig.legend([l1, l2, l3, l4],     # The line objects
           labels=line_labels,   # The labels for each line
           loc="center right",   # Position of legend
           borderaxespad=0.1,    # Small spacing around legend box
           title="Legend Title"  # Title for the legend
           )

# Adjust the scaling factor to fit your legend text completely outside the plot
# (smaller value results in more space being made for the legend)
plt.subplots_adjust(right=0.85)

plt.show()
```

Algo a tener en cuenta sobre el ejemplo anterior es lo siguiente:

```
l1 = ax1.plot(x, y1, color="red")[0]
```

Cuando se llama a `plot()` , devuelve una lista de objetos **line2D** . En este caso, solo devuelve una lista con un solo objeto *line2D* , que se extrae con la indexación `[0]` y se almacena en `l1` .

Una lista de todos los objetos de *line2D* que nos interesa incluir en la leyenda debe pasarse como primer argumento a `fig.legend()` . El segundo argumento de `fig.legend()` también es necesario. Se supone que es una lista de cadenas para usar como etiquetas para cada línea en la leyenda.

Los otros argumentos pasados a `fig.legend()` son puramente opcionales, y solo ayudan a afinar la estética de la leyenda.

Múltiples leyendas en los mismos ejes

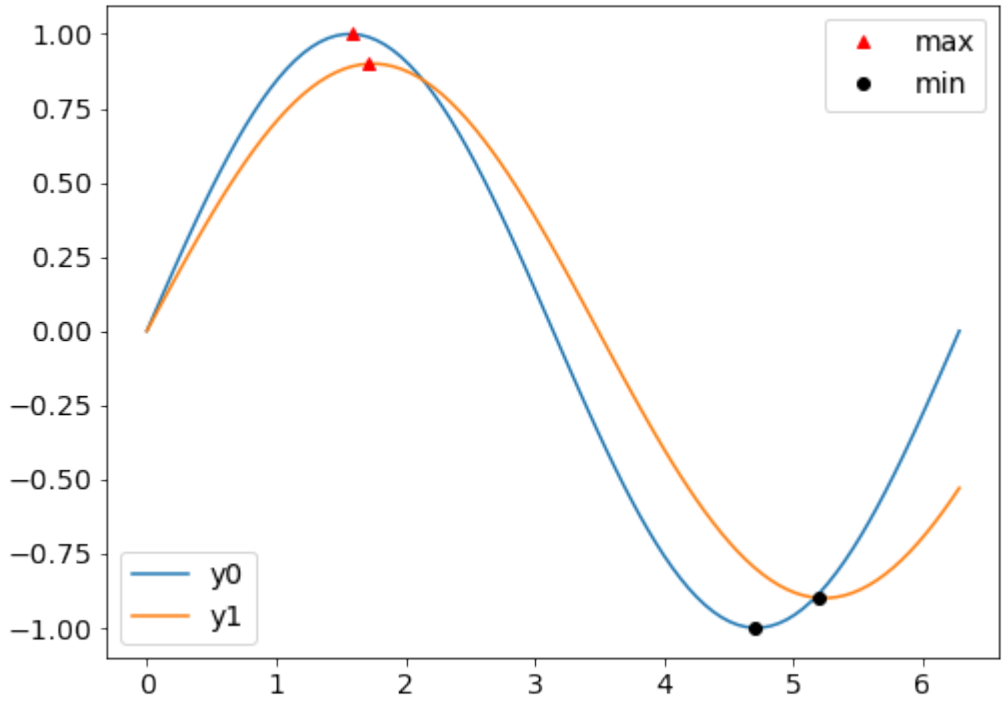
Si llama a `plt.legend()` o `ax.legend()` más de una vez, se eliminará la primera leyenda y se

dibujará una nueva. Según la documentación oficial :

> Esto se ha hecho para que sea posible llamar a legend () repetidamente para actualizar la leyenda a los últimos manejadores de los ejes.

Sin embargo, no se preocupe: todavía es bastante simple agregar una segunda leyenda (o la tercera, o la cuarta ...) a los ejes. En el ejemplo aquí, trazamos dos líneas, luego trazamos marcadores en sus máximos y mínimos respectivos. Una leyenda es para las líneas y la otra para los marcadores.

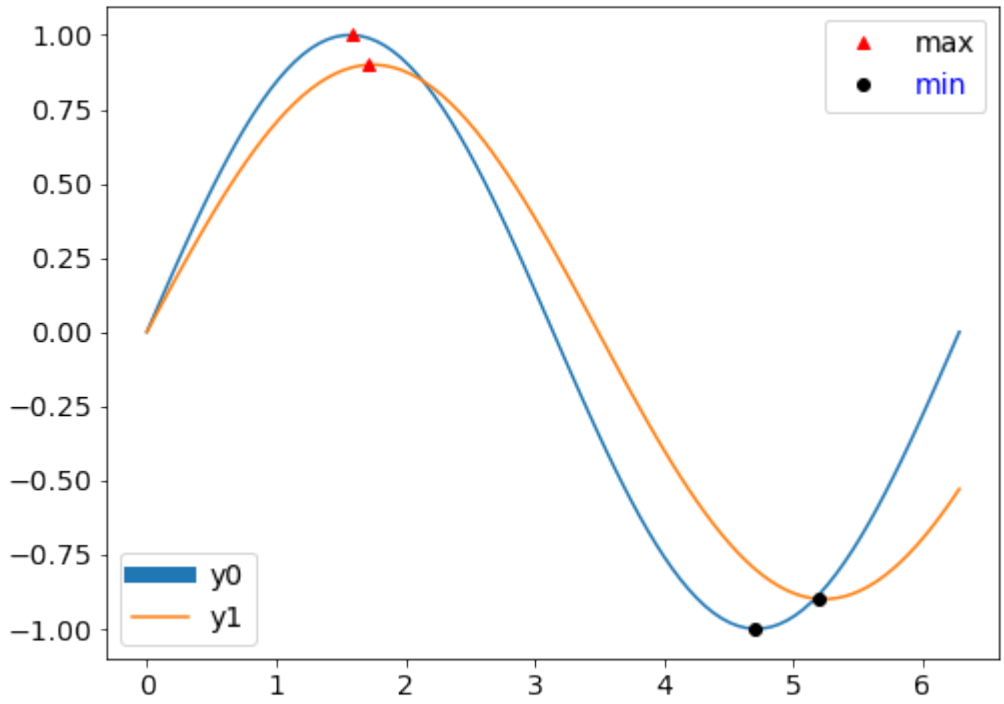
```
import matplotlib.pyplot as plt
import numpy as np

# Generate data for plotting:
x = np.linspace(0,2*np.pi,100)
y0 = np.sin(x)
y1 = .9*np.sin(.9*x)
# Find their maxima and minima and store
maxes = np.empty((2,2))
mins = np.empty((2,2))
for k,y in enumerate([y0,y1]):
    maxloc = y.argmax()
    maxes[k] = x[maxloc], y[maxloc]
    minloc = y.argmin()
    mins[k] = x[minloc], y[minloc]

# Instantiate figure and plot
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y0, label='y0')
ax.plot(x,y1, label='y1')
# Plot maxima and minima, and keep references to the lines
maxline, = ax.plot(maxes[:,0], maxes[:,1], 'r^')
minline, = ax.plot(mins[:,0], mins[:,1], 'ko')

# Add first legend:  only labeled data is included
leg1 = ax.legend(loc='lower left')
# Add second legend for the maxes and mins.
# leg1 will be removed from figure
leg2 = ax.legend([maxline,minline],['max','min'], loc='upper right')
# Manually add the first legend back
ax.add_artist(leg1)
```



La clave es asegurarse de que tiene referencias a los objetos de leyenda. El primero que `leg1` (`leg1`) se elimina de la figura cuando agregas el segundo, pero el objeto `leg1` todavía existe y se puede volver a `ax.add_artist` con `ax.add_artist` .

Lo realmente genial es que aún puedes manipular *ambas* leyendas. Por ejemplo, agregue lo siguiente al final del código anterior:

```
leg1.get_lines()[0].set_lw(8)
leg2.get_texts()[1].set_color('b')
```



Finalmente, vale la pena mencionar que en el ejemplo solo las líneas recibieron etiquetas cuando

se trazaron, lo que significa que `ax.legend()` agrega solo esas líneas al `leg1` . La leyenda para los marcadores (`leg2`), por lo tanto, requería las líneas y etiquetas como argumentos cuando se `leg2` instancias. Podríamos, alternativamente, haber dado etiquetas a los marcadores cuando se trazaron también. Pero entonces *ambas* llamadas a `ax.legend` habrían requerido algunos argumentos adicionales para que cada leyenda contuviera solo los elementos que queríamos.

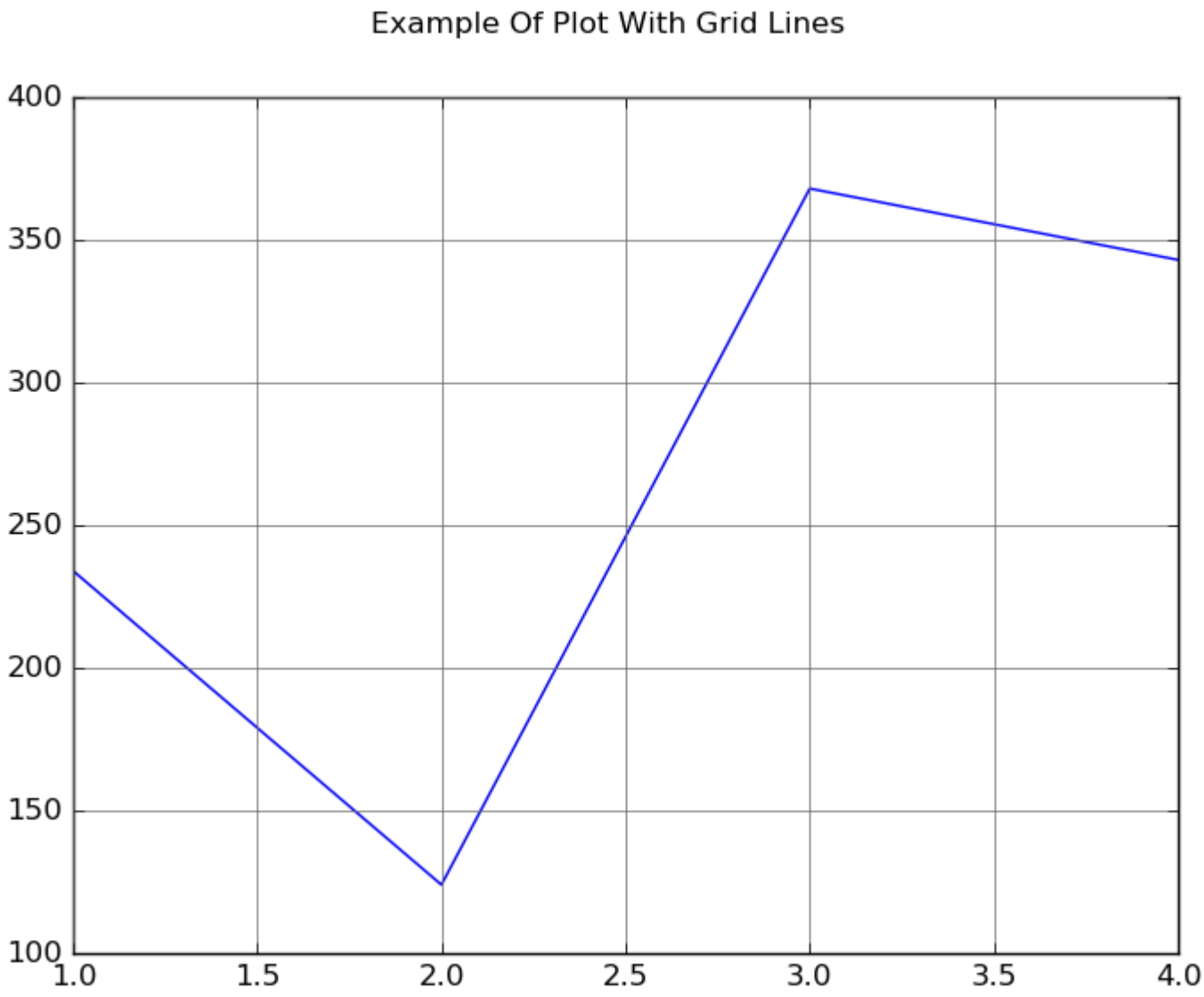
Lea Leyendas en línea: https://riptutorial.com/es/matplotlib/topic/2840/leyendas

Capítulo 11: Líneas de cuadrícula y marcas de garrapatas

Examples

Parcela Con Gridlines

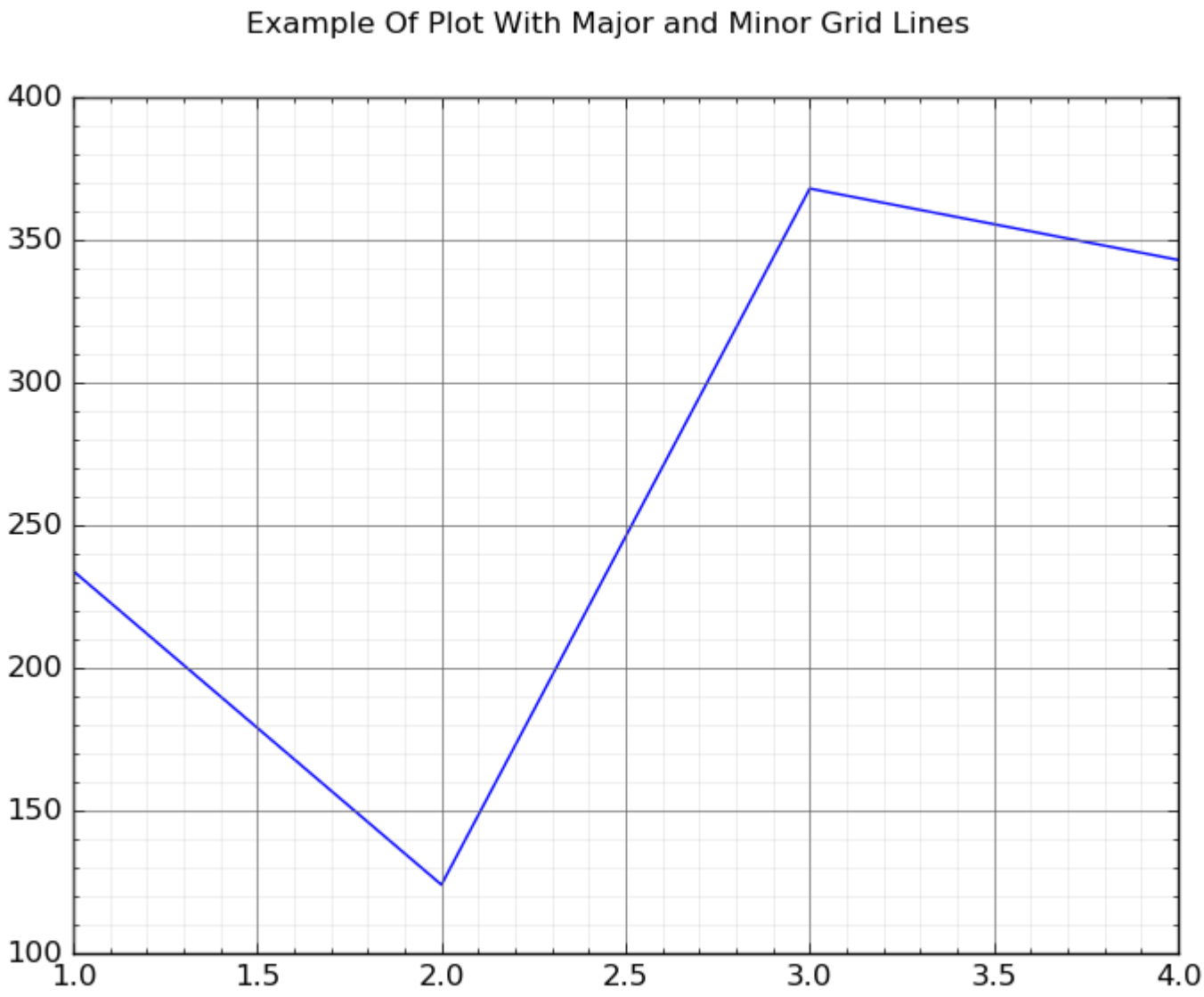
Parcela Con Líneas De Rejilla




```
import matplotlib.pyplot as plt

# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]
```

```
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Grid Lines')

# Plot the data
ax.plot(x,y)

# Show the grid lines as dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')

plt.show()
```

Parcela con líneas de rejilla mayores y menores




```
import matplotlib.pyplot as plt
```

```
# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]

# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Major and Minor Grid Lines')

# Plot the data
ax.plot(x,y)

# Show the major grid lines with dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')

# Show the minor grid lines with very faint and almost transparent grey lines
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show()
```

Lea Líneas de cuadrícula y marcas de garrapatas en línea:
https://riptutorial.com/es/matplotlib/topic/4029/lineas-de-cuadricula-y-marcas-de-garrapatas

Capítulo 12: LogLog Graphing

Introducción

La gráfica LogLog es una posibilidad para ilustrar una función exponencial de una manera lineal.

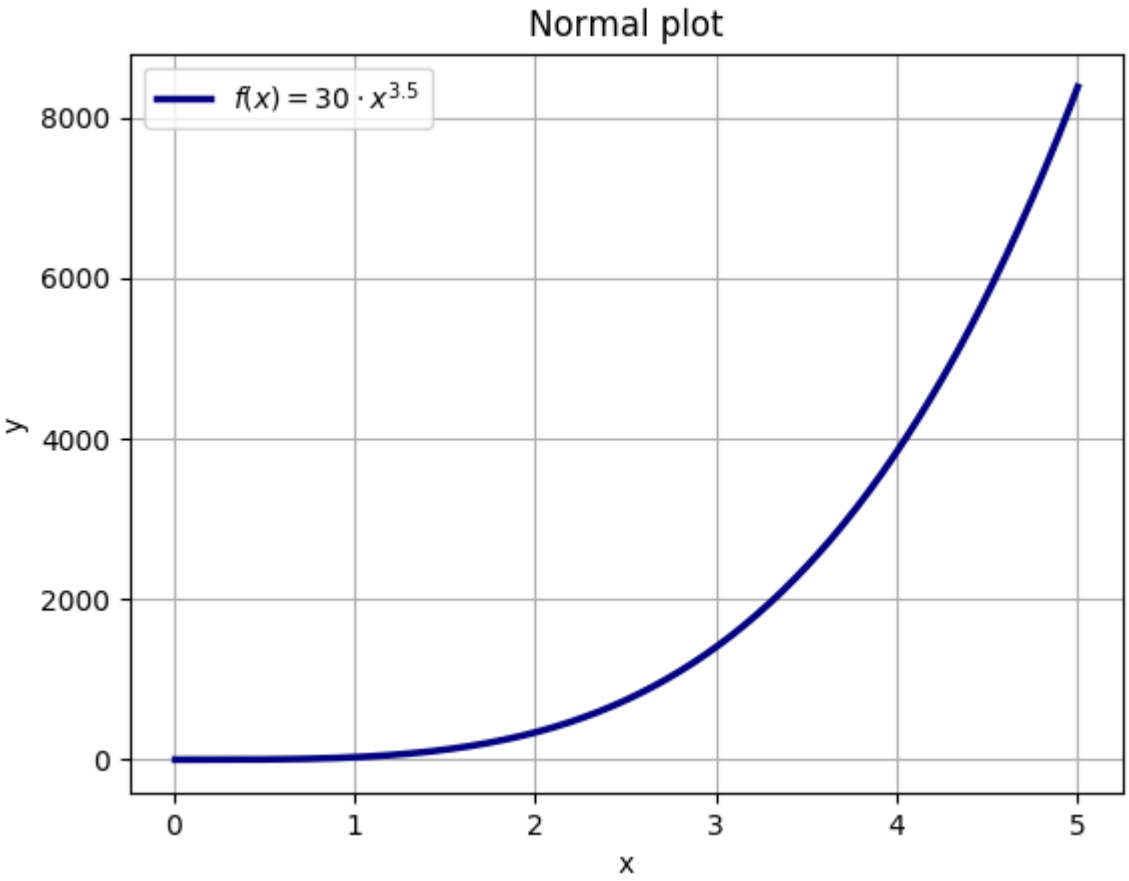
Examples

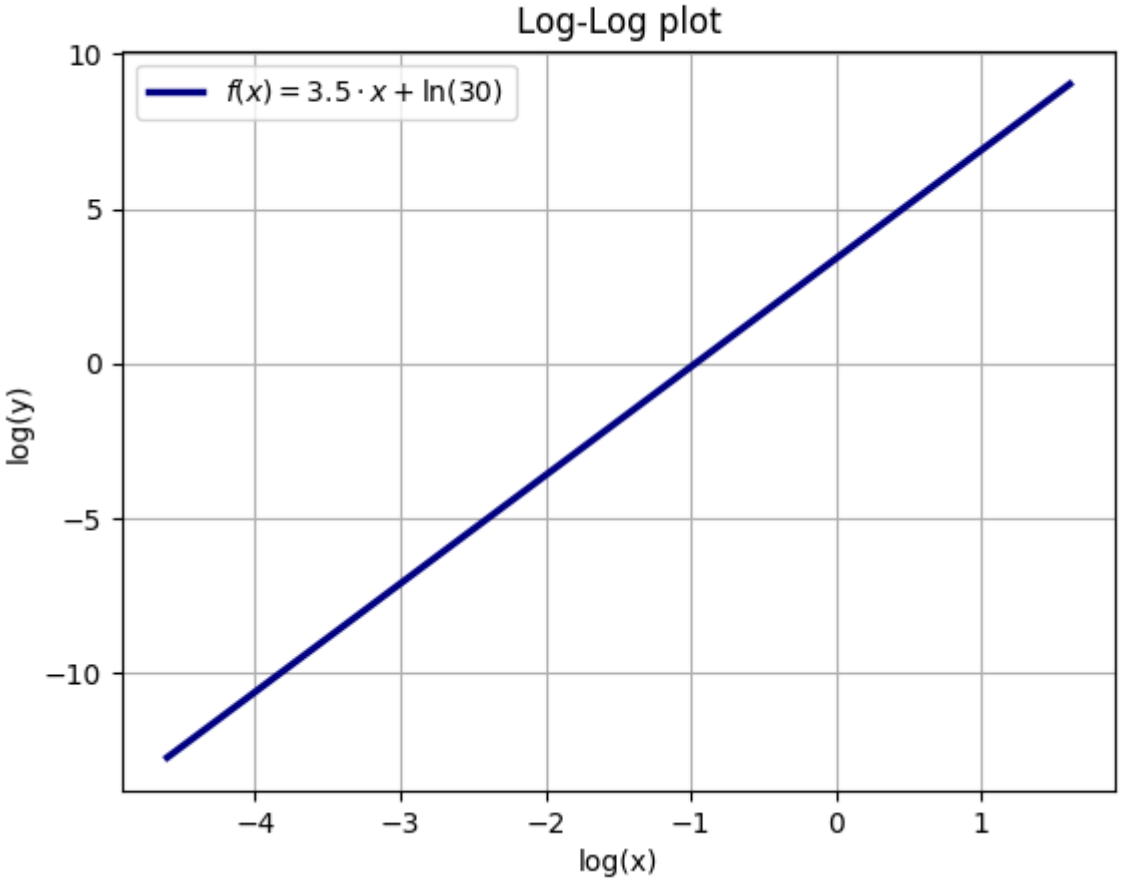
LogLog graficando

Sea y (x) = A * x ^ a, por ejemplo A = 30 y a = 3.5. Si se toma el logaritmo natural (ln) de ambos lados (usando las reglas comunes para logaritmos): ln (y) = ln (A * x ^ a) = ln (A) + ln (x ^ a) = ln (A) + a * ln (x). Por lo tanto, una gráfica con ejes logarítmicos tanto para x como para y será una curva lineal. La pendiente de esta curva es el exponente a de y (x), mientras que el intercepto y y (0) es el logaritmo natural de A, ln (A) = ln (30) = 3.401.

El siguiente ejemplo ilustra la relación entre una función exponencial y el gráfico de loglog lineal (la función es y = A * x ^ a con A = 30 y a = 3.5):

```
import numpy as np
import matplotlib.pyplot as plt
A = 30
a = 3.5
x = np.linspace(0.01, 5, 10000)
y = A * x**a

ax = plt.gca()
plt.plot(x, y, linewidth=2.5, color='navy', label=r'$f(x) = 30 \cdot x^{3.5}$')
plt.legend(loc='upper left')
plt.xlabel(r'x')
plt.ylabel(r'y')
ax.grid(True)
plt.title(r'Normal plot')
plt.show()
plt.clf()

xlog = np.log(x)
ylog = np.log(y)
ax = plt.gca()
plt.plot(xlog, ylog, linewidth=2.5, color='navy', label=r'$f(x) = 3.5\cdot x + \ln(30)$')
plt.legend(loc='best')
plt.xlabel(r'log(x)')
plt.ylabel(r'log(y)')
ax.grid(True)
plt.title(r'Log-Log plot')
plt.show()
plt.clf()
```


Normal plot
$f(x) = 30 \cdot x^{3.5}$
8000
6000
4000
2000
0
y
0
1
2
3
4
5
x


Log-Log plot
$f(x) = 3.5 \cdot x + \ln(30)$
10
5
0
−5
−10
log(y)
−4
−3
−2
−1
0
1
log(x)

Lea LogLog Graphing en línea: https://riptutorial.com/es/matplotlib/topic/10145/loglog-graphing

Capítulo 13: Manipulación de imagen

Examples

Abriendo imagenes

Matplotlib incluye la `image` módulo para la manipulación de imágenes

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

Las imágenes se leen desde un archivo (`.png` solamente) con la función `imread` :

```
img = mpimg.imread('my_image.png')
```

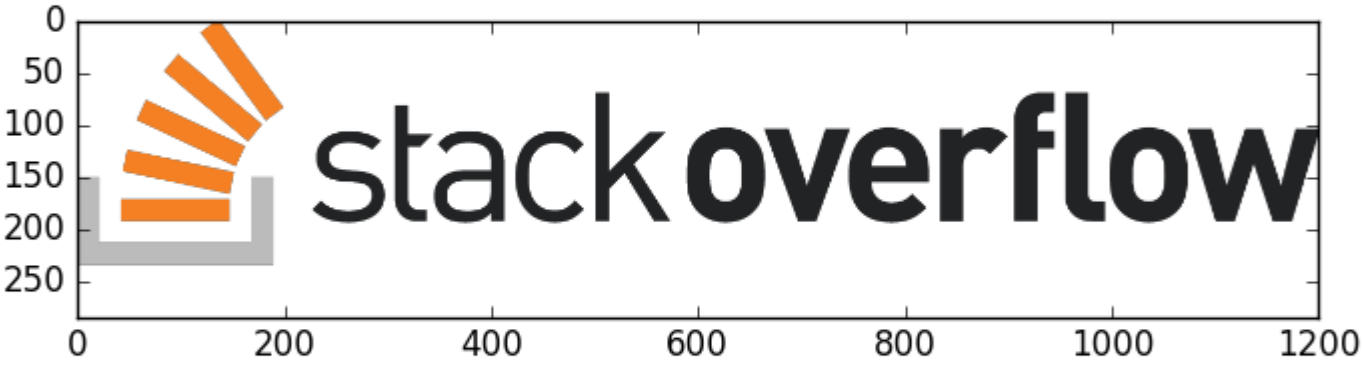
y son representados por la función `imshow` :

```
plt.imshow(img)
```

Vamos a *trazar* el logotipo de desbordamiento de pila :

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
img = mpimg.imread('so-logo.png')
plt.imshow(img)
plt.show()
```

La trama resultante es



Lea Manipulación de imagen en línea:
https://riptutorial.com/es/matplotlib/topic/4575/manipulacion-de-imagen

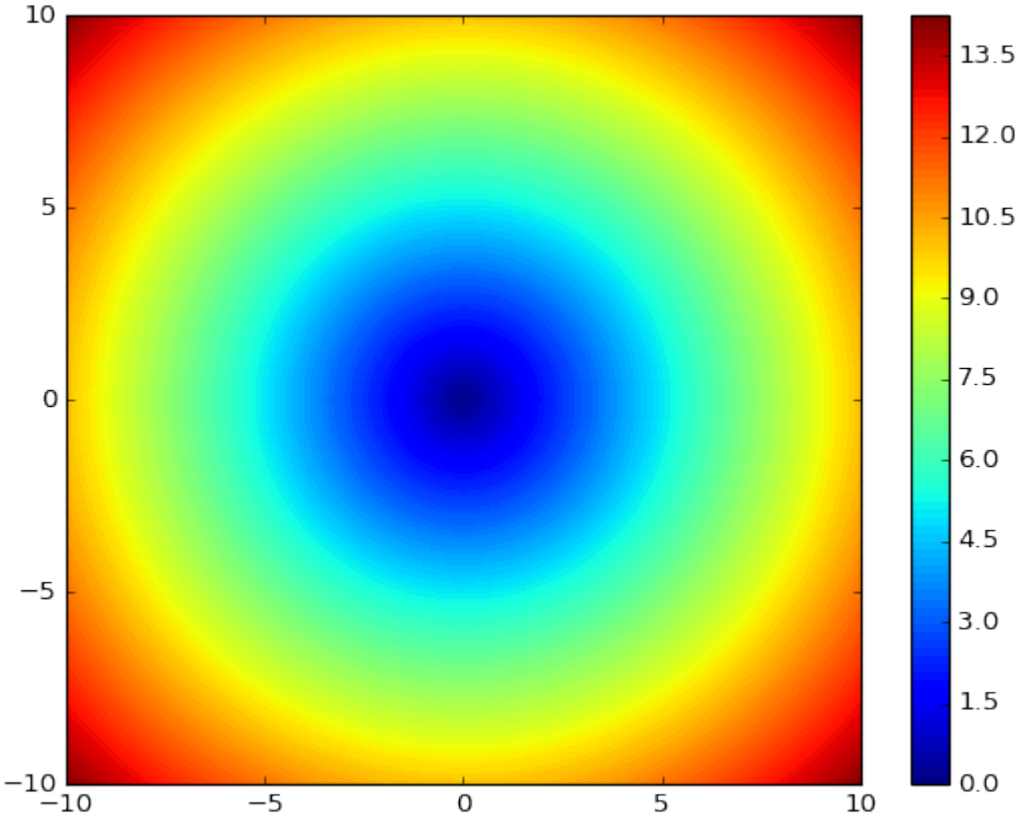
Capítulo 14: Mapas de contorno

Examples

Trazado de contorno rellenado simple

```
import matplotlib.pyplot as plt
import numpy as np

# generate 101 x and y values between -10 and 10
x = np.linspace(-10, 10, 101)
y = np.linspace(-10, 10, 101)

# make X and Y matrices representing x and y values of 2d plane
X, Y = np.meshgrid(x, y)

# compute z value of a point as a function of x and y (z = l2 distance form 0,0)
Z = np.sqrt(X ** 2 + Y ** 2)

# plot filled contour map with 100 levels
cs = plt.contourf(X, Y, Z, 100)

# add default colorbar for the map
plt.colorbar(cs)
```

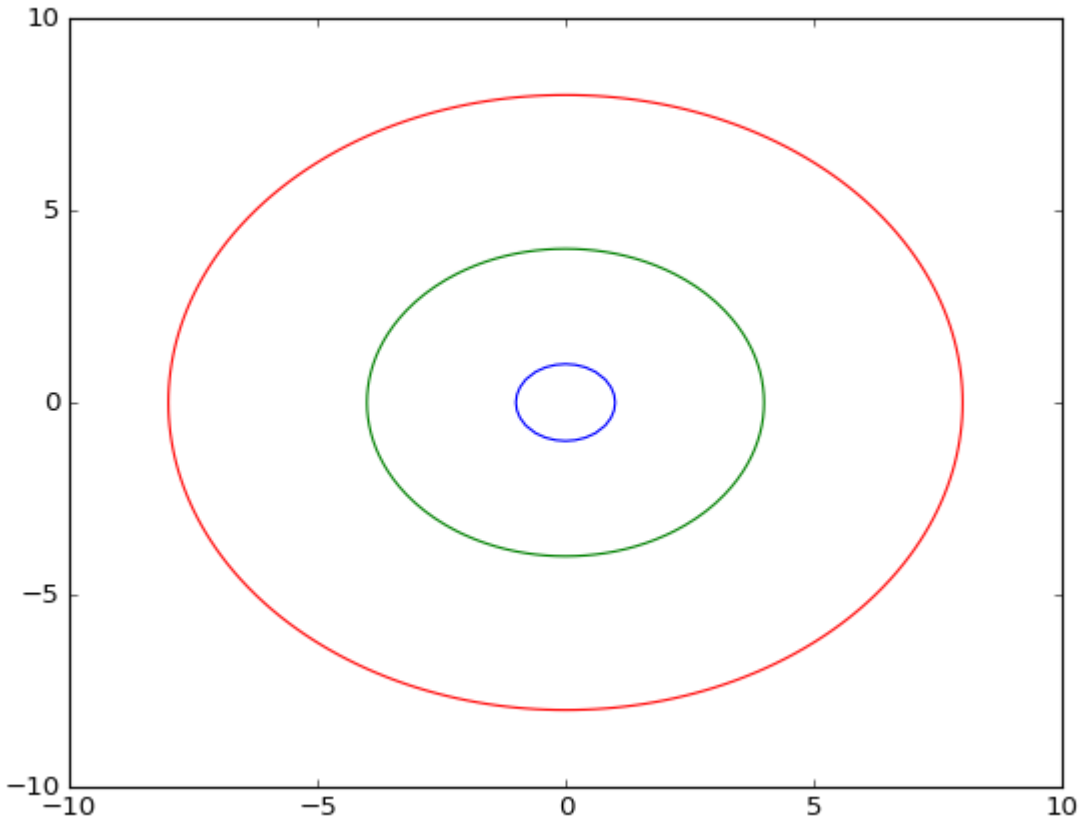
Resultado:



Trazado de contorno simple

```
import matplotlib.pyplot as plt
import numpy as np

# generate 101 x and y values between -10 and 10
x = np.linspace(-10, 10, 101)
y = np.linspace(-10, 10, 101)

# make X and Y matrices representing x and y values of 2d plane
X, Y = np.meshgrid(x, y)

# compute z value of a point as a function of x and y (z = l2 distance form 0,0)
Z = np.sqrt(X ** 2 + Y ** 2)

# plot contour map with 3 levels
# colors: up to 1 - blue,  from 1 to 4 - green, from 4 to 8 - red
plt.contour(X, Y, Z, [1, 4, 8], colors=['b', 'g', 'r'])
```

Resultado:



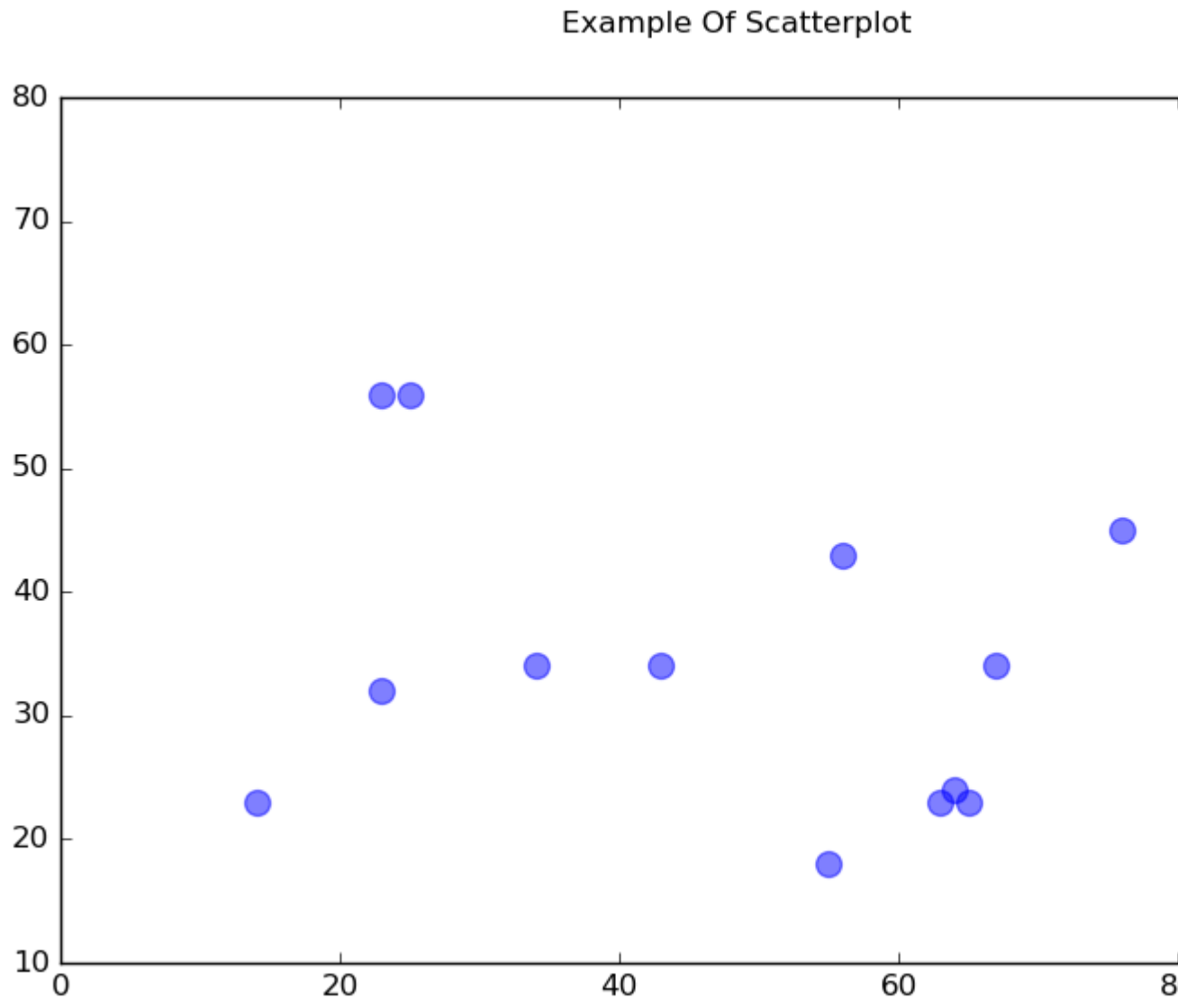
Lea Mapas de contorno en línea: https://riptutorial.com/es/matplotlib/topic/8644/mapas-de-contorno

Capítulo 15: Parcelas básicas

Examples

Gráfico de dispersión

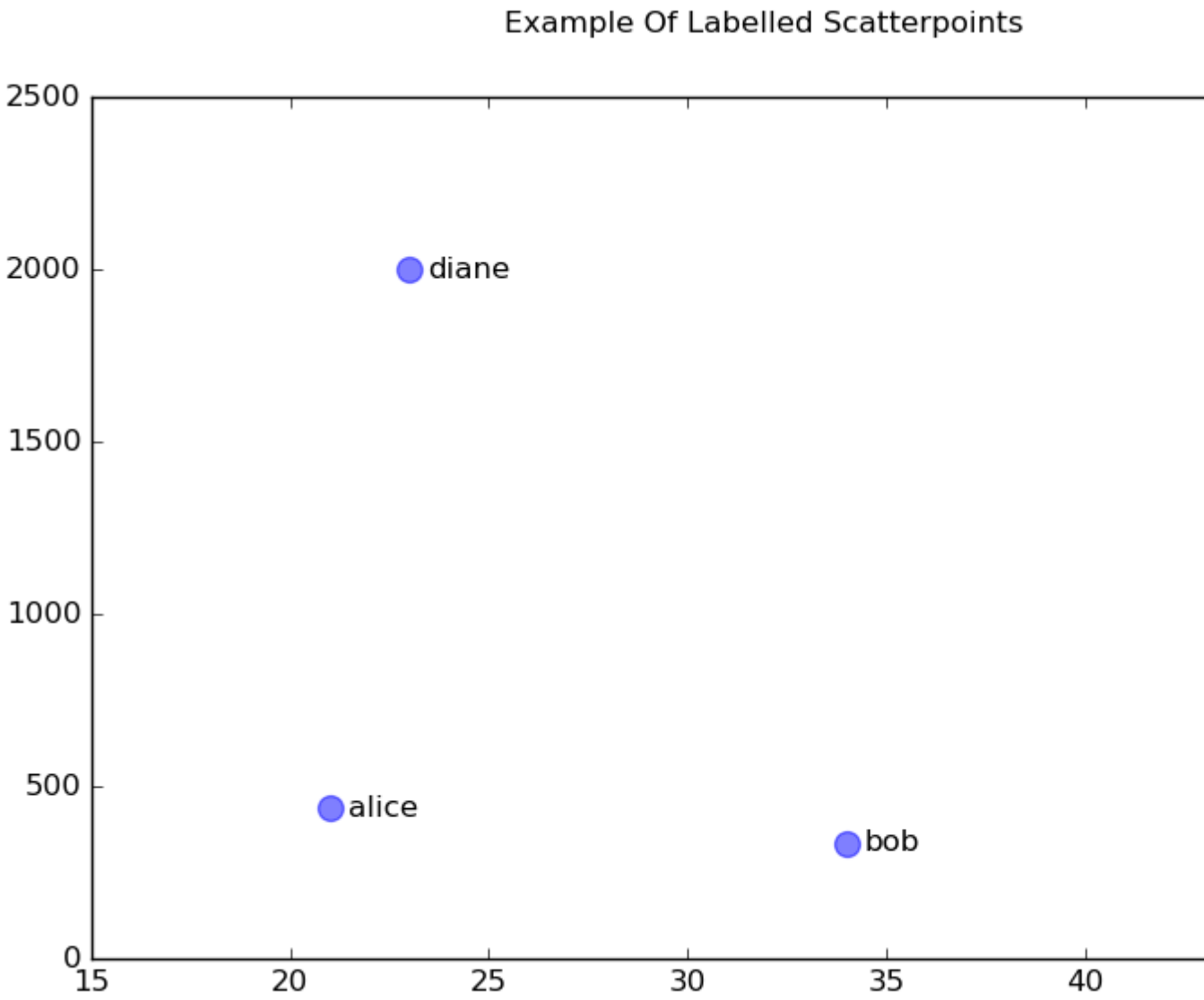
Un simple diagrama de dispersión




```
import matplotlib.pyplot as plt

# Data
x = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
y = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]

fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Scatterplot')
```

```
# Create the Scatter Plot
ax.scatter(x, y,
            color="blue",    # Color of the dots
            s=100,           # Size of the dots
            alpha=0.5,       # Alpha/transparency of the dots (1 is opaque, 0 is transparent)
            linewidths=1)    # Size of edge around the dots

# Show the plot
plt.show()
```

Un diagrama de dispersión con puntos etiquetados

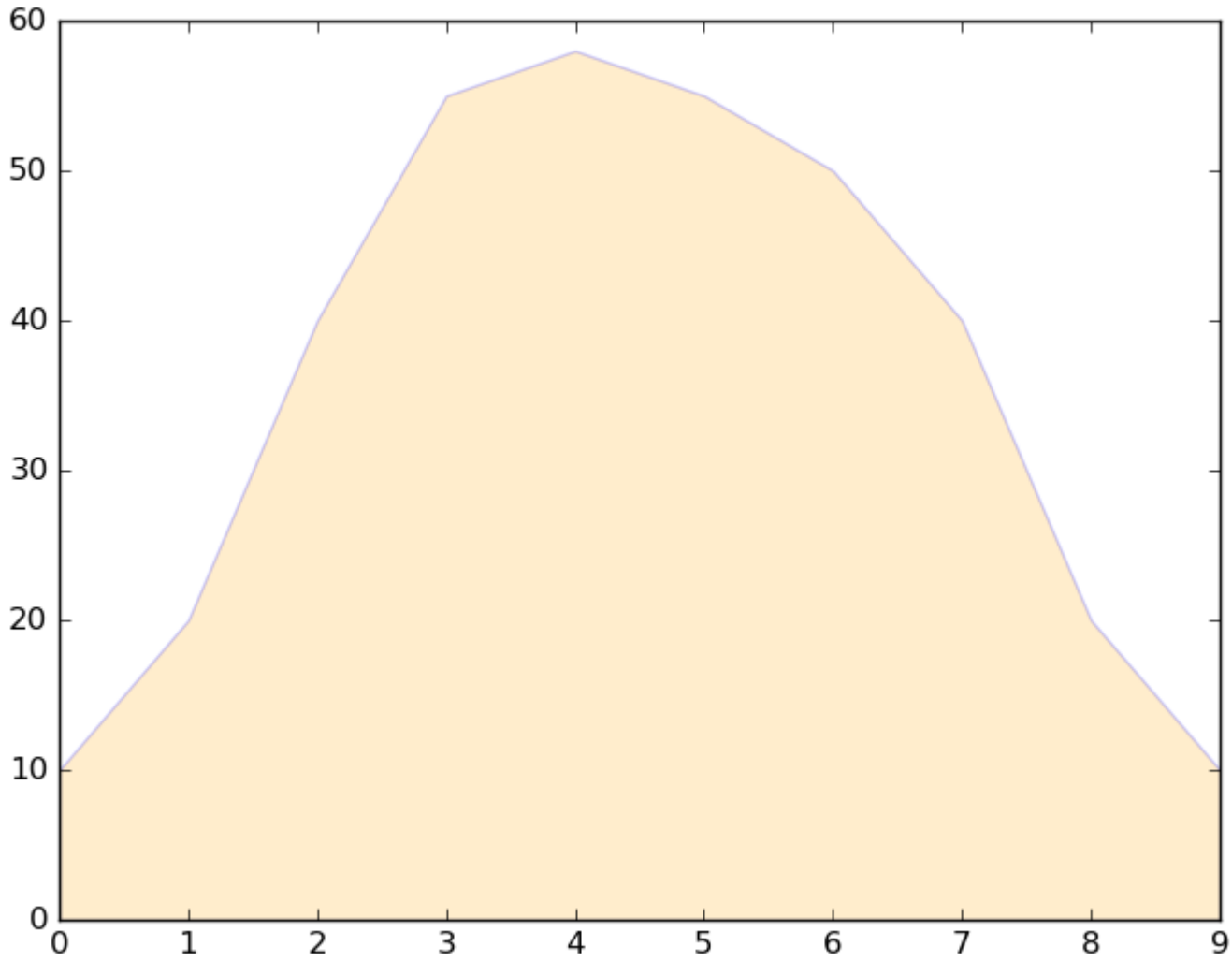



```
import matplotlib.pyplot as plt

# Data
x = [21, 34, 44, 23]
y = [435, 334, 656, 1999]
labels = ["alice", "bob", "charlie", "diane"]

# Create the figure and axes objects
```

```
fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Labelled Scatterpoints')

# Plot the scatter points
ax.scatter(x, y,
           color="blue",  # Color of the dots
           s=100,         # Size of the dots
           alpha=0.5,     # Alpha of the dots
           linewidths=1)  # Size of edge around the dots

# Add the participant names as text labels for each point
for x_pos, y_pos, label in zip(x, y, labels):
    ax.annotate(label,             # The label for this point
                xy=(x_pos, y_pos), # Position of the corresponding point
                xytext=(7, 0),     # Offset text by 7 points to the right
                textcoords='offset points', # tell it to use offset points
                ha='left',         # Horizontally aligned to the left
                va='center')       # Vertical alignment is centered

# Show the plot
plt.show()
```

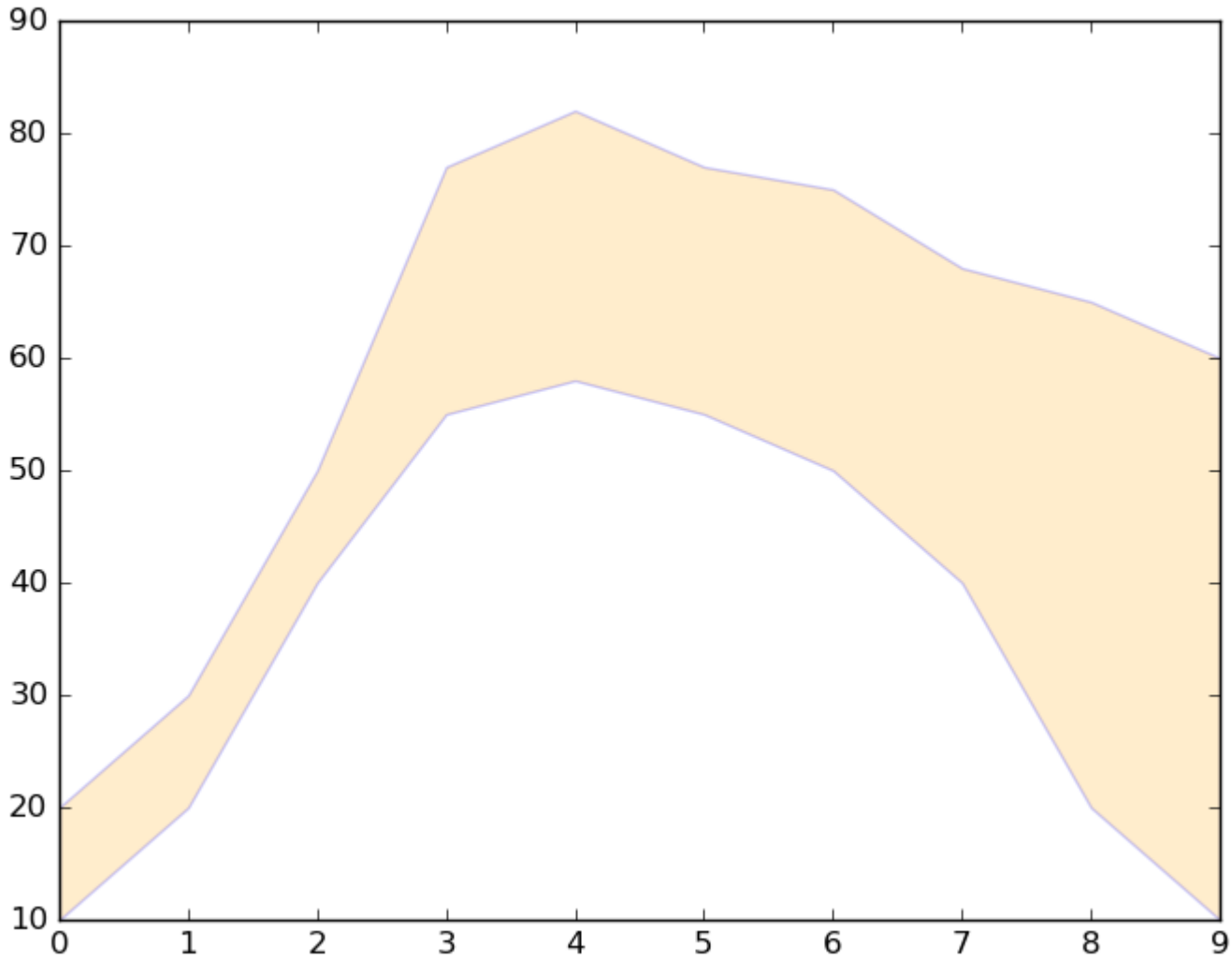
Parcelas Sombreadas

Región sombreada debajo de una línea




```
import matplotlib.pyplot as plt

# Data
x =  [0,1,2,3,4,5,6,7,8,9]
y1 = [10,20,40,55,58,55,50,40,20,10]

# Shade the area between y1 and line y=0
plt.fill_between(x, y1, 0,
                 facecolor="orange", # The fill color
                 color='blue',       # The outline color
                 alpha=0.2)          # Transparency of the fill

# Show the plot
plt.show()
```

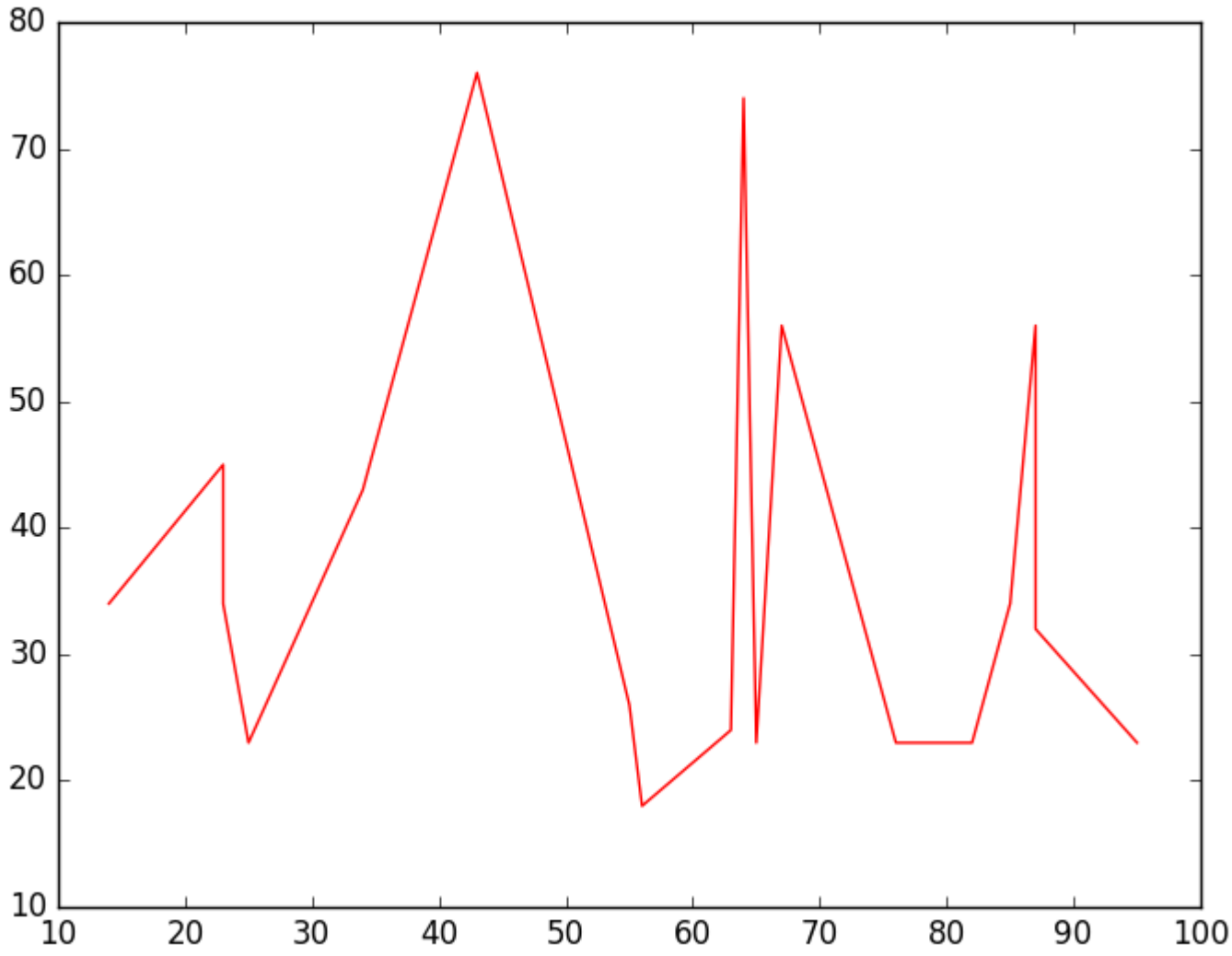
Región sombreada entre dos líneas




```
import matplotlib.pyplot as plt

# Data
x =  [0,1,2,3,4,5,6,7,8,9]
y1 = [10,20,40,55,58,55,50,40,20,10]
y2 = [20,30,50,77,82,77,75,68,65,60]

# Shade the area between y1 and y2
plt.fill_between(x, y1, y2,
                 facecolor="orange", # The fill color
                 color='blue',       # The outline color
                 alpha=0.2)          # Transparency of the fill

# Show the plot
plt.show()
```

Líneas de parcelas

Trazo de línea simple




```
import matplotlib.pyplot as plt

# Data
x = [14,23,23,25,34,43,55,56,63,64,65,67,76,82,85,87,87,95]
y = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]

# Create the plot
plt.plot(x, y, 'r-')
# r- is a style code meaning red solid line

# Show the plot
plt.show()
```

Tenga en cuenta que, en general, `y` no es una función de `x` y que los valores en `x` no necesitan ordenarse. Así es como se ve una gráfica de líneas con valores x sin clasificar:

```
# shuffle the elements in x
np.random.shuffle(x)
plt.plot(x, y, 'r-')
plt.show()
```

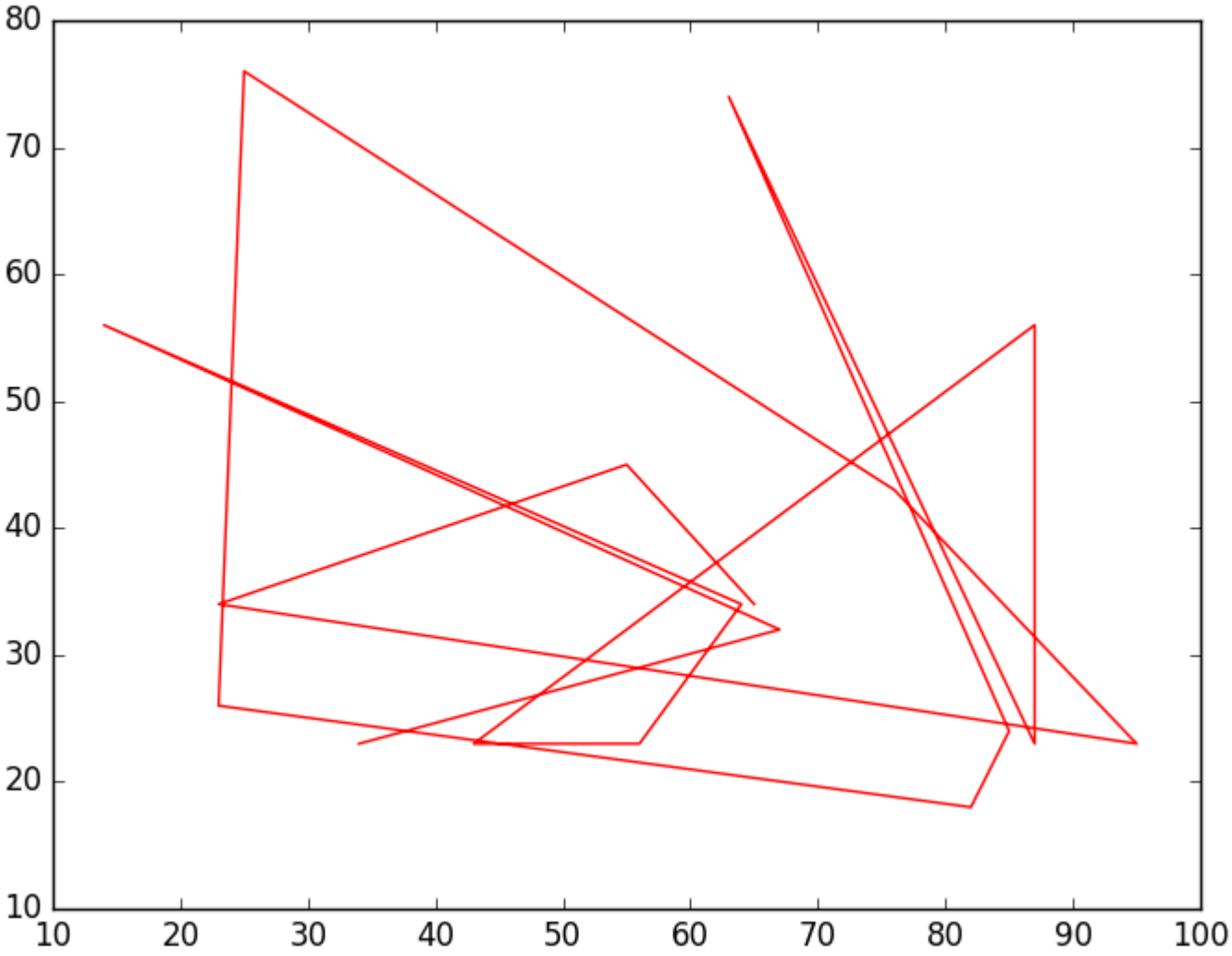
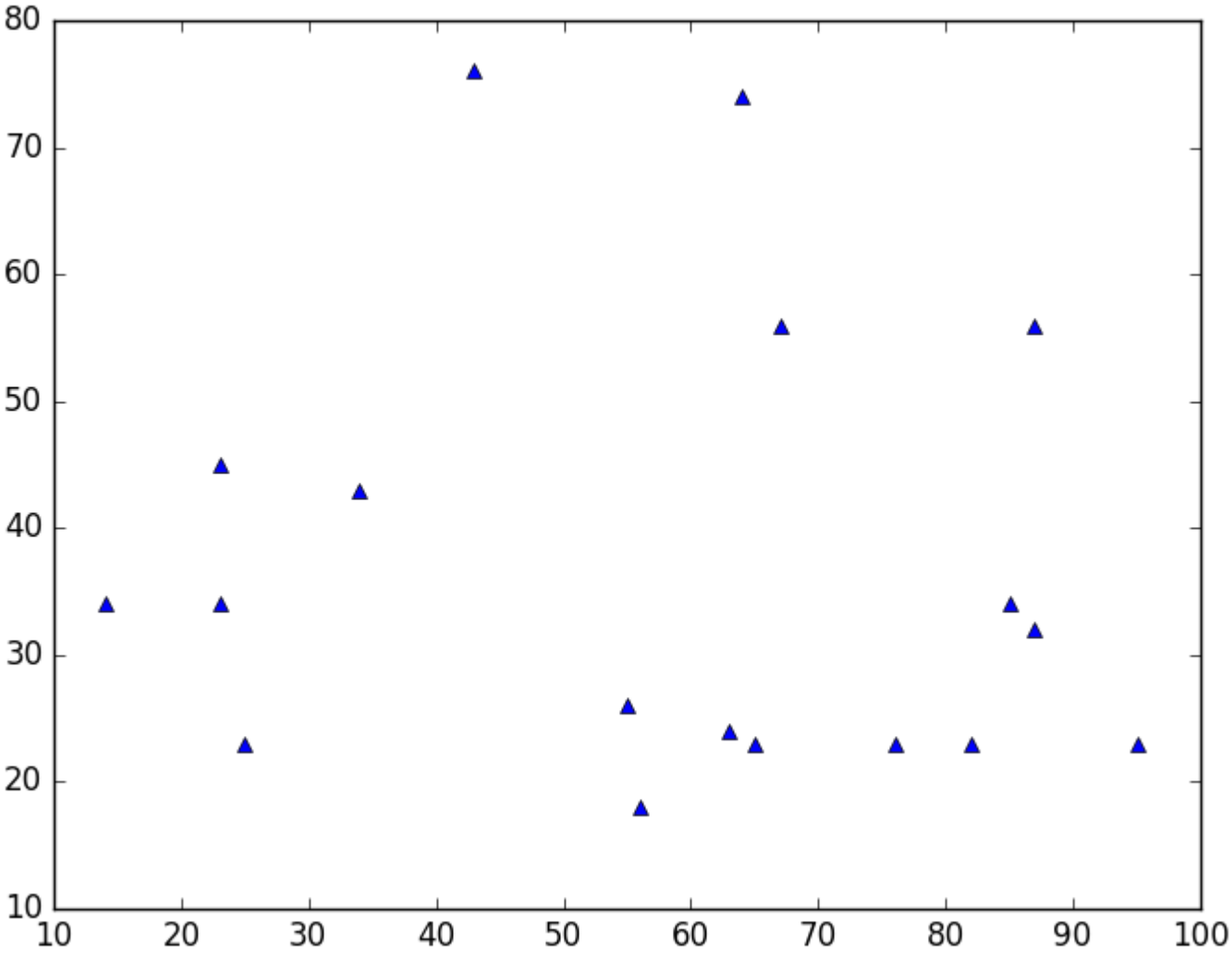


Diagrama de datos

Esto es similar a un diagrama de dispersión , pero usa la función `plot()` lugar. La única diferencia en el código aquí es el argumento de estilo.

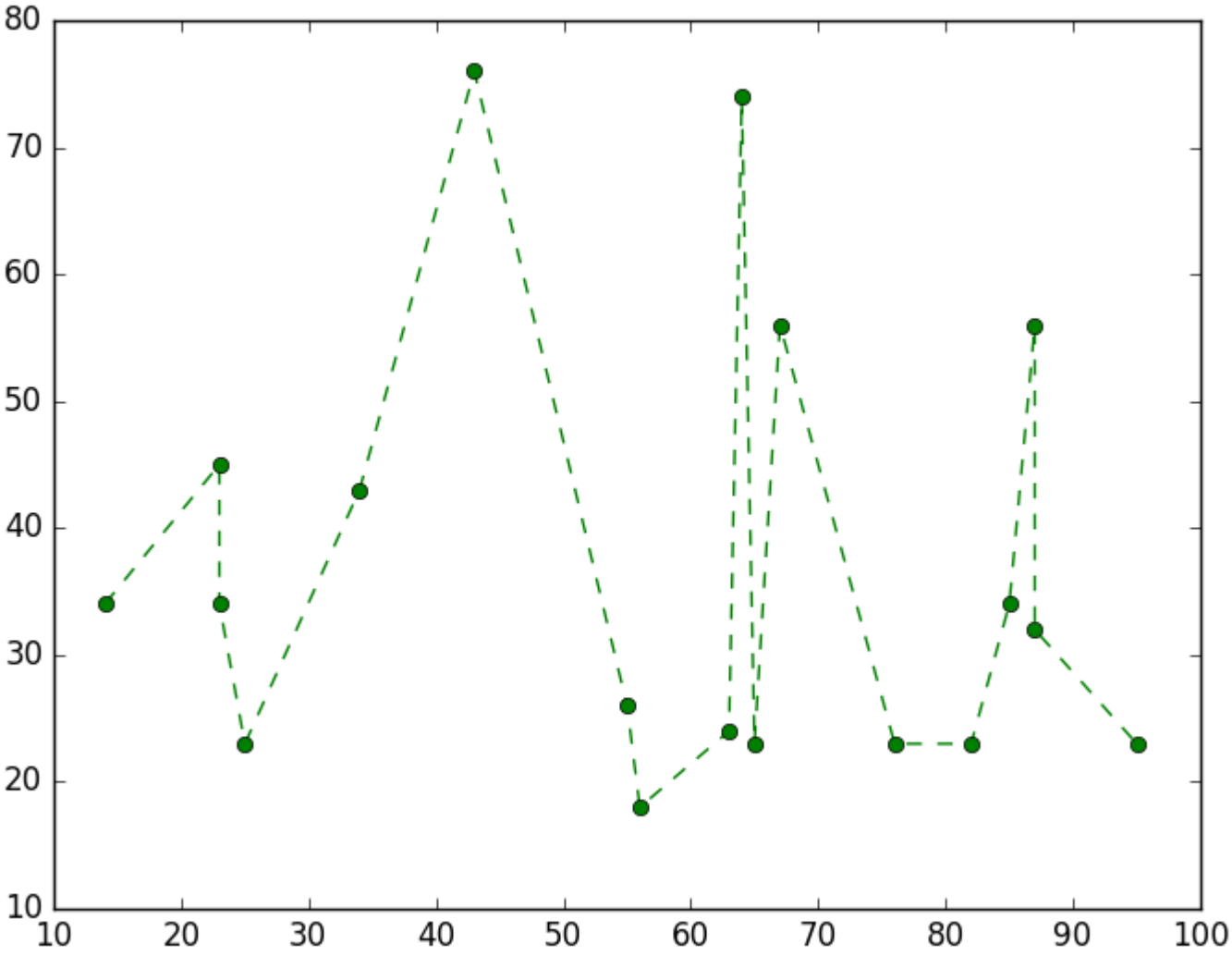
```
plt.plot(x, y, 'b^')
# Create blue up-facing triangles
```



Datos y linea

El argumento de estilo puede tomar símbolos para ambos marcadores y estilo de línea:

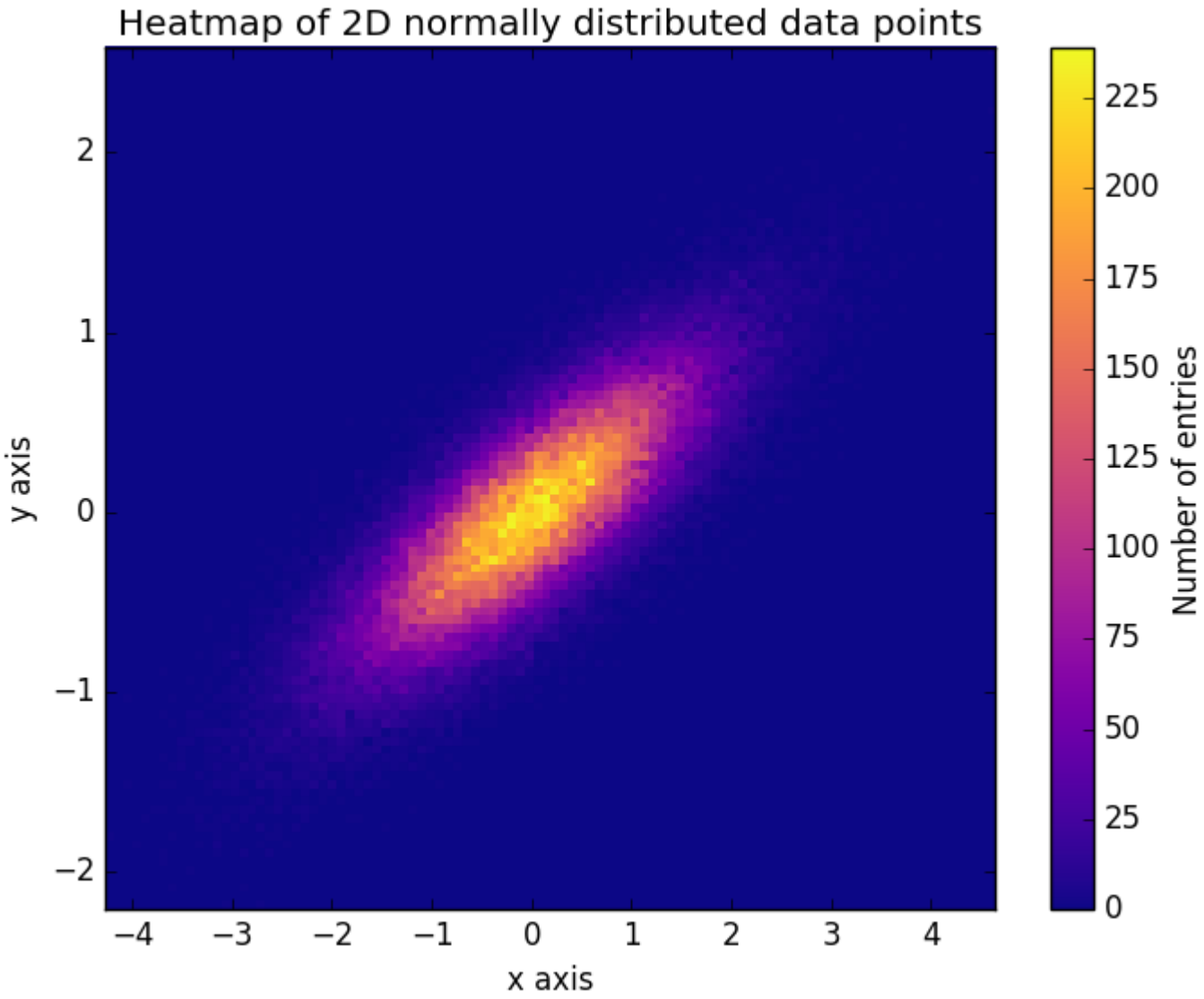
```
plt.plot(x, y, 'go--')
# green circles and dashed line
```



Mapa de calor

Los mapas de calor son útiles para visualizar funciones escalares de dos variables. Proporcionan una imagen “plana” de histogramas bidimensionales (que representan, por ejemplo, la densidad de un área determinada).

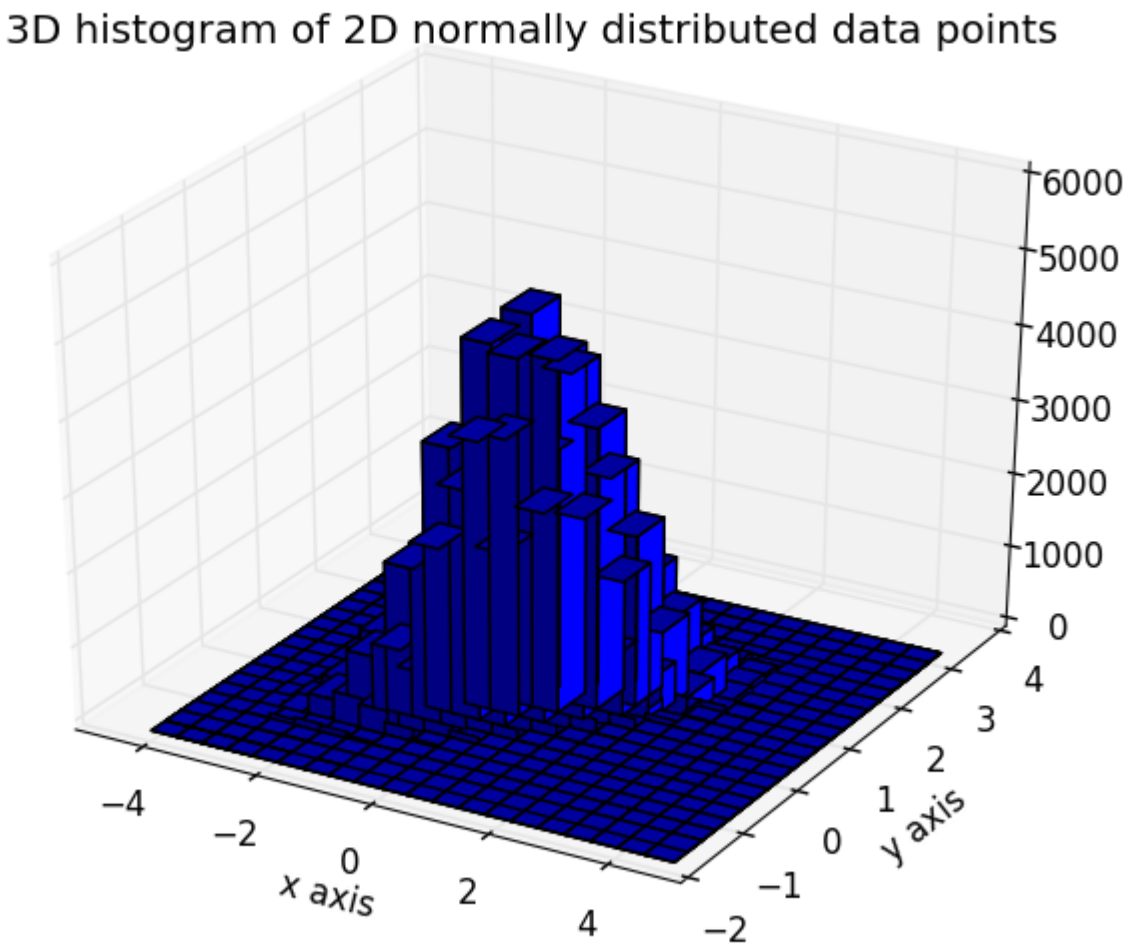
El siguiente código fuente ilustra mapas de calor utilizando números bivariados normalmente distribuidos centrados en 0 en ambas direcciones (medios `[0.0, 0.0]`) y con una matriz de covarianza dada. Los datos se generan utilizando la función numpy numpy.random.multivariate_normal ; A continuación, se alimenta a la `hist2d` función de pyplot matplotlib.pyplot.hist2d .




```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Define numbers of generated data points and bins per axis.
N_numbers = 100000
N_bins = 100

# set random seed
np.random.seed(0)

# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
        mean=[0.0, 0.0],      # mean
        cov=[[1.0, 0.4],
             [0.4, 0.25]],    # covariance matrix
        size=N_numbers
        ).T                   # transpose to get columns


# Construct 2D histogram from data using the 'plasma' colormap
plt.hist2d(x, y, bins=N_bins, normed=False, cmap='plasma')
```

```
# Plot a colorbar with label.
cb = plt.colorbar()
cb.set_label('Number of entries')

# Add title and labels to plot.
plt.title('Heatmap of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')

# Show the plot.
plt.show()
```

Aquí se muestran los mismos datos que en un histograma 3D (aquí usamos solo 20 contenedores para la eficiencia). El código se basa en esta demo matplotlib .




```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Define numbers of generated data points and bins per axis.
```

```
N_numbers = 100000
N_bins = 20

# set random seed
np.random.seed(0)

# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
        mean=[0.0, 0.0],      # mean
        cov=[[1.0, 0.4],
             [0.4, 0.25]],    # covariance matrix
        size=N_numbers
        ).T                   # transpose to get columns

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
hist, xedges, yedges = np.histogram2d(x, y, bins=N_bins)

# Add title and labels to plot.
plt.title('3D histogram of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')

# Construct arrays for the anchor positions of the bars.
# Note: np.meshgrid gives arrays in (ny, nx) so we use 'F' to flatten xpos,
# ypos in column-major order. For numpy >= 1.7, we could instead call meshgrid
# with indexing='ij'.
xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)
xpos = xpos.flatten('F')
ypos = ypos.flatten('F')
zpos = np.zeros_like(xpos)

# Construct arrays with the dimensions for the 16 bars.
dx = 0.5 * np.ones_like(zpos)
dy = dx.copy()
dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average')

# Show the plot.
plt.show()
```

Lea Parcelas básicas en línea: https://riptutorial.com/es/matplotlib/topic/3266/parcelas-basicas

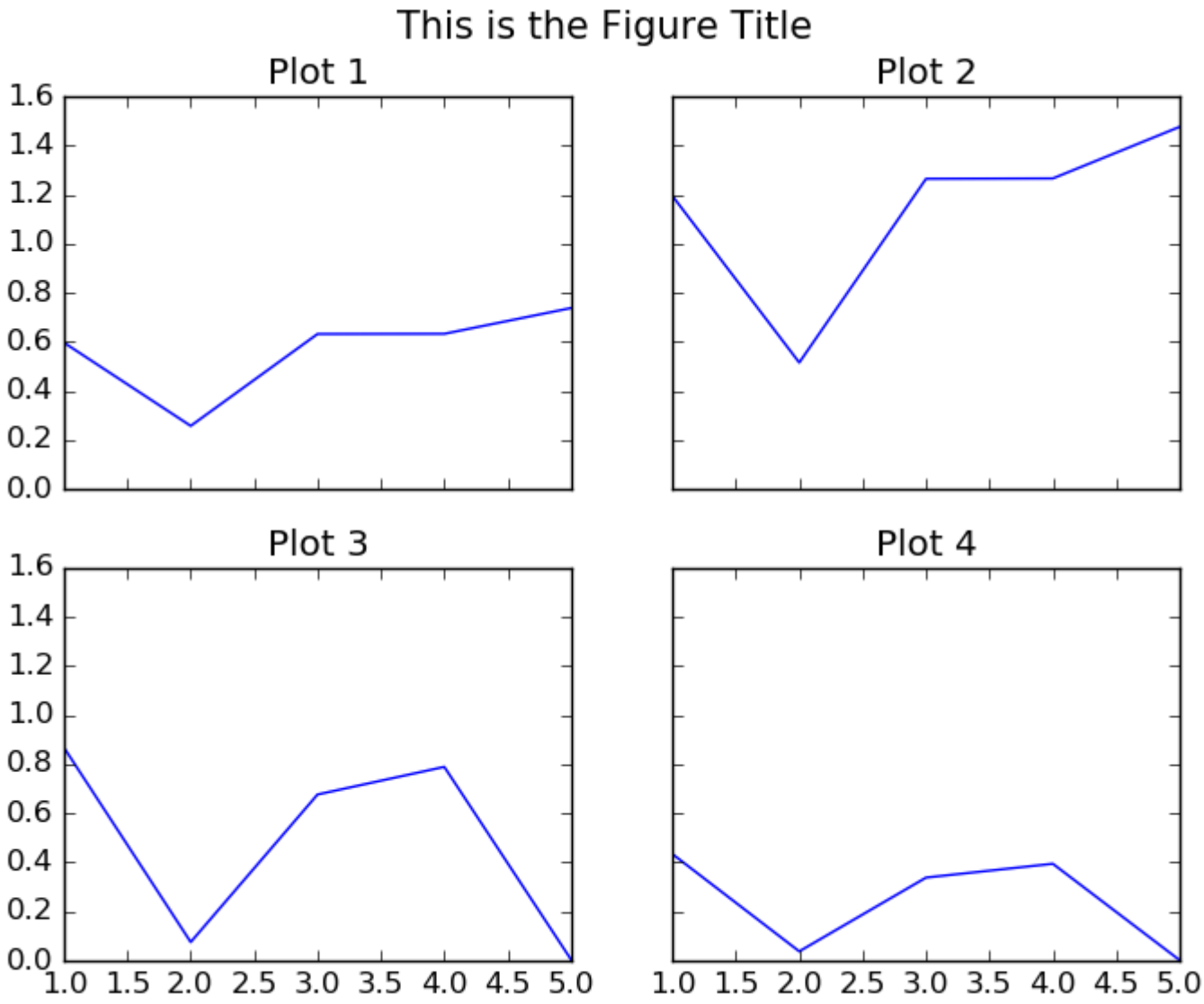
Capítulo 16: Parcelas Múltiples

Sintaxis

- Elemento de lista

Examples

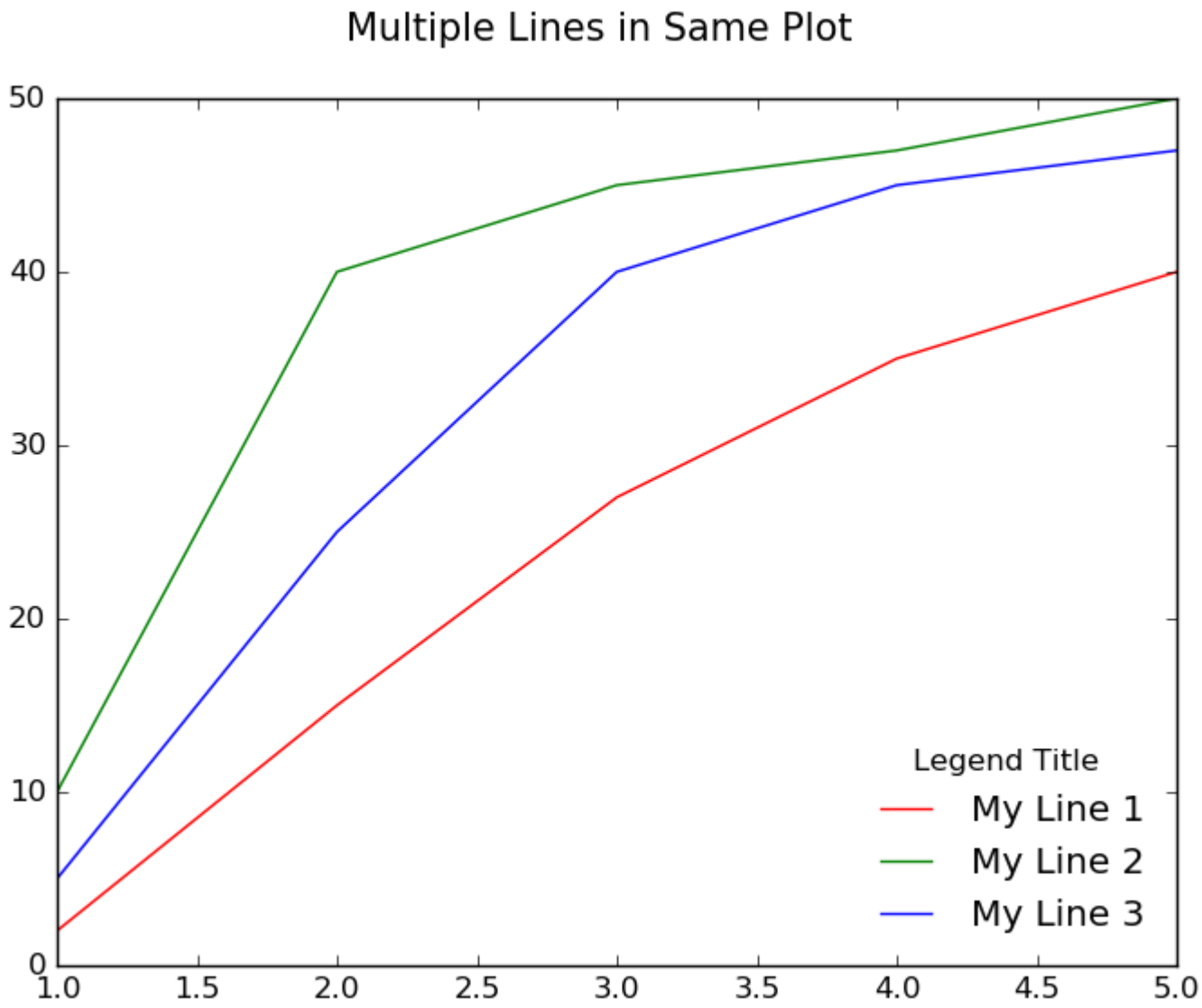
Rejilla de subparcelas usando subparcela




```
"""
=============================================================================
CREATE A 2 BY 2 GRID OF SUB-PLOTS WITHIN THE SAME FIGURE.
=============================================================================
"""
import matplotlib.pyplot as plt
```

```
# The data
x = [1,2,3,4,5]
y1 = [0.59705847, 0.25786401, 0.63213726, 0.63287317, 0.73791151]
y2 = [1.19411694, 0.51572803, 1.26427451, 1.26574635, 1.47582302]
y3 = [0.86793828, 0.07563408, 0.67670068, 0.78932712, 0.0043694]
# 5 more random values
y4 = [0.43396914, 0.03781704, 0.33835034, 0.39466356, 0.0021847]

# Initialise the figure and a subplot axes. Each subplot sharing (showing) the
# same range of values for the x and y axis in the plots.
fig, axes = plt.subplots(2, 2, figsize=(8, 6), sharex=True, sharey=True)

# Set the title for the figure
fig.suptitle('This is the Figure Title', fontsize=15)

# Top Left Subplot
axes[0,0].plot(x, y1)
axes[0,0].set_title("Plot 1")

# Top Right Subplot
axes[0,1].plot(x, y2)
axes[0,1].set_title("Plot 2")

# Bottom Left Subplot
axes[1,0].plot(x, y3)
axes[1,0].set_title("Plot 3")

# Bottom Right Subplot
axes[1,1].plot(x, y4)
axes[1,1].set_title("Plot 4")

plt.show()
```

Múltiples líneas / curvas en la misma parcela




```
"""
================================================================================
                    DRAW MULTIPLE LINES IN THE SAME PLOT
================================================================================
"""
import matplotlib.pyplot as plt

# The data
x =  [1, 2, 3, 4, 5]
y1 = [2,  15, 27, 35, 40]
y2 = [10, 40, 45, 47, 50]
y3 = [5,  25, 40, 45, 47]

# Initialise the figure and axes.
fig, ax = plt.subplots(1, figsize=(8, 6))

# Set the title for the figure
fig.suptitle('Multiple Lines in Same Plot', fontsize=15)

# Draw all the lines in the same plot, assigning a label for each one to be
# shown in the legend.
ax.plot(x, y1, color="red", label="My Line 1")
ax.plot(x, y2, color="green", label="My Line 2")
```

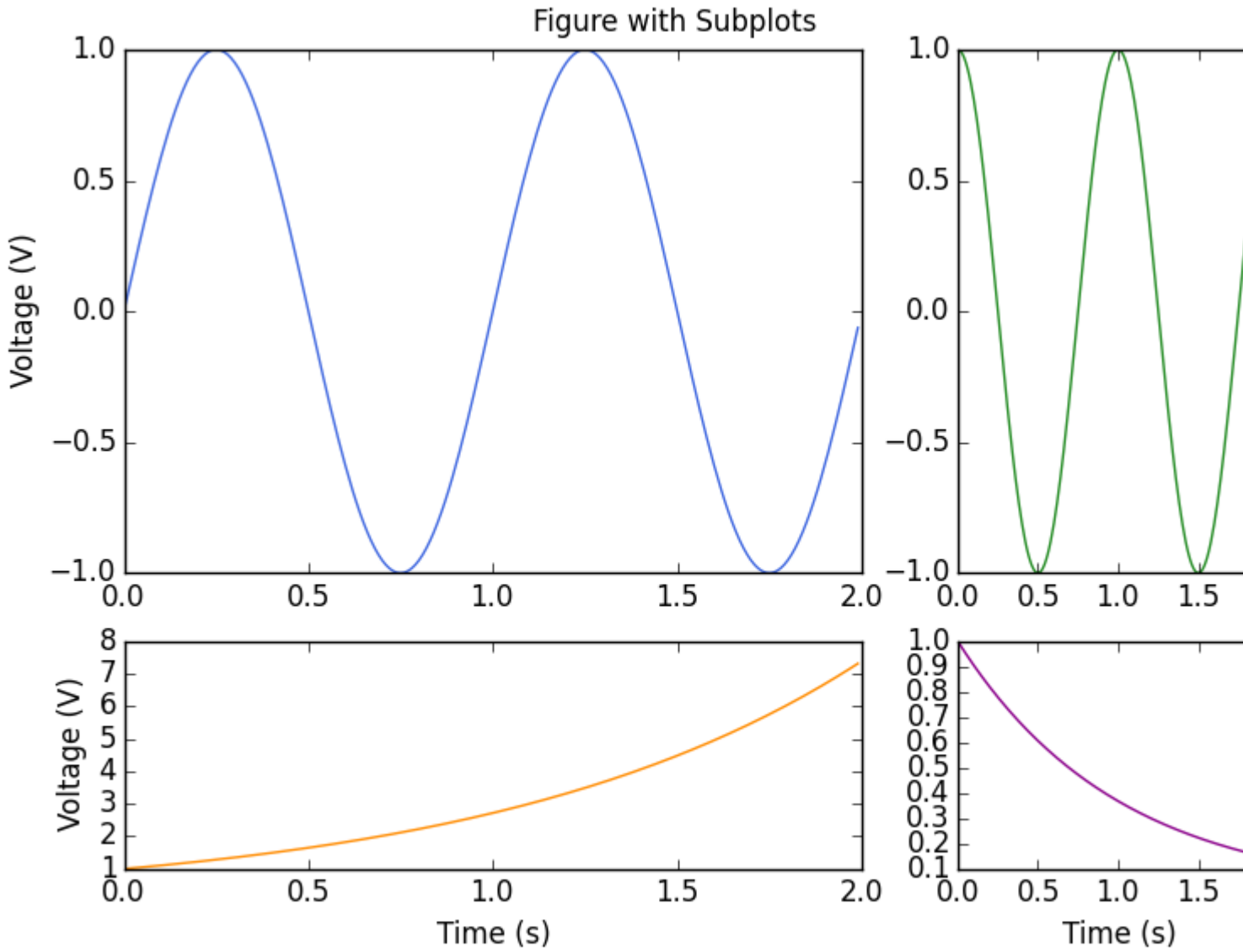
```
ax.plot(x, y3, color="blue", label="My Line 3")

# Add a legend, and position it on the lower right (with no box)
plt.legend(loc="lower right", title="Legend Title", frameon=False)

plt.show()
```

Parcelas Múltiples con Gridspec

El paquete `gridspec` permite un mayor control sobre la ubicación de las subparcelas. Facilita el control de los márgenes de las parcelas y el espaciado entre las subparcelas individuales. Además, permite ejes de diferentes tamaños en la misma figura al definir ejes que ocupan varias ubicaciones de cuadrícula.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import  GridSpec

# Make some data
t = np.arange(0, 2, 0.01)
y1 = np.sin(2*np.pi * t)
y2 = np.cos(2*np.pi * t)
y3 = np.exp(t)
y4 = np.exp(-t)

# Initialize the grid with 3 rows and 3 columns
ncols = 3
nrows = 3
grid = GridSpec(nrows, ncols,
                left=0.1, bottom=0.15, right=0.94, top=0.94, wspace=0.3, hspace=0.3)

fig = plt.figure(0)
fig.clf()

# Add axes which can span multiple grid boxes
ax1 = fig.add_subplot(grid[0:2, 0:2])
ax2 = fig.add_subplot(grid[0:2, 2])
ax3 = fig.add_subplot(grid[2, 0:2])
ax4 = fig.add_subplot(grid[2, 2])

ax1.plot(t, y1, color='royalblue')
ax2.plot(t, y2, color='forestgreen')
ax3.plot(t, y3, color='darkorange')
ax4.plot(t, y4, color='darkmagenta')

# Add labels and titles
fig.suptitle('Figure with Subplots')
ax1.set_ylabel('Voltage (V)')
ax3.set_ylabel('Voltage (V)')
ax3.set_xlabel('Time (s)')
ax4.set_xlabel('Time (s)')
```

Este código produce la gráfica que se muestra a continuación.

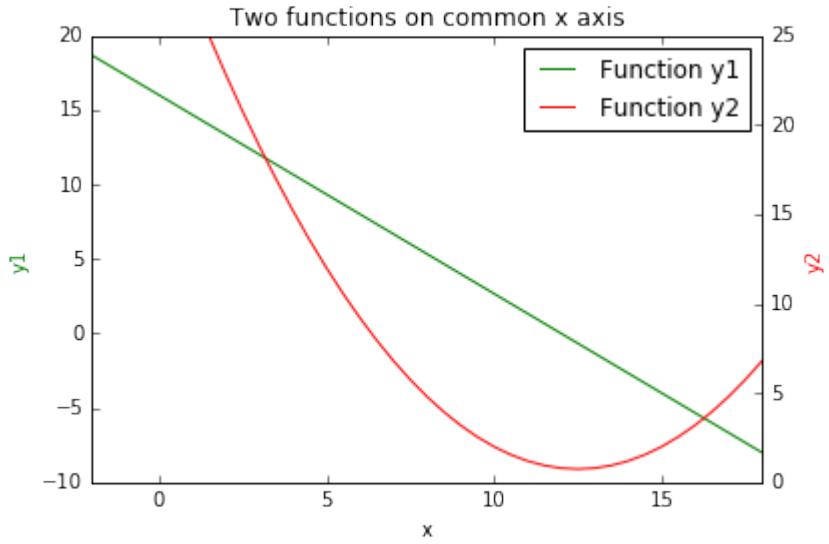


Un gráfico de 2 funciones en el eje x compartido.

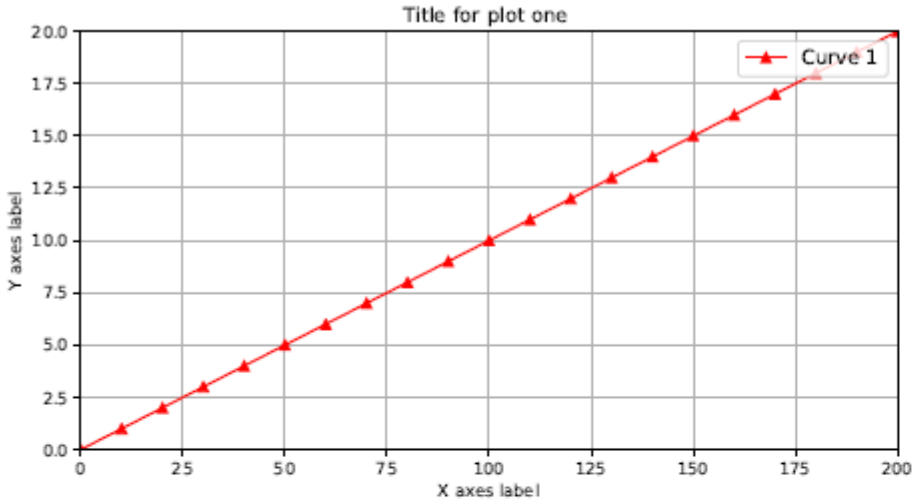
```
import numpy as np
import matplotlib.pyplot as plt

# create some data
x = np.arange(-2, 20, 0.5)                  # values of x
y1 = map(lambda x: -4.0/3.0*x + 16, x)      # values of y1(x)
y2 = map(lambda x: 0.2*x**2 -5*x + 32, x)   # svalues of y2(x)

fig = plt.figure()
ax1 = fig.add_subplot(111)

# create line plot of y1(x)
line1, = ax1.plot(x, y1, 'g', label="Function y1")
ax1.set_xlabel('x')
ax1.set_ylabel('y1', color='g')

# create shared axis for y2(x)
ax2 = ax1.twinx()
```

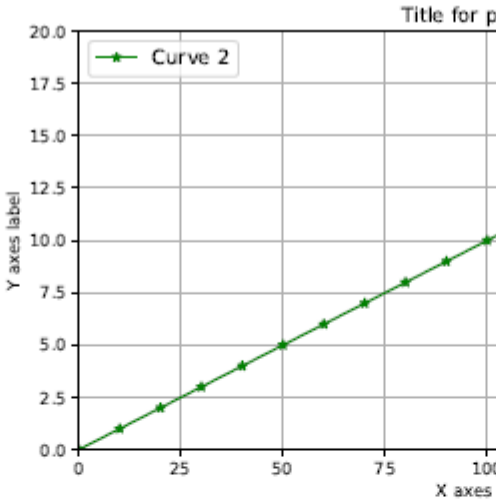
```
# create line plot of y2(x)
line2, = ax2.plot(x, y2, 'r', label="Function y2")
ax2.set_ylabel('y2', color='r')

# set title, plot limits, etc
plt.title('Two functions on common x axis')
plt.xlim(-2, 18)
plt.ylim(0, 25)

# add a legend, and position it on the upper right
plt.legend((line1, line2), ('Function y1', 'Function y2'))

plt.show()
```

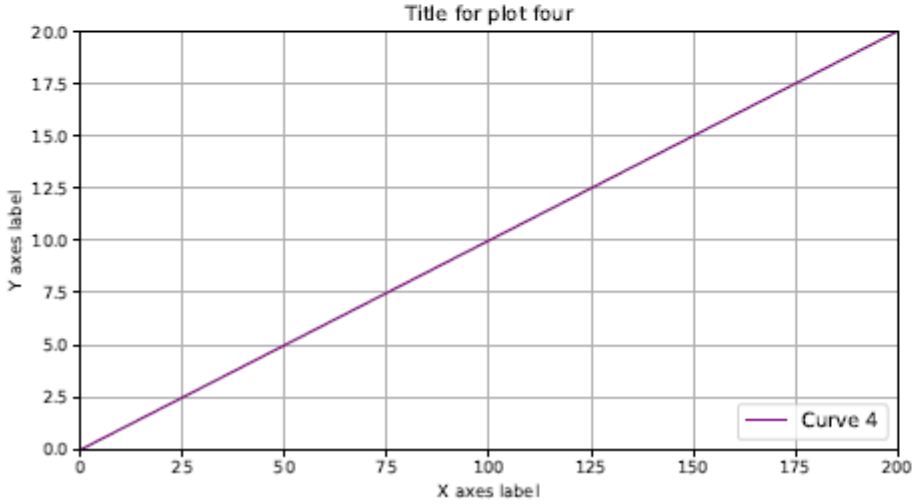
Este código produce la gráfica que se muestra a continuación.

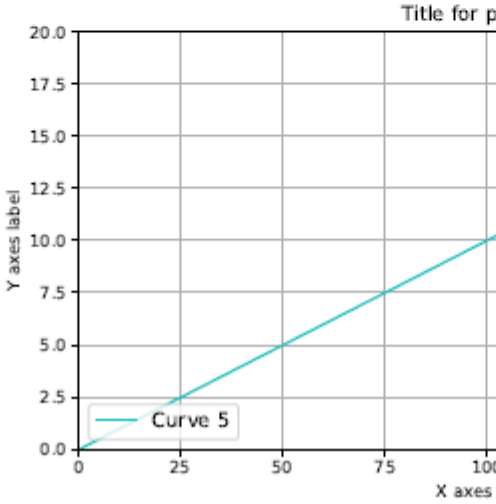


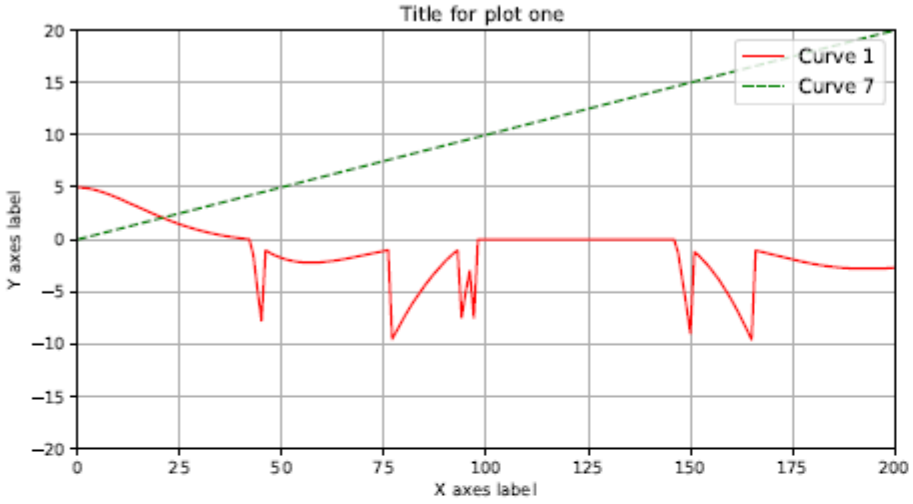
Parcelas múltiples y características de parcelas múltiples

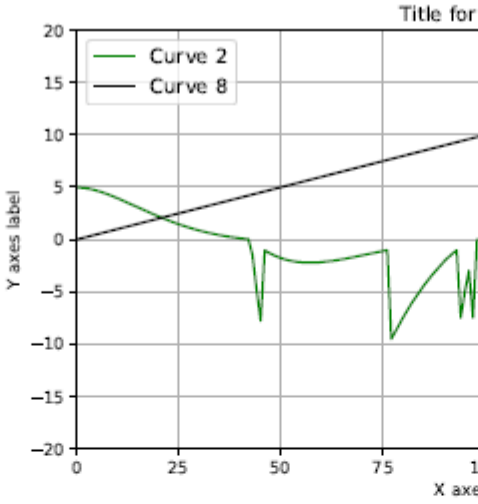

Title for plot one
Curve 1
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200

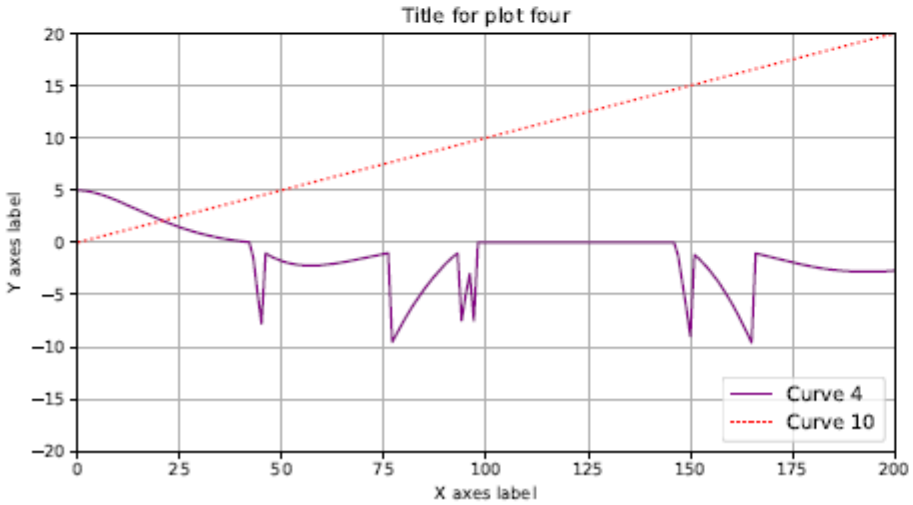

Curve 2
Y axes label
X axes

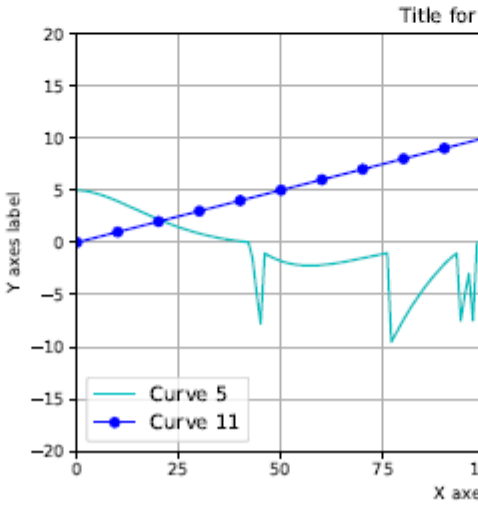

Title for plot four
Curve 4
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200


Curve 5
Y axes label
X axes









CAE.csv

```
TIME,Acceleration
0,4.992235
0.09952711,4.956489
0.1999273,4.915645
0.2994544,4.850395
0.3998545,4.763977
0.4993816,4.65888
0.5997818,4.537595
0.6993089,4.402862
0.799709,4.256423
0.8992361,4.100522
0.9996362,3.937148
1.099163,3.768047
1.199564,3.579082
```

```
import matplotlib
matplotlib.use("TKAgg")

# module to save pdf files
from matplotlib.backends.backend_pdf import PdfPages

import matplotlib.pyplot as plt  # module to plot

import pandas as pd  # module to read csv file
```

```
# module to allow user to select csv file
from tkinter.filedialog import askopenfilename

# module to allow user to select save directory
from tkinter.filedialog import askdirectory


#=============================================================================
#  User chosen Data for plots
#=============================================================================

# User choose csv file then read csv file
filename = askopenfilename() # user selected file
data = pd.read_csv(filename, delimiter=',')

# check to see if data is reading correctly
#print(data)

#=============================================================================
#  Plots on two different Figures and sets the size of the figures
#=============================================================================

# figure size = (width,height)
f1 = plt.figure(figsize=(30,10))
f2 = plt.figure(figsize=(30,10))

#-----------------------------------------------------------------------------
#  Figure 1 with 6 plots
#-----------------------------------------------------------------------------

# plot one
# Plot column labeled TIME from csv file and color it red
# subplot(2 Rows, 3 Columns, First subplot,)
ax1 = f1.add_subplot(2,3,1)
ax1.plot(data[["TIME"]], label = 'Curve 1', color = "r", marker = '^', markevery = 10)
# added line marker triangle


# plot two
# plot column labeled TIME from csv file and color it green
# subplot(2 Rows, 3 Columns, Second subplot)
ax2 = f1.add_subplot(2,3,2)
ax2.plot(data[["TIME"]], label = 'Curve 2', color = "g", marker = '*', markevery = 10)
# added line marker star


# plot three
# plot column labeled TIME from csv file and color it blue
# subplot(2 Rows, 3 Columns, Third subplot)
ax3 = f1.add_subplot(2,3,3)
ax3.plot(data[["TIME"]], label = 'Curve 3', color = "b", marker = 'D', markevery = 10)
# added line marker diamond


# plot four
# plot column labeled TIME from csv file and color it purple
# subplot(2 Rows, 3 Columns, Fourth subplot)
ax4 = f1.add_subplot(2,3,4)
ax4.plot(data[["TIME"]], label = 'Curve 4', color = "#800080")
```

```
# plot five
# plot column labeled TIME from csv file and color it cyan
# subplot(2 Rows, 3 Columns, Fifth subplot)
ax5 = f1.add_subplot(2,3,5)
ax5.plot(data[["TIME"]], label = 'Curve 5', color = "c")


# plot six
# plot column labeled TIME from csv file and color it black
# subplot(2 Rows, 3 Columns, Sixth subplot)
ax6 = f1.add_subplot(2,3,6)
ax6.plot(data[["TIME"]], label = 'Curve 6', color = "k")

#-------------------------------------------------------------------------------
# Figure 2 with 6 plots
#-------------------------------------------------------------------------------

# plot one
# Curve 1: plot column labeled Acceleration from csv file and color it red
# Curve 2: plot column labeled     TIME     from csv file and color it green
# subplot(2 Rows, 3 Columns, First subplot)
ax10 = f2.add_subplot(2,3,1)
ax10.plot(data[["Acceleration"]], label = 'Curve 1', color = "r")
ax10.plot(data[["TIME"]], label = 'Curve 7', color="g", linestyle ='--')
# dashed line


# plot two
# Curve 1: plot column labeled Acceleration from csv file and color it green
# Curve 2: plot column labeled     TIME     from csv file and color it black
# subplot(2 Rows, 3 Columns, Second subplot)
ax20 = f2.add_subplot(2,3,2)
ax20.plot(data[["Acceleration"]], label = 'Curve 2', color = "g")
ax20.plot(data[["TIME"]], label = 'Curve 8', color = "k", linestyle ='-')
# solid line (default)


# plot three
# Curve 1: plot column labeled Acceleration from csv file and color it blue
# Curve 2: plot column labeled     TIME     from csv file and color it purple
# subplot(2 Rows, 3 Columns, Third subplot)
ax30 = f2.add_subplot(2,3,3)
ax30.plot(data[["Acceleration"]], label = 'Curve 3', color = "b")
ax30.plot(data[["TIME"]], label = 'Curve 9', color = "#800080", linestyle ='-.')
# dash_dot line


# plot four
# Curve 1: plot column labeled Acceleration from csv file and color it purple
# Curve 2: plot column labeled     TIME     from csv file and color it red
# subplot(2 Rows, 3 Columns, Fourth subplot)
ax40 = f2.add_subplot(2,3,4)
ax40.plot(data[["Acceleration"]], label = 'Curve 4', color = "#800080")
ax40.plot(data[["TIME"]], label = 'Curve 10', color = "r", linestyle =':')
# dotted line


# plot five
# Curve 1: plot column labeled Acceleration from csv file and color it cyan
# Curve 2: plot column labeled     TIME     from csv file and color it blue
# subplot(2 Rows, 3 Columns, Fifth subplot)
```

```
ax50 = f2.add_subplot(2,3,5)
ax50.plot(data[["Acceleration"]], label = 'Curve 5', color = "c")
ax50.plot(data[["TIME"]], label = 'Curve 11', color = "b", marker = 'o', markevery = 10)
# added line marker circle


# plot six
# Curve 1: plot column labeled Acceleration from csv file and color it black
# Curve 2: plot column labeled     TIME     from csv file and color it cyan
# subplot(2 Rows, 3 Columns, Sixth subplot)
ax60 = f2.add_subplot(2,3,6)
ax60.plot(data[["Acceleration"]], label = 'Curve 6', color = "k")
ax60.plot(data[["TIME"]], label = 'Curve 12', color = "c", marker = 's', markevery = 10)
# added line marker square


#============================================================================
#  Figure Plot options
#============================================================================

#----------------------------------------------------------------------------
#  Figure 1 options
#----------------------------------------------------------------------------

#switch to figure one for editing
plt.figure(1)

# Plot one options
ax1.legend(loc='upper right', fontsize='large')
ax1.set_title('Title for plot one ')
ax1.set_xlabel('X axes label')
ax1.set_ylabel('Y axes label')
ax1.grid(True)
ax1.set_xlim([0,200])
ax1.set_ylim([0,20])

# Plot two options
ax2.legend(loc='upper left', fontsize='large')
ax2.set_title('Title for plot two ')
ax2.set_xlabel('X axes label')
ax2.set_ylabel('Y axes label')
ax2.grid(True)
ax2.set_xlim([0,200])
ax2.set_ylim([0,20])

# Plot three options
ax3.legend(loc='upper center', fontsize='large')
ax3.set_title('Title for plot three ')
ax3.set_xlabel('X axes label')
ax3.set_ylabel('Y axes label')
ax3.grid(True)
ax3.set_xlim([0,200])
ax3.set_ylim([0,20])

# Plot four options
ax4.legend(loc='lower right', fontsize='large')
ax4.set_title('Title for plot four')
ax4.set_xlabel('X axes label')
ax4.set_ylabel('Y axes label')
ax4.grid(True)
ax4.set_xlim([0,200])
```

```
ax4.set_ylim([0,20])

# Plot five options
ax5.legend(loc='lower left', fontsize='large')
ax5.set_title('Title for plot five ')
ax5.set_xlabel('X axes label')
ax5.set_ylabel('Y axes label')
ax5.grid(True)
ax5.set_xlim([0,200])
ax5.set_ylim([0,20])

# Plot six options
ax6.legend(loc='lower center', fontsize='large')
ax6.set_title('Title for plot six')
ax6.set_xlabel('X axes label')
ax6.set_ylabel('Y axes label')
ax6.grid(True)
ax6.set_xlim([0,200])
ax6.set_ylim([0,20])

#------------------------------------------------------------------------------
#  Figure 2 options
#------------------------------------------------------------------------------

#switch to figure two for editing
plt.figure(2)

# Plot one options
ax10.legend(loc='upper right', fontsize='large')
ax10.set_title('Title for plot one ')
ax10.set_xlabel('X axes label')
ax10.set_ylabel('Y axes label')
ax10.grid(True)
ax10.set_xlim([0,200])
ax10.set_ylim([-20,20])

# Plot two options
ax20.legend(loc='upper left', fontsize='large')
ax20.set_title('Title for plot two ')
ax20.set_xlabel('X axes label')
ax20.set_ylabel('Y axes label')
ax20.grid(True)
ax20.set_xlim([0,200])
ax20.set_ylim([-20,20])

# Plot three options
ax30.legend(loc='upper center', fontsize='large')
ax30.set_title('Title for plot three ')
ax30.set_xlabel('X axes label')
ax30.set_ylabel('Y axes label')
ax30.grid(True)
ax30.set_xlim([0,200])
ax30.set_ylim([-20,20])

# Plot four options
ax40.legend(loc='lower right', fontsize='large')
ax40.set_title('Title for plot four')
ax40.set_xlabel('X axes label')
ax40.set_ylabel('Y axes label')
ax40.grid(True)
ax40.set_xlim([0,200])
```

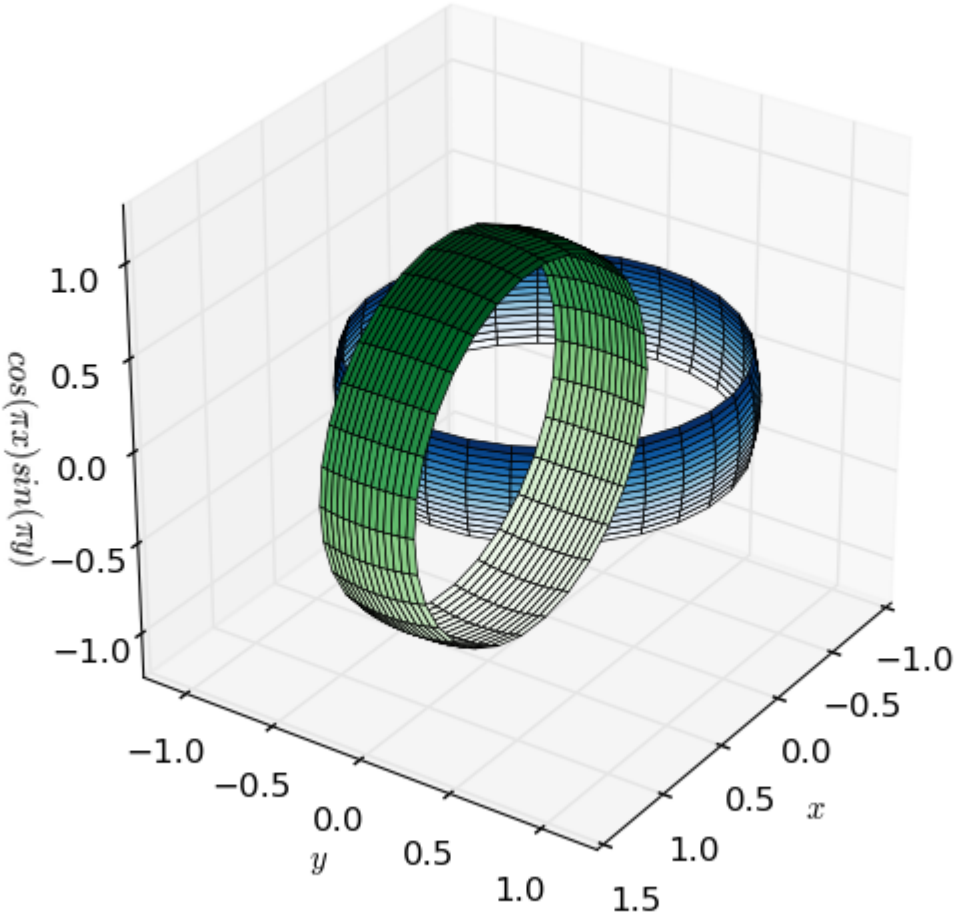
```
ax40.set_ylim([-20,20])

# Plot five options
ax50.legend(loc='lower left', fontsize='large')
ax50.set_title('Title for plot five ')
ax50.set_xlabel('X axes label')
ax50.set_ylabel('Y axes label')
ax50.grid(True)
ax50.set_xlim([0,200])
ax50.set_ylim([-20,20])

# Plot six options
ax60.legend(loc='lower center', fontsize='large')
ax60.set_title('Title for plot six')
ax60.set_xlabel('X axes label')
ax60.set_ylabel('Y axes label')
ax60.grid(True)
ax60.set_xlim([0,200])
ax60.set_ylim([-20,20])


#============================================================================
#  User chosen file location Save PDF
#============================================================================

savefilename = askdirectory()# user selected file path
pdf = PdfPages(f'{savefilename}/longplot.pdf')
# using formatted string literals ("f-strings")to place the variable into the string

# save both figures into one pdf file
pdf.savefig(1)
pdf.savefig(2)

pdf.close()


#============================================================================
#  Show plot
#============================================================================

# manually set the subplot spacing when there are multiple plots
#plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace =None, hspace=None )

# Automaticlly adds space between plots
plt.tight_layout()

plt.show()
```

Lea Parcelas Múltiples en línea: https://riptutorial.com/es/matplotlib/topic/3279/parcelas-multiples

Capítulo 17: Parcelas tridimensionales

Observaciones

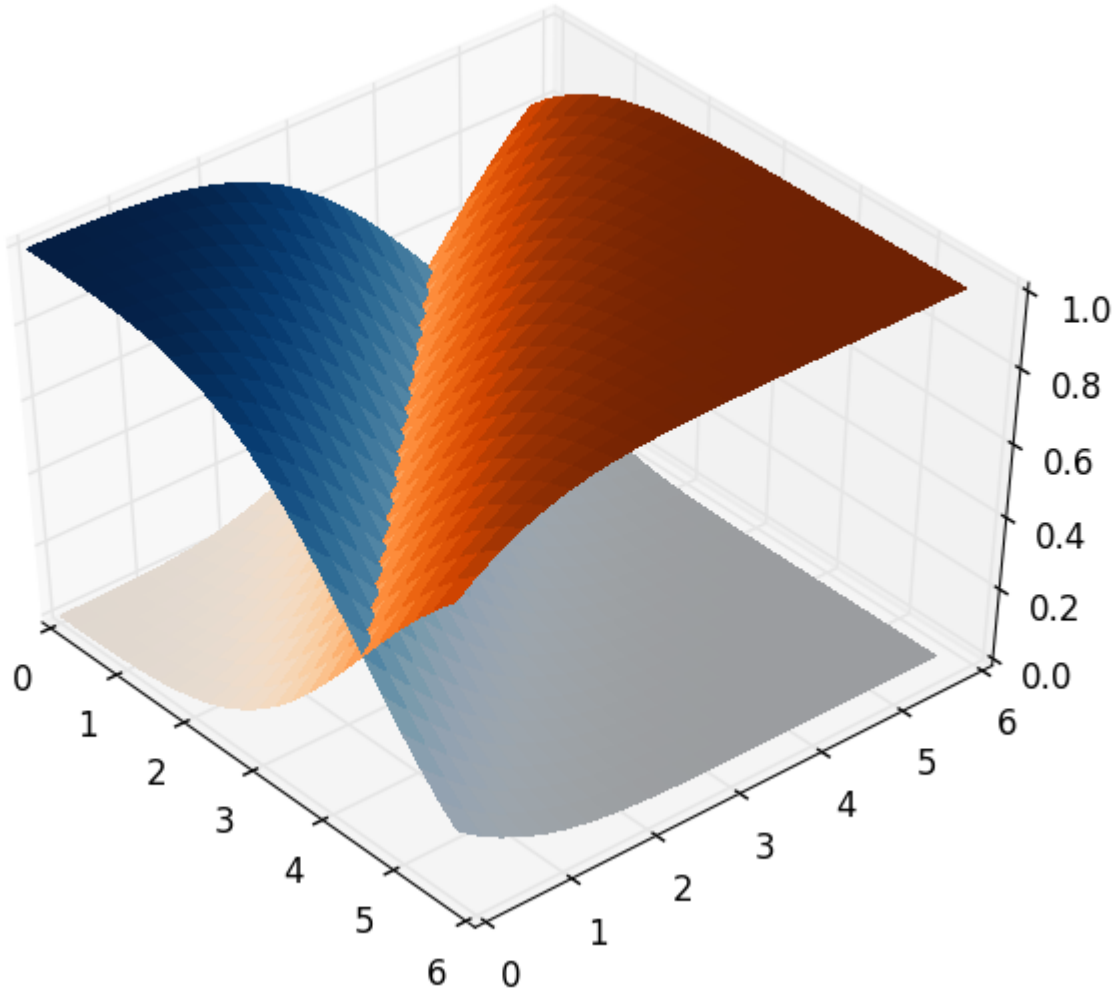
El trazado tridimensional en matplotlib ha sido históricamente un poco kludge, ya que el motor de renderizado es inherentemente 2D. El hecho de que las configuraciones 3d se representen trazando una porción 2d después de la otra implica que a menudo hay problemas relacionados con la profundidad aparente de los objetos. El núcleo del problema es que dos objetos no conectados pueden estar completamente detrás o completamente uno frente al otro, lo que lleva a artefactos como se muestra en la siguiente figura de dos anillos entrelazados (haga clic para ver un gif animado):

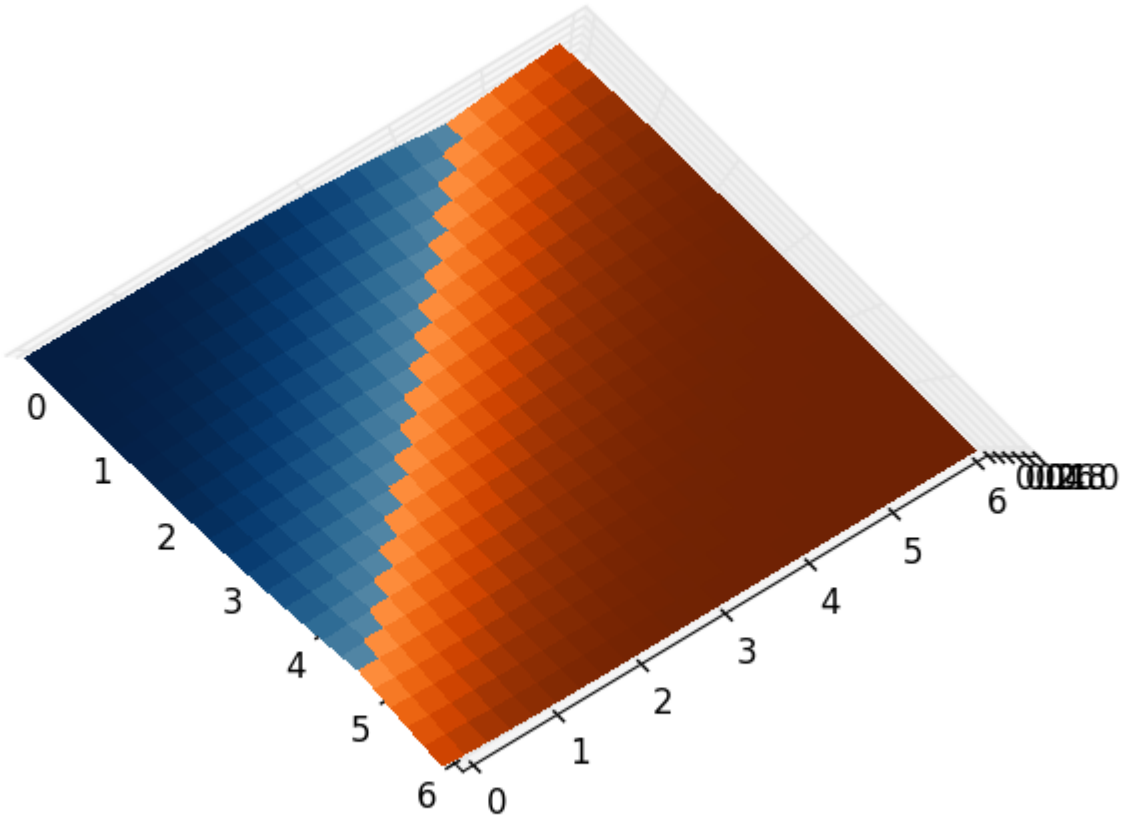


Sin embargo, esto puede ser arreglado. Este artefacto solo existe cuando se trazan múltiples superficies en el mismo trazado, ya que cada una se representa como una forma plana 2D, con un solo parámetro que determina la distancia de la vista. Notará que una sola superficie complicada no sufre el mismo problema.

La forma de remediar esto es unir los objetos de la parcela utilizando puentes transparentes:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import erf

fig = plt.figure()
ax = fig.gca(projection='3d')

X = np.arange(0, 6, 0.25)
Y = np.arange(0, 6, 0.25)
X, Y = np.meshgrid(X, Y)

Z1 = np.empty_like(X)
Z2 = np.empty_like(X)
C1 = np.empty_like(X, dtype=object)
C2 = np.empty_like(X, dtype=object)

for i in range(len(X)):
  for j in range(len(X[0])):
    z1 = 0.5*(erf((X[i,j]+Y[i,j]-4.5)*0.5)+1)
    z2 = 0.5*(erf((-X[i,j]-Y[i,j]+4.5)*0.5)+1)
    Z1[i,j] = z1
    Z2[i,j] = z2

    # If you want to grab a colour from a matplotlib cmap function,
    # you need to give it a number between 0 and 1. z1 and z2 are
    # already in this range, so it just works as is.
    C1[i,j] = plt.get_cmap("Oranges")(z1)
    C2[i,j] = plt.get_cmap("Blues")(z2)


# Create a transparent bridge region
X_bridge = np.vstack([X[-1,:],X[-1,:]])
Y_bridge = np.vstack([Y[-1,:],Y[-1,:]])
Z_bridge = np.vstack([Z1[-1,:],Z2[-1,:]])
color_bridge = np.empty_like(Z_bridge, dtype=object)

color_bridge.fill((1,1,1,0)) # RGBA colour, onlt the last component matters - it represents
the alpha / opacity.

# Join the two surfaces flipping one of them (using also the bridge)
X_full = np.vstack([X, X_bridge, np.flipud(X)])
Y_full = np.vstack([Y, Y_bridge, np.flipud(Y)])
Z_full = np.vstack([Z1, Z_bridge, np.flipud(Z2)])
color_full = np.vstack([C1, color_bridge, np.flipud(C2)])

surf_full = ax.plot_surface(X_full, Y_full, Z_full, rstride=1, cstride=1,
                            facecolors=color_full, linewidth=0,
                            antialiased=False)

plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
0
1
2
3
4
5
6



Examples

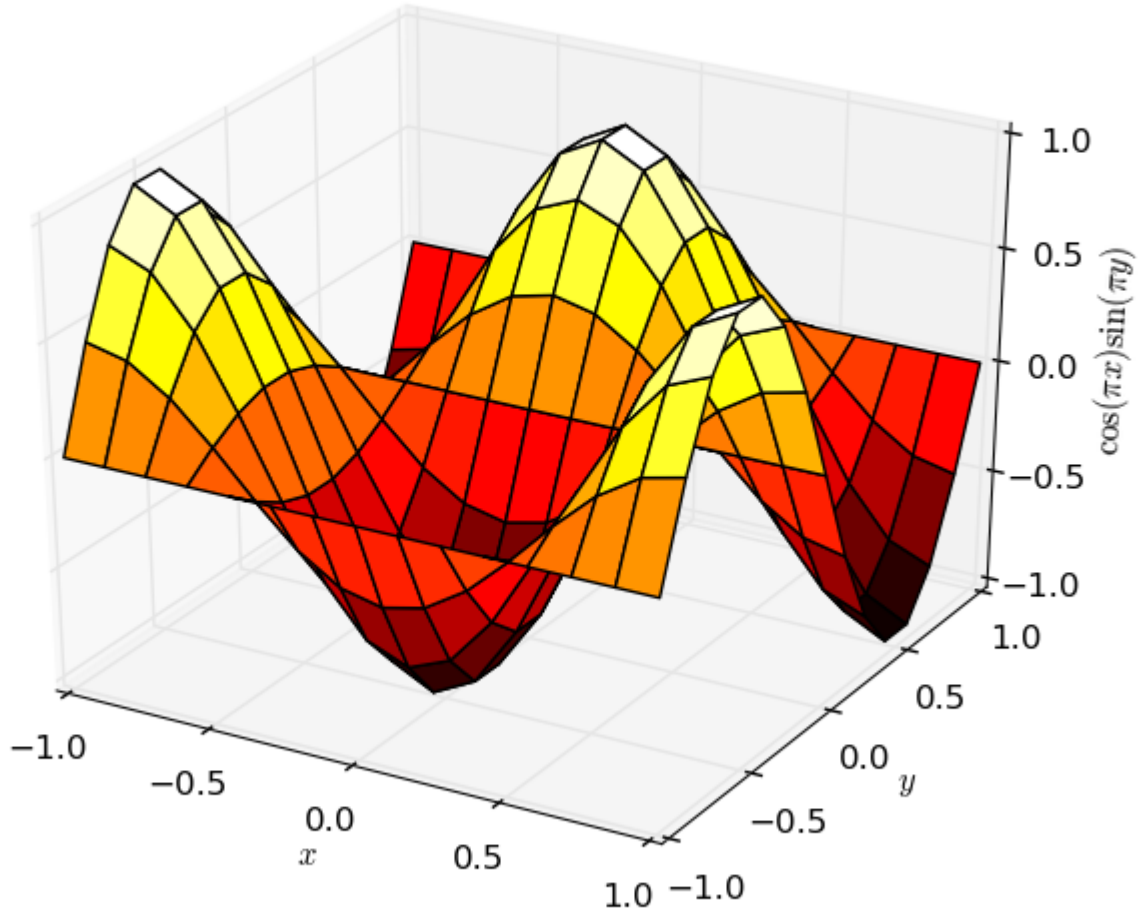
Creando ejes tridimensionales.

Los ejes de matplotlib son bidimensionales por defecto. Con el fin de crear gráficos de tres dimensiones, tenemos que importar el `Axes3D` clase de la caja de herramientas mplot3d , que permitirá a un nuevo tipo de proyección para una ejes, a saber `'3d'` :

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

Además de las generalizaciones sencillas de las gráficas bidimensionales (como las gráficas lineales , las gráficas de dispersión , las gráficas de barras , las gráficas de contorno), hay varios métodos de trazado de superficie disponibles, por ejemplo, `ax.plot_surface` :

```
# generate example data
import numpy as np
x,y = np.meshgrid(np.linspace(-1,1,15),np.linspace(-1,1,15))
z = np.cos(x*np.pi)*np.sin(y*np.pi)

# actual plotting example
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# rstride and cstride are row and column stride (step size)
ax.plot_surface(x,y,z,rstride=1,cstride=1,cmap='hot')
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_zlabel(r'$\cos(\pi x) \sin(\pi y)$')
plt.show()
```



Lea Parcelas tridimensionales en línea: https://riptutorial.com/es/matplotlib/topic/1880/parcelas-tridimensionales

Capítulo 18: Sistemas de coordenadas

Observaciones

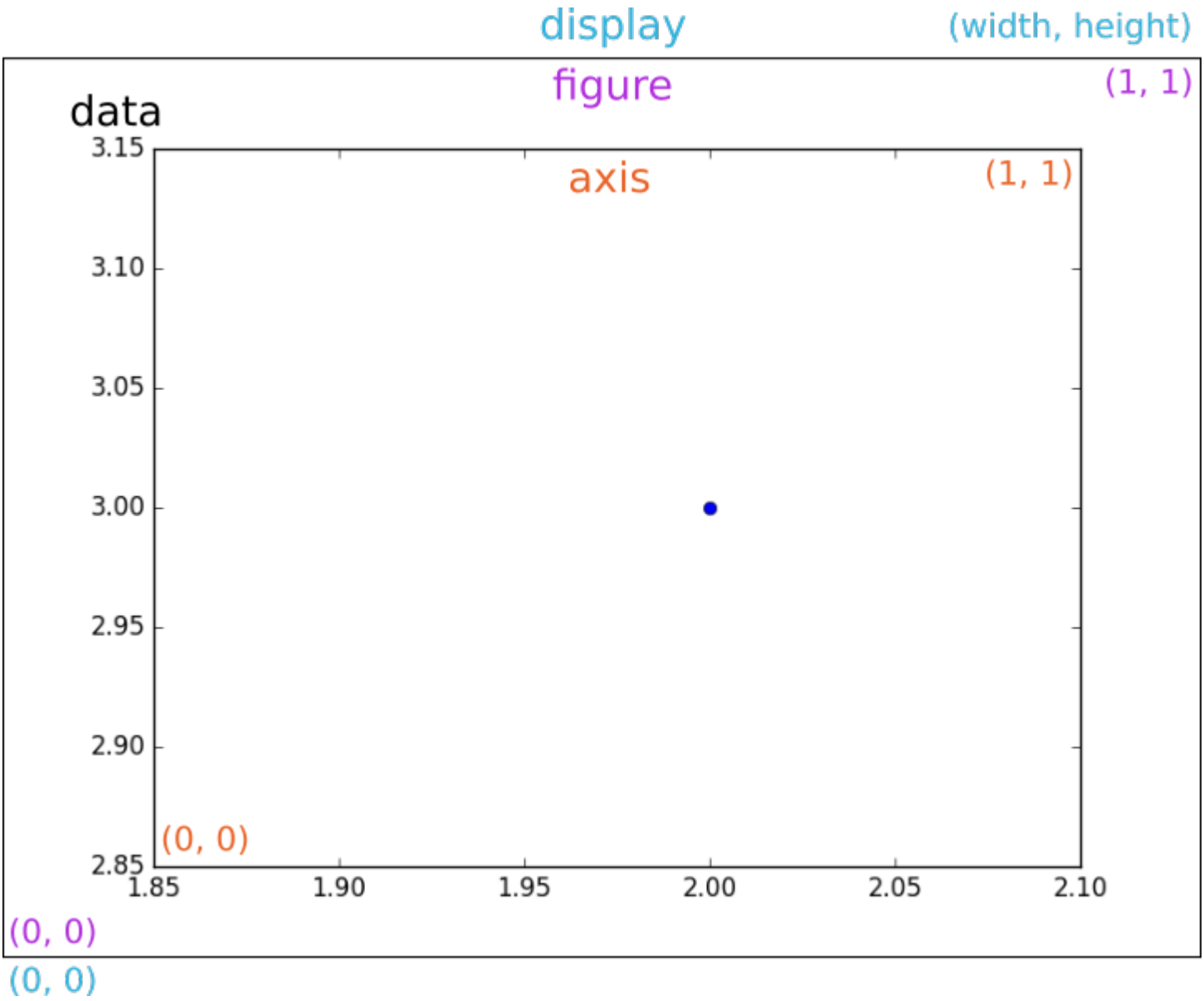
Matplotlib tiene cuatro sistemas de coordenadas distintos que pueden aprovecharse para facilitar el posicionamiento de diferentes objetos, por ejemplo, texto. Cada sistema tiene un objeto de transformación correspondiente que transforma las coordenadas de ese sistema al llamado sistema de coordenadas de visualización.

El sistema de coordenadas de datos es el sistema definido por los datos en los ejes respectivos. Es útil cuando se intenta colocar algún objeto en relación con los datos trazados. El rango viene dado por las propiedades `xlim` y `ylim` de `Axes` . Su objeto de transformación correspondiente es `ax.transData` .

El sistema de coordenadas de ejes es el sistema vinculado a su objeto `Axes` . Los puntos (0, 0) y (1, 1) definen las esquinas inferior izquierda y superior derecha de los ejes. Como tal, es útil cuando se coloca en relación con los ejes, como el centro superior de la trama. Su objeto de transformación correspondiente es `ax.transAxes` .

El sistema de coordenadas de la figura es análogo al sistema de coordenadas de los ejes, excepto que está vinculado a la `Figure` . Los puntos (0, 0) y (1, 1) representan las esquinas inferior izquierda y superior derecha de la figura. Es útil cuando se trata de posicionar algo relativo a toda la imagen. Su objeto de transformación correspondiente es `fig.transFigure` .

El sistema de coordenadas de visualización es el sistema de la imagen dada en píxeles. Los puntos (0, 0) y (ancho, alto) son los píxeles de la parte inferior izquierda y superior derecha de la imagen o visualización. Puede ser utilizado para posicionamiento absolutamente. Como los objetos de transformación transforman las coordenadas en este sistema de coordenadas, el sistema de visualización no tiene asociado ningún objeto de transformación. Sin embargo, `None` o `matplotlib.transforms.IdentityTransform()` se puede usar cuando sea necesario.



Más detalles están disponibles aquí .

Examples

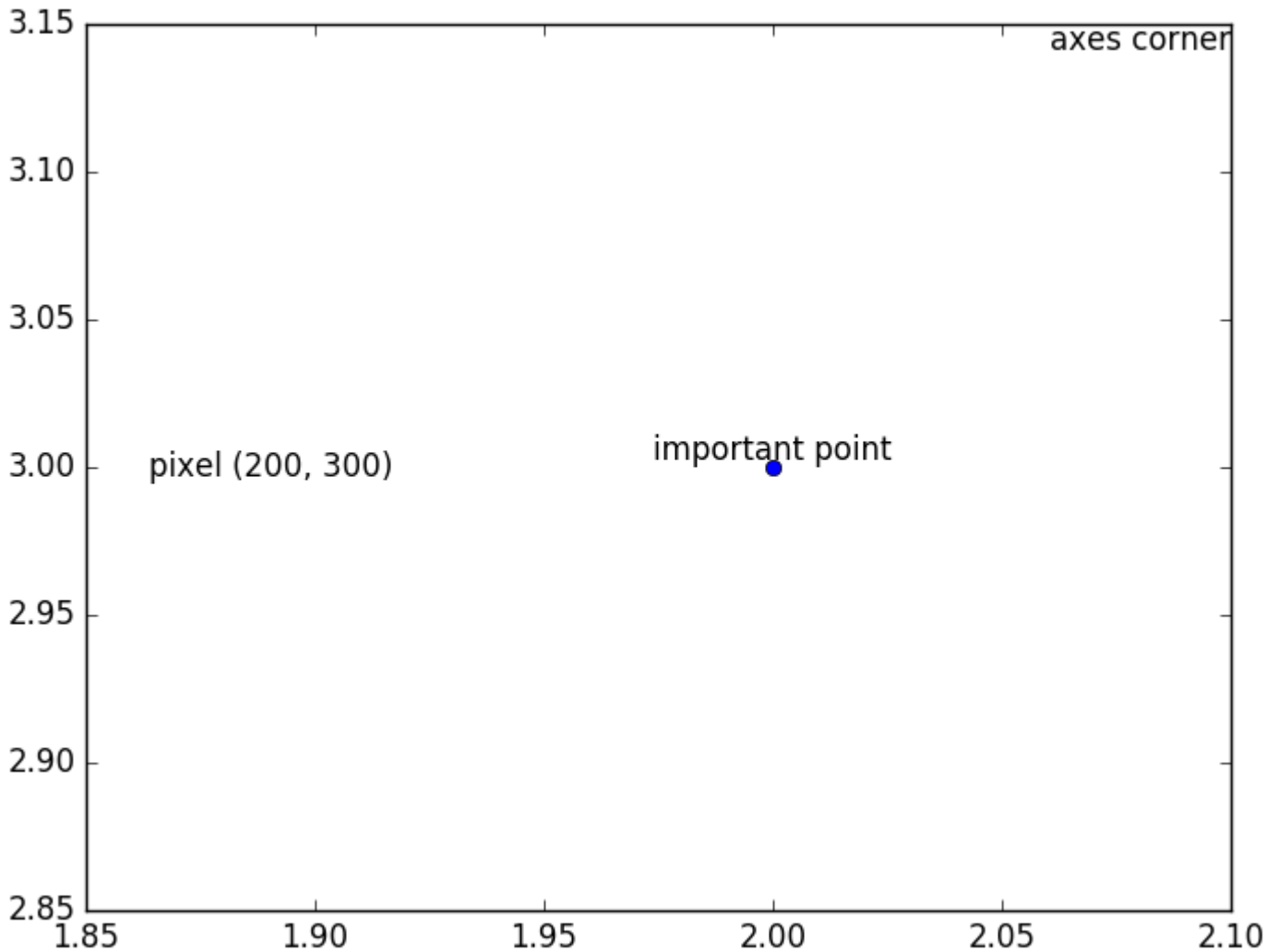
Sistemas de coordenadas y texto.

Los sistemas de coordenadas de Matplotlib son muy útiles al tratar de anotar los gráficos que realiza. A veces, le gustaría colocar el texto en relación con sus datos, como cuando intenta etiquetar un punto específico. Otras veces tal vez le gustaría agregar un texto en la parte superior de la figura. Esto se puede lograr fácilmente seleccionando un sistema de coordenadas apropiado pasando un objeto de `transform` parámetro de `transform` en llamada a `text()` .

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot([2.], [3.], 'bo')

plt.text(  # position text relative to data
```

```
    2., 3., 'important point',  # x, y, text,
    ha='center', va='bottom',   # text alignment,
    transform=ax.transData      # coordinate system transformation
)
plt.text(  # position text relative to Axes
    1.0, 1.0, 'axes corner',
    ha='right', va='top',
    transform=ax.transAxes
)
plt.text(  # position text relative to Figure
    0.0, 1.0, 'figure corner',
    ha='left', va='top',
    transform=fig.transFigure
)
plt.text(  # position text absolutely at specific pixel on image
    200, 300, 'pixel (200, 300)',
    ha='center', va='center',
    transform=None
)

plt.show()
```

figure corner



Lea Sistemas de coordenadas en línea: https://riptutorial.com/es/matplotlib/topic/4566/sistemas-de-coordenadas

Creditos

S. No	Capítulos	Contributors
1	Empezando con matplotlib	Amitay Stern, ChaoticTwist, Chr, Chris Mueller, Community, dermen, evtoh, farenorth, Josh, jrjc, pmos, Serenity, tacaswell
2	Animaciones y tramas interactivas.	FiN, smurfendrek123, user2314737
3	Cerrar una ventana de figura	Brian, David Zwicker
4	Colormaps	Andras Deak, Xevaquor
5	Figuras y objetos de ejes	David Zwicker, Josh, Serenity, tom
6	Gráficas de caja	Luis
7	Histograma	Yegor Kishilov
8	Integración con TeX / LaTeX	Andras Deak, Bosoneando, Chris Mueller, Næreen, Serenity
9	Leyendas	Andras Deak, Franck Dernoncourt, ronrest, saintsfan342000, Serenity
10	Líneas de cuadrícula y marcas de garrapatas	ronrest
11	LogLog Graphing	ml4294
12	Manipulación de imagen	Bosoneando
13	Mapas de contorno	Eugene Loy, Serenity
14	Parcelas básicas	Franck Dernoncourt, Josh, ml4294, ronrest, Scimonster, Serenity, user2314737
15	Parcelas Múltiples	Chris Mueller, Robert Branam, ronrest, swatchai
16	Parcelas tridimensionales	Andras Deak, Serenity, will

17	Sistemas de coordenadas	jure