EBook Gratuito

APPRENDIMENTO matplotlib

Free unaffiliated eBook created from
Stack Overflow contributors.

#matplotlib

Sommario

Di

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capitolo 1: Iniziare con matplotlib

Osservazioni

Panoramica

matplotlib è una libreria di plottaggio per Python. Fornisce API orientate agli oggetti per incorporare grafici in applicazioni. È simile a MATLAB in termini di capacità e sintassi.

È stato originariamente scritto da JDHunter ed è in fase di sviluppo. È distribuito sotto una licenza BSD-Style.

Versioni

Versione	Versioni di Python supportate	Osservazioni	Data di rilascio
1.3.1	2.6, 2.7, 3.x	Versione stabile precedente	2013/10/10
1.4.3	2.6, 2.7, 3.x	Versione stabile precedente	2015/07/14
1.5.3	2.7, 3.x	Versione stabile attuale	2016/01/11
2.x	2.7, 3.x	Ultima versione di sviluppo	2016/07/25

Examples

Installazione e configurazione

Esistono diversi modi per installare matplotlib, alcuni dei quali dipenderanno dal sistema che si sta utilizzando. Se sei fortunato, sarai in grado di utilizzare un gestore di pacchetti per installare facilmente il modulo matplotlib e le sue dipendenze.

finestre

Su macchine Windows puoi provare a utilizzare il gestore di pacchetti pip per installare matplotlib. Vedi qui per informazioni sulla configurazione di pip in un ambiente Windows.

OS X

Si consiglia di utilizzare il gestore di pacchetti pip per installare matplotlib. Se hai bisogno di installare alcune delle librerie non Python sul tuo sistema (es. `libfreetype`) allora considera di

usare homebrew .

Se non è possibile utilizzare pip per qualsiasi motivo, provare a installare dal sorgente .

Linux

Idealmente, il gestore di pacchetti di sistema o pip dovrebbe essere usato per installare matplotlib, sia installando il pacchetto `python-matplotlib` sia eseguendo il programma di `pip install matplotlib` .

Se ciò non è possibile (ad esempio non si dispone dei privilegi sudo sulla macchina che si sta utilizzando), è possibile eseguire l'installazione dal sorgente utilizzando l'opzione `--user` : `python setup.py install --user` . Tipicamente, questo installerà matplotlib in `~/.local` .

Debian / Ubuntu

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

Fedora / Red Hat

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

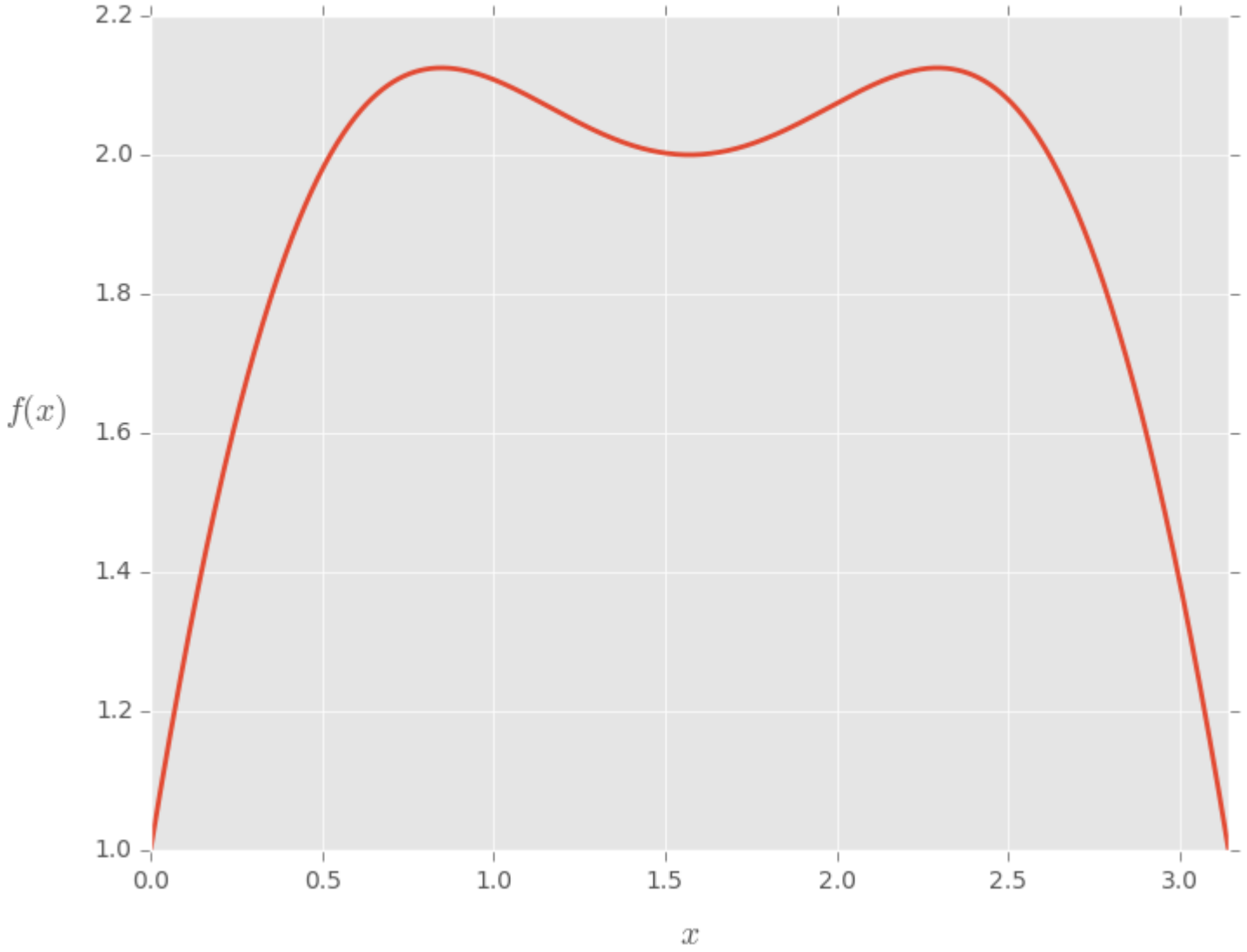
Risoluzione dei problemi

Vedere il sito web matplotlib per consigli su come riparare un matplotlib rotto.

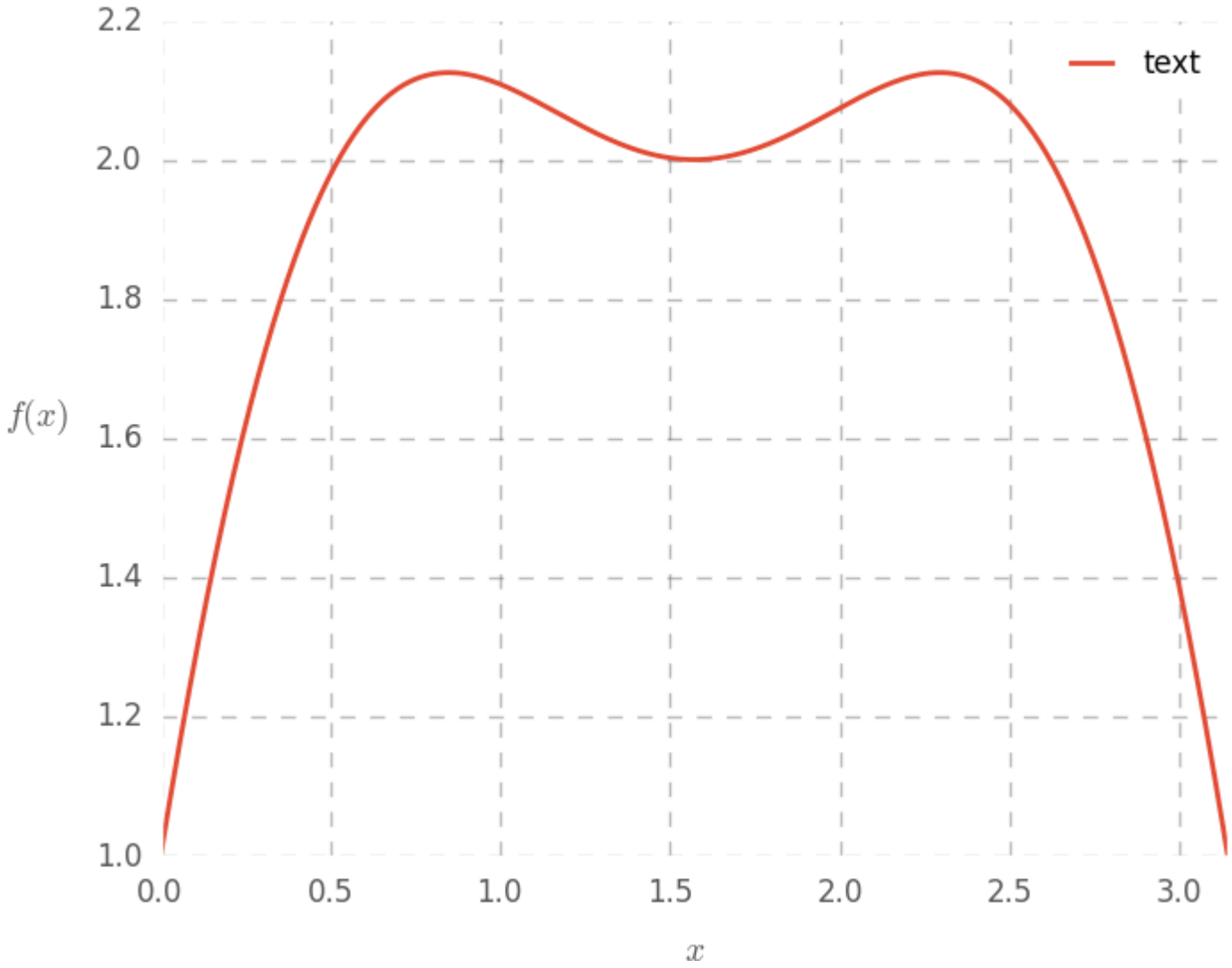
Personalizzazione di un grafico matplotlib

```
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()
```



Sintassi imperativa e orientata agli oggetti

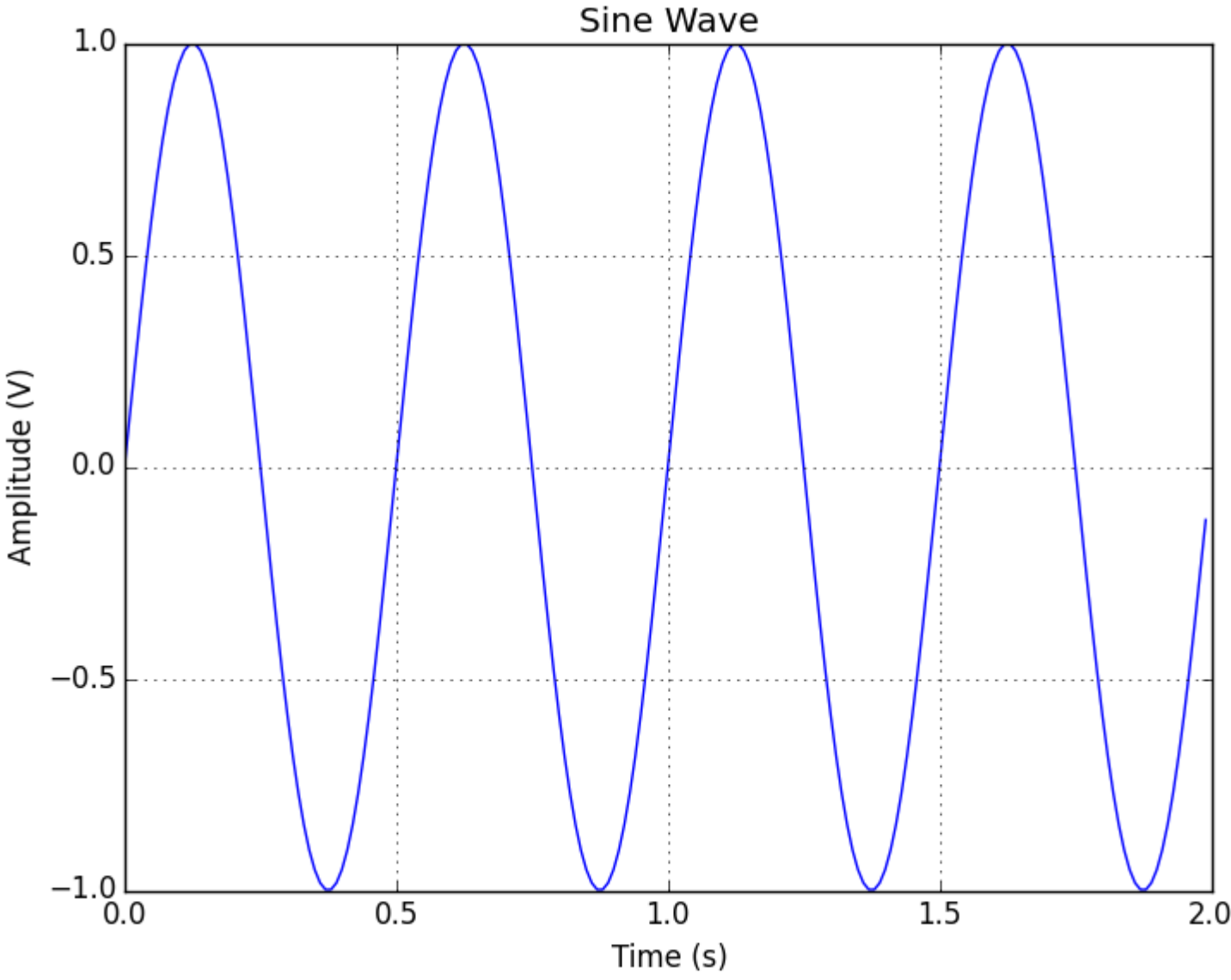
Matplotlib supporta la sintassi sia orientata agli oggetti che imperativa per la stampa. La sintassi imperativa è intenzionalmente progettata per essere molto vicina alla sintassi di Matlab.

La sintassi imperativa (talvolta chiamata sintassi 'state-machine') emette una stringa di comandi che agiscono tutti sulla figura o sull'asse più recenti (come Matlab). La sintassi orientata agli oggetti, d'altra parte, agisce esplicitamente sugli oggetti (figura, asse, ecc.) Di interesse. Un punto chiave nello zen di Python afferma che l'esplicito è meglio di quello implicito, quindi la sintassi orientata agli oggetti è più pitonica. Tuttavia, la sintassi imperativa è conveniente per i nuovi convertiti da Matlab e per la scrittura di piccoli script di trama "usa e getta". Di seguito è riportato un esempio dei due diversi stili.

```
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)
```

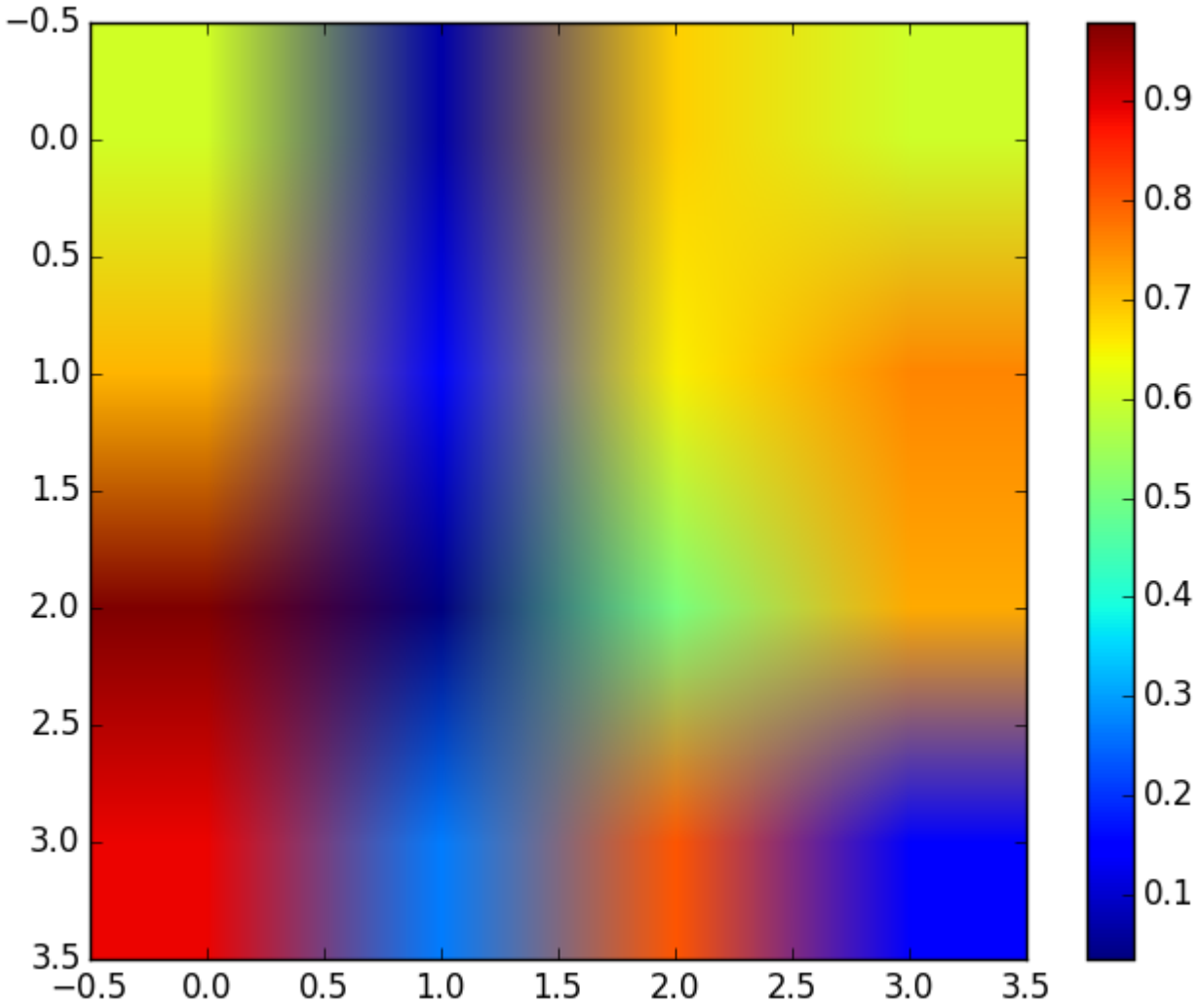
Entrambi gli esempi producono lo stesso grafico che viene mostrato di seguito.



Array bidimensionali (2D)

Mostra una matrice bidimensionale (2D) sugli assi.

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

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



Leggi Iniziare con matplotlib online: https://riptutorial.com/it/matplotlib/topic/881/iniziare-con-matplotlib

Capitolo 2: Animazioni e trama interattiva

introduzione

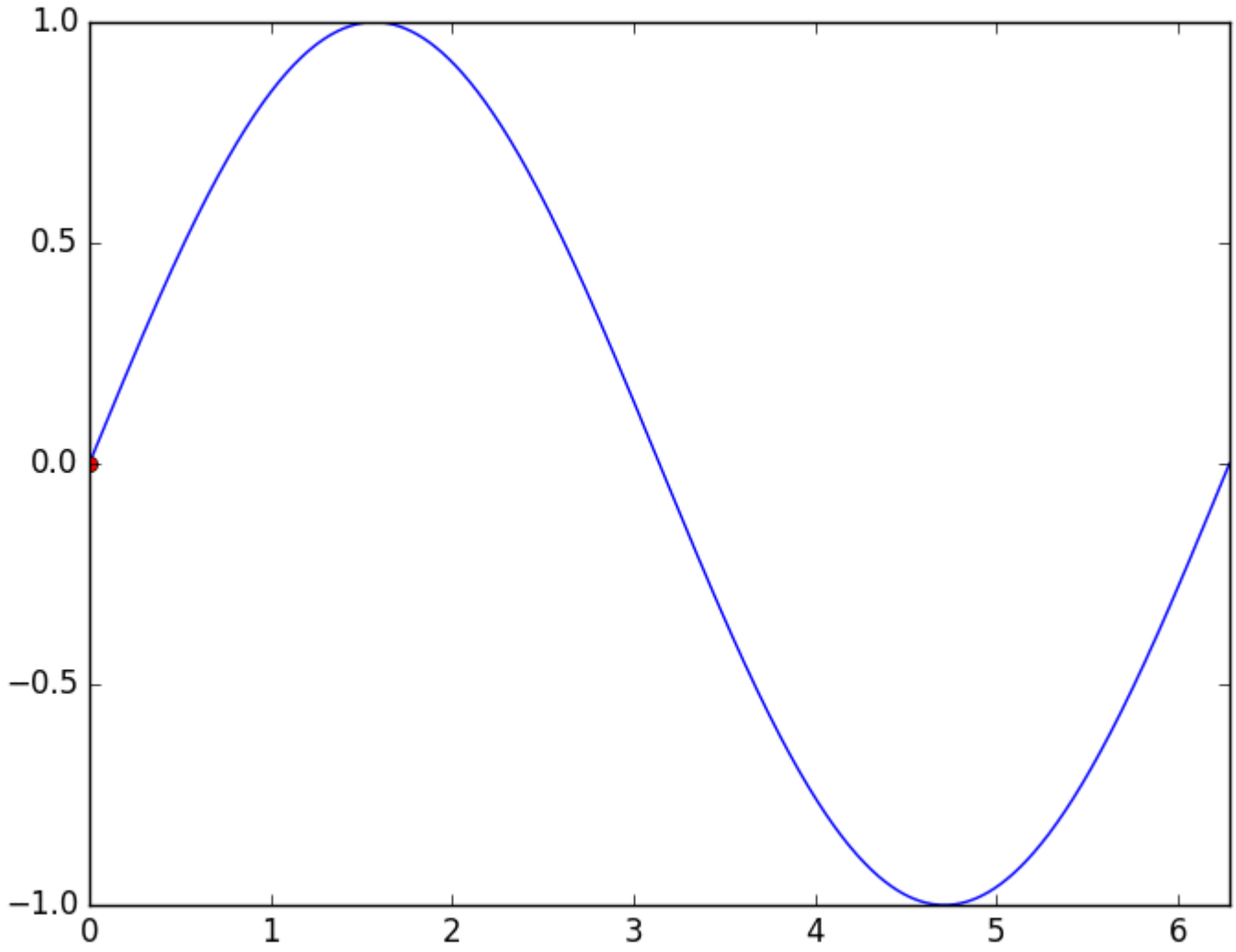
Con python matplotlib puoi creare correttamente grafici animati.

Examples

Animazione di base con FuncAnimation

Il pacchetto matplotlib.animation offre alcune classi per la creazione di animazioni. `FuncAnimation` crea animazioni chiamando ripetutamente una funzione. Qui usiamo una funzione `animate()` che cambia le coordinate di un punto sul grafico di una funzione seno.

```
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()
```



Salva l'animazione in gif

In questo esempio usiamo il `save` metodo per salvare un `Animation` oggetto usando 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)
```

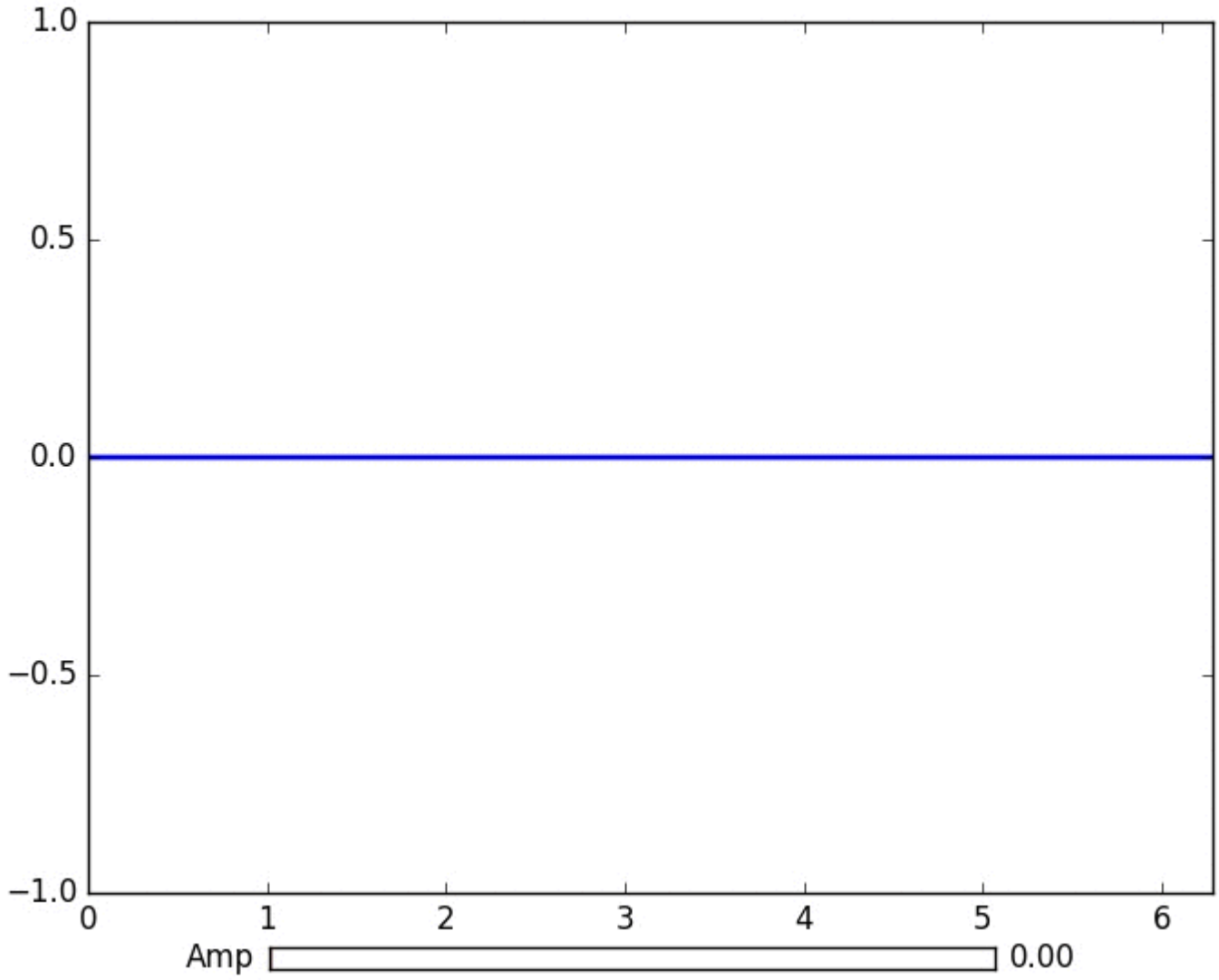
```
ax = plt.axis([0,TWOPI,-1,1])

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)
```

Controlli interattivi con matplotlib.widgets

Per interagire con i grafici Matplotlib offre i widget neutri della GUI. I widget richiedono un oggetto `matplotlib.axes.Axes` .

Ecco una demo del widget slider che aggiorna l'ampiezza di una curva sinusoidale. La funzione di aggiornamento viene attivata `on_changed()` del dispositivo di scorrimento.

```
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()
```



Altri widget disponibili:

- AxesWidget
- Pulsante
- CheckButtons
- Cursore
- EllipseSelector
- laccio
- LassoSelector
- LockDraw
- MultiCursor
- Tasti della radio
- RectangleSelector
- SpanSelector
- SubplotTool
- ToolHandles

Traccia i dati in tempo reale da pipe con matplotlib

Questo può essere utile quando si desidera visualizzare i dati in arrivo in tempo reale. Questi dati potrebbero, ad esempio, provenire da un microcontrollore che campiona continuamente un segnale analogico.

In questo esempio otterremo i nostri dati da una named pipe (nota anche come fifo). Per questo esempio, i dati nella pipe devono essere numeri separati da caratteri newline, ma è possibile adattarli a proprio piacimento.

Dati di esempio:

```
100
123.5
1589
```

Maggiori informazioni sulle pipe denominate

Utilizzeremo anche il deque del tipo di dati, dalle raccolte di librerie standard. Un oggetto deque funziona parecchio come un elenco. Ma con un oggetto di deque è abbastanza facile aggiungervi qualcosa mantenendo l'oggetto di deque a una lunghezza fissa. Questo ci permette di mantenere l'asse x a una lunghezza fissa invece di crescere sempre e schiacciare insieme il grafico. Ulteriori informazioni sugli oggetti deque

La scelta del back-end giusto è vitale per le prestazioni. Controlla quali backend funzionano sul tuo sistema operativo e scegli quello veloce. Per me solo qt4agg e il backend predefinito funzionavano, ma quello predefinito era troppo lento. Maggiori informazioni sui backend in matplotlib

Questo esempio è basato sull'esempio matplotlib di tracciare dati casuali .

Nessuno dei 'caratteri in questo codice deve essere rimosso.

```
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()
```

Se la trama inizia a essere ritardata dopo un po ', prova ad aggiungere altri dati datalist.append, in modo che più linee vengano lette ogni fotogramma. Oppure scegli un backend più veloce se puoi.

Questo ha funzionato con dati 150hz da una pipe sul mio 1.7ghz i3 4005u.

Leggi Animazioni e trama interattiva online:
https://riptutorial.com/it/matplotlib/topic/6983/animazioni-e-trama-interattiva

Capitolo 3: Chiusura di una finestra di figura

Sintassi

- plt.close () # chiude la figura attiva corrente
- plt.close (fig) # chiude la figura con la maniglia 'fig'
- plt.close (num) # chiude il numero di figura 'num'
- plt.close (nome) # chiude la figura con l'etichetta 'nome'
- plt.close ('all') # chiude tutte le cifre

Examples

Chiusura della cifra attiva corrente utilizzando pyplot

L'interfaccia di `matplotlib` su `matplotlib` potrebbe essere il modo più semplice per chiudere una figura.

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

Chiudere una figura specifica usando plt.close ()

Una cifra specifica può essere chiusa mantenendo la sua maniglia

```
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
```

Leggi Chiusura di una finestra di figura online:
https://riptutorial.com/it/matplotlib/topic/6628/chiusura-di-una-finestra-di-figura

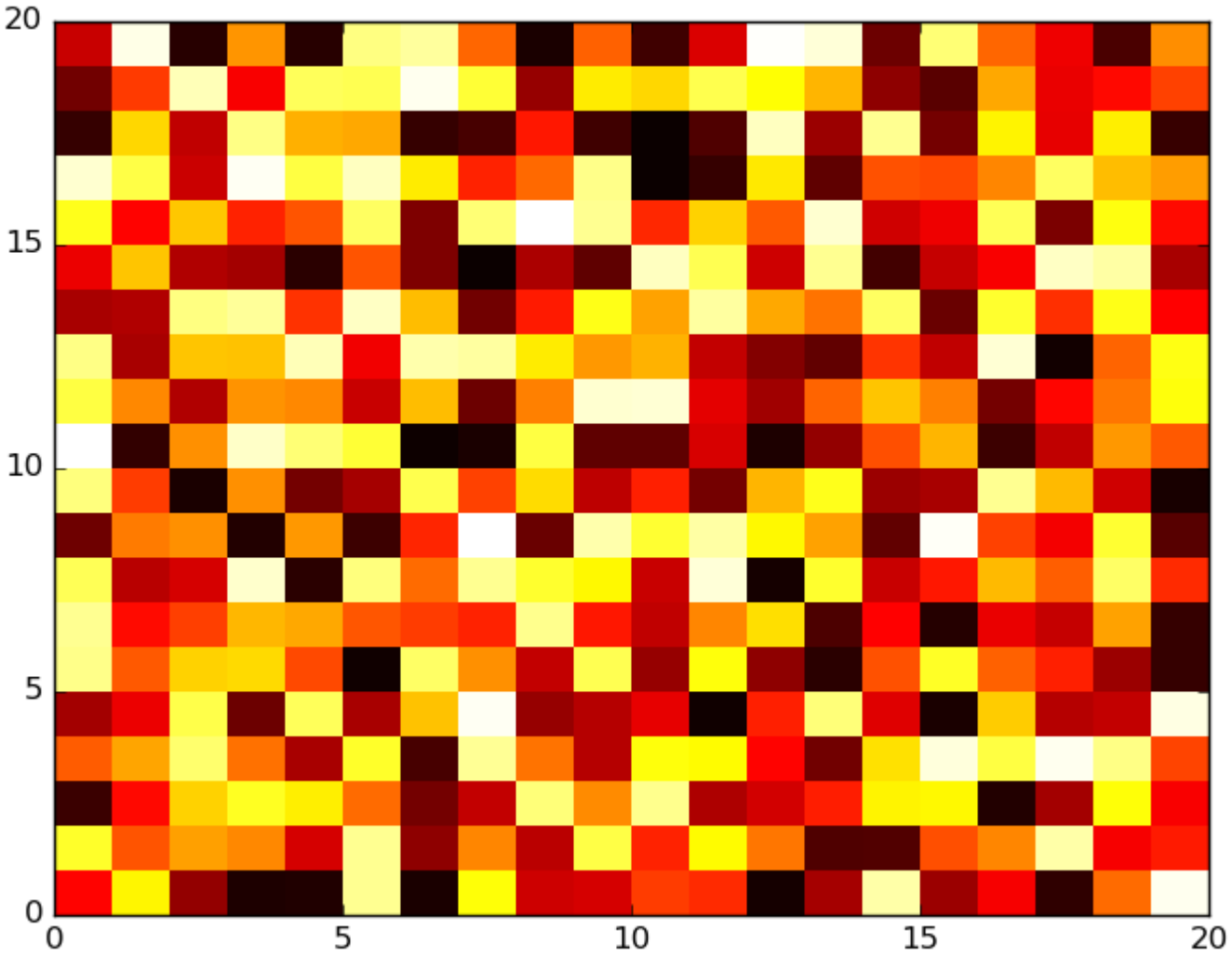
Capitolo 4: colormap

Examples

Utilizzo di base

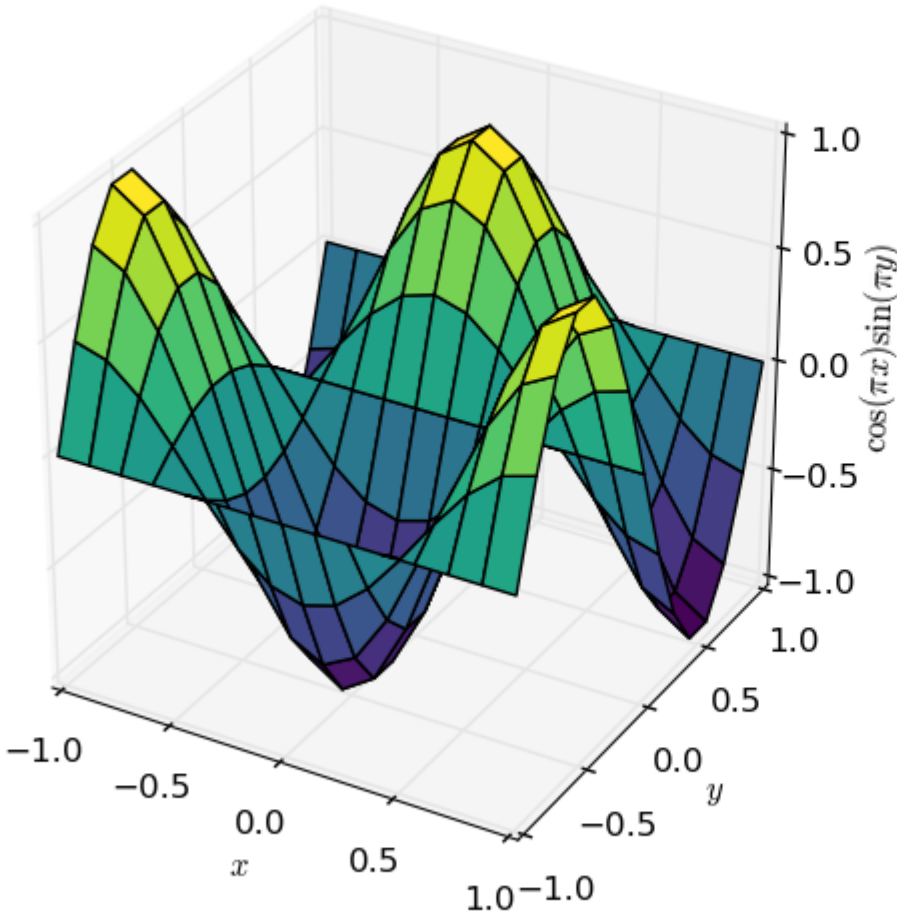
Usare le mappe dei colori built-in è semplice come passare il nome della mappa di colori richiesta (come indicato nel riferimento a colormap) alla funzione di `pcolormesh` (come `pcolormesh` o `contourf`) che si aspetta, solitamente sotto forma di un argomento della parola chiave `cmap` :

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

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



I colormap sono particolarmente utili per visualizzare i dati tridimensionali su grafici bidimensionali, ma una buona mappa colori può anche rendere molto più chiara una trama tridimensionale appropriata:

```
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()
```



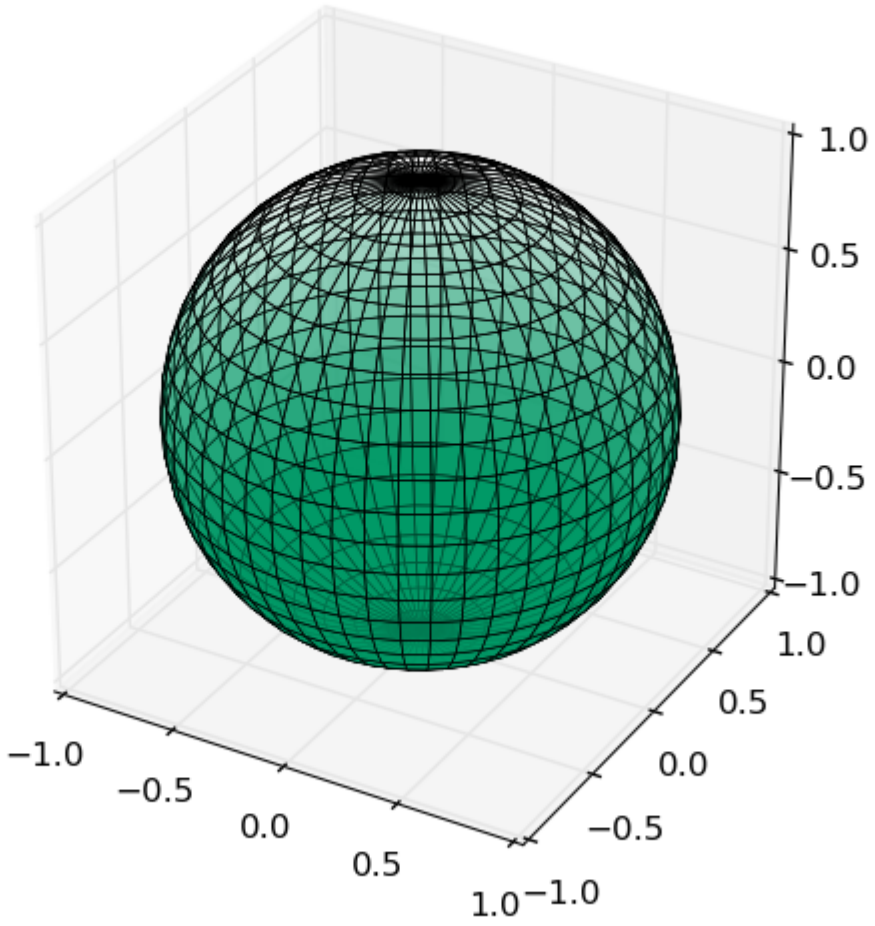
Utilizzo di mappe colori personalizzate

A parte le mappe di colori incorporate definite nel riferimento colormaps (e le loro mappe invertite, con `'_r'` aggiunto al loro nome), è anche possibile definire mappe di colori personalizzate. La chiave è il modulo `matplotlib.cm` .

L'esempio seguente definisce una mappa di colori molto semplice che utilizza `cm.register_cmap` , contenente un singolo colore, con l'opacità (valore alfa) del colore che interpola tra completamente opaco e completamente trasparente nell'intervallo di dati. Si noti che le linee importanti dal punto di vista della mappa di colori sono l'importazione di `cm` , la chiamata a `register_cmap` e il passaggio della mappa di colori 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()
```



In scenari più complicati, è possibile definire un elenco di valori R / G / B (/ A) in cui matplotlib interpola linearmente per determinare i colori utilizzati nei grafici corrispondenti.

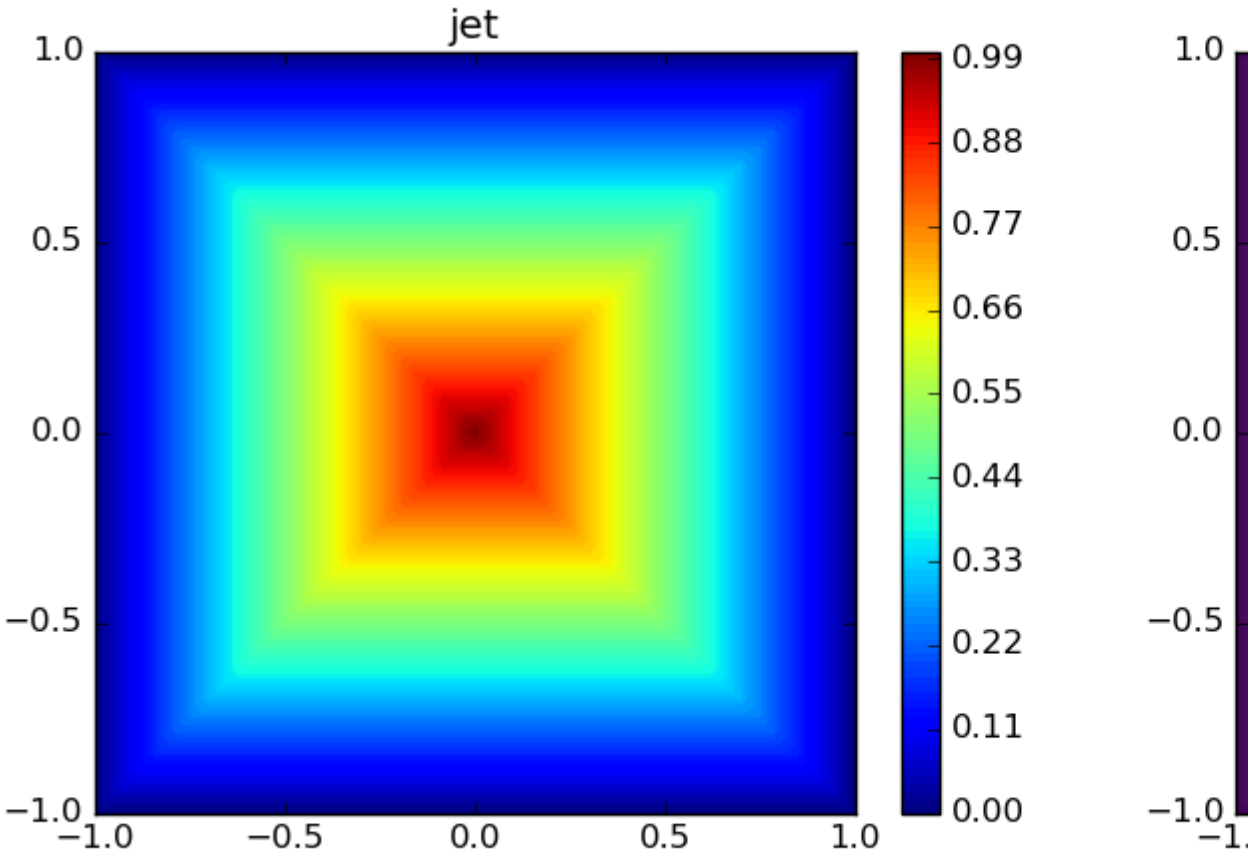
Mappe colore percettualmente uniformi

La colorazione predefinita predefinita di MATLAB (sostituita nella versione R2014b) chiamata `jet` è onnipresente per il suo alto contrasto e familiarità (ed era il default di matplotlib per ragioni di compatibilità). Nonostante la sua popolarità, le mappe di colori tradizionali hanno spesso delle lacune quando si tratta di rappresentare i dati con precisione. Il cambiamento percepito in questi colori non corrisponde ai cambiamenti nei dati; e una conversione della mappa dei colori in scala di grigi (ad esempio stampando una figura usando una stampante in bianco e nero) potrebbe causare la perdita di informazioni.

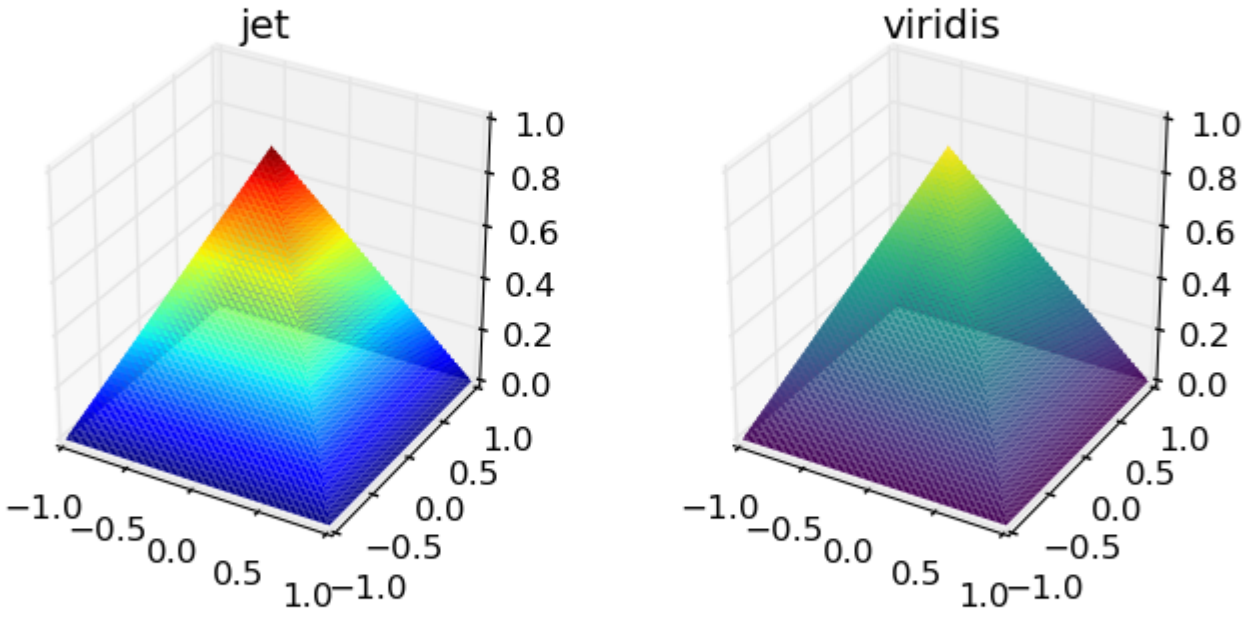
Sono state introdotte mappe di colori percettualmente uniformi per rendere la visualizzazione dei dati il più accurata e accessibile possibile. Matplotlib ha introdotto quattro nuovi colormap percettivamente uniformi nella versione 1.5, con uno di essi (denominato `viridis`) come predefinito dalla versione 2.0. Queste quattro colormap (`viridis` , `inferno` , `plasma` e `magma`) sono tutte ottimali dal punto di vista della percezione e dovrebbero essere utilizzate per la

visualizzazione dei dati per impostazione predefinita, a meno che non ci siano ottime ragioni per non farlo. Queste colormap introducono il più piccolo pregiudizio possibile (non creando caratteristiche dove non ce ne sono per cominciare), e sono adatte per un pubblico con una percezione del colore ridotta.

Come esempio per distorcere visivamente i dati, prendi in considerazione i seguenti due diagrammi di vista dall'alto di oggetti simili a piramide:



Quale delle due è una piramide corretta? La risposta è ovviamente che entrambi sono, ma questo è tutt'altro che ovvio dalla trama che usa la mappa `jet` colori del `jet` :



Questa caratteristica è al centro dell'uniformità percettiva.

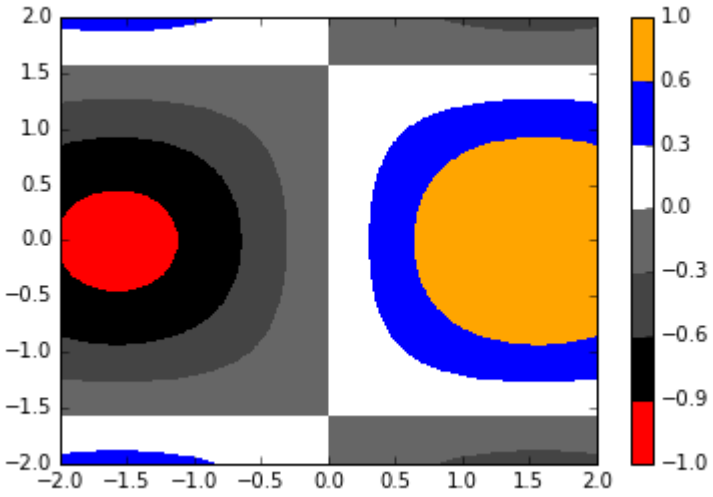
Mappa discreta personalizzata

Se hai intervalli predefiniti e desideri utilizzare colori specifici per tali intervalli, puoi dichiarare la mappa colori personalizzata. Per esempio:

```
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



Color *i* verrà utilizzato per valori compresi tra confine *iei* + *1.* I colori possono essere specificati con i nomi (`'red'` , `'green'`), i codici HTML (`'#ffaa44'` , `'#441188'`) o le tuple RGB (`(0.2, 0.9, 0.45)`).

Leggi colormap online: https://riptutorial.com/it/matplotlib/topic/3385/colormap

Capitolo 5: Figure e oggetti degli assi

Examples

Creare una figura

La figura contiene tutti gli elementi della trama. Il modo principale per creare una figura in `matplotlib` è usare `pyplot` .

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

È possibile fornire facoltativamente un numero, che è possibile utilizzare per accedere a una figura creata in precedenza. Se non viene fornito un numero, l'ID della figura creata per ultimo verrà incrementato e utilizzato; le cifre sono indicizzate a partire da 1, non 0.

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

Invece di un numero, le figure possono anche essere identificate da una stringa. Se si utilizza un back-end interattivo, verrà impostato anche il titolo della finestra.

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

Per scegliere l'uso della figura

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

Creare un asse

Esistono due modi principali per creare un asse in matplotlib: utilizzando pyplot o utilizzando l'API orientata agli oggetti.

Utilizzando pyplot:

```
import matplotlib.pyplot as plt

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

Utilizzando l'API orientata agli oggetti:

```
import matplotlib.pyplot as plt

fig = plt.figure()
```

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

La funzione di comodità `plt.subplots()` può essere usata per produrre una figura e una collezione di sottotrame in un unico comando:

```
import matplotlib.pyplot as plt

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

Leggi Figure e oggetti degli assi online: https://riptutorial.com/it/matplotlib/topic/2307/figure-e-oggetti-degli-assi

Capitolo 6: grafici a scatole

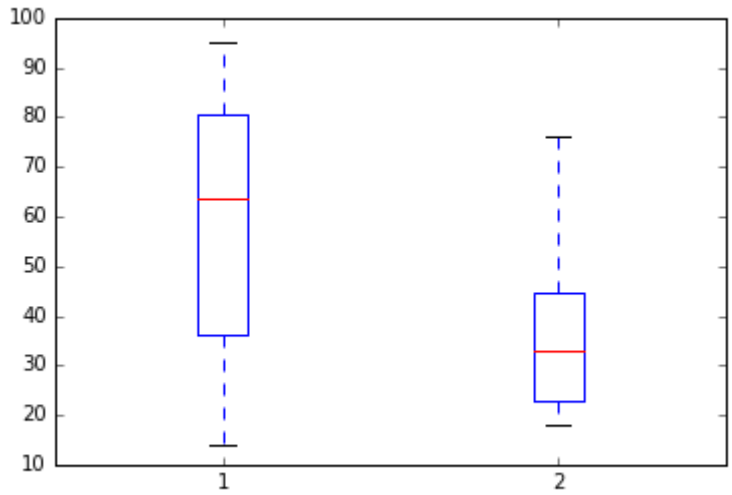
Examples

Boxplot di base

I boxplot sono diagrammi descrittivi che aiutano a confrontare la distribuzione di diverse serie di dati. Sono *descrittivi* perché mostrano misure (ad esempio la *mediana*) che non assumono una distribuzione di probabilità sottostante.

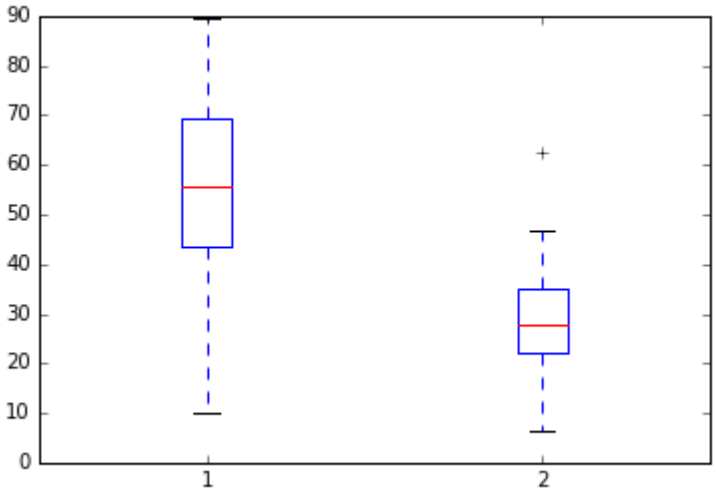
L'esempio più semplice di un boxplot in matplotlib può essere ottenuto semplicemente passando i dati come un elenco di liste:

```
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 )
```



Tuttavia, è prassi comune utilizzare `numpy` array `numpy` come parametri per i grafici, poiché sono spesso il risultato di calcoli precedenti. Questo può essere fatto come segue:

```
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 )
```



Leggi grafici a scatole online: https://riptutorial.com/it/matplotlib/topic/6086/grafici-a-scatole

Capitolo 7: grafici a scatole

Examples

Funzione Boxplot

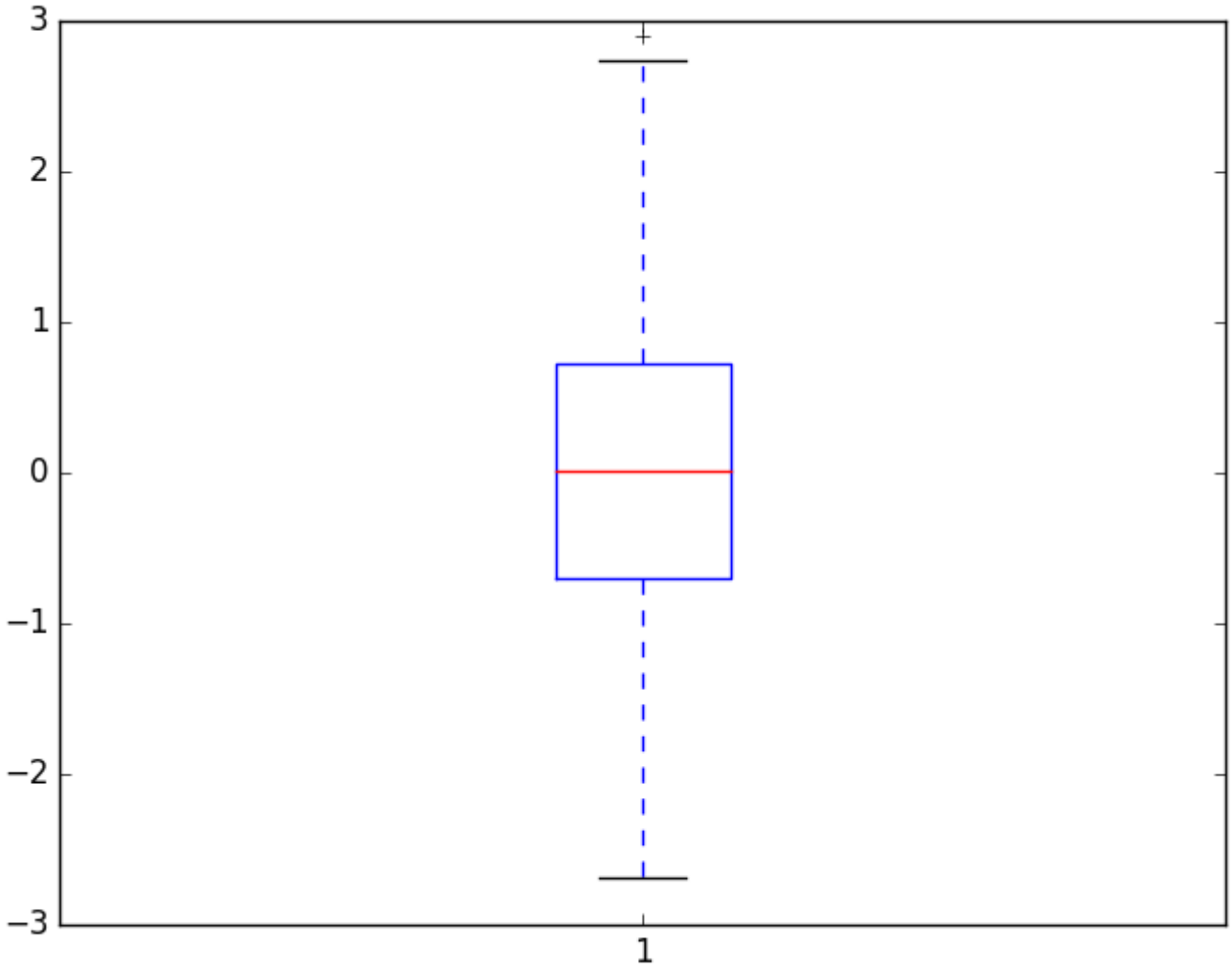
Matplotlib ha la propria implementazione di boxplot . Gli aspetti rilevanti di questa funzione sono che, per impostazione predefinita, il grafico a scatole mostra la mediana (percentile 50%) con una linea rossa. La scatola rappresenta Q1 e Q3 (percentili 25 e 75), e i baffi danno un'idea del range dei dati (possibilmente a Q1 - 1.5 *IQR; Q3 + 1.5* IQR, essendo IQR l'intervallo interquartile, ma questo manca di conferma). Si noti inoltre che i campioni oltre questo intervallo sono indicati come indicatori (questi sono chiamati volantini).

> **NOTA:** non tutte le implementazioni di *boxplot* seguono le stesse regole. Forse il diagramma del boxplot più comune usa i baffi per rappresentare il minimo e il massimo (rendendo inesistenti i volantini). Si noti inoltre che questa trama è talvolta chiamato *plot box-and-whisker* e *schema box-and-whisker.*

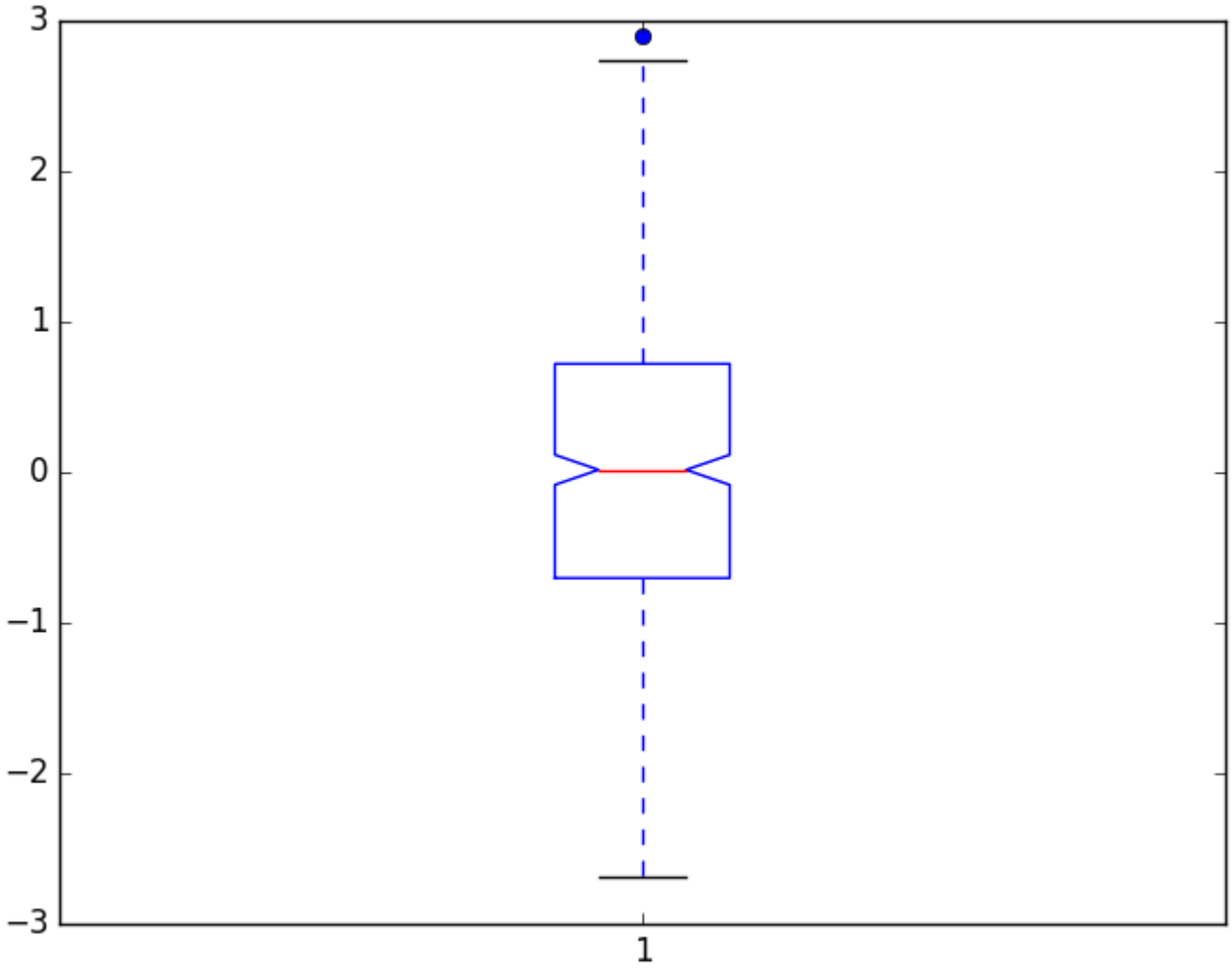
La seguente ricetta mostra alcune delle cose che puoi fare con l'attuale implementazione di matplotlib di boxplot:

```
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()
```

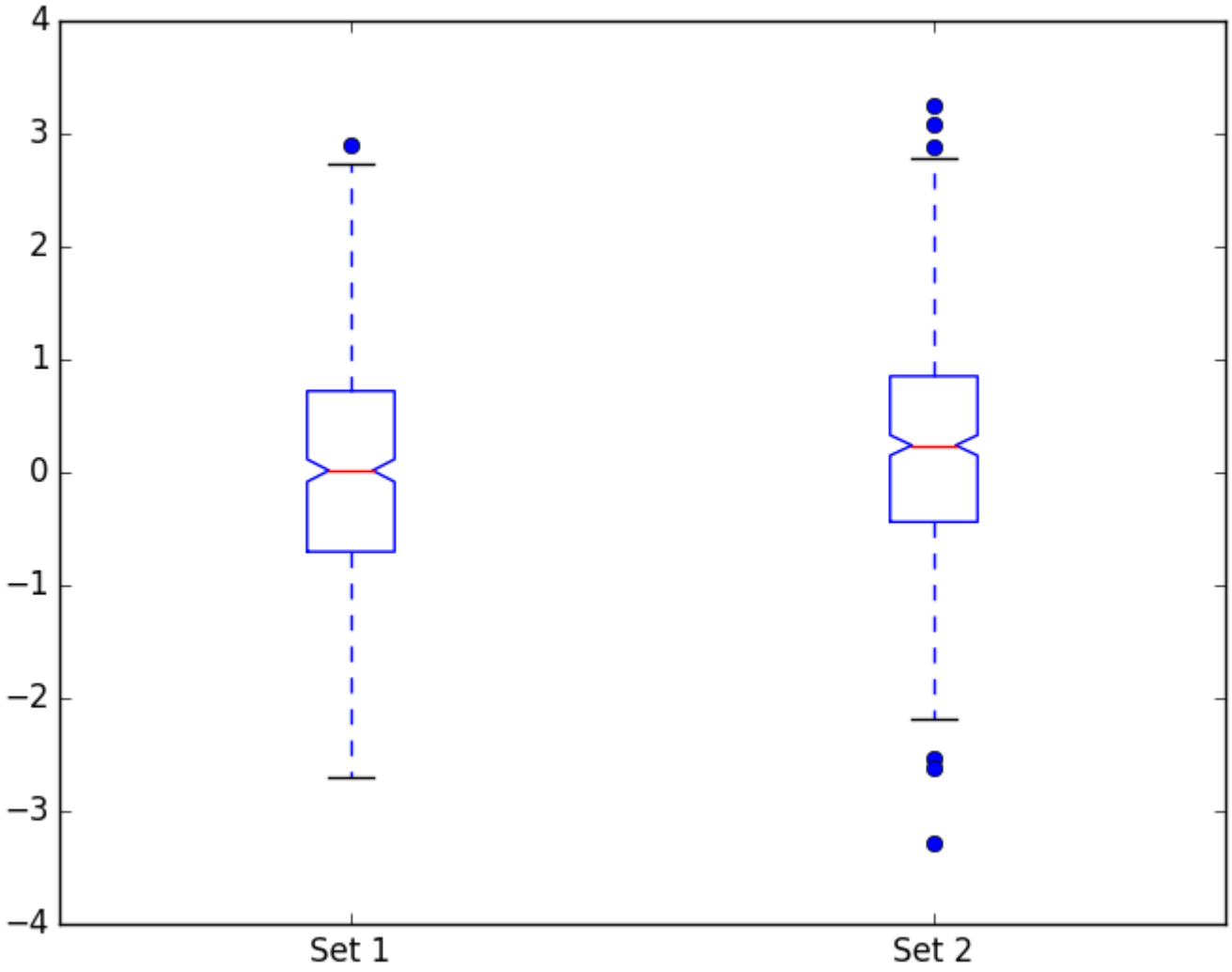
Questo risulta nei seguenti grafici:



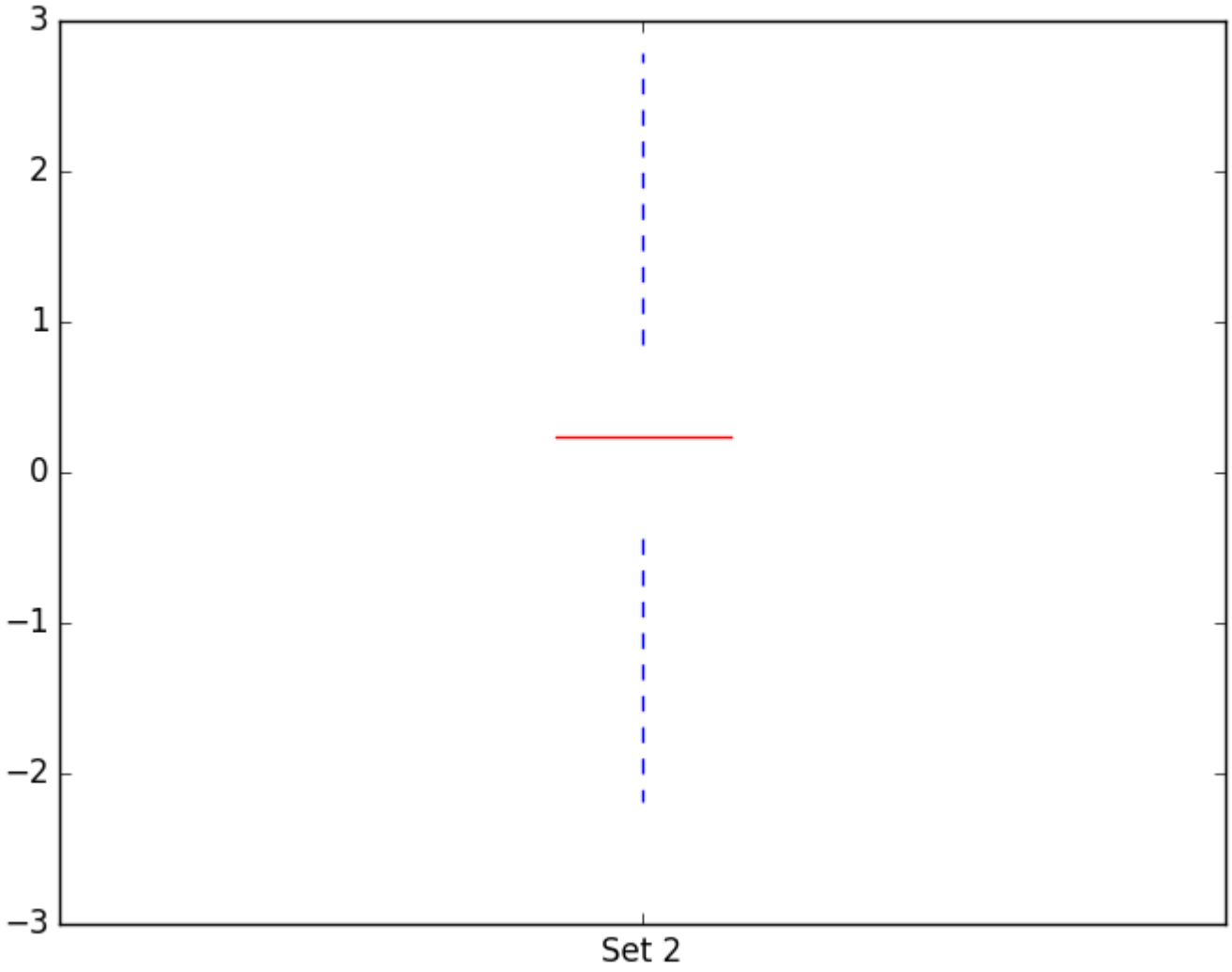
1. *Boxplot matplotlib predefinito*



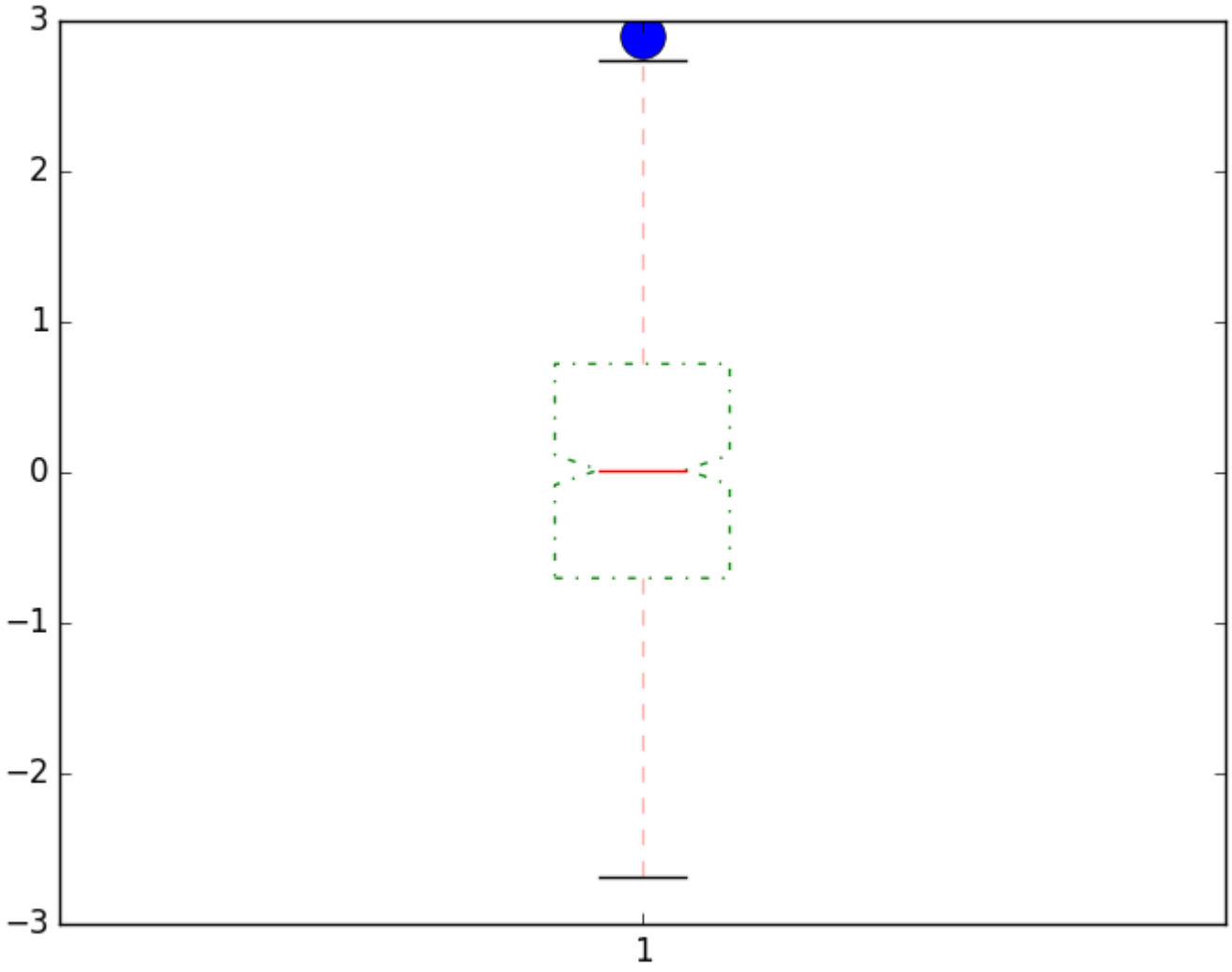
2. *Modifica di alcune funzionalità del boxplot utilizzando gli argomenti della funzione*



3. *Boxplot multiplo nella stessa finestra del grafico*



4. *Hidding alcune caratteristiche del boxplot*



5. *Personalizzazione avanzata di un boxplot utilizzando oggetti di scena*

Se hai intenzione di fare alcune personalizzazioni avanzate del tuo boxplot dovresti sapere che i dizionari dei puntelli che hai costruito (per esempio):

```
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()
```

... si riferiscono principalmente (se non tutti) a oggetti Line2D . Ciò significa che solo gli argomenti disponibili in quella classe sono modificabili. Noterai l'esistenza di parole chiave come `whiskerprops` , `boxprops` , `flierprops` e `capprops` . Questi sono gli elementi necessari per fornire un dizionario di oggetti di scena per personalizzarlo ulteriormente.

> NOTA: un'ulteriore personalizzazione del boxplot che utilizza questa implementazione potrebbe rivelarsi difficile. In alcuni casi l'uso di altri elementi matplotlib come le patch

per costruire il proprio boxplot può essere vantaggioso (ad esempio, modifiche considerevoli all'elemento box).

Leggi grafici a scatole online: https://riptutorial.com/it/matplotlib/topic/6368/grafici-a-scatole

Capitolo 8: Integrazione con TeX / LaTeX

Osservazioni

- Il supporto LaTeX di Matplotlib richiede un'installazione LaTeX funzionante, dvipng (che può essere incluso nell'installazione di LaTeX) e Ghostscript (si consiglia GPL Ghostscript 8.60 o successivo).
- Il supporto pgf di Matplotlib richiede una recente installazione di LaTeX che include i pacchetti TikZ / PGF (come TeXLive), preferibilmente con XeLaTeX o LuaLaTeX installati.

Examples

Inserimento di formule TeX nei grafici

Le formule TeX possono essere inserite nel grafico usando la funzione `rc`

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

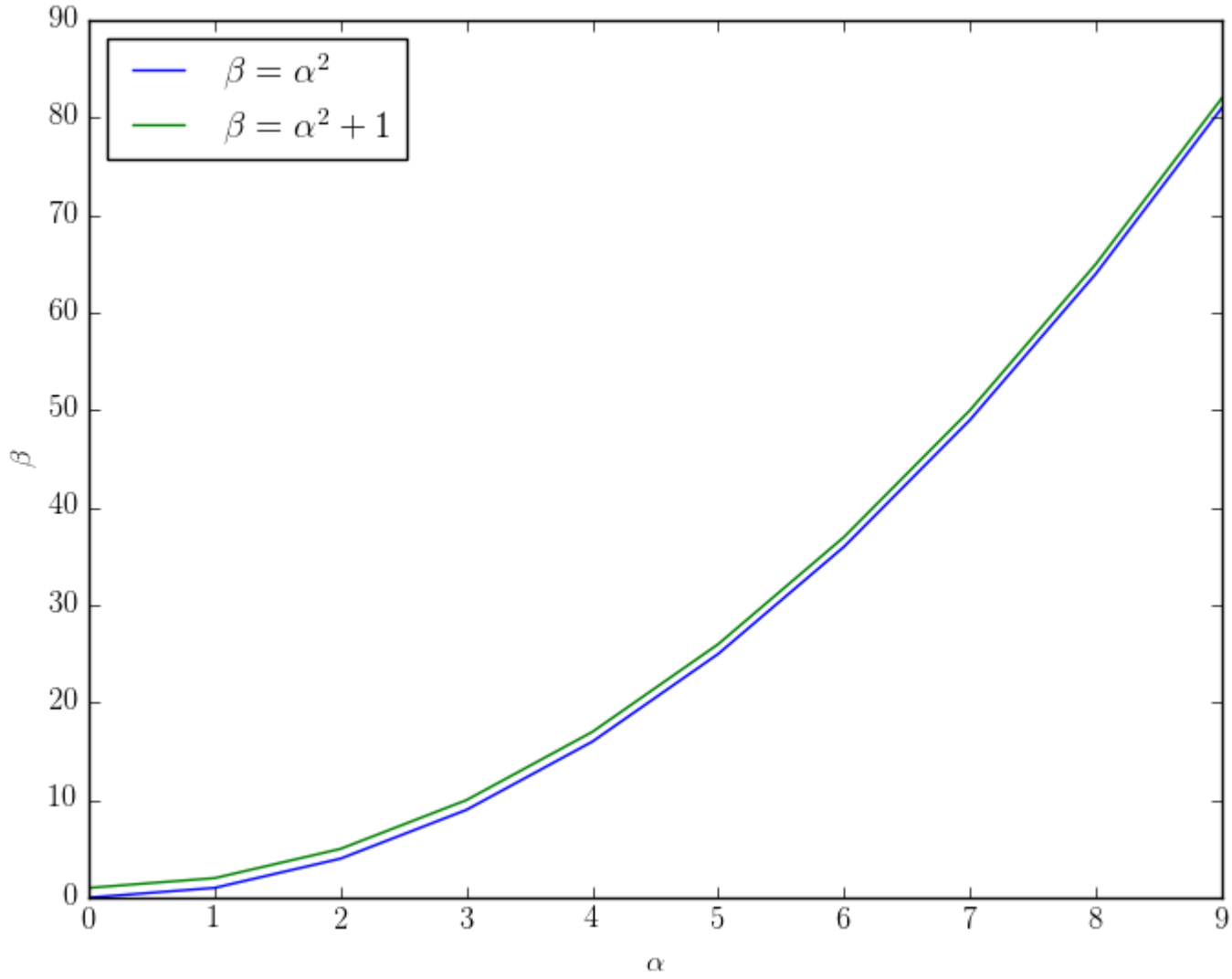
o accedendo a `rcParams` :

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

TeX usa il backslash `\` per comandi e simboli, che possono entrare in conflitto con caratteri speciali nelle stringhe Python. Per utilizzare backslash letterali in una stringa Python, devono essere sottoposti a escape o incorporati in una stringa non elaborata:

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

La seguente trama



può essere prodotto dal codice

```
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()
```

Le equazioni visualizzate (come `$$...$$` o `\begin{equation}...\end{equation}`) non sono supportate. Tuttavia, lo stile matematico visualizzato è possibile con `\displaystyle` .

Per caricare i pacchetti in lattice, utilizzare l'argomento `tex.latex.preamble` :

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

Notare, tuttavia, l'avvertenza nel file matplotlibrc di esempio :

```
#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
```

Salvataggio ed esportazione di grafici che utilizzano TeX

Per includere i grafici creati con matplotlib nei documenti TeX, dovrebbero essere salvati come file `pdf` o `eps` . In questo modo, qualsiasi testo nel grafico (incluse le formule TeX) viene visualizzato come testo nel documento finale.

```
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
```

I grafici in matplotlib possono essere esportati in codice TeX usando il pacchetto macro `pgf` per visualizzare la grafica.

```
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')
```

Utilizzare il comando `rc` per modificare il motore TeX utilizzato

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

Per includere la figura `.pgf` , scrivi nel documento LaTeX

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

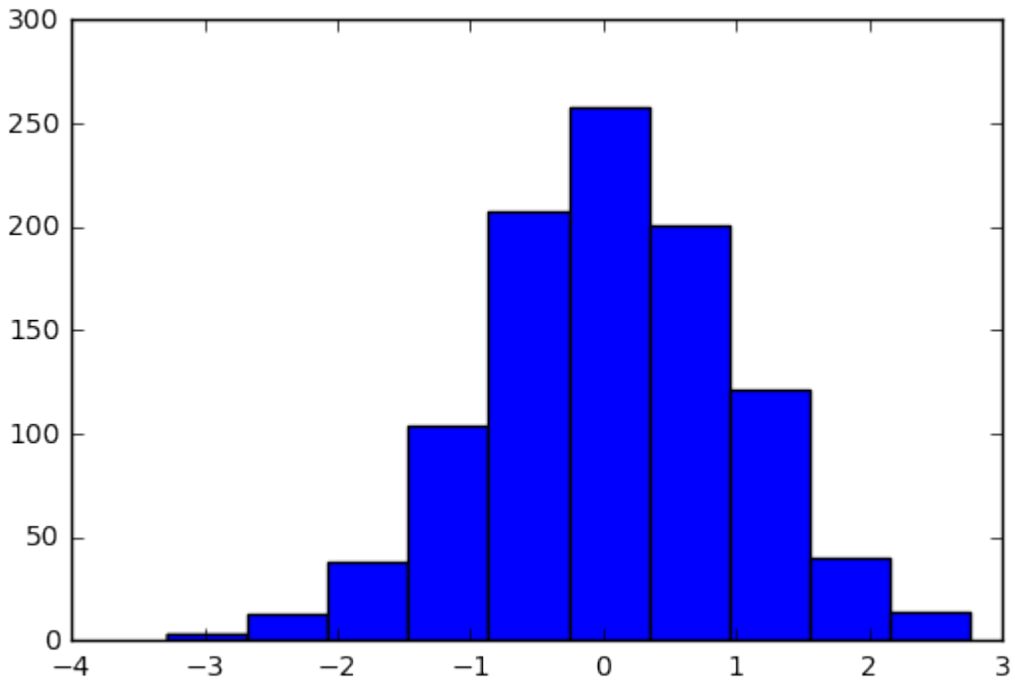
Leggi Integrazione con TeX / LaTeX online:
https://riptutorial.com/it/matplotlib/topic/2962/integrazione-con-tex---latex

Capitolo 9: Istogramma

Examples

Istogramma semplice

```
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()
```



Leggi Istogramma online: https://riptutorial.com/it/matplotlib/topic/7329/istogramma

Capitolo 10: Legends

Examples

Leggenda semplice

Supponiamo che tu abbia più linee nello stesso grafico, ognuna di un colore diverso, e desideri creare una legenda per dire cosa rappresenta ciascuna linea. È possibile farlo passando un'etichetta a ciascuna delle linee quando si chiama `plot()` , ad esempio, la seguente riga sarà etichettata come *"My Line 1"* .

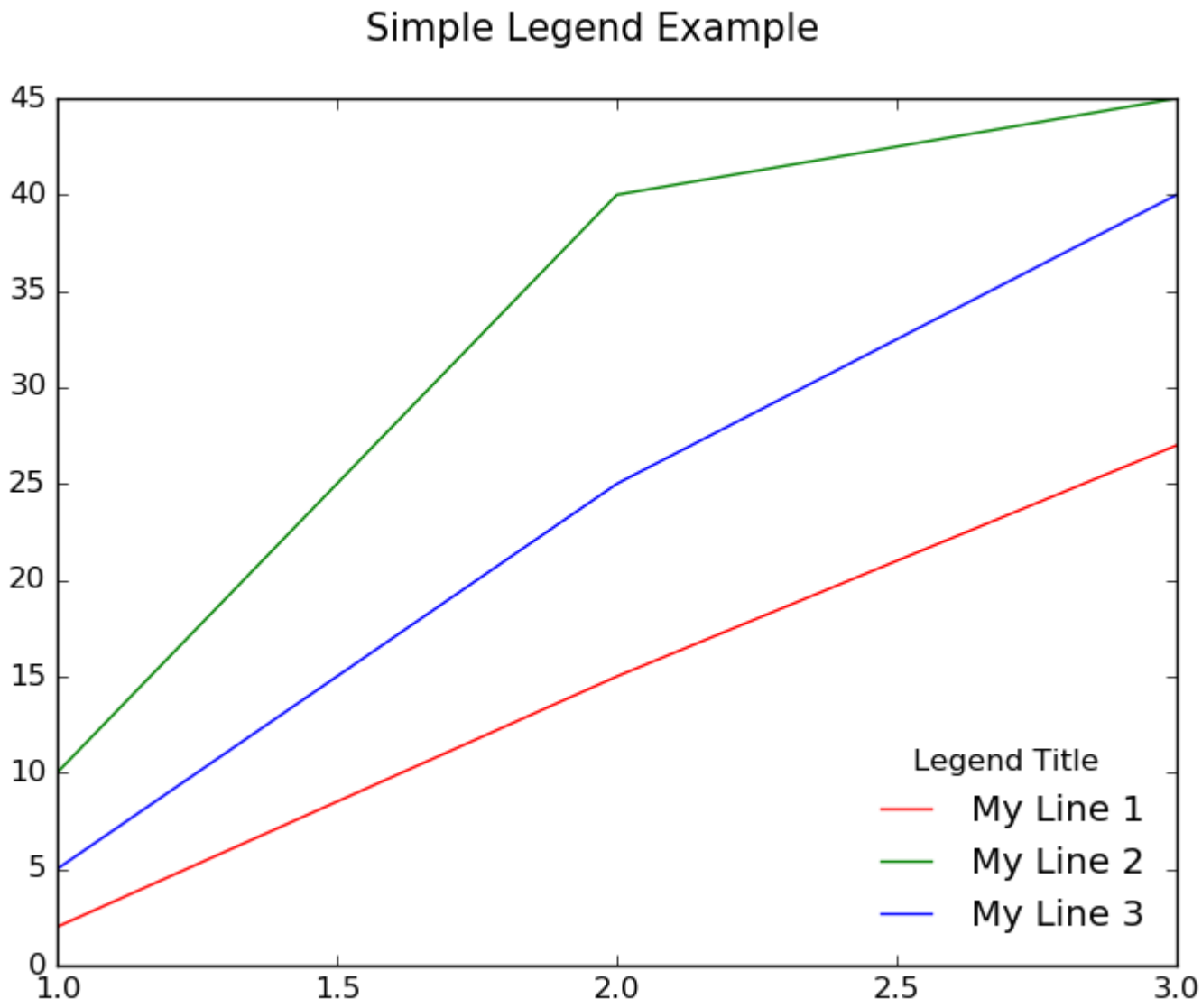
```
ax.plot(x, y1, color="red", label="My Line 1")
```

Questo specifica il testo che apparirà nella legenda per quella linea. Ora per rendere visibile la legenda effettiva, possiamo chiamare `ax.legend()`

Per impostazione predefinita verrà creata una legenda all'interno di una casella nell'angolo in alto a destra del grafico. È possibile passare argomenti a `legend()` per personalizzarlo. Ad esempio possiamo posizionarlo nell'angolo in basso a destra, senza una cornice che lo circonda, e creando un titolo per la legenda chiamando il seguente:

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

Di seguito è riportato un esempio:

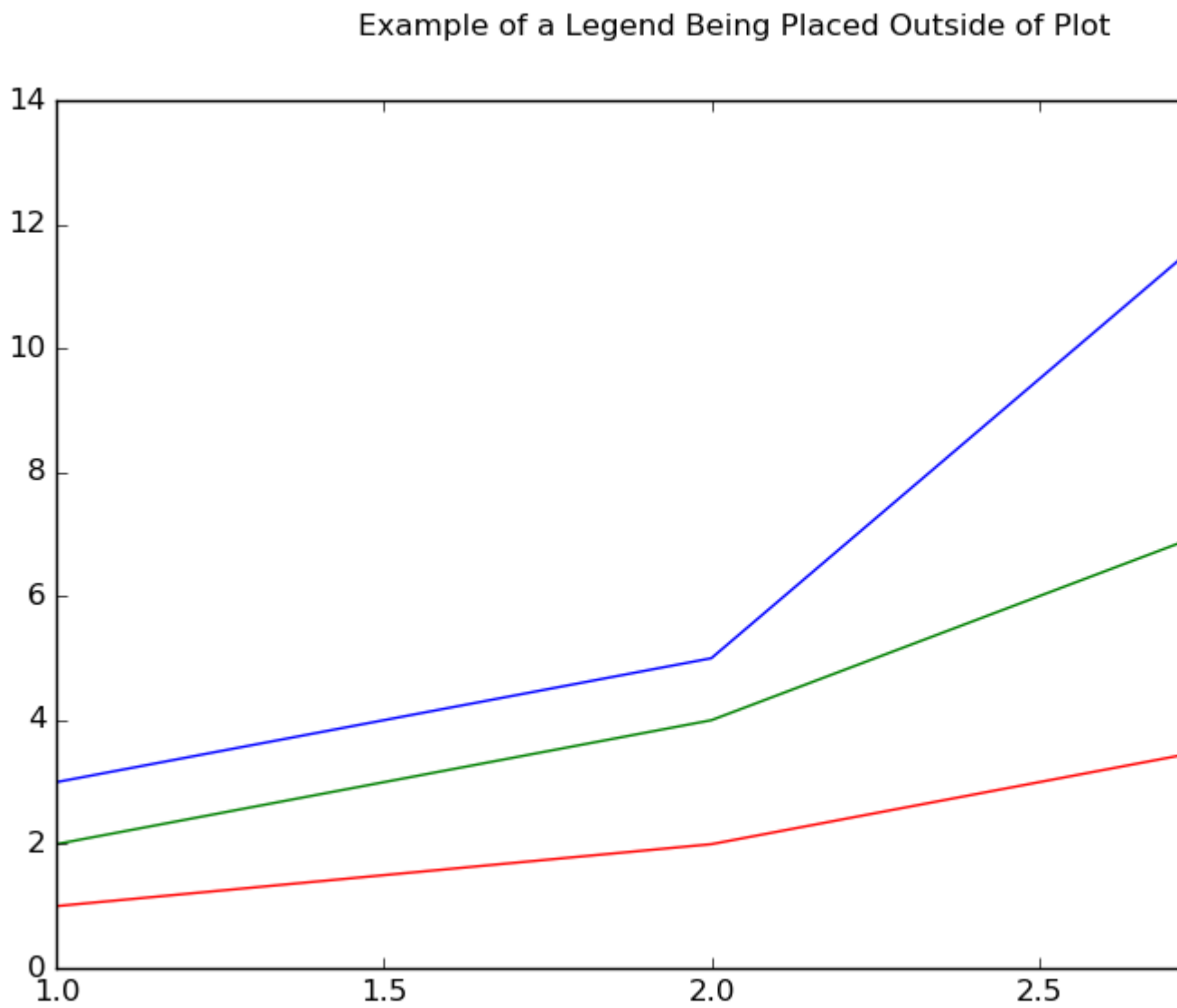



```
import matplotlib.pyplot as plt

# 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()
```

Legenda posizionata all'esterno del grafico

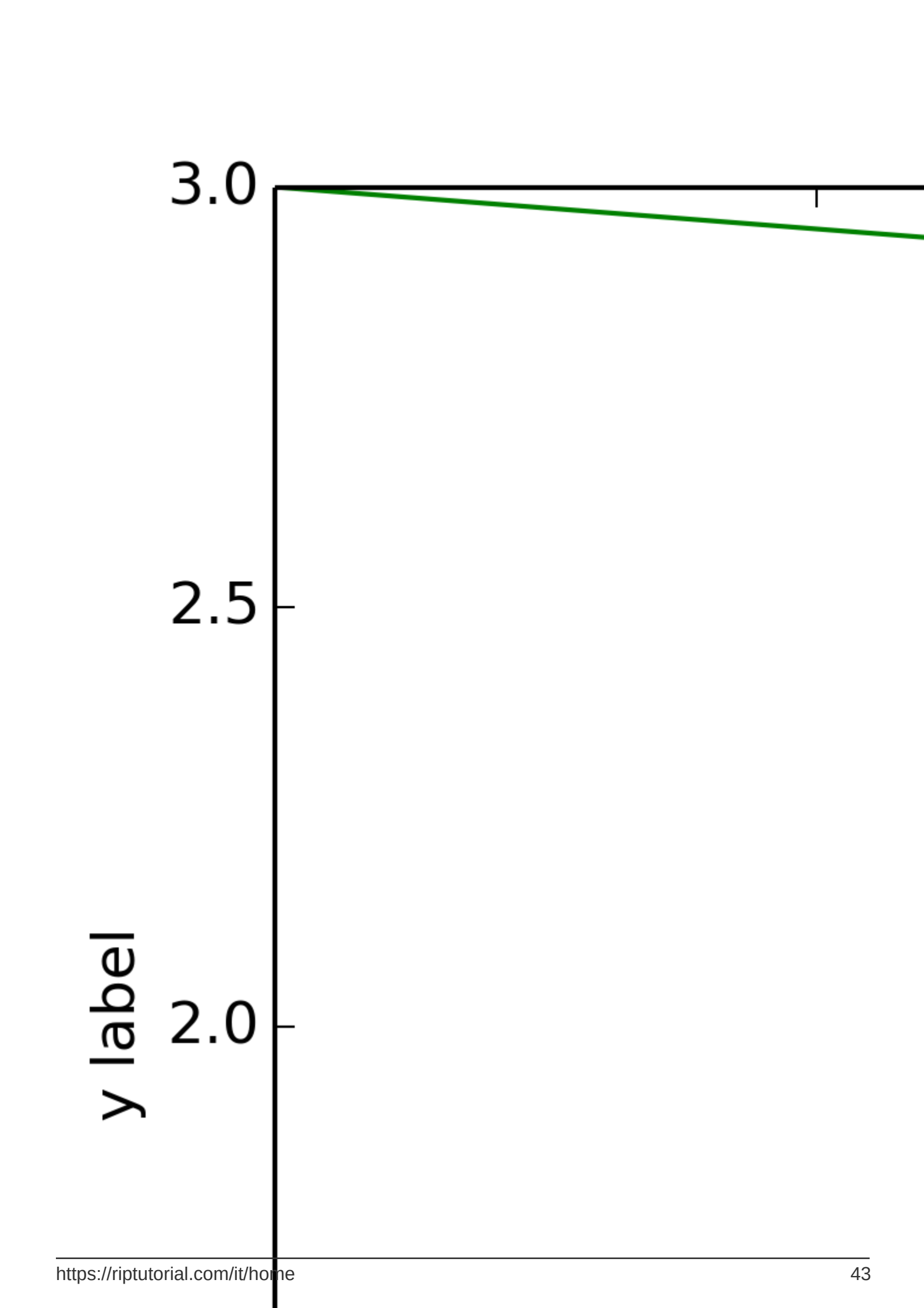
A volte è necessario o desiderabile posizionare la legenda al di fuori della trama. Il seguente codice mostra come farlo.




```
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()
```

Un altro modo per posizionare la legenda al di fuori della trama è utilizzare `bbox_to_anchor` + `bbox_extra_artists` + `bbox_inches='tight'` , come mostrato nell'esempio seguente:


3.0
2.5
2.0
y label

invece di creare una legenda a livello di *assi* (che creerà una legenda separata per ogni sottotrama). Ciò si ottiene chiamando `fig.legend()` come si può vedere nel codice per il codice seguente.

```
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()
```

Qualcosa da notare sull'esempio sopra è il seguente:

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

Quando viene chiamato `plot()` , restituisce una lista di oggetti **line2D** . In questo caso restituisce semplicemente una lista con un oggetto *line2D* singolo, che viene estratto con l'indicizzazione `[0]` e archiviato in `l1` .

Un elenco di tutti gli oggetti *line2D* che ci interessa includere nella legenda deve essere passato come primo argomento di `fig.legend()` . È inoltre necessario il secondo argomento di `fig.legend()` . Dovrebbe essere un elenco di stringhe da utilizzare come etichette per ogni riga nella legenda.

Gli altri argomenti passati a `fig.legend()` sono puramente opzionali e aiutano solo a perfezionare l'estetica della leggenda.

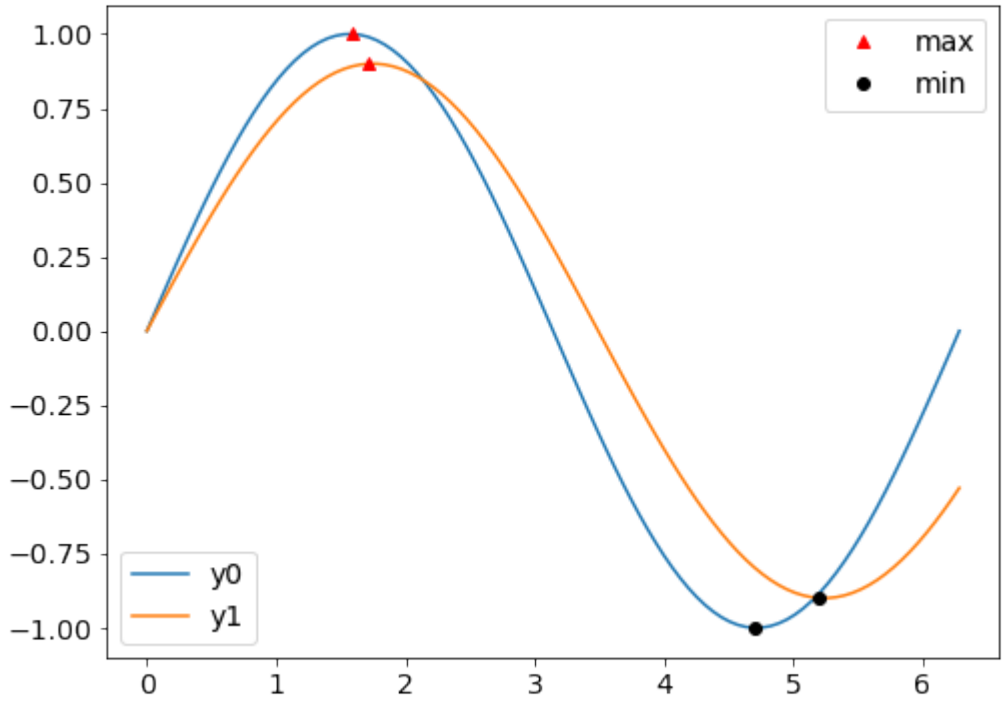
Legende multiple sullo stesso asse

Se si chiama `plt.legend()` o `ax.legend()` più di una volta, la prima legenda viene rimossa e ne viene disegnata una nuova. Secondo la documentazione ufficiale :

> Questo è stato fatto in modo che sia possibile chiamare ripetutamente legend () per aggiornare la legenda con gli ultimi handle degli Axe

Non temere, però: è ancora abbastanza semplice aggiungere una seconda legenda (o terza o quarta ...) ad un asse. Nell'esempio qui, tracciamo due linee, quindi tracciamo i marcatori sui rispettivi massimi e minimi. Una legenda è per le linee e l'altra è per i marcatori.

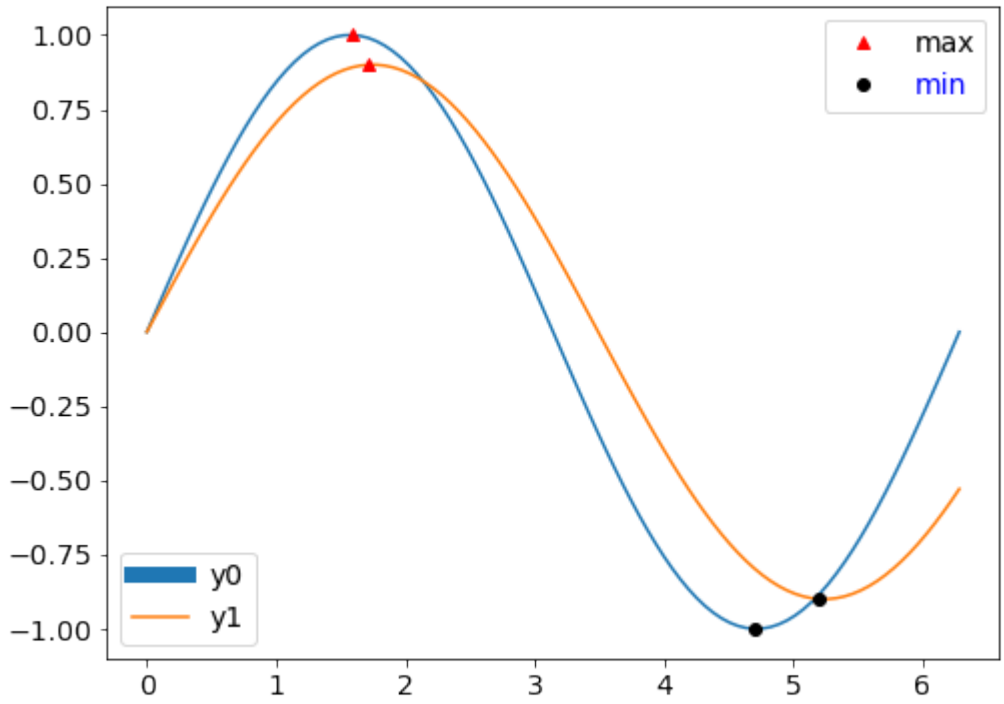
```
import matplotlib.pyplot as plt
import numpy as np

# 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 chiave è assicurarsi di avere riferimenti agli oggetti della legenda. Il primo che `leg1` (`leg1`) viene rimosso dalla figura quando aggiungi il secondo, ma l'oggetto `leg1` esiste ancora e può essere aggiunto nuovamente con `ax.add_artist` .

La cosa davvero fantastica è che puoi ancora manipolare *entrambe le* leggende. Ad esempio, aggiungi quanto segue alla fine del codice sopra:

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



Infine, vale la pena ricordare che nell'esempio solo le linee hanno dato etichette quando sono

state tracciate, il che significa che `ax.legend()` aggiunge solo quelle linee alla `leg1` . La legenda per i marcatori (`leg2`) richiedeva quindi le linee e le etichette come argomenti quando veniva istanziata. Potremmo avere, in alternativa, dato etichette ai marcatori anche quando sono stati tracciati. Ma poi *entrambe le* chiamate a `ax.legend` avrebbero richiesto alcuni argomenti extra in modo che ogni leggenda contenesse solo gli elementi che volevamo.

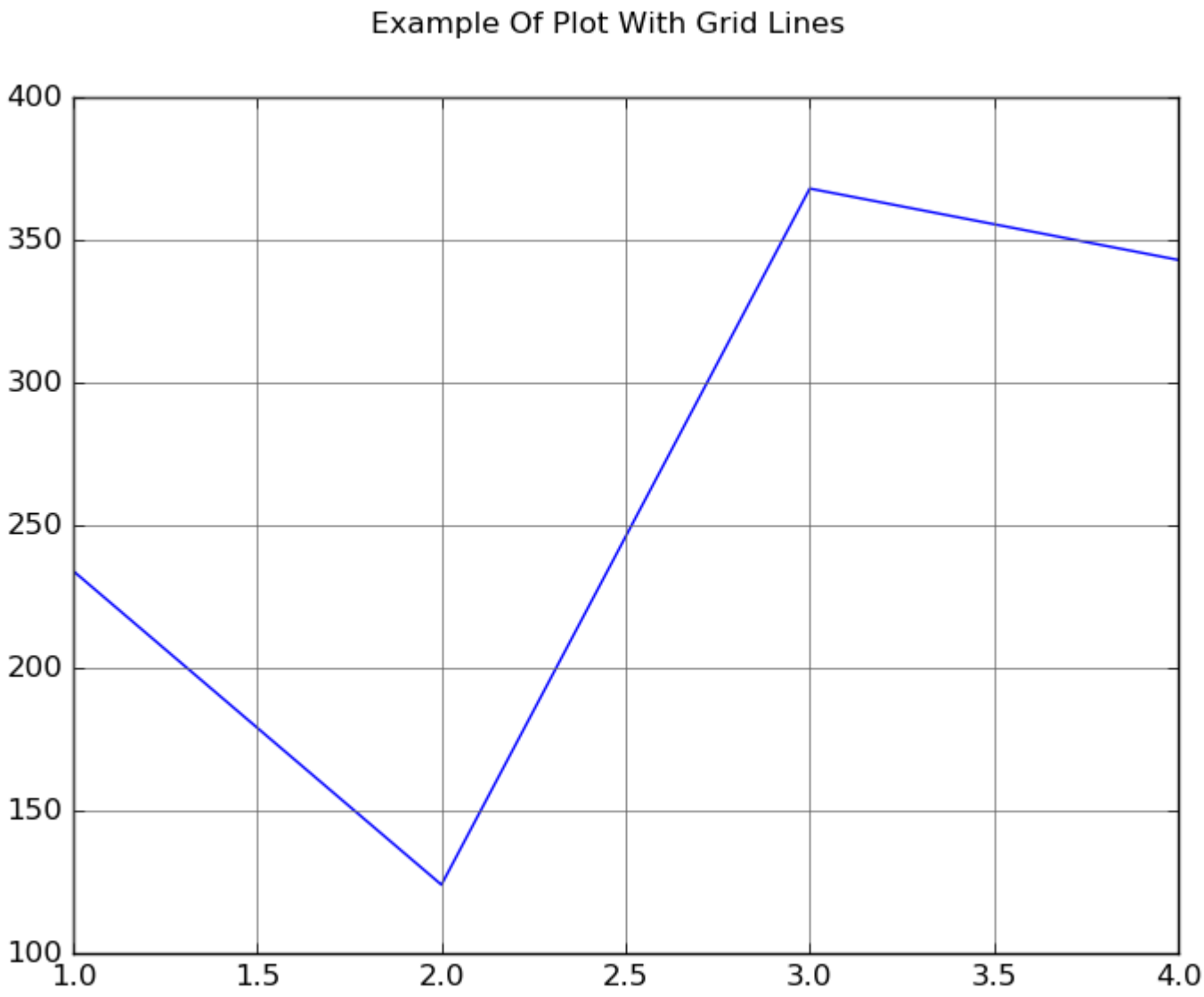
Leggi Legends online: https://riptutorial.com/it/matplotlib/topic/2840/legends

Capitolo 11: Linee di griglia e segni di graduazione

Examples

Plot With Gridlines

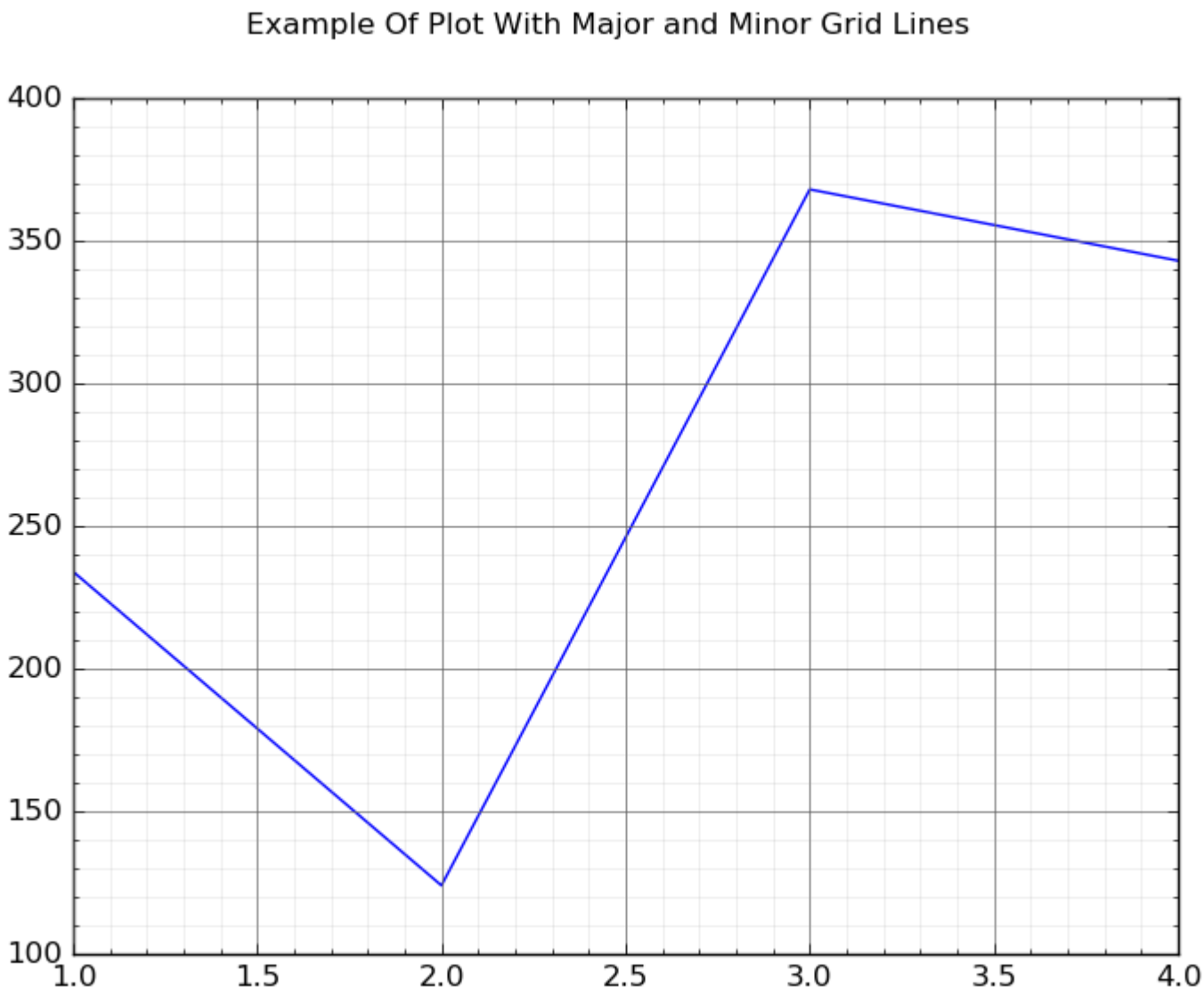
Trama con linee griglia




```
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()
```

Tracciare linee di griglia maggiori e minori




```
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()
```

Leggi Linee di griglia e segni di graduazione online:
https://riptutorial.com/it/matplotlib/topic/4029/linee-di-griglia-e-segni-di-graduazione

Capitolo 12: LogLog Graphing

introduzione

Il log Log è una possibilità per illustrare una funzione esponenziale in modo lineare.

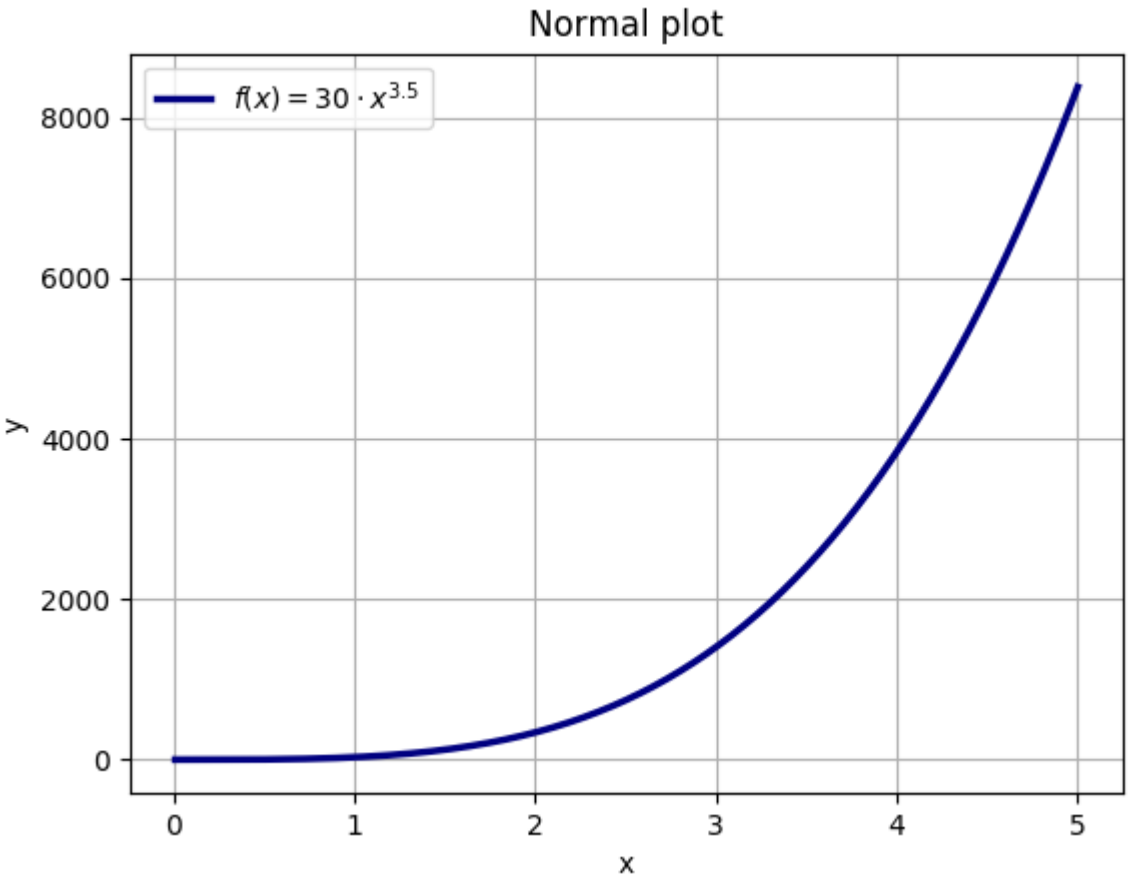
Examples

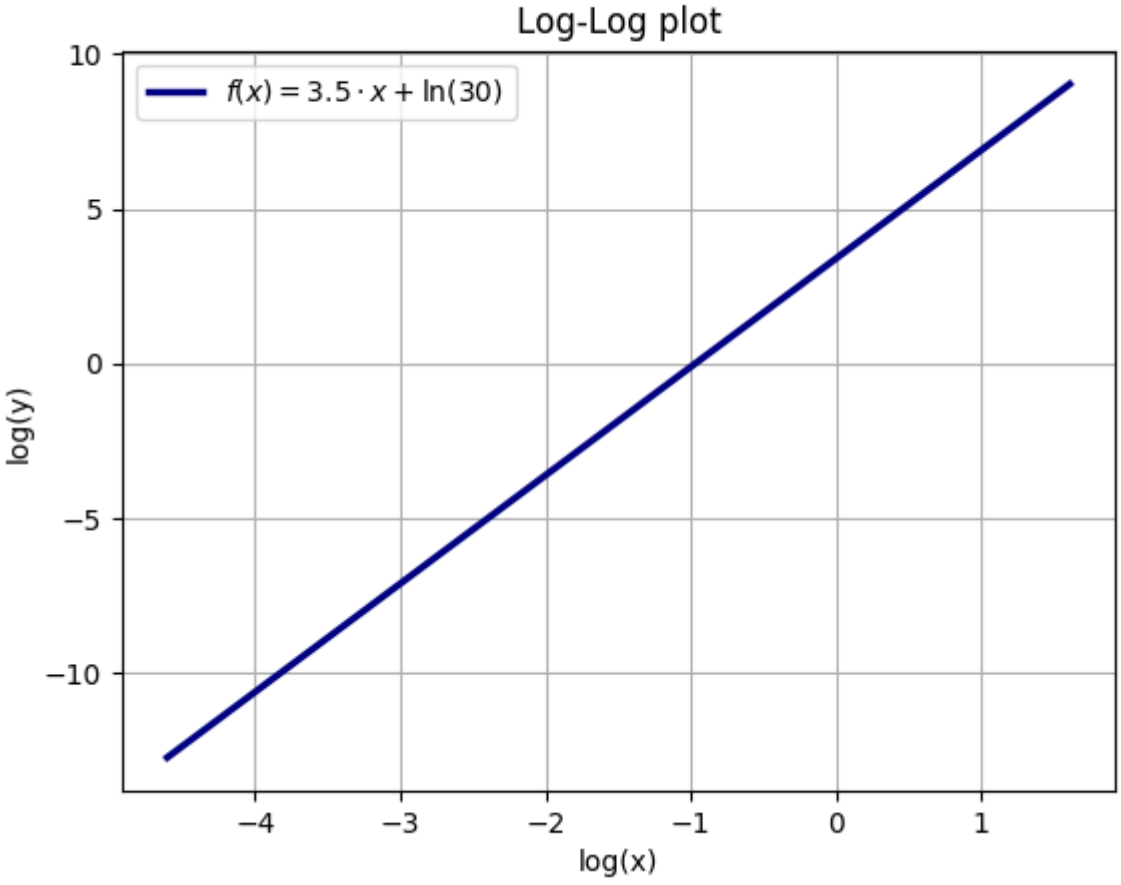
LogLog grafico

Sia y (x) = A * x ^ a, ad esempio A = 30 e a = 3.5. Prendendo il logaritmo naturale (ln) di entrambi i lati cede (usando le regole comuni per i logaritmi): ln (y) = ln (A * x ^ a) = ln (A) + ln (x ^ a) = ln (A) + a * ln (x). Pertanto, una trama con assi logaritmici per x e y sarà una curva lineare. La pendenza di questa curva è l'esponente a di y (x), mentre l'y-intercetta y (0) è il logaritmo naturale di A, ln (A) = ln (30) = 3.401.

L'esempio seguente illustra la relazione tra una funzione esponenziale e il grafico del loglog lineare (la funzione è y = A * x ^ a con A = 30 e 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)

Leggi LogLog Graphing online: https://riptutorial.com/it/matplotlib/topic/10145/loglog-graphing

Capitolo 13: Manipolazione dell'immagine

Examples

Immagini di apertura

Matplotlib include l' `image` modulo per la manipolazione di immagini

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

Le immagini vengono lette dal file (solo `.png`) con la funzione `imread` :

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

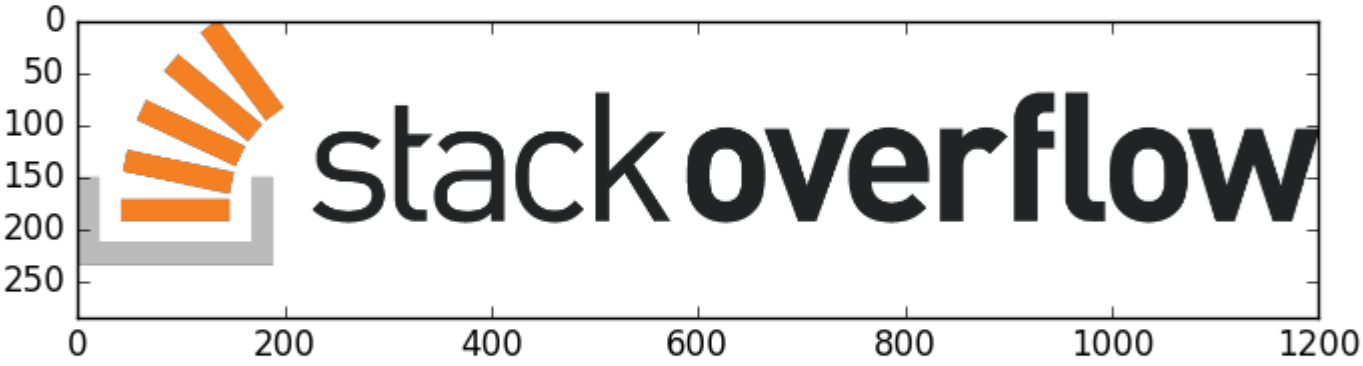
e sono resi dalla funzione `imshow` :

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

Cerchiamo di *tracciare* il logo Stack Overflow :

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

La trama risultante è



Leggi Manipolazione dell'immagine online:
https://riptutorial.com/it/matplotlib/topic/4575/manipolazione-dell-immagine

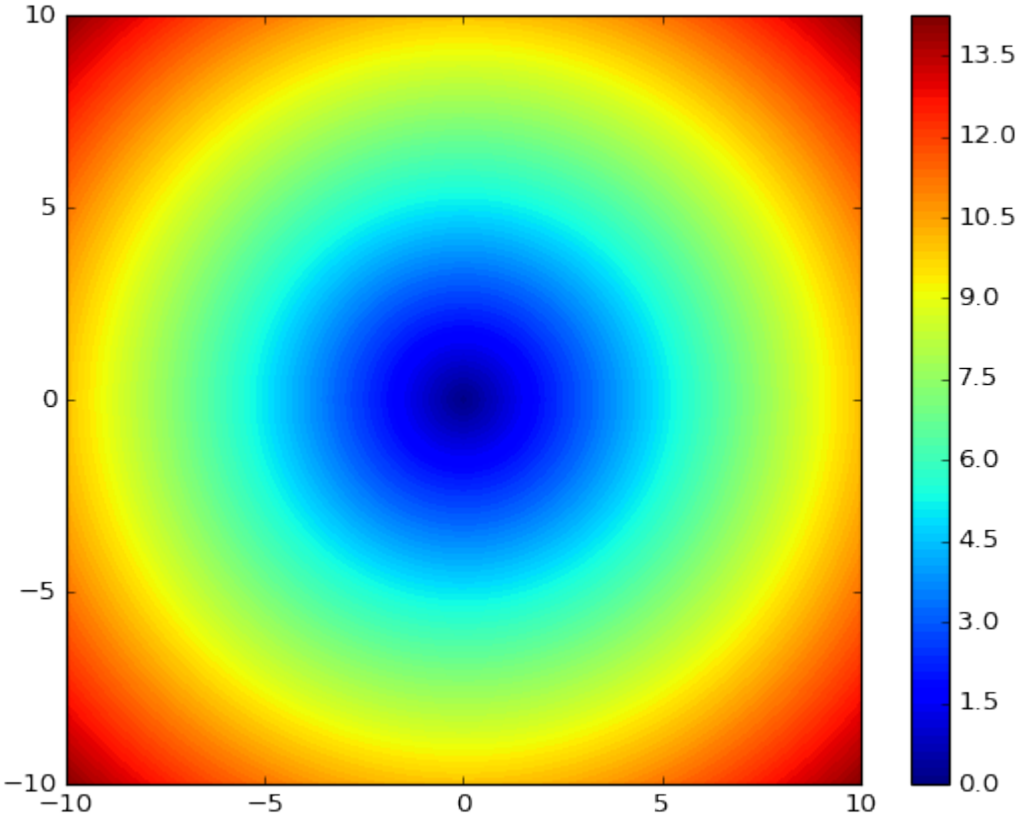
Capitolo 14: Mappe di contorno

Examples

Semplice disegno del contorno riempito

```
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)
```

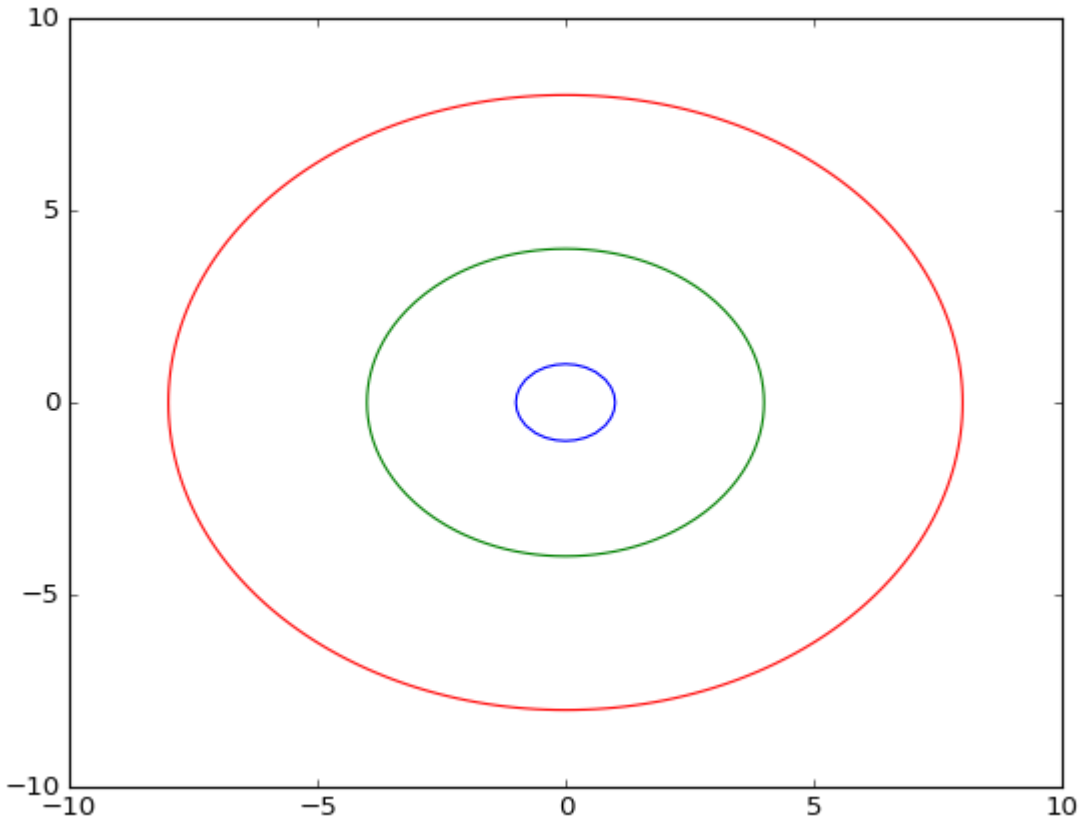
Risultato:



Semplice tracciatura dei contorni

```
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'])
```

Risultato:



Leggi Mappe di contorno online: https://riptutorial.com/it/matplotlib/topic/8644/mappe-di-contorno

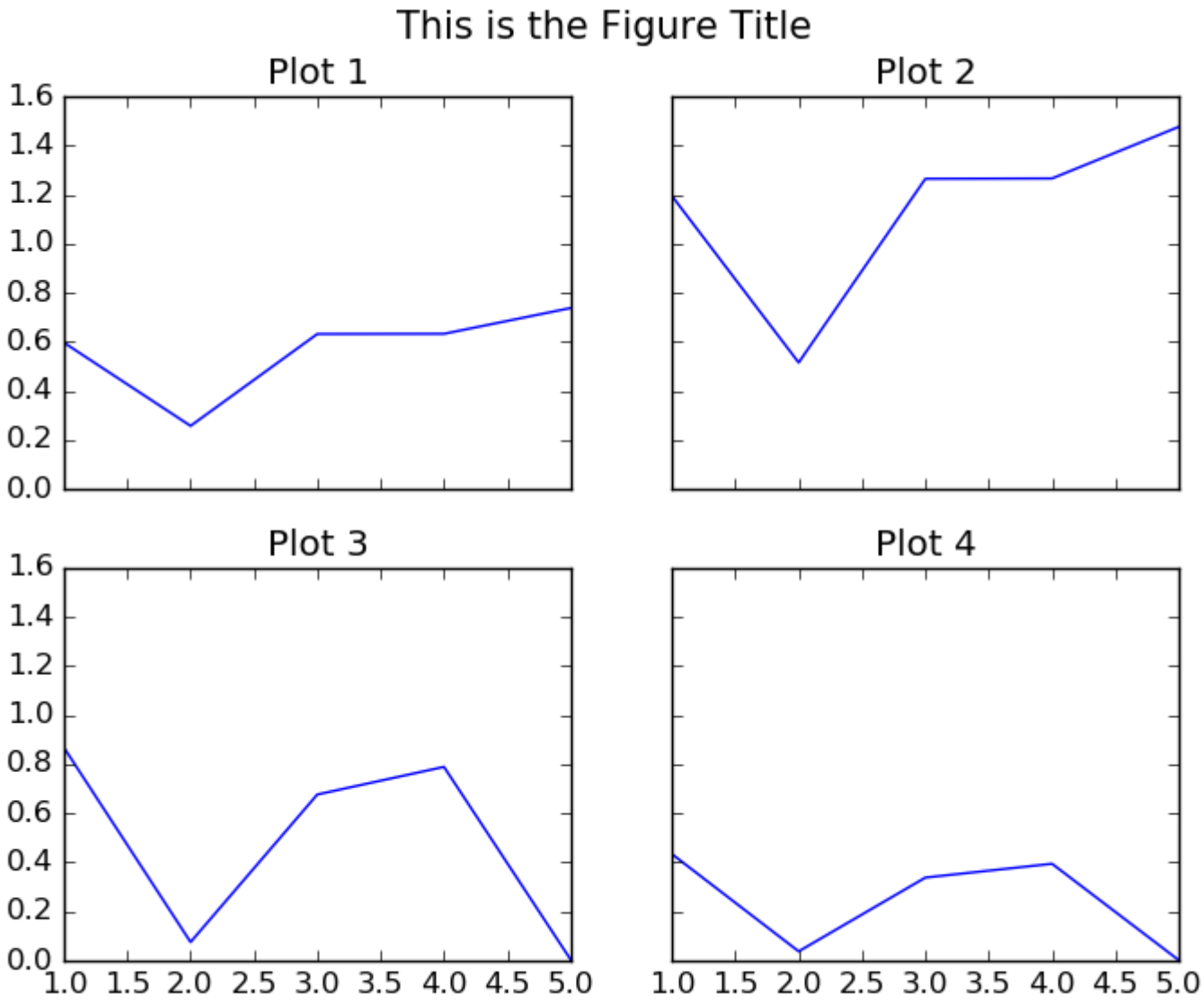
Capitolo 15: Plot multipli

Sintassi

- Elemento dell'elenco

Examples

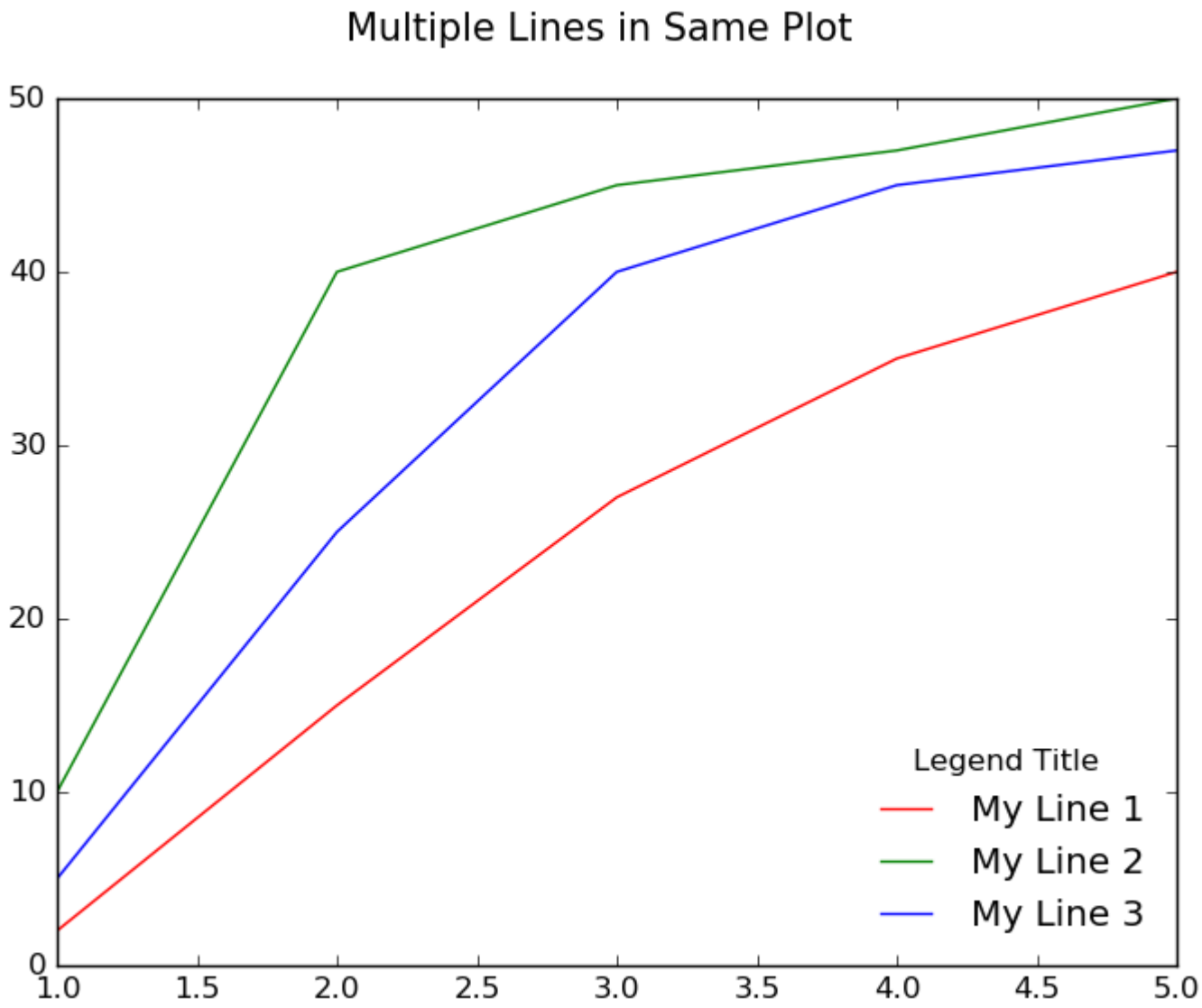
Griglia di sottotrama usando sottotrama




```
"""
================================================================================
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()
```

Più linee / curve nello stesso grafico




```
"""
================================================================================
                    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")
```

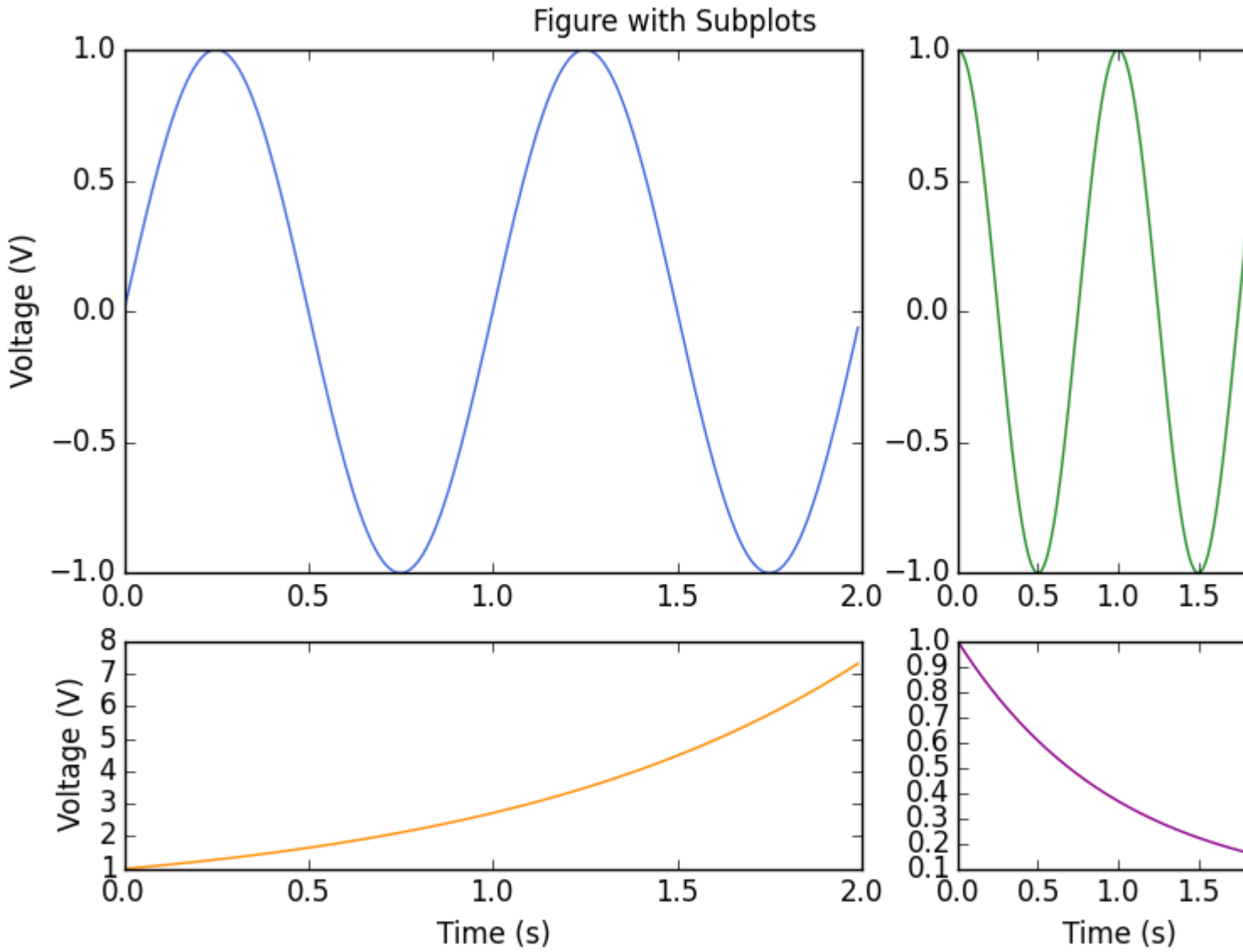
```
ax.plot(x, y3, color="blue", label="My Line 3")

# 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()
```

Plot multipli con gridspec

Il pacchetto `gridspec` consente un maggiore controllo sul posizionamento delle sottotrame. Rende molto più facile controllare i margini dei grafici e la spaziatura tra le singole sottotrame. Inoltre, consente di avere assi di dimensioni diverse sulla stessa figura definendo assi che occupano più posizioni di griglia.

```
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)')
```

Questo codice produce la trama mostrata sotto.

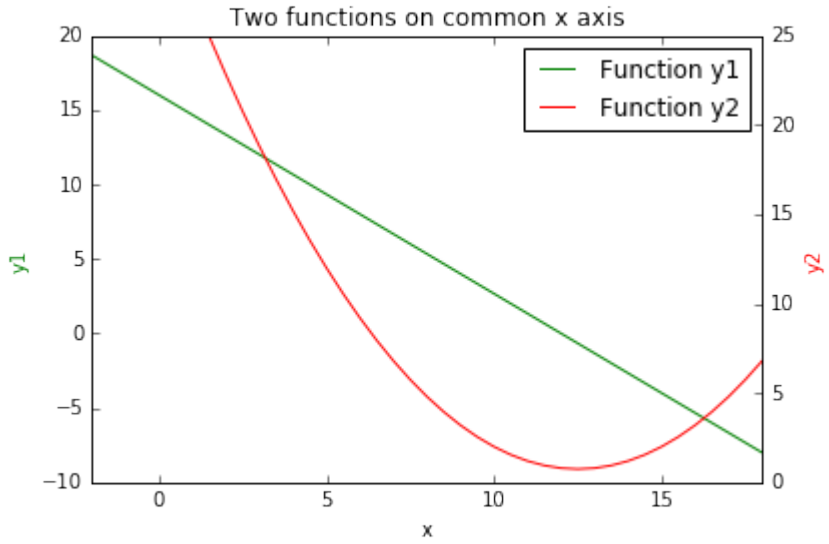


Un grafico di 2 funzioni sull'asse x condiviso.

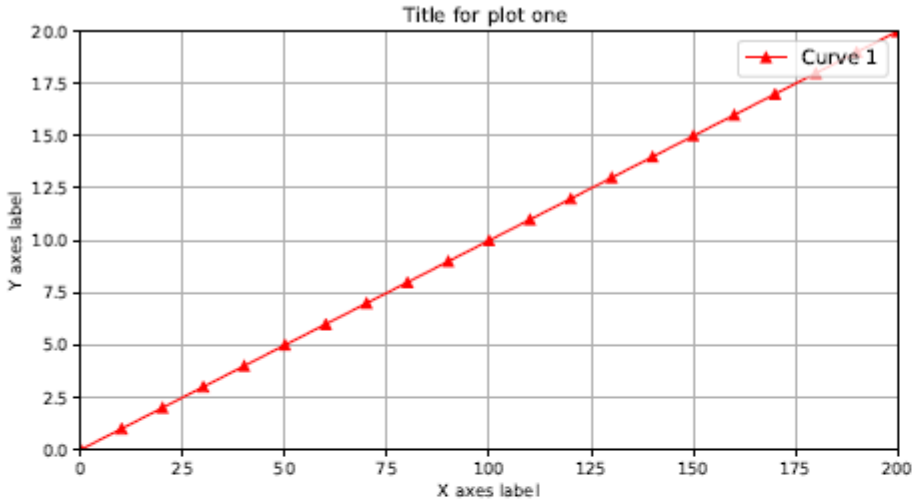
```
import numpy as np
import matplotlib.pyplot as plt

# 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()
```

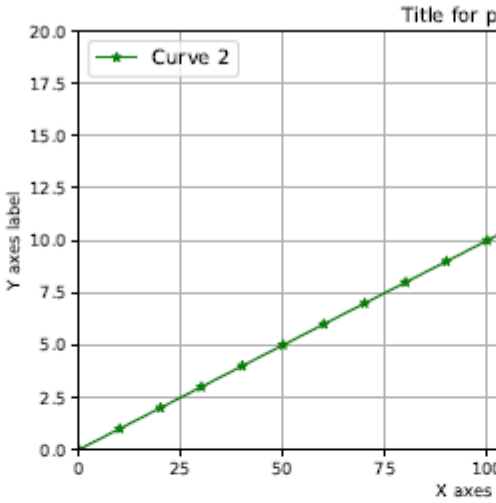
```
# create line plot of y2(x)
line2, = ax2.plot(x, y2, 'r', label="Function y2")
ax2.set_ylabel('y2', color='r')

# 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()
```

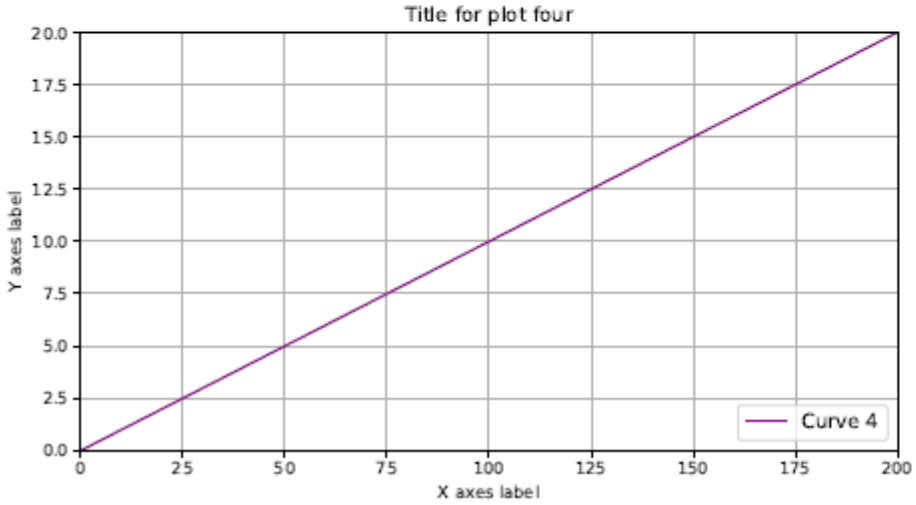
Questo codice produce la trama mostrata sotto.

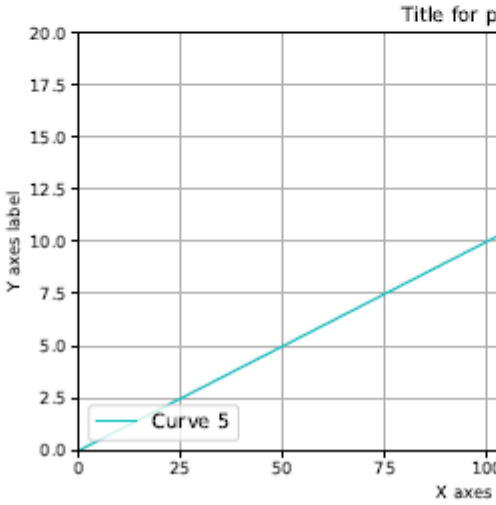


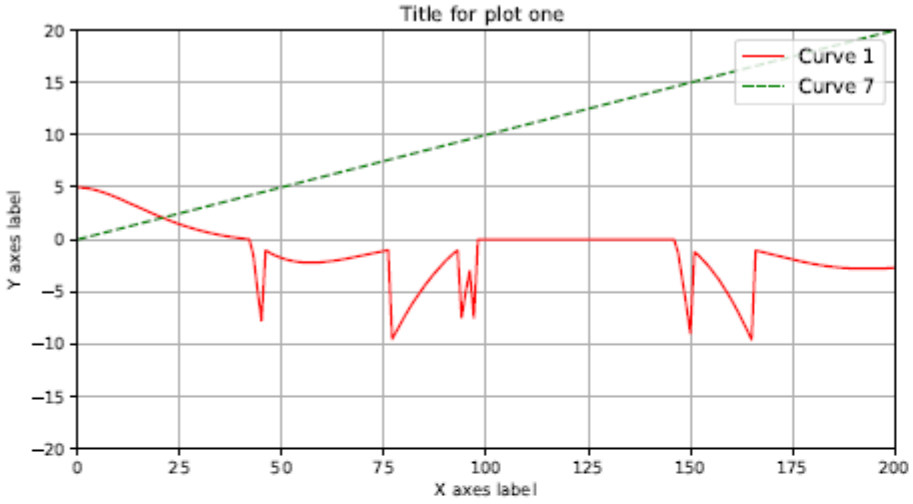
Più lotti e più funzioni di stampa

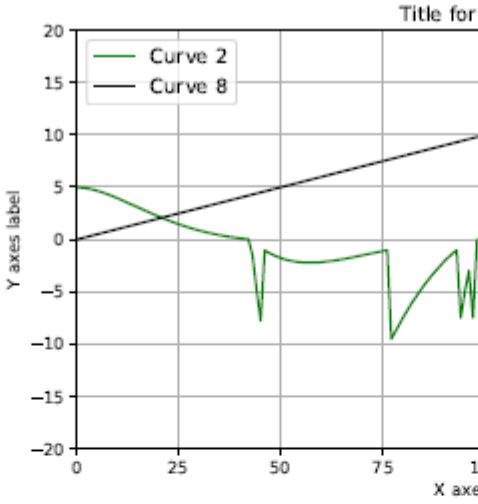

Title for plot one
Curve 1
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200

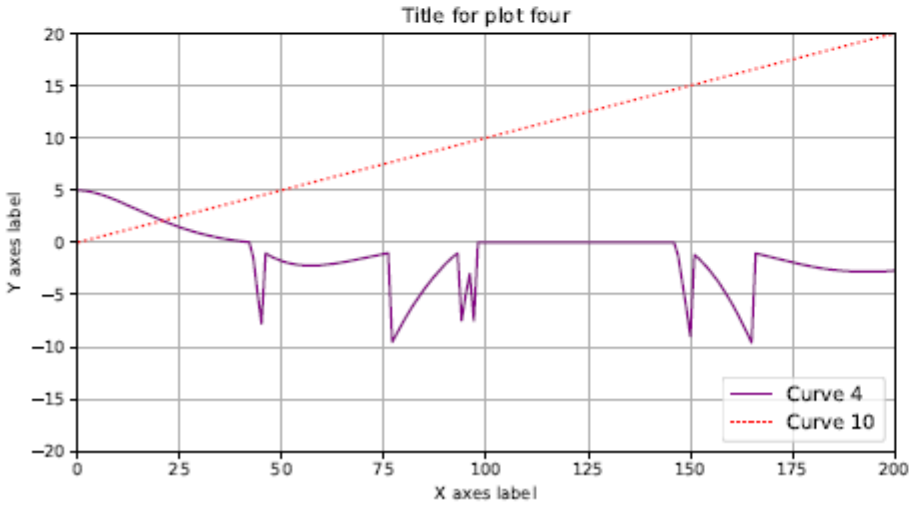

Curve 2
Y axes label
X axes
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75

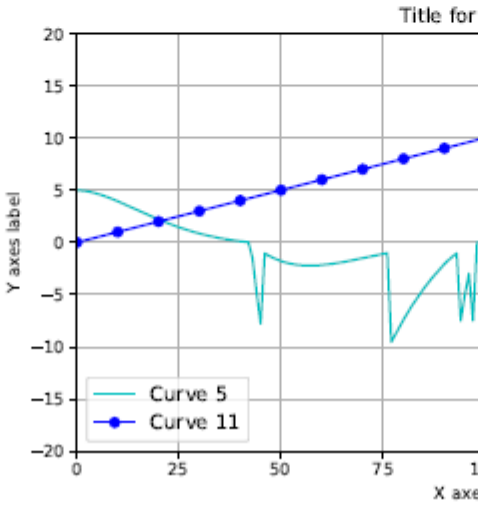

Title for plot four
Curve 4
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200


Curve 5
Y axes label
X axes
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75









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()
```

Leggi Plot multipli online: https://riptutorial.com/it/matplotlib/topic/3279/plot-multipli

Capitolo 16: Sistemi di coordinate

Osservazioni

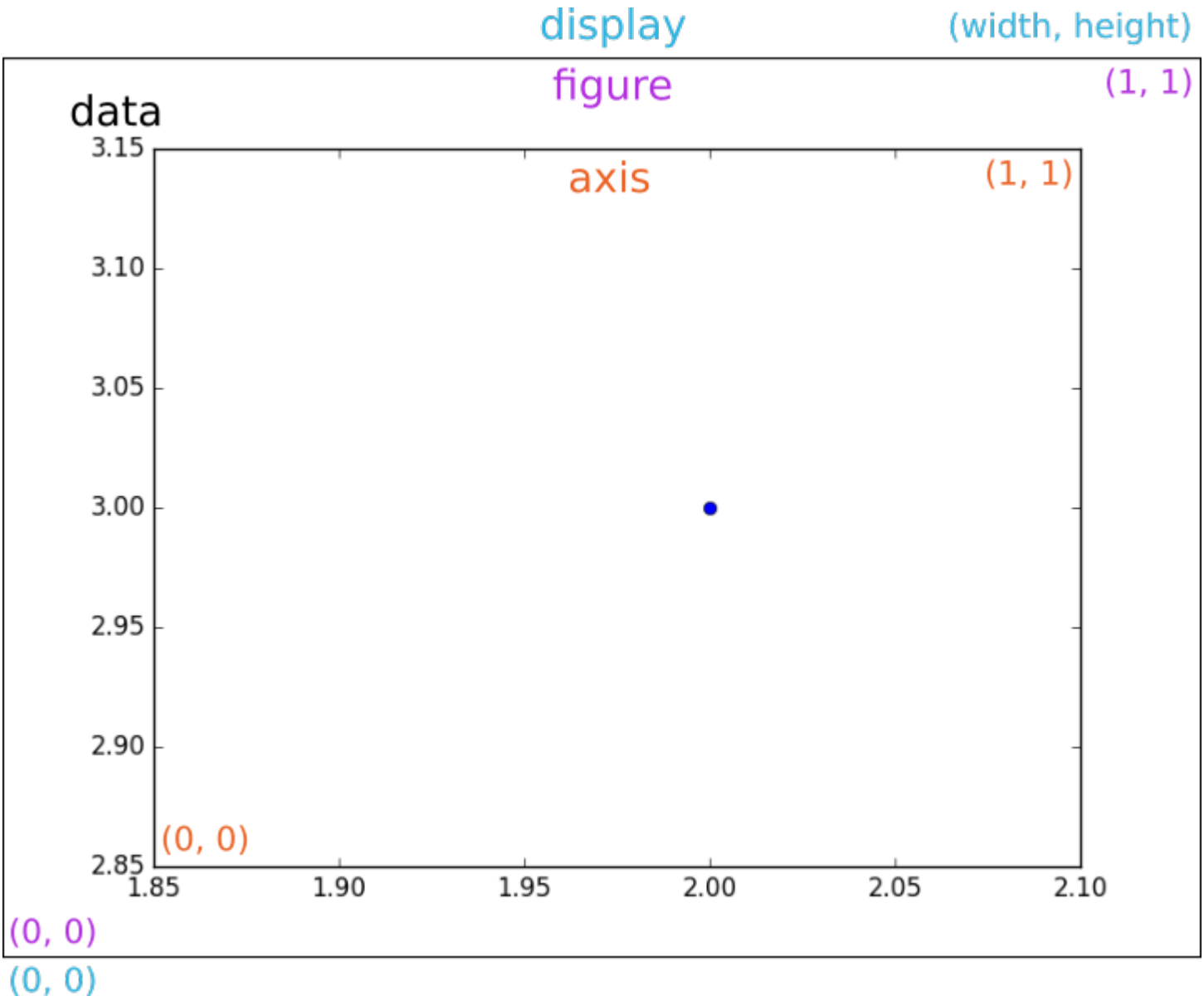
Matplotlib ha quattro distinti sistemi di coordinate che possono essere sfruttati per facilitare il posizionamento di oggetti diversi, ad esempio il testo. Ogni sistema ha un oggetto di trasformazione corrispondente che trasforma le coordinate da quel sistema al cosiddetto sistema di coordinate di visualizzazione.

Il sistema di coordinate di dati è il sistema definito dai dati sui rispettivi assi. È utile quando si tenta di posizionare un oggetto rispetto ai dati tracciati. L'intervallo è dato dalle proprietà `xlim` e `ylim` degli `Axes` . L'oggetto di trasformazione corrispondente è `ax.transData` .

Il sistema di coordinate degli assi è il sistema legato al suo oggetto `Axes` . I punti (0, 0) e (1, 1) definiscono gli angoli in basso a sinistra e in alto a destra degli assi. Come tale è utile quando si posiziona rispetto agli assi, come in alto al centro del grafico. L'oggetto di trasformazione corrispondente è `ax.transAxes` .

Il sistema di coordinate della figura è analogo al sistema di coordinate degli assi, tranne per il fatto che è legato alla `Figure` . I punti (0, 0) e (1, 1) rappresentano gli angoli in basso a sinistra e in alto a destra della figura. È utile quando si tenta di posizionare qualcosa rispetto all'intera immagine. Il suo oggetto di trasformazione corrispondente è `fig.transFigure` .

Il sistema di coordinate di visualizzazione è il sistema dell'immagine espressa in pixel. Punti (0, 0) e (larghezza, altezza) sono i pixel in basso a sinistra e in alto a destra dell'immagine o del display. Può essere usato per posizionarsi in modo assoluto. Poiché gli oggetti di trasformazione trasformano le coordinate in questo sistema di coordinate, il sistema di visualizzazione non ha alcun oggetto di trasformazione ad esso associato. Tuttavia, se necessario, è possibile utilizzare `None` o `matplotlib.transforms.IdentityTransform()` .



Maggiori dettagli sono disponibili qui .

Examples

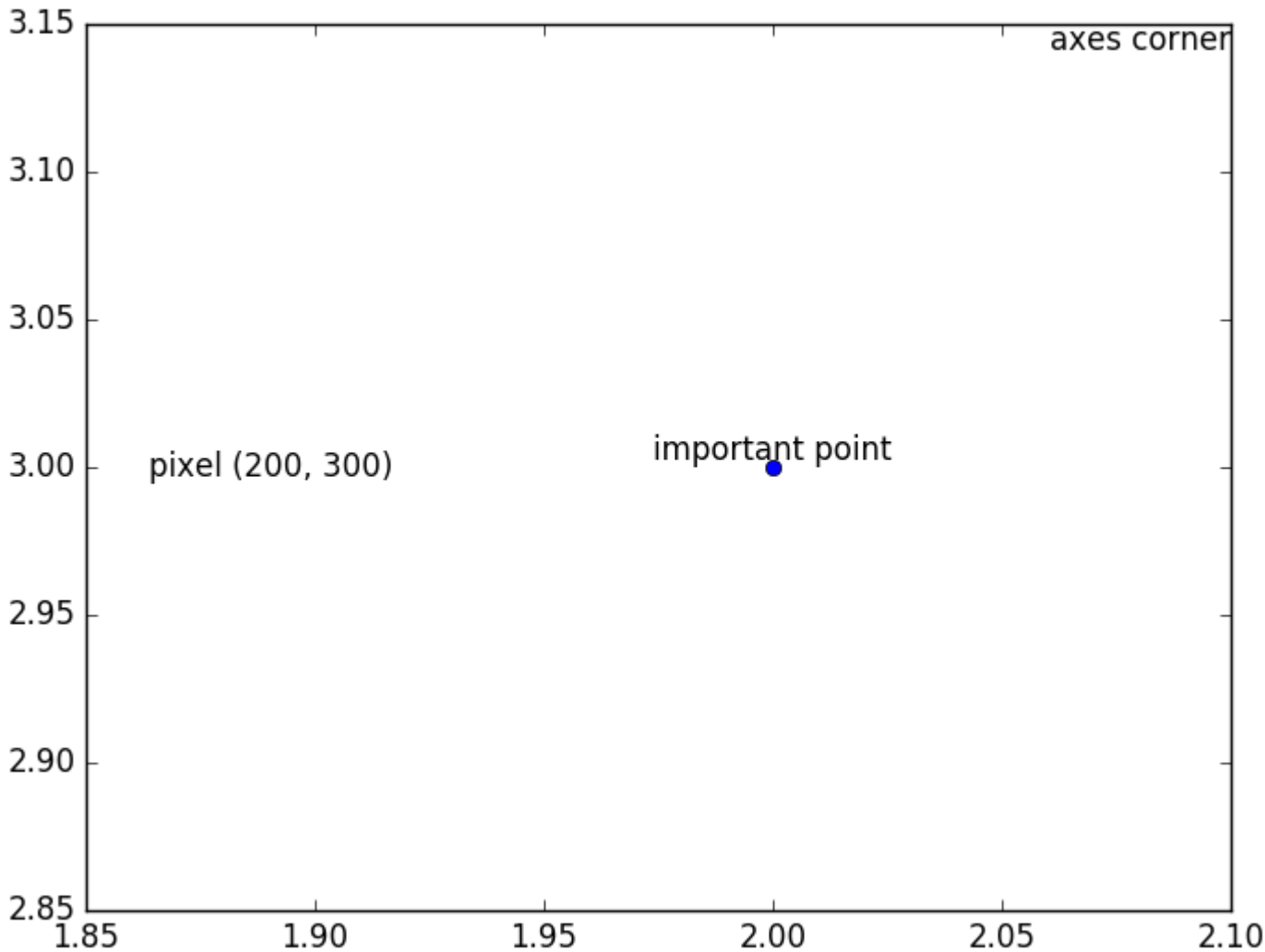
Coordinare sistemi e testo

I sistemi di coordinate di Matplotlib sono molto utili quando si tenta di annotare le trame che si fanno. A volte ti piacerebbe posizionare il testo relativamente ai tuoi dati, come quando cerchi di etichettare un punto specifico. Altre volte vorresti aggiungere un testo in cima alla figura. Questo può essere facilmente ottenuto selezionando un sistema di coordinate appropriato passando un oggetto di `transform` parametro di `transform` in call to `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



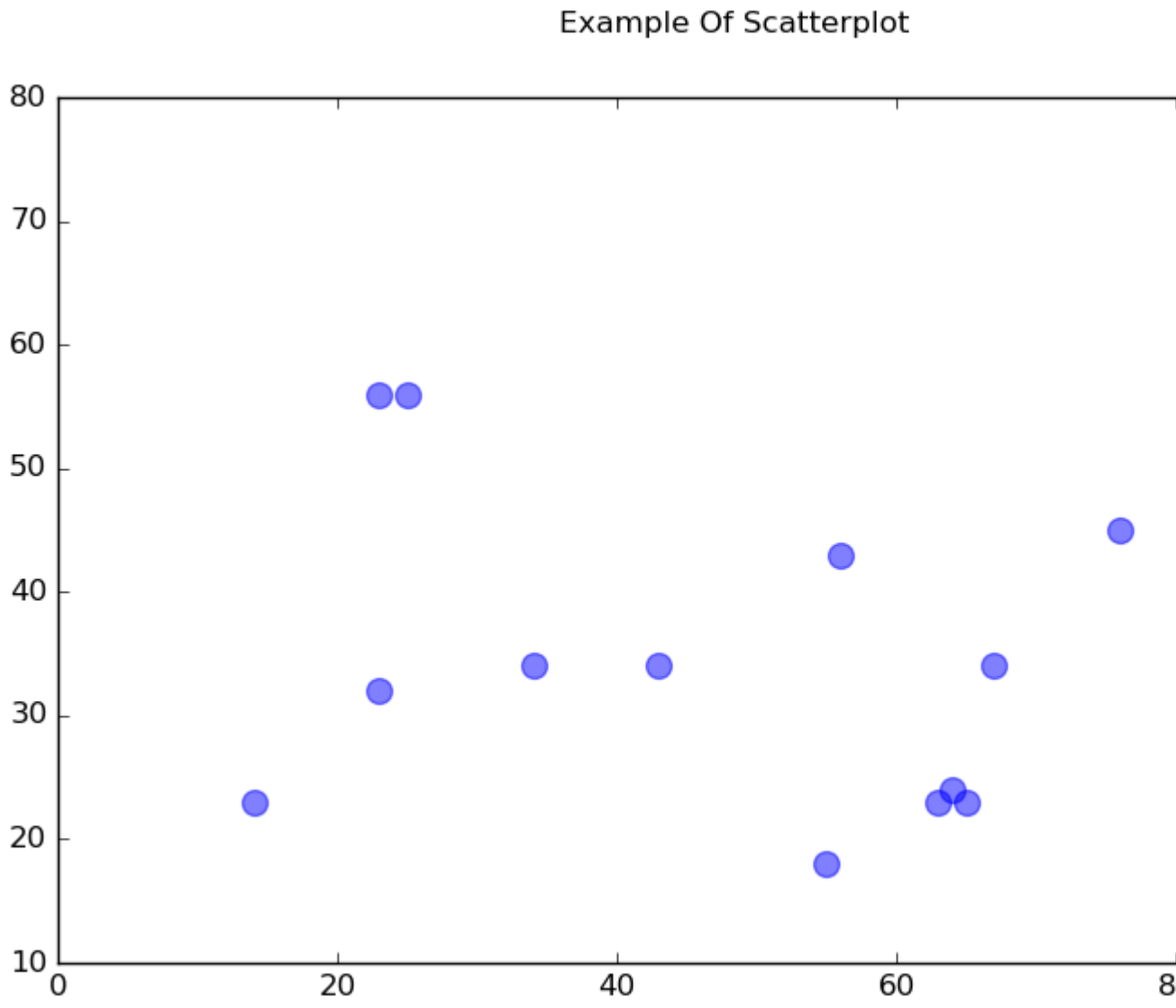
Leggi Sistemi di coordinate online: https://riptutorial.com/it/matplotlib/topic/4566/sistemi-di-coordinate

Capitolo 17: Trame di base

Examples

Grafici a dispersione

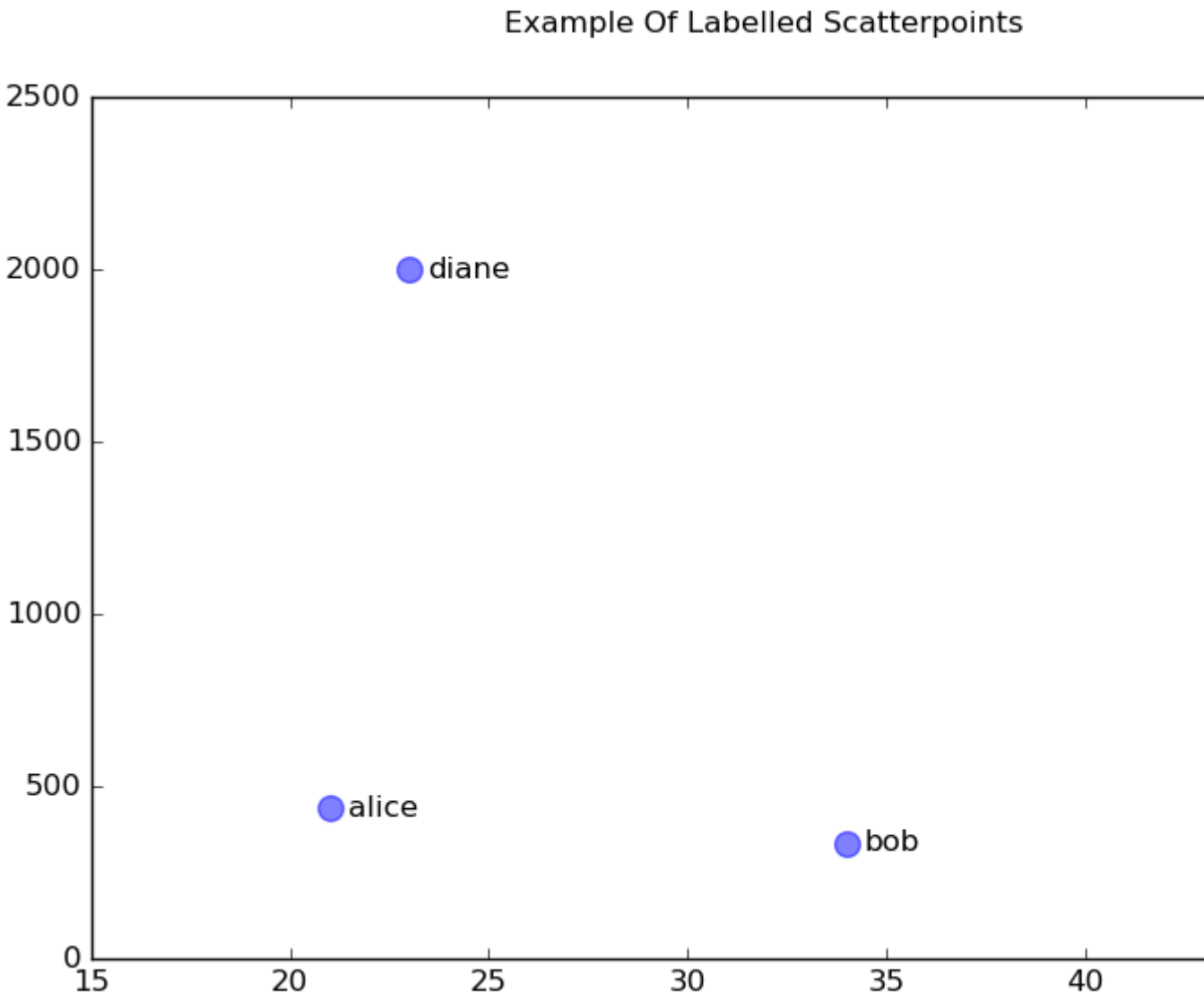
Una trama a dispersione semplice




```
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 piano di dispersione con punti etichettati

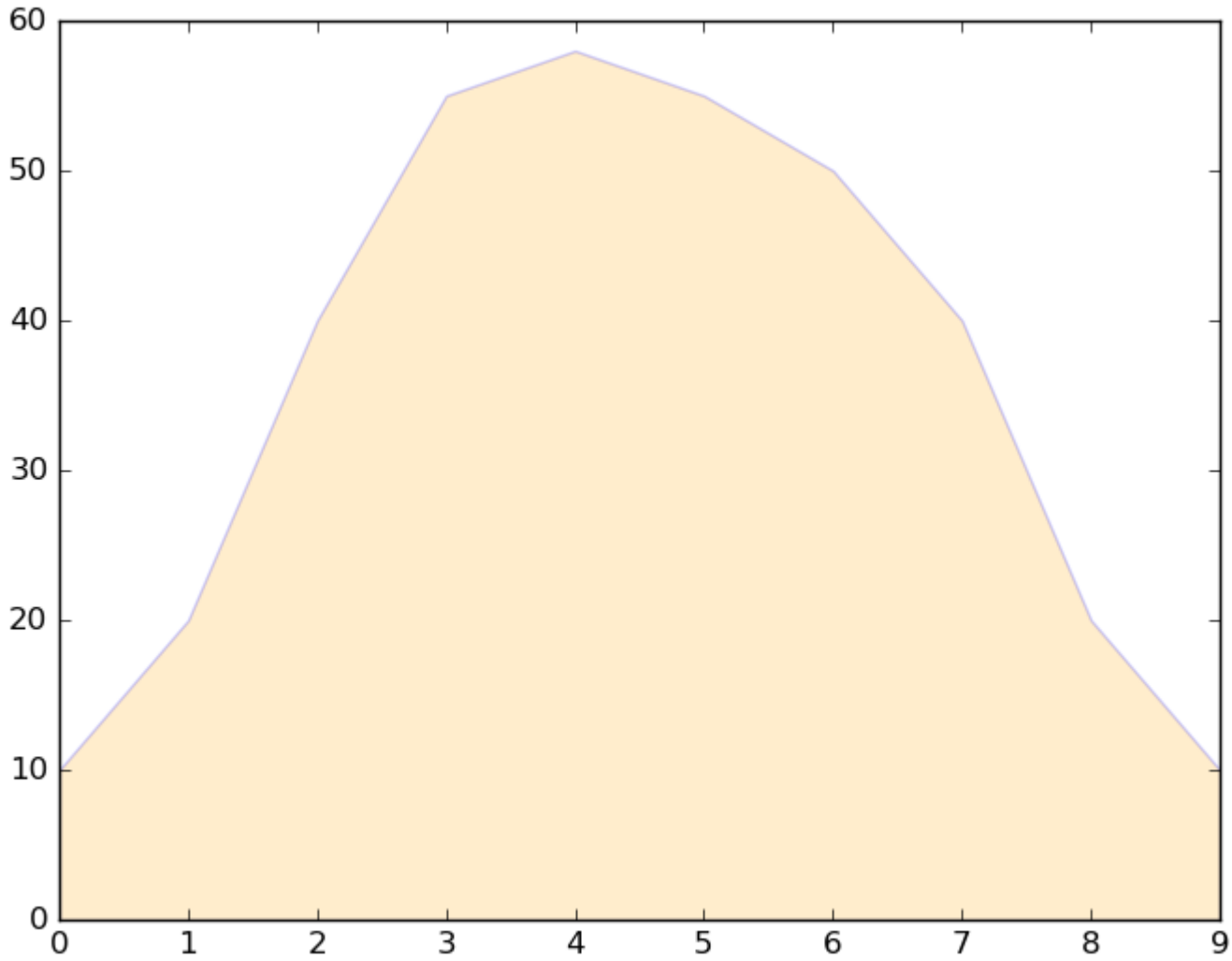



```
import matplotlib.pyplot as plt

# 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()
```

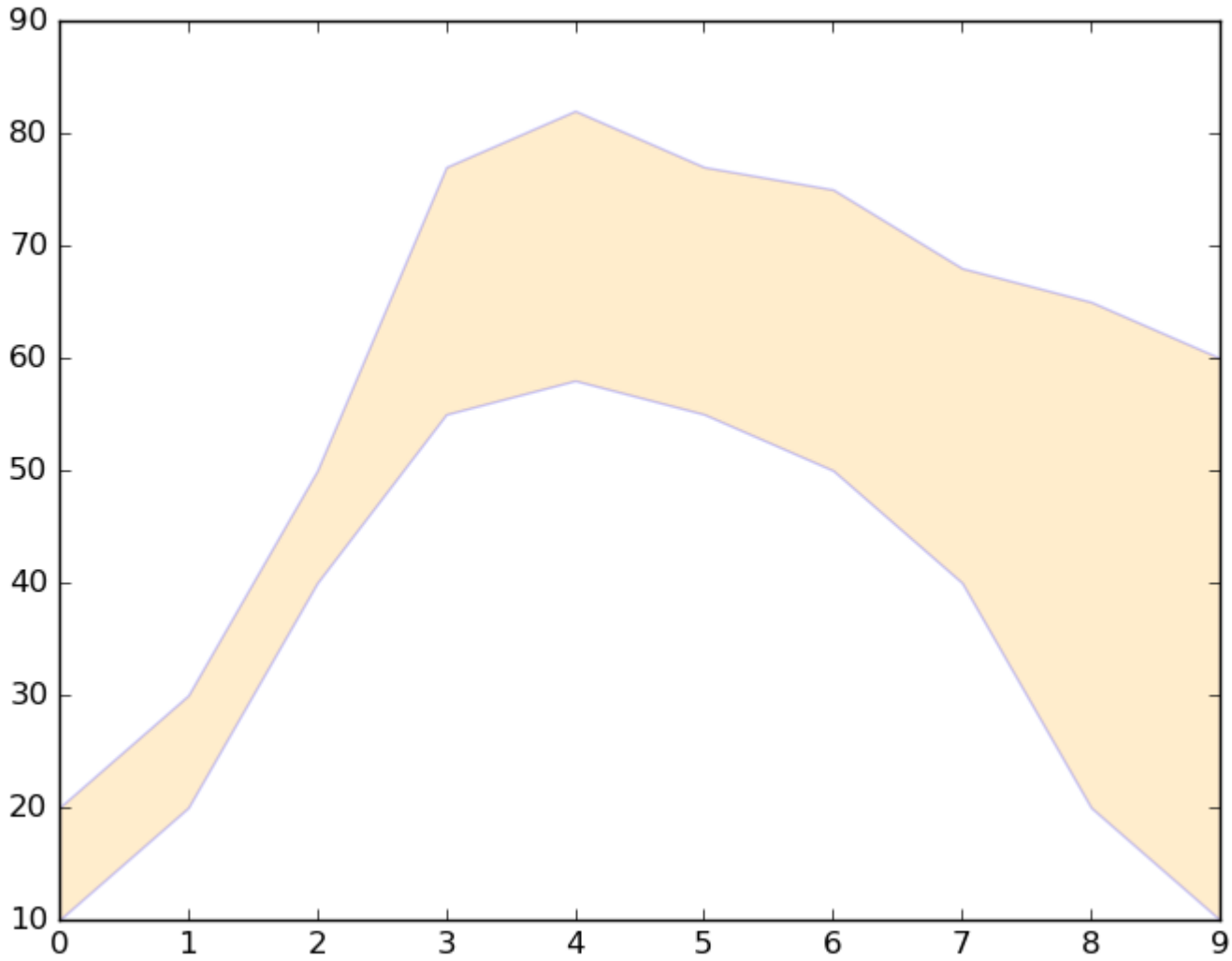
Piazzole ombreggiate

Regione ombreggiata sotto una linea




```
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()
```

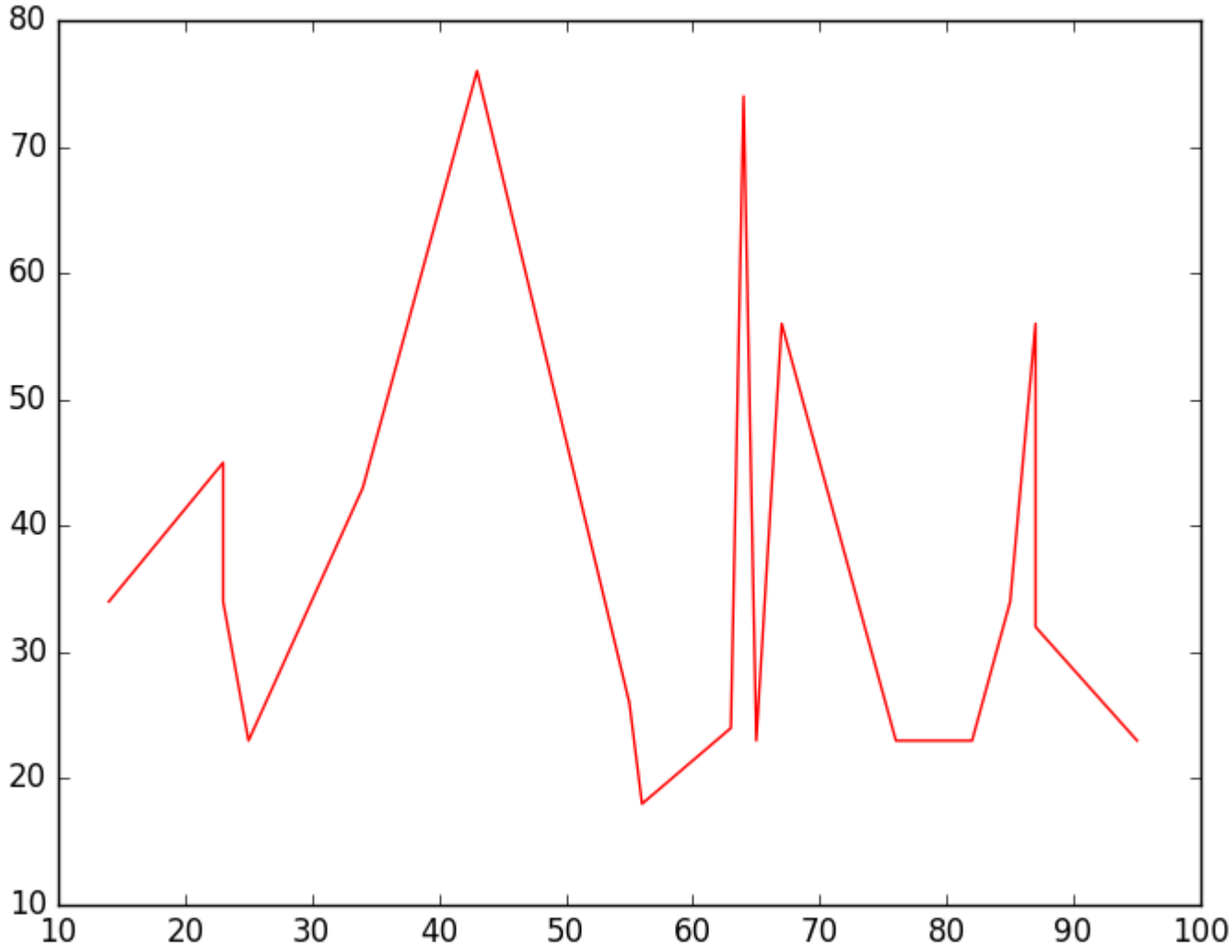
Regione ombreggiata tra due linee




```
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()
```

Line plot

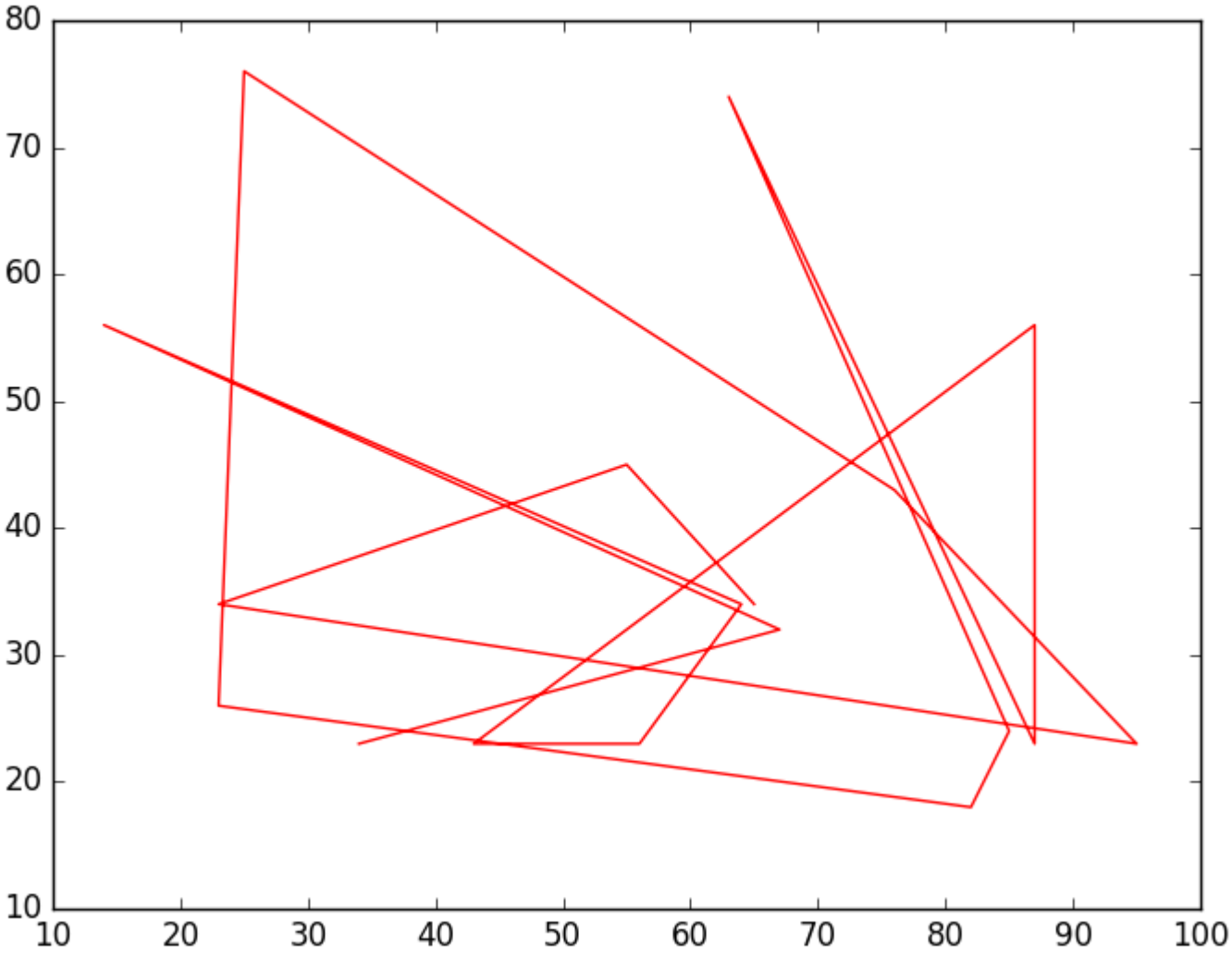
Trama semplice




```
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()
```

Nota che in generale `y` non è una funzione di `x` e anche che i valori in `x` non hanno bisogno di essere ordinati. Ecco come appare un grafico a linee con valori x non ordinati:

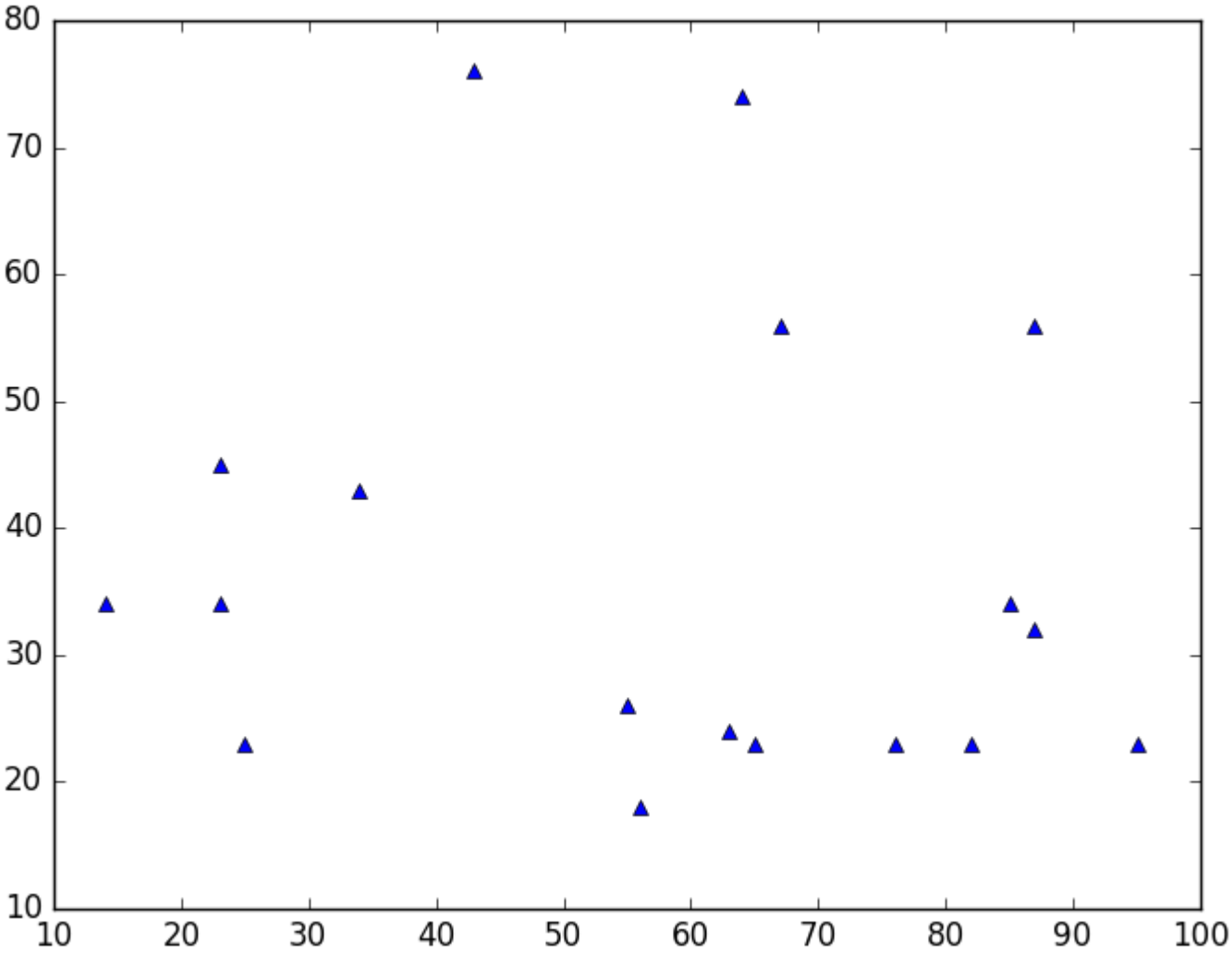
```
# shuffle the elements in x
np.random.shuffle(x)
plt.plot(x, y, 'r-')
plt.show()
```



Trama dei dati

È simile a un grafico a dispersione , ma utilizza invece la funzione `plot()` . L'unica differenza nel codice qui è l'argomento di stile.

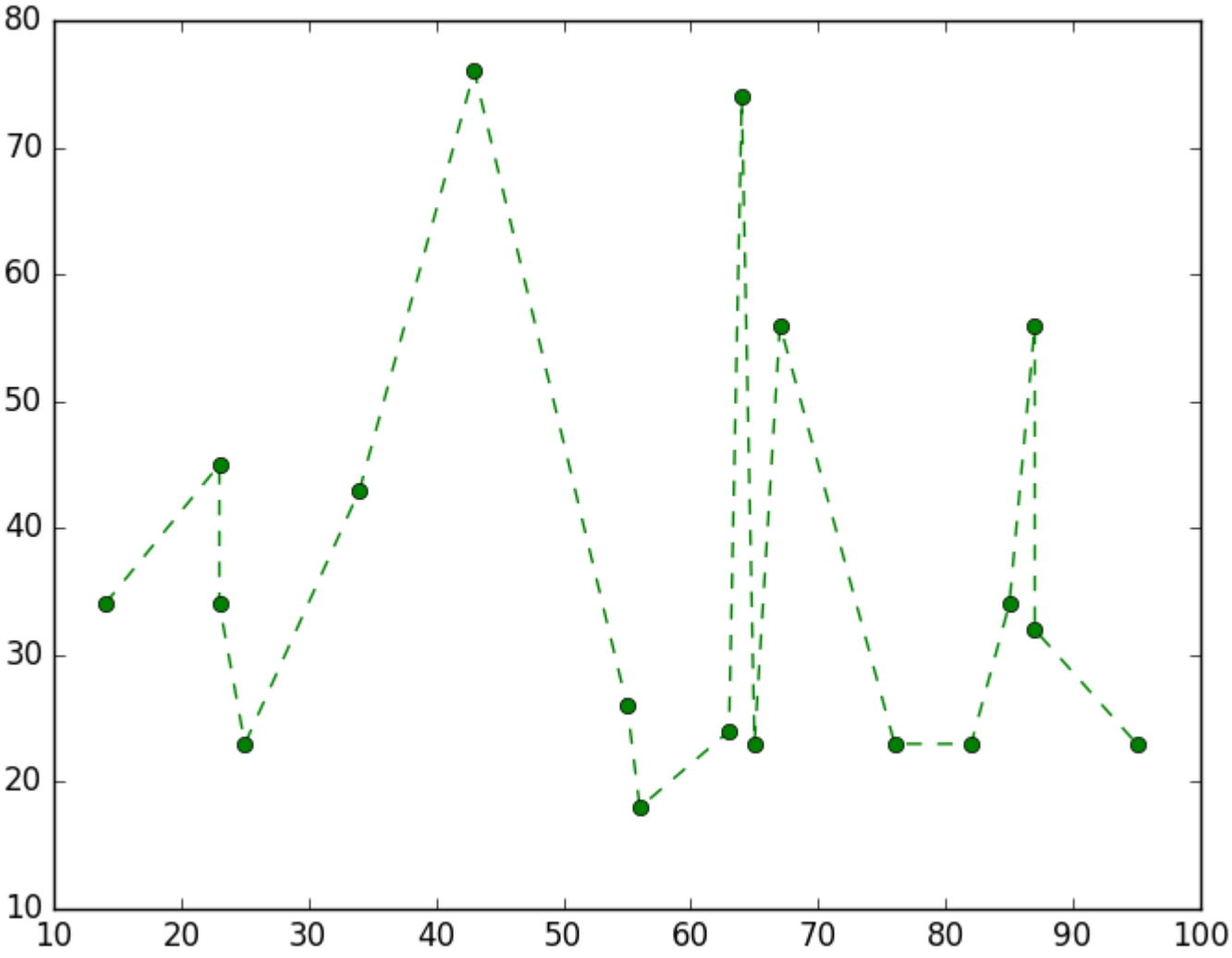
```
plt.plot(x, y, 'b^')
# Create blue up-facing triangles
```



Dati e linea

L'argomento di stile può assumere simboli sia per i marcatori che per lo stile della linea:

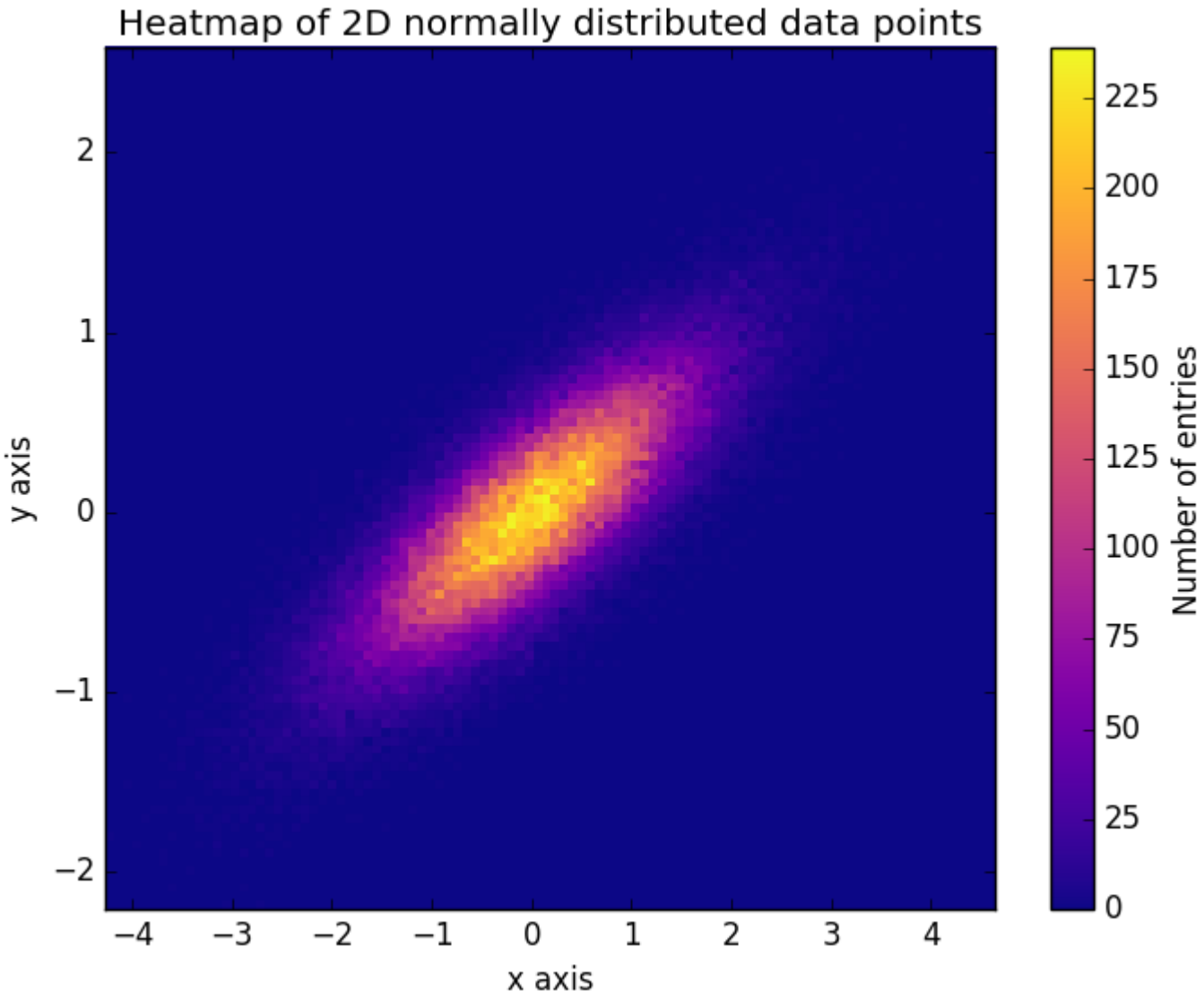
```
plt.plot(x, y, 'go--')
# green circles and dashed line
```



Mappa di calore

Le Heatmap sono utili per visualizzare le funzioni scalari di due variabili. Forniscono un'immagine "piatta" di istogrammi bidimensionali (che rappresentano ad esempio la densità di una determinata area).

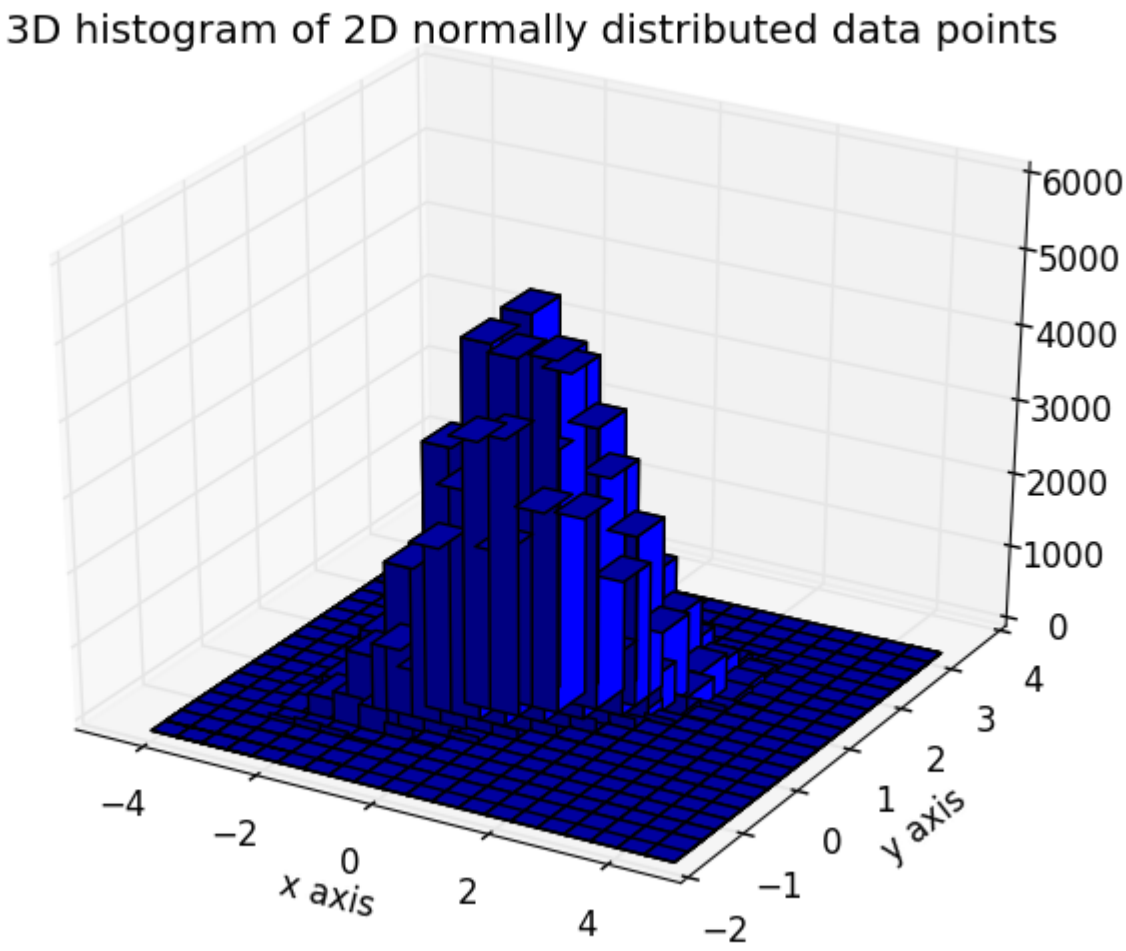
Il seguente codice sorgente illustra mappe di calore usando numeri distribuiti normalmente bivariati centrati a 0 in entrambe le direzioni (significa `[0.0, 0.0]`) e una con una data matrice di covarianza. I dati sono generati usando la funzione numpy numpy.random.multivariate_normal ; viene quindi `hist2d` funzione hist2d di 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()
```

Ecco gli stessi dati visualizzati come un istogramma 3D (qui usiamo solo 20 contenitori per l'efficienza). Il codice è basato su questa demo di 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.
```

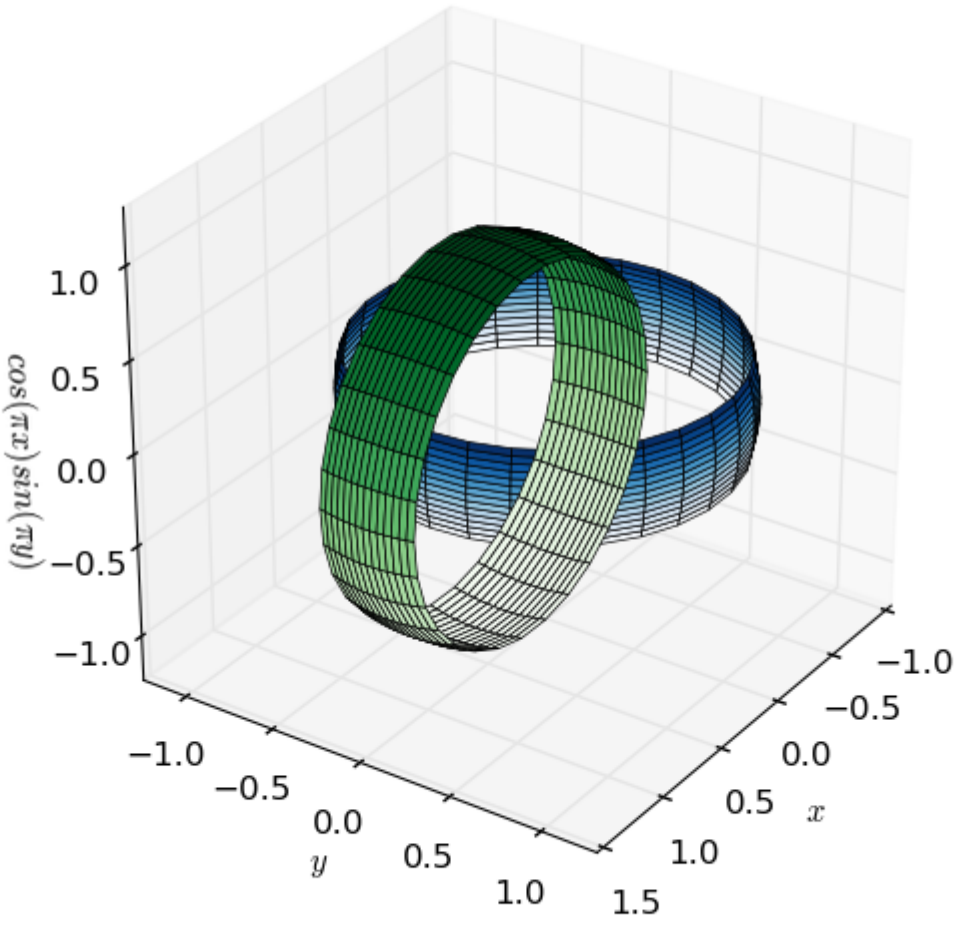
```
N_numbers = 100000
N_bins = 20

# 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()
```

Leggi Trame di base online: https://riptutorial.com/it/matplotlib/topic/3266/trame-di-base

Capitolo 18: Trame tridimensionali

Osservazioni

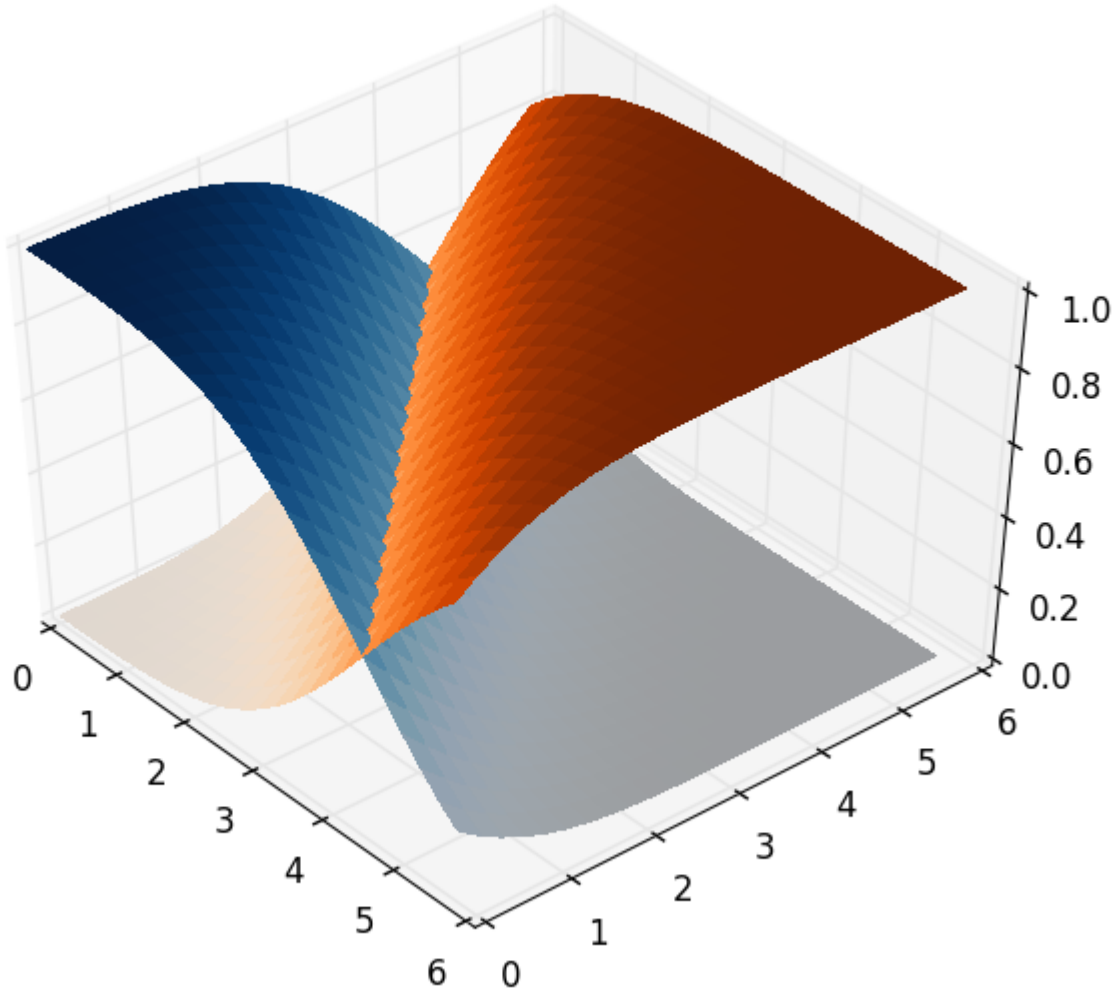
Il plottaggio tridimensionale in matplotlib è stato storicamente un po 'un kludge, in quanto il motore di rendering è intrinsecamente 2d. Il fatto che le configurazioni 3d siano renderizzate tracciando un blocco 2d dopo l'altro implica che spesso ci sono problemi di rendering relativi alla profondità apparente degli oggetti. Il nocciolo del problema è che due oggetti non connessi possono essere completamente dietro, o completamente uno di fronte all'altro, il che porta a artefatti come mostrato nella figura sottostante di due anelli interbloccati (fare clic per le GIF animate):

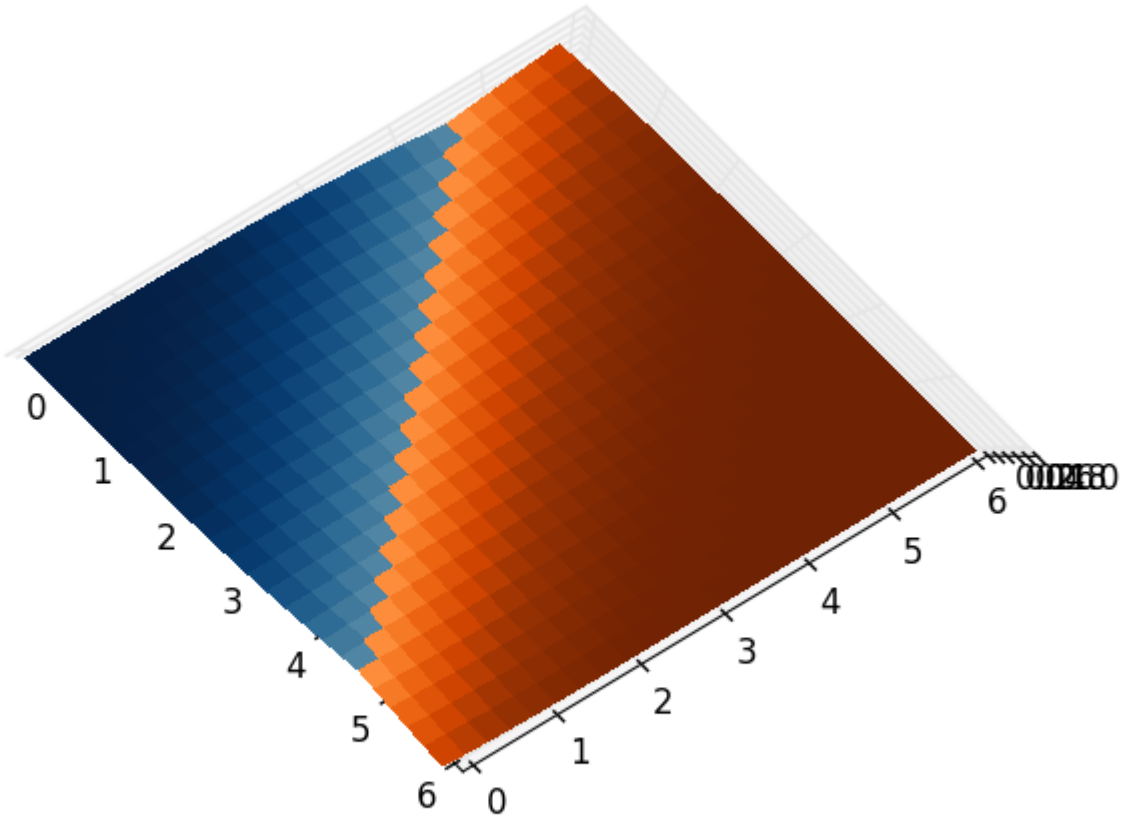


Questo può tuttavia essere risolto. Questo artefatto esiste solo quando si tracciano più superfici sullo stesso grafico, poiché ciascuna viene rappresentata come una forma 2D piatta, con un singolo parametro che determina la distanza di visualizzazione. Noterai che una singola superficie complicata non ha lo stesso problema.

Il modo per ovviare a questo è unire gli oggetti della trama usando ponti trasparenti:

```
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

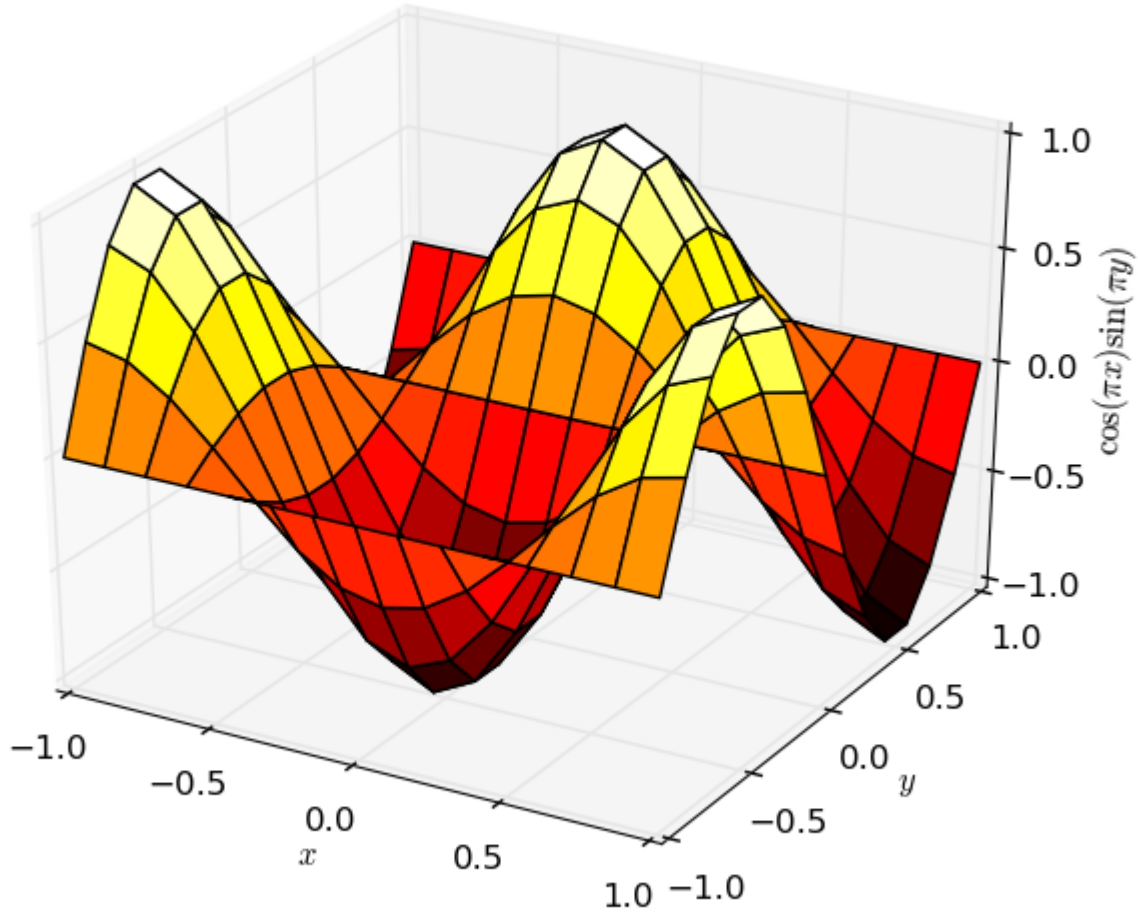
Creazione di assi tridimensionali

Gli assi Matplotlib sono bidimensionali per impostazione predefinita. Per creare grafici tridimensionali, dobbiamo importare la classe `Axes3D` dal toolkit mplot3d , che abiliterà un nuovo tipo di proiezione per un asse, ovvero `'3d'` :

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

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

Oltre alle generalizzazioni dirette di grafici bidimensionali (come grafici a linee , grafici a dispersione , grafici a barre , grafici di contorno), sono disponibili diversi metodi di tracciatura superficiale , ad esempio `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()
```



Leggi Trame tridimensionali online: https://riptutorial.com/it/matplotlib/topic/1880/trame-tridimensionali

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con matplotlib	Amitay Stern, ChaoticTwist, Chr, Chris Mueller, Community, dermen, evtoh, farenorth, Josh, jrjc, pmos, Serenity, tacaswell
2	Animazioni e trama interattiva	FiN, smurfendrek123, user2314737
3	Chiusura di una finestra di figura	Brian, David Zwicker
4	colormap	Andras Deak, Xevaquor
5	Figure e oggetti degli assi	David Zwicker, Josh, Serenity, tom
6	grafici a scatole	Luis
7	Integrazione con TeX / LaTeX	Andras Deak, Bosoneando, Chris Mueller, Næreen, Serenity
8	Istogramma	Yegor Kishilov
9	Legends	Andras Deak, Franck Dernoncourt, ronrest, saintsfan342000, Serenity
10	Linee di griglia e segni di graduazione	ronrest
11	LogLog Graphing	ml4294
12	Manipolazione dell'immagine	Bosoneando
13	Mappe di contorno	Eugene Loy, Serenity
14	Plot multipli	Chris Mueller, Robert Branam, ronrest, swatchai
15	Sistemi di coordinate	jure
16	Trame di base	Franck Dernoncourt, Josh, ml4294, ronrest, Scimonster, Serenity, user2314737
17	Trame tridimensionali	Andras Deak, Serenity, will