eBook Gratuit

APPRENEZ matplotlib

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#matplotlib

Table des matières

À propos

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

Chapitre 1: Commencer avec matplotlib

Remarques

Vue d'ensemble

matplotlib est une bibliothèque de traçage pour Python. Il fournit des API orientées objet pour l'intégration de tracés dans les applications. Il est similaire à MATLAB en termes de capacité et de syntaxe.

Il a été écrit à l'origine par JDHunter et est activement développé. Il est distribué sous une licence de style BSD.

Versions

Version	Versions Python prises en charge	Remarques	Date de sortie
1.3.1	2.6, 2.7, 3.x	Ancienne version stable	2013-10-10
1.4.3	2.6, 2.7, 3.x	Version stable précédente	2015-07-14
1.5.3	2.7, 3.x	Version stable actuelle	2016-01-11
2.x	2.7, 3.x	Dernière version de développement	2016-07-25

Examples

Installation et configuration

Il existe plusieurs manières d'installer matplotlib, dont certaines dépendent du système que vous utilisez. Si vous avez de la chance, vous pourrez utiliser un gestionnaire de paquets pour installer facilement le module matplotlib et ses dépendances.

les fenêtres

Sur les machines Windows, vous pouvez essayer d'utiliser le gestionnaire de paquets pip pour installer matplotlib. Voir ici pour plus d'informations sur la configuration de pip dans un environnement Windows.

OS X

Il est recommandé d'utiliser le gestionnaire de paquets pip pour installer matplotlib. Si vous devez installer certaines des bibliothèques non-Python sur votre système (par exemple, `libfreetype`), envisagez d'utiliser l' homebrew .

Si vous ne pouvez pas utiliser pip pour quelque raison que ce soit, essayez d'installer depuis le source .

Linux

Idéalement, le gestionnaire de paquets système ou pip devrait être utilisé pour installer matplotlib, en installant le paquet `python-matplotlib` ou en exécutant `pip install matplotlib` .

Si ce n'est pas possible (par exemple, vous n'avez pas de privilèges sudo sur la machine que vous utilisez), vous pouvez installer depuis la source en utilisant l'option `--user` : `python setup.py install --user` . En règle générale, cela va installer matplotlib dans `~/.local` .

Debian / Ubuntu

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

Fedora / Red Hat

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

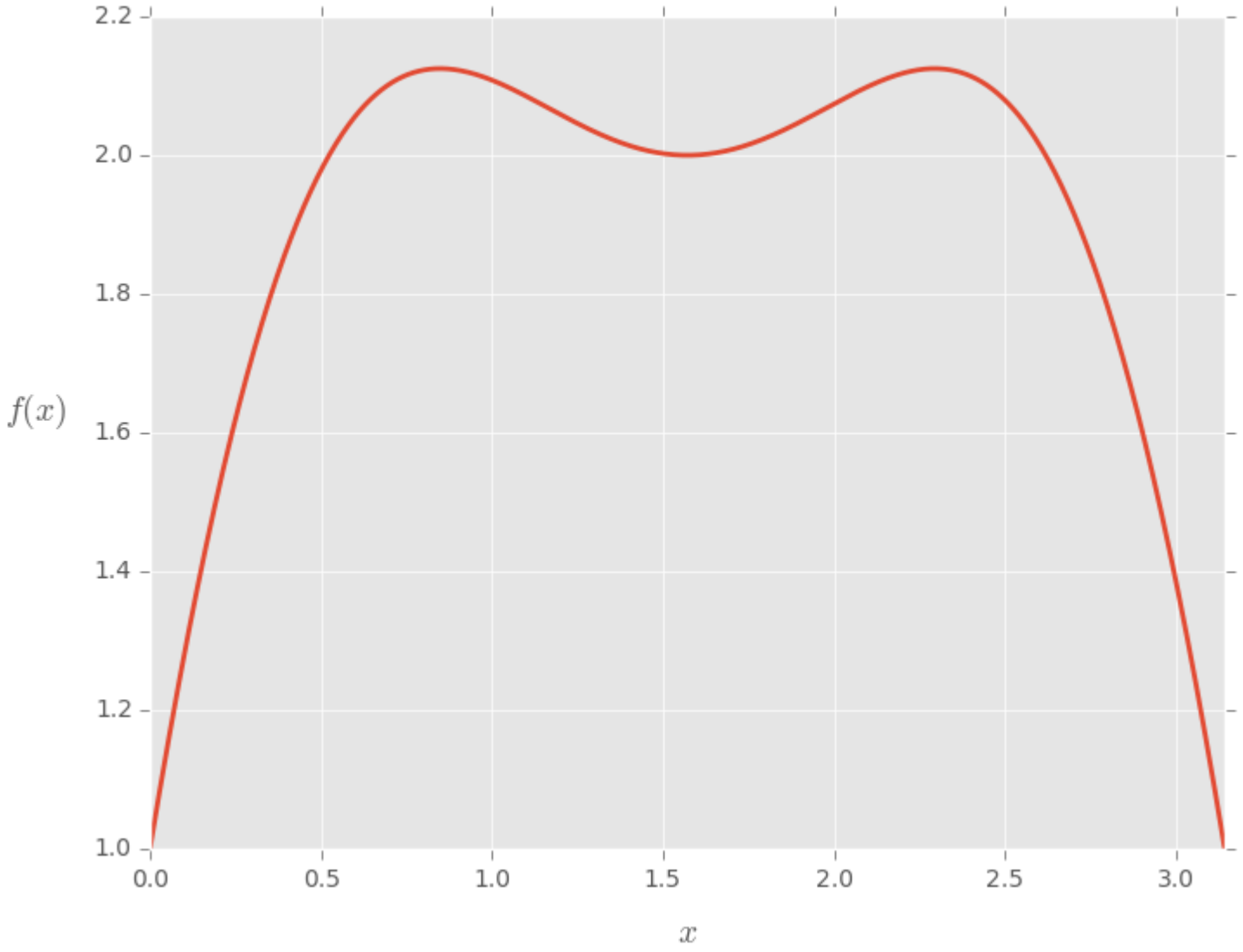
Dépannage

Voir le site Web matplotlib pour des conseils sur la façon de réparer un matplotlib cassé.

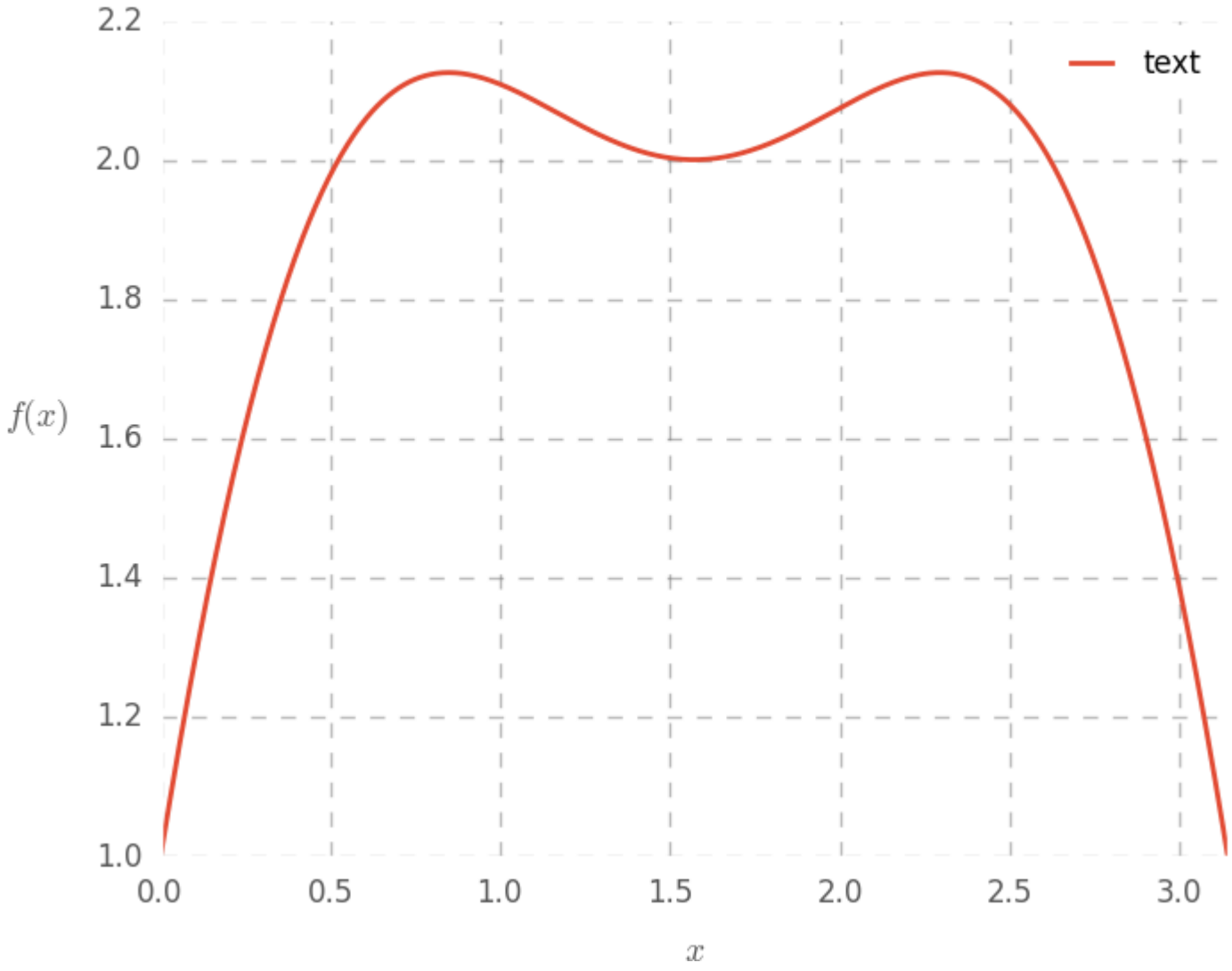
Personnalisation d'un tracé matplotlib

```
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()
```



Syntaxe impérative vs orientée objet

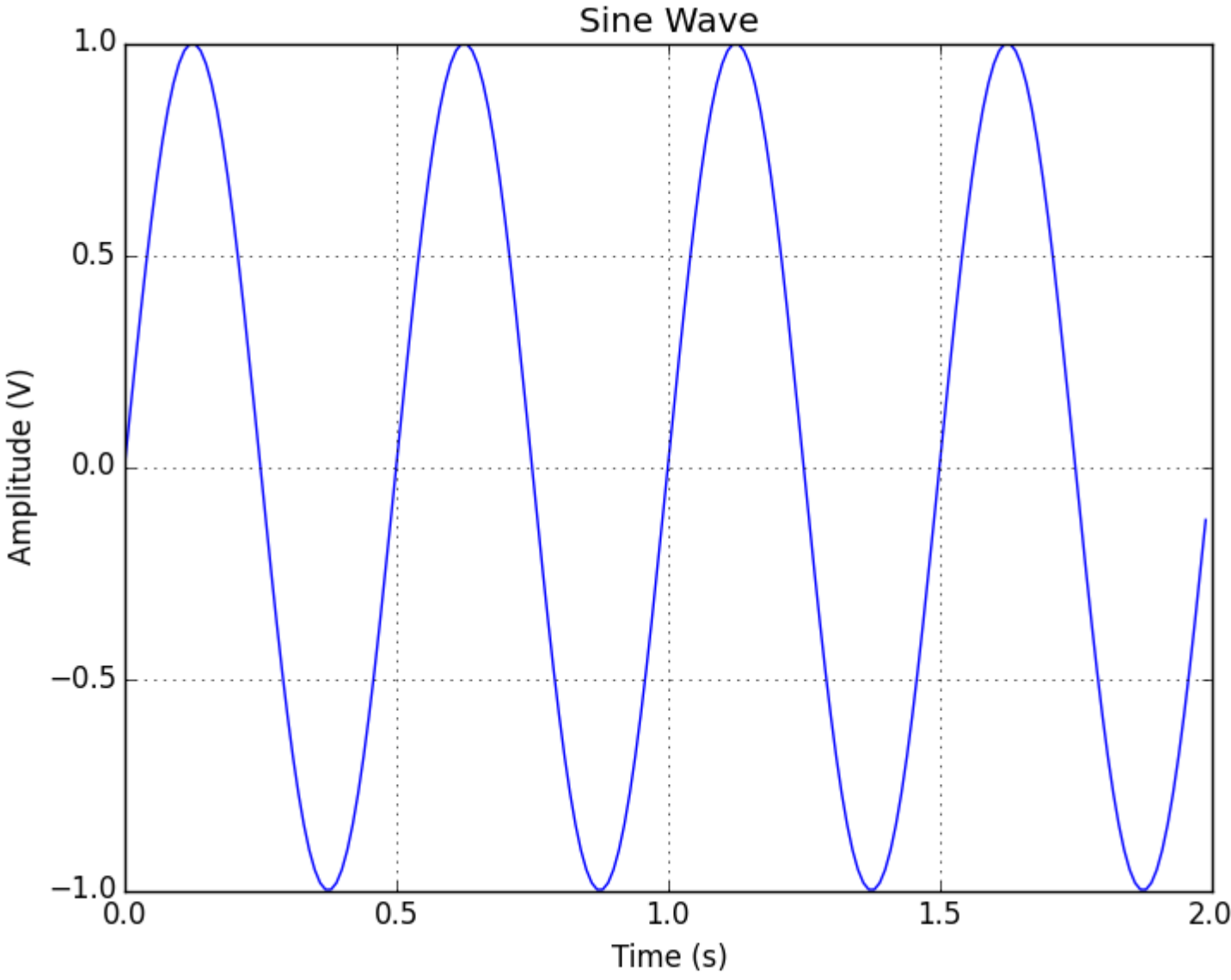
Matplotlib prend en charge à la fois la syntaxe orientée objet et la syntaxe impérative pour le traçage. La syntaxe impérative est intentionnellement conçue pour être très proche de la syntaxe Matlab.

La syntaxe impérative (parfois appelée syntaxe «state-machine») émet une chaîne de commandes qui agissent toutes sur la figure ou l'axe le plus récent (comme Matlab). La syntaxe orientée objet, d'autre part, agit explicitement sur les objets (figure, axe, etc.) d'intérêt. Un point clé du zen de Python stipule qu'explicite vaut mieux qu'implicite, donc la syntaxe orientée objet est plus pythonique. Cependant, la syntaxe impérative est pratique pour les nouveaux convertis de Matlab et pour l'écriture de petits scripts "jetables". Voici un exemple des deux styles différents.

```
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)
```

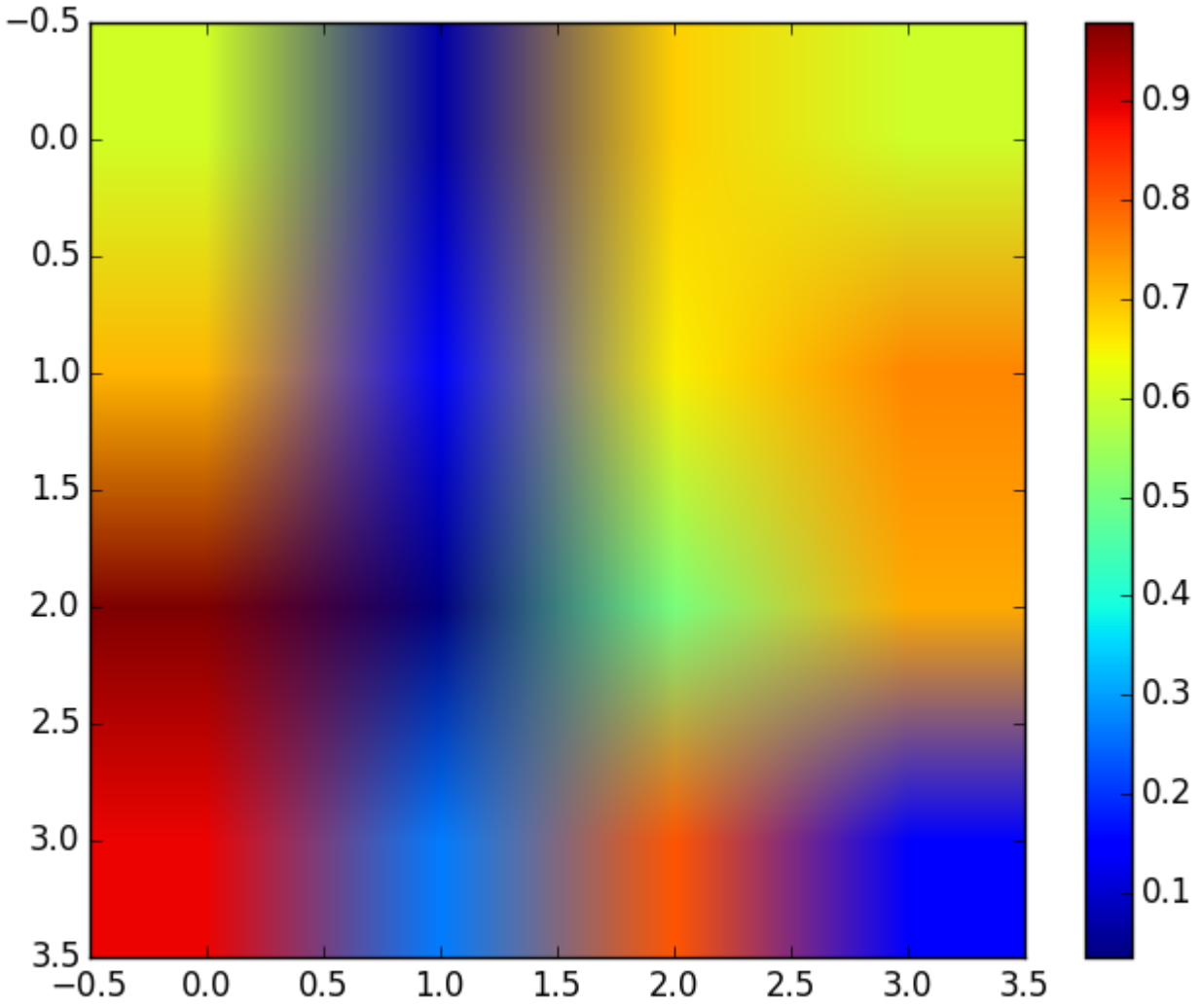
Les deux exemples produisent le même tracé que celui illustré ci-dessous.



Tableaux bidimensionnels (2D)

Affiche un tableau bidimensionnel (2D) sur les axes.

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

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



Lire Commencer avec matplotlib en ligne: https://riptutorial.com/fr/matplotlib/topic/881/commencer-avec-matplotlib

Chapitre 2: Animations et traçage interactif

Introduction

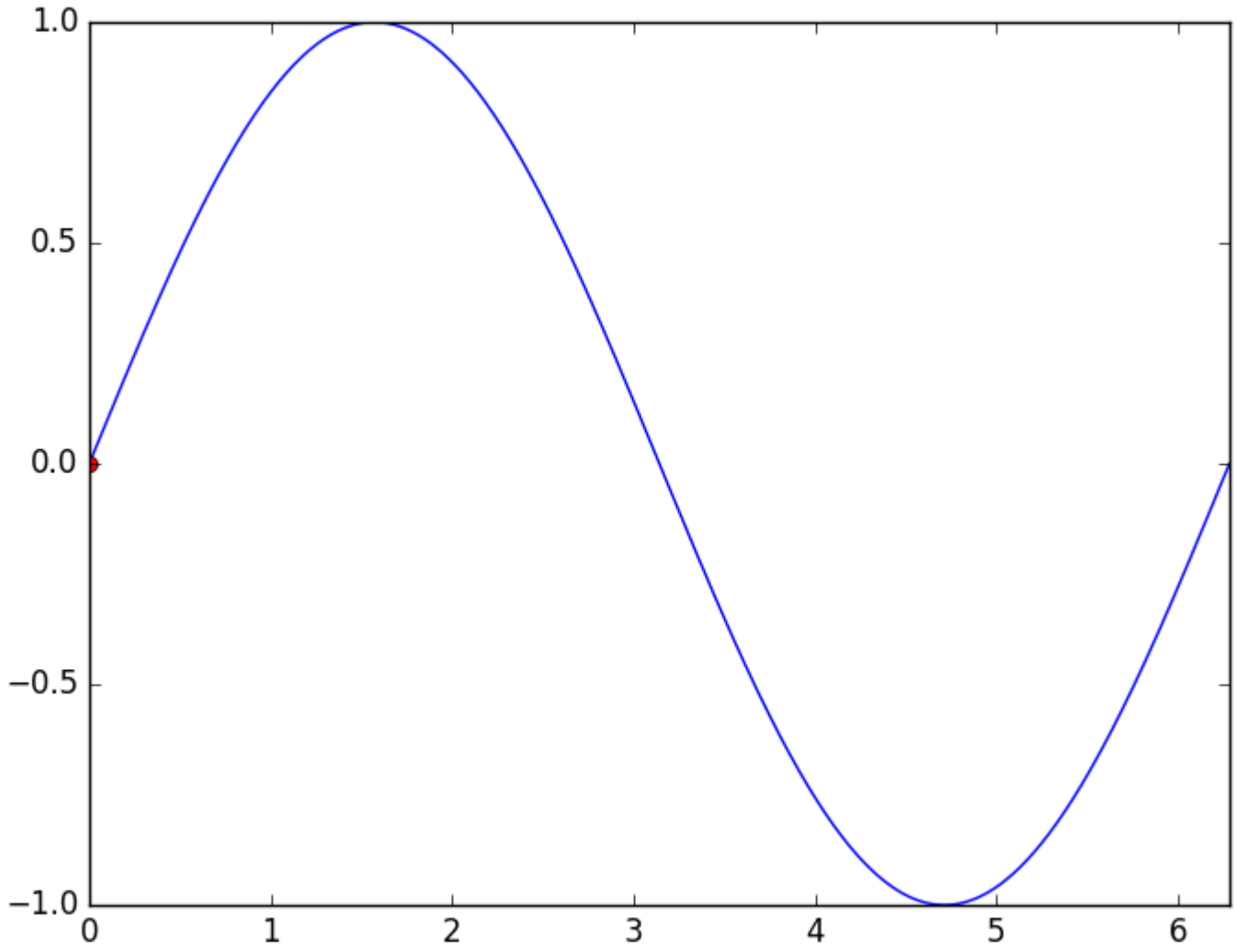
Avec python matplotlib, vous pouvez créer des graphiques animés correctement.

Examples

Animation de base avec FuncAnimation

Le package matplotlib.animation propose des classes pour créer des animations. `FuncAnimation` crée des animations en appelant plusieurs fois une fonction. Nous utilisons ici une fonction `animate()` qui modifie les coordonnées d'un point sur le graphique d'une fonction sinus.

```
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()
```



Enregistrer l'animation sur gif

Dans cet exemple , nous utilisons la `save` méthode pour enregistrer une `Animation` objet en utilisant 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)
```

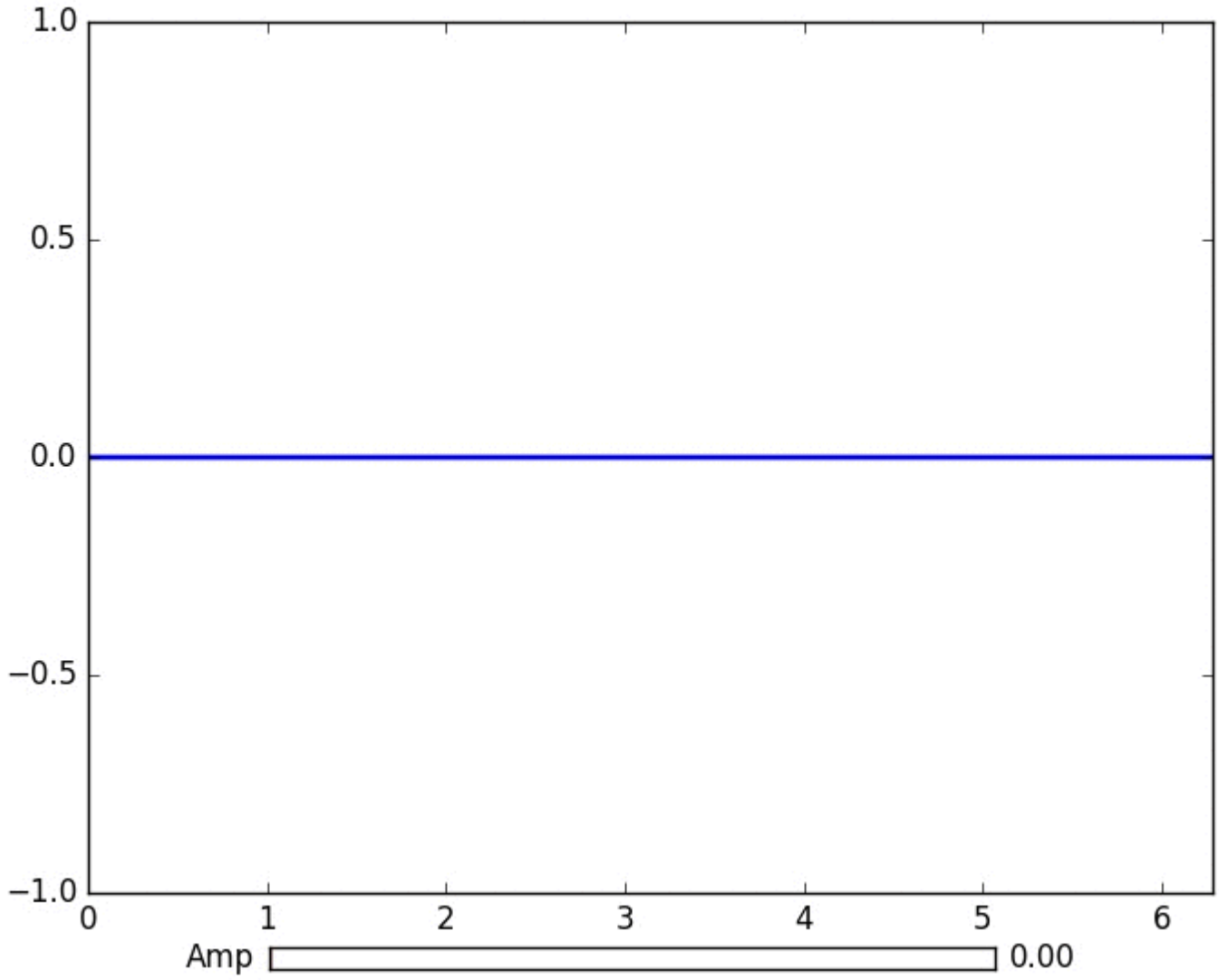
```
ax = plt.axis([0,TWOPI,-1,1])

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)
```

Contrôles interactifs avec matplotlib.widgets

Pour interagir avec les tracés, Matplotlib propose des widgets neutres pour l'interface graphique. Les widgets nécessitent un objet `matplotlib.axes.Axes` .

Voici une démo de widget de curseur qui indique l'amplitude d'une courbe sinusoïdale. La fonction de mise à jour est déclenchée par l'événement `on_changed()` du curseur.

```
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()
```



Autres widgets disponibles:

- AxesWidget
- Bouton
- Boutons de contrôle
- Le curseur
- EllipseSelector
- Lasso
- LassoSelector
- LockDraw
- MultiCursor
- RadioButtons
- RectangleSelector
- SpanSelector
- SubplotTool
- Des outils

Tracez des données en direct depuis un tuyau avec matplotlib

Cela peut être utile lorsque vous souhaitez visualiser les données entrantes en temps réel. Ces données pourraient, par exemple, provenir d'un microcontrôleur qui échantillonne en permanence un signal analogique.

Dans cet exemple, nous allons obtenir nos données à partir d'un canal nommé (également appelé fifo). Pour cet exemple, les données dans le tube doivent être des nombres séparés par des caractères de nouvelle ligne, mais vous pouvez l'adapter à votre goût.

Exemple de données:

```
100
123.5
1589
```

Plus d'informations sur les canaux nommés

Nous utiliserons également le type de données, provenant des collections de bibliothèques standard. Un objet deque fonctionne beaucoup comme une liste. Mais avec un objet deque, il est assez facile d'y ajouter quelque chose tout en gardant l'objet deque à une longueur fixe. Cela nous permet de garder l'axe x à une longueur fixe au lieu de toujours croître et écraser le graphique ensemble. Plus d'informations sur les objets deque

Choisir le bon backend est vital pour la performance. Vérifiez ce que les backends fonctionnent sur votre système d'exploitation et choisissez-en un rapide. Pour moi, seul qt4agg et le backend par défaut fonctionnaient, mais celui par défaut était trop lent. Plus d'informations sur les backends en matplotlib

Cet exemple est basé sur l'exemple matplotlib de traçage de données aléatoires .

Aucun des caractères de ce code ne doit être supprimé.

```
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 votre tracé commence à être retardé après un certain temps, essayez d'ajouter plus de données datalist.append, afin que davantage de lignes soient lues à chaque image. Ou choisissez un backend plus rapide si vous le pouvez.

Cela a fonctionné avec 150hz données d'un tuyau sur mon 1.7ghz i3 4005u.

Lire Animations et traçage interactif en ligne:
https://riptutorial.com/fr/matplotlib/topic/6983/animations-et-tracage-interactif

Chapitre 3: Boîtes à moustaches

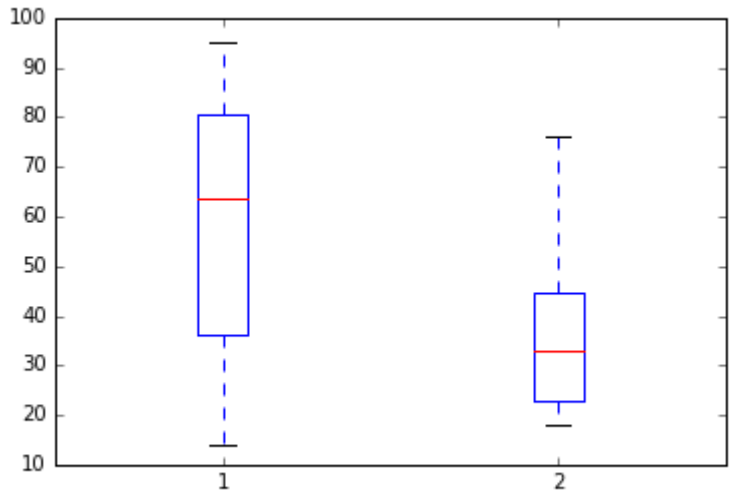
Examples

Boxplots de base

Les boîtes à moustaches sont des diagrammes descriptifs qui permettent de comparer la distribution de différentes séries de données. Ils sont *descriptifs* car ils montrent des mesures (par exemple la *médiane*) qui ne supposent pas une distribution de probabilité sous-jacente.

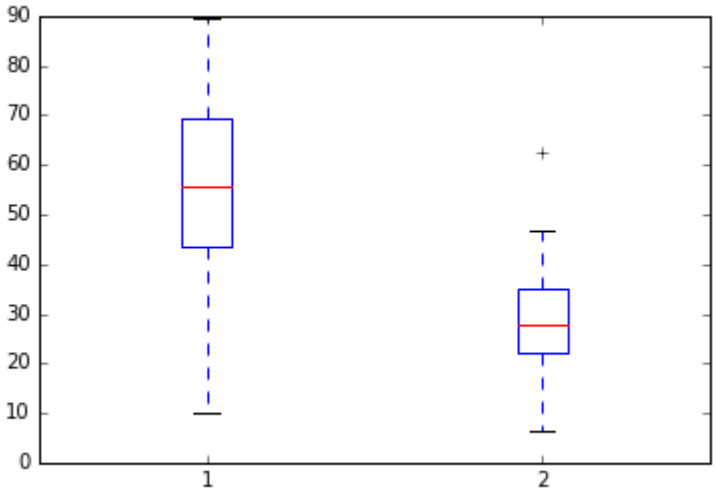
L'exemple le plus élémentaire d'une boîte à moustaches dans matplotlib peut être obtenu en passant simplement les données sous forme de liste de listes:

```
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 )
```



Cependant, il est courant d'utiliser des tableaux `numpy` comme paramètres pour les tracés, car ils résultent souvent de calculs antérieurs. Cela peut être fait comme suit:

```
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 )
```



Lire Boîtes à moustaches en ligne: https://riptutorial.com/fr/matplotlib/topic/6086/boites-a-moustaches

Chapitre 4: Boîtes à moustaches

Examples

Fonction Boxplot

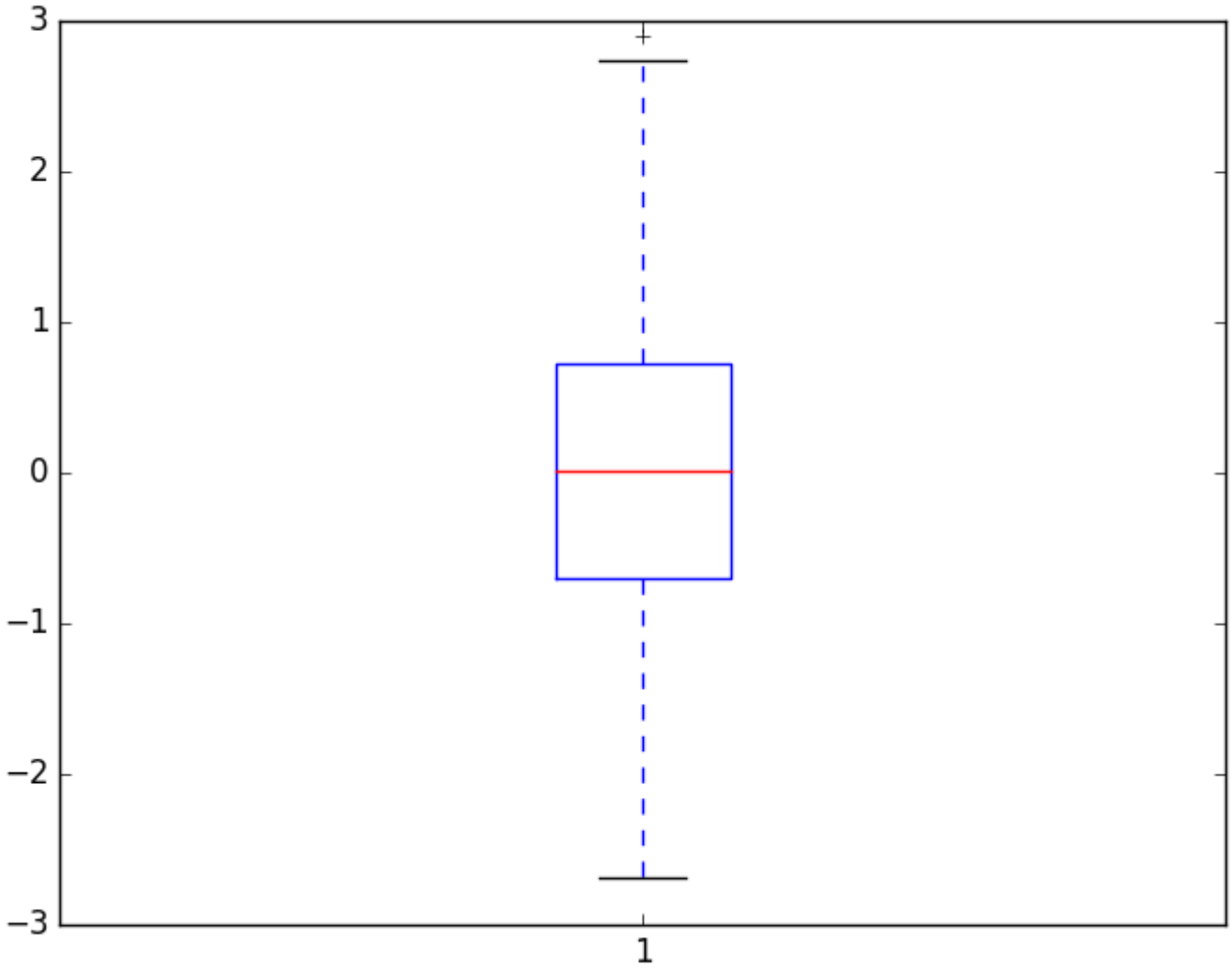
Matplotlib a sa propre implémentation de boxplot . Les aspects pertinents de cette fonction sont les suivants: par défaut, le plotplet affiche la médiane (percentile 50%) avec une ligne rouge. La case représente Q1 et Q3 (percentiles 25 et 75) et les moustaches donnent une idée de la gamme des données (possiblement à Q1 - 1,5 *IQR; Q3 + 1,5* IQR; IQR étant l'intervalle interquartile, mais cela manque de confirmation). Notez également que les échantillons situés au-delà de cette plage sont affichés sous forme de marqueurs (ceux-ci sont nommés «flyers»).

> **Remarque:** toutes les implémentations de *boxplot ne* suivent pas les mêmes règles. Peut-être le diagramme de boîte à moustaches le plus commun utilise les moustaches pour représenter le minimum et le maximum (rendant les prospectus inexistants). Notez également que cette parcelle est parfois appelée *parcelle à boîte et à moustaches et diagramme à boîte et à moustaches* .

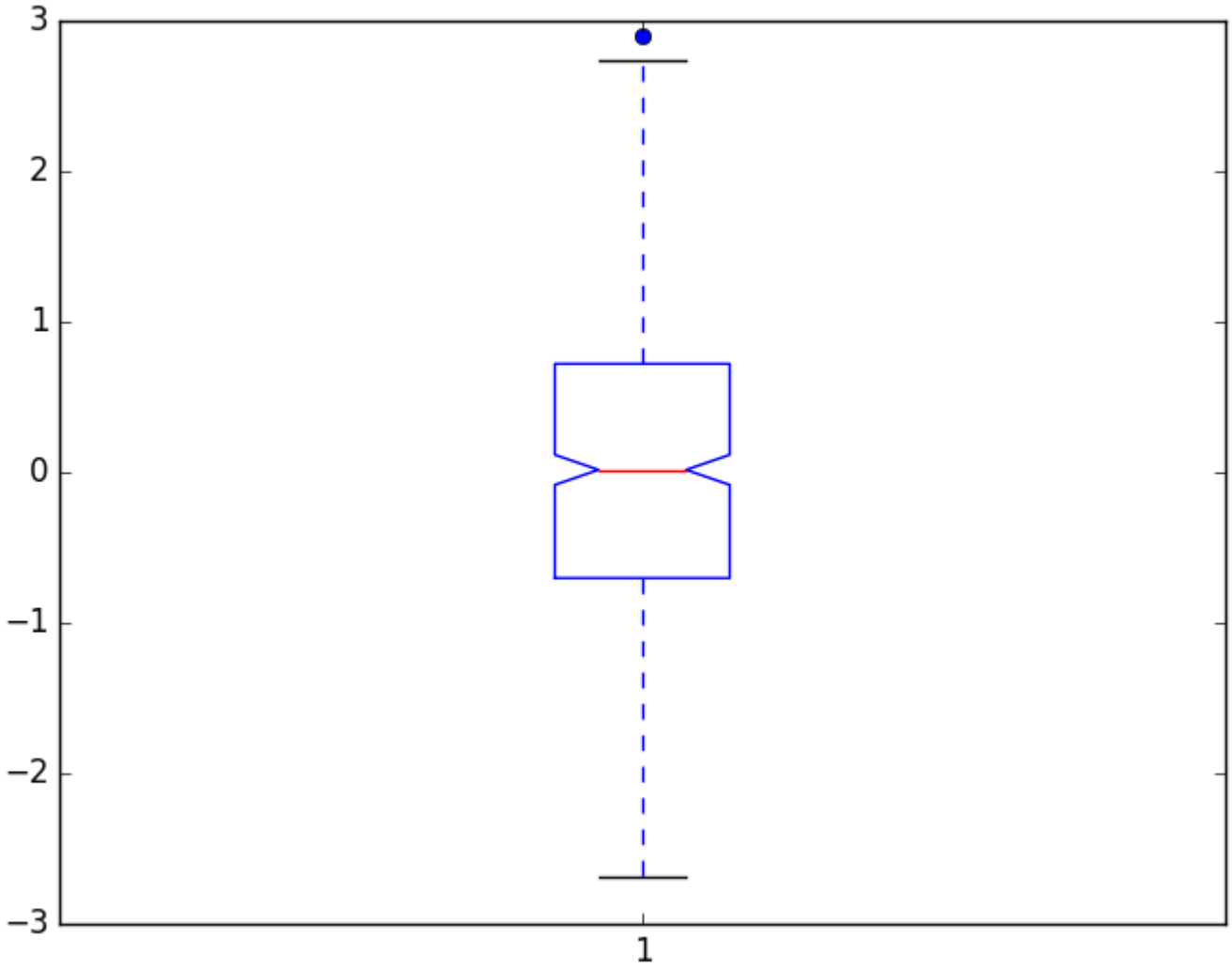
La recette suivante montre certaines des choses que vous pouvez faire avec l'implémentation actuelle de matplotlib de boxplot:

```
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()
```

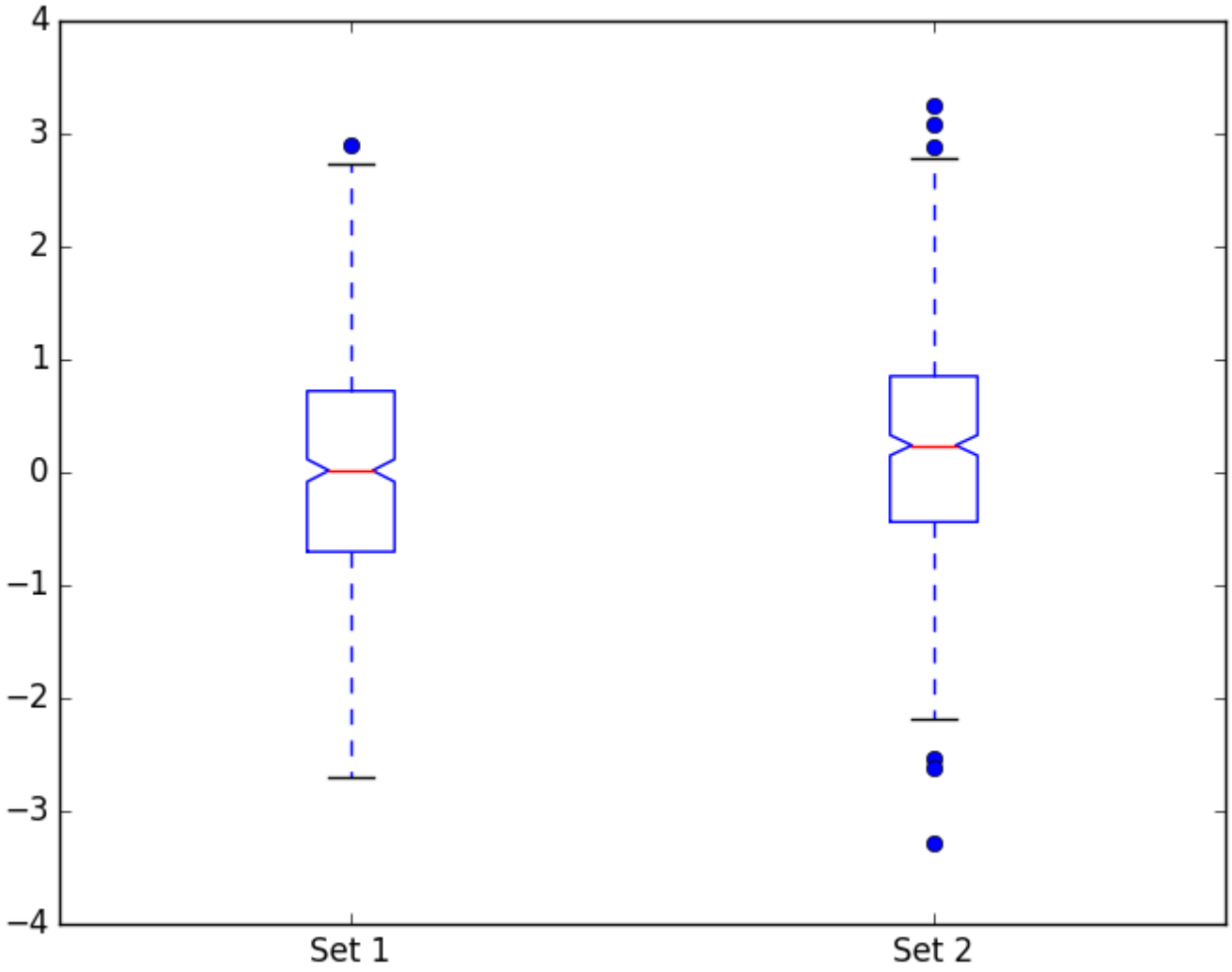
Cela se traduit par les parcelles suivantes:



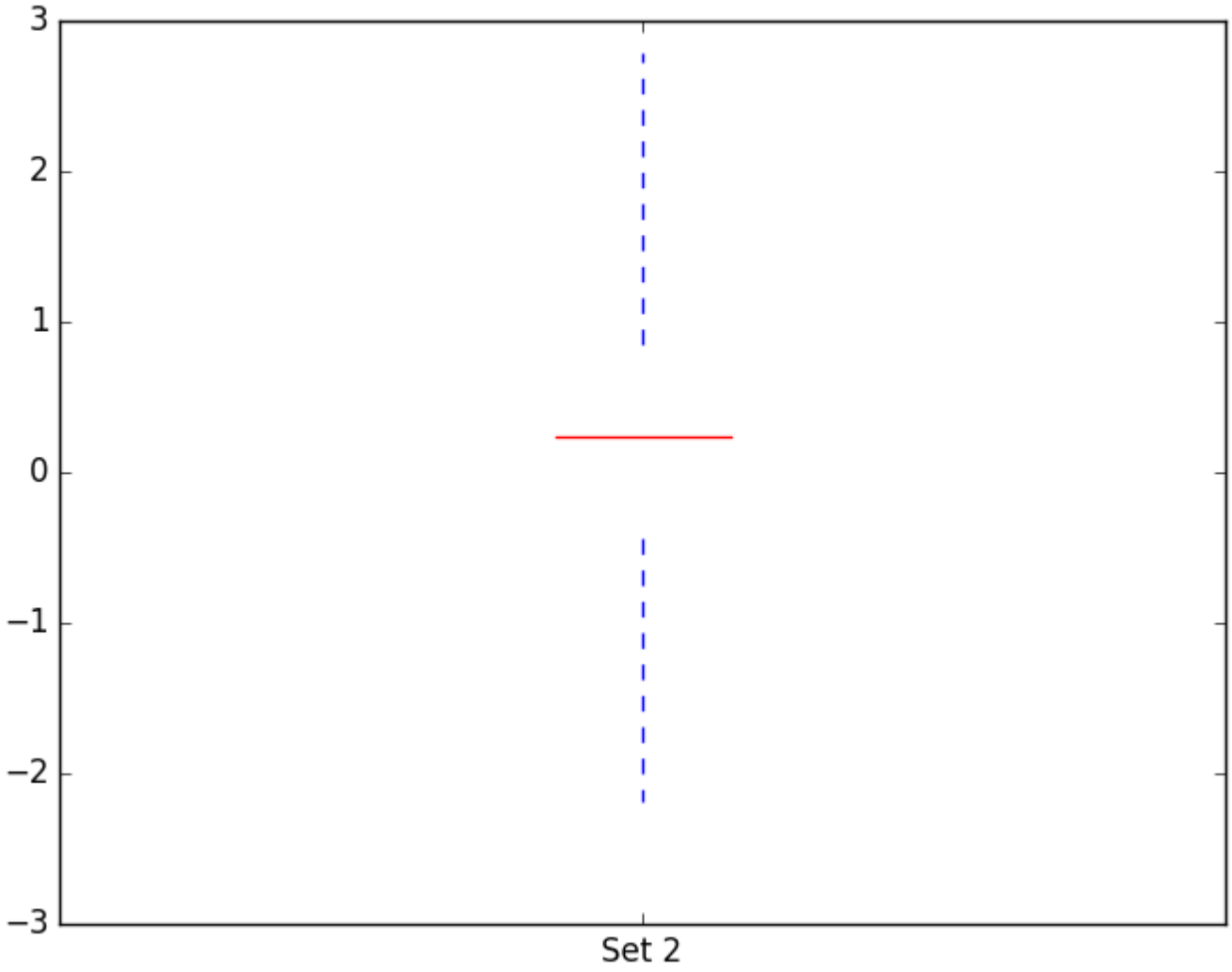
1. *Matplotlib boxplot par défaut*



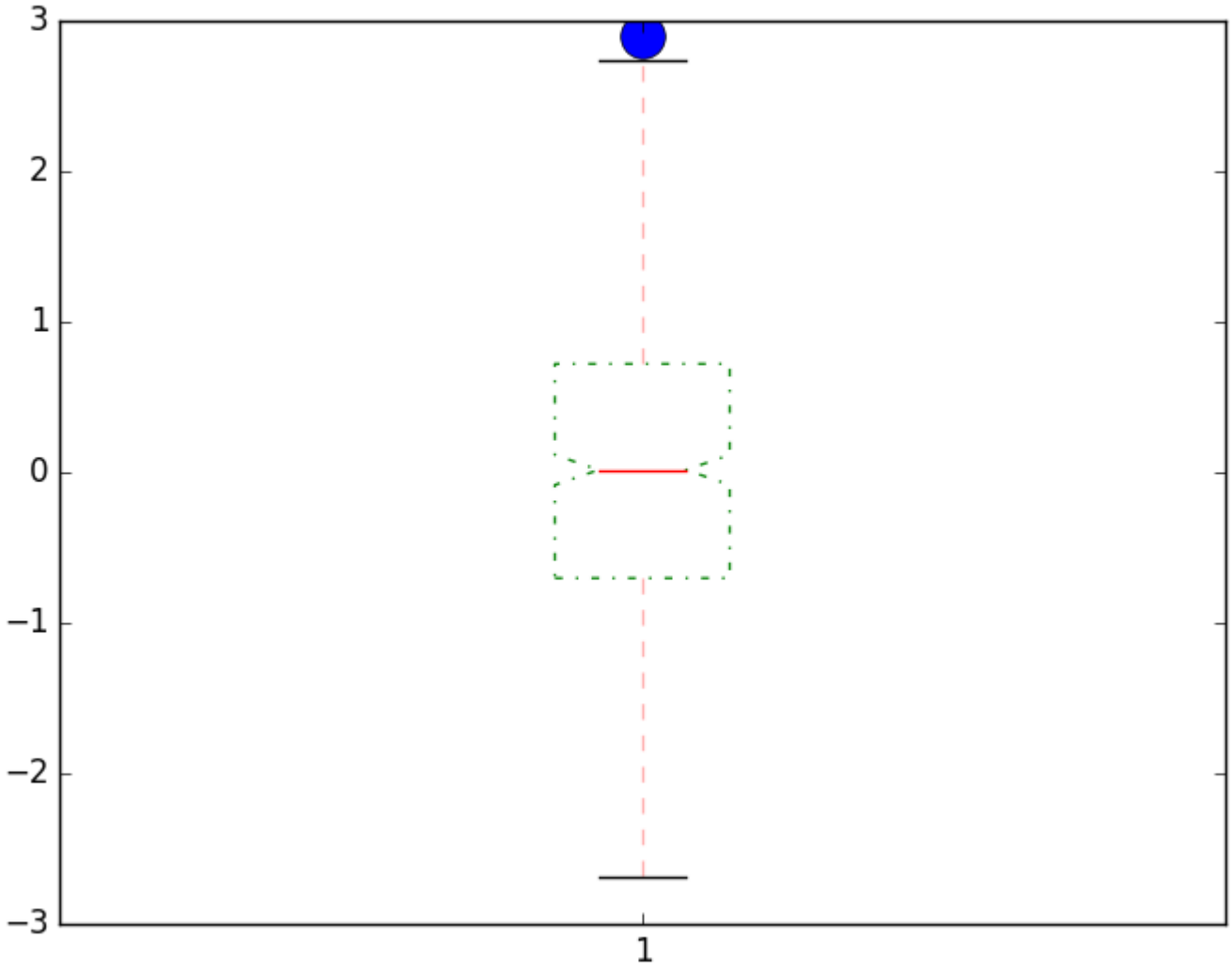
2. *Modification de certaines fonctionnalités de la boîte à moustaches en utilisant des arguments de fonction*



3. *Multiple boxplot dans la même fenêtre de tracé*



4. *Cacher certaines fonctionnalités du boxplot*



5. *Personnalisation avancée d'une boîte à moustaches en utilisant des accessoires*

Si vous avez l'intention de faire une personnalisation avancée de votre boîte à moustaches, vous devez savoir que les dictionnaires d' accessoires que vous créez (par exemple):

```
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()
```

... référez-vous principalement (sinon tous) aux objets Line2D . Cela signifie que seuls les arguments disponibles dans cette classe sont modifiables. Vous remarquerez l'existence de mots-clés tels que `whiskerprops` , `boxprops` , `flierprops` et `capprops` . Ce sont les éléments dont vous avez besoin pour fournir un dictionnaire d'accessoires afin de le personnaliser davantage.

> REMARQUE: Une personnalisation supplémentaire de la boîte à moustaches en utilisant cette implémentation peut s'avérer difficile. Dans certains cas, l'utilisation

d'autres éléments matplotlib, tels que les correctifs pour créer un propre plotbox, peut être avantageuse (changements considérables apportés à l'élément box, par exemple).

Lire Boîtes à moustaches en ligne: https://riptutorial.com/fr/matplotlib/topic/6368/boites-a-moustaches

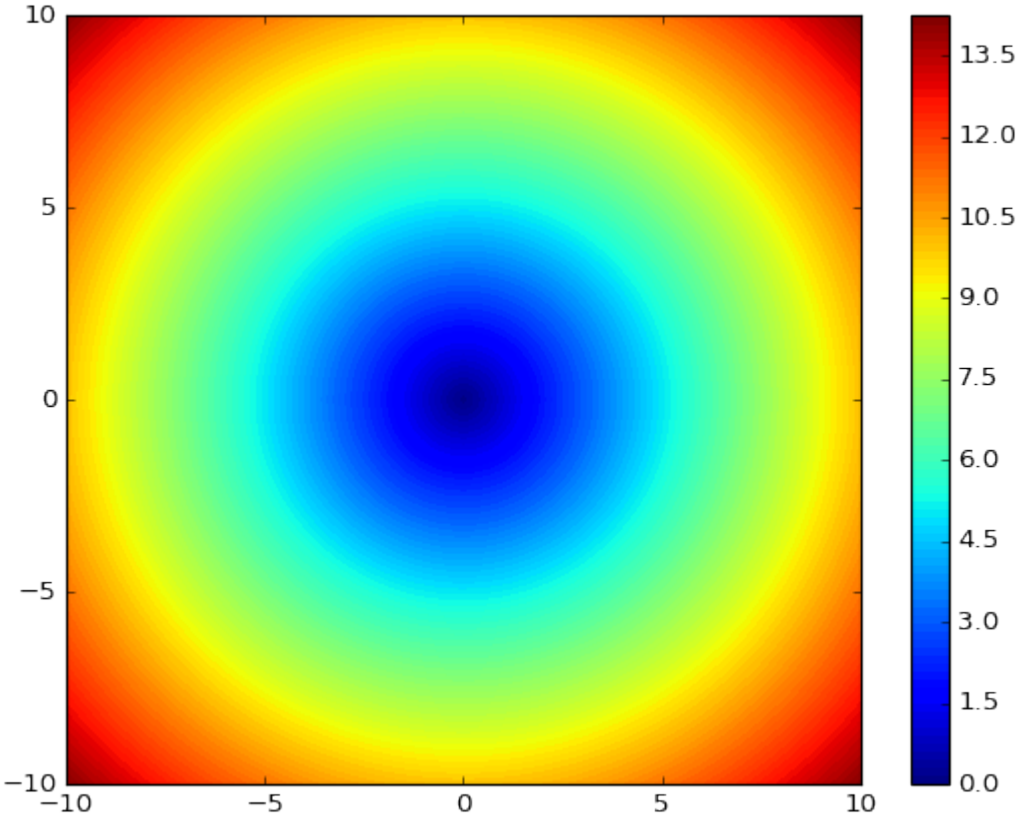
Chapitre 5: Cartes de contour

Examples

Tracé de contour simple rempli

```
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)
```

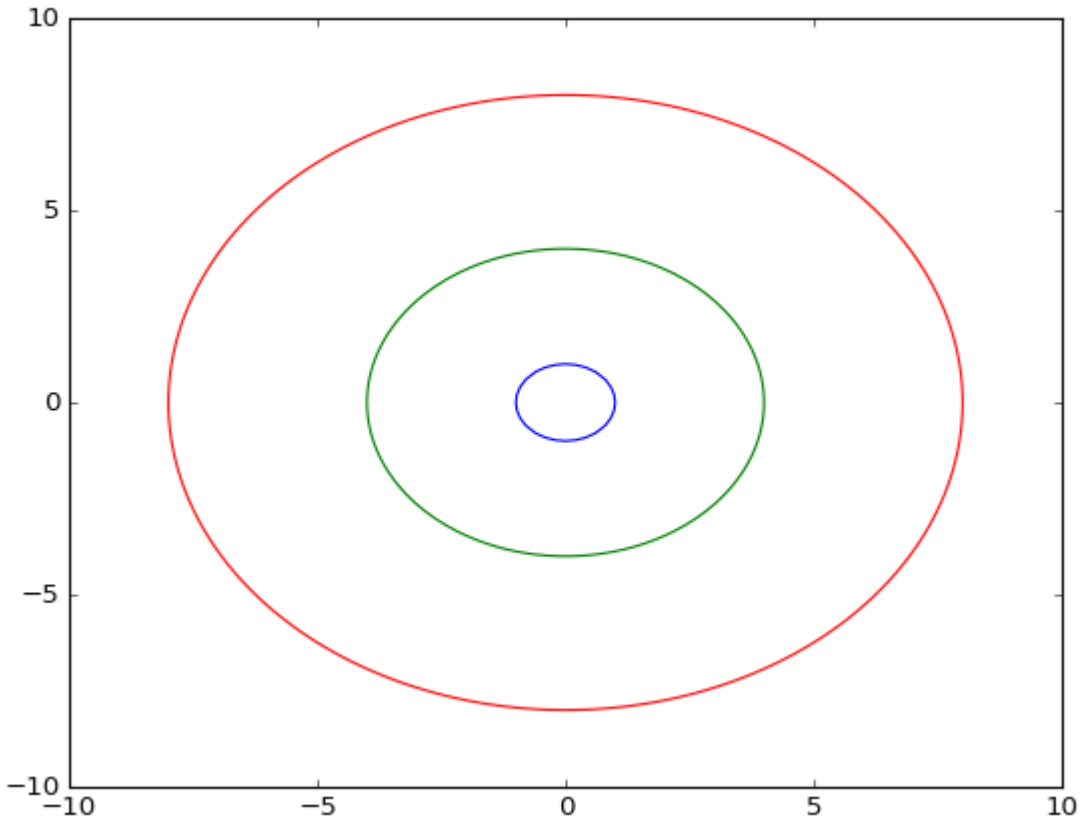
Résultat:



Tracé de contour 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'])
```

Résultat:



Lire Cartes de contour en ligne: https://riptutorial.com/fr/matplotlib/topic/8644/cartes-de-contour

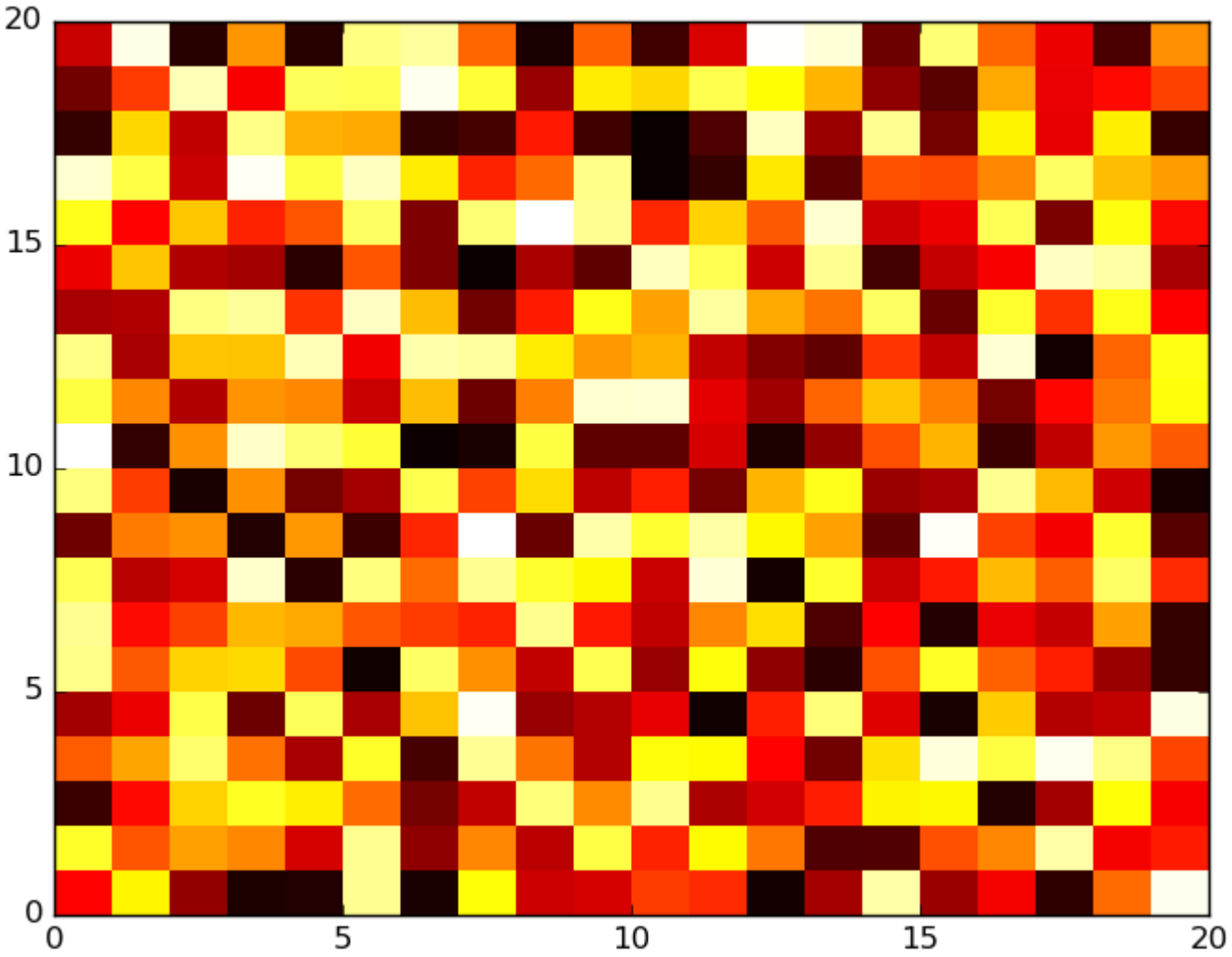
Chapitre 6: Colormaps

Examples

Utilisation de base

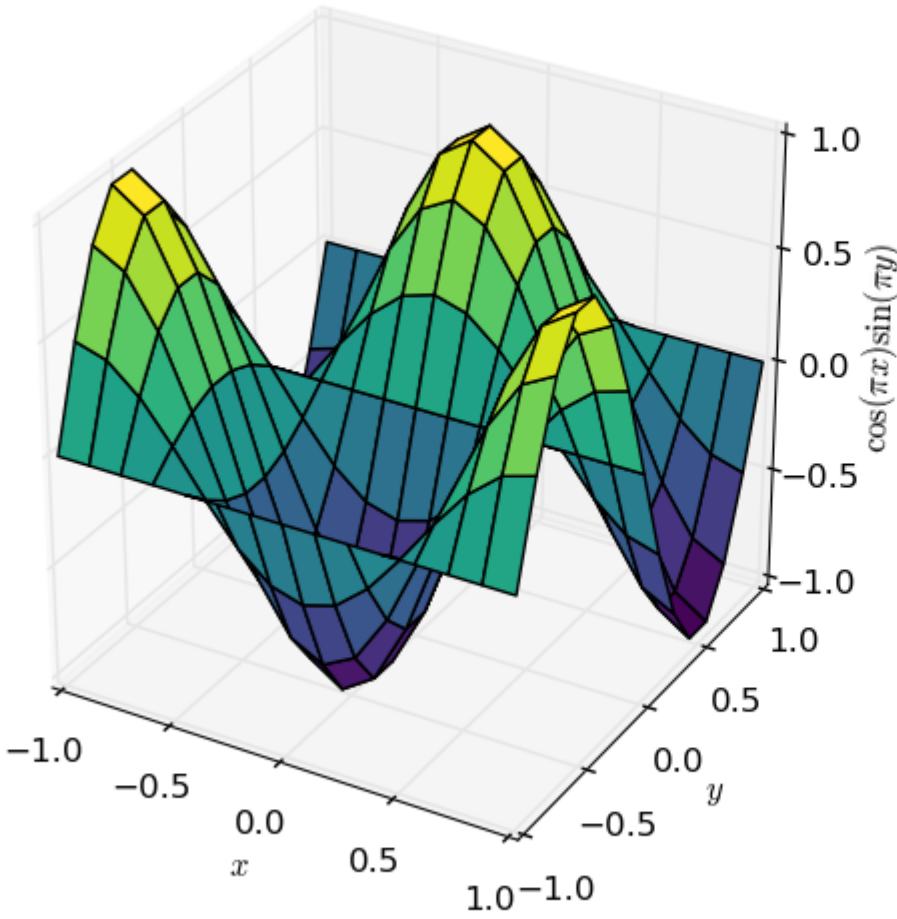
L' utilisation colormaps intégré est aussi simple que passer le nom de la palette de couleurs nécessaire (comme indiqué dans la référence de colormaps) à la fonction de traçage (comme `pcolormesh` ou `contourf`) qui l' attend, le plus souvent sous la forme d'un `cmap` argument mot - clé:

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

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



Les Colormaps sont particulièrement utiles pour visualiser des données tridimensionnelles sur des tracés bidimensionnels, mais une bonne palette de couleurs peut également rendre un tracé tridimensionnel correct plus clair:

```
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()
```



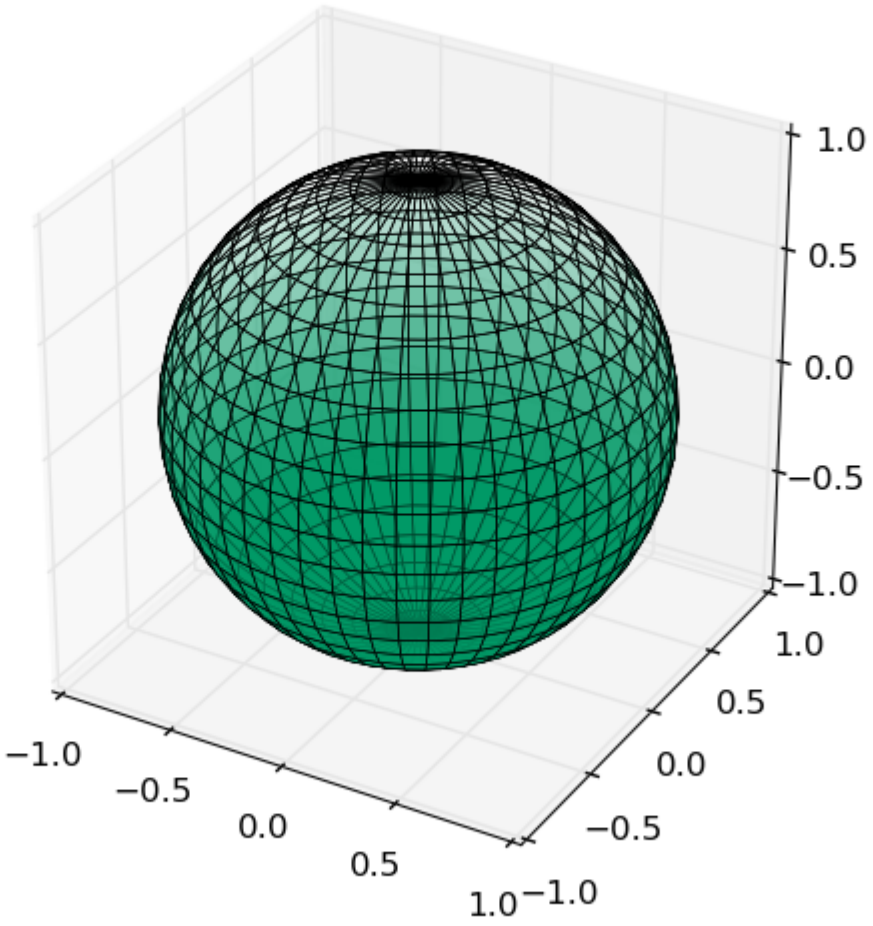
Utilisation de couleurs personnalisées

Outre les colonnes de couleurs intégrées définies dans la référence de couleurs (et leurs mappes inversées, avec `'_r'` ajouté à leur nom), des couleurs personnalisées peuvent également être définies. La clé est le module `matplotlib.cm` .

L'exemple ci-dessous définit une palette de `cm.register_cmap` très simple à l'aide de `cm.register_cmap` , contenant une seule couleur, l'opacité (valeur alpha) de la couleur interpolant entièrement opaque et totalement transparente dans la plage de données. Notez que les lignes importantes du point de vue de la palette de couleurs sont l'importation de `cm` , l'appel à `register_cmap` et le passage de la palette de `plot_surface` à `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()
```



Dans des scénarios plus compliqués, on peut définir une liste de valeurs R / G / B (/ A) dans lesquelles matplotlib interpole linéairement afin de déterminer les couleurs utilisées dans les tracés correspondants.

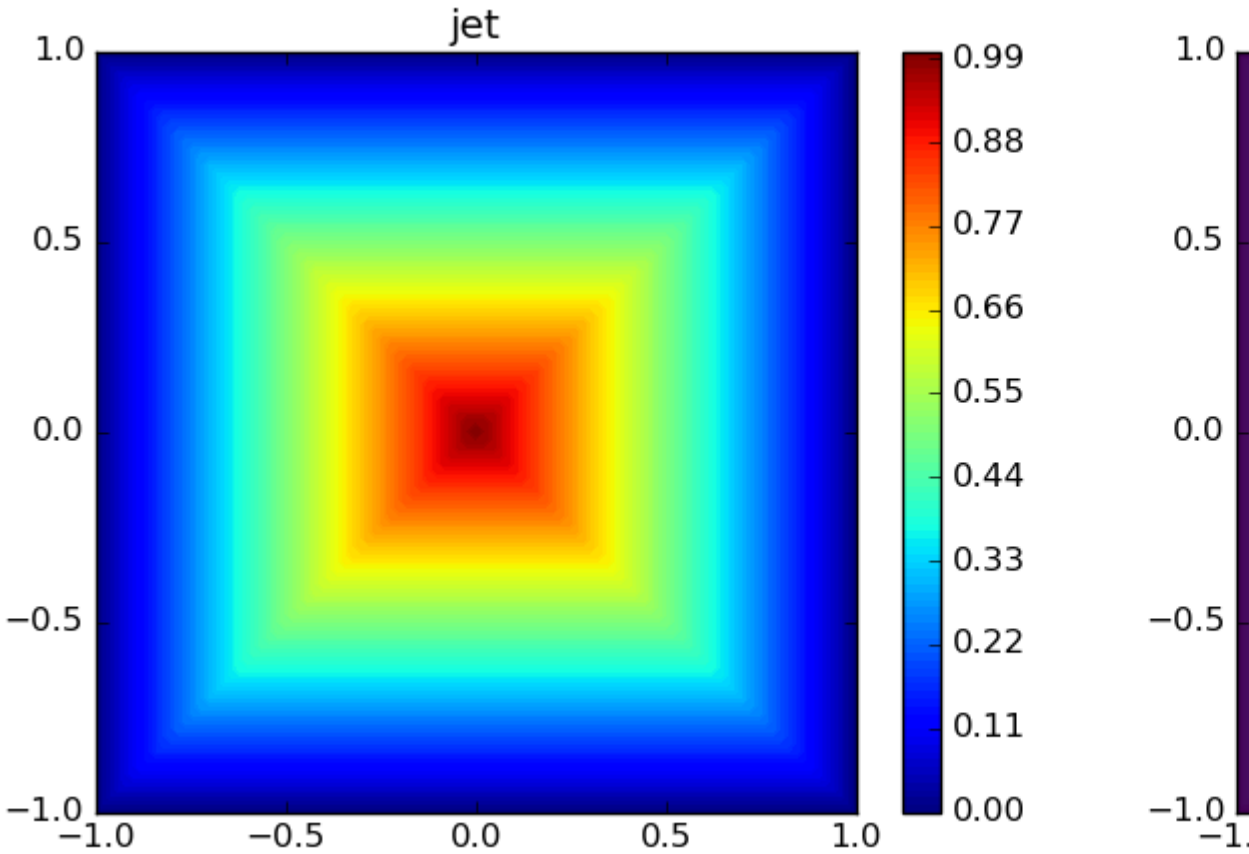
Colormaps perceptuellement uniformes

La carte de couleurs originale par défaut de MATLAB (remplacée dans la version R2014b) appelée `jet` est omniprésente en raison de son contraste élevé et de sa familiarité (et était la valeur par défaut de matplotlib pour des raisons de compatibilité). En dépit de sa popularité, les cartes de couleurs traditionnelles présentent souvent des lacunes en matière de représentation précise des données. Le changement perçu dans ces couleurs ne correspond pas aux changements de données; et une conversion de la palette de couleurs en niveaux de gris (en imprimant, par exemple, une image en utilisant une imprimante noir et blanc) peut entraîner une perte d'informations.

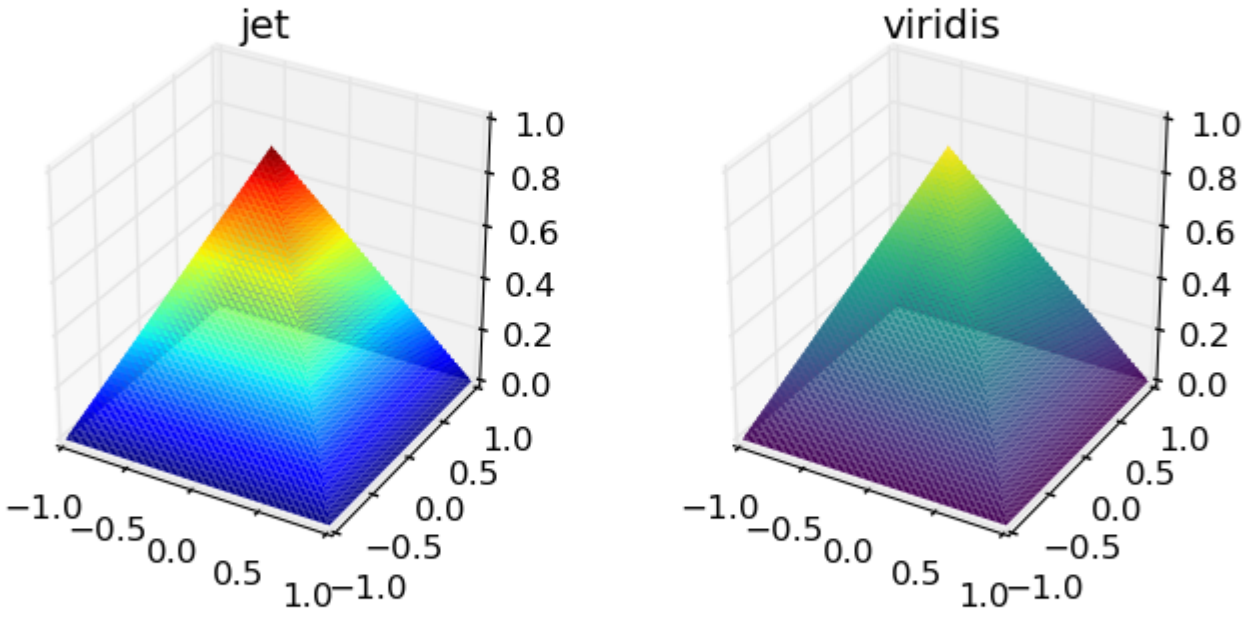
Des palettes de couleurs perceptuellement uniformes ont été introduites pour rendre la visualisation des données aussi précise et accessible que possible. Matplotlib a introduit quatre nouvelles couleurs personnalisées dans la version 1.5, dont l'une (nommée `viridis`) par défaut

depuis la version 2.0. Ces quatre couleurs (`viridis` , `inferno` , `plasma` et `magma`) sont toutes optimales du point de vue de la perception et devraient être utilisées par défaut pour la visualisation des données, à moins de bonnes raisons de ne pas le faire. Ces couleurs permettent d'introduire le moins de biais possible (en ne créant pas de fonctionnalités là où il n'y en a pas), et conviennent à un public dont la perception des couleurs est réduite.

À titre d'exemple pour les données de distorsion visuelle, considérez les deux tracés de vue supérieure d'objets pyramidaux suivants:



Lequel des deux est une véritable pyramide? La réponse est bien sûr que les deux sont, mais ceci est loin d'être évident à partir de l'intrigue utilisant la palette de `jet` :



Cette caractéristique est au cœur de l'uniformité perceptuelle.

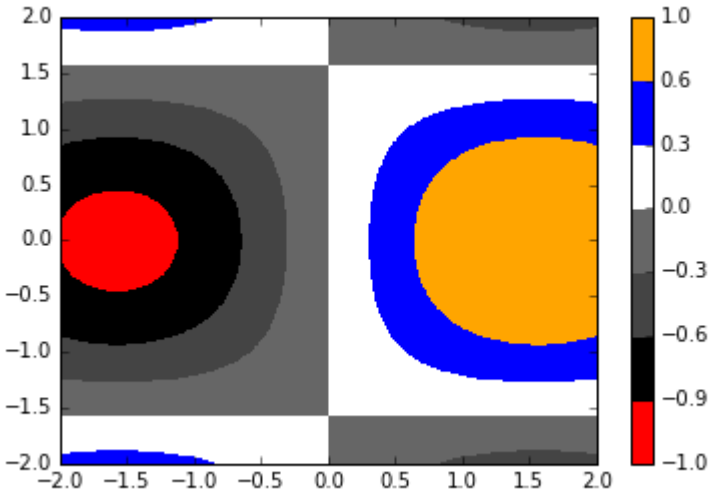
Palette de couleurs discrète personnalisée

Si vous avez des plages prédéfinies et que vous souhaitez utiliser des couleurs spécifiques pour ces plages, vous pouvez déclarer une palette de couleurs personnalisée. Par exemple:

```
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()
```

Produit



La couleur *i* sera utilisée pour les valeurs comprises entre la limite *i* et *i + 1* . Les couleurs peuvent être spécifiées par des noms (`'red'` , `'green'`), des codes HTML (`'#ffaa44'` , `'#441188'`) ou des `'#441188'` RVB (`(0.2, 0.9, 0.45)`).

Lire Colormaps en ligne: https://riptutorial.com/fr/matplotlib/topic/3385/colormaps

Chapitre 7: Fermer une fenêtre de figure

Syntaxe

- plt.close () # ferme la figure active en cours
- plt.close (fig) # ferme la figure avec la poignée 'fig'
- plt.close (num) # ferme le numéro de chiffre 'num'
- plt.close (name) # ferme la figure avec l'étiquette 'name'
- plt.close ('all') # ferme tous les chiffres

Examples

Fermeture de la figure active actuelle à l'aide de pyplot

L'interface pyplot à `matplotlib` pourrait être le moyen le plus simple de fermer une figure.

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

Fermer un chiffre en utilisant plt.close ()

Une figure spécifique peut être fermée en gardant sa poignée

```
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
```

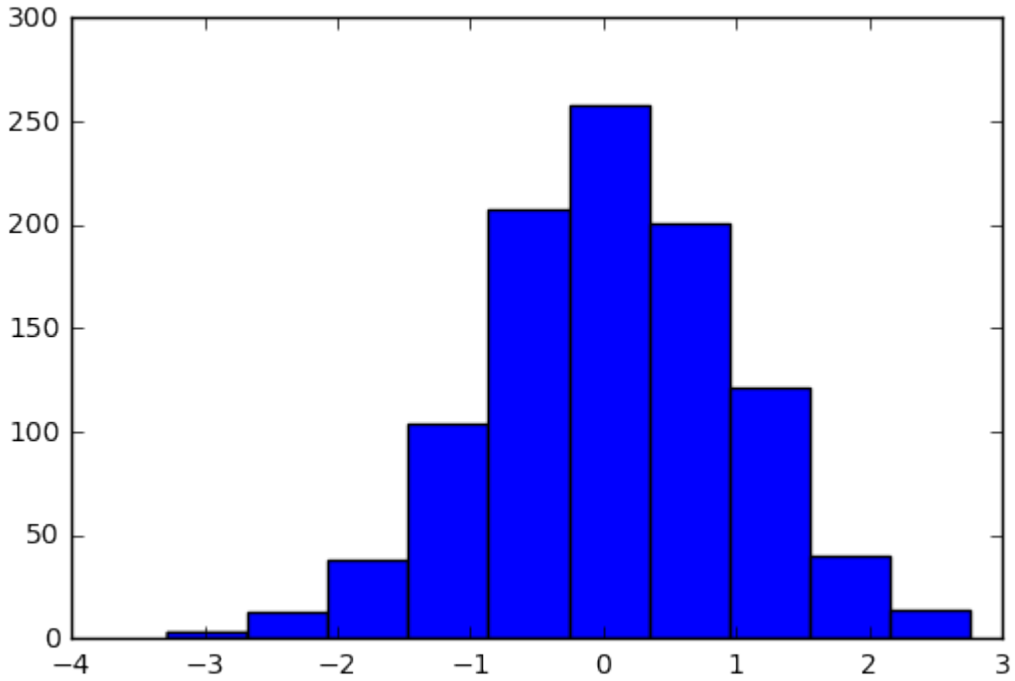
Lire Fermer une fenêtre de figure en ligne: https://riptutorial.com/fr/matplotlib/topic/6628/fermer-une-fenetre-de-figure

Chapitre 8: Histogramme

Examples

Histogramme 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()
```



Lire Histogramme en ligne: https://riptutorial.com/fr/matplotlib/topic/7329/histogramme

Chapitre 9: Intégration avec TeX / LaTeX

Remarques

- La prise en charge de LaTeX par Matplotlib nécessite une installation LaTeX opérationnelle, dvipng (qui peut être incluse avec votre installation LaTeX) et Ghostscript (GPL Ghostscript 8.60 ou version ultérieure est recommandé).
- Le support pgf de Matplotlib nécessite une installation récente de LaTeX incluant les packages TikZ / PGF (tels que TeXLive), de préférence avec XeLaTeX ou LuaLaTeX.

Examples

Insertion de formules TeX dans les parcelles

Les formules TeX peuvent être insérées dans le tracé à l'aide de la fonction `rc`

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

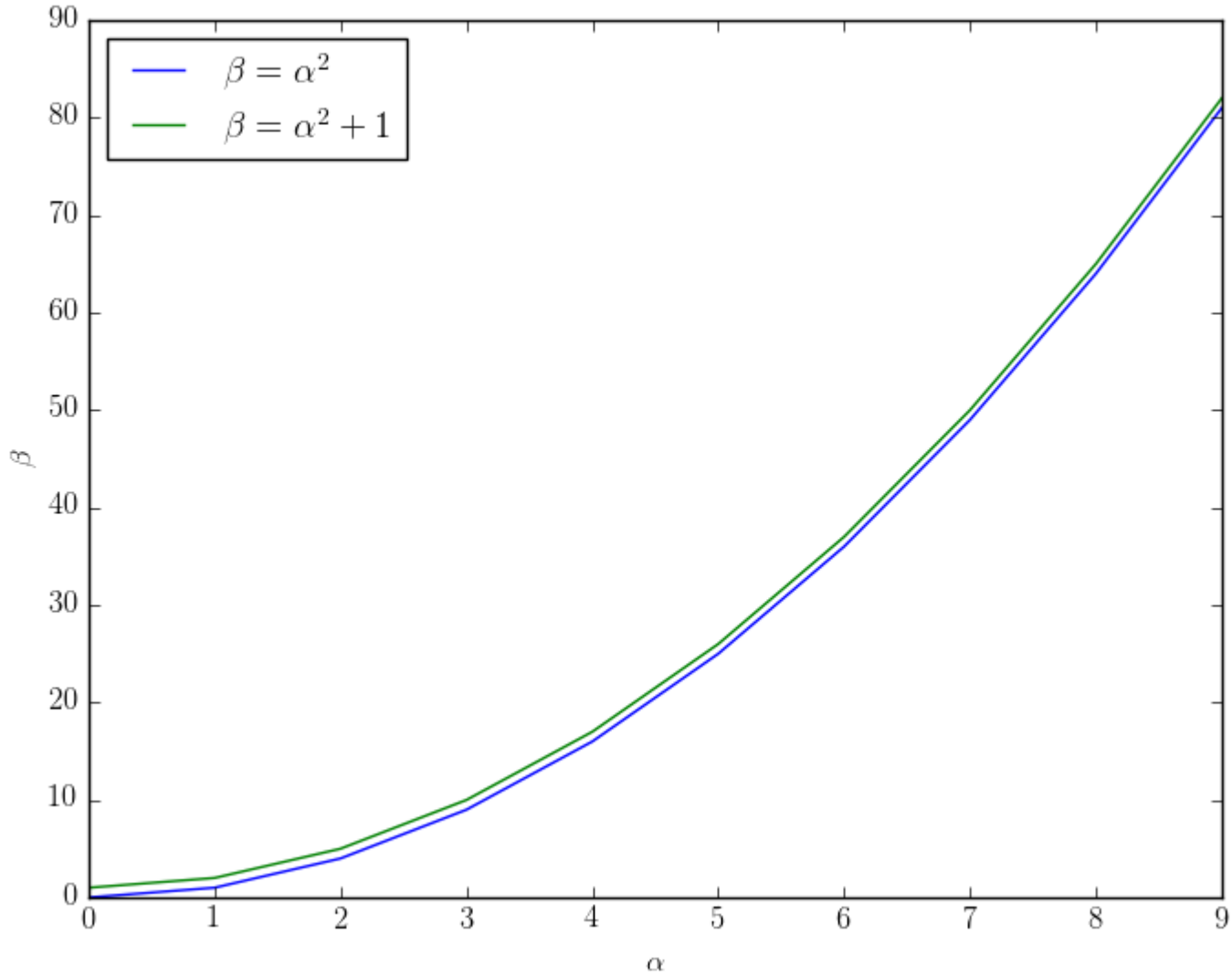
ou accéder aux `rcParams` :

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

TeX utilise la barre oblique inverse `\` pour les commandes et les symboles, ce qui peut entrer en conflit avec des caractères spéciaux dans les chaînes Python. Pour utiliser des barres obliques inverses littérales dans une chaîne Python, elles doivent soit être échappées, soit incorporées dans une chaîne brute:

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

La parcelle suivante



peut être produit par le code

```
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()
```

Les équations affichées (telles que `$$...$$` ou `\begin{equation}...\end{equation}`) ne sont pas prises en charge. Néanmoins, le style mathématique affiché est possible avec `\displaystyle` .

Pour charger des paquets latex, utilisez l'argument `tex.latex.preamble` :

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

Notez cependant l'avertissement dans l' exemple de fichier matplotlibrc :

```
#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
```

Enregistrement et exportation de tracés utilisant TeX

Afin d'inclure les tracés créés avec matplotlib dans les documents TeX, ils doivent être enregistrés sous `pdf` fichiers `pdf` ou `eps` . De cette manière, tout texte du tracé (y compris les formules TeX) est rendu sous forme de texte dans le document 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
```

Les diagrammes dans matplotlib peuvent être exportés vers le code TeX en utilisant le `pgf` macro `pgf` pour afficher les graphiques.

```
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')
```

Utilisez la commande `rc` pour changer le moteur TeX utilisé

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

Pour inclure la figure `.pgf` , écrivez dans votre document LaTeX

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

Lire Intégration avec TeX / LaTeX en ligne:
https://riptutorial.com/fr/matplotlib/topic/2962/integration-avec-tex---latex

Chapitre 10: Légendes

Examples

Légende Simple

Supposons que vous ayez plusieurs lignes dans un même tracé, chacune de couleur différente, et que vous souhaitiez créer une légende pour indiquer ce que chaque ligne représente. Vous pouvez le faire en passant une étiquette à chacune des lignes lorsque vous appelez `plot()` , par exemple, la ligne suivante sera intitulée *"My Line 1"* .

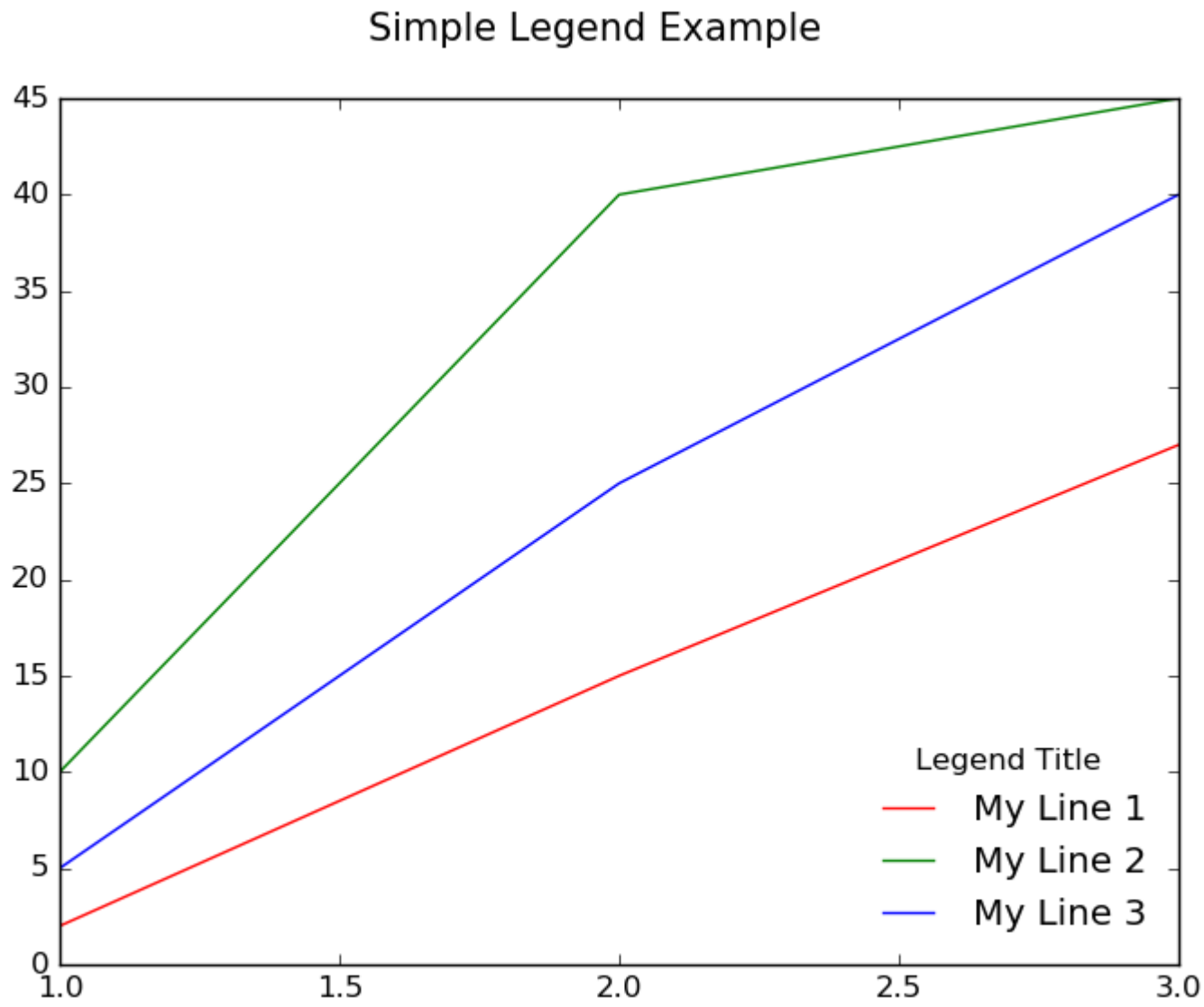
```
ax.plot(x, y1, color="red", label="My Line 1")
```

Ceci spécifie le texte qui apparaîtra dans la légende de cette ligne. Maintenant, pour rendre visible la légende, on peut appeler `ax.legend()`

Par défaut, il créera une légende dans une case située dans le coin supérieur droit du tracé. Vous pouvez transmettre des arguments à `legend()` pour le personnaliser. Par exemple, nous pouvons le positionner dans le coin inférieur droit, sans encadrer un cadre, et créer un titre pour la légende en appelant ce qui suit:

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

Voici un exemple:

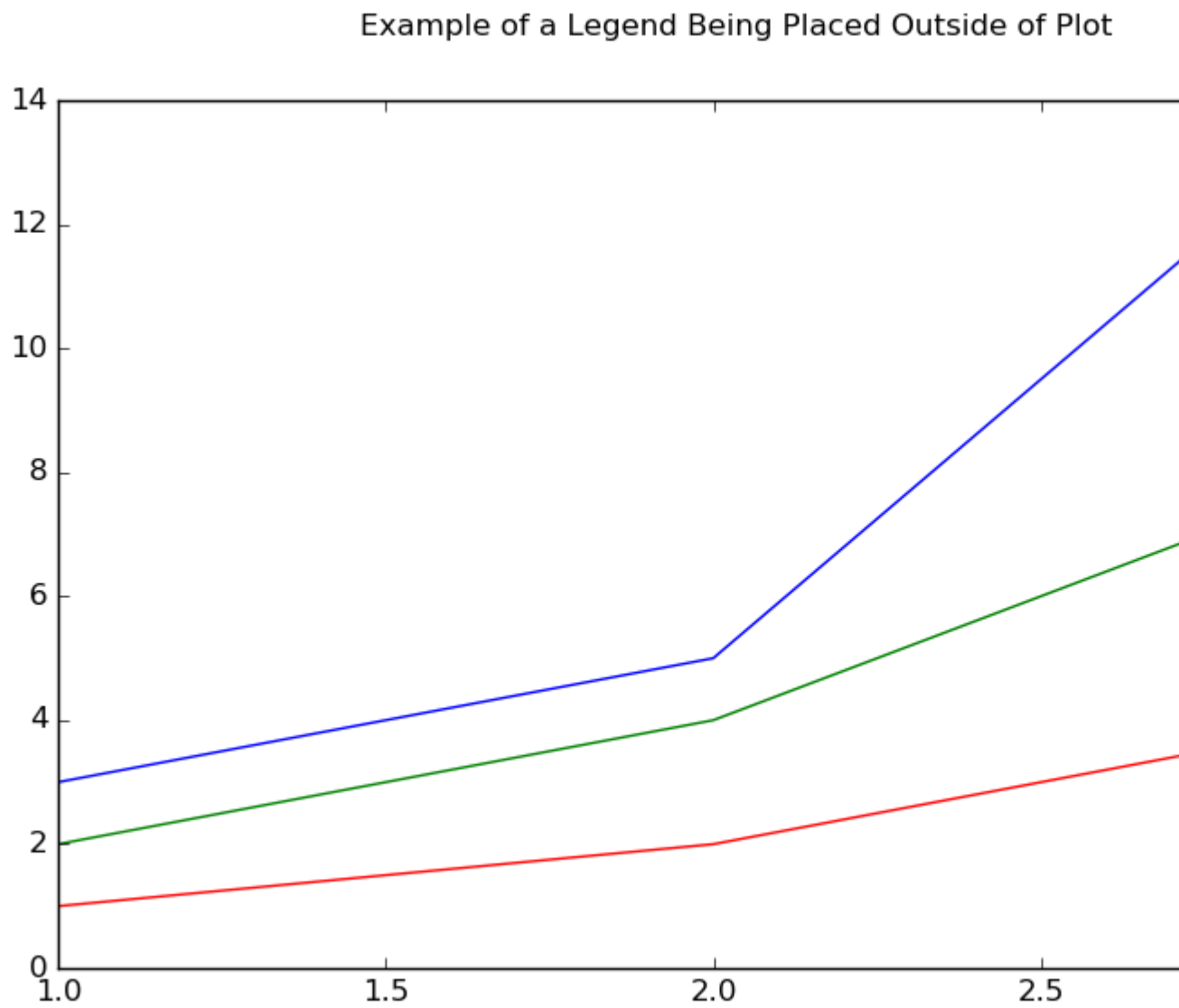



```
import matplotlib.pyplot as plt

# 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()
```

Légende placée à l'extérieur de la parcelle

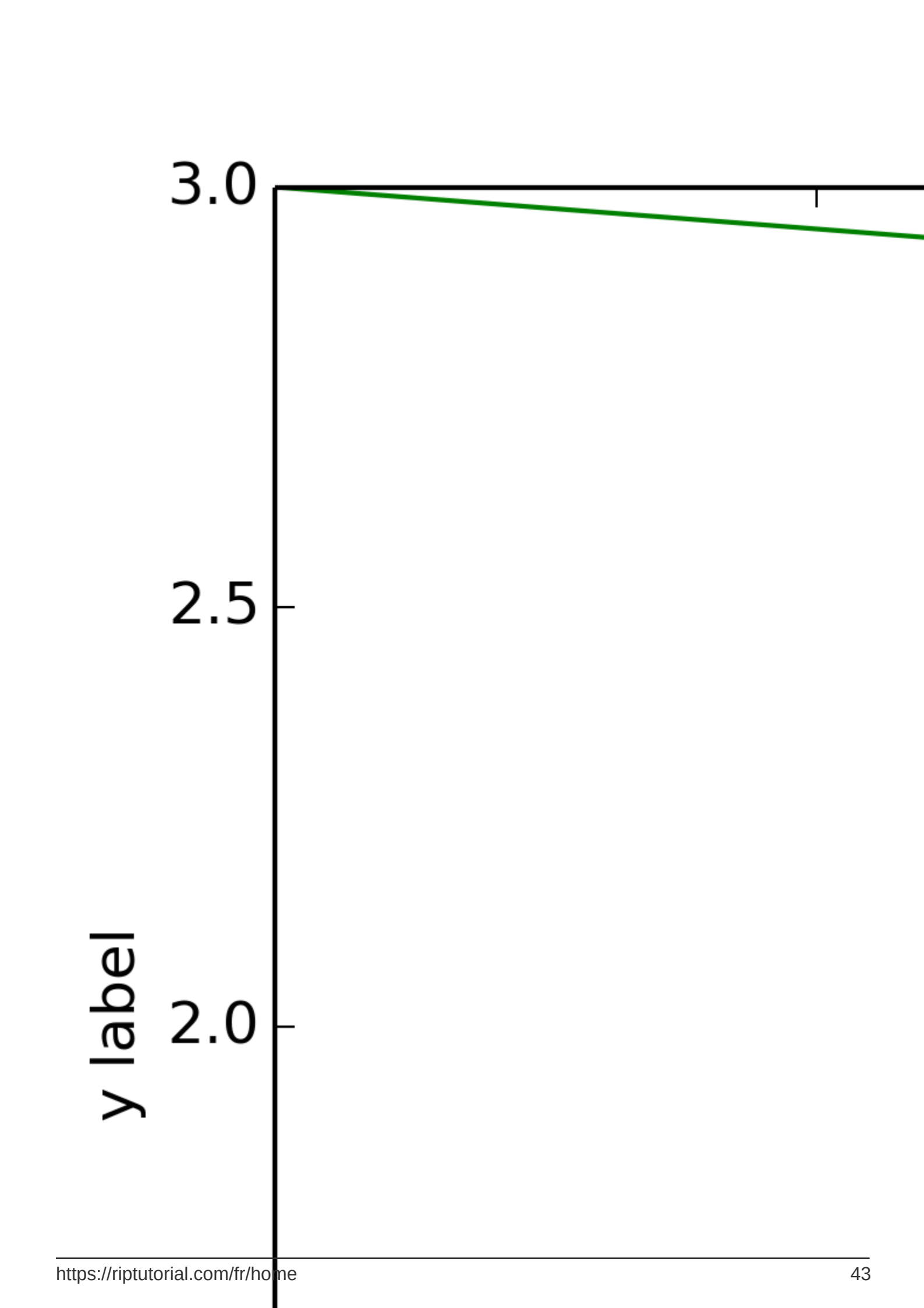
Parfois, il est nécessaire ou souhaitable de placer la légende en dehors de l'intrigue. Le code suivant montre comment le faire.




```
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()
```

Une autre façon de placer la légende en dehors du tracé consiste à utiliser `bbox_to_anchor` + `bbox_extra_artists` + `bbox_inches='tight'` , comme illustré dans l'exemple ci-dessous:


3.0
2.5
2.0
y label

au lieu de créer une légende au niveau des *axes* (ce qui créera une légende distincte pour chaque sous-intrigue). Ceci est réalisé en appelant `fig.legend()` comme on peut le voir dans le code pour le code suivant.

```
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()
```

Quelque chose à noter à propos de l'exemple ci-dessus est le suivant:

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

Lorsque `plot()` est appelé, il renvoie une liste d'objets **line2D** . Dans ce cas, il retourne simplement une liste avec un seul objet *line2D* , extrait avec l'indexation `[0]` et stocké dans `l1` .

Une liste de tous les objets *line2D* que nous souhaitons inclure dans la légende doit être transmise en premier argument à `fig.legend()` . Le deuxième argument de `fig.legend()` est également nécessaire. Il est supposé être une liste de chaînes à utiliser comme étiquettes pour chaque ligne de la légende.

Les autres arguments transmis à `fig.legend()` sont purement optionnels, et ne font que contribuer à affiner l'esthétique de la légende.

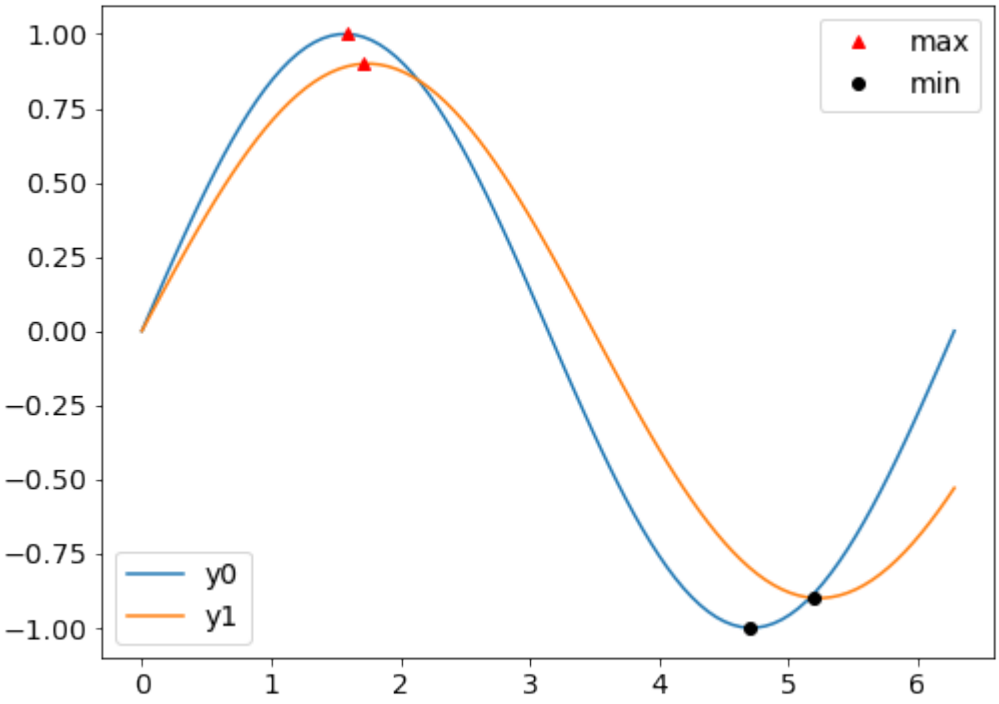
Plusieurs légendes sur les mêmes axes

Si vous appelez `plt.legend()` ou `ax.legend()` plus d'une fois, la première légende est supprimée et une nouvelle est dessinée. Selon la documentation officielle :

> Cela a été fait de sorte qu'il est possible d'appeler la légende () à plusieurs reprises pour mettre à jour la légende aux dernières poignées sur les axes

N'ayez crainte, cependant: il est encore assez simple d'ajouter une seconde légende (ou troisième ou quatrième ...) à un axe. Dans l'exemple ici, nous traçons deux lignes, puis tracons les marqueurs sur leurs maxima et minima respectifs. Une légende concerne les lignes et l'autre les marqueurs.

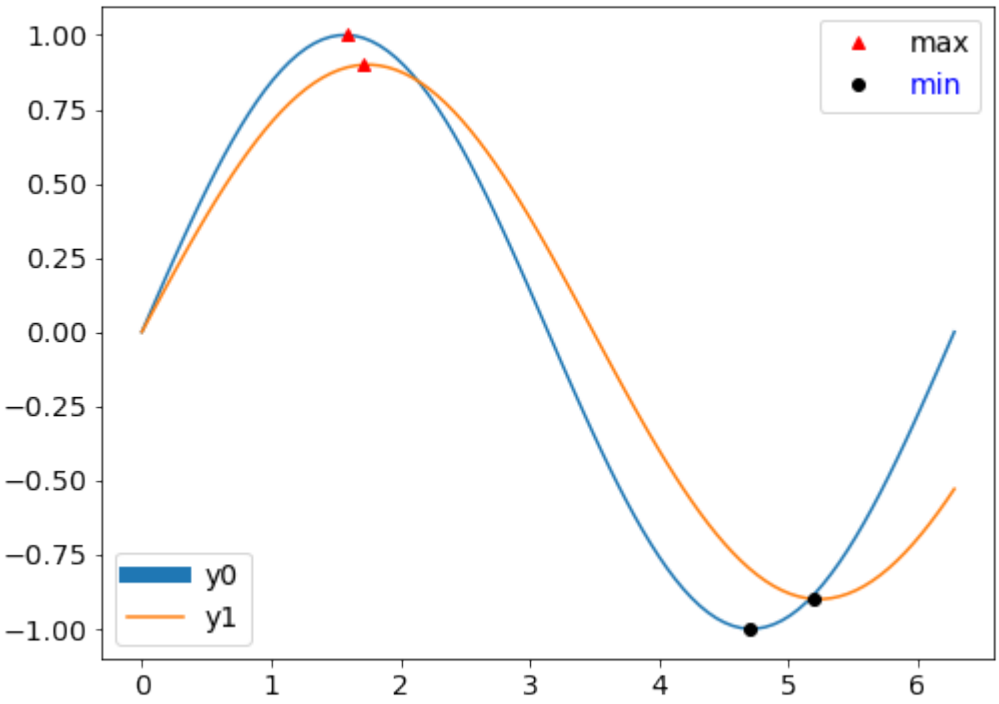
```
import matplotlib.pyplot as plt
import numpy as np

# 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 clé est de vous assurer que vous avez des références aux objets de légende. Le premier que vous instanciez (`leg1`) est supprimé de la figure lorsque vous ajoutez le second, mais l'objet `leg1` existe toujours et peut être rajouté avec `ax.add_artist` .

La grande chose est que vous pouvez toujours manipuler les *deux* légendes. Par exemple, ajoutez ce qui suit au bas du code ci-dessus:

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



Enfin, il convient de mentionner que dans l'exemple, seules les lignes ont reçu des étiquettes

lorsqu'elles ont été tracées, ce qui signifie que `ax.legend()` ajoute uniquement ces lignes à la `leg1` . La légende des marqueurs (`leg2`) nécessitait donc que les lignes et les étiquettes `leg2` arguments lorsqu'elle était instanciée. Nous pourrions aussi avoir des étiquettes sur les marqueurs lorsqu'ils ont été tracés. Mais alors, les *deux* appels à `ax.legend` auraient nécessité des arguments supplémentaires pour que chaque légende ne contienne que les éléments souhaités.

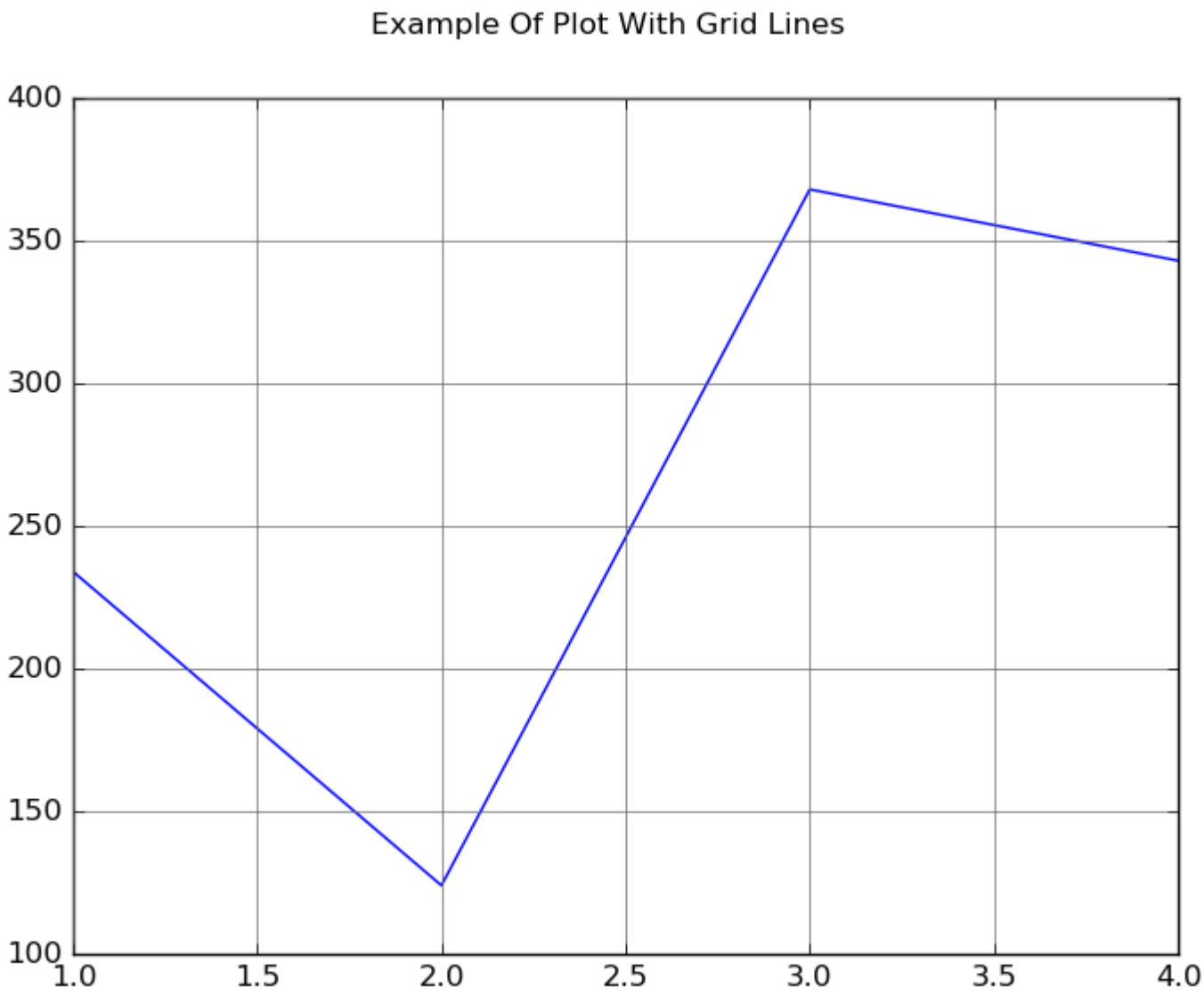
Lire Légendes en ligne: https://riptutorial.com/fr/matplotlib/topic/2840/legendes

Chapitre 11: Lignes de quadrillage et repères

Examples

Terrain avec lignes de quadrillage

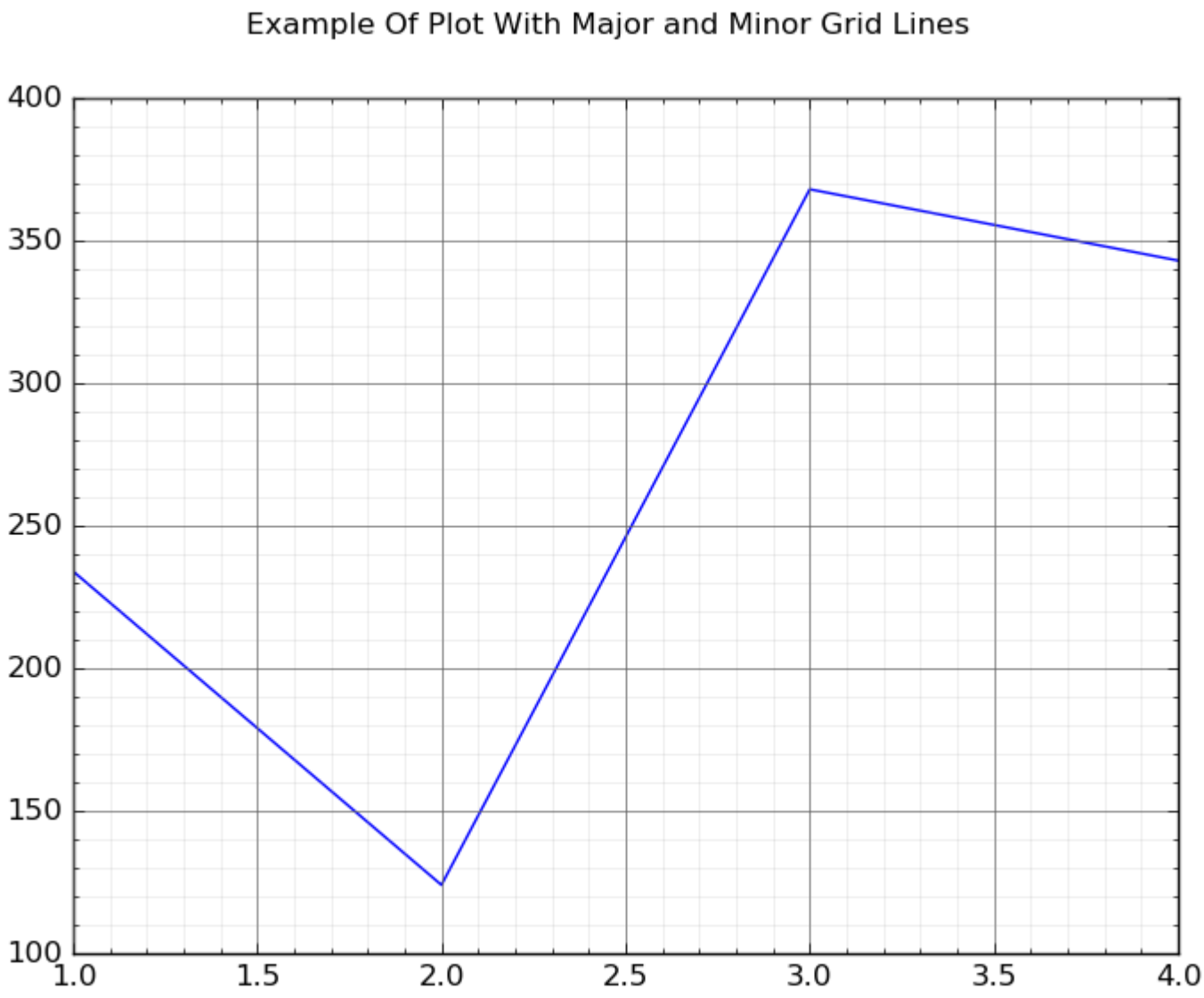
Tracer avec des lignes de grille




```
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()
```

Tracer avec des lignes de quadrillage majeures et mineures




```
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()
```

Lire Lignes de quadrillage et repères en ligne: https://riptutorial.com/fr/matplotlib/topic/4029/lignes-de-quadrillage-et-reperes

Chapitre 12: LogLog Graphing

Introduction

LogLog graphing est une possibilité d'illustrer une fonction exponentielle de manière linéaire.

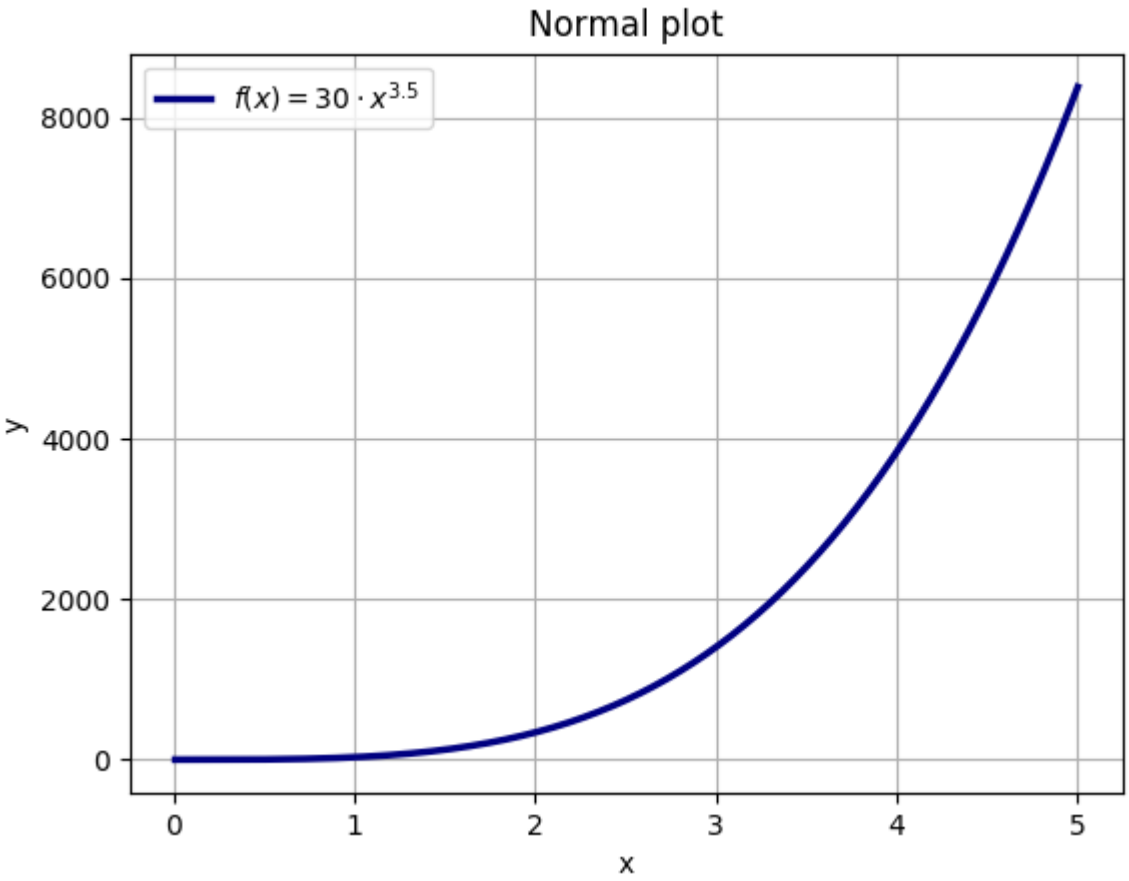
Examples

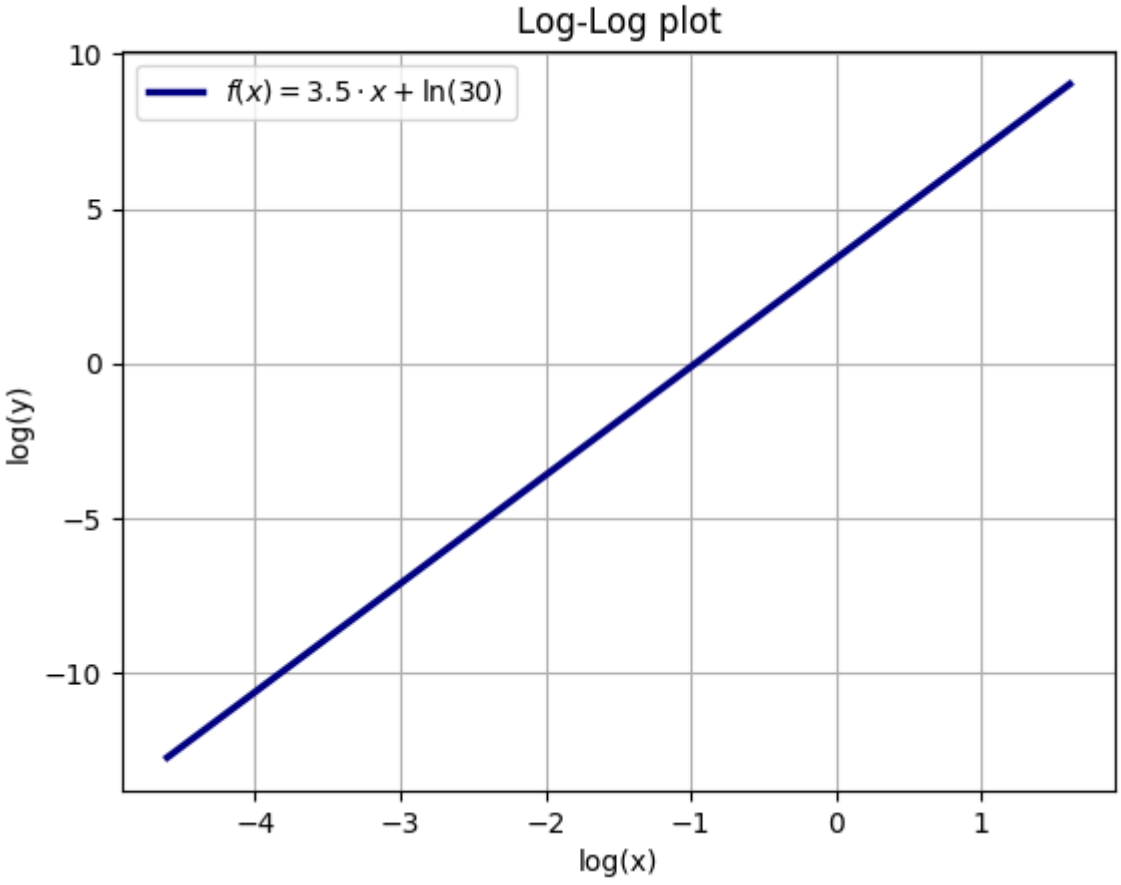
LogLog graphique

Soit y (x) = A * x ^ a, par exemple A = 30 et a = 3.5. En prenant le logarithme naturel (ln) des deux côtés, on obtient (en utilisant les règles communes pour les logarithmes): ln (y) = ln (A * x ^ a) = ln (A) + ln (x ^ a) = ln (A) + a * ln (x). Ainsi, un tracé avec des axes logarithmiques pour x et y sera une courbe linéaire. La pente de cette courbe est l'exposant a de y (x), tandis que l'ordonnée à l'origine y (0) est le logarithme naturel de A, ln (A) = ln (30) = 3,401.

L'exemple suivant illustre la relation entre une fonction exponentielle et le tracé de loglog linéaire (la fonction est y = A * x ^ a avec A = 30 et 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)

Lire LogLog Graphing en ligne: https://riptutorial.com/fr/matplotlib/topic/10145/loglog-graphing

Chapitre 13: Manipulation d'image

Examples

Images d'ouverture

Matplotlib comprend l' `image` de module de manipulation d'image

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

Les images sont lues depuis un fichier (`.png` uniquement) avec la fonction `imread` :

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

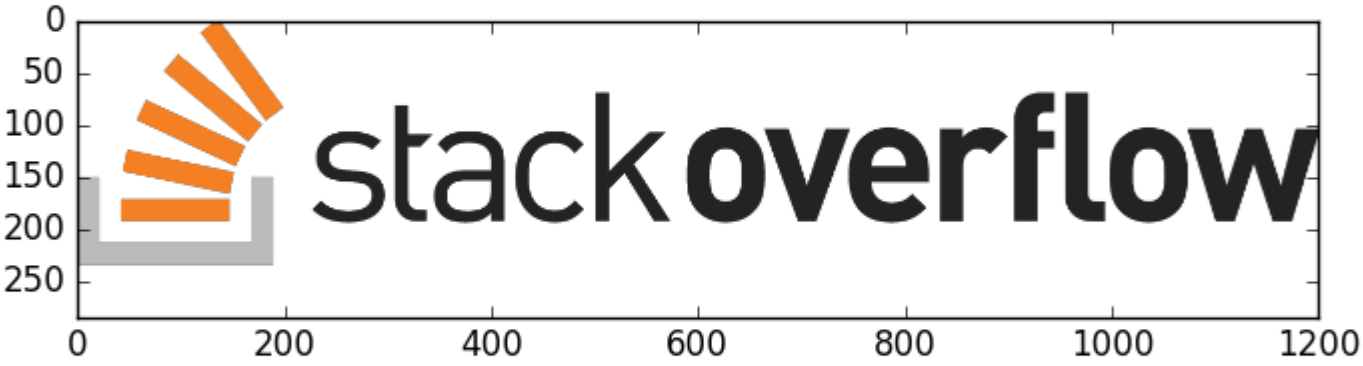
et ils sont rendus par la fonction `imshow` :

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

Nous allons *tracer* le logo Stack Overflow :

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

Le tracé résultant est



Lire Manipulation d'image en ligne: https://riptutorial.com/fr/matplotlib/topic/4575/manipulation-d-image

Chapitre 14: Objets de figures et d'axes

Examples

Créer une figure

La figure contient tous les éléments du tracé. Le principal moyen de créer une figure dans `matplotlib` consiste à utiliser le `pyplot` .

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

Vous pouvez éventuellement fournir un numéro que vous pouvez utiliser pour accéder à une image précédemment créée. Si aucun numéro n'est fourni, l'ID du dernier personnage créé sera incrémenté et utilisé à la place; les chiffres sont indexés à partir de 1 et non de 0.

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

Au lieu d'un nombre, les chiffres peuvent également être identifiés par une chaîne. Si vous utilisez un backend interactif, cela définira également le titre de la fenêtre.

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

Pour choisir l'utilisation de la figure

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

Créer un axe

Il existe deux manières principales de créer un axe dans matplotlib: utiliser pyplot ou utiliser l’API orientée objet.

En utilisant pyplot:

```
import matplotlib.pyplot as plt

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

En utilisant l'API orientée objet:

```
import matplotlib.pyplot as plt

fig = plt.figure()
```

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

La fonction de commodité `plt.subplots()` peut être utilisée pour produire une figure et une collection de sous-parcelles dans une commande:

```
import matplotlib.pyplot as plt

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

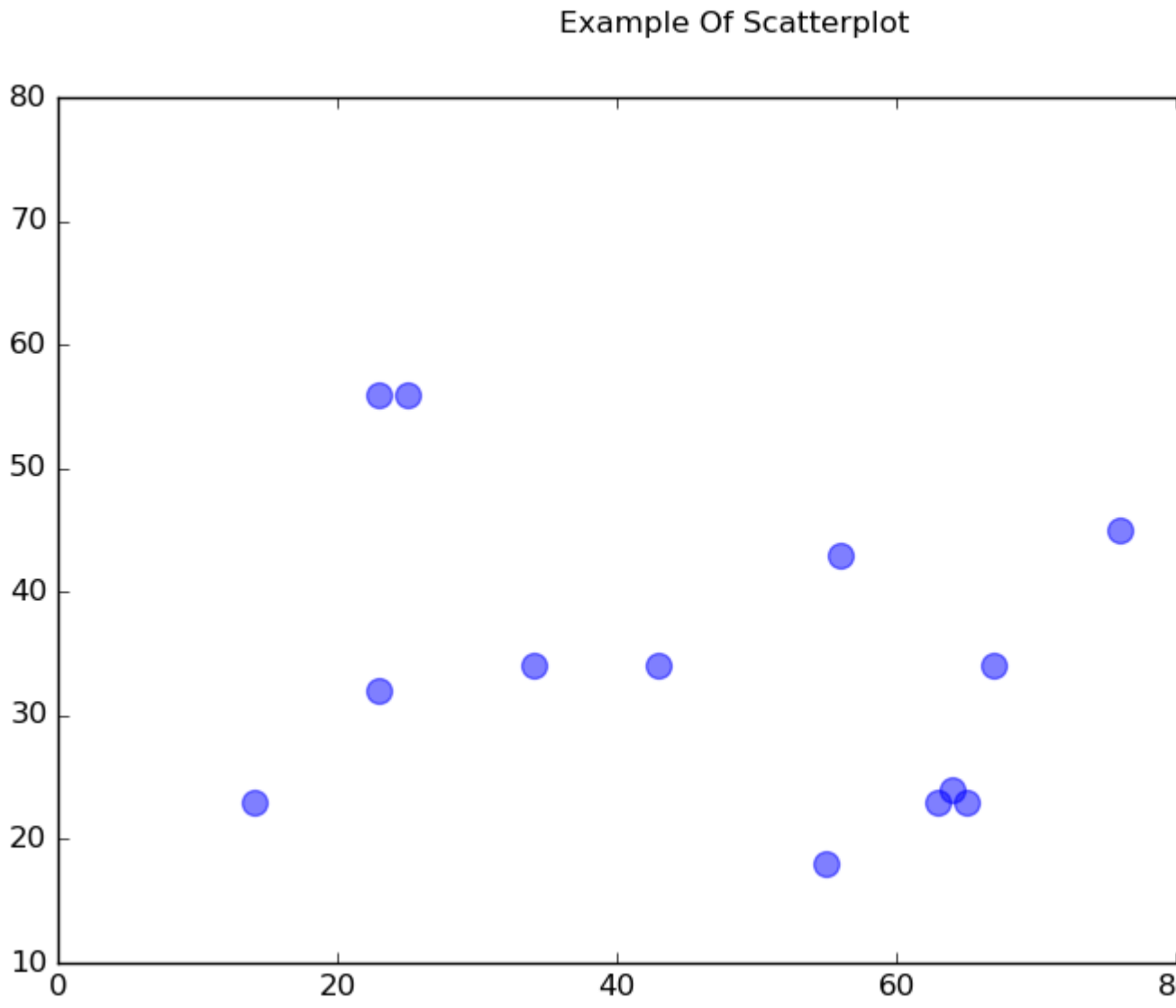
Lire Objets de figures et d'axes en ligne: https://riptutorial.com/fr/matplotlib/topic/2307/objets-de-figures-et-d-axes

Chapitre 15: Parcelles de base

Examples

Scatter Plots

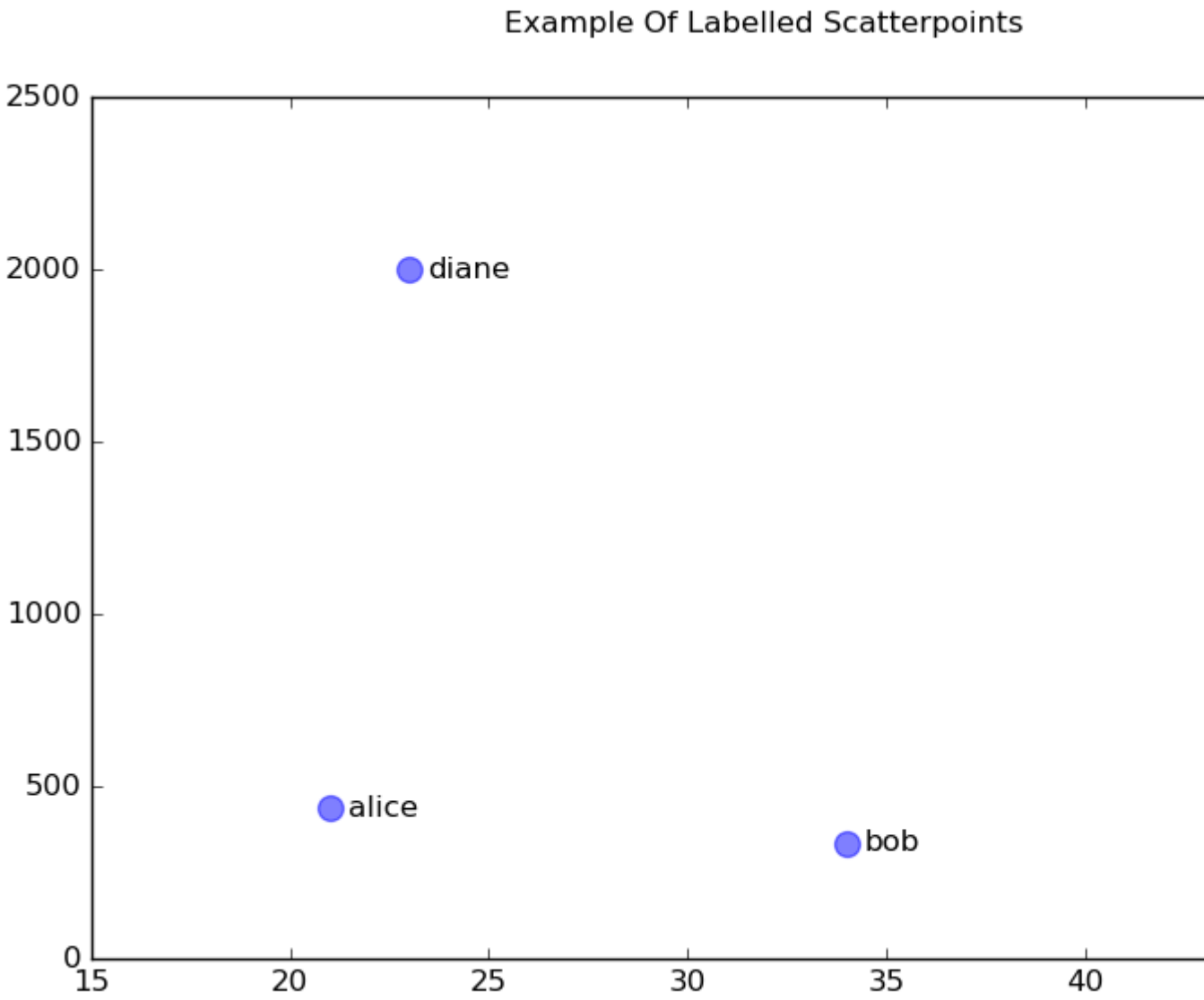
Un simple nuage de points




```
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 nuage de points avec des points étiquetés

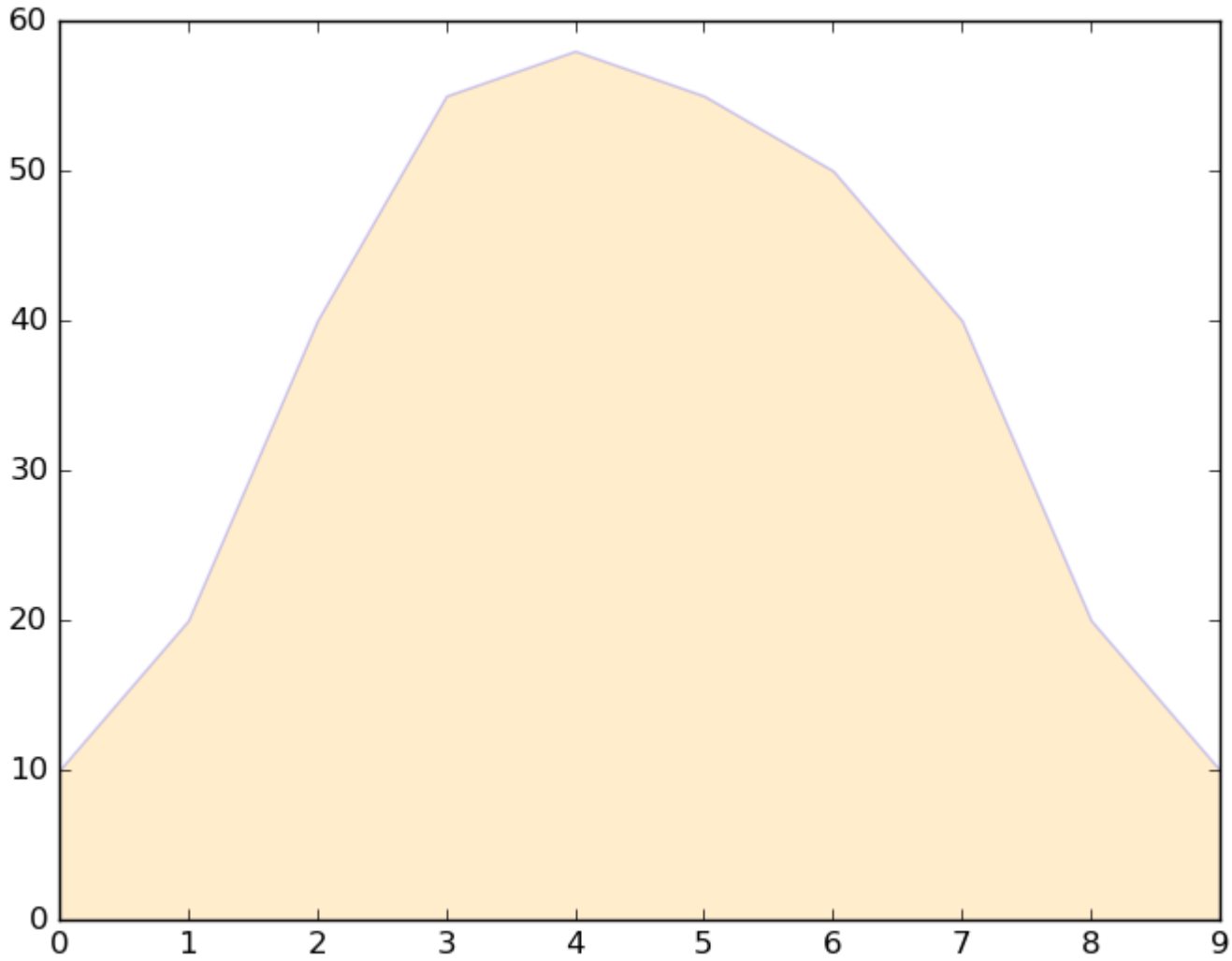



```
import matplotlib.pyplot as plt

# 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()
```

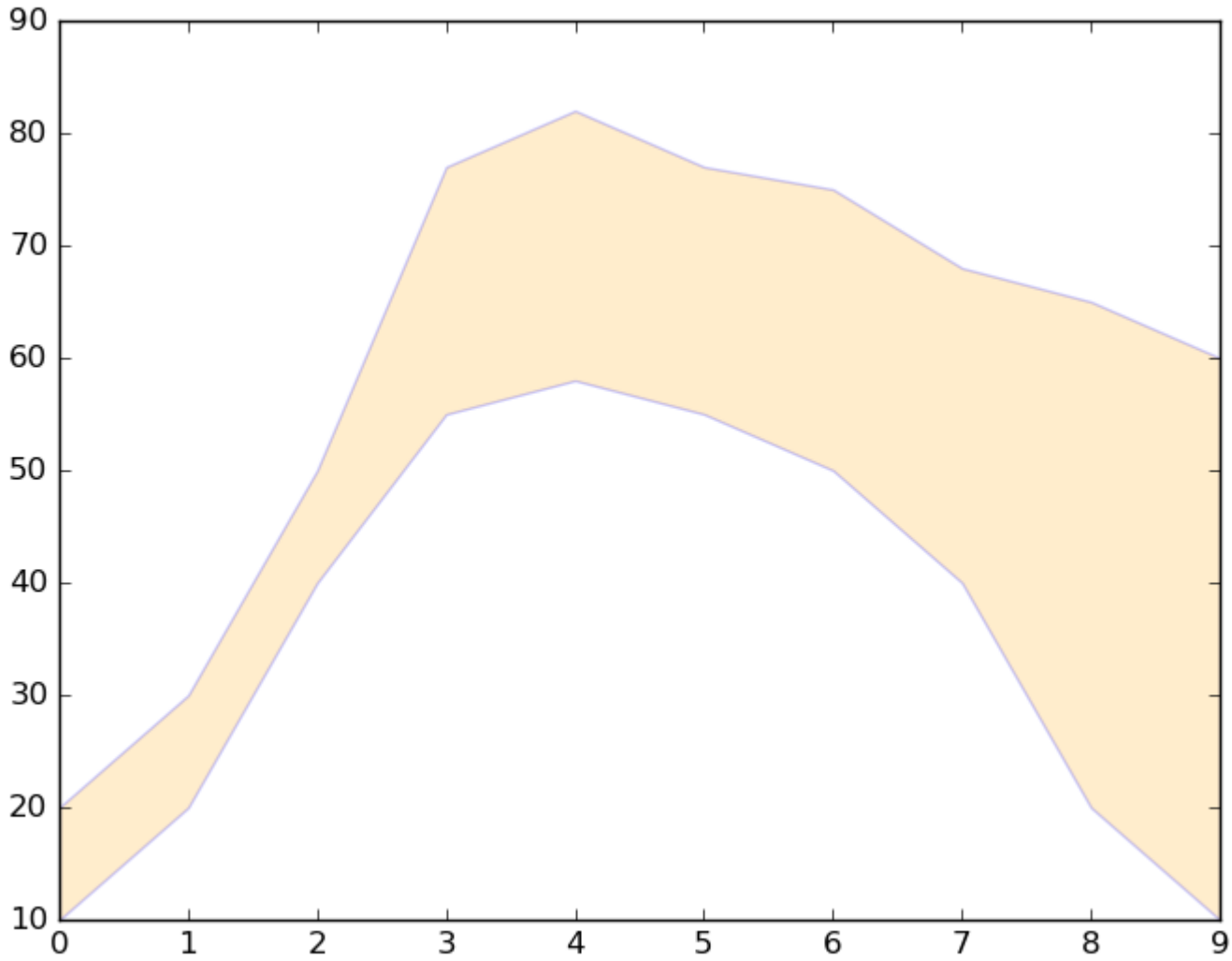
Parcelles ombrées

Région ombrée sous une ligne




```
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()
```

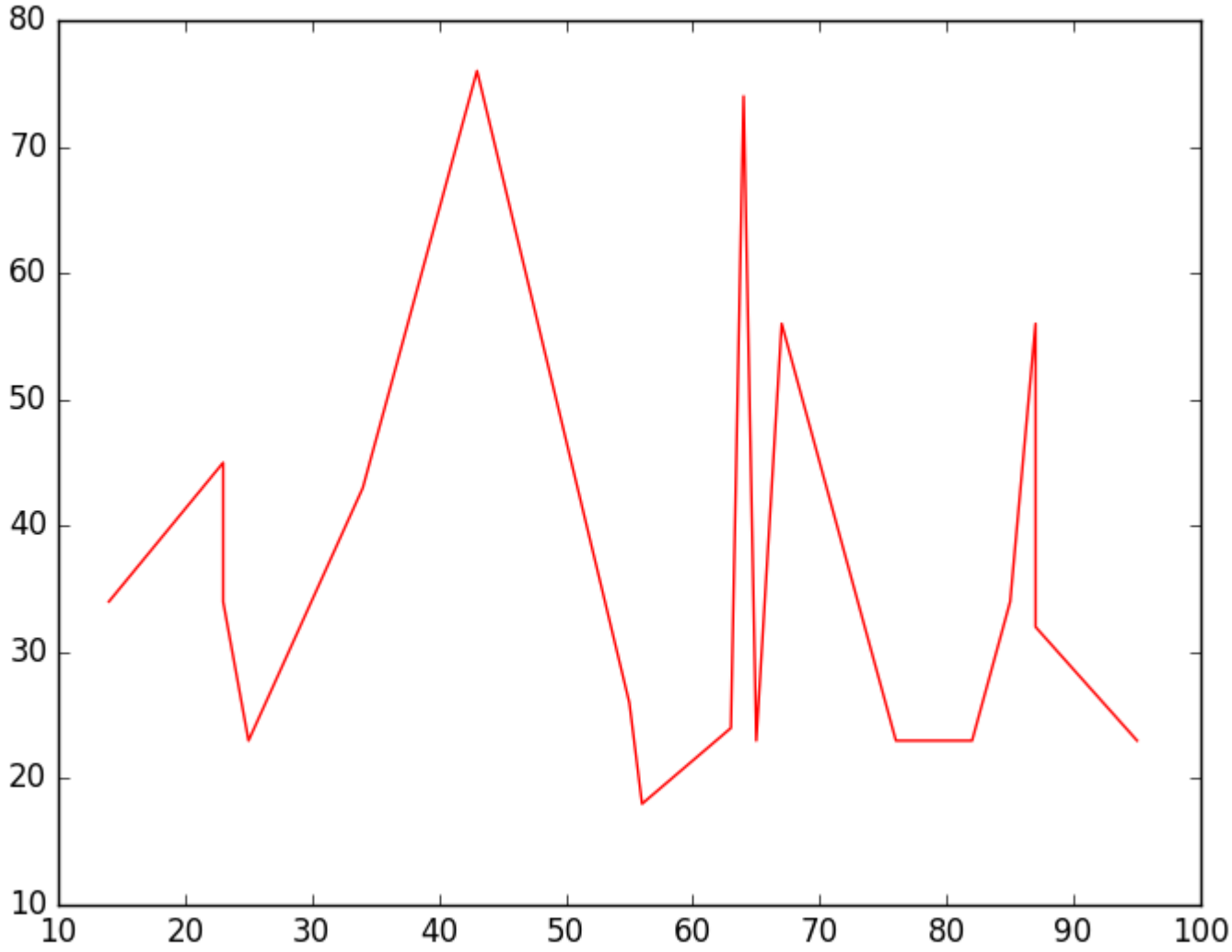
Région ombrée entre deux lignes




```
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()
```

Tracés de ligne

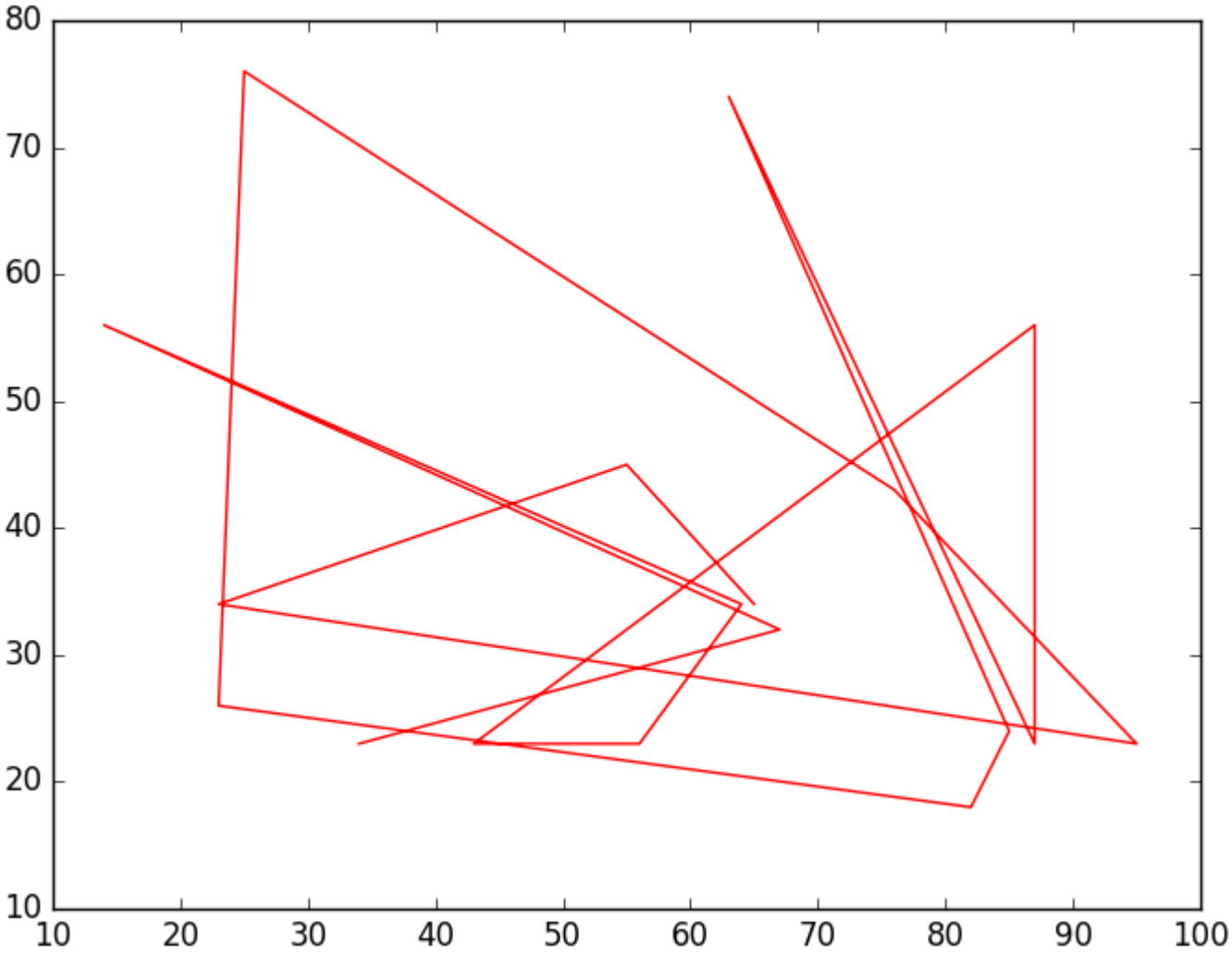
Tracé simple




```
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()
```

Notez qu'en général, `y` n'est pas une fonction de `x` et que les valeurs de `x` n'ont pas besoin d'être triées. Voici à quoi ressemble un tracé avec des valeurs de x non triées:

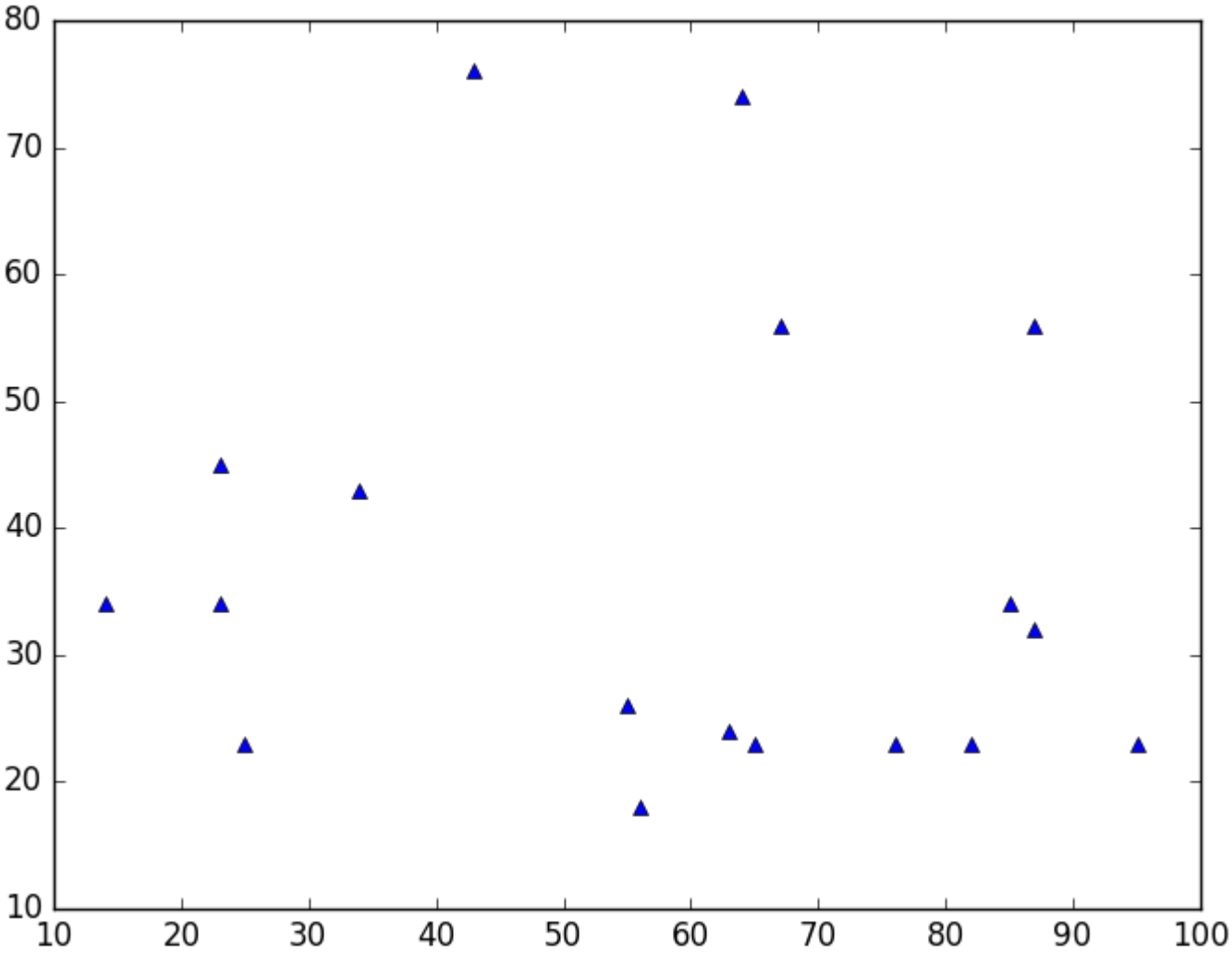
```
# shuffle the elements in x
np.random.shuffle(x)
plt.plot(x, y, 'r-')
plt.show()
```



Tracé de données

Ceci est similaire à un nuage de points , mais utilise la fonction `plot()` place. La seule différence dans le code est l'argument de style.

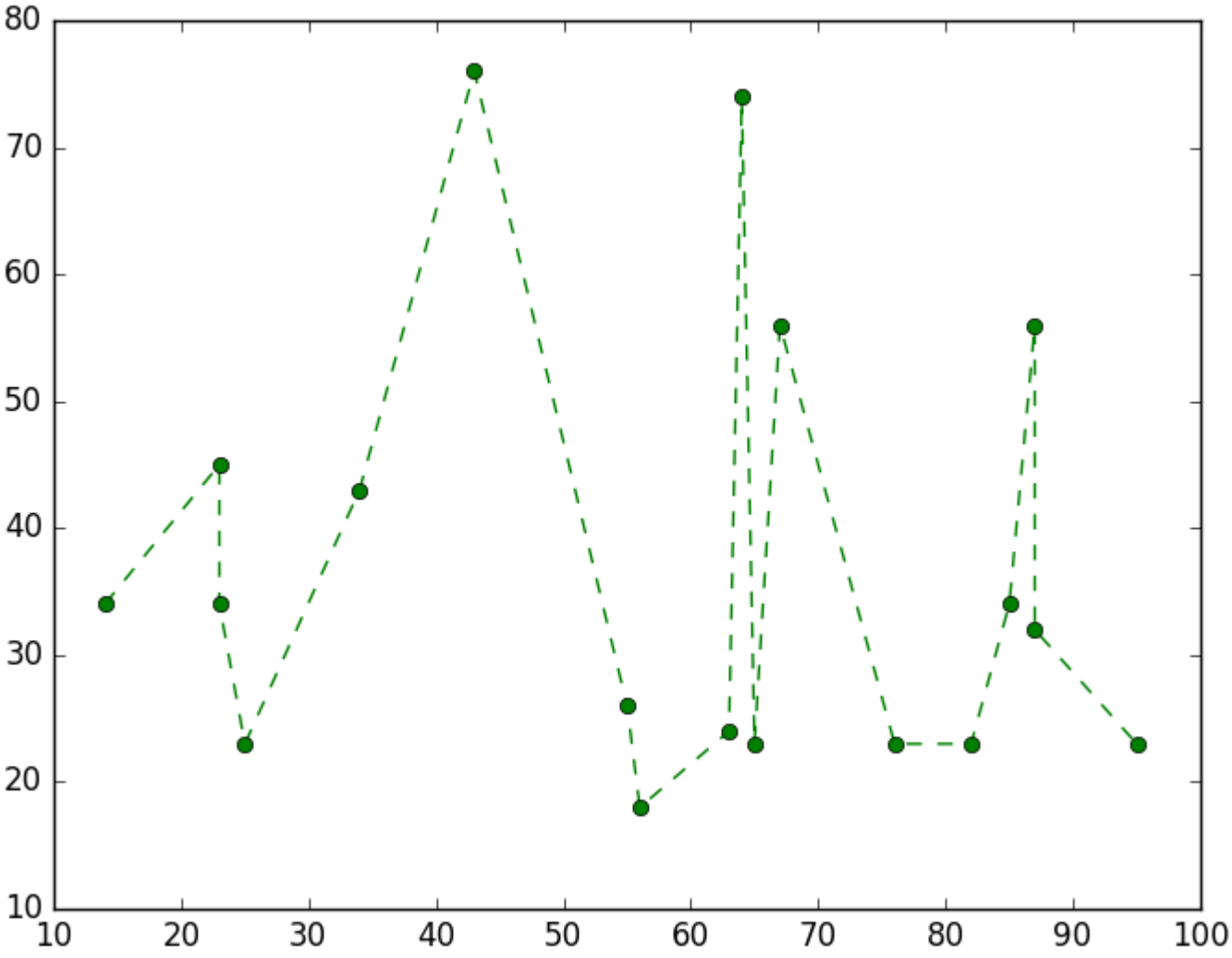
```
plt.plot(x, y, 'b^')
# Create blue up-facing triangles
```



Données et ligne

L'argument de style peut prendre des symboles pour les marqueurs et le style de ligne:

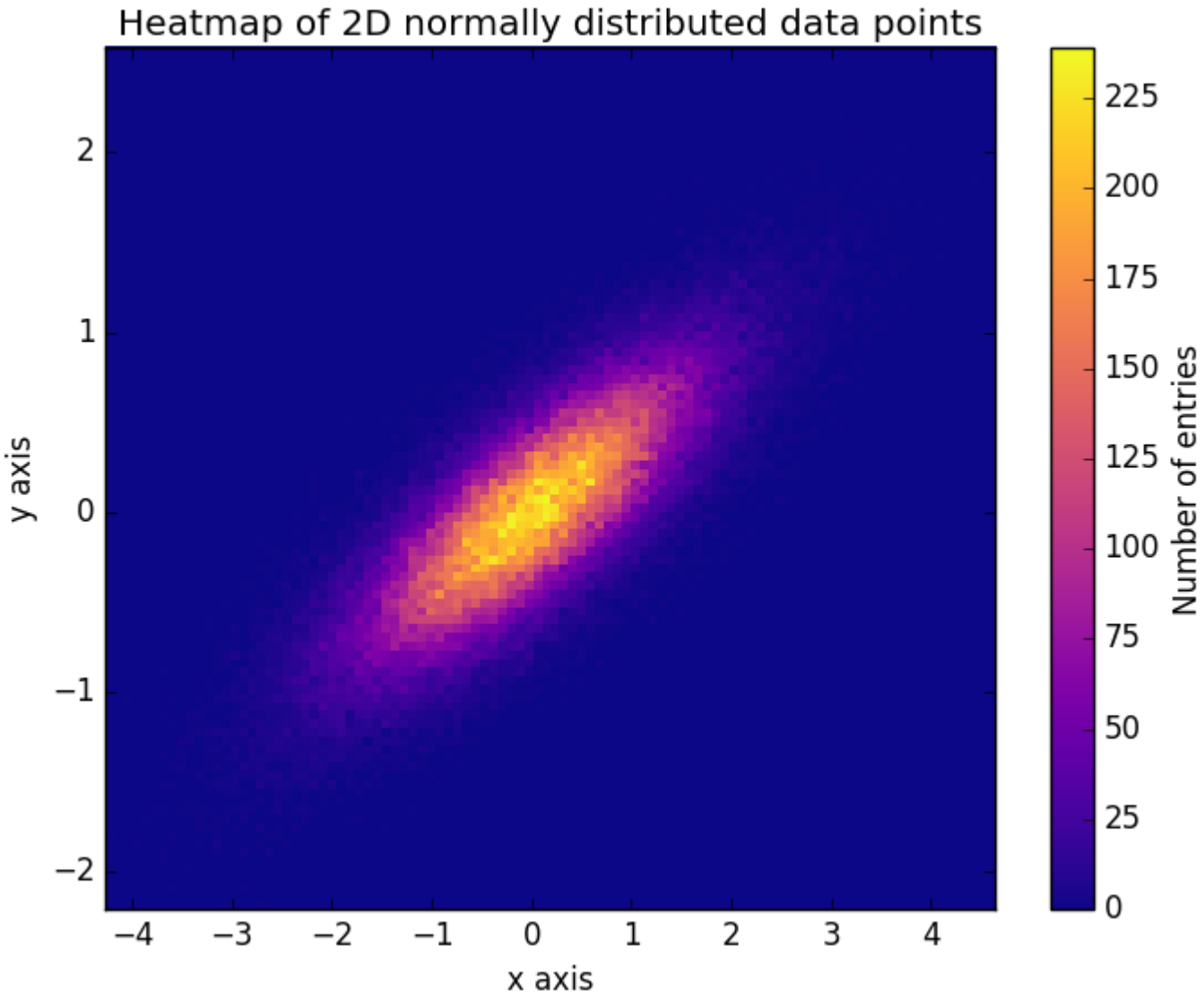
```
plt.plot(x, y, 'go--')
# green circles and dashed line
```



Carte de chaleur

Les Heatmaps sont utiles pour visualiser les fonctions scalaires de deux variables. Ils fournissent une image «plate» des histogrammes bidimensionnels (représentant par exemple la densité d'une certaine zone).

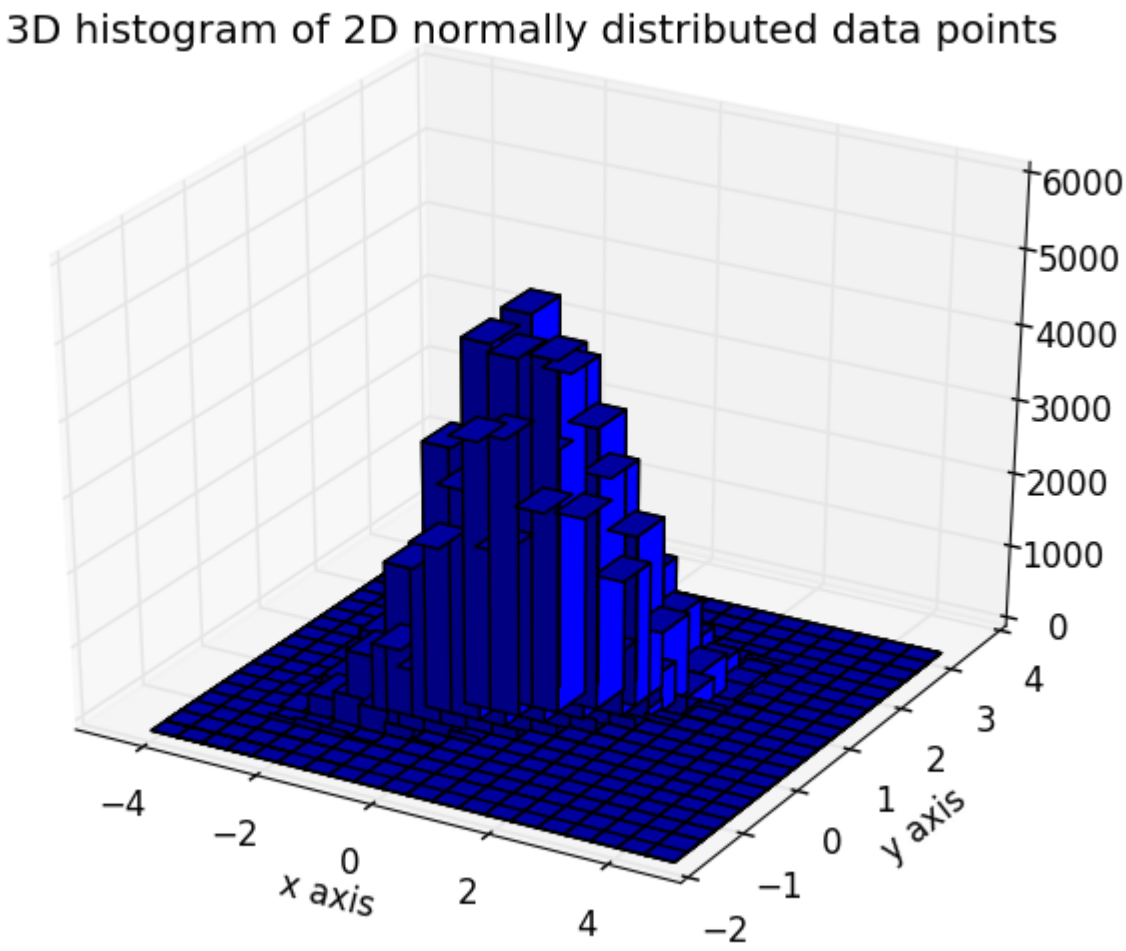
Le code source suivant illustre les cartes thermiques en utilisant des nombres bivariés normalement distribués centrés sur 0 dans les deux directions (moyennes `[0.0, 0.0]`) et a avec une matrice de covariance donnée. Les données sont générées à l'aide de la fonction numpy numpy.random.multivariate_normal ; il est ensuite introduit dans le `hist2d` fonction 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()
```

Voici les mêmes données visualisées sous forme d'histogramme 3D (nous utilisons ici seulement 20 cases pour l'efficacité). Le code est basé sur cette démo 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()
```

Lire Parcelles de base en ligne: https://riptutorial.com/fr/matplotlib/topic/3266/parcelles-de-base

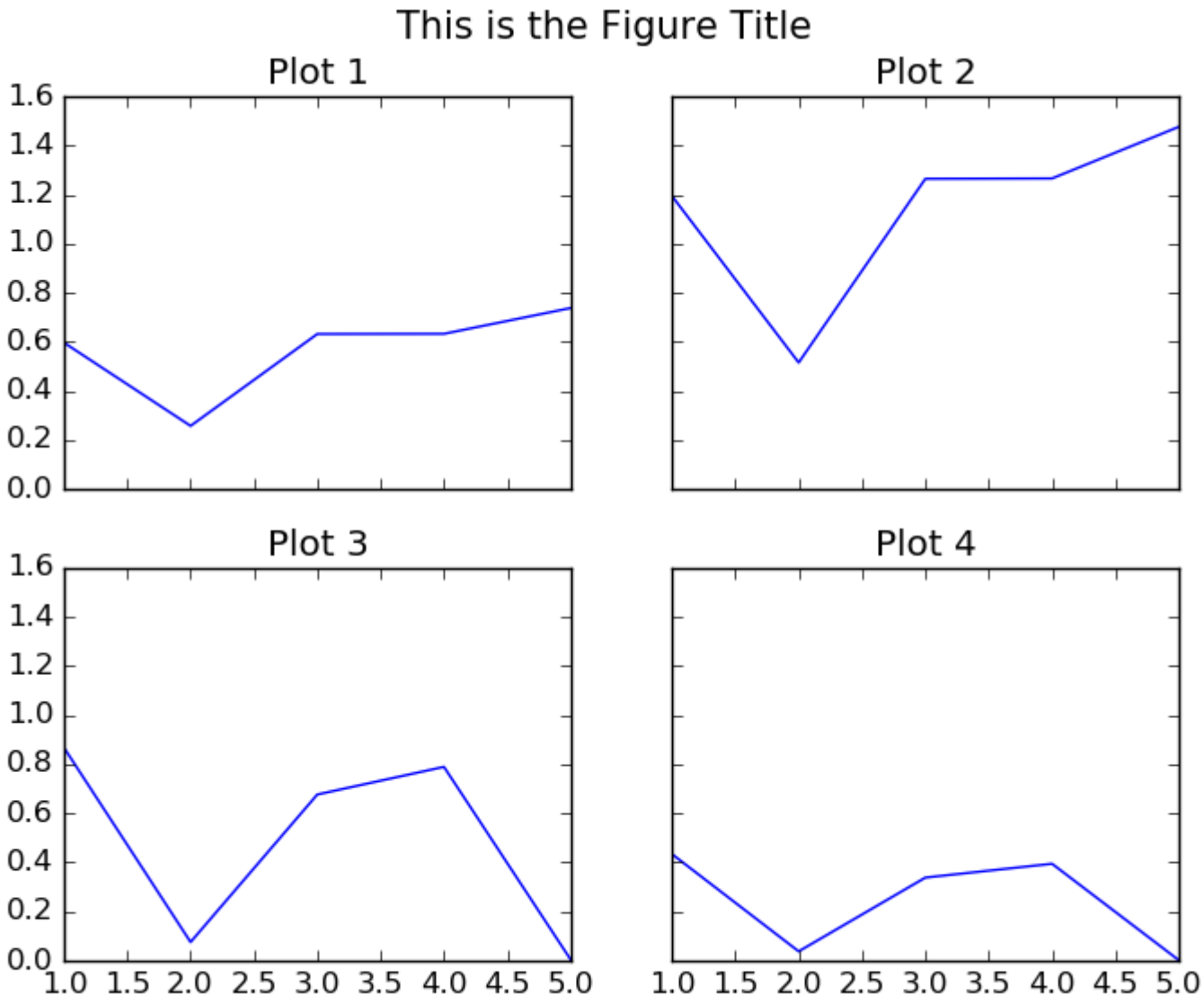
Chapitre 16: Plusieurs parcelles

Syntaxe

- Élément de la liste

Examples

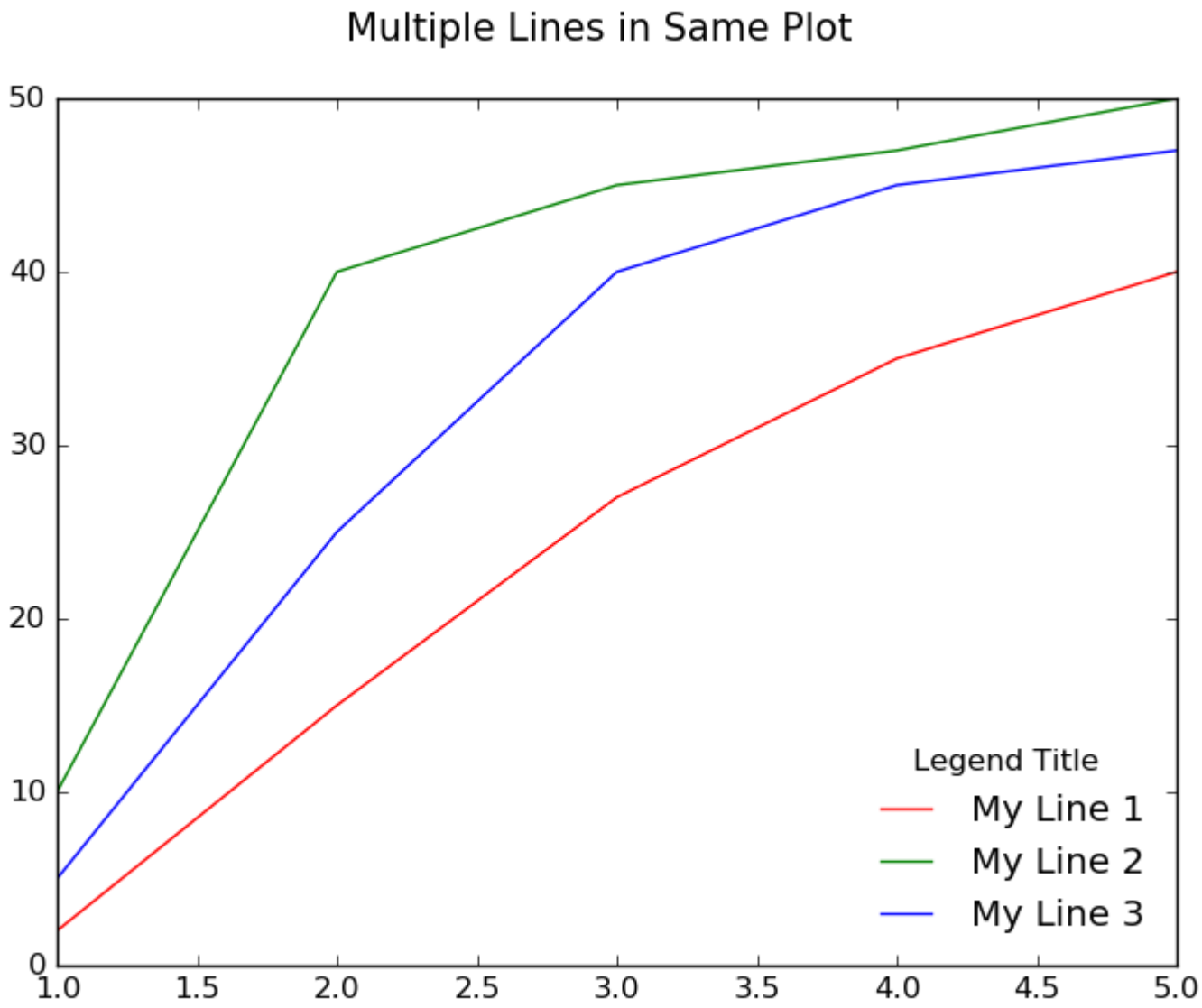
Grille de sous-parcelles utilisant la sous-parcelle




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

Plusieurs lignes / courbes dans le même tracé




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

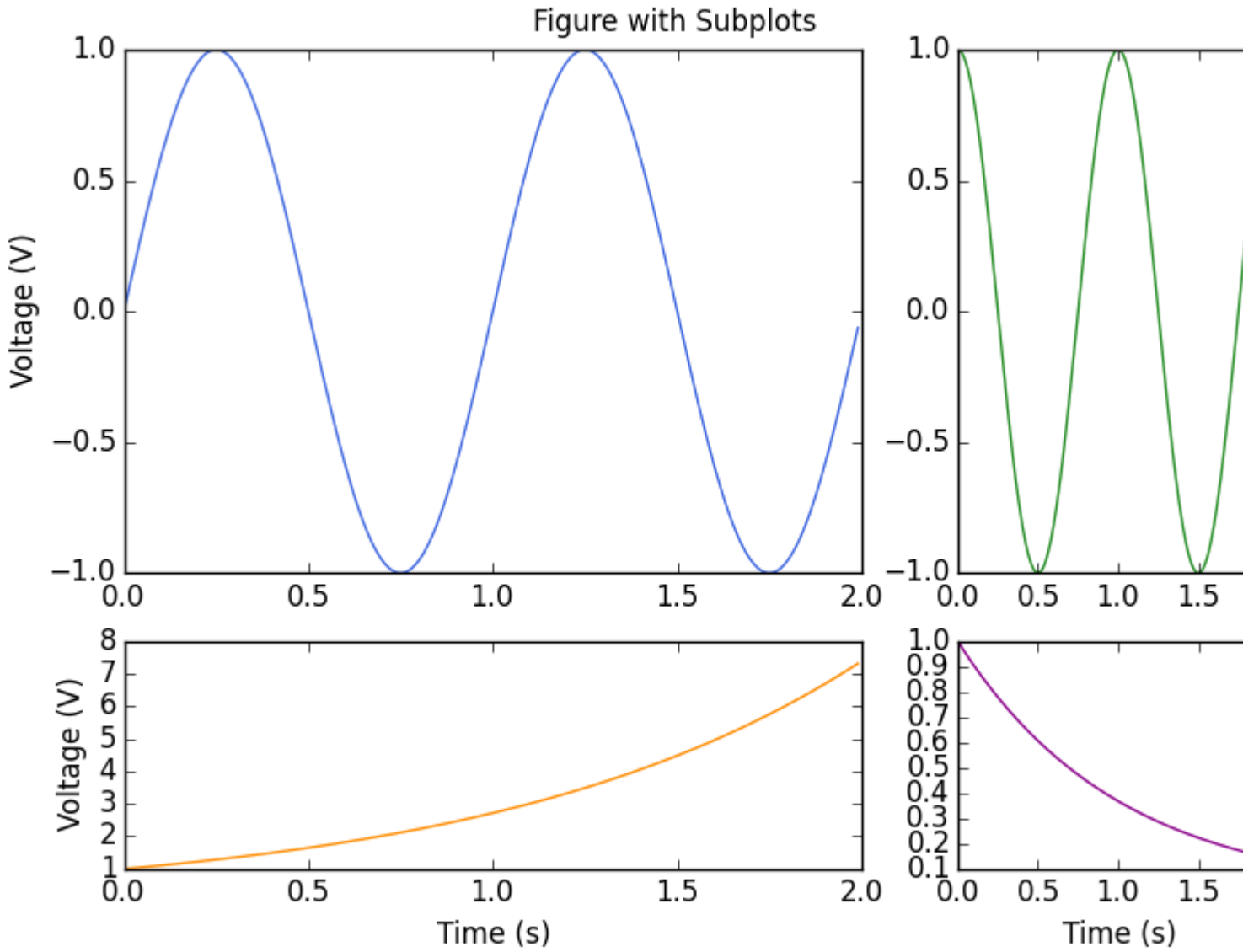
```
ax.plot(x, y3, color="blue", label="My Line 3")

# 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()
```

Plusieurs parcelles avec gridspec

Le paquet `gridspec` permet de mieux contrôler le placement des sous-parcelles. Il est beaucoup plus facile de contrôler les marges des parcelles et l'espacement entre les sous-parcelles individuelles. De plus, il permet des axes de tailles différentes sur la même figure en définissant des axes qui occupent plusieurs emplacements de grille.

```
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)')
```

Ce code produit le tracé ci-dessous.

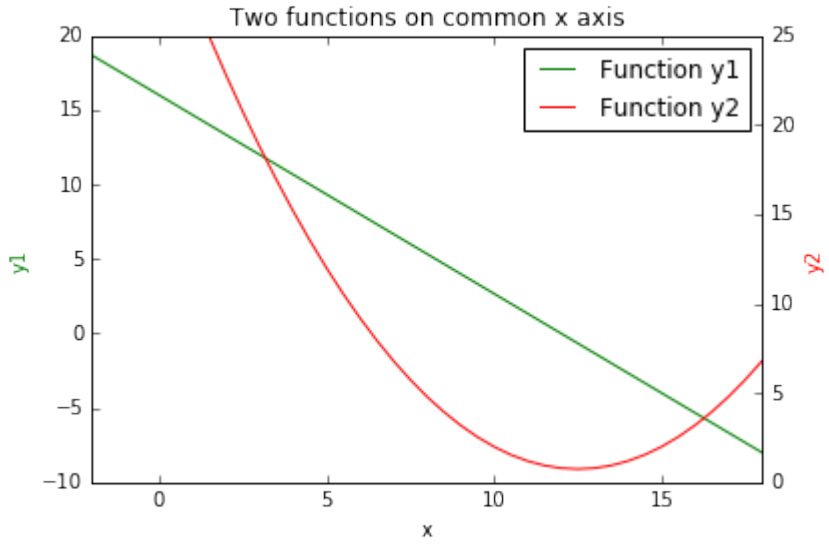


Un tracé de 2 fonctions sur l'axe des x partagé.

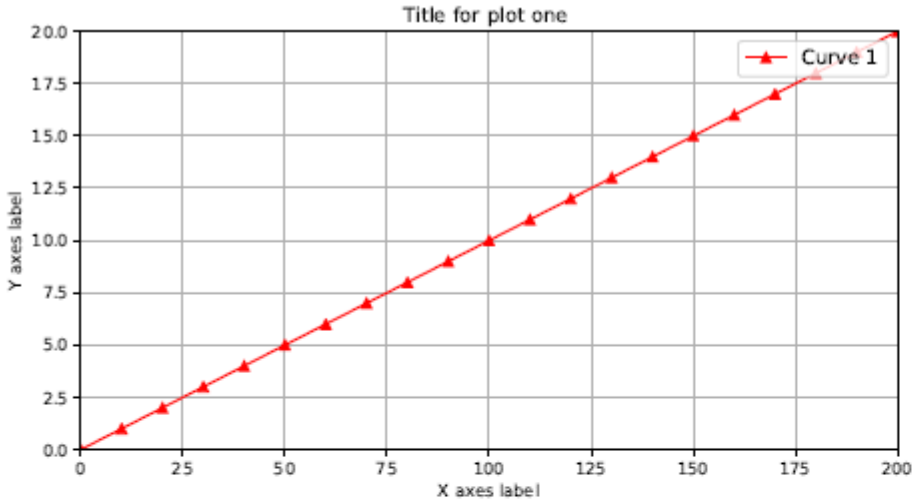
```
import numpy as np
import matplotlib.pyplot as plt

# 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()
```

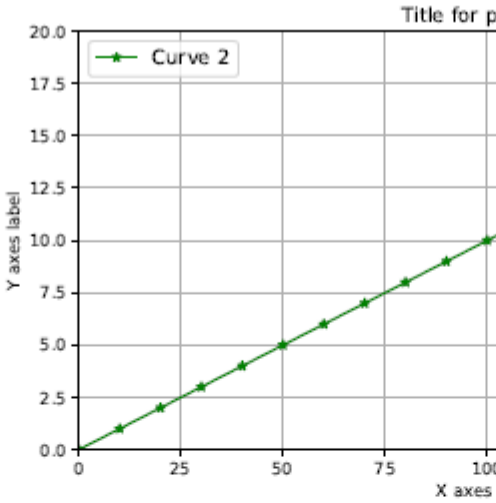
```
# create line plot of y2(x)
line2, = ax2.plot(x, y2, 'r', label="Function y2")
ax2.set_ylabel('y2', color='r')

# 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()
```

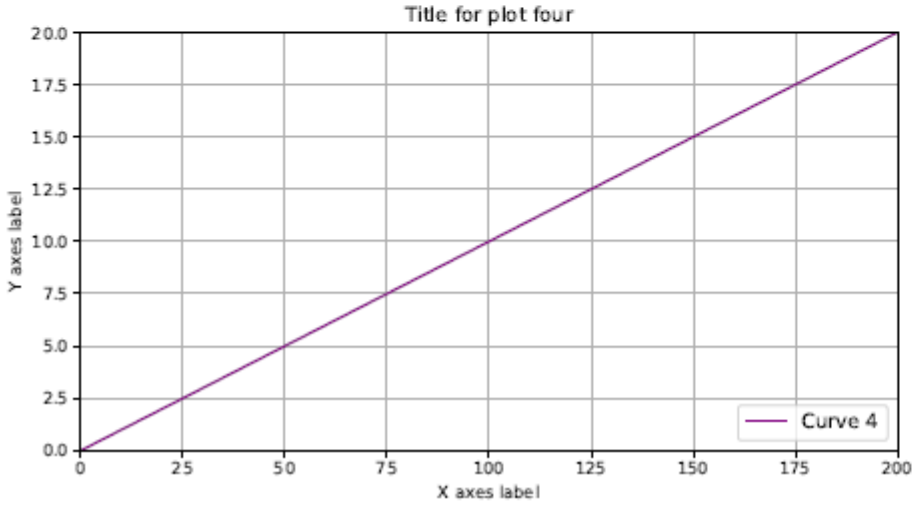
Ce code produit le tracé ci-dessous.

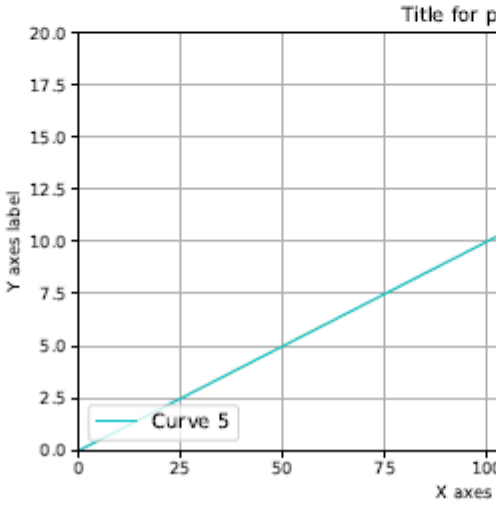


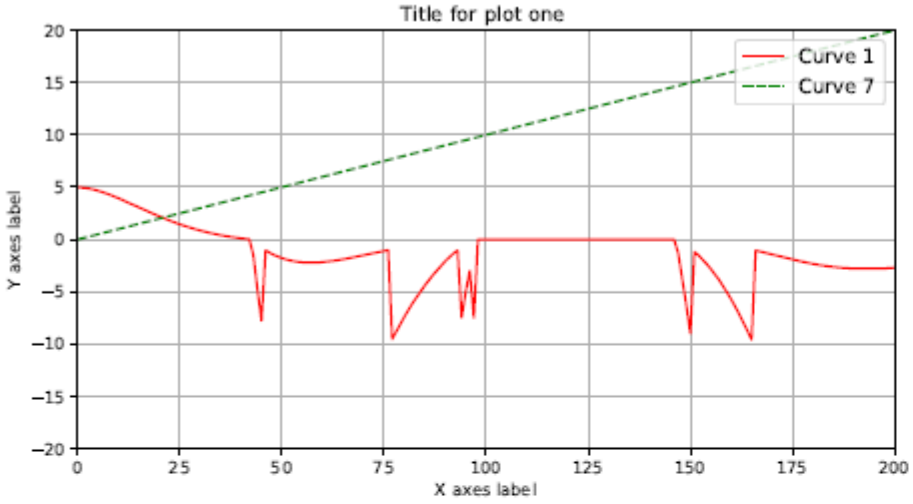
Plusieurs parcelles et plusieurs caractéristiques de parcelles

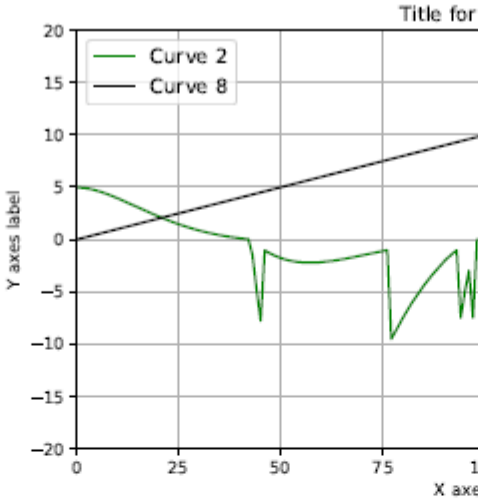

Title for plot one
Curve 1
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200

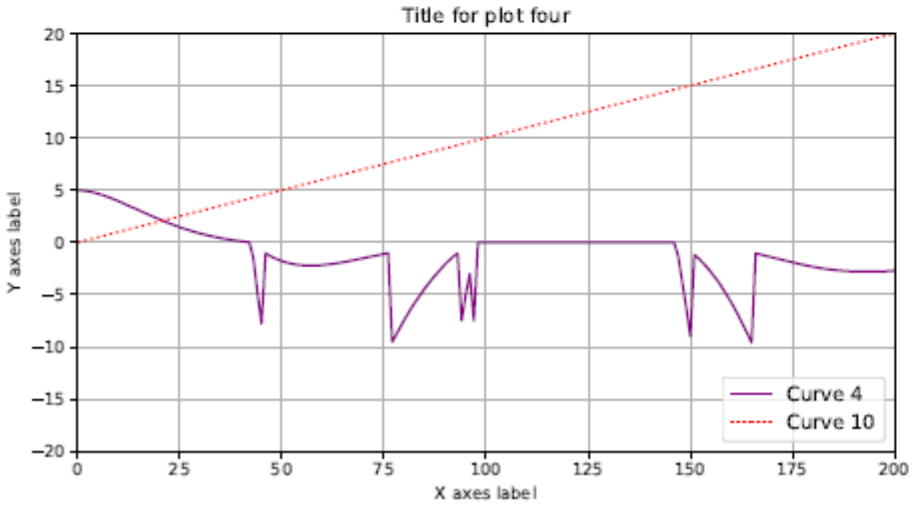

Curve 2
Y axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75

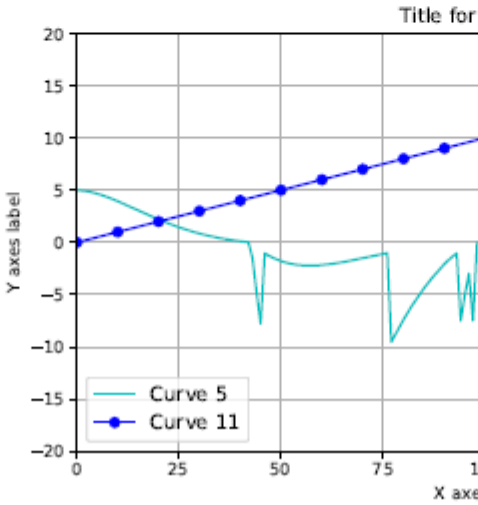

Title for plot four
Curve 4
Y axes label
X axes label
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
25
50
75
100
125
150
175
200


Curve 5
Y axes label
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()
```

Lire Plusieurs parcelles en ligne: https://riptutorial.com/fr/matplotlib/topic/3279/plusieurs-parcelles

Chapitre 17: Systèmes de coordonnées

Remarques

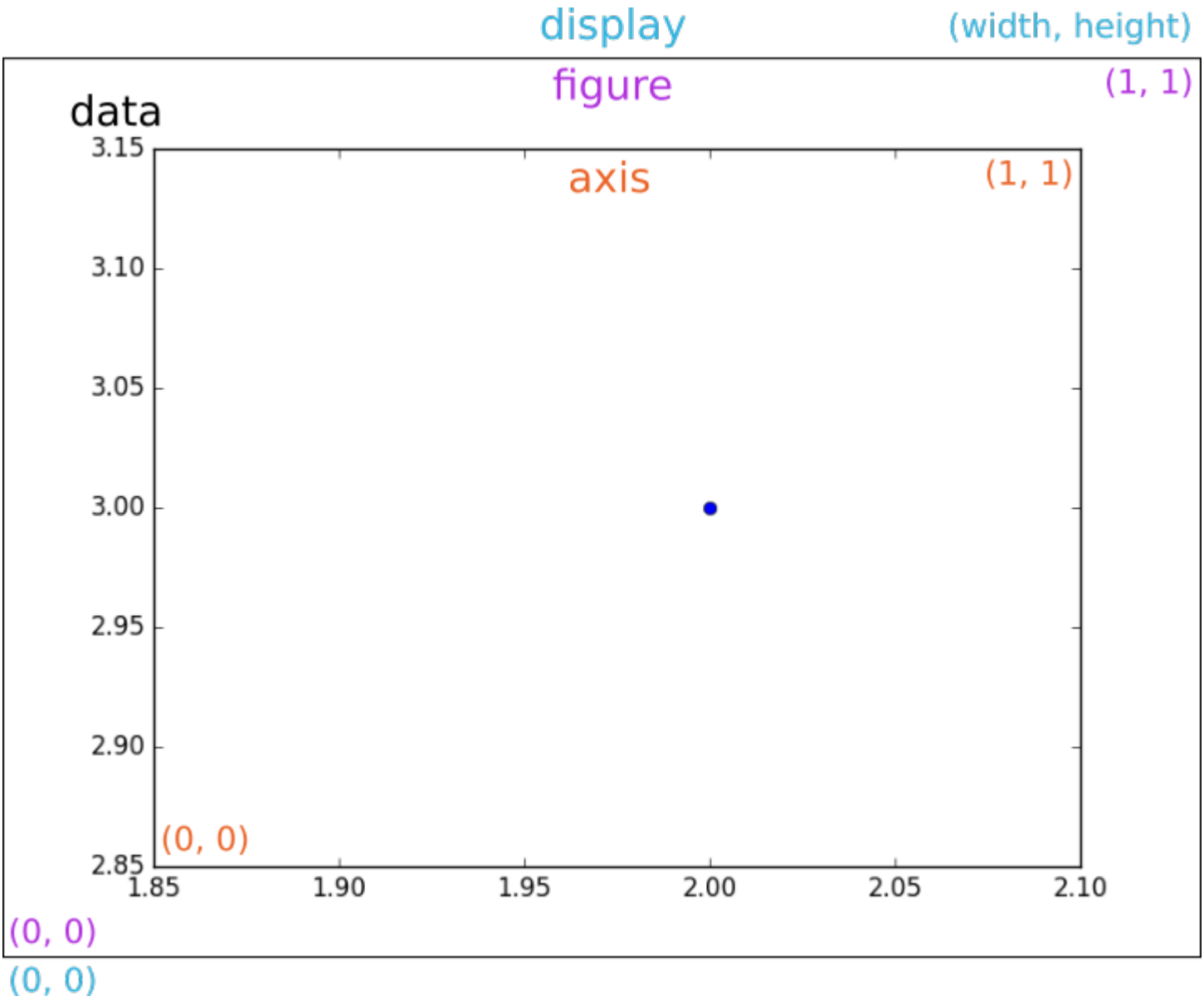
Matplotlib dispose de quatre systèmes de coordonnées distincts pouvant être utilisés pour faciliter le positionnement de différents objets, par exemple du texte. Chaque système possède un objet de transformation correspondant qui transforme les coordonnées de ce système en ce que l'on appelle le système de coordonnées d'affichage.

Le système de coordonnées de données est le système défini par les données sur les axes respectifs. C'est utile lorsque vous essayez de positionner un objet par rapport aux données tracées. La plage est donnée par les propriétés `xlim` et `ylim` des `Axes` . Son objet de transformation correspondant est `ax.transData` .

Le système de coordonnées d'axes est le système lié à son objet `Axes` . Les points (0, 0) et (1, 1) définissent les coins inférieur gauche et supérieur droit des axes. En tant que tel, il est utile lors du positionnement par rapport aux axes, comme au centre du graphique. Son objet de transformation correspondant est `ax.transAxes` .

Le système de coordonnées de la figure est analogue au système de coordonnées des axes, sauf qu'il est lié à la `Figure` . Les points (0, 0) et (1, 1) représentent les coins inférieur gauche et supérieur droit de la figure. C'est utile lorsque vous essayez de positionner quelque chose par rapport à l'image entière. Son objet de transformation correspondant est `fig.transFigure` .

Le système de coordonnées d'affichage est le système de l'image donné en pixels. Les points (0, 0) et (largeur, hauteur) sont les pixels inférieurs gauche et supérieur droit de l'image ou de l'affichage. Il peut être utilisé pour le positionnement absolument. Comme les objets de transformation transforment les coordonnées dans ce système de coordonnées, aucun objet de transformation n'est associé au système d'affichage. Cependant, `None` ou `matplotlib.transforms.IdentityTransform()` peut être utilisé si nécessaire.



Plus de détails sont disponibles ici .

Examples

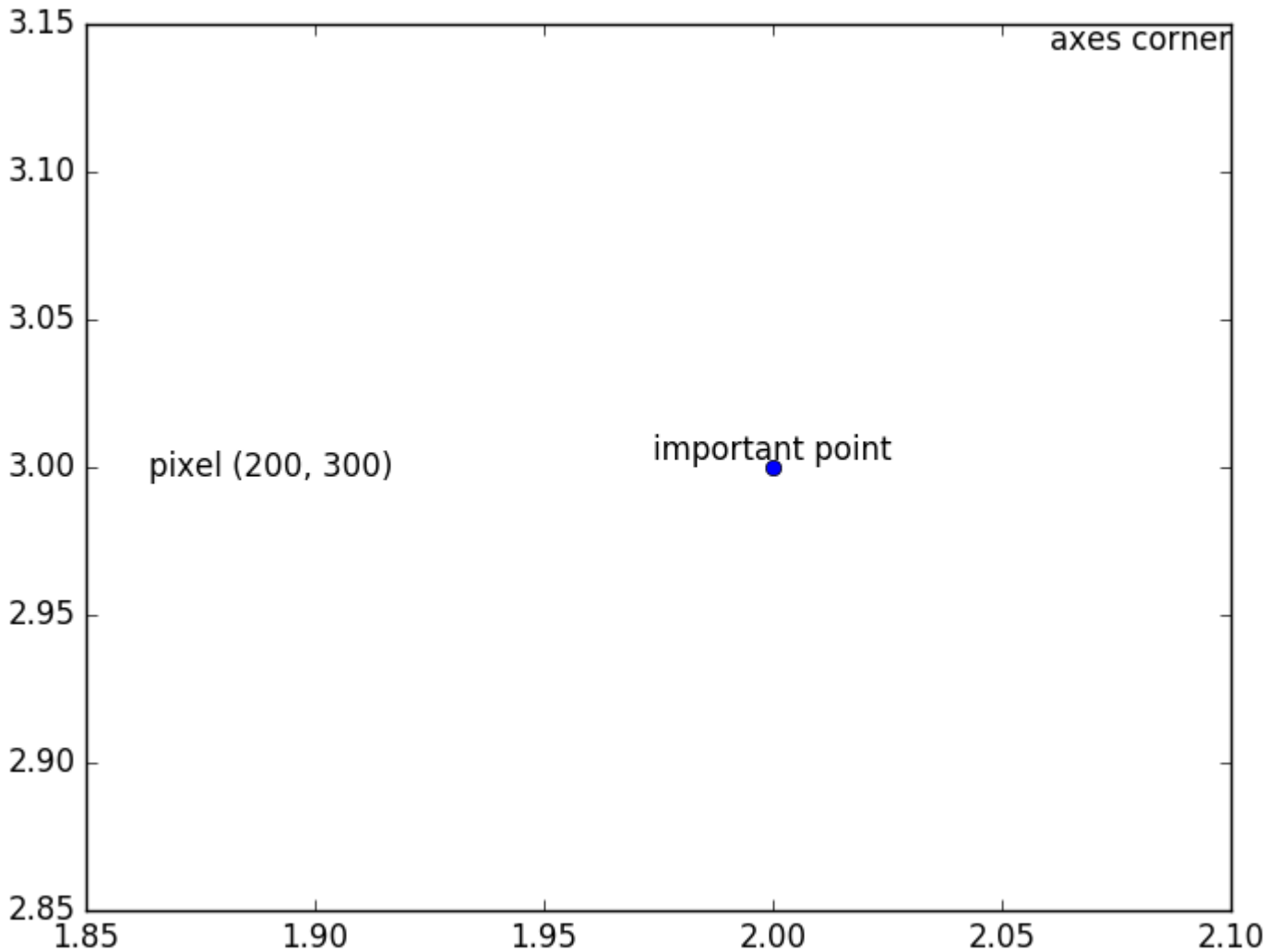
Systèmes de coordonnées et texte

Les systèmes de coordonnées de Matplotlib sont très utiles pour annoter les tracés que vous faites. Parfois, vous souhaitez placer le texte par rapport à vos données, comme lorsque vous tentez d'étiqueter un point spécifique. D'autres fois, vous voudrez peut-être ajouter un texte en haut de la figure. Cela peut facilement être réalisé en sélectionnant un système de coordonnées approprié en transmettant un objet de `transform` paramètre de `transform` dans l'appel à `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

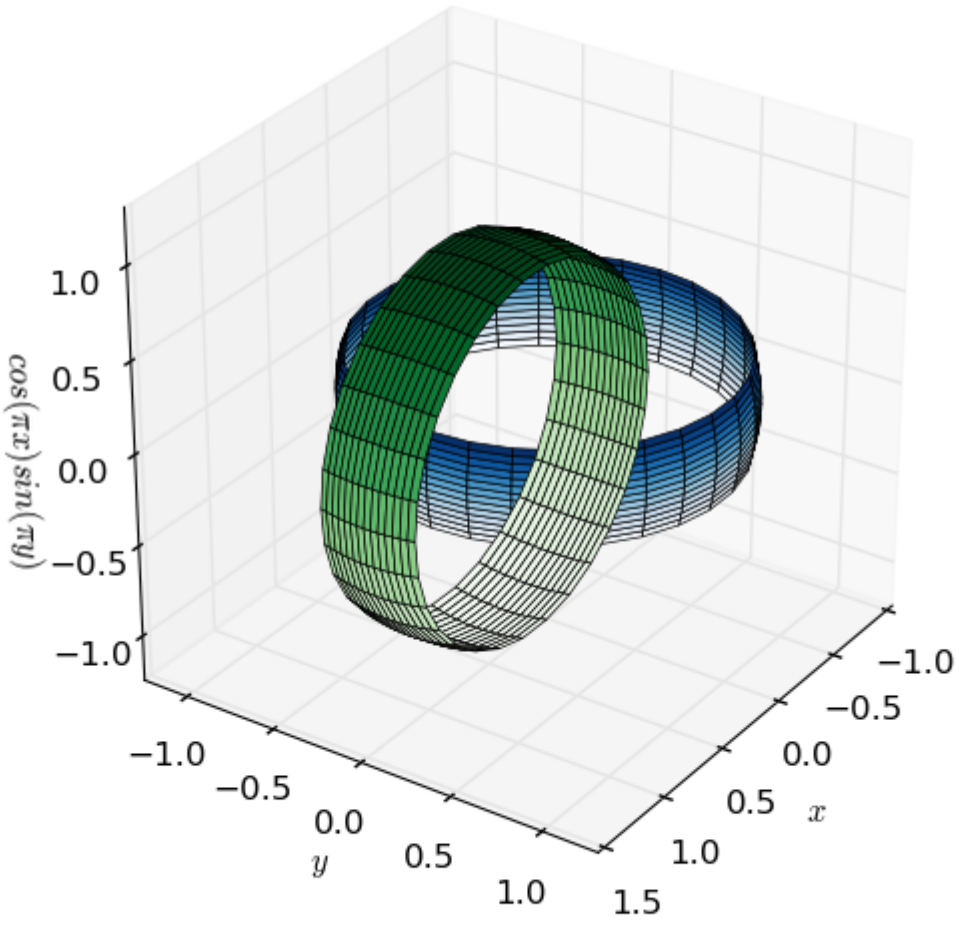


Lire Systèmes de coordonnées en ligne: https://riptutorial.com/fr/matplotlib/topic/4566/systemes-de-coordonnees

Chapitre 18: Tracés en trois dimensions

Remarques

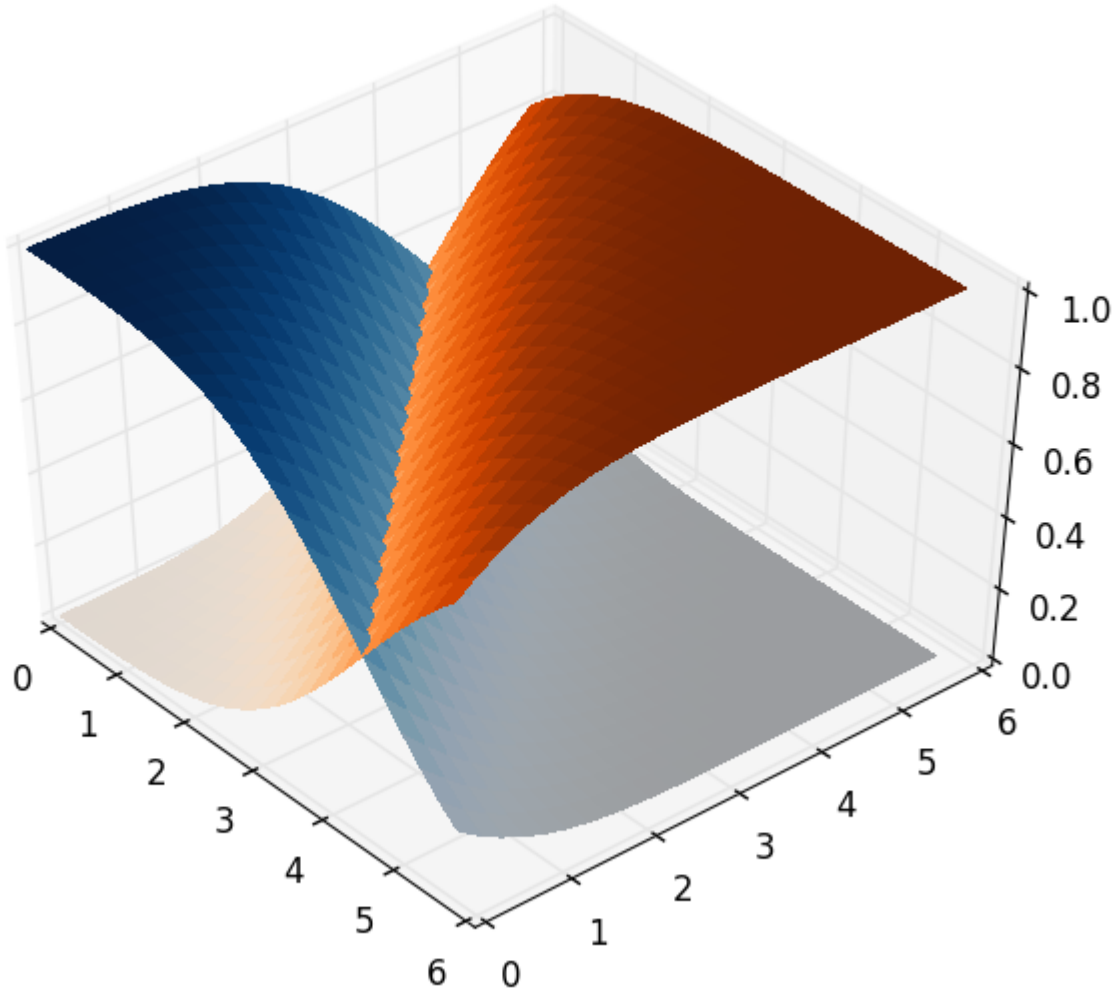
Le traçage tridimensionnel dans matplotlib a toujours été un peu compliqué, car le moteur de rendu est intrinsèquement 2d. Le fait que les configurations 3D soient rendues en traçant un segment 2d après l'autre implique que les problèmes liés à la profondeur apparente des objets sont souvent rendus . Le cœur du problème est que deux objets non connectés peuvent être soit complètement en arrière, soit complètement l'un en face de l'autre, ce qui conduit à des artefacts comme le montre la figure ci-dessous de deux anneaux entrelacés (cliquez pour gif animé):

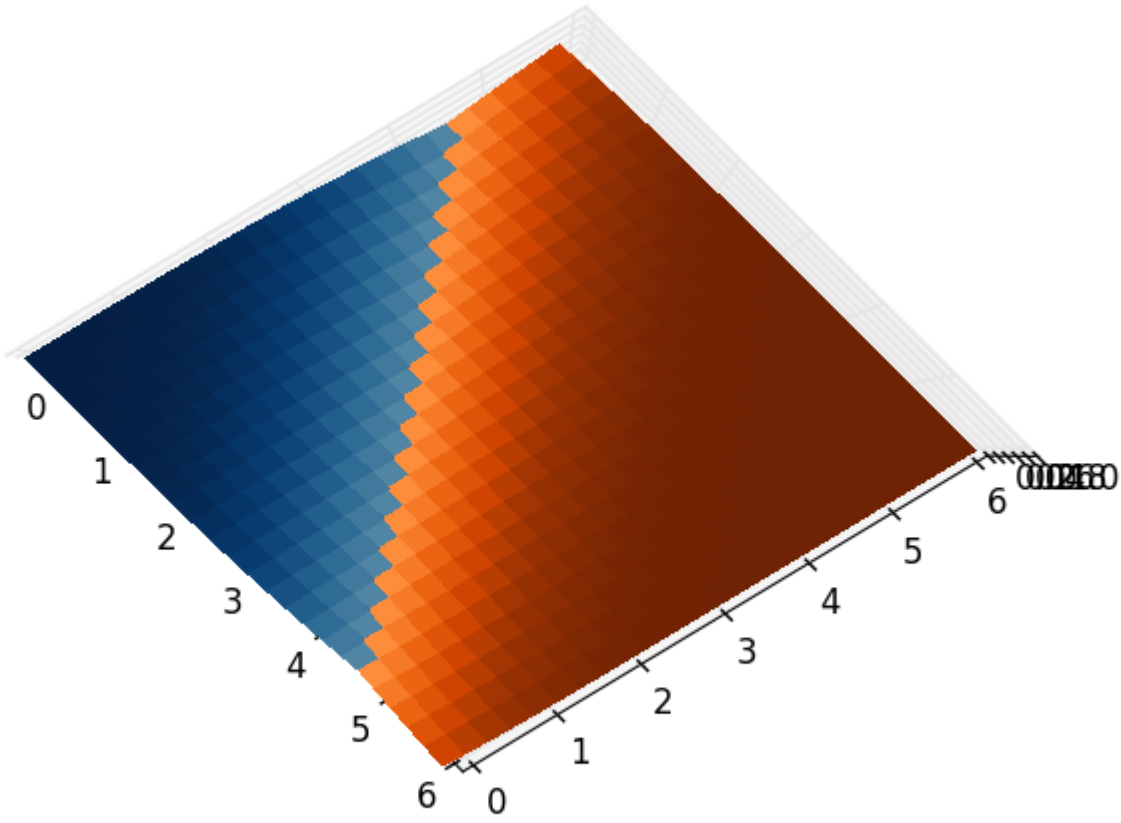


Cela peut cependant être corrigé. Cet artefact n'existe que lors du traçage de plusieurs surfaces sur le même tracé, chacune étant représentée sous la forme d'une forme 2D plate, avec un seul paramètre déterminant la distance de vue. Vous remarquerez qu'une seule surface compliquée ne souffre pas du même problème.

La manière de remédier à cela consiste à joindre les objets du tracé en utilisant des ponts transparents:

```
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

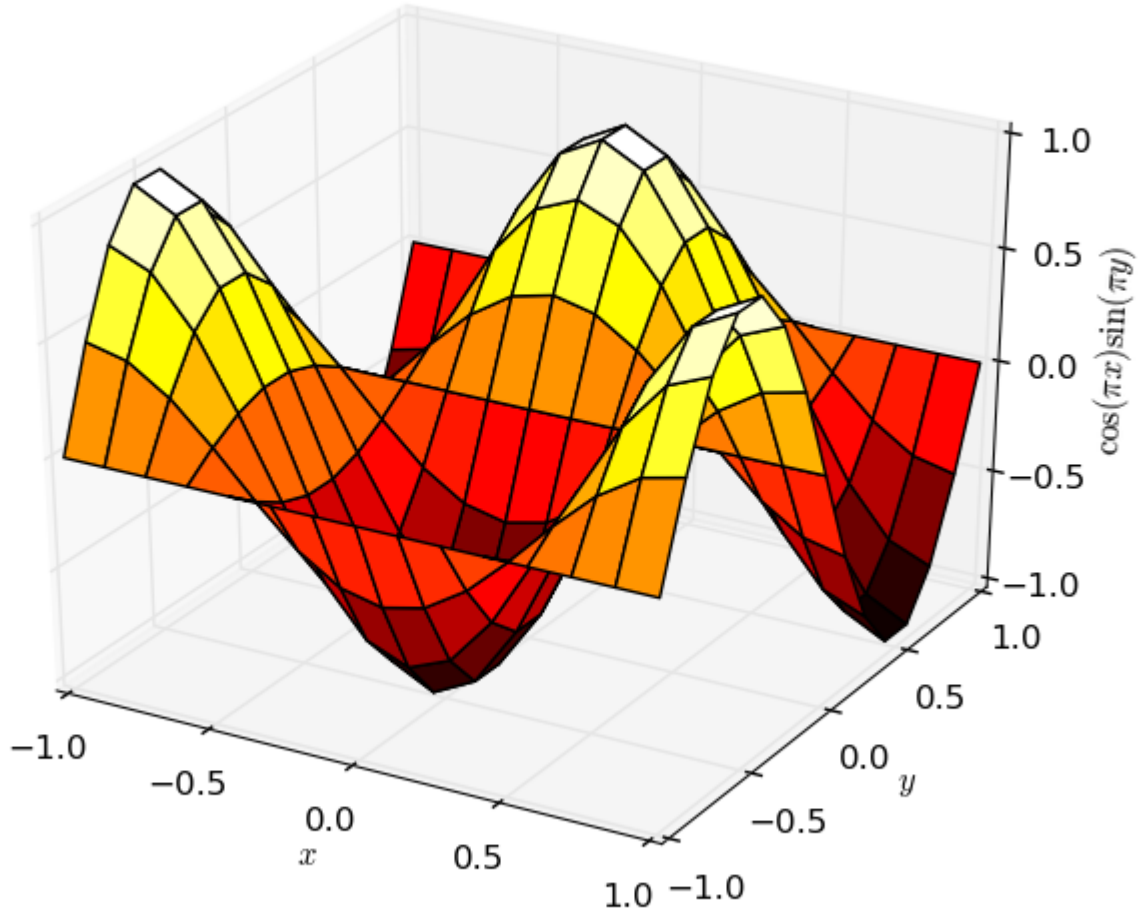
Création d'axes tridimensionnels

Les axes Matplotlib sont bidimensionnels par défaut. Afin de créer des tracés en trois dimensions, nous devons importer la classe `Axes3D` partir de la boîte à outils mplot3d , ce qui permettra un nouveau type de projection pour un axe, à savoir `'3d'` :

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

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

Outre les généralisations simples des tracés bidimensionnels (comme les tracés de lignes , les diagrammes de dispersion, les tracés de barres , les tracés de contour), plusieurs méthodes de tracé de surface sont disponibles, par exemple `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()
```



Lire Tracés en trois dimensions en ligne: https://riptutorial.com/fr/matplotlib/topic/1880/traces-en-trois-dimensions

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec matplotlib	Amitay Stern, ChaoticTwist, Chr, Chris Mueller, Community, dermen, evtoh, farenorth, Josh, jrjc, pmos, Serenity, tacaswell
2	Animations et traçage interactif	FiN, smurfendrek123, user2314737
3	Boîtes à moustaches	Luis
4	Cartes de contour	Eugene Loy, Serenity
5	Colormaps	Andras Deak, Xevaquor
6	Fermer une fenêtre de figure	Brian, David Zwicker
7	Histogramme	Yegor Kishilov
8	Intégration avec TeX / LaTeX	Andras Deak, Bosoneando, Chris Mueller, Næreen, Serenity
9	Légendes	Andras Deak, Franck Dernoncourt, ronrest, saintsfan342000, Serenity
10	Lignes de quadrillage et repères	ronrest
11	LogLog Graphing	ml4294
12	Manipulation d'image	Bosoneando
13	Objets de figures et d'axes	David Zwicker, Josh, Serenity, tom
14	Parcelles de base	Franck Dernoncourt, Josh, ml4294, ronrest, Scimonster, Serenity, user2314737
15	Plusieurs parcelles	Chris Mueller, Robert Branam, ronrest, swatchai
16	Systèmes de coordonnées	jure
17	Tracés en trois	Andras Deak, Serenity, will

	dimensions	