L1 - Calcul Scientifique

Youssef Chahir
youssef.chahir@unicaen.fr

Université de Caen Normandie

Chapitre III : Matplotlib
une bibliothèque Python
pour la visualisation de données
et le dessin de graphiques

Rappel : Commandes magiques Jupyter
commence par un %

- %matplotlib inline & %matplotlib notebook
  - matplotlib inline: la sortie est affichée directement sous la cellule de code qui l'a produite pour éviter pop-ups
  - matplotlib notebook : La sortie d'une commande est dans une fenêtre séparée.
- **%pylab inline :**
  - Importe les modules numpy et matplotlib.
  - L'option inline indique que les figures Matplotlib seront insérées **dans le notebook** lui-même plutôt que dans une fenêtre graphique à part.
- %autosave 300 : sauvegarde automatiquement le notebook tous les 300 secondes (=5 minutes)

Wikipedia

Matplotlib est une bibliothèque du langage de programmation Python destinée à **tracer** et **visualiser** des *données sous forme de graphiques*. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy. (…)

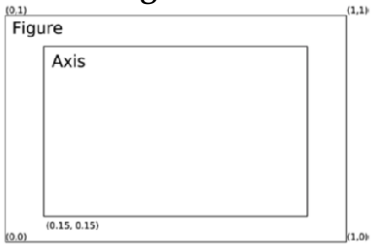
- Plusieurs points rendent cette bibliothèque intéressante :
  - Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...)
  - Documentation en ligne en quantité, nombreux exemples disponibles sur internet
  - Forte communauté très active
  - Interface pylab : reproduit fidèlement la syntaxe MATLAB
  - Bibliothèque haut niveau : idéale pour le calcul interactif

Introduction

- Installation
  - pip3 install matplotlib
  - http://www.matplotlib.org
- Importation des modules dans un programme python :

```
In [1]: %matplotlib inline
In [2]: import matplotlib as mpl
In [3]: import matplotlib.pyplot as plt
In [4]: from mpl_toolkits.mplot3d.axes3d import Axes3D
```

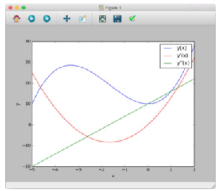
- Matplotlib crée des '**figures**' contenant des '**Axes**'



Utilisation de différents backends

- Matplotlib peut fonctionner sous différentes plateformes, avec différents environnements graphiques
- Peut par exemple générer des graphiques au format : PNG, PDF, Postscript, ou SVG
- Travailler dans différents environnements graphiques comme : Qt, GTK, wxWidgets ou Cocoa pour Mac OS X
- Choix avec la fonction **mpl.use** appelée juste après l'import

```
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.use('qt4agg') ou mpl.use('agg')
```



Utilisation de différents backends (bis)

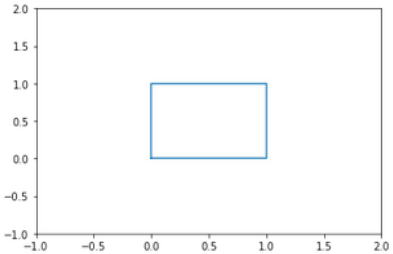
- Lorsque l'on travaille dans des notebook (jupyter), il est préférable d'utiliser des graphiques inclus au notebook, ce qui peut se faire grâce à la commande :
  - %**matplotlib inline**
- Par défaut les graphiques sont générés en format png, mais possibilité de changer :
  - %config InlineBackend.figure_format='svg'
- En mode interactif, il est nécessaire d'appeler les fonctions :
  - plt.show et plt.draw pour effectuer le rendu,
  - Ce n'est pas nécessaire dans les notebooks car il est fait automatiquement.

Premier graphique dans une figure

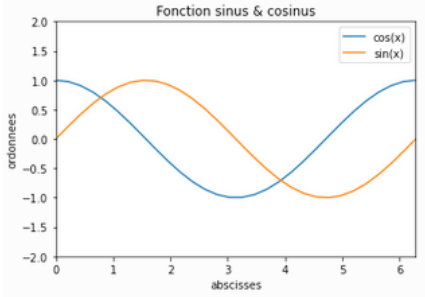
- La création d'une figure se fait par la commande : `plt.figure()` qui prend différents arguments optionnels parmi lesquels :
  - `figsize=(width, height)` : taille de la figure, unités : inches
  - `facecolor="#f1f1f1"` : couleur du 'fond' de la figure
  - Une fois la figure créée : `plt.plot` permet de faire le tracé
- L'instruction `plot()` permet de tracer des courbes qui relient des points dont les abscisses et ordonnées sont fournies dans des tableaux
  - x = array([1, 3, 4, 6]); y = array([2, 3, 5, 1]); plt.plot(x, y, label='…')
  - plt.plot(x) : uniquement ds ordonnée ==> la lib remplit la valeur des abscisses d'elle même (à partir de 0 .. n-1)
- Définition du domaine des axes : `xlim(xmin,xmax)` (axe des abscisses) et `ylim(ymin,ymax)` (axe des ordonnées)
- Ajout d'un titre : `title()`
- Ajout d'une légende : `legend()`
- Labels sur les axes : `xlabel()` et `ylabel()`
- `Délimiter les limites du graphique : axis(xmin, xmax, ymin, ymax)`

Premier graphique avec plot

```
x = np.array([0, 1, 1, 0, 0])
y = np.array([0, 0, 1, 1, 0])
plt.plot(x, y)
plt.xlim(-1, 2)
plt.ylim(-1, 2)

plt.show()
```



```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1, label="cos(x)")
plt.plot(x, y2, label="sin(x)")
plt.xlabel("abscisses")
plt.ylabel("ordonnees")
plt.xlim(0, 2*np.pi)
plt.ylim(-2, 2)
plt.legend()
plt.title("Fonction sinus & cosinus")
plt.show()
```

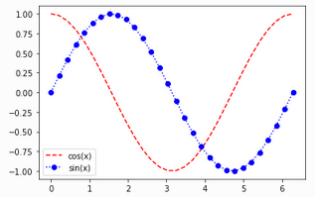


`np.linspace(A,B,NB)` permet d'obtenir un tableau 1D allant d'une valeur de départ à A une valeur de fin B avec un nombre donné d'éléments NB.

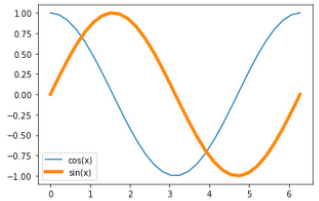
Changer le visuel

- Il est possible de préciser la couleur, le style de ligne et de symbole (« marker ») en ajoutant une chaîne de caractères de la façon suivante :
  - Courbes possibles: - -- -. :
  - Couleurs possibles: b g r c m y k w
- On peut modifier la largeur des lignes, avec `linewidth` (unités en points)

```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1, "r--", label="cos(x)")
plt.plot(x, y2, "b:o", label="sin(x)")
plt.legend()

plt.show()
```




```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1, label="cos(x)")
plt.plot(x, y2, label="sin(x)", linewidth=4)
plt.legend()

plt.show()
```



Si on ne veut pas faire apparaître de ligne, il suffit d'indiquer un symbole sans préciser de style de ligne.

Style de ligne + Symbol « marker » + Couleur

Chaîne	Effet
-	ligne continue
--	tirets
:	ligne en pointillé
-.	tirets points

Chaîne	Couleur en anglais	Couleur en français
b	blue	bleu
g	green	vert
r	red	rouge
c	cyan	cyan
m	magenta	magenta
y	yellow	jaune
k	black	noir
w	white	blanc

Chaîne	Effet
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker
|	vline marker
_	hline marker

Propriétés des lignes

Argument	Example values	Description
color	A color specification can be a string with a color name, such as "red," "blue," etc., or a RGB color code on the form "#aabbcc."	A color specification.
alpha	Float number between 0.0 (completely transparent) to 1.0 (completely opaque).	The amount of transparency.
linewidth, lw	Float number.	The width of a line.
linestyle, ls	'-' – solid '--' – dashed ':' – dotted '.-' – dash-dotted	The style of the line, i.e., whether the line is to be draw as a solid line, or if it should be, for example, dotted or dashed.
marker	+, o, * = cross, circle, star s = square . = small dot 1, 2, 3, 4, . . . = triangle-shaped symbols with different angles.	Each data point, whether or not it is connected with adjacent data points, can be represented with a marker symbol as specified with this argument.
markersize	Float number.	The marker size.
makerfacecolor	Color specification (see above).	The fill color for the marker.
markeredgewidth	Float number.	The line width of the marker edge.
markeredgecolor	Color specification (see above).	The marker edge color.

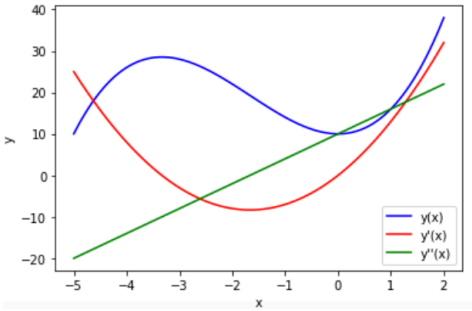
Créer une Figure (mode simple)

- La création d'une figure se fait par la commande : `plt.figure()` qui prend différents arguments optionnels parmi lesquels :
  - `figsize=(width, height)` : taille de la figure, unités : inches
  - `facecolor="#f1f1f1"` : couleur du 'fond' de la figure
- Une fois la figure créée : `plt.plot` permet de faire le tracé

Simple plot

```
x = np.linspace(-5, 2, 100)
y1 = x**3 + 5*x**2 + 10
y2 = 3*x**2 + 10*x
y3 = 6*x + 10
fig, ax = plt.subplots()
ax.plot(x, y1, color="blue", label="y(x)")
ax.plot(x, y2, color="red", label="y'(x)")
ax.plot(x, y3, color="green", label="y''(x)")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x12ade6790>
```

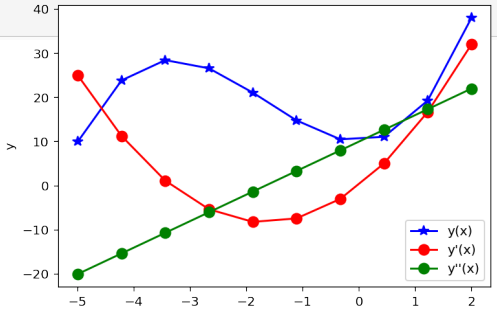


Propriétés des lignes

Argument	Example values	Description
color	A color specification can be a string with a color name, such as "red," "blue," etc., or a RGB color code on the form "#aabbcc."	A color specification.
alpha	Float number between 0.0 (completely transparent) to 1.0 (completely opaque).	The amount of transparency.
linewidth, lw	Float number.	The width of a line.
linestyle, ls	'-' – solid '--' – dashed ':' – dotted '.-' – dash-dotted	The style of the line, i.e., whether the line is to be draw as a solid line, or if it should be, for example, dotted or dashed.
marker	+, o, * = cross, circle, star s = square . = small dot 1, 2, 3, 4, ... = triangle-shaped symbols with different angles.	Each data point, whether or not it is connected with adjacent data points, can be represented with a marker symbol as specified with this argument.
markersize	Float number.	The marker size.
makerfacecolor	Color specification (see above).	The fill color for the marker.
markeredgewidth	Float number.	The line width of the marker edge.
markeredgecolor	Color specification (see above).	The marker edge color.

Texte du titre

```
x = np.linspace(-5, 2, 10)
y1 = x**3 + 5*x**2 + 10
y2 = 3*x**2 + 10*x
y3 = 6*x + 10
fig, ax = plt.subplots()
ax.plot(x, y1, color="blue", label="y(x)",marker="*",markersize=8)
ax.plot(x, y2, color="red", label="y'(x)",marker="o",markersize=8)
ax.plot(x, y3, color="green",
label="y''(x)",ls='-',marker="o",markersize=8)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.legend()
```

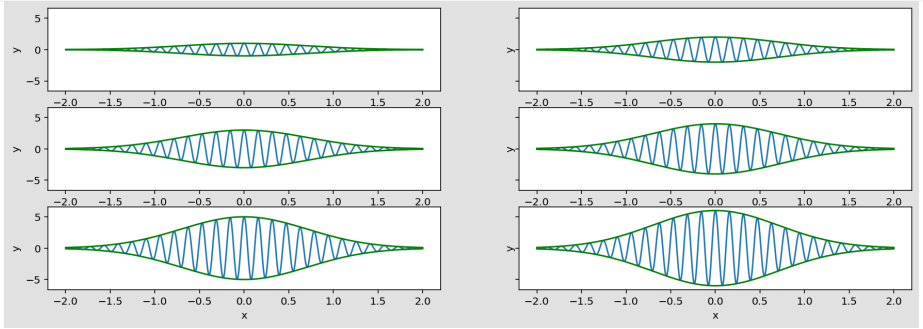


Création simplifiée des axes

- La méthode précédente permet de créer des zones de tracés en contrôlant précisément les positions, mais nécessite de calculer les tailles.
- Possibilité de rendre le travail plus simple avec `subplot` : `fig, axes = plt.subplots(nrows=3, ncols=2)`
- Tableau de graphiques (ncols, nrows)
- Possibilité de partager les coordonnées x et y : argument sharex et sharey à True/False
- Arguments fig_kw et subplot_kw (sous forme de dictionnaires) qui permettent de contrôler toutes les propriétés des figures et des axes subplot_kw={'facecolor': "#ebf5ff"}

Exemple

```
fig,axes = plt.subplots(nrows=3, ncols=2, facecolor="#e1e1e1",figsize=(15, 5), sharey=True)
x = np.linspace(-2, 2, 1000)
y1 = np.cos(40 * x)
y2 = np.exp(-x**2)
i=1
for ax in axes.flatten():
    ax.plot(x, i*y1 * y2)
    ax.plot(x, i*y2, 'g')
    ax.plot(x, -y2*i, 'g')
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    i=i+1
```

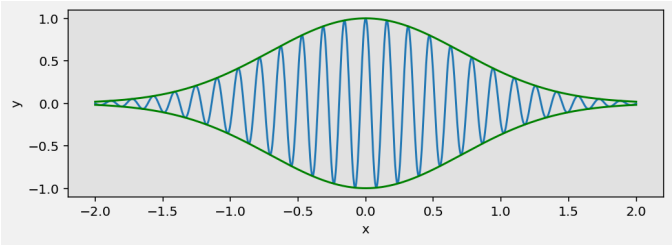


Créer une figure (meilleur contrôle des zones)

- La fonction **`add_axes()`** permet un contrôle plus avancé
- Une fois la figure créée, ajout d'**`axes`** (zones de tracé) : Appel à la fonction **`ax = add_axes()`** avec comme arguments :
  - La taille sous forme (left, bottom, width, height)
  - La couleur du fond : `facecolor=« #e1e1e1"`
  - Nombreux autres arguments que nous verrons plus loin
- Le tracé se fait avec : **ax.plot()**

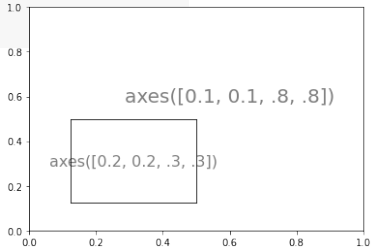
Exemple

```
fig = plt.figure(figsize=(8, 2.5), facecolor="#f1f1f1")
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes((left, bottom, width, height), facecolor="#e1e1e1")
x = np.linspace(-2, 2, 1000)
y1 = np.cos(40 * x)
y2 = np.exp(-x**2)
ax.plot(x, y1 * y2)
ax.plot(x, y2, 'g')
ax.plot(x, -y2, 'g')
ax.set_xlabel("x")
ax.set_ylabel("y")
fig.savefig("graph.png", dpi=100, facecolor="#f1f1f1")
```



Axes

```
plt.axes([.1, .1, .8, .8])

plt.text(.6, .6, 'axes([0.1, 0.1, .8, .8])',
         ha='center', va='center',
         size=20, alpha=.5)
plt.axes([.2, .2, .3, .3])
plt.xticks(())
plt.yticks(())
plt.text(.5, .5, 'axes([0.2, 0.2, .3, .3])',
         ha='center', va='center',
         size=16, alpha=.5)
plt.show()
```



Propriétés des axes et titres(I)

- Label d'un axe : plt.xlabel(label, fontdict=None, labelpad=None, **kwargs)

Nom	Description
label	texte des étiquettes
fontdict	dictionnaire des polices de texte des étiquettes, comme la famille, la couleur, le po taille
labelpad	Espacement en points entre l'étiquette et l'axe des x

01. `Times(s)`
Voici le texte de l'étiquette de l'axe des x

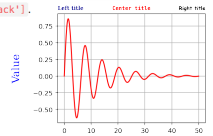
02. `family='serif'`
Il spécifie que la famille de polices du texte de l'étiquette doit être `serif`. Vous pouvez choisir la famille parmi les options populaires comme `[ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]`.

03. `color='r'`
Le texte de la police a la couleur du rouge. Reportez-vous à l'option de couleur dans le dernier chapitre pour choisir d'autres couleurs.

04. `weight = 'normal'`
Il spécifie que le texte de l'étiquette doit avoir un poids normal. L'option de poids est `['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']`.

05. `size=16`
Il attribue la taille de la police à 16.

06. `labelpad = 6`
La distance entre l'axe des x et l'étiquette est de 6 points.

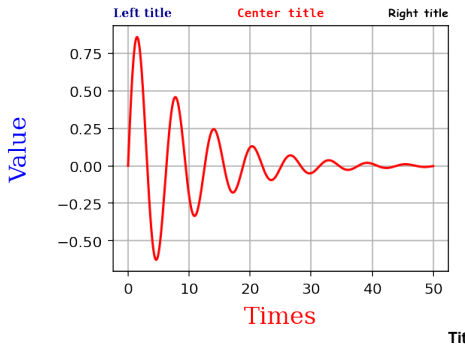



```
x = np.linspace(0, 50, 500)
y = np.sin(x) * np.exp(-x/10)
plt.figure(figsize=(4, 3))
plt.plot(x, y, "r")
plt.xlabel("Times",
           family='serif',
           color='r',
           weight='normal',
           size = 16,
           labelpad = 6)
plt.ylabel("Value",
           family='serif',
           color='b',
           weight='normal',
           size = 16,
           labelpad = 16)
plt.title("Left title",
          fontdict={'family': 'serif',
                    'color' : 'darkblue',
                    'weight': 'bold',
                    'size': 8},
          loc='left')
plt.title("Center title",
          fontdict={'family': 'monospace',
                    'color' : 'red',
                    'weight': 'bold',
                    'size': 8},
          loc='center')
plt.title("Right title",
          fontdict={'family': 'fantasy',
                    'color' : 'black',
                    'weight': 'bold',
                    'size': 8},
          loc='right')
plt.grid(True)
plt.show()
```


Propriétés des axes et titres(II)

- Titre des axes : plt.title(label, fontdict=None, loc=None=None, **kwargs)

Nom	Type de données	Description
label	str	texte des étiquettes
fontdict	dict	dictionnaire des polices de texte des étiquettes, comme la famille, la couleur, le poids et la taille
loc	str	L'emplacement du titre. Il y a trois options, `center`, `left`, `right` et l'option par défaut est `center`.



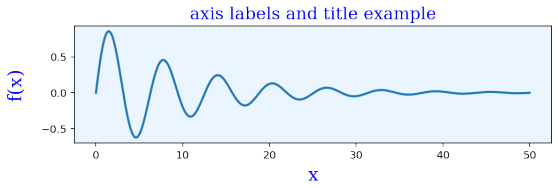

```
x = np.linspace(0, 50, 500)
y = np.sin(x) * np.exp(-x/10)
plt.figure(figsize=(4, 3))
plt.plot(x, y, "r")
plt.xlabel("Times",
           family='serif',
           color='r',
           weight='normal',
           size = 16,
           labelpad = 6)
plt.ylabel("Value",
           family='serif',
           color='b',
           weight='normal',
           size = 16,
           labelpad = 16)
plt.title("Left title",
          fontdict={'family': 'serif',
                    'color' : 'darkblue',
                    'weight': 'bold',
                    'size': 8},
          loc='left')
plt.title("Center title",
          fontdict={'family': 'monospace',
                    'color' : 'red',
                    'weight': 'bold',
                    'size': 8},
          loc='center')
plt.title("Right title",
          fontdict={'family': 'fantasy',
                    'color' : 'black',
                    'weight': 'bold',
                    'size': 8},
          loc='right')
plt.grid(True)
plt.show()
```

Titre à l'intérieur: plt.title("Title", position=(0.5,0.9), fontdict=0...)
0<pos<1 ==>**Coin inférieur:(0,0) et coin supérieur : (1,1)**

Propriétés des axes et titres(III)

- set_xlabel et set_ylabel permettent de mettre une légende sur les axes du graphique labelpad (espacement en points entre l'étiquette et l'axe des x) color, fontsize et fontname également disponibles
- set_title() permet de spécifier le titre

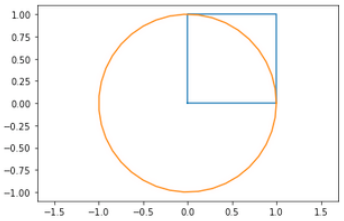
```
x = np.linspace(0, 50, 500)
y = np.sin(x) * np.exp(-x/10)
fig, ax = plt.subplots(figsize=(8, 2), subplot_kw={'facecolor':
"#ebf5ff"})
ax.plot(x, y, lw=2)
ax.set_xlabel("x", labelpad=5, fontsize=18, fontname='serif',
color="blue")
ax.set_ylabel("f(x)", labelpad=15, fontsize=18, fontname='serif',
color="blue")
ax.set_title("axis labels and title example", fontsize=16,
fontname='serif', color="blue")
```



Tracé de formes

- L'instruction `axis("equal")` permet d'avoir la même échelle sur l'axe des abscisses et l'axe des ordonnées afin de préserver la forme lors de l'affichage.

```
x1 = np.array([0, 1, 1, 0, 0])
y1 = np.array([0, 0, 1, 1, 0])
plt.plot(x1, y1)

plt.axis("equal")


plt.xlim(-1, 2)
plt.ylim(-1, 2)
theta = np.linspace(0, 2*np.pi, 40)

x2 = np.cos(theta)
y2 = np.sin(theta)
plt.plot(x2, y2)
plt.axis("equal")

plt.show()
```

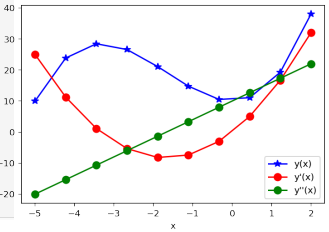


Grâce à cette commande un carré/cercle apparaît vraiment comme un carré/cercle.

Légende

- Possibilité d'ajouter des légendes pour chaque courbe: avec l'argument : label="y(x)" puis l'appel à la fonction ax.legend()
- Possibilité de positionner la légende dans la figure avec l'argument
  - loc : loc=1, loc=2, loc=2 ou loc=4 pour les 4 coins de l'axe
  - bbox_to_anchor =(x,y) pour un positionnement à un endroit spécifié par ses coordonnées

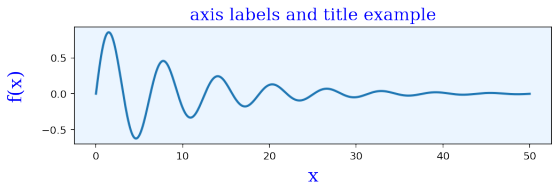
```
x = np.linspace(-5, 2, 10)
y1 = x**3 + 5*x**2 + 10
y2 = 3*x**2 + 10*x
y3 = 6*x + 10
fig, ax = plt.subplots()
ax.plot(x, y1, color="blue", label="y(x)",marker="*",markersize=8)
ax.plot(x, y2, color="red", label="y'(x)",marker="o",markersize=8)
ax.plot(x, y3, color="green",
label="y''(x)",ls='-',marker="o",markersize=8)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.legend()
```



Propriétés des axes et titres

- set_xlabel et set_ylabel permettent de mettre une légende sur les axes du graphique labelpad (espacement en points entre l'étiquette et l'axe des x) color, fontsize et fontname également disponibles
- set_title() permet de spécifier le titre

```
x = np.linspace(0, 50, 500)
y = np.sin(x) * np.exp(-x/10)
fig, ax = plt.subplots(figsize=(8, 2), subplot_kw={'facecolor':
"#ebf5ff"})
ax.plot(x, y, lw=2)
ax.set_xlabel("x", labelpad=5, fontsize=18, fontname='serif',
color="blue")
ax.set_ylabel("f(x)", labelpad=15, fontsize=18, fontname='serif',
color="blue")
ax.set_title("axis labels and title example", fontsize=16,
fontname='serif', color="blue")
```



Formatage du texte

- On peut ajouter une grille (et des marqueurs) :
  - marqueurs possibles : o + . x * ^
- Possibilité d'ajouter du texte aux figures, y compris avec du formatage latex "Regular text: $f(x)=1-x^2$"
- Ajout de texte : plt.text /ax.text() permet d'ajouter du texte en indiquant les coordonnées:
- Annotations : ax.annotate() + autres: ex. une flèche descriptiv

Argument	Description
fontsize	The size of the font, in points.
family	The font type.
backgroundcolor	Color specification for the background of the text label.
color	Color specification for the font color.
alpha	Transparency of the font color.
rotation	Rotation angle of the text label.

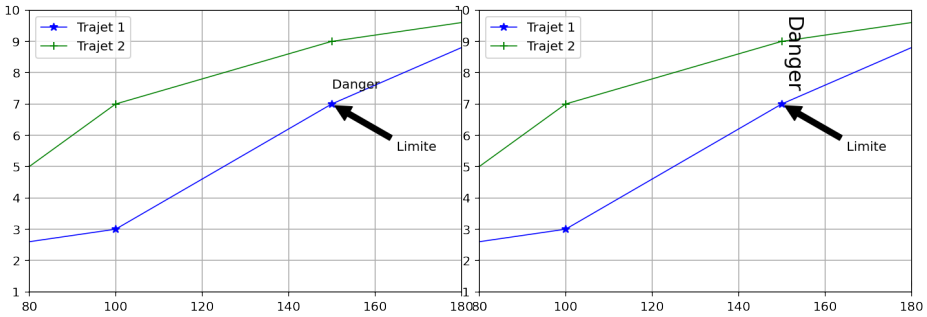
Affichage de la grille

- L'affichage de la grille s'obtient par : axes.grid()
- La grille suit les **ticks** (les graduations)
- Possibilité de fixer les propriétés : color, linestyle et linewidth
  - ax.grid(color="grey", which="major", axis='x', linestyle='-', linewidth=0.5)
- Courbes en log :
  - loglog, semilogx, et semilogy (remplace plot)
  - Twin axes : possibilité de mettre deux gradations sur la même figure (gauche et droite)n, pour deux courbes distinctes
  - Tracer de la première courbe (ax1) :
    - ax2 = ax1.twinx()
    - puis tracé de la seconde : ax2.plot(…)

Exemple : Annotation dans une grille

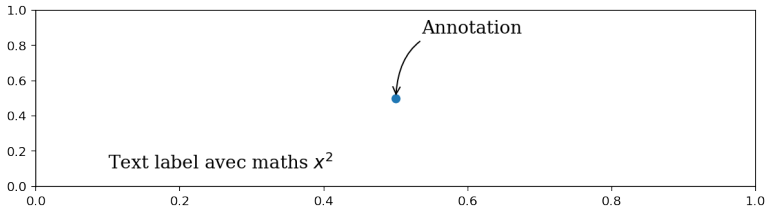
```
x = array([1, 3, 4, 6])
y = array([2, 3, 5, 1])
plt.grid(True)
plt.axis([80, 180, 1, 10])
plt.plot([50,100,150,200], [2,3,7,10], "b",
         linewidth=0.8, marker="*",label="Trajet 1")
plt.plot([50,100,150,200], [2,7,9,10], "g",
         linewidth=0.8, marker="+",label="Trajet 2")

plt.text(150, 7.5, "Danger")

plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.annotate('Limite', xy=(150, 7), xytext=(165, 5.5),
arrowprops={'facecolor':'black', 'shrink':0.05} )
plt.legend()
plt.show()
```

```
x = array([1, 3, 4, 6])
y = array([2, 3, 5, 1])
plt.grid(True)
plt.axis([80, 180, 1, 10])
plt.plot([50,100,150,200], [2,3,7,10], "b",
         linewidth=0.8, marker="*",label="Trajet 1")
plt.plot([50,100,150,200], [2,7,9,10], "g",
         linewidth=0.8, marker="+",label="Trajet 2")

plt.text(150, 7.5, "Danger",fontsize=16, rotation = 270)
plt.xlabel('Vitesse')
plt.ylabel('Temps')
plt.annotate('Limite', xy=(150, 7), xytext=(165, 5.5),
arrowprops={'facecolor':'black', 'shrink':0.05} )
plt.legend()
plt.show()
```



Exemple

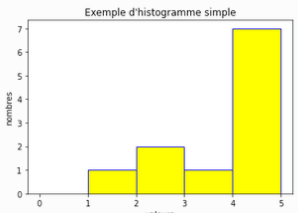
```
fig, ax = plt.subplots(figsize=(10, 2.5))
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.plot(.5, .5, "o")
ax.annotate("Annotation",
fontsize=14, family="serif",
xy=(.5, .5), xycoords="data",
xytext=(+20, +50), textcoords="offset points",
arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.5"))
ax.text(0.1, 0.1, "Text label avec maths $x^2$", fontsize=14,family="serif")
```



Histogramme

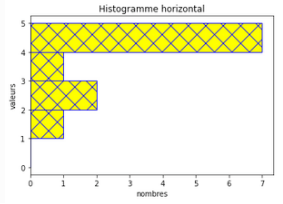
- La méthode `plt.hist` permet de créer un **histogramme** .
  - On peut lui donner des données brutes et il s'occupera de faire les calculs nécessaires à la présentation du graphique.
  - La valeur retournée est un triplet avec :
    - une array donnant les nombres de points dans chaque bin
    - une array donnant les limites de chaque bin
    - une array de 5 objets matplotlib.patches.Rectangle.
  - si on passe l'argument density = True, au lieu d'avoir les nombres de points, on a les fréquences normalisées dont la somme vaut 1.

```
x = [1, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5]
pyplot.hist(x, range = (0, 5), bins = 5, color = 'yellow', edgecolor = 'blue')
pyplot.xlabel('valeurs')
pyplot.ylabel('nombres')
pyplot.title('Exemple d\'histogramme simple')


plt.show()
```




```
x = [1, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5]
pyplot.hist(x, range = (0, 5), bins = 5, color = 'yellow',
            edgecolor = 'blue', hatch = 'x',
            orientation = 'horizontal')
pyplot.ylabel('valeurs')
pyplot.xlabel('nombres')
pyplot.title('Histogramme horizontal')
```

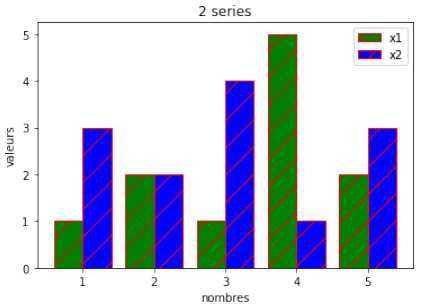
```
Text(0.5, 1.0, 'Histogramme horizontal')
```

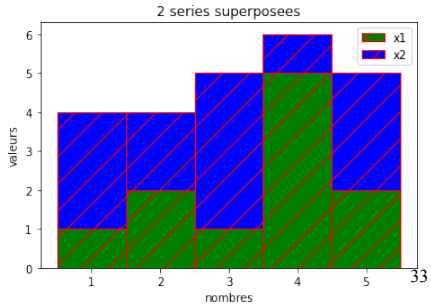


Affichage de plusieurs histogrammes

```
x1 = [1, 2, 2, 3, 4, 4, 4, 4, 5, 5]
x2 = [1, 1, 1, 2, 2, 3, 3, 3, 4, 5, 5, 5]
bins = [x + 0.5 for x in range(0, 6)]
pyplot.hist([x1, x2], bins = bins, color = ['green', 'blue'],
            edgecolor = 'red', hatch = '/', label = ['x1', 'x2'],
            histtype = 'bar') # bar est le defaut
pyplot.ylabel('valeurs')
pyplot.xlabel('nombres')
pyplot.title('2 series')
pyplot.legend()
```

```
x2 = [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5]
bins = [x + 0.5 for x in range(0, 6)]
pyplot.hist([x1, x2], bins = bins, color = ['green', 'blue'],
            edgecolor = 'red', hatch = '/', label = ['x1', 'x2'],
            histtype = 'barstacked')
pyplot.ylabel('valeurs')
pyplot.xlabel('nombres')
pyplot.title('2 series superposees')
pyplot.legend()
```

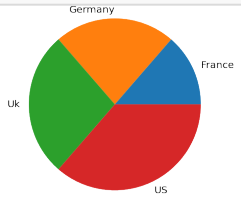




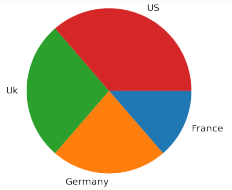
Graphique circulaire

- La méthode `pie permet de` créer un camembert
  - plt .pie(x, explode=None, labels=None, colors=None, autopct=None, `pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=None, radius=None, counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, hold=None, data=None)`
  - Si l'argument `counterclock` est réglé sur `False`, alors le camembert sera dessiné dans le sens des aiguilles d'une montre.
  - Le paramètre `explode` contrôle l'éclatement des tranches dans les camemberts. Il spécifie la fraction du rayon avec laquelle chaque tranche est décalée.

```
x = np.array([15, 25, 30, 40])
label = ["France", "Germany", "Uk", "US"]

plt.pie(x, labels=label)
plt.show()
```

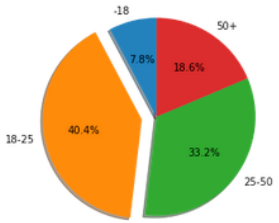



```
x = np.array([15, 25, 30, 40])
label = ["France", "Germany", "Uk", "US"]

plt.pie(x, labels=label, counterclock=False)
plt.show()
```

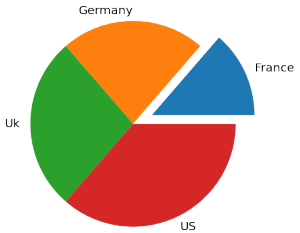


Graphique circulaire : explode

```
name = ['-18', '18-25', '25-50', '50+']
data = [5000, 26000, 21400, 12000]

explode=(0, 0.15, 0, 0)
plt.pie(data, explode=explode, labels=name, autopct='%1.1f%%', startangle=90, shadow=True)
plt.axis('equal')
plt.show()
```



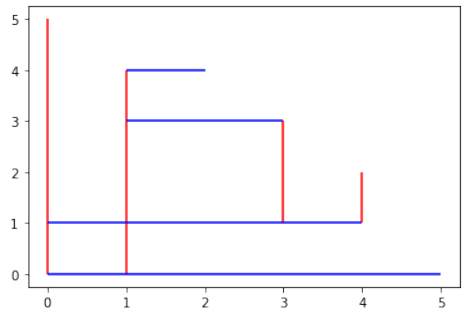

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt


x = np.array([15, 25, 30, 40])
label = ["France", "Germany", "Uk", "US"]

plt.pie(x, labels=label, explode=(0.2, 0, 0, 0))
plt.show()
```



Lignes

- plt.vlines / plt.hlines : Lignes verticales / horizontales :

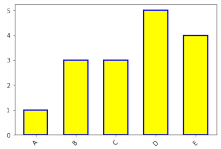
```
plt.vlines([0, 1, 3, 4], [0, 0, 1, 1], [5, 4, 3, 2],
  color = 'red', linestyle = 'solid', label = ['a', 'b', 'c', 'd'])
plt.hlines([0, 1, 3, 4], [0, 0, 1, 1], [5, 4, 3, 2],
  color = 'blue', linestyle = 'solid', label = ['a', 'b', 'c', 'd'])
```



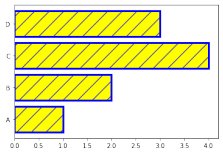
Barplot

- plt.bar / plt.barh : Barplot **verticale/horizontale** avec une seule série de valeurs
  - edgecolor = 'blue' : la couleur des encadrements. / edgecolor = 'none' : pas d'encadrement.
  - linewidth = 2 : l'épaisseur des traits.
  - yerr = [0.3, 0.3, 0.2] : les valeurs des barres d'erreur.
  - ecolor = 'magenta' : la couleur des barres d'erreur.
  - capsize = 3 : la longeur du petit trait des barres d'erreur.
  - log = True : pour avoir les coordonnées y en log.
  - linestyle = 'solid'' : valeurs possibles : 'solid', 'dashed', 'dashdot', 'dotted'.
  - hatch = '/' : les hachures. Valeurs possibles : '/', '\', '|', '-', '+', 'x', 'o', 'O', '.', '*'.

```
plt.bar(range(5), [1, 3, 3, 5, 4], width = 0.6, color = 'yellow',
  edgecolor = 'blue', linewidth = 2,# yerr = [0.5, 1, 2, 1, 2],
  ecolor = 'magenta', capsize = 10)
pyplot.xticks(range(5), ['A', 'B', 'C', 'D', 'E'], rotation = 45)
```



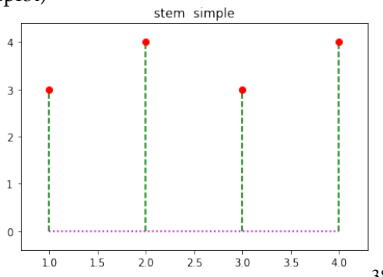
```
barWidth = 0.8
y1 = [1, 2, 4, 3]
y2 = [3, 4, 4, 3]
r = range(len(y1))

pyplot.barh(r, y1, height = barWidth, color = ['yellow' for i in y1],
            edgecolor = ['blue' for i in y1], linestyle = 'solid', hatch ='/',
            linewidth = 3)
pyplot.yticks(range(len(y1)), ['A', 'B', 'C', 'D'])
```



Barre verticale par point : Stem plot

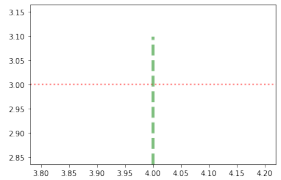
- plot.stem : Graphe qui représente une barre verticale par point à la coordonnée indiquée
  - Coordonnées en x et coordonnées y correspondantes
  - markerfmt = 'ro' : format pour les points en (x, y) avec la couleur (rouge) et le symbole du point (un rond).
  - markerfmt = 'r-' : trace une ligne continue reliant les points.
  - linefmt = 'g--' : le format des lignes verticales avec la couleur et (vert) le type de trait (trait interrompu).
  - basefmt = 'm:' : le format de l'axe horizontal avec la couleur (magenta) et le type de trait (pointillés).
  - pour les couleurs et les types de traits, (cf. plt.plot)

```
x = [1, 2, 4, 3]
y = [3, 4, 4, 3]
pyplot.stem(x,y,markerfmt = 'ro', linefmt = 'g--',
            basefmt = 'm:', use_line_collection = True)
pyplot.margins(0.1, 0.1)
pyplot.title('stem  simple')
```



Droite horizontale/verticale

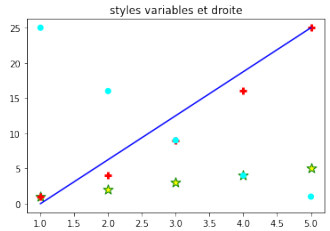
- Traçage de droites horizontales ou verticales :
  - plt.axhline(y = ?) : droite horizontale.
  - plt.axvline(x = ?) : droite verticale.
  - On peut tracer la droite sur une partie en donnant les bornes entre 0 et 1 :
    - pltt.axvline(x = 4, ymin = 0, ymax = 0.8)
  - paramètres des droites :
    - colorcolor : la couleur
    - alpha : le niveau de transparence, entre 0 et 1.
    - linestyle : le style : '-', '--', '-.', ':', ''.



```
plt.axhline(y = 3, xmin = 0, xmax  = 1, color = 'red', alpha = 0.5, linestyle = ':', linewidth = 2)
plt.axvline(x = 4, ymin = 0, ymax  = 0.8, color = 'green', alpha = 0.5, linestyle = '--', linewidth = 4)
```


Nuage de points : Scatter plot

- plt.scatter trace un ou plusieurs nuages de points sur le même graphe .Paramètres :
  - c = 'red' ou color = 'red' : la couleur des points (défaut est 'black').
  - edgecolor = 'none' : évite que le symbole soit entouré d'un trait noir.
  - s = 10 : taille du symbole au carré
  - marker = 'o' : le type de symbole, ici un rond. Les principaux symboles sont les suivants :
    - 'o' : rond.
    - 's' : carré (square).
    - '+' : croix en forme de +.
    - 'x' : croix en forme de x.
    - '*' : étoile.
    - 'D' : losange (diamond).
    - 'd' : losange allongé.
    - 'H' : hexagone ('h' est aussi un hexagone, mais tourné).
    - 'p' : pentagone.
    - '.' : point.
    - '>' : triangle vers la droite ('<' pour vers la gauche).
    - 'v' : triangle vers le bas ('^' pour vers la haut).
    - '|' : trait vertical ('_' pour trait horizontal).
    - '1' : croix à 3 branches vers le bas ('2' vers le haut, '3' vers la gauche, '4' vers la droite).

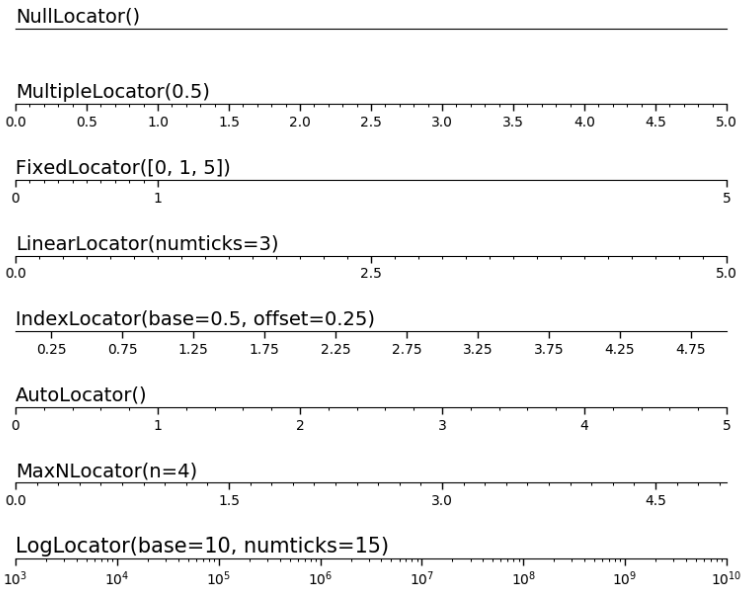



```
x = [1, 2, 3, 4, 5]
y1 = [1, 2, 3, 4, 5]
y2 = [1, 4, 9, 16, 25]
y3 = [25, 16, 9, 4, 1]
pyplot.scatter(x, y1, s = 130, c = 'yellow', marker = '*', edgecolors = 'green')
pyplot.scatter(x, y2, s = 50, c = 'red', marker = '+', linewidth = 3)
pyplot.scatter(x, y3, s = 50, c = 'cyan', marker = 'o', edgecolors = 'none')

pyplot.plot([1, 5], [0, 25], color = 'blue', linestyle = 'solid')

pyplot.title('styles variables et droite')
```

Amplitude des axes

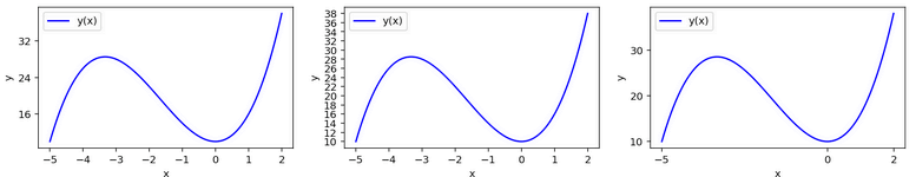
- Matplotlib ajuste automatiquement les amplitudes de manière à avoir une courbe la plus grande possible
- Ajustable à la main avec set_xlim et set_ylim
- Ajout de marques (tick) et de grilles (grid)
- Les graduations sont les marqueurs indiquant les points de données sur les axes.
- Le module mpl.ticker permet d'ajuster les ticks
  - Choix d'une stratégie de placement parmi : mpl.ticker.MaxNLocator : plus grand nombre de ticks mpl.ticker.MultipleLocator : même base de positionnement mpl.ticker.FixedLocator : position fixe
  - La stratégie est fixée avec : set_major_locator et the set_minor_ locator methods (axes)

Graduations

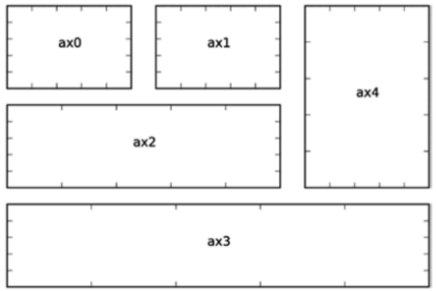


Exemple

```
def draw(ax):
    ax.plot(x, y1, color="blue", label="y(x)")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend()
x = np.linspace(-5, 2, 100)
y1 = x**3 + 5*x**2 + 10
fig, axes = plt.subplots(1,3,figsize=(15, 2.5))
mx = mpl.ticker.MaxNLocator(8)
my = mpl.ticker.MaxNLocator(4)
axes[0].xaxis.set_major_locator(mx)
axes[0].yaxis.set_major_locator(my)
draw(axes[0])
mx = mpl.ticker.MultipleLocator(1)
my = mpl.ticker.MultipleLocator(2)
axes[1].xaxis.set_major_locator(mx)
axes[1].yaxis.set_major_locator(my)
draw(axes[1])
mx = mpl.ticker.FixedLocator([-5,0,2])
my = mpl.ticker.FixedLocator([10,20,30])
axes[2].xaxis.set_major_locator(mx)
axes[2].yaxis.set_major_locator(my)
draw(axes[2])
```

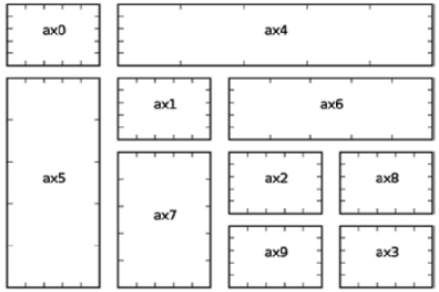


Agencement de graphiques

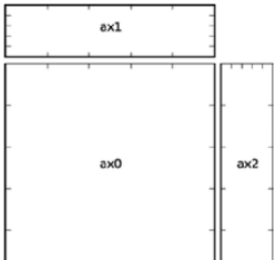



```
ax0 = plt.subplot2grid((3, 3), (0, 0))
ax1 = plt.subplot2grid((3, 3), (0, 1))
ax2 = plt.subplot2grid((3, 3), (1, 0),
      colspan=2)
ax3 = plt.subplot2grid((3, 3), (2, 0),
      colspan=3)
ax4 = plt.subplot2grid((3, 3), (0, 2),
      rowspan=2)
```

Agencement de graphiques (suite)

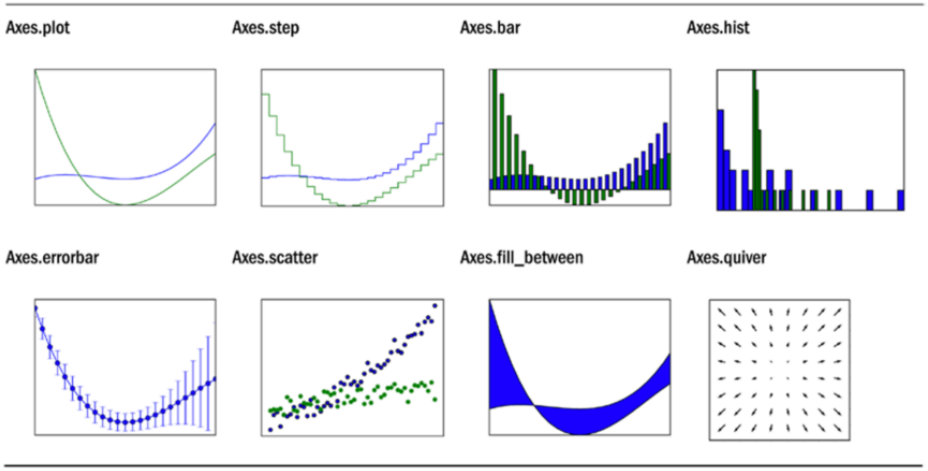



```
fig = plt.figure(figsize=(6, 4))

gs = mpl.gridspec.GridSpec(4, 4)

ax0 = fig.add_subplot(gs[0, 0])
ax1 = fig.add_subplot(gs[1, 1])
ax2 = fig.add_subplot(gs[2, 2])
ax3 = fig.add_subplot(gs[3, 3])
ax4 = fig.add_subplot(gs[0, 1:])
ax5 = fig.add_subplot(gs[1:, 0])
ax6 = fig.add_subplot(gs[1, 2:])
ax7 = fig.add_subplot(gs[2:, 1])
ax8 = fig.add_subplot(gs[2, 3])
ax9 = fig.add_subplot(gs[3, 2])
```




```
fig = plt.figure(figsize=(4, 4))

gs = mpl.gridspec.GridSpec(
        2, 2,
        width_ratios=[4, 1],
        height_ratios=[1, 4],
        wspace=0.05, hspace=0.05)

ax0 = fig.add_subplot(gs[1, 0])
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[1, 1])
```

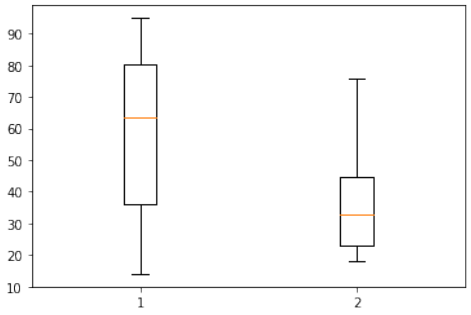
Les différents types de graphiques 2D



Boîtes à moustaches

Ce 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.

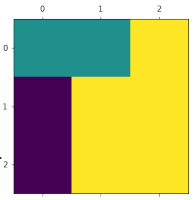
```
import matplotlib.pyplot as plt
dataline1 = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
dataline2 = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
data = [ dataline1, dataline2 ]
plt.boxplot( data )
```



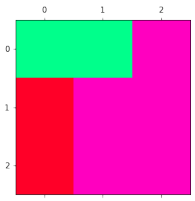
Heatmap

- Représentation d'une matrice :
  - un carré par valeur
  - une couleur qui dépend de cette valeur
- Affichage d'une matrice : pyplot.matshow(mat)
  - origin = 'lower' : la première ligne est mise tout en bas et la dernière tout en haut.
  - **cmap = plt.cm.gist_rainbow** : indique une carte des couleurs alternative.
  - plt.xticks/plt.yticks : permet d'indiquer des labels
  - mat = [[0.5, 0.5, 1], [0, 1, 1], [0, 1, 1]]
  - `pyplot.matshow(mat)` : les lignes de la matrice sont représentées en ligne et les colonnes en colonnes, et la première ligne est en haut (correspond à la représentation matricielle habituelle).
  - plt.gray() : Représentation de la matrice en niveaux de gris

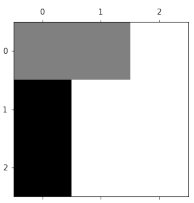
```
mat = [[0.5, 0.5, 1], [0, 1, 1], [0, 1, 1]]
plt.matshow(mat)
```




```
plt.matshow(mat, cmap = pyplot.cm.gist_rainbow)
```



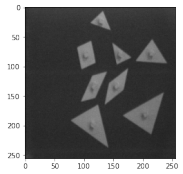

```
plt.gray()
plt.matshow(mat)
```



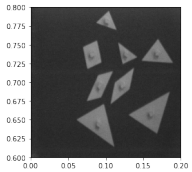
Images

- Lecture d'une image : plt.imread renvoie une array numpy.
  - im = plt.imread('ex.jpg')
- Affichage d'une image : plt.imshow.
  - Positionnement dans un rectangle (xmin,xmax,ymin,ymax) :
    - plt.imshow(im, extent =[xmin, xmax, ymin, ymax])

```
im = plt.imread('tangram.png')
plt.imshow(im)
```



```
plt.imshow(im, extent = [0, 0.2, 0.6, 0.8])
```



zorder

- zorder : indique dans quel ordre afficher les différents éléments. Plus le zorder est élevé, plus l'élément est devant les autres.
  - zorder par défaut :
    - 1 pour les patchs
    - 2 pour les lignes
    - 3 pour le texte

Fermer une fenêtre de figure

- plt.close () # ferme la figure active en cours
- plt.close (fig) # ferme la figure avec la poignée 'fig'
  - fig1 = plt.figure(1) # 1ère figure
  - plt.title("exemple")
  - plt.close(fig1)
- 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

Encore …

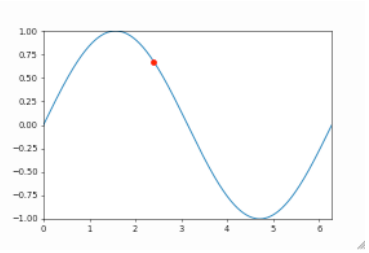
- Réinitalisation / effaçage :
  - pyplot.close() : ferme la figure et libère toutes les ressources liées à cette figure.
  - pyplot.clf() : efface la figure courante.
  - pyplot.cla() : efface le graphe courant.
- Sauvegarde de l'image dans un fichier : pyplot.savefig('image.png'); pyplot.close().
  - pyplot.savefig('image.png', dpi = 600) : fixe la résolution.
  - pyplot.savefig('image.png', transparent = True) : fond transparent.
  - pyplot.savefig('image, format = 'pdf') : indique le format. Les formats supportés sont : png, pdf, eps, svg, sinon le format est deviné d'après l'extension du fichier.
- Figure et axe courants :on peut avoir la figure courante sur laquelle les propriétés s'appliquent par : `pyplot.gcf()`
  - pour avoir les axes courants : `pyplot.gca()`
  - pour fixer l'axe sur lequel on veut travailler : `pyplot.sca(ax)`
  - on peut avoir la liste des graphes de la figure courante par : `pyplot.gcf().get_axes()`

Inline interactive

- %matplotlib notebook
- %matplotlib nbagg

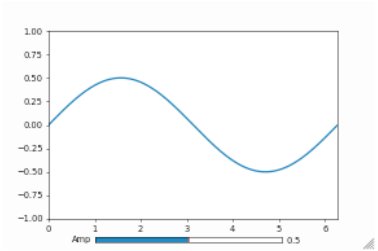
Animation gif

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



Contrôles interactifs avec matplotlib.widgets

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Slider
%matplotlib notebook
%matplotlib nbagg
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()
```



Annexes

Vous pouvez consulter :

- https://riptutorial.com/Download/matplotlib-fr.pdf
- http://www.python-simple.com/python-matplotlib/matplotlib-intro.php
- http://apprendre-python.com/page-creer-graphiques-scientifiques-python-apprendre
- https://www.courspython.com/introduction-courbes.html