L1 — Calcul Scientifique

Frédéric Jurie (frederic.jurie@unicaen.fr)
Université de Caen Normandie
Année universitaire 2019-2020

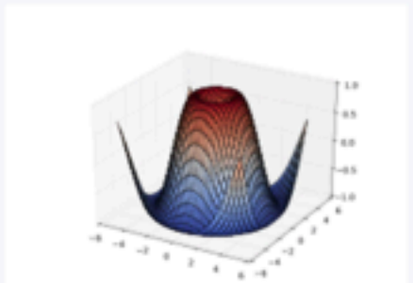
Le module matplolib

Objectifs de ce cours

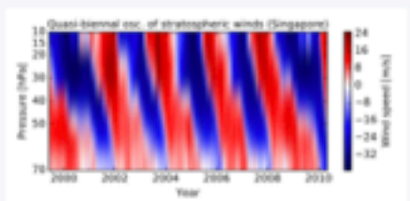
- Présentation de la librairie Python Matplotbib
- Wikipedia:
  “Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous forme de graphiques4. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy5. (...)

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”

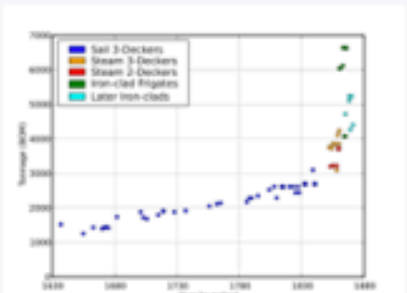
Exemples de graphiques



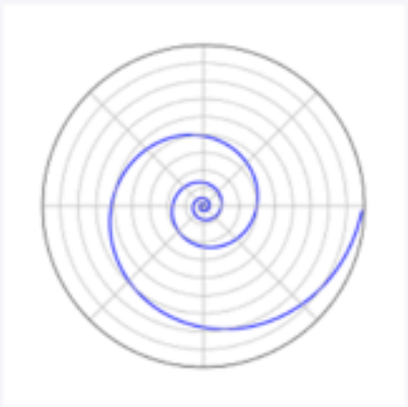
Représentation 3D



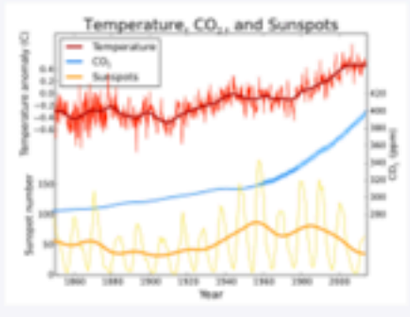
Direction des vents stratosphériques au-dessus de Singapour



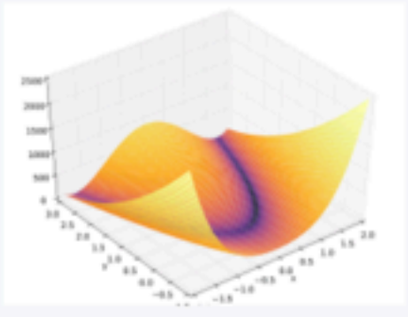
Croissance de la capacité des navires entre 1630 et 1875



Spirale logarithmique



Températures globales, CO_2 atmosphérique et activité solaire depuis 1850.



Fonction de Rosenbrock

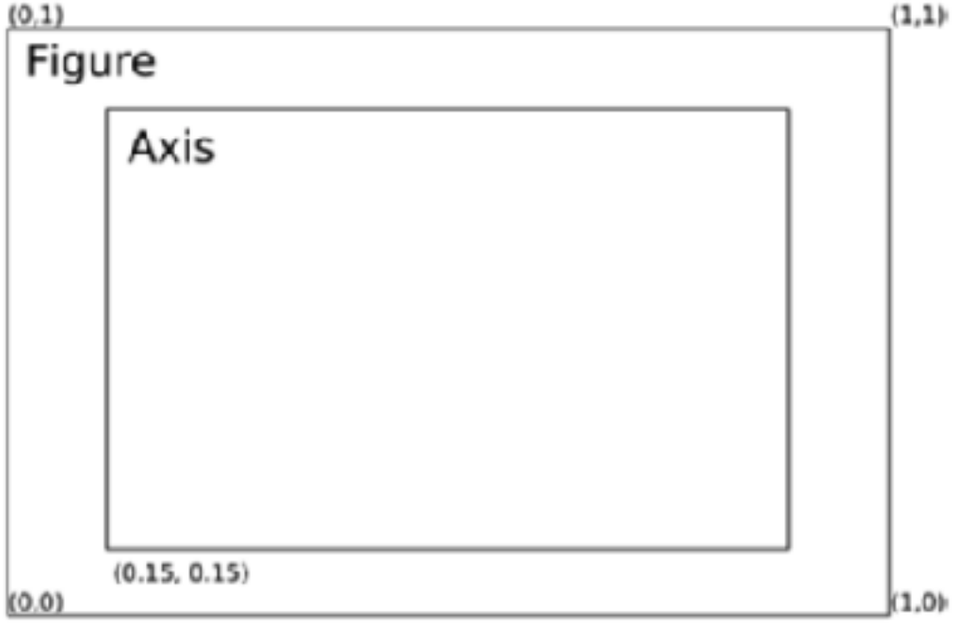
(source Wikipédia)

Introduction

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

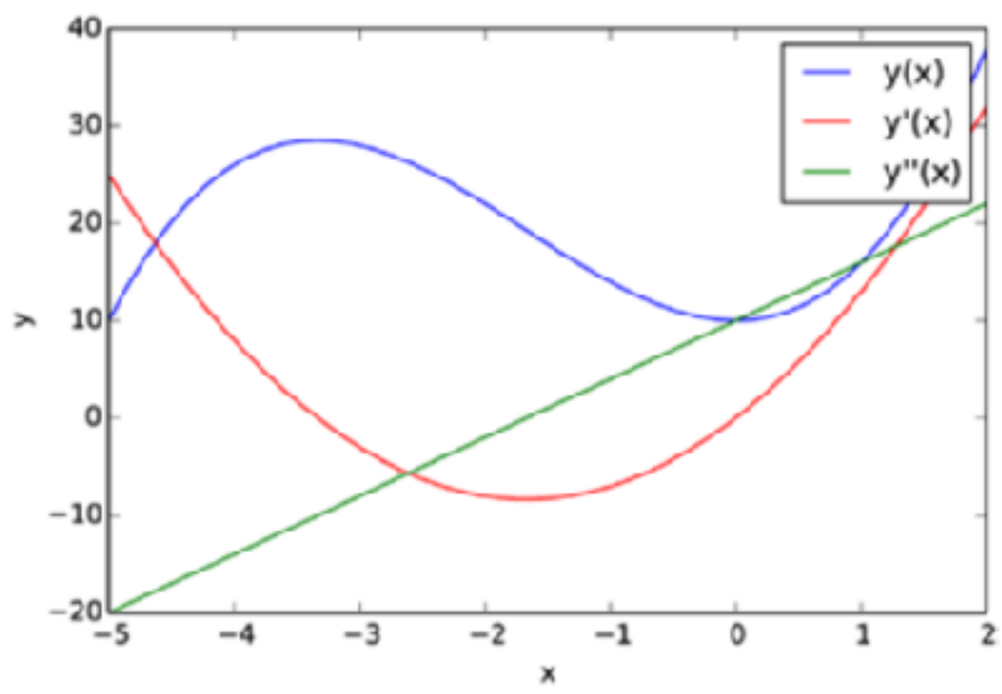
```
In [1]: %matplotlib inline
In [2]: import matplotlib as mpl
In [3]: import matplotlib.pyplot as plt
In [4]: from mpl_toolkits.mplot3d.axes3d import Axes3D
```

- Matplotlib créée des ‘Figures' contenant des ‘Axes'



Exemple de création d'une figure simple

```
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()
```

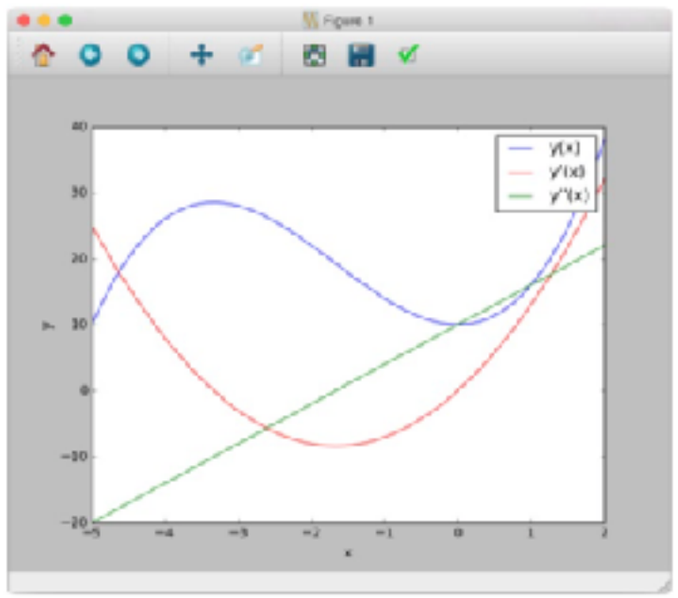


Utilisation de différents backends

- 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, ou en éditant le fichier de ressources Matplotlib

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

.



Environnement QT4 sous macosx

Utilisation de différents backends (suite)

- 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 (fait automatiquement).

Création de ‘Figures’ (mode le plus simple)

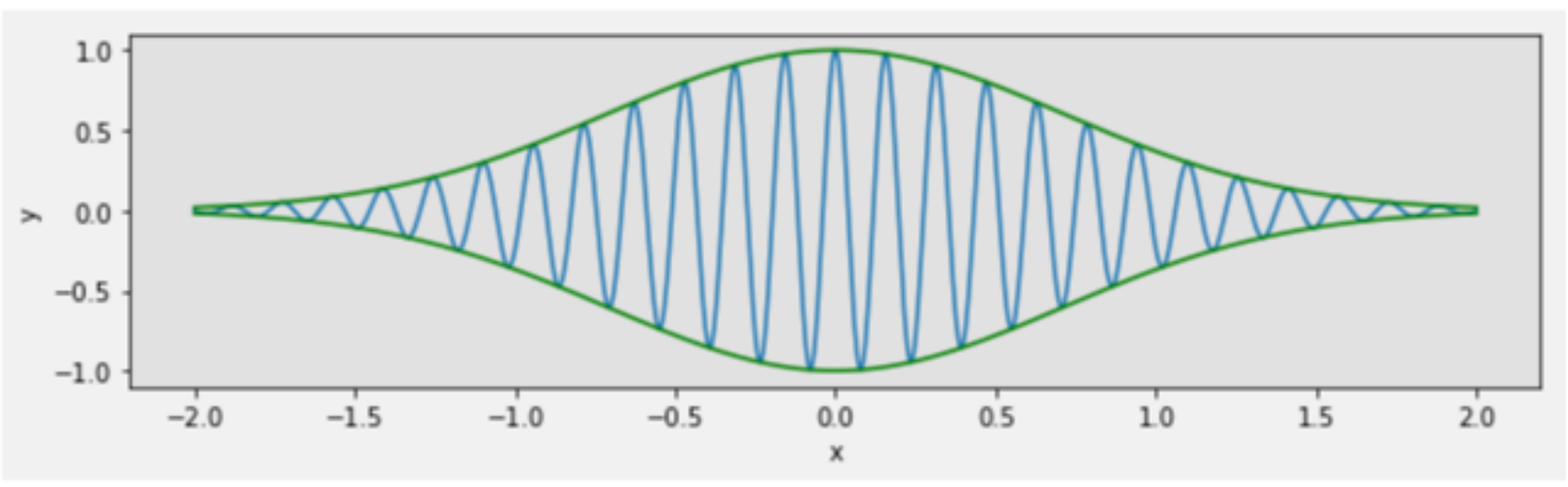
- La création d’une figure se fait par la commande : **`plt.figure()`** qui prend différents arguments optionnels parmi lesquels :
  - `figsize=(width, height)` taille de la figure, unités : inches
  - `facecolor=“#f1f1f1"` : couleur du ‘fond’ de la figure
- Une fois la figure créée :
  - `plt.plot(x,y)`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

Création de ‘Figures’ (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 argument
    - 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()**

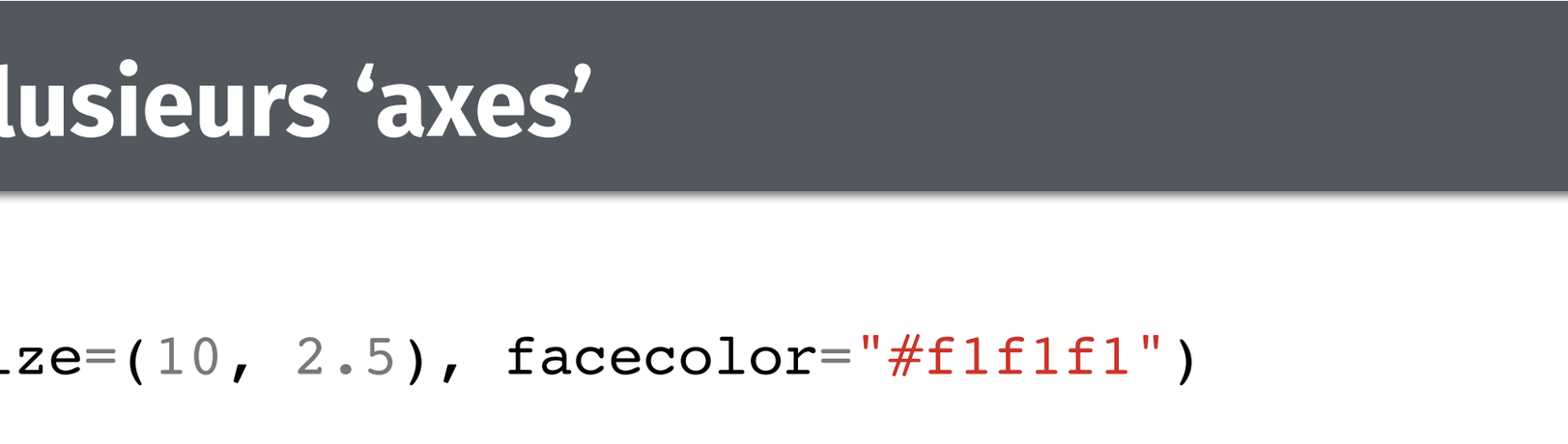
Exemple :

```
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")
```



Exemple avec plusieurs ‘axes’

```
fig = plt.figure(figsize=(10, 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")

left, bottom, width, height = 0.25, 0.25, 0.5, 0.5
ax = fig.add_axes((left, bottom, width, height), facecolor="#e1e1e1")
ax.plot(x, y1 * y2)
ax.plot(x, y2, 'g')
ax.plot(x, -y2, 'g')
```

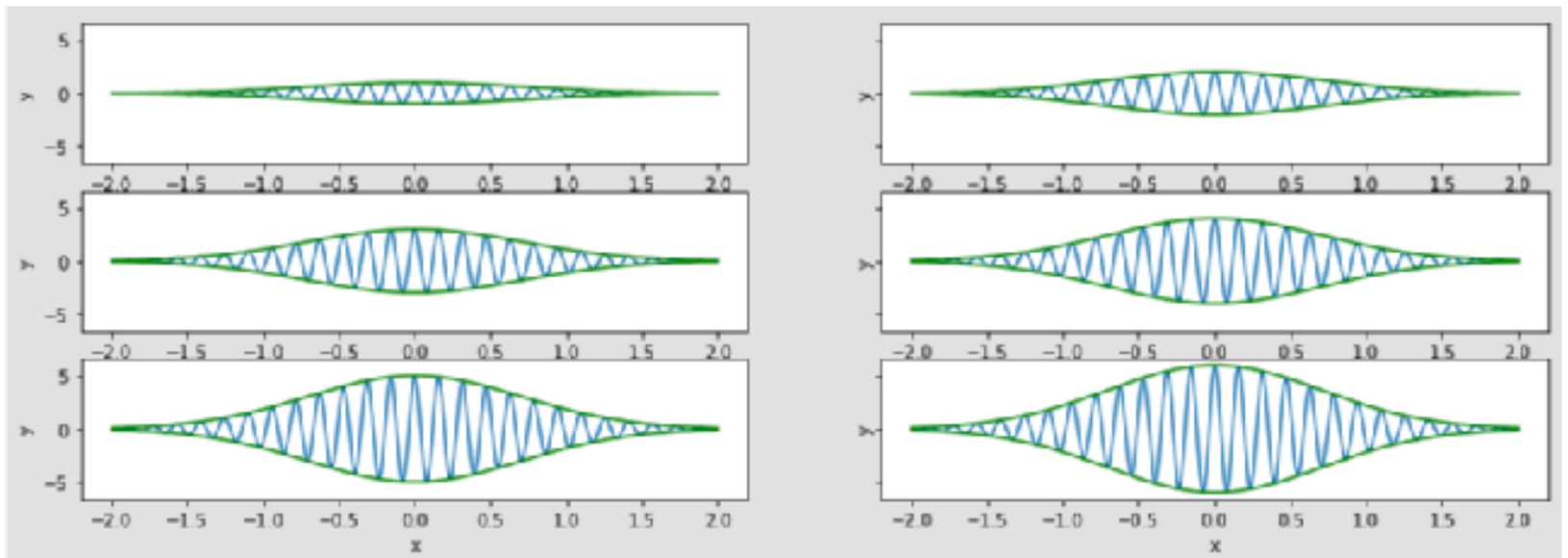


Création simplifiée des axes

- 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"}

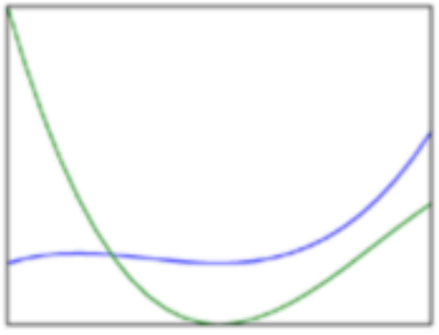
Exemple

```
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
```

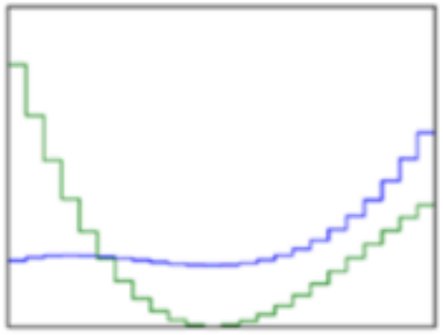


Les différents types de graphiques 2D

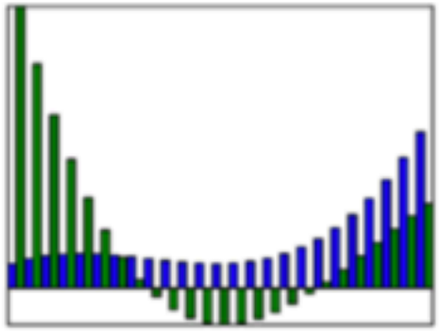
Axes.plot



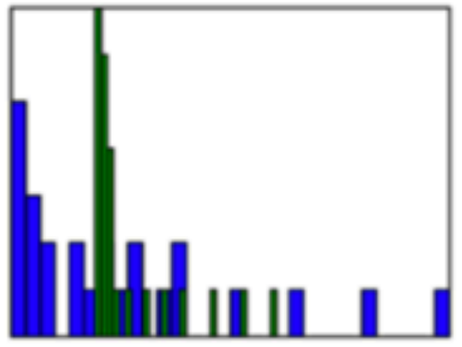
Axes.step



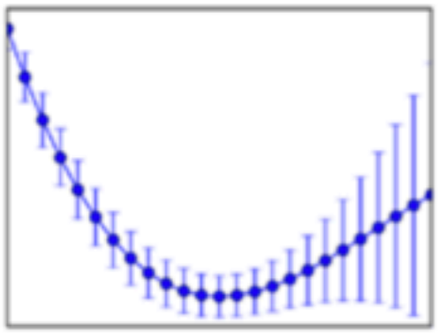
Axes.bar



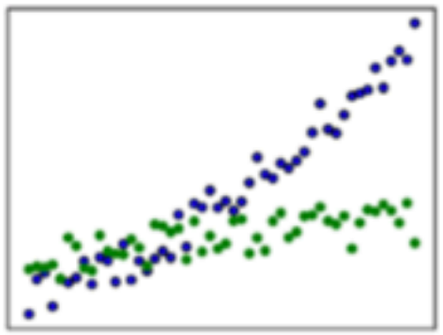
Axes.hist



Axes.errorbar

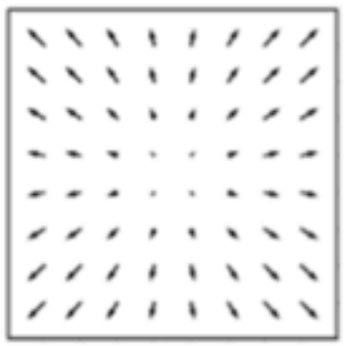


Axes.scatter



Axes.fill_between

Axes.quiver

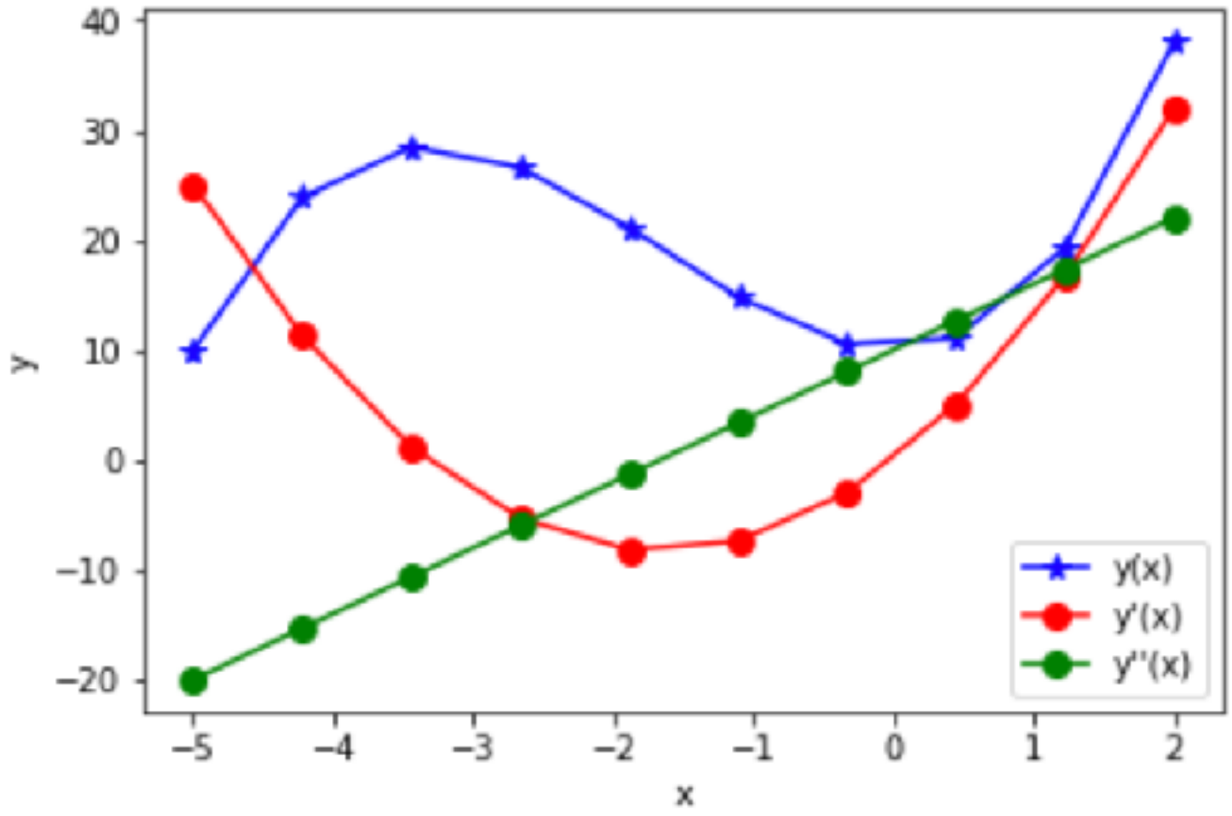


Propriétés des lignes

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.

Exemple

```
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()
```



Légendes

- 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
  ou bbox_to_anchor =(x,y) pour un positionnement à un endroit spécifié par ses coordonnées

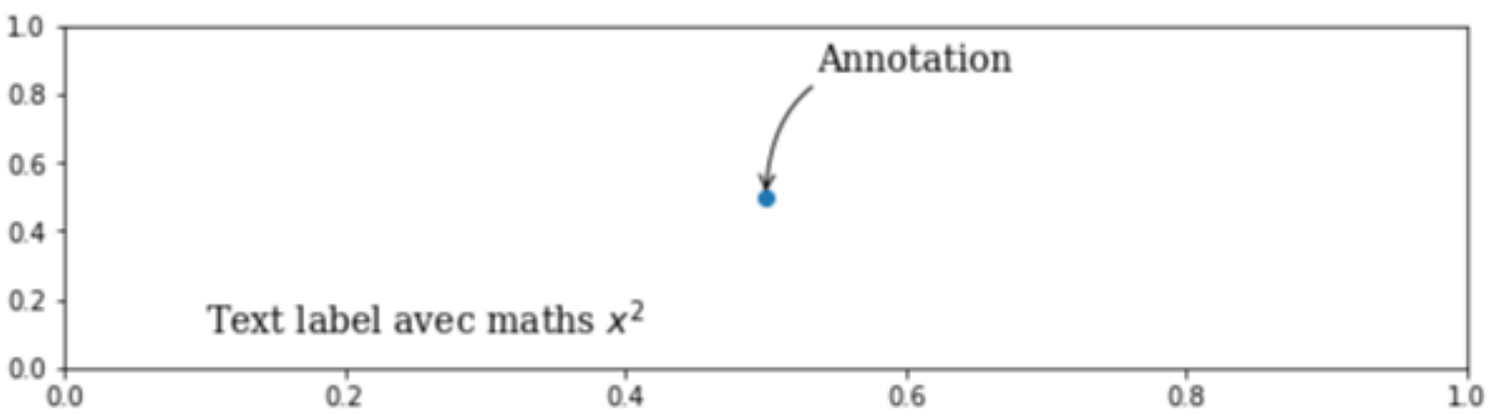
Formatage du texte

- Possibilité d'ajouter du texte aux figures, y compris avec du formatage latex
  "Regular text: $f(x)=1-x^2$"
- Ajout de texte :
  ax.text()
- Annotations :
  ax.annotate()

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.

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")
```



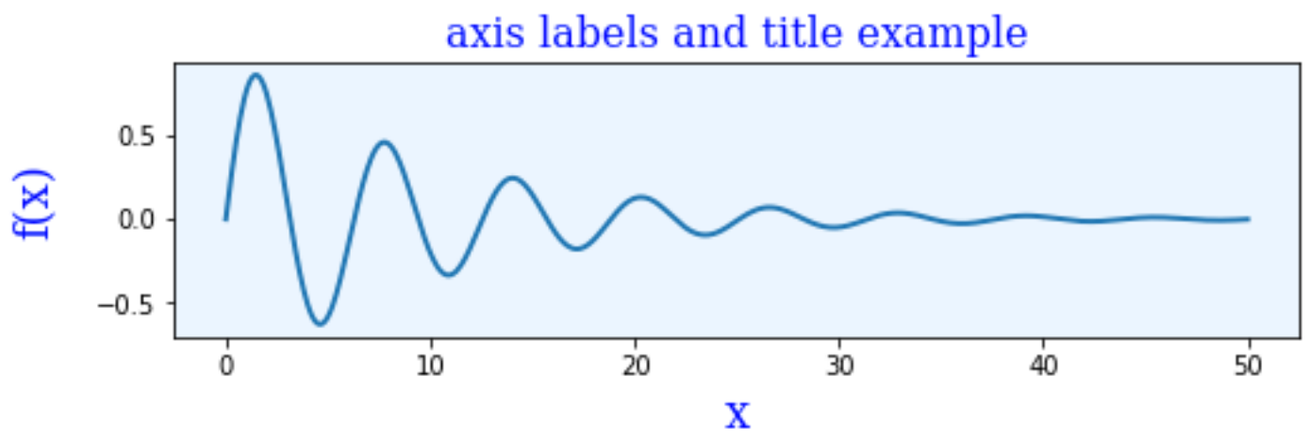
Propriétés des axes et titres

- Comme nous l'avons vu dans les exemples précédents,
  set_xlabel et set_ylabel permettent de mettre une légende sur les axes du graphique
  labelpad (espacement, en points, avec le graphique)
  color, fontsize et fontname également disponibles

- set_title() permet de spécifier le titre

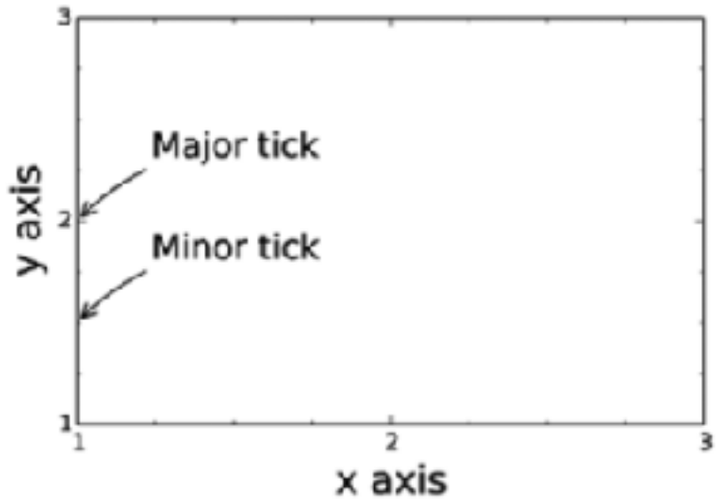
Exemple

```
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")
```



Amplitude des axes

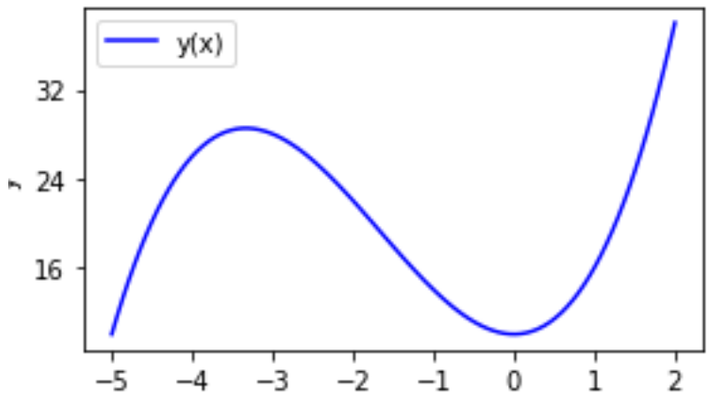
- Matplotlib ajuste automatiquement les amplitudes de manière à avoir une courbe la plus grande possible
- Ajustable à la main avec set_xlim et set_ylim
  ax. set_xlim(min,max)
- Ajout de marques (tick) et de grilles (grid)
- Le module mpl.ticker permet d'ajuster les ticks

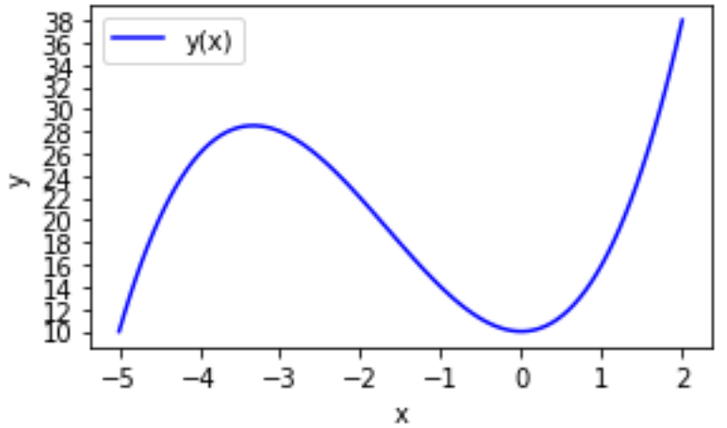


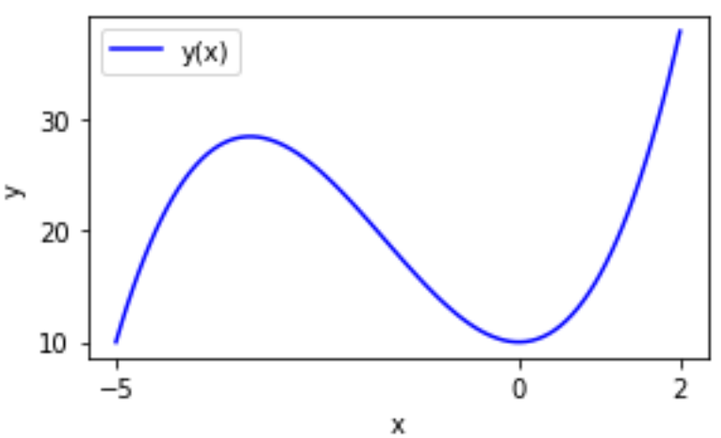
  - Choix d'une stratégie de placement parmi :
    mpl.ticker.MaxNLocator : plus grand nombre de ticks
    mpl.ticker.MultipleLocator : même base de positionnement
    mpl.ticker.FixedLocator : position fixe
  - La stratégie est fixée avec :
    set_major_locator et the set_minor_ locator methods (axes)

Exemple

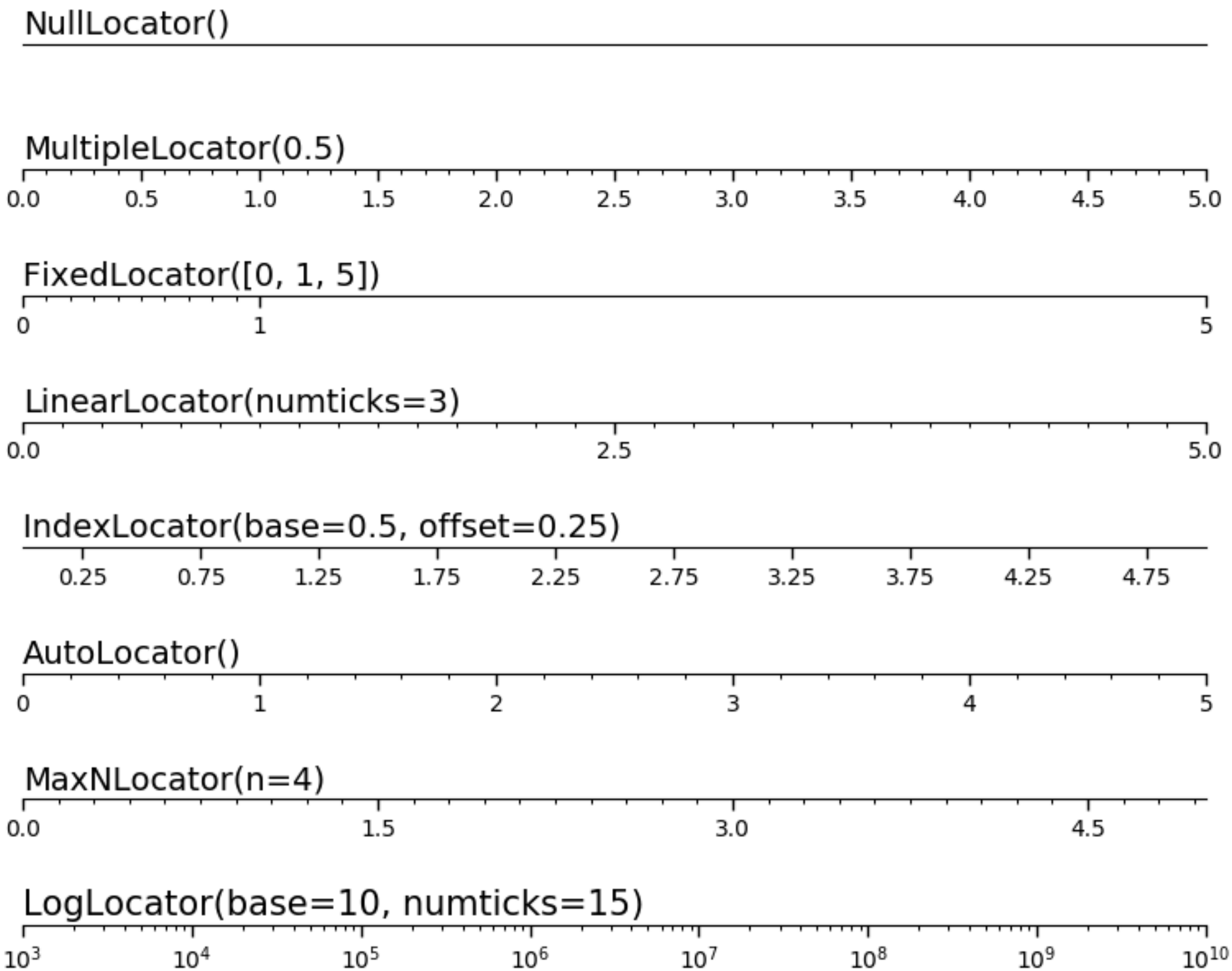
```
def draw(ax):
    ax.plot(x, y1, color="blue", label="y(x)")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend()
x = np.linspace(-5, 2, 100)
y1 = x**3 + 5*x**2 + 10

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])
```







Suite


NullLocator()
MultipleLocator(0.5)
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
FixedLocator([0, 1, 5])
0 1 5
LinearLocator(numticks=3)
0.0 2.5 5.0
IndexLocator(base=0.5, offset=0.25)
0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75
AutoLocator()
0 1 2 3 4 5
MaxNLocator(n=4)
0.0 1.5 3.0 4.5
LogLocator(base=10, numticks=15)
10^{3} 10^{4} 10^{5} 10^{6} 10^{7} 10^{8} 10^{9} 10^{10}

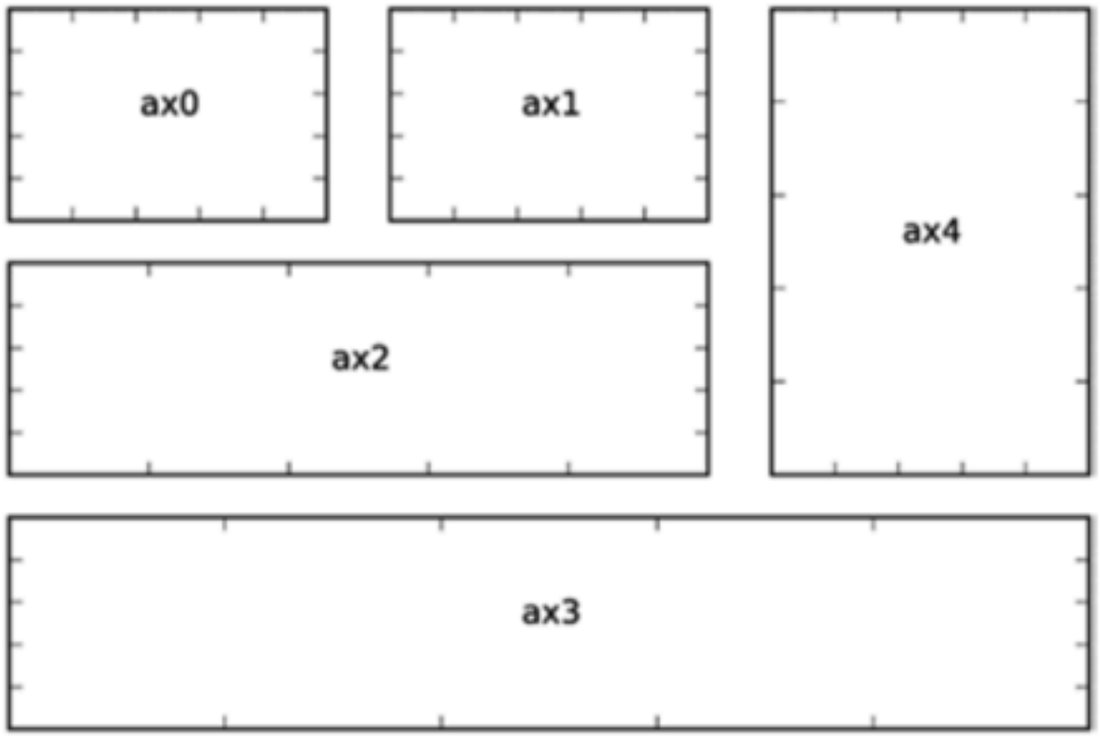
La grille

- L’affichage de la grille s’obtient par :
  axes.grid()
- La grille suit les ticks
- 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) puis :
  ax2 = ax1.twinx()
  puis tracé de la seconde : ax2.plot(...)

Agencements de graphiques plus complexes

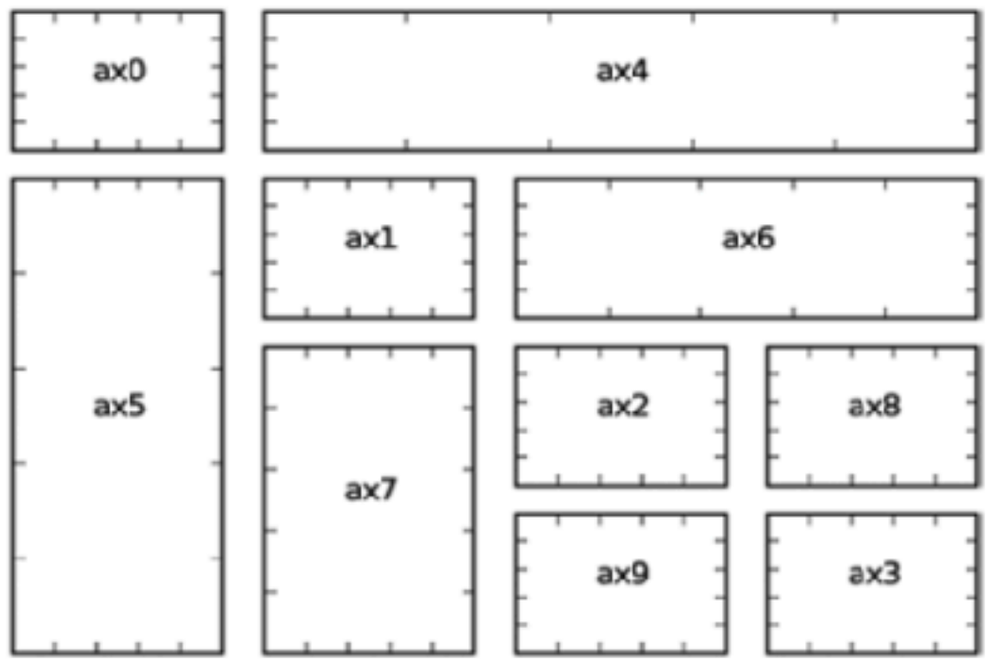
- subplot2grid



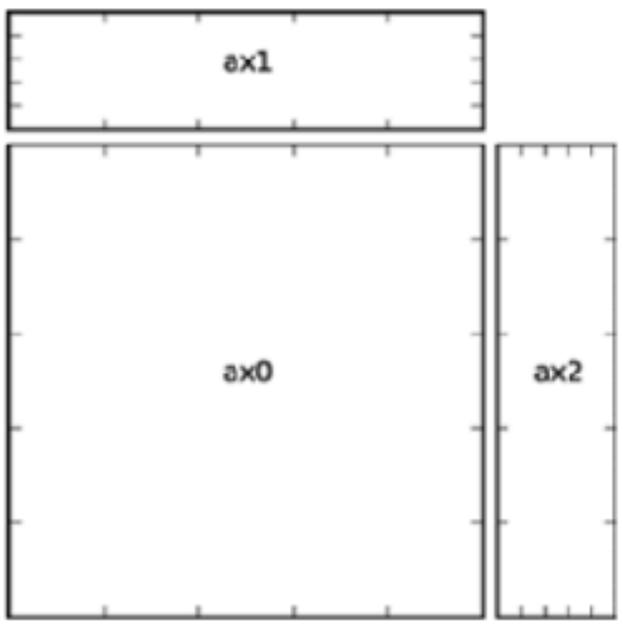

```
ax0 = plt.subplot2grid((3, 3), (0, 0))
ax1 = plt.subplot2grid((3, 3), (0, 1))
ax2 = plt.subplot2grid((3, 3), (1, 0),
      colspan=2)
ax3 = plt.subplot2grid((3, 3), (2, 0),
      colspan=3)
ax4 = plt.subplot2grid((3, 3), (0, 2),
      rowspan=2)
```

Agencements de graphiques plus complexes

- gridspec




```
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])
```

Références

Ouvrages utilisés pour préparer ce cours