Université Paris-Sud
Crédits : A. Decoene

Représentation graphique avec le module Matplotlib

Représentation discrète d'une fonction

Pour tracer la courbe représentative d'une fonction f avec Matplotlib, il faut tout d'abord la "discrétiser" ; c'est-à-dire définir une liste de points $(x_i, f(x_i))$ qui va permettre d'approcher le graphe de la fonction par une ligne brisée. Bien sûr, plus on augmente le nombre de points, plus la ligne brisée est "proche" du graphe (en un certain sens).

Plus précisément, pour représenter le graphe d'une fonction réelle f définie sur un intervalle $[a, b]$, on commence par construire un vecteur X, discrétisant l'intervalle $[a, b]$, en considérant un ensemble de points équidistants dans $[a, b]$. Pour ce faire, on se donne un naturel N *grand* et on considère `X=numpy.linspace(a,b,N)` (ou, ce qui revient au même, on pose $h = \frac{b-a}{N-1}$ et on considère `X=numpy.arange(a,b+h,h)`).

On construit ensuite le vecteur Y dont les composantes sont les images des X_i par f et on trace la ligne brisée reliant tous les points (X_i, Y_i).

La fonction `plot` du module matplotlib.pyplot

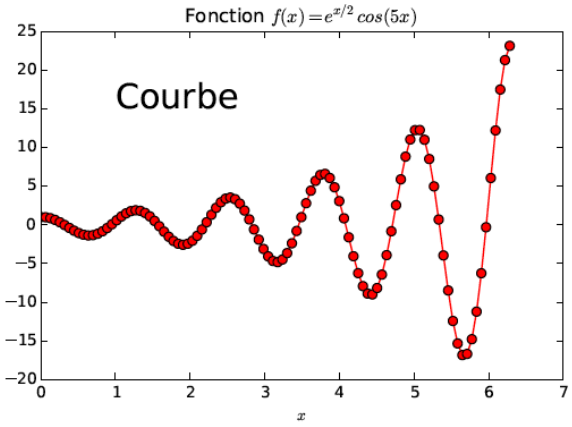
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0., 2*np.pi, 100)
plt.plot(x, np.exp(x/2)*np.cos(5*x), '-ro')

plt.title('Fonction $f(x)=e^{x/2} cos(5x)$')
plt.xlabel('$x$')
plt.text(1, 15, 'Courbe', fontsize=22)
plt.savefig('1D_exemple.pdf')
plt.show()
```

Cette fonction permet de tracer des lignes brisées. Si X et Y sont deux vecteurs-lignes (ou vecteurs-colonnes) de même taille n, alors `plt.plot(X,Y)` permet de tracer la ligne brisée qui relie les points de coordonnées (X(i),Y(i)) pour $i = 1, \ldots, n$.

Pour connaître toutes les options, le mieux est de se référer à la documentation de Matplotlib. Voyons ici quelques unes d'entre elles :

— **bornes** : spécifier un rectangle de représentation, ce qui permet un zoom, d'éviter les grandes valeurs des fonctions par exemple, se fait via la commande `plt.axis([xmin,xmax,ymin,ymax])`

— **couleur du trait** : pour changer la couleur du tracé une lettre g vert (green), r rouge (red), k noir, b bleu, c cyan, m magenta, y jaune (yellow), w blanc (white). `plt.plot(np.sin(x),'r')` tracera notre courbe sinus en rouge. Pour avoir différents niveaux de gris on peut utiliser `color='(un`



`flottant entre 0 et 1)'`. Enfin pour avoir encore plus de couleurs, on peut utiliser le système RGB via `color=( R, G, B)` avec trois paramètres compris entre 0 et 1 (RGBA est possible aussi).

— **symboles** : mettre des symboles aux points tracés se fait via l'option `marker`. Les possibilités sont nombreuses parmi

```
'+' | '*' | ',' | '.' | '1' | '2' | '<' | '>' | 'D' | 'H' |

'^' | 'Đ' | 'd' | 'h' | 'o'  | 'p' | 's' | 'v' | 'x' | '|' |

TICKUP | TICKDOWN | TICKLEFT | TICKRIGHT | 'None' | ' ' | " .
```

— **style du trait** : pointillés, absences de trait, etc se décident avec `linestyle`. Au choix `'-'` ligne continue, `'-'` tirets, `'-.'` points-tirets, `':'` pointillés, sachant que `'None'`, `''`, `' '` donnent « rien-du-tout ». Plutôt que `linestyle`, on peut utiliser la raccourci `ls`.

— **épaisseur du trait** : `linewidth`=flottant (comme `linewidth=2`) donne un trait, pointillé (tout ce qui est défini par style du trait) d'épaisseur "flottant". Il est possible d'utiliser le raccourci `lw`.

— **taille des symboles** : `markersize`=flottant comme pour l'épaisseur du trait. D'autres paramètres sont modifiables `markeredgecolor` la couleur du trait du pourtour, `markerfacecolor` la couleur de l'intérieur, `markeredgsize`=flottant l'épaisseur du trait du pourtour.

— **étiquettes** sur l'axe des abcisses/ordonnées : Matplotlib décide tout seul des graduations sur les axes. Tout ceci se modifie via `plt.xticks`(tf), `plt.yticks`(tl) où tf est un vecteur de flottants ordonnés de façon croissante.

— ajouter un titre : `plt.title`("Mon titre")

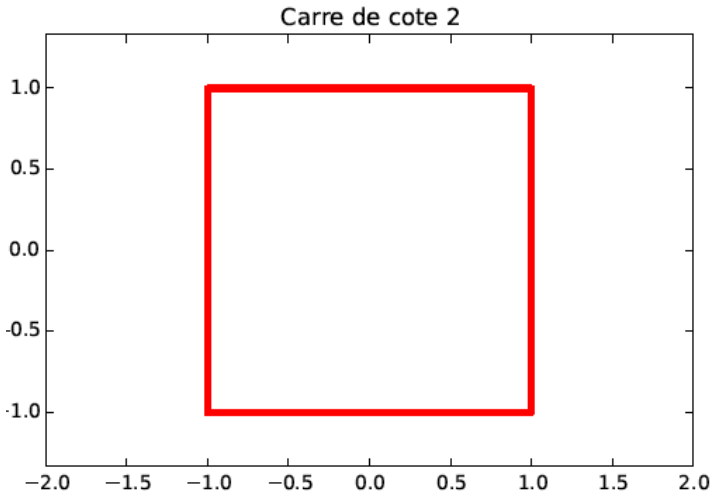
— légendes : `plt.legend`().

Exemple : tracé d'un carré

Coordonnées des sommets : $X = (-1; 1; 1; -1; -1)$ et $Y = (-1; -1; 1; 1; -1)$.

```
X=np.array([-1, 1, 1,-1,-1])
Y=np.array([-1, -1, 1,1,-1])
plt.plot(X,Y,'r',lw=3)
plt.axis('equal')
```

```
plt.axis([-2,2,-2,2])

plt.title('Carre de cote 2')
plt.savefig('carre_exemple.pdf')
plt.show()
```



Tracé de plusieurs lignes brisées

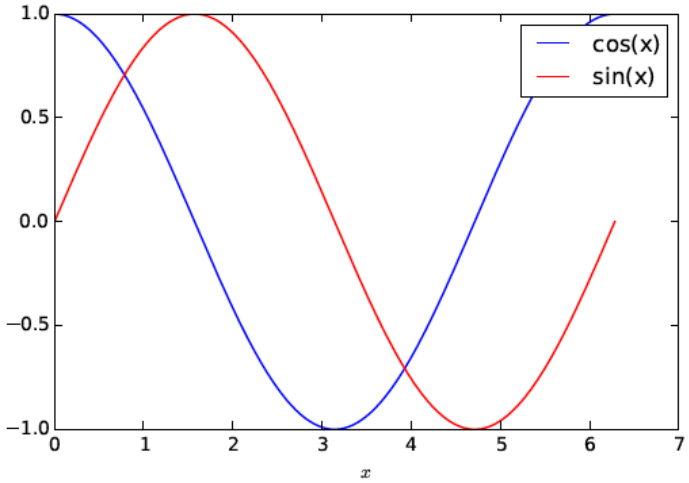
On peut superposer plusieurs courbes dans le même graphique.

```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(0, 2*np.pi, 256)
Ycos = np.cos(X)
Ysin = np.sin(X)
plt.plot(X,Ycos,'b')
plt.plot(X,Ysin,'r')
pp.show()
```

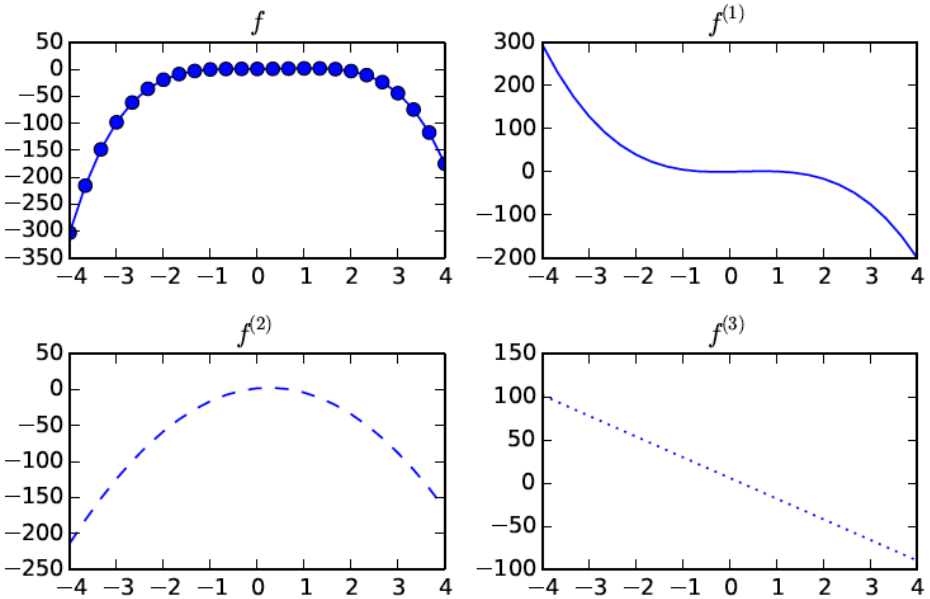
Création de plusieurs graphiques dans une même fenêtre

La commande `subplot` permet de découper la fenêtre graphique en plusieurs sous-fenêtres. Voici un exemple d'utilisation de cette commande.

```
x = np.linspace(-4., 4., 25)
plt.subplot(2, 2, 1)
plt.plot(x, -x**4 + x**3 + x**2 + 1, 'o-')
plt.title("$f$")
plt.subplot(2, 2, 2)
plt.plot(x, -4*x**3 + 3*x**2 + 2*x, '-')
plt.title("$f^{(1)}$")
plt.subplot(2, 2, 3)
```




```
plt.plot(x, -12*x**2 + 6*x + 2, '--')
plt.title("$f^{(2)}$")
plt.subplot(2, 2, 4)
plt.plot(x, -24*x + 6, ':')
plt.title("$f^{(3)}$")
plt.tight_layout()
plt.savefig("1D_subplot2.pdf")
plt.show()
```



Matplotlib dispose d'autres commandes de tracé comme `loglog, polar`...