# Tracer avec Matplotlib

Laurent Signac – CC-BY-SA – 01-03-22 1711 8b8dfaf2810a3c7df027

Le présent document done quelques bases pour tracer des courbes avec `matplotlib`. Le module `matplotlib` ne fait pas partie de la librairie standard, il soit donc être installé en plus. Il est très souvent utilisé, conjointement avec `numpy` (calcul matriciel) et `scipy` (méthodes numériques). La plupart des autres modules de tracé de courbe sont basés sur `matplotlib`.

Nous verrons ici comment utiliser conjointement `matplotlib` et `numpy` pour réaliser des graphiques.

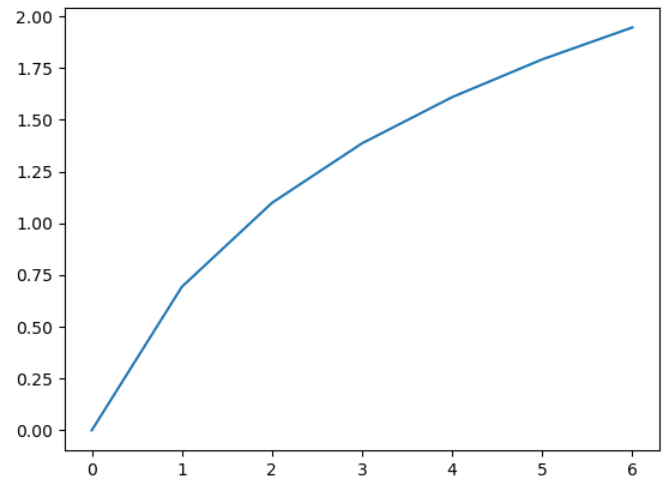
Les informations complémentaires peuvent être trouvés dans la documentation de matplotlib, qui regorge d'exemples : https://matplotlib.org/contents.html

## 1 Commandes de base

Importation du module : `import matplotlib.pyplot as plt`. On accède ensuite aux commandes par l'espace de nom `plt`. Pour un graphe standard 2D, la commande est `plot`. L'idée générale est de fournir la liste des abscisses, la liste des ordonnées, puis des réglages de styles (style de tracé, couleur etc., pour une liste complète, voire les Notes dans la dic de plot[1]).

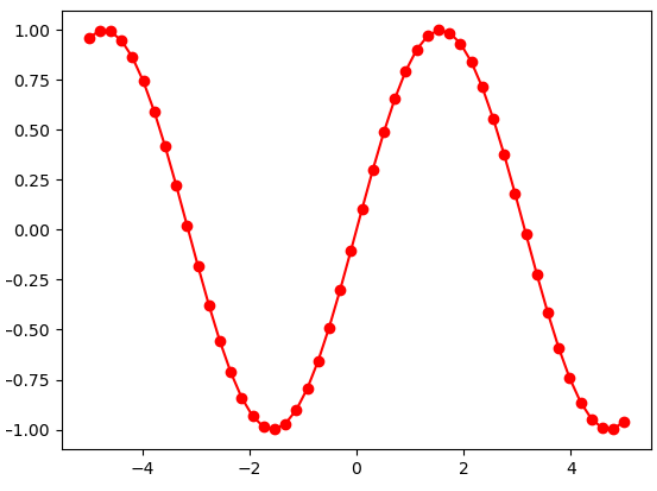
Notez que matplotlib ne trace pas de «courbe», mais simplement des points, qu'il peut éventuellement relier par des *segments*. L'impression de courbe est donnée par le fait que les point sont rapprochés.

```
import math
import matplotlib.pyplot as plt
# Liste des abscisses
lx = [0, 1, 2, 3, 4, 5, 6]
# Liste des ordonnées
ly = [math.log(x + 1) for x in lx]
plt.plot(lx, ly)
plt.show()
```




`numpy` contient des fonctions pour créer plus facilement les vecteurs d'abscisses. Il contient aussi les fonctions mathématiques pour appliquer une fonction à un vecteur (voir cours sur la vectorisation).

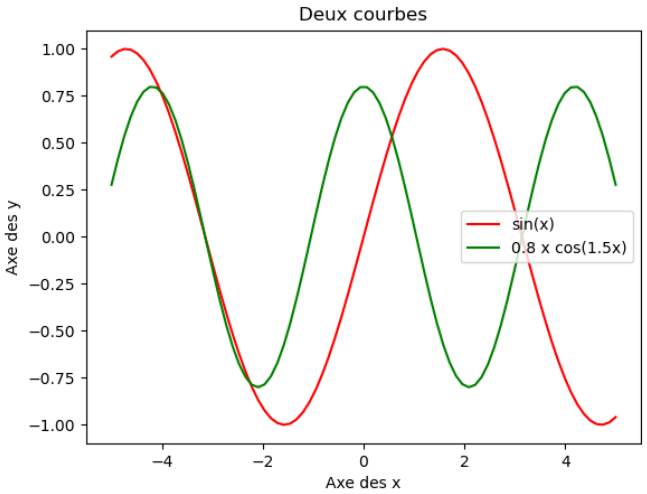
```
import math
import numpy as np
import matplotlib.pyplot as plt
# Abscisses : 50 valeurs de -5 à + 5
lx = np.linspace(-5, 5, 50)
# Ordonnées
ly = np.sin(lx)
# Équivalent sans numpy :
# ly = [math.sin(x) for x in lx]
# Style : red (r), points reliés (-) et
#  marqués par o (o)
plt.plot(lx, ly, "r-o")
plt.show()
```




[1] https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.plot.html

Il est possible de tracer plusieurs courbes sur le même graphe, d'ajouter une légende pour les courbes et les aces et un titre.

```
import numpy as np
import matplotlib.pyplot as plt
lx = np.linspace(-5, 5, 80)
ly1 = np.sin(lx)
ly2 = 0.8 * np.cos(lx * 1.5)
plt.plot(lx, ly1, "r-", lx, ly2, "g-")
plt.legend(['sin(x)', '0.8 x cos(1.5x)'])
plt.xlabel('Axe des x')
plt.ylabel('Axe des y')
plt.title('Deux courbes')
plt.show()
```



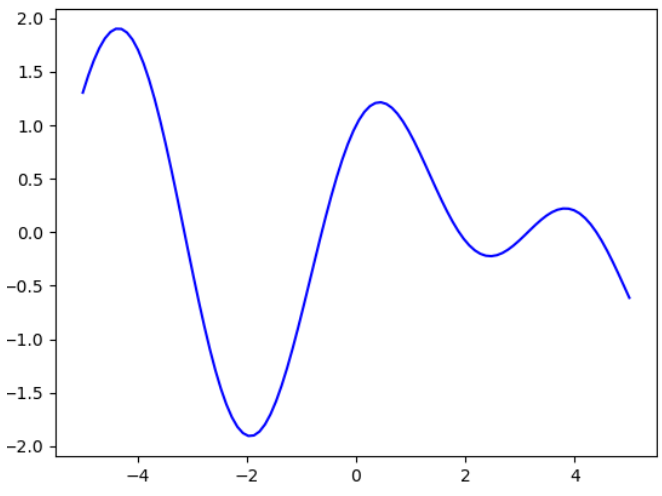

## 1.1 Tracer le graphe d'une fonction

Supposons qu'on dispose d'une fonction (python) qui donne l'ordonnée en fonction de l'abscisse. Par exemple, si on veut tracer la courbe d'équation $y = \sin(x) - \cos\left(\frac{3x}{2}\right)$.

```
def f1(x):
    from math import cos, sin
    y = sin(x) + cos(3 * x / 2)
    return y
```

On trace le graphe en choisissant un vecteur d'abscisse (plus il y a de points, plus la courbe est lissse), puis on calcule les ordonnées correspondantes, et enfin on trace :

```
import numpy as np
import matplotlib.pyplot as plt
vec_x = np.linspace(-5, 5, 100)
vec_y = [f1(x) for x in vec_x]
plt.plot(vec_x, vec_y, "b-")
plt.show()
```

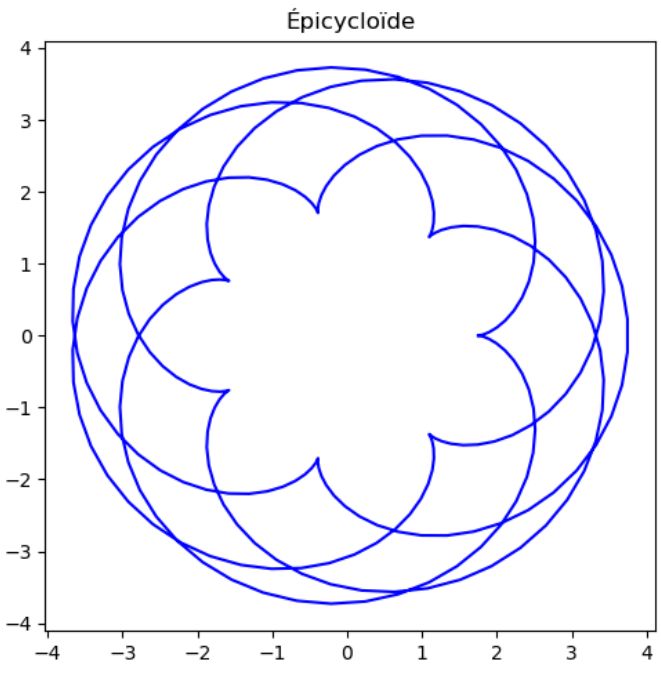



Supposons maintenant qu'on dispose d'une équation paramétrique, et donc d'une fonction qui renvoie par exemple $x$ et $y$ en fonction d'un paramètre $t$ :

```
def f2(t):
    from math import cos, sin
    q = 7/4
    x = (q + 1) * cos(t) - cos((q+1) * t)
    y = (q + 1) * sin(t) - sin((q+1) * t)
    return x, y
```

On obtient le tracé pour $t \in [0, 8\pi]$ :

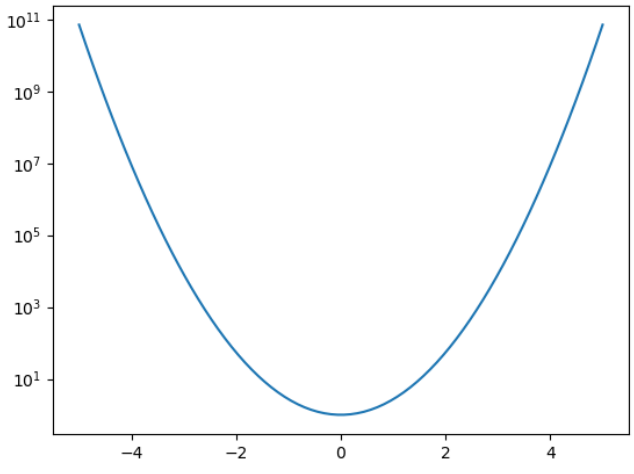
```
import math
import numpy as np
import matplotlib.pyplot as plt
vec_t = np.linspace(0, 8 * math.pi, 300)
points = [f2(t) for t in vec_t]
# points est une liste de couple.
# on veut la liste des x et la liste des y.
vec_x = [v[0] for v in points]
vec_y = [v[1] for v in points]
# autre méthode :
# vecx, vecy = zip(*points)
plt.plot(vec_x, vec_y, "b-")
plt.title("Épicycloïde")
plt.show()
```



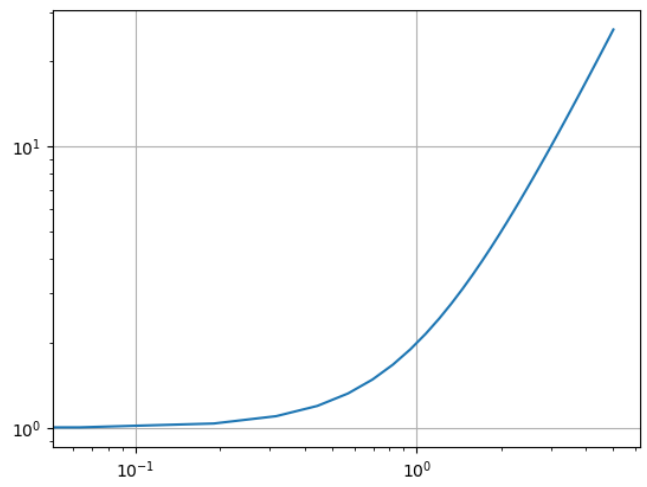

## 2 Autres types de courbes

Il existe nombre d'autres fonctions que `plot` pour tracer. On peut par exemple obtenir des graphiques *semilog* ou *loglog* :

```
import numpy as np
import matplotlib.pyplot as plt
lx = np.linspace(-5, 5, 80)
ly = np.exp(lx ** 2)
plt.semilogy(lx, ly)
plt.show()
```

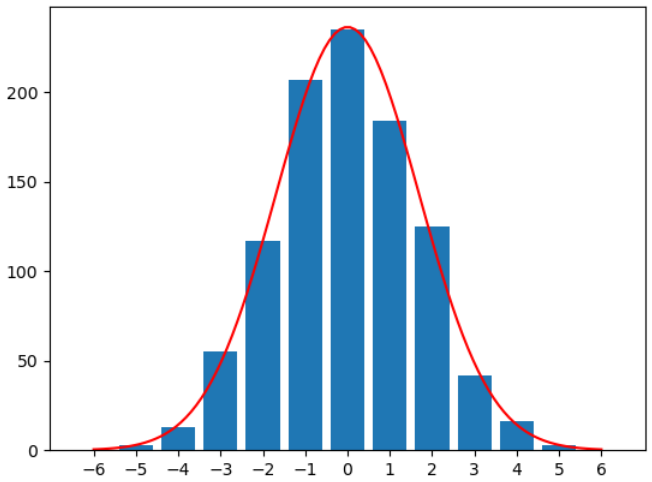


```
import numpy as np
import matplotlib.pyplot as plt
lx = np.linspace(-5, 5, 80)
ly = np.exp(np.log(lx ** 2 + 1))
plt.loglog(lx, ly)
plt.grid()
plt.show()
```



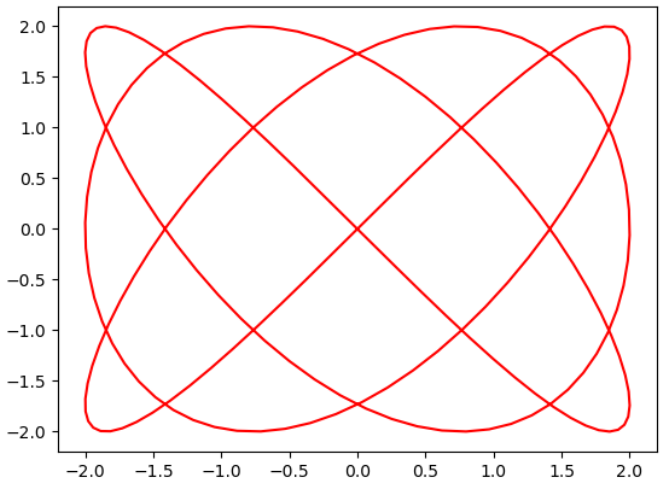
`matplotlib` sait faire des diagrammes à barre :

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

lx = np.linspace(-6, 6, 13)
ly = [0, 3, 13, 55, 117, 207, 235, 184,
      125, 42, 16, 3, 0]

lxf = np.linspace(-6, 6, 100)
lyth = 236.4 * np.exp(-0.5 *
                      (lxf / 1.6875) ** 2)

plt.bar(lx, ly)
plt.plot(lxf,lyth,"r")
plt.xticks(lx)
plt.show()
```



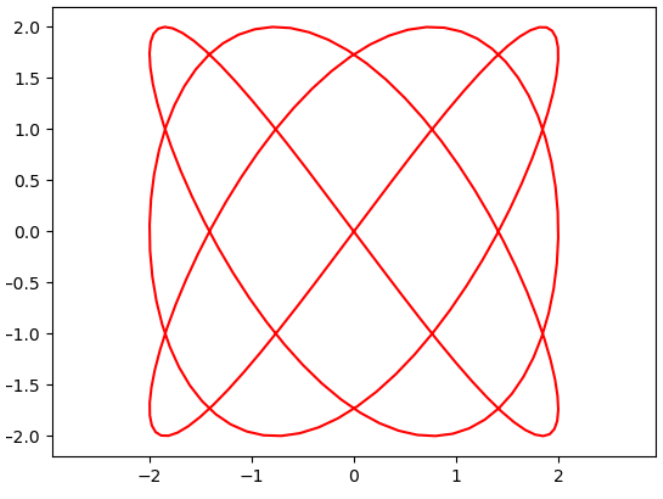

## 3 Ajustement des courbes

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

t = np.linspace(0, 2 * np.pi, 200)
x = 2 * np.sin(3 * t)
y = 2 * np.sin(4 * t)
plt.plot(x, y, "-r")
plt.show()
```



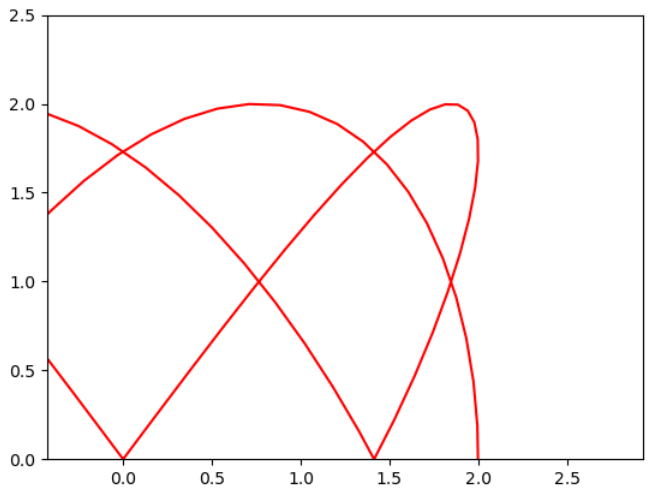

■ Repère orthonormé

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

t = np.linspace(0, 2 * np.pi, 200)
x = 2 * np.sin(3 * t)
y = 2 * np.sin(4 * t)
plt.plot(x, y, "-r")
plt.axis("equal") # <=======
plt.show()
```



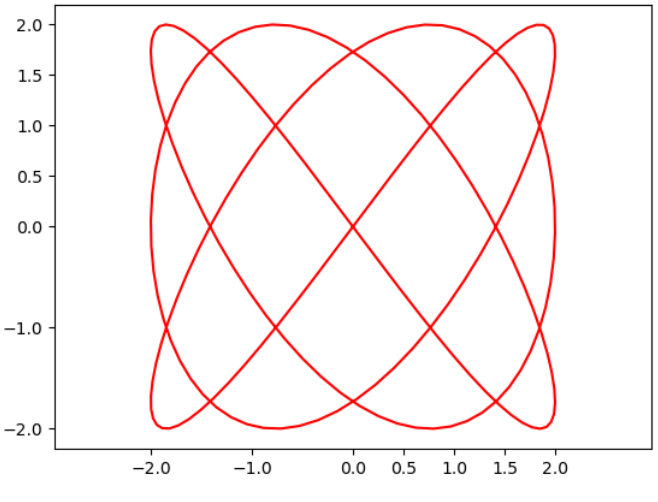

■ Choix du cadrage

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

t = np.linspace(0, 2 * np.pi, 200)
x = 2 * np.sin(3 * t)
y = 2 * np.sin(4 * t)
plt.plot(x, y, "-r")
plt.axis("equal")
plt.xlim(0, 2.5) # <===
plt.ylim(0, 2.5) # <===
plt.show()
```



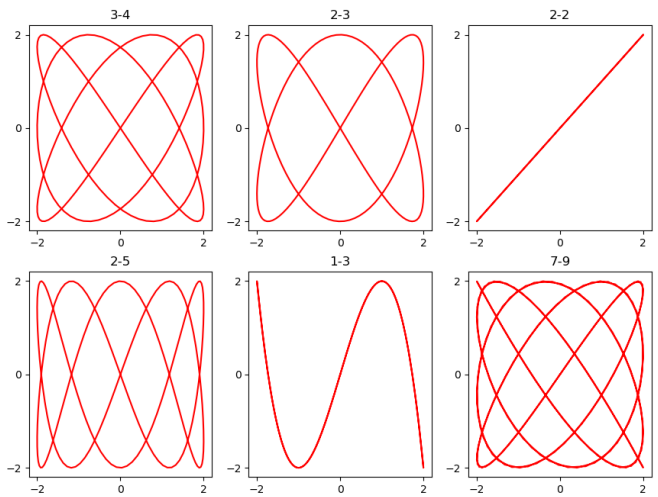

■ Position des ticks

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

t = np.linspace(0, 2 * np.pi, 200)
x = 2 * np.sin(3 * t)
y = 2 * np.sin(4 * t)
plt.plot(x, y, "-r")
plt.axis("equal")
ticks = [-2, -1, 0, 0.5, 1, 1.5, 2]) # <==
plt.xticks(ticks)                    # <==
plt.yticks(ticks)                    # <==
plt.show()
```




■ Plusieurs axes sur un même graphe

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

t = np.linspace(0, 2 * np.pi, 200)

n = 1
for p,q in [(3, 4), (2, 3), (2, 2),
            (2, 5), (1, 3), (7, 9)]:
    x = 2 * np.sin(p * t)
    y = 2 * np.sin(q * t)
    plt.subplot(2, 3, n)      # <==
    plt.plot(x, y, "-r")
    plt.xticks([-2, 0, 2])
    plt.yticks([-2, 0, 2])
    plt.gca().set_title(p, "-", q)
    n = n + 1
plt.show()
```




## 4 Spécificités notebooks

On peut changer la taille des graphiques dans un notebook avec la commande ésotérique suivante :

```
import matplotlib.pyplot as plt
plt.rcParams.update({"figure.figsize": (30, 20)})
```

De même, on peut modifier la taille de la police utilisé (car l'emploi de la commande précédente peut donner des caractères trop petits) :

```
plt.rcParams.update({'font.size': 22})
```

Problème : tous les graphiques ultérieurs seront modifiés. On peut donc prendre soin de sauvegarder les paramètres par défaut pour les restaurer ensuite :

```
import matplotlib.pyplot as plt
svtaille = plt.rcParams.copy()
plt.rcParams.update({'font.size': 22})
plt.rcParams.update({"figure.figsize": (30, 20)})
# Tracés
# ...
# Restauration
plt.rcParams = svtaille # Pas terrible... mais j'ai pas mieux pour le moment :)
```

## 5 Quelques règles pour obtenir de bons tracés

Lorsqu'on trace des points expérimentaux, il est généralement préférable de matérialiser ces points sur la courbe. Par exemple, si on trace 10 points, et qu'ils sont alignés, on aura simplement un segment. Impossible de savoir alors où sont placés les points sur le segment. Il faut donc utiliser comme style de tracé : `'-o'` par exemple, ou `'o'`, mais pas `'-'`.

Au contraire, pour des tracés dont on a une expression analytique (pas des mesures), on gagne souvent en lisibilité à ne pas matérialiser les points de tracé qui n'ont pas de signification particulière (et on utilise alors le style `'-'` par exemple).

Si on souhaite avoir plusieurs tracé dans un notebook, chaque tracé étant produit par un tour de boucle, on n'est pas obligé d'utiliser `subplots`. On peut tracer normalement, chaque appel à `plt.show()` produisant un nouveau graphique (si on met `plt.show()` dans la boucle, on a donc plusieurs graphiques indépendants).