Python : numpy et matplotlib avec une variable

Vidéo ■ partie 2.1. Numpy
Vidéo ■ partie 2.2. Matplotlib

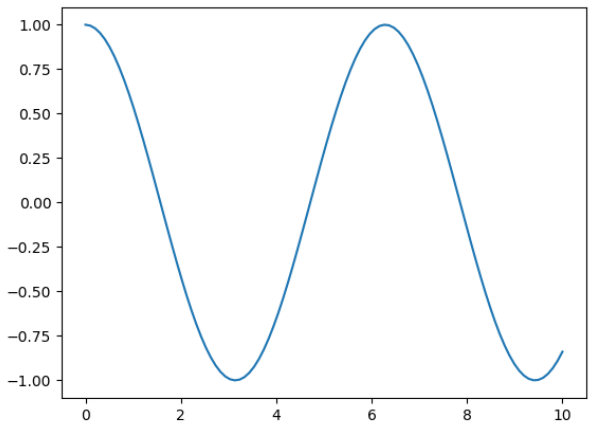
Le but de ce court chapitre est d'avoir un aperçu de deux modules Python : numpy et matplotlib. Le module numpy aide à effectuer des calculs numériques efficacement. Le module matplotlib permet de tracer des graphiques.

Ce chapitre est un chapitre plutôt technique, il suppose une connaissance de base de *Python*. Dans un premier temps, nous allons uniquement nous intéresser aux fonctions d'une variable. Il faut prendre garde à ne pas trop s'attarder sur les détails et les multiples fonctionnalités de ces deux modules. Vous pourrez revenir vers ce chapitre en fonction de vos besoins futurs.
Voici un aperçu du code le plus simple que l'on puisse écrire pour tracer une fonction. Il s'agit de tracer le graphe de la fonction $x \mapsto \cos(x)$ sur $[0, 10]$.

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

X = np.linspace(0,10,num=100)
Y = np.cos(X)

plt.plot(X,Y)
plt.show()
```



Attention ! *numpy* et *matplotlib* ne sont pas toujours installés avec votre distribution *Python*. Vous devrez peut-être les installer vous-même.

1. Numpy (une variable)

On appelle le module *numpy* et on le renomme de façon raccourcie « np » :

```
import numpy as np
```

1.1. Définition d'un vecteur

Les principaux objets de *numpy* sont des « tableaux ». Dans ce chapitre nous étudierons seulement les tableaux à une dimension que l'on appelle des ***vecteurs***.

- Définition à partir d'une liste :

```
X = np.array([1,2,3,4])
```

- Affichage par `print(X)` :

```
[1 2 3 4]
```

  Cela ressemble beaucoup à une liste, mais ce n'est pas une liste usuelle de *Python*. Le type de `X` est `numpy.ndarray`. Notez qu'il n'y a pas de virgules dans l'affichage du vecteur.
- Définition d'une suite d'éléments avec `arange()`. Cette fonction a le même comportement que la fonction classique `range()` avec en plus la possibilité d'utiliser un pas non entier. Par exemple :

```
np.arange(1,8,0.5)
```

  renvoie :

```
[1. 1.5 2. 2.5 3. 3.5 4. 4.5 5. 5.5 6. 6.5 7. 7.5]
```

- Définition d'une division d'intervalle avec `linspace()`. C'est l'une des méthodes les plus utilisées. Elle s'utilise avec la syntaxe `linspace(a,b,num=n)` pour obtenir une subdivision régulière de $[a, b]$ de n valeurs (donc en $n-1$ sous-intervalles). Exemple :

```
np.linspace(0,1,num=12)
```

  vaut :

```
[0.         0.09090909 0.18181818 0.27272727 0.36363636 0.45454545
 0.54545455 0.63636364 0.72727273 0.81818182 0.90909091 1.        ]
```

1.2. Opérations élémentaires

Voici les opérations élémentaires utiles pour manipuler des vecteurs. Elles ont toutes la particularité d'agir sur les coordonnées.
Définissons un vecteur X par :

```
X = np.array([1,2,3,4])
```

- Multiplication par un scalaire. Par exemple `2*X` renvoie le vecteur `[2 4 6 8]`.
- Addition d'une constante. Par exemple `X+1` renvoie le vecteur `[2 3 4 5]`.
- Carré. Par exemple `X**2` renvoie le vecteur `[1 4 9 16]`.
- Inverse. Par exemple `1/X` renvoie le vecteur `[1. 0.5 0.33333333 0.25]`.
- Somme. Par exemple `np.sum(X)` renvoie le nombre 10.
- Minimum, maximum. Par exemple `np.min(X)` renvoie 1 et `np.max(X)` renvoie 4.

Si Y est un autre vecteur défini par `Y = np.array([10,11,12,13])` alors :

- Somme terme à terme. Par exemple `X+Y` renvoie `[11 13 15 17]`.
- Produit terme à terme. Par exemple `X*Y` renvoie `[10 22 36 52]`.

1.3. Définition d'un vecteur (suite)

Voici plusieurs façons d'initialiser un vecteur.

- Avec des zéros. Exemple : `np.zeros(5)` renvoie `[0. 0. 0. 0. 0.]`.
- Avec des uns. Exemple : `X = np.ones(5)`, alors `X` vaut `[1. 1. 1. 1. 1.]`. Ainsi `7*X` renvoie `[7. 7. 7. 7. 7.]`.
- Avec des nombres aléatoires entre 0 et 1. Exemple : `np.random.random(5)` renvoie un vecteur dont les coordonnées sont tirées aléatoirement à chaque appel, on obtient par exemple :
  `[0.21132407 0.30685886 0.94111979 0.39597993 0.63275735]`

1.4. Utilisation comme une liste

Prenons l'exemple du vecteur X :

```
[1.  1.11111111 1.22222222 1.33333333 1.44444444 1.55555556 1.66666667
 1.77777778 1.88888889 2. ]
```

défini par la commande :

```
X = np.linspace(1,2,num=10)
```

- Récupérer une coordonnée. `X[0]` renvoie le premier élément de `X`, `X[1]` renvoie le second élément, etc.
- Longueur d'un vecteur. `len(X)` renvoie le nombre d'éléments. Une commande similaire est `np.shape(X)`.
- Parcourir tous les éléments. Deux méthodes :

```
for x in X:
    print(x)
```

```
for i in range(len(X)):
    print(i,X[i])
```

1.5. Application d'une fonction

Avec *numpy*, on peut appliquer une fonction directement sur chaque coordonnée d'un vecteur.
Prenons l'exemple de :

```
X = np.array([0,1,2,3,4,5])
```

- Racine carré. Par exemple `np.sqrt(X)` renvoie le vecteur `[0. 1. 1.41421356 1.73205081 2. 2.23606798]`. Pour chaque composante du vecteur X on a calculé sa racine carrée.
- Puissance. Par exemple `X**2` renvoie le vecteur `[0 1 4 9 16 25]`.
- Les fonctions mathématiques essentielles sont implémentées ainsi que la constante π : `np.pi`. Par exemple `np.cos(X)` (calcul des cosinus, unité d'angle le radian) ou bien `np.cos(2*np.pi/360*X)` (calcul des cosinus, unité d'angle le degré).

Pourquoi est-il est important d'utiliser `np.sqrt(X)` plutôt qu'une boucle du type `for x in X: np.sqrt(x)` ? D'une part le code est plus lisible, mais surtout les calculs sont optimisés. En effet, *numpy* profite de la présence de plusieurs processeurs (ou cœurs) dans la machine et effectue les calculs en parallèle. Les calculs sont beaucoup plus rapides.
Attention ! *numpy* se substitue au module `math` qu'il ne faut pas utiliser. (Par exemple la commande `math.cos(X)` renverrait une erreur.)
Voici les principales fonctions utiles pour ce cours. Il en existe plein d'autres !

`x ** a`	$x^a \quad (a \in \mathbb{R})$
`sqrt(x)`	$\sqrt{x}$
`exp(x)`	$\exp x$
`log(x)`	$\ln x$ logarithme népérien
`log10(x)`	$\log x$ logarithme décimal
`cos(x), sin(x), tan(x)`	$\cos x$, $\sin x$, $\tan x$ en radians
`arccos(x), arcsin(x), arctan(x)`	$\arccos x$, $\arcsin x$, $\arctan x$ en radians
`cosh(x), sinh(x), tanh(x)`	$\mathrm{ch}\, x$, $\mathrm{sh}\, x$, $\tanh x$

2. Matplotlib (une variable)

Nous allons tracer les graphes de fonctions d'une variable à l'aide des modules *matplotlib* et *numpy*.

2.1. Un exemple

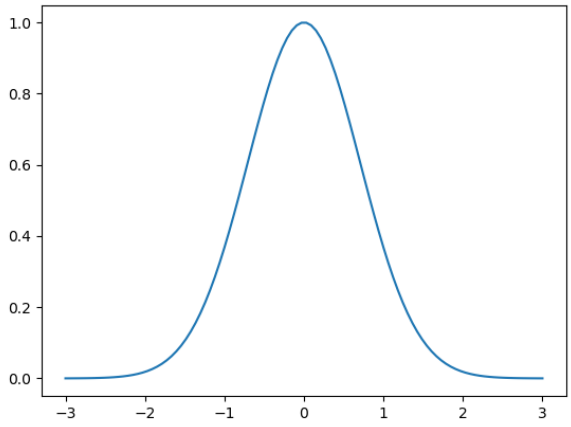
Voici comment tracer la fonction $f : [-3,3] \to \mathbb{R}$ définie par $f(x) = e^{-x^2}$, c'est la courbe de Gauss.

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

def f(x):
    return np.exp(-x**2)

a,b = -3,3
X = np.linspace(a,b,num=100)
Y = f(X)

plt.plot(X,Y)
plt.show()
```

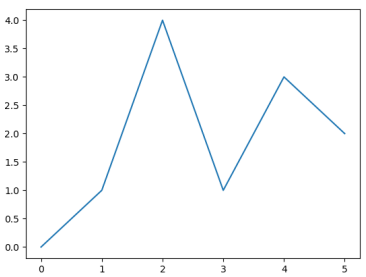


2.2. Tracé de fonctions point par point

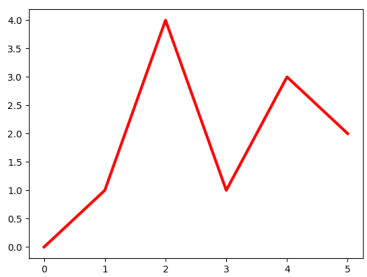
Soient deux vecteurs :

`X = np.array([0,1,2,3,4,5])` `Y = np.array([0,1,4,1,3,2])`

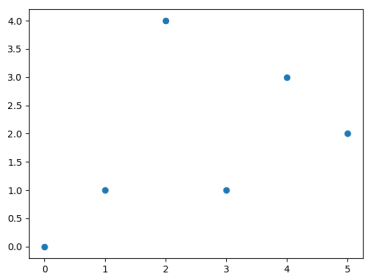
Alors la commande `plt.plot(X,Y)` affiche les points (x,y) pour x parcourant X et y parcourant Y et les relie entre eux (`plt` est le nom raccourci que nous avons donné au sous-module `pyplot` du module *matplotlib*). Lorsque les points sont suffisamment rapprochés cela donne l'impression d'une courbe lisse.



`plt.plot(X,Y)`



`plt.plot(X,Y,linewidth=3,color='red')`



`plt.scatter(X,Y)`

On peut bien sûr changer le style du tracé ou n'afficher que les points.

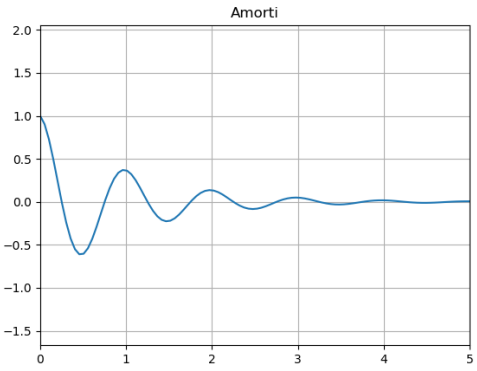
2.3. Axes

On peut exiger que le repère soit orthonormé, ajouter une grille pour plus de lisibilité. Enfin, la commande `plt.savefig()` permet de sauvegarder l'image.

```
def f(x):
    return np.exp(-x) * np.cos(2*np.pi*x)

a,b = 0,5
X = np.linspace(a,b,num=100)
Y = f(X)

plt.title('Amorti') # titre
plt.axis('equal')   # repère orthonormé
plt.grid()          # grille
plt.xlim(a,b)       # bornes de l'axe des x
plt.plot(X,Y)
plt.savefig('amorti.png')
plt.show()
```

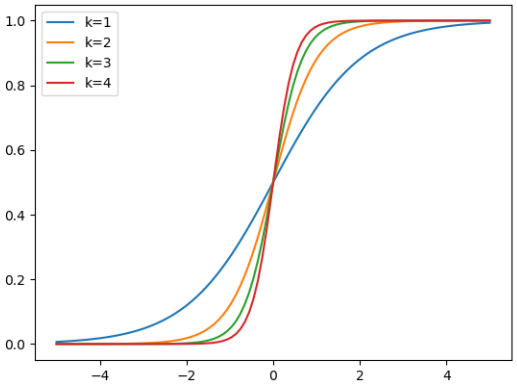


2.4. Plusieurs tracés

On peut effectuer plusieurs tracés sur le même dessin. Ici des fonctions sigmoïdes

$$f_k(x) = \frac{1}{1+e^{-kx}}$$

avec un paramètre k qui varie.




```
def f(x,k):
    return 1/(1+np.exp(-k*x))

a,b = -5,5
X = np.linspace(a,b,num=100)

for k in range(1,5):
    Y = f(X,k)
    plt.plot(X,Y, label="k={}".format(k))

plt.legend()
plt.show()
```