Leçon d'informatique : Tracé de courbes. Matplotlib et Numpy

S. Benlhajlahsen - PCSI$_1$

Sommaire

I Tracer la courbe représentative d'une fonction

I.A Représenter des listes de points

Utilisation du module `matplotlib` :

La page officielle de ce module est https://matplotlib.org/.

Ce module permet de tracer des courbes de toutes sortes en 2D et même en 3D. Les options sont très nombreuses et la documentation officielle fait plus de 1300 pages ! Par convention, on importera cette bibliothèque comme en figure 1.

```
1 import matplotlib.pyplot  as plt
```

FIGURE 1

Synthaxe : La synthaxe générale est donnée en figure 2. C'est la synthaxe de base. On peut ensuite ajouter des éléments esthétiques (couleur, quadrillage, ...). On obtient la figure 3. Si on note x_i les éléments de la liste `Lx` et y_i les éléments de `Ly`, on voit en figure 4 que l'on a joint les points de coordonnées (x_i, y_i).

```
1 import matplotlib.pyplot as plt
2 Lx = [1, 2, 3, 4]              # liste des abscisses
3 Ly = [1, 4, 2, 3]              # listes des ordonnées
4 plt.figure()                   # Ouverture d'une nouvelle fenêtre
5 plt.plot(Lx, Ly)               # Tracé
6 plt.show()                     # affiche la fenêtre graphique
```

FIGURE 2

I.B Représenter la courbe représentative d'une fonction

Un premier exemple : On souhaite représenter la courbe représentative de la fonction $f : x \mapsto \sin(x)$. On prend alors une liste `Lx` composée de 7 points entre 0 et 6 et on crée ensuite une liste `Ly` composée des images par sinus. On pourra se reporter aux figures 5 et 6.

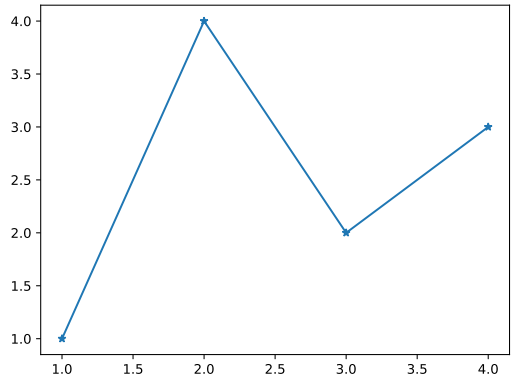

FIGURE 3

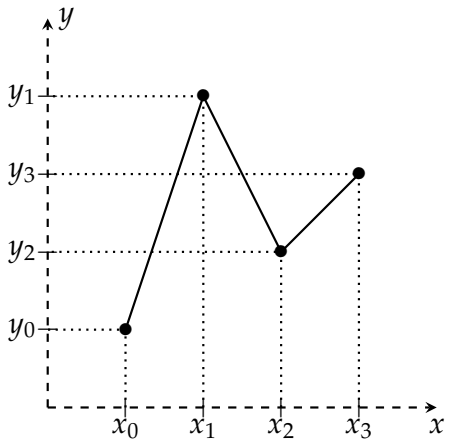

FIGURE 4

```python
import matplotlib.pyplot as plt
from math import sin
Lx = [0, 1, 2, 3, 4, 5, 6]
Ly = [sin(x) for x in Lx]
plt.figure()
plt.plot(Lx, Ly)
plt.show()
```

FIGURE 5

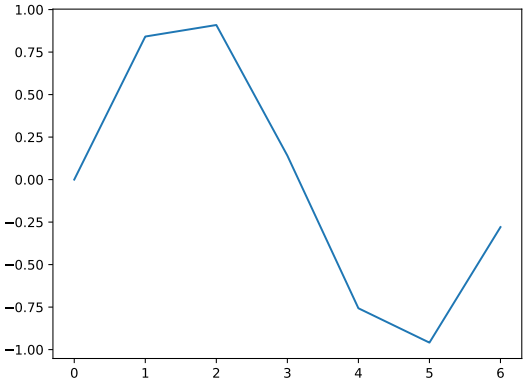

FIGURE 6

Amélioration : On peut améliorer la représentation graphique en augmentant le nombre de points. Si on souhaite tracer N points $(x_k)_{k\in[0,N-1]}$ régulièrement repartis entre a et b, ils seront séparés de $h = \frac{b-a}{N-1}$ et suivront la loi :

$$\forall k \in [0, N-1], \quad x_k = a + k \cdot h = a + k \cdot \frac{b-a}{N-1}$$

On voit que $x_0 = a$ et $x_{N-1} = b$. On propose alors le scripts de la figure 11 qui donne la courbe de la figure 12 avec $N = 101$ points de calculs $a = 0$ et $b = 6$.

II Utiliser des tableaux `numpy`

Le module `numpy` :

Le module `numpy` fournit un ensemble de méthodes adaptées au calcul scientifique. Par convention, on importera cette bibliothèque comme sur la figure 9.

Type `numpy.ndarray` : Le module `numpy` fournit un nouveau type de tableau/liste : le type `numpy.ndarray`. On voit en figure 10 qu'on a convertit la liste `[0,1,2,3,4]` en un tableau numpy.

```python
import matplotlib.pyplot as plt
from math import sin
Lx = [6*i/100 for i in range(101)]
Ly = [sin(x) for x in Lx]
plt.figure()
plt.plot(Lx, Ly)
plt.show()
```

FIGURE 7

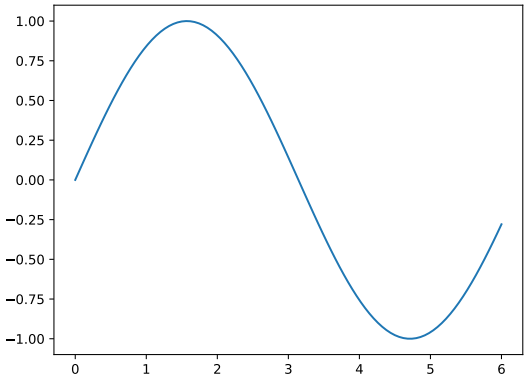

FIGURE 8

```python
import numpy  as np
```

FIGURE 9

Comparaison entre listes python et tableaux `numpy` : On raisonne sur la liste `L = [10, 20, 30]` et le tableau `numpy` `T = np.array([10, 20, 30])`.

- Les deux objets sont indexés entre 0 et `len(L)-1` ou `len(T)-1`;
- `L[0]` et `T[0]` renvoie 10.
- les listes python peuvent être hétérogènes, c'est-à-dire composées d'éléments de type différents comme `[1, "abc", True]` alors que les **tableaux `numpy` doivent être homogènes**;
- Les tableaux `numpy` ont une taille fixée.

```
>>> tab = np.array([0,1,2,3,4])
>>> type(tab)
<class 'numpy.ndarray'>
```

FIGURE 10

Méthodes du module `numpy` : `numpy` fournit quelques méthodes pratiques en calcul scientifique :

- la plupart des fonctions mathématiques de bases et les constantes commme `np.pi`, `np.sin`, `np.exp`, `np.log`[a]
- la méthode `np.linspace(a, b, N)` fournit un tableau `numpy` de N valeurs régulièrement espacées entre a et b (compris). Ainsi, `np.linspace(0, 5, 6)` renvoie `array([0., 1., 2., 3., 4., 5.])`.
- `np.arange(a,b,step)` a le même usage que `range`. Ainsi, `np.arange(0,6,1)` renvoie `array([0, 1, 2, 3, 4, 5])`.

a. Vous noterez que `np.log`, `np.arcsin`, `np.sqrt`...est le logarithme népérien et `np.log10` le logarithme décimal.

Avantage des tableaux `numpy` : la vectorisation L'avantage principal des tableaux numpy est le suivant : une opération sur le tableau `T` est appliqué directement aux élements du tableau.

- `3*T` renvoie un tableau de même taille où tous les éléments ont été multipliés par 3. Ainsi, si `T = np.array([1,2,3])`, alors `3*T` renvoie `array([3, 6, 9])`.
- `T**2` renvoie le tableau des carrés. Ainsi, `T**2` renvoie array([1, 4, 9]).

- `1/T` renvoie le tableau des inverses. Ainsi, `1/T` renvoie `array([1., 0.5, 0.33333333])`.
- On peut appliquer une fonction sur le tableau. Ainsi, `np.sin(T)` renvoie le tableau du sinus des éléments.
- Enfin, entre deux tableaux de même dimension, l'opération `+` ou l'opération `-` réalise l'addition deux à deux éléments ou la soustraction. Ainsi, `np.array([10, 20])+np.array([1, 2])` renvoie `np.array([11, 22])`.

On donne un exemple de tracé de fonction utilisant `numpy` pour la vectorisation en figure 11.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 Tx = np.linspace(0, 6, 101)
4 Ty = np.sin(Tx)
5 plt.figure()
6 plt.plot(Tx, Ty)
7 plt.show()
```

FIGURE 11

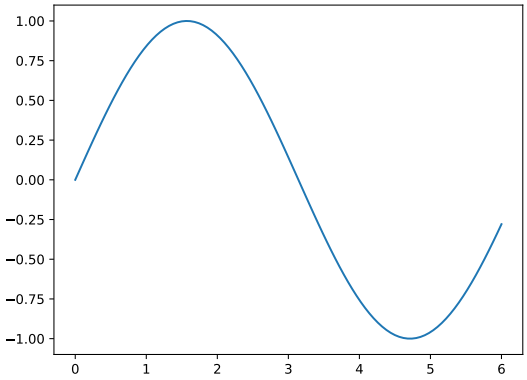

FIGURE 12

III Histogramme de données

Lancer de dé : On lance N fois un dé à six faces. On stocke les résultats de ces tirs dans une liste python ou dans un tableau `numpy`. Pour représenter les occurences de chaque face, on peut faire un histogramme (voir figures 15 et 16). Pour un histogramme, il faut préciser le nombre de « bâtons ou barres »[a]. Chaque barre a alors une taille correspondant au nombre d'occurence de chaque face.

a. *bins* en anglais

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 N = 20
4 L = []
5 for i in range(N):
6     L.append(np.random.randint(1,7))
7 T = np.array(L) # facultatif
8
9 plt.figure()
10 plt.hist(T, bins = 6, edgecolor="black" )
11 plt.show()
```

FIGURE 13 – Lancer de dé.

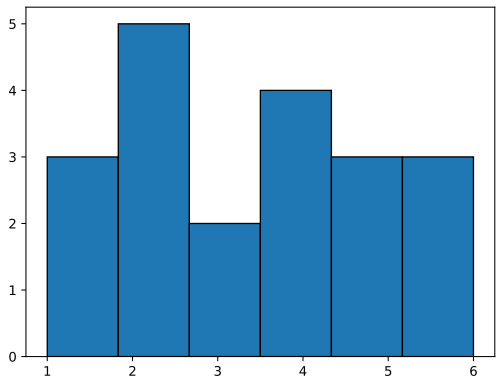

FIGURE 14 – Histogramme pour le lancer de dé.

Exemple de la gaussienne : On peut aussi faire un tirage qui suit une loi normale ou gaussienne qui s'écrit :

$$f : x \mapsto \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right)$$

où μ est la valeur moyenne et σ l'écart-type. On pourra se reporter aux figure 15 et 16.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  N = 100000
4  moy = 10
5  sigma = 1
6  L = []
7  for i in range(N):
8      L.append(np.random.normal(moy,sigma))
9  T = np.array(L) # facultatif
10
11 plt.figure()
12 plt.hist(T, bins = 20, edgecolor="black" )
13 plt.savefig("gaussienne.pdf")
14 plt.show()
```

FIGURE 15 – Tirage aléatoire avec une loi gaussienne.

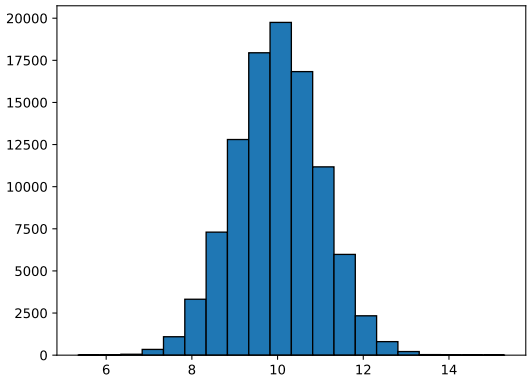

FIGURE 16 – Histogramme.

IV Représenter graphiquement les données d'un fichier

Données issues d'un capteur :

Supposons qu'un capteur enregistre ses données dans un fichier `donnees.txt` ou `donnees.csv`[a] composée de deux colonnes : le temps pour la première et une tension pour la deuxième. On en donne quelques lignes :

1,23	2,51
2,10	4,34
3,21	6,52
4,56	8,99

et l'allure du fichier :

```
1,23;2,51\n
2,10;4,34\n
3,21;6,52\n
4,56;8,99\n
```

a. Un fichier `.csv` est comparable à un fichier `.txt` qui organise les données sous forme de colonnes.

Quelques méthodes : Si on importe une ligne dans une variable `line`, on doit :

- enlever le caractère `"\n"` avec le hachage `line[:-1]` ;
- remplacer la virgule `","` par le point `"."` par l'instruction `line.replace("," , ".")` ;
- séparer les deux colonnes au niveau du point-virgule par l'instruction `line.split(";")` ;
- convertir une chaîne en flottant par l'instruction `float("1.23")`.

Le script de la figure 17 donne la courbe de la figure 18.

```python
import matplotlib.pyplot as plt
Lx = []
Ly = []
with open("donnees.csv", "r") as file:
    for line in file:
        t = line.replace(",", ".")[:-1].
            split(";")
        x = float(t[0])
        y = float(t[1])
        Lx.append(x)
        Ly.append(y)
plt.figure()
plt.plot(Lx, Ly, "bo", linestyle = "--")
plt.savefig("donnees.pdf")
plt.show()
```

FIGURE 17

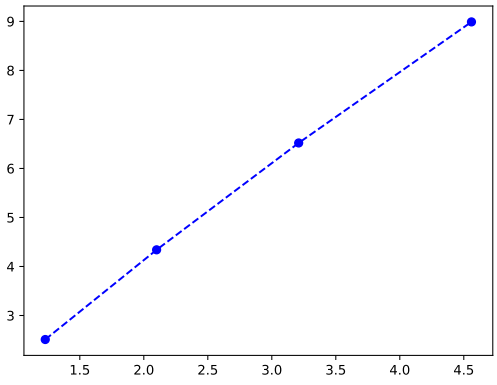

FIGURE 18