Gavroche

Zeste de savoir

Les bases de numpy et matplotlib

21 mai 2023

Table des matières

Introduction

Dans ce mini-tutoriel, on va s'intéresser aux bibliothèques Python numpy et matplotlib. Numpy permet la manipulation de grands volumes de données, sous forme de tableau, de manière bien plus efficace que Python. Matplotlib sert quant à elle, à afficher des graphiques comme des courbes, des histogrammes…

Ces fonctionnalités leur permettent d'être très utilisées dans de nombreux domaines comme la bio-informatique, le traitement du signal, l'intelligence artificielle… Il peut donc être intéressant d'en maîtriser les bases.

Prérequis
Maîtriser les bases Python: variable, boucle, liste, module…
Savoir ouvrir la ligne de commande
Objectif
Acquérir les bases numpy et de matplotlib

1. Installation

Je suppose que Python est déjà installé sur votre ordinateur. Pour obtenir numpy et matplotlib on utilise PIP, un gestionnaire de paquets, qui permet d'installer facilement de nombreux modules.

Une fois le terminal ouvert, on met d'abord à jour PIP en utilisant la commande `pip install --upgrade pip`. Ensuite on installe ces deux modules en se servant des commandes suivantes:

```
$ pip install matplotlib
$ pip install numpy
```

Dès que l'installation est finie, on peut importer ces deux modules. Pour cela, on utilise les instructions:

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

Si l'installation a bien fonctionné, alors ce code ne devrait pas générer d'erreurs.

Par convention, on importe numpy et matplotlib.pyplot à travers les alias "np" et "plt".

Voilà, vous êtes prêt à apprendre numpy et matplotlib! 

2. Partons à la découverte de numpy

2.1. Opérations de bases sur les tableaux

Après avoir importé numpy, on peut commencer à créer nos premiers tableaux. Pour cela, il suffit d'utiliser l'instruction `np.array`, qui prend en paramètre la liste des nombres qui seront contenus dans le tableau, par exemple si on veut initialiser un tableau avec les valeurs 1, 1.5, 2 et 2.5:

```
>>> import numpy as np
>>>
>>> tableau = np.array([1, 1.5, 2, 2.5])
>>> tableau
array([1, 1.5, 2, 2.5])
```

Les différences majeures entre les listes proposées par Python et les tableaux de Numpy sont:

— dans un tableau, tous les éléments doivent être du même type ;

— les tableaux ont une longueur fixe, on ne peut directement ajouter une nouvelle valeur ;
— il y a un gain élevé de performance avec les tableaux.

> Bien qu'il soit possible d'utiliser des tableaux contenant autre chose que des nombres, dans ce tutoriel, nous allons uniquement nous intéresser à ce cas.

Ensuite, on parcourt le tableau avec une boucle `for`, comme pour une liste:

```
>>> tableau = np.array([1, 2, 4])
>>> for i in tableau:
        print(i)

1
2
4
```

On peut aussi accéder manuellement à un élément du tableau (comme avec les listes, on commence à compter à partir de 0) et connaitre le nombre d'objet contenu dans le tableau:

```
>>> tableau = np.array([1,2,3,4])
>>> tableau[0]
1
>>> tableau[2]
3
>>> len(tableau)
4
```

Pour sélectionner une partie du tableau, on peut utiliser les *slices*, voici des exemples simples, si vous voulez en savoir plus vous pouvez regarder ce tutoriel .

```
>>> tableau = np.array([1,2,3,4])
>>> tableau[0:2]
array([1, 2])
>>> tableau[1:]
array([2, 3, 4])
>>> tableau[:-1]
array([1, 2, 3])
```

Cette bibliothèque permet d'ajouter, de supprimer et d'insérer des éléments dans les tableaux:

```
>>> tableau = np.array([1, 2, 4, 6])
>>> np.append(tableau, 8) # Ajoute l'élément 8 à la fin du tableau
array([1, 2, 4, 6, 8])
>>>
>>> np.delete(tableau, 1) # Supprime l'élément à l'index 1 (ici
    l'entier 2)
array([1, 4, 6])
>>>
>>> np.insert(tableau, 2, 3) # Insère l'élément 3 à l'index 2
array([1, 2, 3, 4, 6])
```

Ces fonctions ne modifient pas directement le tableau, elles retournent un nouveau tableau.

Les formules mathématiques s'utilisent directement sur les tableaux, celles-ci seront alors appliquées à tous les éléments du tableau:

```
>>> tableau = np.array([1, 2, 4, 6])
>>> tableau ** 2 + 1 # Met au carré puis ajoute 1
array([2, 5, 17, 37])
```

On peut employer des fonctions comme le cosinus ou la racine carrée: `np.cos`, `np.sqrt`. La liste de toutes les fonctions se trouve ici .

2.2. Générer des tableaux

Pour générer un tableau avec des données comprises dans un intervalle, on utilise principalement deux fonctions: arange et linspace . La première est analogue à la fonction `range`. La seconde prend trois paramètres: le début de l'intervalle (inclus), la fin de l'intervalle (non inclus) et le nombre de valeurs que l'on veut entre ces deux valeurs. Par exemple:

```
>>> np.linspace(0, 10, 5) # 5 nombres compris entre 0 et 10
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
>>> np.arange(1, 5, 0.5) # Nombres de 1 et à 5 (exclu) avec un pas
    de 0.5
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

Numpy permet aussi de créer un tableau avec des données aléatoires, en utilisant la fonction `np.random.rand(n)` qui va renvoyer un tableau avec n valeurs comprises entre 0 et 1:

```
>>> np.random.rand(3)
array([0.06246263, 0.4517823 , 0.66618466])
```

On trouve la fonction `np.random.randint(max, size=n)` qui va retourner un tableau avec n valeurs comprises entre 0 et max (exclu).

```
>>> np.random.randint(10, size=5)
array([2, 8, 3, 7, 8])
```

Cette bibliothèque permet de générer des tableaux de nombres suivant une loi de probabilité:

```
>>> np.random.normal(loc=10, scale=0.5, size=10) # Tableau de 10 éléments
array([ 9.88225653, 10.4609955 ,  9.51217795, 10.50906955, 10.52551522,
        9.1359539 ,  9.60488407, 10.53728603,  9.55069133, 10.4908086 ])
>>>
>>> np.random.binomial(n=10, p=0.5, size=10)
array([2, 7, 5, 5, 6, 6, 4, 7, 7, 3])
```

Numpy gère de nombreuses autres lois de probabilités, pour en savoir plus, vous pouvez regarder cette page

Enfin, voici une liste de fonctions biens pratiques pour générer des tableaux:

```
>>> np.zeros(3) # Un tableau de longueur 3 avec que le chiffre 0
array([0., 0., 0.])
>>> np.ones(4) # Un tableau de longueur 4 avec que le chiffre 1
array([1., 1., 1., 1.])
```

2.3. Rechercher dans un tableau

La fonction `np.where` prend en paramètre une condition sur les valeurs d'un tableau et va retourner l'index des éléments qui vérifient la condition. La fonction `np.extract` renvoie les éléments qui vérifient la condition. Ce qui donne:

```
>>> tableau = np.arange(1, 25) # Tableau avec les nombres entiers
    de 1 à 24
>>>
>>> np.where(tableau % 5 == 0) # L'index des nombres divisibles
    par 5
(array([4,  9, 14, 19])
>>>
>>> np.extract(tableau % 5 == 0, tableau) # Les nombres divisibles
    par 5
array([5, 10, 15, 20])
```

Pour la fonction `extract`, il faut préciser le nom du tableau en argument.

2.4. Fonctions biens utiles

Numpy propose aussi d'autres fonctions, voici une liste non exhaustive:

```
>>> tableau = np.array([5, 1, 6, 9])
>>> np.sort(tableau) # Trie le tableau
array([1, 5, 6, 9])
>>>
>>> np.mean(tableau) # La moyenne
5.25
>>> np.median(tableau) # La médiane
5.5
>>> np.count_nonzero(tableau) # Le nombre de valeurs différentes
    de 0
4
```

Ces fonctions ont d'autres arguments, pour en savoir plus: sort , mean , median ...

2.5. Les tableaux de deux dimensions

Numpy permet de créer des tableaux de plusieurs dimensions:

```
>>> tableau = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> tableau
array([[1, 2, 3],
```

```
4         [4, 5, 6],
5         [7, 8, 9]])
```

Dans ce mini-tuto, on se limite à deux dimensions, même si numpy gère les tableaux de dimensions supérieures.

Pour parcourir ce tableau, on utilise alors deux boucles `for` imbriquées.

```
1  >>> tableau = np.array([[1, 2, 3], [4, 5, 6]])
2  >>> for i in tableau:
3  ...     for j in i:
4  ...             print(j)
5  ...
6  1
7  2
8  3
9  4
10 5
11 6
```

Si on veut accéder à un élément en particulier, numpy possède l'instruction `tableau[ligne, colonne]`:

```
1  >>> tableau = np.array([[1, 2, 3], [4, 5, 6]])
2  >>> tableau[1, 2]
3  6
4  >>> tableau[0, 0]
5  1
```

Dans le cas où on souhaiterait générer un tableau d'entiers compris entre 0 et n, il faudra utiliser la fonction `np.random.randint` avec le paramètre `size`, une liste contenant le nombre de lignes et de colonnes du tableau:

```
1  >>> np.random.randint(5, size=[3, 2]) # Nombres compris entre 0
       (inclus) et 5 (exclus)
2  array([[1, 0],
3         [2, 3],
4         [0, 4]])
```

2.6. Fonctions spécifiques aux tableaux de plusieurs dimensions

Parmi les fonctions utiles pour les tableaux de 2 dimensions, on trouve la fonction `transpose` qui va échanger les lignes et les colonnes:

```
>>> tableau = np.array([[1, 2, 3], [4, 5, 6]])
>>> tableau
array([[1, 2, 3],
       [4, 5, 6]])
>>>
>>> np.transpose(tableau)
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Numpy propose la fonction `reshape` qui permet de changer les dimensions d'un tableau, sans changer les valeurs, si la taille est compatible:

```
>>> tableau = np.array([1, 2, 3, 4, 5, 6])
>>> tableau.reshape(3, 2) # 3 lignes et 2 colonnes
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> tableau.reshape(2, 3) # 2 lignes et 3 colonnes
array([[1, 2, 3],
       [4, 5, 6]])
>>> tableau.reshape(3, 3) # Impossible, il faudrait que le tableau
    ait une longueur de 9
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    t.reshape(3, 3)
ValueError: cannot reshape array of size 6 into shape (3,3)
```

2.7. Aller plus loin avec numpy

Vous devriez donc avoir les bases de numpy, mais il vous reste de nombreuses choses à apprendre, comme:

- la fonction load qui permet de créer un tableau à partir d'un fichier
- les fonctions de bases pour l'algèbre linéaire : `vdot` pour le produit scalaire, `solve` pour résoudre un système ...

Sinon, n'hésitez pas à consulter la documentation .

3. Traçons des graphiques avec matplotlib

3.1. Une première courbe

Pour tracer une courbe, il faut d'abord utiliser la fonction `plt.plot`, qui prend en argument deux tableaux de même longueur: le premier correspond aux abscisses des points et le second aux ordonnées des points. Puis on utilise la fonction `plt.show` pour afficher le graphique. Par exemple, pour afficher une courbe représentant la fonction x^2 entre 0 et 20, on utilise le code ci-dessous :

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

x = np.linspace(0, 20, 200)
y = x ** 2
plt.plot(x, y)
plt.show()
```

On obtient alors ce graphique:

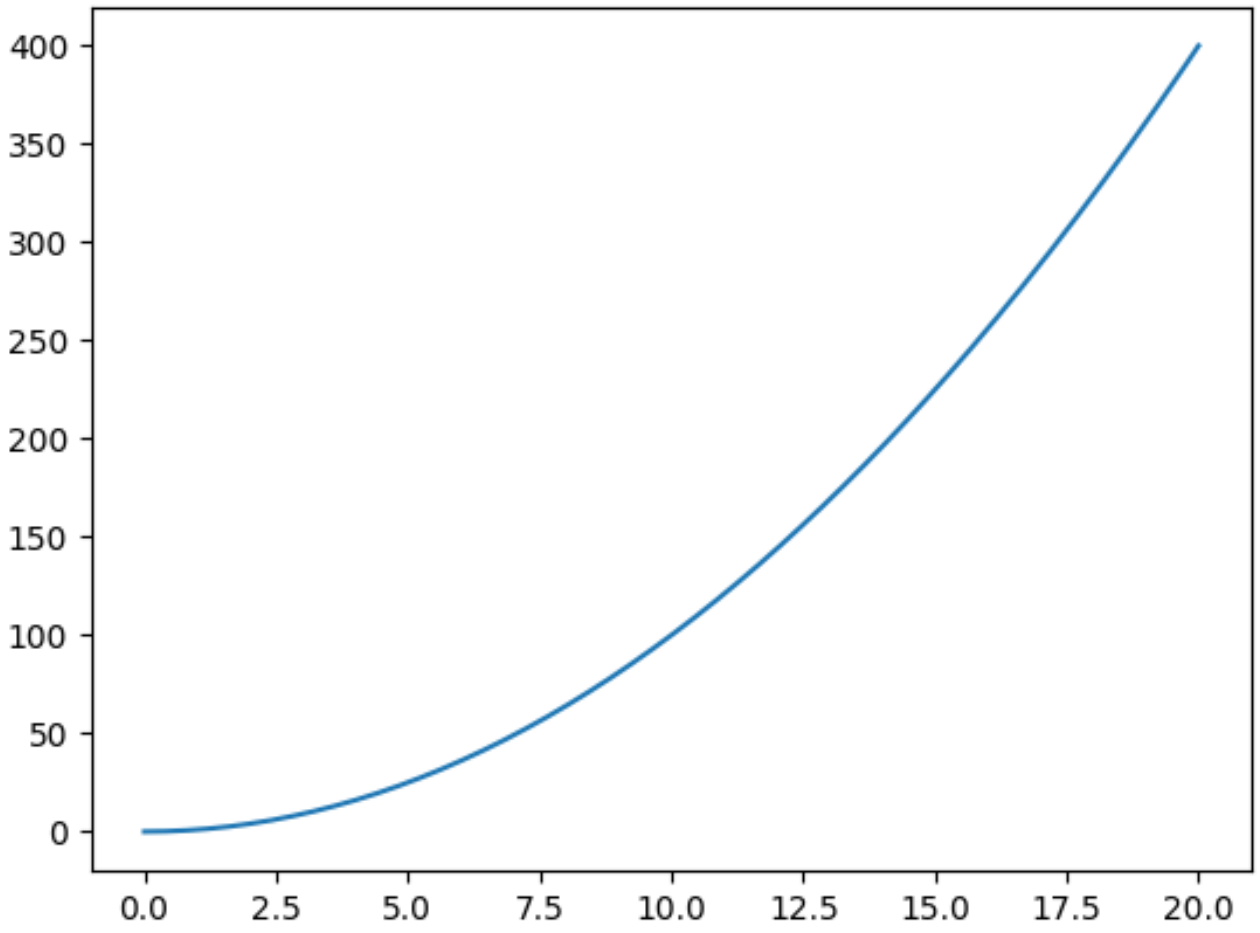

FIGURE 3.1. – Une courbe affichée avec matplotlib

On peut utiliser plusieurs fois la fonction `plt.plot` pour afficher plusieurs courbes à la fois. Dans ce cas, il ne faut pas oublier de placer la fonction `show` à la fin.

3.2. Plus loin avec la fonction plot

La fonction `plot` permet de changer l'apparence de nos courbes. Il doit alors utiliser un nouveau paramètre: une chaine de caractère. Celle-ci doit être sous la forme: `(couleur)(type de point)(type de ligne)`. Voici des tableaux qui résument ce que peuvent prendre en argument les 3 paramètres:

 Couleur

 Point

 Ligne

Si on précise le type de point mais pas le type de ligne, on a un nuage de points. Si on ne précise ni l'un, ni l'autre, on a une courbe.

Par exemple, ce code va afficher une courbe verte, avec une ligne pointillée, et des points en forme de cercle:

```
1 x = np.linspace(0, 20, 20)
2 y = x ** 2
3
4 plt.plot(x, y, 'go:')
5 plt.show()
```

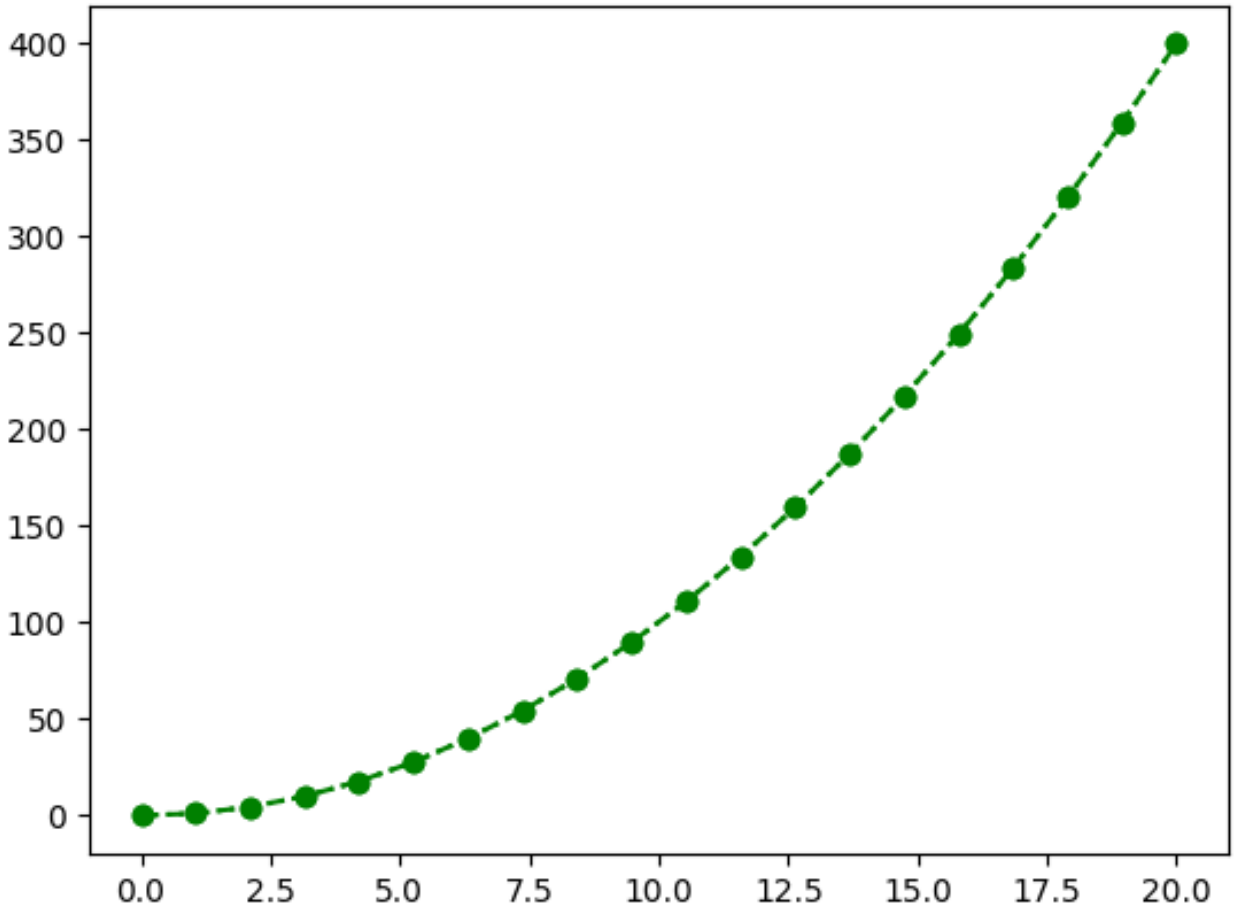

FIGURE 3.2. – Une courbe dont on a changé les caractéristiques

La fonction `plot` possède d'autres arguments, pour en savoir plus, vous pouvez regarder la documentation .

3.3. Afficher une légende

En science, le titre et les noms des axes doivent toujours apparaitre dans un graphique. Matplotlib propose trois fonctions, qui prennent chacune en argument une chaine de caractère: `plt.ylabel`, `plt.xlabel` et `plt.title`.

Cette libraire nous permet aussi d'ajouter une légende pour chaque courbe. Il faut alors ajouter un nouvel argument aux fonctions `plot`: `label="Légende de la courbe"`, et appeler la fonction `plt.legend`.

On a aussi la fonction `grid()` qui sert à afficher une grille (je sais c'est dur 🍊).

Voici un code qui montre l'usage de ces fonctions:

```
x = np.linspace(0, 20, 20)
y1 = x ** 2
y2 =  2 * x

```

```
plt.plot(x, y1, 'g', label="fonction x²")
plt.plot(x, y2, 'r', label="fonction 2x")

plt.xlabel('axe x')
plt.ylabel('axe y')
plt.title('Graphique montrant deux fonctions')
plt.legend()
plt.grid()

plt.show()
```

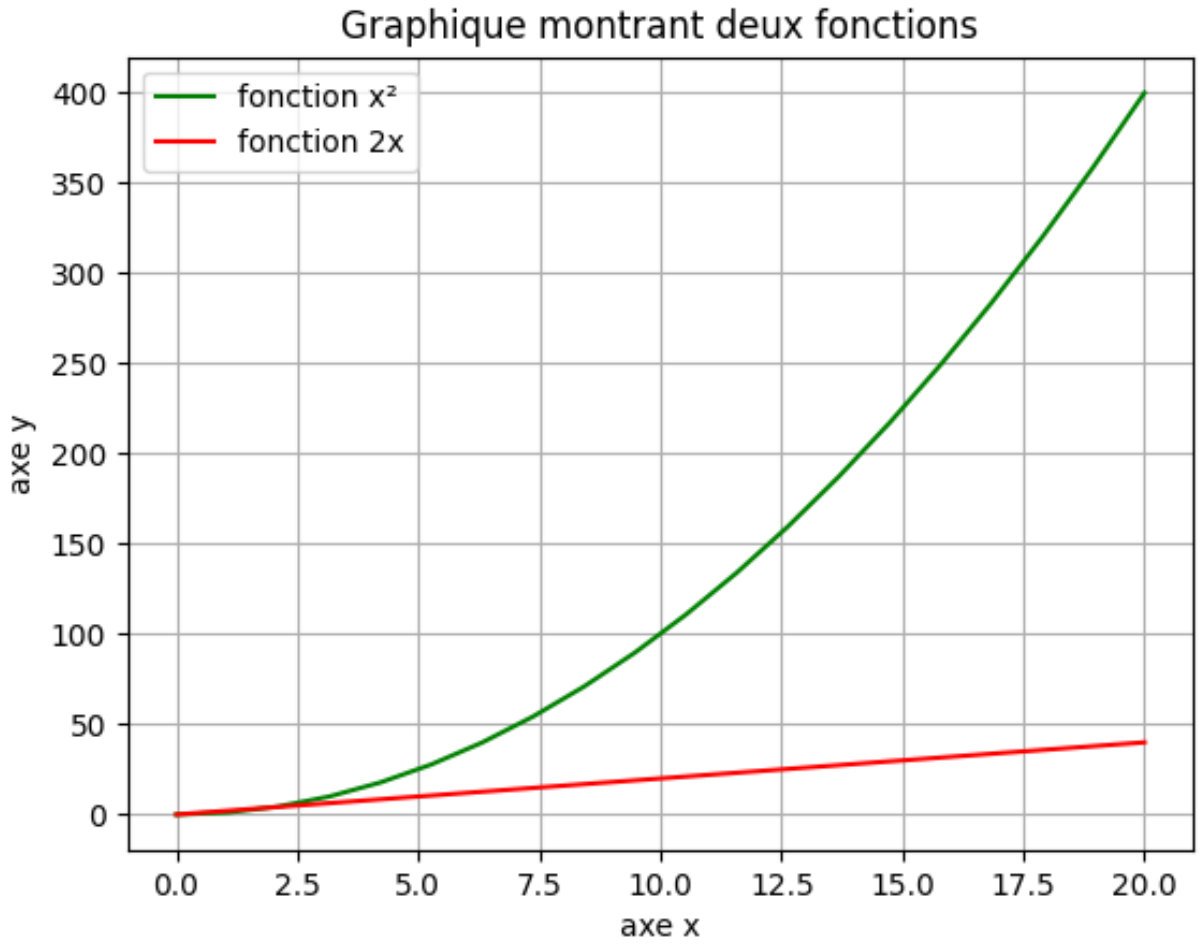

FIGURE 3.3. – Un graphique avec une légende

3.4. Les diagrammes circulaires

Matplotlib trace d'autres types de graphiques comme des camemberts grâce à la fonction `plt.pie`.

```
nb_utilisateurs = np.array([2.9, 2.3, 2, 1.4]) # En milliard
    d'utilisateurs
reseaux_sociaux = [
    "Facebook",
    "YouTube",
    "WhatsApp",
    "Instagram",
]

plt.title("Nombre d'utilisateurs des différents réseaux sociaux")
plt.pie(nb_utilisateurs, labels=reseaux_sociaux)
plt.show()
```

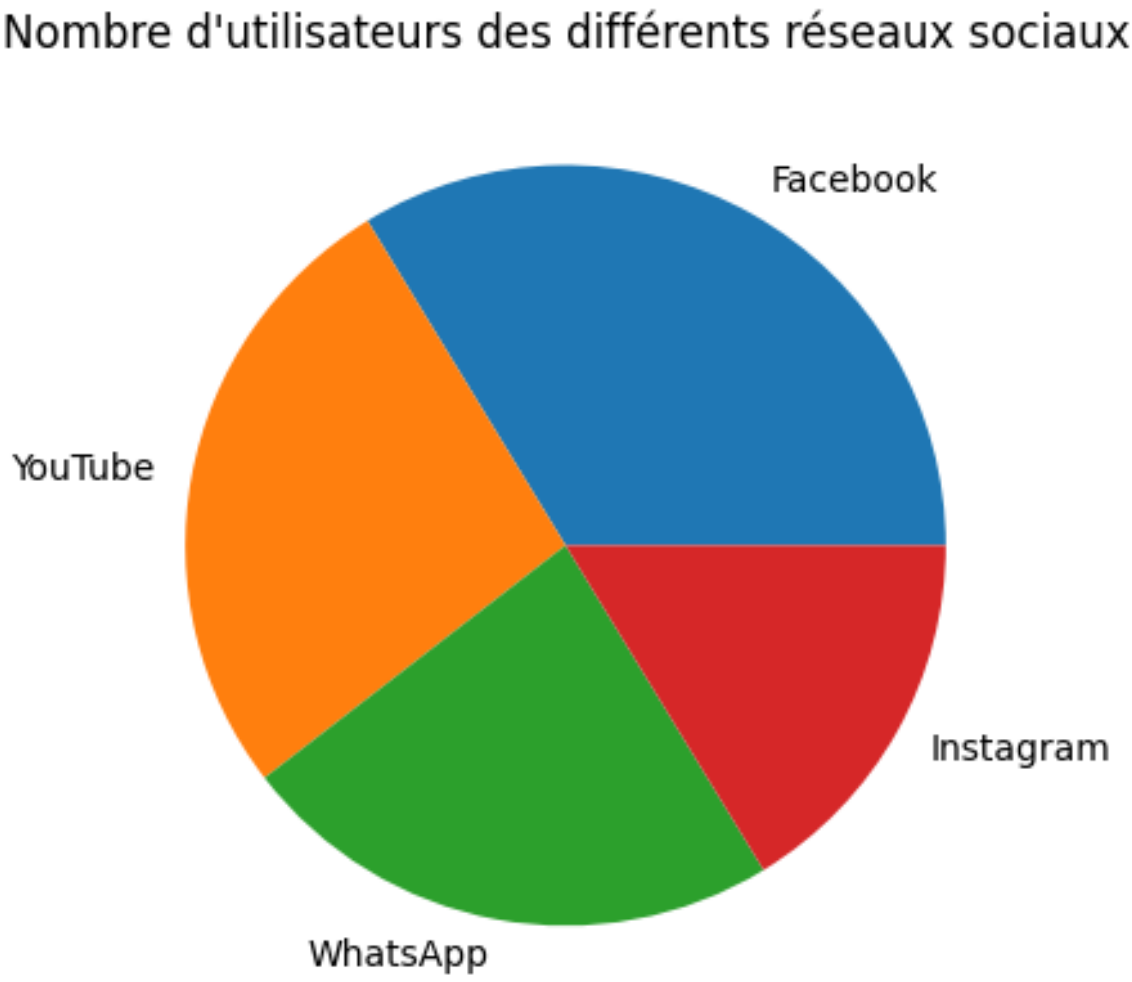

FIGURE 3.4. – Diagramme circulaire montrant le nombre d'utilisateurs des réseaux sociaux

3.5. Les diagrammes en bâtons

L'affichage des diagrammes bâtons se fait avec la fonction `plt.bar`, par exemple, si on reprend les données utilisées pour le diagramme circulaire:

```
nb_utilisateurs = np.array([2.9, 2.3, 2, 1.4])
reseaux_sociaux = [
    "Facebook",
    "YouTube",
    "WhatsApp",
    "Instagram",
]

plt.bar(reseaux_sociaux, nb_utilisateurs)
plt.show()
```

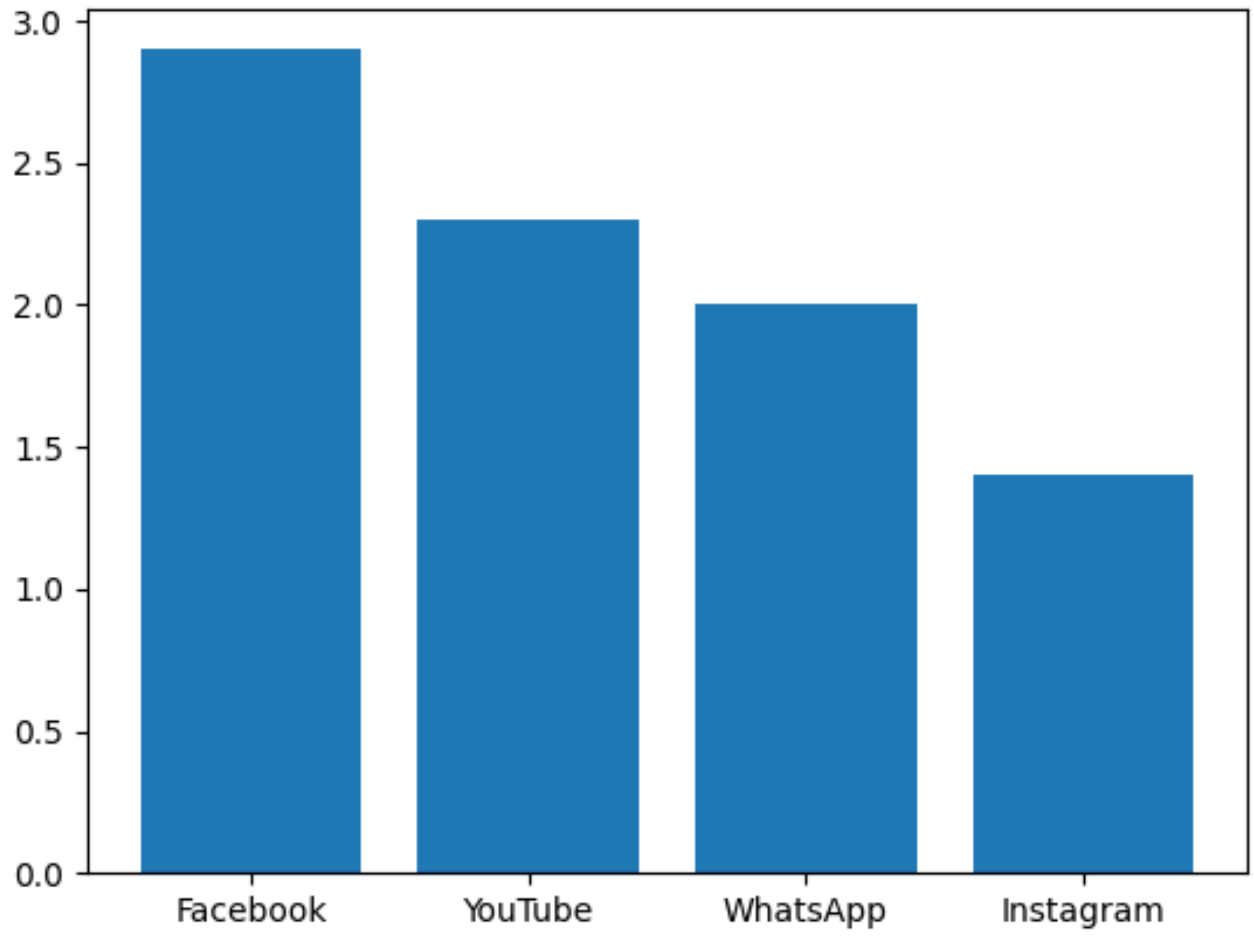

FIGURE 3.5. – Diagramme en bâton

3.6. Les histogrammes

Afficher un histogramme avec matplotlib est simple, on utilise la fonction `hist(x)`:

```
1 x = np.random.binomial(n=20, p=0.2, size=1000) # Génère 1000
      nombres selon une loi binomiale
2 plt.hist(x)
3 plt.show()
```

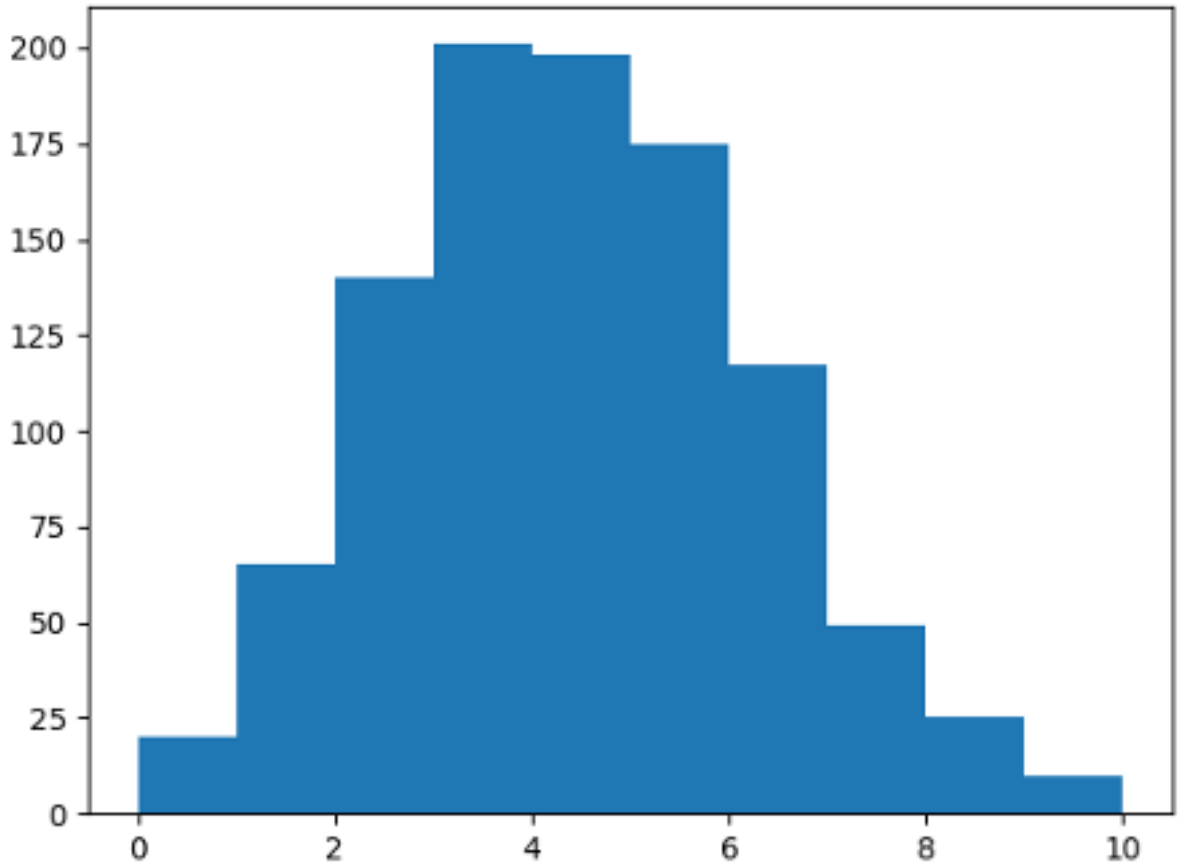

FIGURE 3.6. – Histogramme d'une loi binomiale

3.7. Aller plus loin

Pour continuer votre apprentissage de Matplotlib, je ne peux que vous conseiller de regarder la documentation , qui vous permettra d'en apprendre plus sur les fonctions présentées dans cette partie.

De plus, celle-ci propose de nombreuses autres fonctionnalités avancées comme :

— Création de diagramme en 3 dimensions
— Les graphiques interactif
— Les animations

Conclusion

Voilà, c'est la fin de ce mini-tutoriel, vous devriez donc maîtriser les bases de numpy et de matplotlib! 🍊

Vous êtes maintenant capable de:

— Créer des tableaux à une et deux dimensions.
— Faire les manipulations de bases sur les tableaux.
— Afficher des courbes avec la fonction `plot`.
— Visualiser des données avec des diagrammes, des histogrammes ou des camemberts.

Il vous reste néanmoins de nombreuses choses à apprendre. Pour cela, il peut être intéressant de regarder la documentation de ces deux modules:

— https://numpy.org/doc/stable/index.html
— https://matplotlib.org/

Il existe également plein d'autres bibliothèques Python qui utilisent numpy et matplotlib. On pourrait par exemple citer SciPy qui propose de nombreuses fonctions pour l'algèbre linéaire, le traitement du signal, l'optimisation… ou encore scikit-learn qui permet de faire de l'apprentissage automatique.

Je tiens aussi à remercier @etherpin, @Gabbro et @Sioul pour leurs relectures.

Contenu masqué

Contenu masqué n°1 : Couleur

Chaîne de caractère	Couleur
b	bleu
r	rouge
g	vert
y	jaune
k	noir
w	blanc

Retourner au texte.

Contenu masqué n°2 : Point

Chaîne de caractère	Type de point
.	point
o	cercle
v	triangle vers le bas
x	croix
+	plus

Retourner au texte.

Contenu masqué n°3 : Ligne

Chaîne de caractère	Type de ligne
-	trait continue
:	pointillé
–	tireté

Retourner au texte.