Chapitre :
Le module matplotlib

Programmation en Python

MPSI 2-3

mazza8azzouz@gmail.com

6 janvier 2025

Plan

Introduction

Comme tous les modules, il faut le charger et plus précisément c'est un sous-module qui va nous intéresser : `pyplot`. Pour cela, nous n'allons pas charger toutes les fonctions comme d'habitude mais l'importer sous un nom plus court à utiliser. On utilisera donc `import matplotlib.pyplot as plt`. Ce qui signifie que pour utiliser une fonction de ce module comme `show()` par exemple, on devra écrire `plt.show()` (puisqu'on a importé le module sous le nom `plt`).
De plus, le module `matplotlib` est très lié à un autre module qui sert à faire du calcul numérique qui s'appelle `numpy` et qu'on import souvent sous le nom np.

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

Les fonctions de base

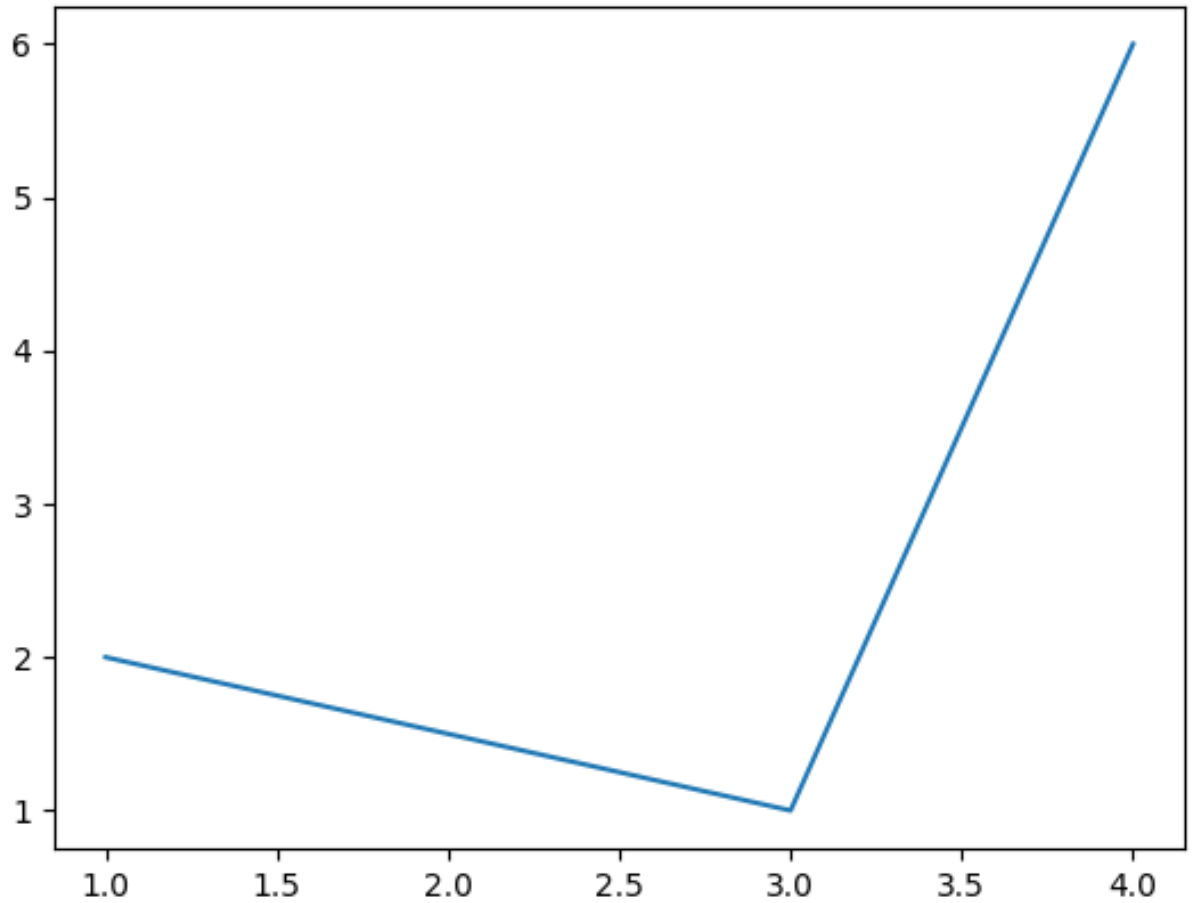
- `plt.show()` : Pour afficher le résultat. Toute les fonctions qui suivent servent à préparer le graphique mais si on ne demande pas de l'afficher, rien ne se passera (exactement comme la fonction `print` : aucun calcul ne s'affiche si on ne demande pas de l'afficher avec `print`).
- `plt.plot(liste_x,liste_y)` : Où `liste_x` est une liste de nombres $[x\_1, x\_2, ..., x\_n]$ et `liste_y` une liste de nombres $[y\_1, y\_2, ..., y\_n]$ avec le même nombre d'éléments. Alors `plt.plot(liste_x,liste_y)` placera les points de coordonnées $(x\_1, y\_1), (x\_2, y\_2), ..., (x\_n, y\_n)$ et les reliera de proche en proche par un segment.

Exemple :

Voici un exemple où on relie les points $(1;2), (3;1) et (4;6)$:

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,3,4],[2,1,6])
plt.show()
```

Trace



créer une liste de N nombres

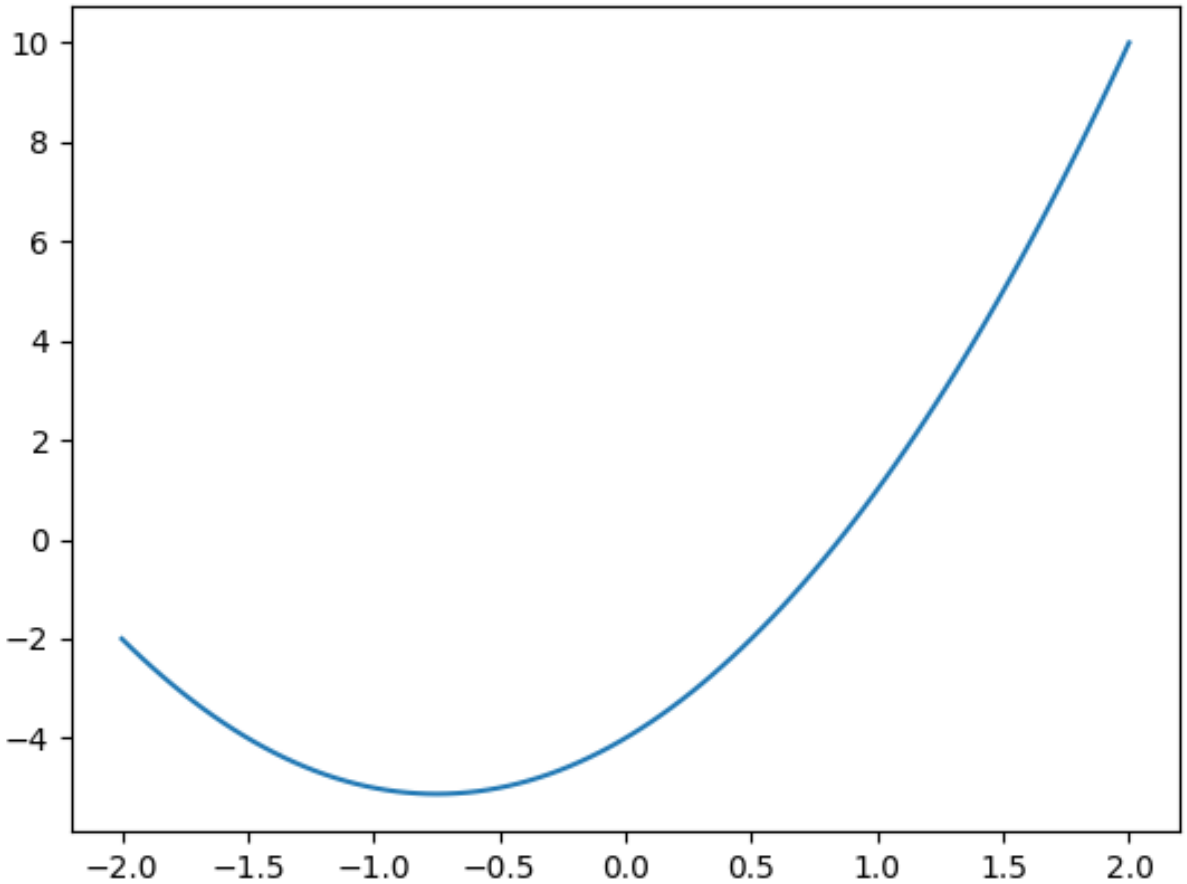
`np.linspace(debut, fin, N)` : C'est ici que le module `numpy` intervient. Pour tracer correctement une fonction, il va nous falloir beaucoup de points qu'il est hors de question de rentrer à la main comme dans l'exemple précédent. La fonction `np.linspace(debut, fin, N)` permet de créer une liste de N nombres qui commencent à la valeur `debut` et s'arrête à la valeur `fin` et uniformément répartis.
De plus, si on fait une opération sur cette liste comme par exemple multiplier par 2, alors cette opération sera automatiquement appliquée à chaque terme de la liste (ce qui n'est pas vrai si on utilise une liste classique).

Exemple :

Par exemple, traçons la fonction définie par $y = 2x^2 + 3x - 4$ entre -2 et 2 en utilisant 100 points :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2, 2, 100)
y = 2*x**2+3*x-4
plt.plot(x,y)
plt.show()
```

Trace



A vous

Amusez vous à modifier la fonction, **les bornes** et **le nombre de points** (par exemple 10) utilisés pour bien comprendre le fonctionnement.

Repère et quadrillage

- `plt.axis(x_min, x_max, y_min, y_max)` : Cette fonction permet de modifier les axes du repère qui sera affiché. Si on ne l'utilise pas, le choix des axes sera fait automatiquement mais des fois ce choix n'est pas pertinent et il faudra donc le modifier avec cette fonction. Les deux premières valeurs qu'on donne sont les valeurs minimale et maximale pour l'axe des abscisses et les deux suivantes sont celles pour l'axe des ordonnées.
- `plt.grid()` : Affiche un quadrillage en plus sur

Supposons qu'on veuille tracer des fonctions faisant intervenir des fonctions comme par exemple des cosinus, sinus, exponentielle, logarithme... Dans ce cas on ne peut pas faire exactement comme dans l'exemple précédent.

Une première façon de faire est de créer "à la main" la liste des y correspondants aux x c'est à dire créer une liste composée des f(x) pour x dans la liste des abscisses.
Par exemple si on veut tracer la fonction $y = cos(x) + 3sin(2x)$ entre -4 et 4, on pourra faire ainsi :

```
import matplotlib.pyplot as plt
import numpy as np
from math import *
abscisses = np.linspace(-4,4,100)
ordonnées = [cos(x)+3*sin(2*x) for x in abscisses]
plt.plot(abscisses,ordonnées)
```

Une seconde méthode

Une seconde méthode consiste à utiliser les fonctions classiques modifiées contenues dans `numpy` (en écrivant np.cos pour le cosinus par exemple). Pourquoi modifiées ? Car elles s'appliquent directement à toute la liste. Si on garde le même exemple de fonction $y = cos(x) + 3sin(2x)$ entre -4 et 4, cela donnera :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-4,4,100)
y = np.cos(x)+3*np.sin(2*x)
plt.plot(x,y)
plt.show()
```

Plusieurs fonctions

Pour tracer plusieurs fonctions dans un même repère, il suffit de tracer plusieurs fois une fonction... Elles seront automatiquement dans le même repère. Par exemple, si je veux vérifier graphiquement que l'équation de droite $y = -2x + 3$ que j'ai obtenu correspond bien à l'équation de la tangente en 1 de la fonction $y = x - 4x + 4$, il suffira d'écrire :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-1,3,100)
y = -2*x+3
plt.plot(x,y)
y = x**2 - 4*x + 4
plt.plot(x,y)
plt.show()
```

Tracé de plusieurs fonctions

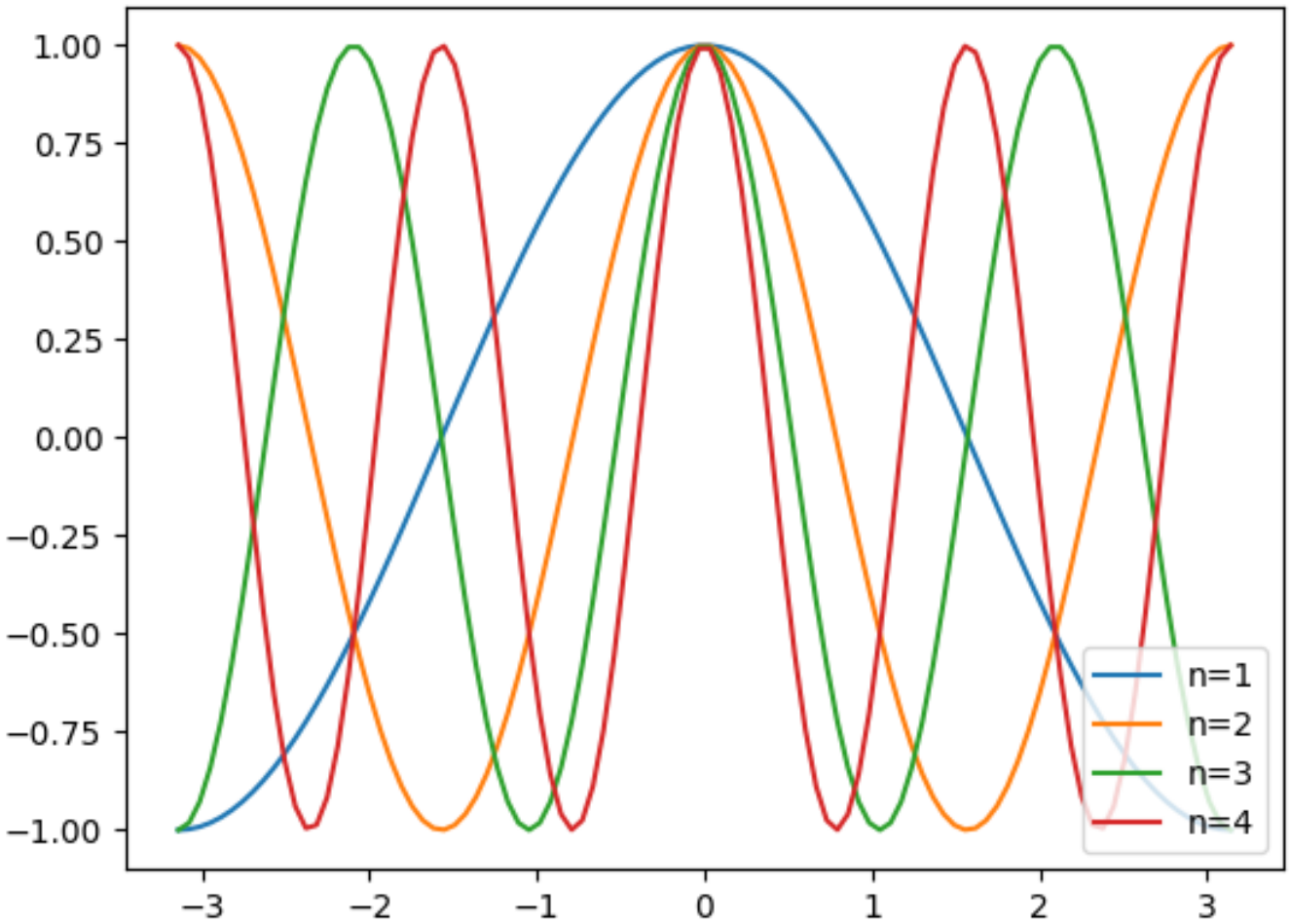
Voici un autre exemple où on trace une famille de fonction $y = cos(nx)$ avec n allant de 1 à 4 :

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

x = np.linspace(-np.pi,np.pi,100)
for n in range(1,5):
    y = np.cos(n*x)
    plt.plot(x,y, label="n="+str(n))

plt.legend(loc="lower right")
plt.show()
```

Tracé de plusieurs fonctions

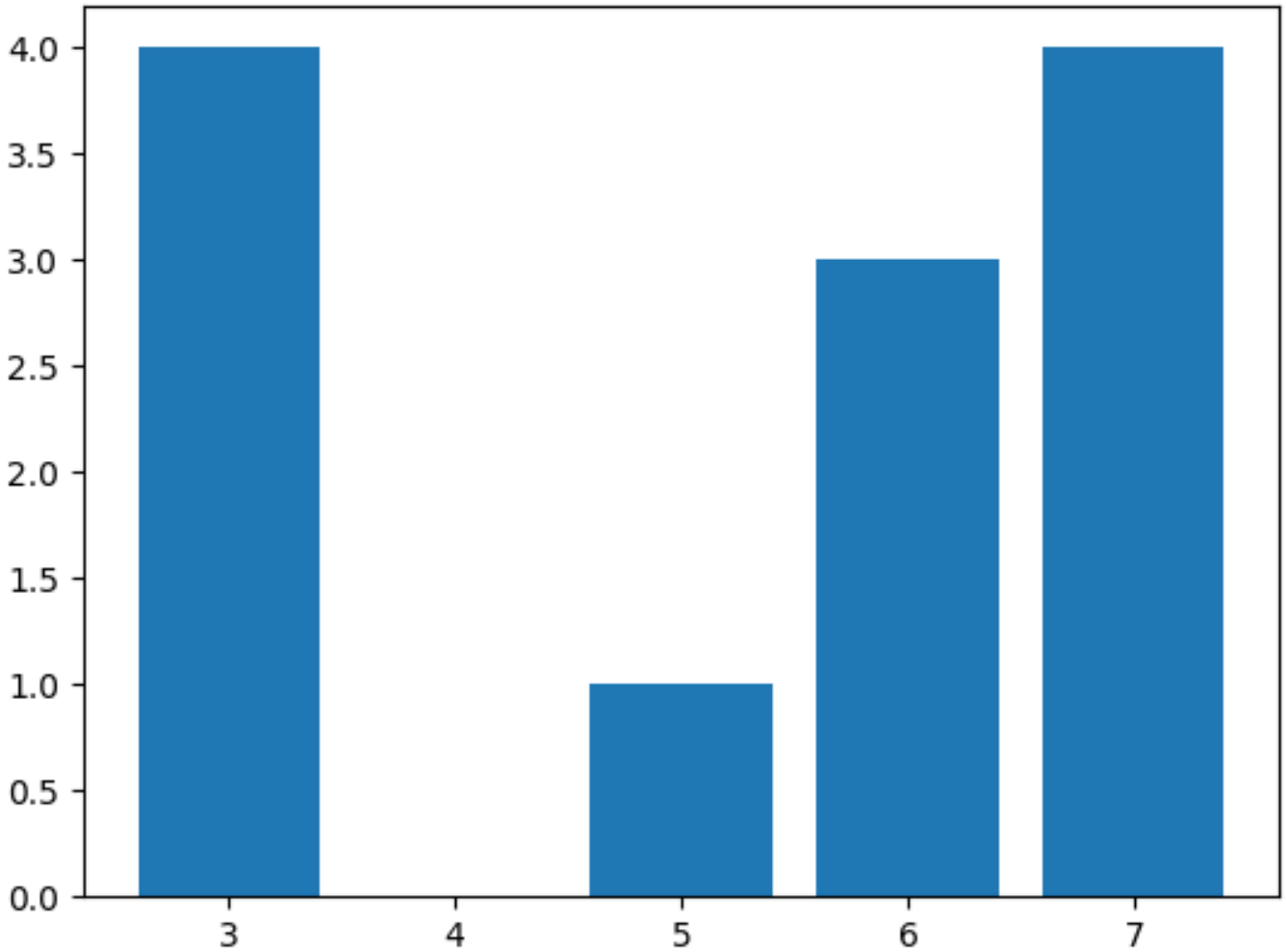


les diagrammes en bâtons

On peut tracer assez facilement à partir de séries statistiques des diagrammes en bâtons et des histogrammes. Voici un exemple de chaque :
Pour **les diagrammes en bâtons**, on utilise la fonction `bar(valeurs,effectifs)`.

```
import matplotlib.pyplot as plt
import numpy as np
x = [3, 5, 6, 7]
y = [4, 1, 3, 4]
plt.bar(x,y)
plt.show()
```

les diagrammes en bâtons

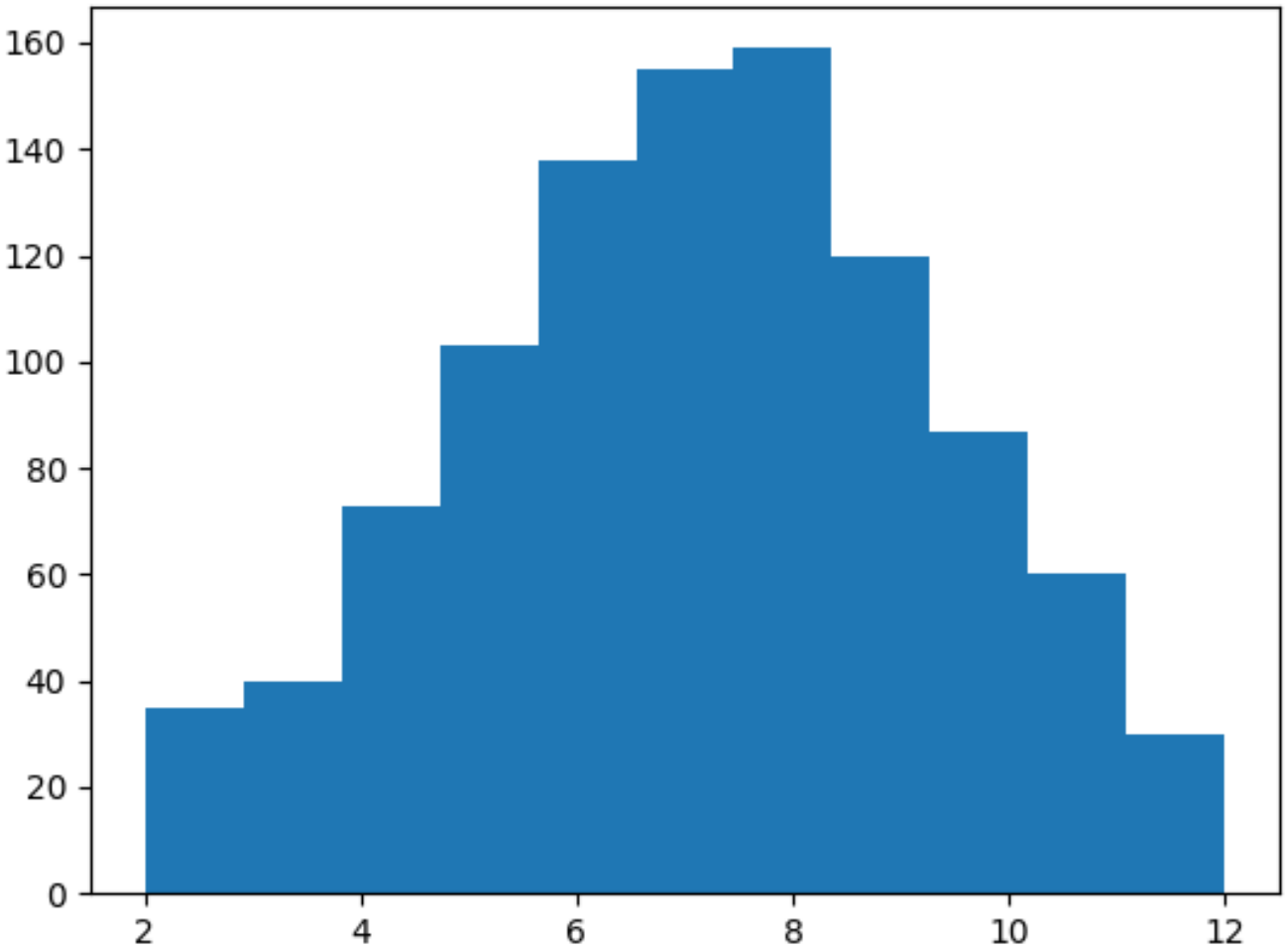


les histogrammes

Pour **les histogrammes**, on utilise la fonction `hist(liste,[n])` qui trace l'histogramme de la liste répartie en n groupes(si n est précisé). Par exemple si on lance deux dés au hasard et qu'on fait leur somme et qu'on répète 1000 fois ceci. On peut tracer l'histogramme des résultats en écrivant :

```
import matplotlib.pyplot as plt
import numpy as np
from random import *
liste =[randint(1,6)+randint(1,6) for _ in range(1000)]
plt.hist(liste,11)
plt.show()
```

les histogrammes

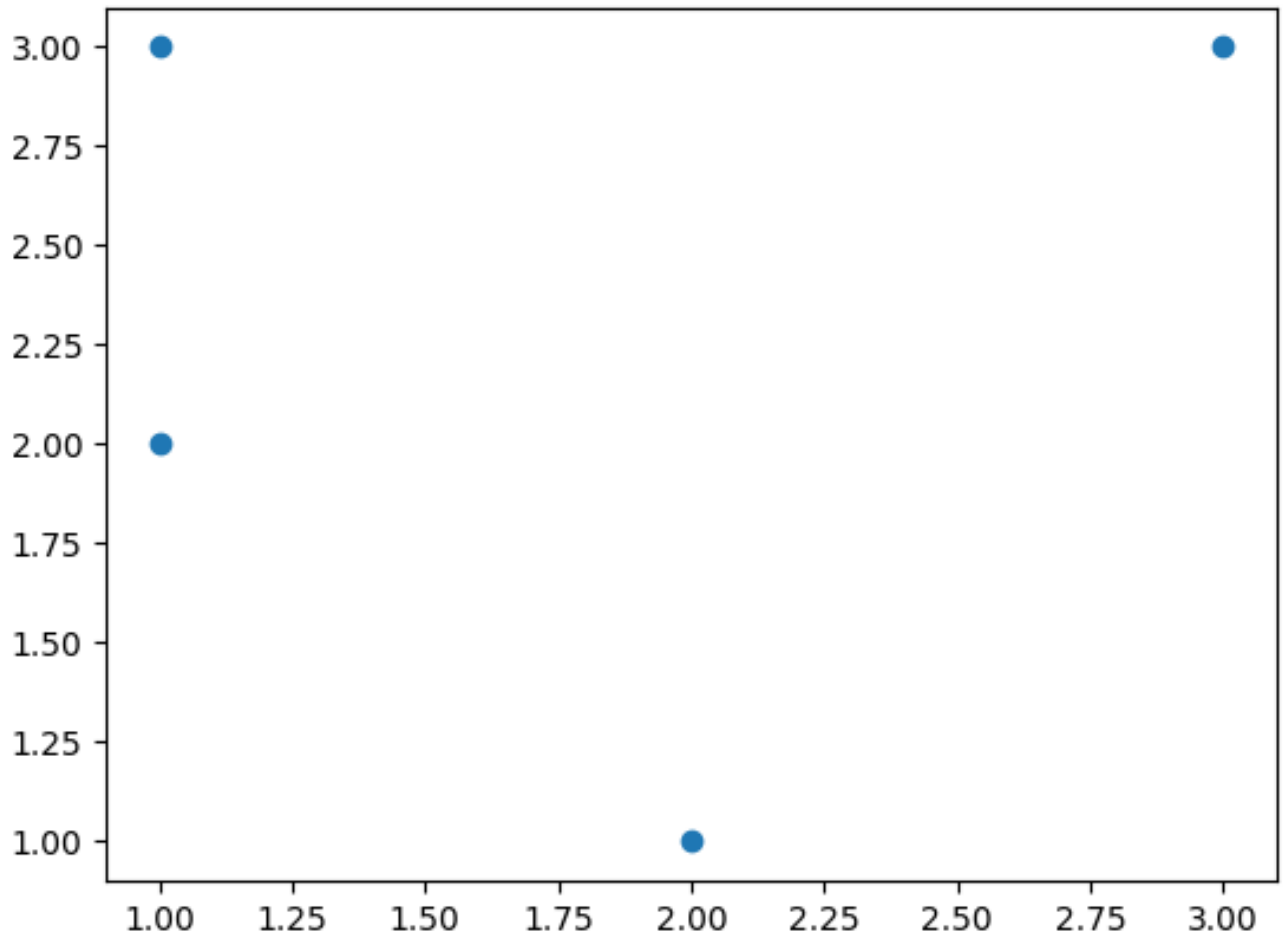


Nuages de points

Pour tracer des points, il suffit d'utiliser la fonction `plt.scatter(abscisses, ordonnées)` où abscisses est la liste des abscisses des points qu'on veut tracer et ordonnées la liste des ordonnées. Par exemple si on veut placer les points (1,2), (3,3), (2,1) et (1,3) :

```
import matplotlib.pyplot as plt
import numpy as np
x = [ 1, 3, 2, 1]
y = [ 2, 3, 1, 3]
plt.scatter(x,y)
plt.show()
```

Nuage de points



Nuage de points

Voici un nuage de 5000 points pris au hasard. On a coloré en rouge ceux qui vérifient la condition $y < x^2$.

```
from random import *
#Fonction qui crée la liste de couleurs en fonction de la
def donner_couleur(x,y):
    couleurs=[]
    for i in range(len(x)):
        if y[i]<x[i]**2 : couleurs.append("r")
        else : couleurs.append("b")
    return couleurs
x = [random() for _ in range(5000)]
y = [random() for _ in range(5000)]
plt.scatter(x,y,s=1,c=donner_couleur(x,y))
plt.show()
```

Nuage de points