Informatique et Calcul Scientifique

Cours 7 : NumPy et Matplotlib

09.04.2025

La fois passée, on a vu..

Le gestionnaire d'exceptions :

- qui permet de tester une portion de code sans interrompre son exécution.
- syntaxe `try : ... except : ... else : ... finally : ...`.
- On peut générer des exceptions personnalisées avec le mot clé `raise`.

But de la leçon

Aujourd'hui on...

Aujourd'hui, on verra deux modules :

Le module `NumPy` qui :

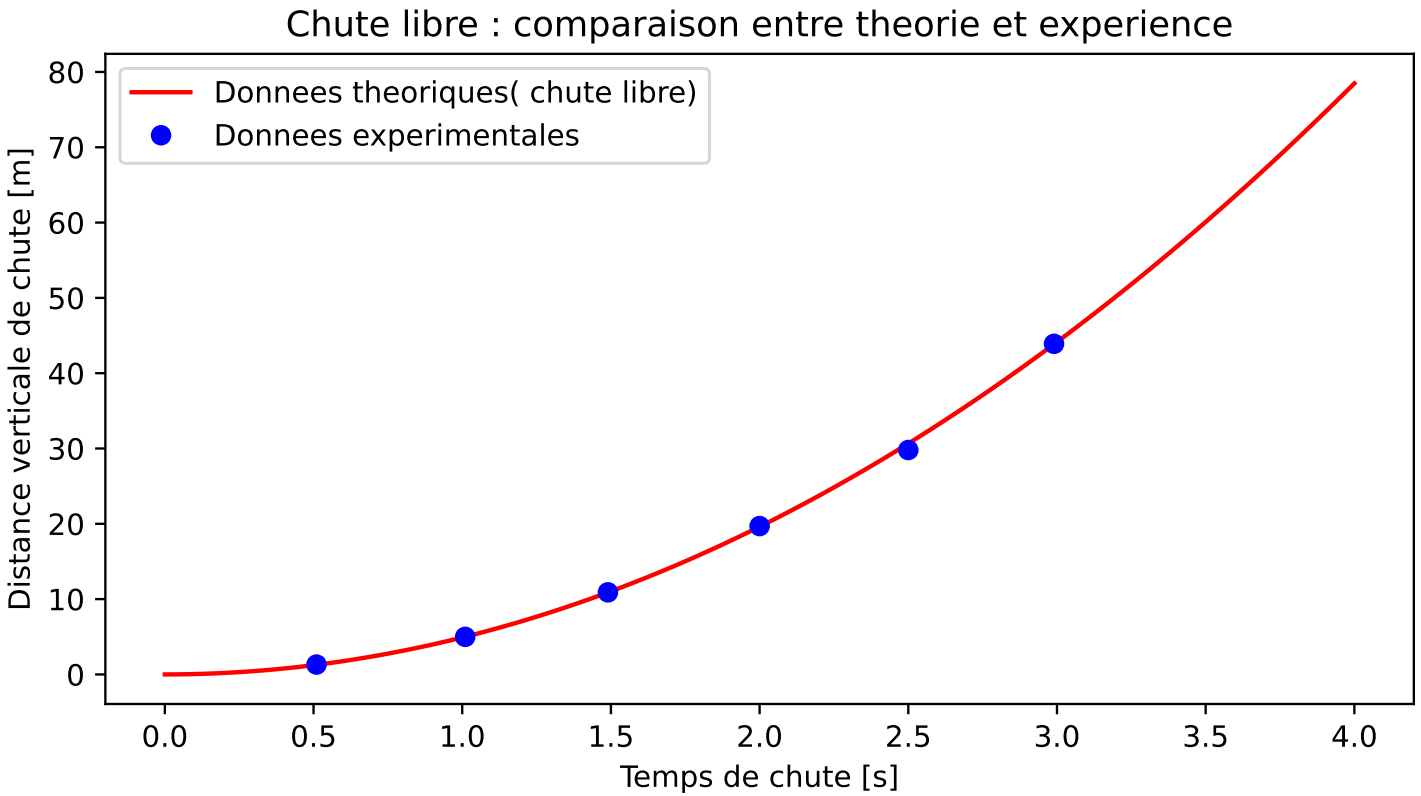
- permet d'effectuer des opérations mathématiques et statistiques sur des vecteurs
- est extrêmement répandue en science des données (data science)

Le module `Matplotlib` qui :

- permet la représentation graphique de données ou de fonctions

But de la leçon

Réaliser ce graphe à partir de données expérimentales :



Question : Comment faire pour effectuer la même opération sur tous les éléments d'une liste ?

- Il faut modifier chaque élément **individuellement** en passant par une compréhension de liste.

```
L = [1, 2, 3, 4]

L1 = [i**2 for i in L]
L2 = [k+5 for k in L]
L3 = [L1[i]+L2[i] for i in range(len(L))]

print(L)
print(L1)
print(L2)
print(L3)
```

```
Output :
[1, 2, 3, 4]
[1, 4, 9, 16]
[6, 7, 8, 9]
[7, 11, 17, 25]
```

Cette manière de faire n'est pas très intuitive et alourdit le code.

Introduction à NumPy

Il est possible de modifier une struture de données de manière **vectorisée** en utilisant la bibliothèque **NumPy**.

- Celle-ci comprend des objets, les `ndarray` que l'on peut utiliser pour effectuer des opérations mathématiques sur des vecteurs et des matrices de manière plus rapide et intuitive.

Comme tout module, cette bibliothèque se charge en rajoutant cette ligne en début de document : `import numpy as np`.

- Une méthode de ce module est alors appelée par `np.nom_fonction`.

```
import numpy as np
x = 9
print(np.sqrt(x))
```

Output :

```
3.0
```

Introduction à NumPy

L'élément central de NumPy est la structure de données `ndarray` (*n-dimensional array*). Il s'agit de tableaux :

- multidimensionnels,
- de taille fixée à leur création,
- homogènes,
- mutables,
- itérables.

```
import numpy as np
L = [1,2,3]
x = np.array([1,2,3])
print(L, type(L))
print(x, type(x))
```

```
Output :
[1, 2, 3] <class 'list'>
[1 2 3] <class 'numpy.ndarray'>
```

Ces objets sont tels qu'ils autorisent des opérations *vectorisées*, pour lesquelles le `ndarray` est traité comme une unité.

Création d'un ndarray

La méthode `array()` permet de convertir un container (liste, tuple) en `ndarray`.

```
import numpy as np
L = [1,2,3]
t = (4,5,6)
La = np.array(L)
ta = np.array(t)
print(La, type(La))
print(ta, type(ta))
```

Output :

```
[1 2 3] <class 'numpy.ndarray'>
[4 5 6] <class 'numpy.ndarray'>
```

Un `ndarray` ne contient que des données homogènes. Tous les éléments du container d'entrée sont donc convertis en le type le plus général de ce tableau, si ceci est possible

```
import numpy as np
L = [1,2,'4']
print(np.array(L))
```

Output :

```
['1' '2' '4']
```

Création d'un ndarray unidimensionnel

Il existe deux méthodes permettant de créer un `ndarray` de nombres séparés d'un intervalle constant.

- La méthode `arange(start, stop, step)` crée un tableau de nombres allant de `start` à `stop` (**non compris**) séparés d'une distance `step`.

```
import numpy as np
print(np.arange(2,3,0.2))
```

Output :

```
[2.  2.2 2.4 2.6 2.8]
```

- La méthode `linspace(start, stop, Npoints)` crée un tableau de `Npoints` nombres allant de `start` à `stop` compris.

```
import numpy as np
print(np.linspace(2,3,6))
```

Output :

```
[2.  2.2 2.4 2.6 2.8 3.]
```

Création d'un ndarray unidimensionnel

Dans certaines conditions, il est préférable d'initialiser tous les éléments d'un vecteur à une certaine valeur.

- La méthode `zeros(dim)` permet de créer un vecteur à `dim` éléments valant tous 0
- La méthode `ones(dim)` permet de créer un vecteur à `dim` éléments valant tous 1
- La méthode `full(dim, val)` permet de créer un vecteur à `dim` éléments valant tous `val`.
- La méthode `empty(dim)` permet de créer un vecteur à `dim` éléments initialisés à une valeur aléatoire.

Si on veut créer un vecteur de la même taille et du même type que le vecteur `vec`, on peut rajouter le suffixe `_like(vec)` aux fonctions présentées ci-dessus.

Création d'un ndarray unidimensionnel : exemples

```python
import numpy as np
## zeros
a = np.zeros(3)
## ones
b = np.ones(5)
## full
c = np.full(4,0.3)
## empty
d = np.empty(2)
print(a,b,c,d,sep="\n")
## _like
print("***")
r = np.logspace(1,3,4)
rl = np.ones_like(r)
rl2 = np.zeros_like(r)
print(r,rl,rl2,sep='\n')
```

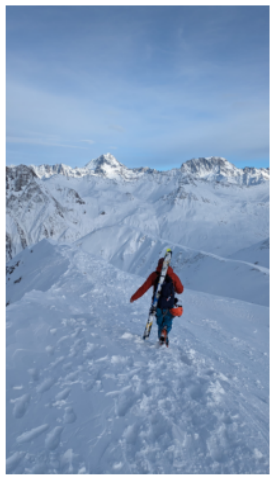
```
Output :
[0. 0. 0.]
[1. 1. 1. 1. 1.]
[0.3 0.3 0.3 0.3]
[-2.00000000e+000 6.95330421e-310]
***
[10. 46.41588834 215.443469 1000.]
[1. 1. 1. 1.]
[0. 0. 0. 0.]
```

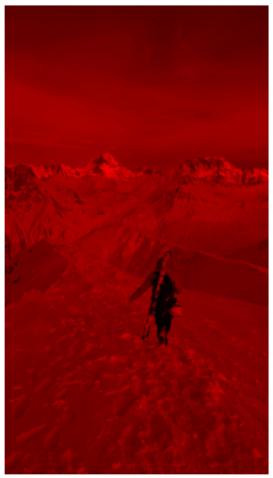
ndarray multidimensionnels

Un `ndarray` à une dimension est omniprésent dans le calcul scientifique pour représenter des données.

Un `ndarray` à deux dimensions est également largement répandu pour représenter des matrices, ou une image en niveaux de gris.

Un `ndarray` de dimension trois peut être utilisé pour représenter une image en couleurs (rgb).





Création d'un ndarray multidimensionnel

On peut créer un `ndarray` multidimensionnel à partir d'un container de container en utilisant la méthode `array`.

```python
import numpy as np
L = [[1,2,3],[4,5,6],[7,8,9]]
M = np.array(L)
print(M)
```

Output :

```
[[1 2 3]
[4 5 6]
[7 8 9]]

2 (3, 3)
```

Les méthodes `ndim` et `shape` retournent le nombre de dimensions et la forme du `ndarray` étudié.

```python
print(M.ndim)
print(M.shape)
```

Output :

```
2
(3, 3)
```

Par exemple, l'image étudiée à la slide précédente a la forme `image.shape = (4032, 2268, 3)`.
Saurez-vous dire à quoi correspondent ces nombres ?

Création d'un ndarray multidimensionnel

Il est également possible de créer un `ndarray` multidimensionnel `x` à partir d'un `ndarray` de dimension supérieure à l'aide de la méthode `reshape(x, shape)`, où `shape` est la forme de l'objet final.

```python
import numpy as np
L = np.arange(12)
print(L, end='\n***\n')
M = np.reshape(L,(2,6))
N = np.reshape(L,(3,4))
P = np.reshape(L,(2,3,2))
print(M,N,P,sep='\n***\n')
```

```
Output :
[ 0 1 2 3 4 5 6 7 8 9 10 11]
***
[[ 0 1 2 3 4 5]
[ 6 7 8 9 10 11]]
***
[[ 0 1 2 3]
[ 4 5 6 7]
[ 8 9 10 11]]
***
[[[ 0 1] [ 2 3] [ 4 5]]
[[ 6 7] [ 8 9] [10 11]]]
```

Il faut bien entendu que les dimensions soient compatibles ! On ne peut pas créer une matrice de taille (3x4) à partir d'une liste de longueur 11 par exemple.

Création d'un ndarray multidimensionnel

On peut généraliser les méthodes `np.zeros(shape)`, `np.ones(shape)`, `np.full(shape, val)` et `np.empty(shape)` vues précédemment aux matrices. Dans ce cas, l'argument `shape` n'est plus un scalaire, mais un tuple contenant la longueur de chaque dimension.

```
import numpy as np
print(np.zeros((2,6)))
print(np.full((3,4),1.1))
```

Output :

```
[[0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0.]]

[[1.1 1.1 1.1 1.1]
[1.1 1.1 1.1 1.1]
[1.1 1.1 1.1 1.1]]
```

Il est également possible de créer des matrices diagonales à l'aide de la méthode `eye`

```
print(np.eye(3))
```

Output :

```
[[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]]
```

Accès à un élément d'un ndarray

Pour accéder à un élément d'un `ndarray`, on peut soit :

- utiliser la même syntaxe que pour les listes,

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

Output :

```
[[1 2 3]
[4 5 6]
[7 8 9]]
6
```

- soit utiliser la syntaxe "matricielle" (ligne, colonne).

```
print(a[1,2])
```

Output :

```
6
```

On peut également utiliser cette syntaxe pour effectuer des opérations de slicing, si on veut accéder à une ligne ou à une colonne particulière.

```
print(a[0,:])
print(a[:,0])
print(a[::2,1])
```

Output :

```
[1 2 3]
[1 4 7]
[2 8]
```

Opérations sur les ndarray

La force de `NumPy` est de permettre d'effectuer des opérations vectorisées sur plusieurs `ndarray`, du moment qu'ils ont la même taille. Celles-ci sont alors effectuées composante par composante.

Par exemple, soit `a = [a1 a2]` et `b = [b1 b2]` deux `ndarray`. Alors :

- `a+b` = `[a1+b1 a2+b2]`
- `a-b` = `[a1-b1 a2-b2]`
- `a*b` = `[a1*b1 a2*b2]`
- `a/b` = `[a1/b1 a2/b2]`

Note : Attention à ne pas confondre `a*b` avec le produit scalaire de deux vecteurs, ou avec le produit matriciel.

Opérations sur les ndarray unidimensionnels

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
print(a+b, a-b, sep='\n')
print('***')
print(a*b, a/b, sep='\n')
```

```
Output :
[11 22 33 44]
[-9 -18 -27 -36]
***
[10 40 90 160]
[0.1 0.1 0.1 0.1]
```

De la même manière, toute multiplication d'un vecteur par un scalaire est autorisée par `NumPy`

```
import numpy as np
a = np.array([1,2,3,4])
b = a+1
c = 4*a
d = a/4
e = a**2
print(a,b,c,d,e,sep='\n')
```

```
Output :
[1 2 3 4]
[2 3 4 5]
[4 8 12 16]
[0.25 0.5 0.75 1.]
[1 4 9 16]
```

Opérations sur les ndarray multidimensionnels

Ces opérations sont généralisables aux matrices.

```python
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
b = a**2
print(b)
print(a+b)
print(b/a)
print(a*b)
```

```
Output :

[[ 1  4  9]
 [16 25 36]
 [49 64 81]]

[[ 2  6 12]
 [20 30 42]
 [56 72 90]]

[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]

[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
```

Les opérations sont toujours effectuées élément par élément.

Opérations matricielles

Pour une multiplication standard de matrices ou de vecteurs, on utilise la méthode `dot()`.

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = 2*np.eye(3)
print(np.dot(A,B))
print(np.dot(A[0,:],B[1,:]))
```

Output :

```
[[ 2. 4. 6.]
[ 8. 10. 12.]
[14. 16. 18.]]
4.0
```

Bon à savoir : beaucoup d'opérations matricielles plus avancées sont définies dans le sous-module `linalg` de `NumPy`.

Les méthodes `det`, `inv`, `transpose`, `eig` sont particulièrement utiles pour l'étude de systèmes matriciels.

```
print(np.linalg.det(A))
```

Output :

```
30
```

Fonctions mathématiques en NumPy

En plus de ces méthodes sur les matrices, `NumPy` fournit une série de fonctions mathématiques[1].

De la même manière, ces méthodes s'appliquent composante par composante peu importe la dimension du `ndarray`.

```python
import numpy as np
a = np.array([[1,4],[9,16]])
print(np.cos(a)) # en radian
print(np.sqrt(a))
print(np.log10(a))
print(np.exp(a))
```

Output :

```
[[ 0.54030231 -0.65364362]
[-0.91113026 -0.95765948]]

[[1. 2.]
[3. 4.]]

[[0. 0.60205999]
[0.95424251 1.20411998]]

[[2.7182818e+00 5.4598150e+01]
[8.1030839e+03 8.8861105e+06]]
```

1. la liste complète est disponible dans la doc.

Opérations logiques

Il est possible d'évaluer une condition sur tout le `ndarray`.

On obtient alors un `ndarray` de booléens contenant le résultat de la comparaison, élément par élément.

```
import numpy as np
v = np.linspace(-2,3,6)
y = v > 0
print(v)
print(y)
print(v!=0)
print('***')
v2 = np.reshape(v,(2,3))
print(v2)
print(v2>0)
```

```
Output :
[-2. -1. 0. 1. 2. 3.]
[False False False True True True]
[True True False True True True]
***
[[-2. -1. 0.]
[ 1. 2. 3.]]
[[False False False]
[ True True True]]
```

Masque booléen

Grâce à cette syntaxe, il est possible de ne sélectionner que certains éléments d'un `ndarray`.

```
import numpy as np
v = np.linspace(-2,3,6)
v2 = np.reshape(v,(2,3))
print(v[v>0])
print(v2[v2<0])
t = np.array([1, 2, 3, 4, 5, 6])
print(v[t>2])
```

```
Output :
[1. 2. 3.]
[-2. -1.]
[0. 1. 2. 3.]
```

Remarquez qu'on perd alors la dimensionnalité du `ndarray` initial, ce qui n'est pas très pratique...

Masque booléen

... cependant, on peut utiliser cette syntaxe pour modifier **certains** éléments du `ndarray` pour lesquels une condition choisie est satisfaite.

```python
import numpy as np
A = np.random.rand(3,3)
A = np.round(A,2)
print(A)
A[A>0.5] = 1; A[A<0.5]=0
print(A)
```

Output :

```
[[0.56 0.45 0.87]
[0.55 0.7 0.56]
[0.36 0.29 0.78]]

[[1. 0. 1.]
[1. 1. 1.]
[0. 0. 1.]]
```

Ce type de manipulation est extrêmement puissant pour filtrer de grands ensembles de données.

ndarray multidimensionnels

L'opération de concaténation ne se fait plus au travers de l'opérateur `+`. On utilise pour cela la méthode `concatenate()` qui prend en argument un **tuple** contenant les `ndarray` à concaténer.

```
import numpy as np
x = np.array([1,2,3])
y = np.array([4,5,6])
print(np.concatenate((x,y)))
```

Output :

```
[1 2 3 4 5 6]
```

Pour les `ndarray` de dimension supérieure, on peut spécifier selon quel axe se fait la concaténation grâce à l'argument par mot-clé `axis`.

Il faut cependant que les dimensions soient compatibles !

```
a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
print(np.concatenate((a,b), axis=0))
print(np.concatenate((a,b), axis=1))
```

Output :

```
[[1 2]
[3 4]
[5 6]
[7 8]]
[[1 2 5 6]
[3 4 7 8]]
```

Gestion des fichiers

Le module `NumPy` possède deux fonctions simplifiant le chargement ainsi que l'exportation de données.

- La méthode `loadtxt()` permet de lire et d'exploiter un fichier possédant des données sous forme de texte, placées dans plusieurs colonnes séparées par un délimiteur.
- La méthode `savetxt()` facilite la sauvegarde de données dans un fichier texte.

Gestion des fichiers

Prenons l'exemple d'un fichier de données reçu par email d'un collègue physicien qui étudie une expérience de chute libre.

Fichier 'ChuteLibre.txt'

```
Donnees collectees (masse en chute libre)
Date : 16 avril 2024
Mesure Temps [s] Distance parcourue [m] Incertitude [m]
0 0 0 0.03
1 0.51 1.31 0.02
2 1.01 4.99 0.03
3 1.49 10.9 0.03
4 2. 19.7 0.03
5 2.5 29.8 0.03
6 2.99 43.9 0.02
```

Importation de données

Le code suivant permet de récupérer les trois premières colonnes (paramètre `usecols`) du fichier `ChuteLibre.txt` situé dans le même répertoire que ce code, sans tenir compte des 4 premières lignes (paramètre `skiprows`).

```
import numpy as np
Data_array = np.loadtxt('ChuteLibre.txt',\
  usecols = (0,1,2), skiprows = 4, \
  unpack = False)
print(np.shape(Data_array))
identification, xdata, ydata = \
np.loadtxt('ChuteLibre.txt', \
  usecols = (0,1,2) , skiprows = 4, \
  unpack = True)
print(xdata)
```

Output :

```
(6, 3)
[0.51 1.01 1.49
2. 2.5 2.99]
```

Ces données sont stockées automatiquement dans un `ndarray`.
Si le paramètre `unpack` vaut `True`, celles-ci sont en plus déballées dans des `ndarray` à une dimension.

Exportation de données

La forme générale de la fonction `np.savetxt` est la suivante :

```
np.savetext(nom_fichier,data,fmt='%0.18e',delimiter='\t',
  newline='\n', header="", footer="", comments="#")
```

- 1e argument : nom du fichier à sauver
- 2e argument : données devant être sauvées sous forme de `ndarray`.
- Les arguments restants sont optionnels.

Cette fonction créera un fichier `nom_fichier` dans le répertoire courant avec les données stockées dans le `ndarray` `data`.

Exportation de données

```
import numpy as np
# Importation
mes, xdata, ydata, erreur = np.loadtxt('ChuteLibre.txt',
      skiprows=4, unpack=True)
# Modifications
infos = 'Donnees de la chute libre apres avoir ete
      retravaillees'
infos += '\nDate de creation : 17 avril 2024'
infos += '\nVersion : 2.02'
infos += '\n\n\nMesure'
infos += '\ttemps_en_[s]'
infos += '\tdistance_en_[m]'
infos += '\terreur_estimee_en_[m])'
ydata += 1
erreur /= 2
# Exportation
np.savetxt ('ChuteLibre2.txt',
      np.transpose([mes, xdata, ydata, erreur]),
      header=infos , fmt='%5.2f', delimiter = '\t')
```

Exportation de données

Voici le fichier créé :

Fichier 'ChuteLibre2.txt'

```
# Donnees de la chute libre apres avoir ete retravaillees
# Date de creation : 17 avril 2024
# Version : 2.02
#
#
# Mesure temps_en_[s] distance_en_[m] erreur_estimee_en_[m])
 1.00   0.51    2.31   0.01
 2.00   1.01    5.99   0.01
 3.00   1.49   11.90   0.01
 4.00   2.00   20.70   0.01
 5.00   2.50   30.80   0.01
 6.00   2.99   44.90   0.01
```

Matplotlib

La représentation graphique des données est essentielle au scientifique pour analyser ces données, qu'elles soient issues de mesures expérimentales ou d'un modèle théorique.

Malheureusement, la version de base de Python ne possède pas d'outil dédié à la représentation graphique de données.

- On utilise pour cela la librairie externe `Matplotlib` qui permet de créer et personnaliser des figures, puis de les enregistrer en haute qualité dans différents formats.

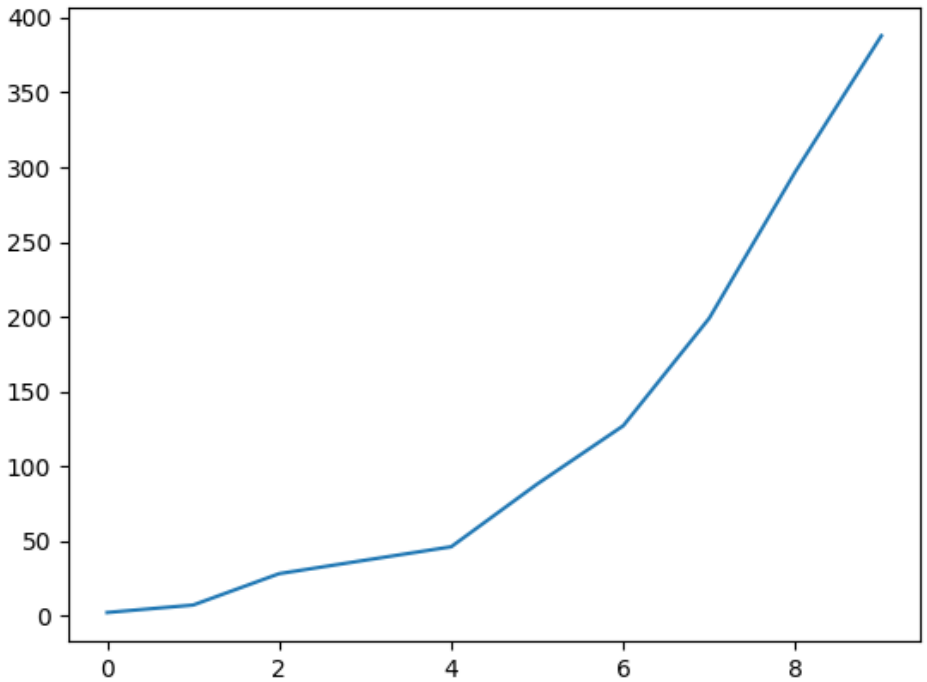
Celle-ci est souvent utilisée de pair avec `NumPy`. Il faut également la charger en début de code :

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

Premier plot

Trois lignes de code suffisent en général à obtenir une première visualisation d'un ensemble de données :

```
import matplotlib.pyplot as plt
plt.plot([2,7,28,37,46,88,127,199,297,388])
plt.show()
```



Premier plot : explications

Par défaut, la fonction `plot` relie les points (x_1, y_1), (x_2, y_2), ..., (x_N, y_N) par des segments de droite. Elle admet deux listes ou `ndarray` comme arguments :

- optionnel une liste (ou `ndarray`) `x = [x1, x2, ..., xN]` contenant les abcisses x_i
- obligatoire une liste (ou `ndarray`) `y = [y1, y2, ..., yN]` contenant les ordonnées y_i

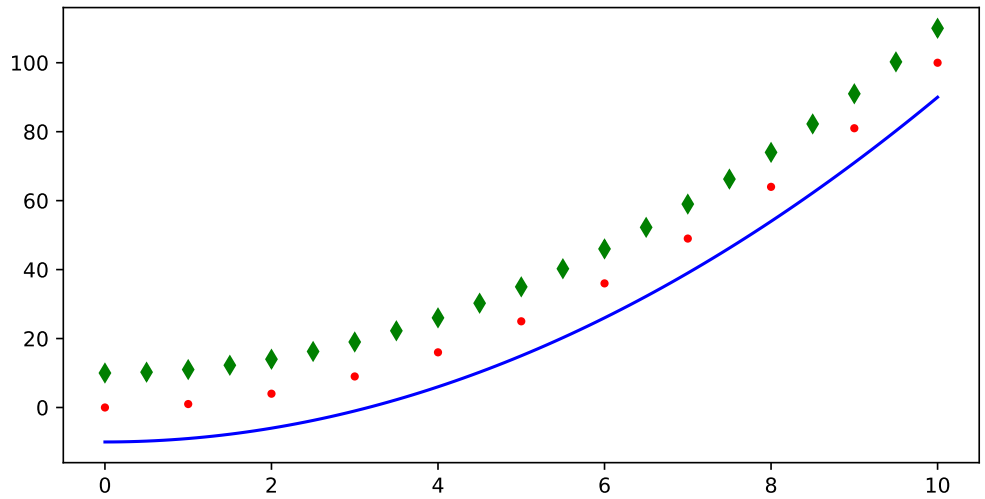
Si la première liste est omise, l'axe des abcisses prendra les valeurs $0, 1, ..., N$.
De plus, selon l'environnement utilisé, la fonction `show()` peut être nécessaire à l'affichage du graphique.

Exemple

Il est possible d'afficher plusieurs courbes sur la même figure.

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

x = np.linspace(0,10,21)
y = np.arange(0,11,1)
z = np.linspace(0,10,101)
plt.plot(x, x**2+10, 'dg')
plt.plot(y, y**2, '.r')
plt.plot(z, z**2-10, '-b')
plt.show()
```



Exemple détaillé

Pour cet exemple, nous nous baserons sur le fichier `ChuteLibre.txt` étudié précédemment.

```
import numpy as np
import matplotlib.pyplot as plt
# lecture des donnees depuis un fichier texte
tdata , ydata = np.loadtxt('ChuteLibre.txt' , usecols = (1 ,2),skiprows=4,
                                        unpack=True)
# creation des tableaux abscisses et ordonnees pour la courbe theorique
t = np.linspace(0, 4, 100)
y = 0.5*9.81*t**2
# creation de la representation
plt.figure('Ma fenetre de representation' , figsize = (8 ,4))
plt.plot(t, y, 'r-', label='Donnees theoriques( chute libre)')
plt.plot(tdata , ydata , 'bo' , label='Donnees experimentales')
plt.xlabel('Temps de chute [s]')
plt.ylabel('Distance verticale de chute [m]')
plt.legend(loc='upper left')
plt.title('Chute libre : comparaison entre theorie et experience')
# sauvegarder la representation dans un fichier pdf
plt.savefig('ChuteLibre.pdf')
# afficher la representation
plt.show()
```

Exemple détaillé

- `plt.figure()` crée une fenêtre destinée à contenir la figure. Il est possible de donner un nom à cette figure et de préciser ses dimensions par l'intermédiaire d'un argument nommé supplémentaire `figsize`.
- `plt.plot(t,y,arg_opt)` représente les données (t, y) qui se trouvent dans les `ndarray` `t` et `y`.
  Le troisième argument est une chaîne de caractères qui spécidie la couleur et le type de ligne ou de symbole qui doivent être utilisés pour représenter les données.
  Le `kwarg` `label` est une chaîne de caractères utilisée ensuite par la fonction `legend`.

Exemple détaillé

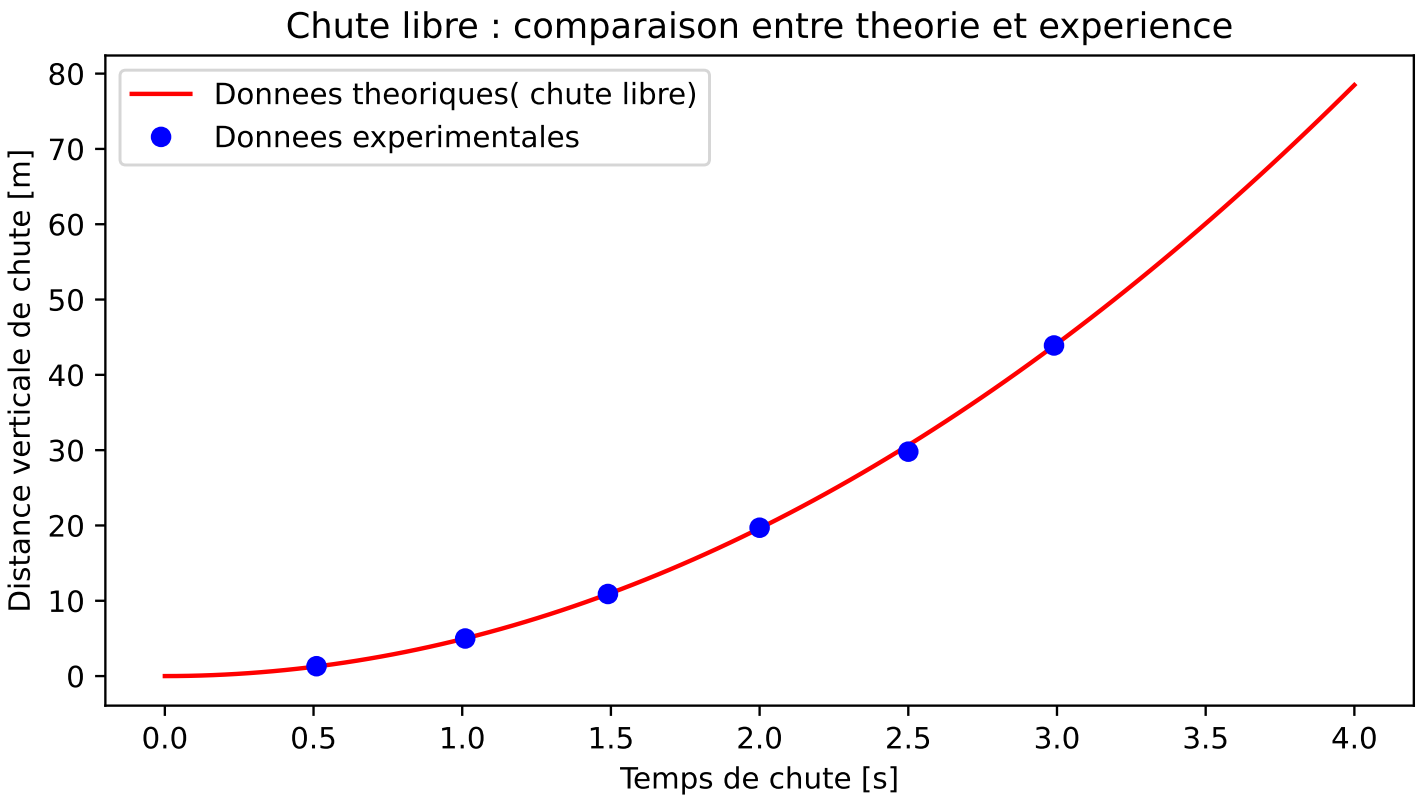
- `plt.xlabel(str)` utilise la chaîne de caractères donnée en argument pour spécifier la légende de l'axe x.
- `plt.ylabel(str)` utilise la chaîne de caractères donnée en argument pour spécifier la légende de l'axe y.
- `plt.legend()` crée la légende du graphique et la place à la position `loc`
- `plt.title(str)` ajoute un titre au graphique

Les dernières lignes permettent de :

- sauvegarder la figure au format et sous le nom souhaités
- d'afficher la représentation graphique.

Exemple détaillé

Voici le résultat final :



Remarques finales

Les `ndarray` ne sont pas particulièrement adaptés lorsqu'on :

- travaille avec des données hétérogènes,
- effectue des opérations sur des données possédant une hiérarchie (on utilise alors plutôt `Pandas`).

Il existe d'autres bibliothèques de visualisation graphiques (`Seaborn`, `Plotly`, ...).

Nous n'avons fait qu'effleurer les possibilités que nous offrent `NumPy` et `Matplotlib`. Vous pouvez réaliser quasiment tout ce qui est possible et imaginable !

Il n'y a qu'en pratiquant qu'on peut maitriser ces outils, n'hésitez donc pas à les utiliser pour vos futurs rapports, analyses de données, ou projets personnels !

Pensez toujours à consulter les *nombreux* forums en ligne, sites de soutien ou de documentation pour mieux appréhender ces outils !

Take Home Message

`Numpy` et `Matplotlib` sont deux librairies extrêmement répandues dans la communauté scientifique.

- On utilise des `ndarray` lorsqu'on veut effectuer des opérations mathématiques et logiques de manière efficace
- Matplotlib permet de visualiser graphiquement un ensemble de données, et de créer des figures professionnelles.