Bases de python et utilisation de numpy

September 27, 2021

1 Objectifs du cours

- Python 1 : Outils de bases pour les scientifiques
- Python 2 : Programmation orientée objet. Projet informatique (second semestre)

Python 1 : * Utilisation de numpy (aujourd'hui) * Approfondissement de Python * Statistiques, ajustement de courbes * Equations différentielles et calcul numérique avec la libraire scipy. * Transformée de Fourier

- TP les 7 et 8 octobre

1.1 Installation de Python

- Il existe plusieurs interpréteurs open source de Python. Le principal est CPython
- Il est fortement conseillé d'installer Anaconda et Python 3.8 (quelque soit le système d'exploitation)

1.2 Comment exécuter Python

- **Jupyter notebook** : très pratique (compte rendu de TP, exercices, cours). Pas pour des projets
- Spyder : éditeur de texte adapté à Python et l'environnement scientifique (à la matlab)
- IPython (inclus dans spyder) : terminal interactif. Faire des essais avant de copier dans un programme
- Console : interface graphique, test automatique, …

Rq : les TDs et l'examen se font sur des jupyter notebooks

2 Plan du cours

Aujourd'hui : Bases de python.

- Les types de bases de python
- Structures de contrôle
- Les fichiers
- Les exceptions

3 Types de données

3.1 Nombres

- entiers : pas de limite de tailles
- réels : flottant (float). Par défaut 64bits, précision relative de environ 10^{-15}
- Complexe : deux réels ; `a = 1 + 3J`
- +, -, *, /, ** (puissance)
- Modulo, division entière

[2]:
```
2**145
```

[2]:
```
44601490397061246283071436545296723011960832
```

[7]:
```
(1.0 + 1E-15) - 1
```

[7]:
```
1.1102230246251565e-15
```

[1]:
```
a = 4
print(a**2)
```

```
16
```

[14]:
```
z = 1 + 3J
print(z.imag)
```

```
3.0
```

[15]:
```
z.real
```

[15]:
```
1.0
```

[16]:
```
(1+3J).imag
```

[16]:
```
3.0
```

[13]:
```
T1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-274b7731451a> in <module>
----> 1 T1

NameError: name 'T1' is not defined
```

[11]:
```
a_1 = 13
```

[17]:
```
print(5%2)
print(5//2)
```

```
1
2
```

3.2 Booléens

- `True` et `False`
- Comparaison : `>`, `>=`, `==`, `<=`, `<`, `!=`
- Opérations : and, or et not (attention aux priorités)
- “évaluation paresseuse”

[18]:
```
(3>4)
```

[18]: `False`

[ ]:
```

```

[19]:
```
# a and b => si a return b else return False
```

[19]: 5

[21]:
```
from math import sqrt
x = -1

x>=0 and sqrt(x)>2
#if x>0:
#    return sqrt(x)
# else:
#    return False
(x>=0) & (sqrt(x)>2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-c02b465406b9> in <module>
      7 # else:
      8 #    return False
----> 9 (x>=0) & (sqrt(x)>2)

ValueError: math domain error
```

3.3 Chaînes de caractères (str)

- Plusieurs façon d’écrire une chaine : ”, ‘, ”””,”’ .
- Caractère spéciaux : \n (retour à la ligne), \t (tabulation)
- Unicode

- Concaténation
- Quelques méthodes sur les chaînes
- Formatage de chaîne de caractère

[23]:
```
s0 = 'Bonjour'
s1 = "Pierre"
s2 = "Aujourd'hui"
print(s1[2])
print(s1[2:5])
print(s0 + ' ' + s1) # Concaténation
'Aujourd\'hui'
```

```
e
err
Bonjour Pierre
```

[23]:
```
"Aujourd'hui"
```

[25]:
```
s3 = """Une chaine
sur plusieurs
lignes"""

print(s3)
```

```
Une chaine
sur plusieurs
lignes
```

[ ]:
```

```

[28]:
```
print("Rayon \u03B3")
print("Rayon  ")
  = 13
```

```
Rayon
Rayon
```

[28]:
```
13
```

[31]:
```
s1 = 'pierre'
s1.upper()
```

[31]:
```
'PIERRE'
```

[34]:
```
# Quelques méthodes
s = 'Bonjour'
print(s.replace('o', 'r'))
```

```
s = "1;2;4;3"
print(s.split(';'))
print(','.join(['1', '45']))

s = 'monfichier.txt'
print(s.endswith('.txt'))
```

```
Brnjrur
['1', '2', '4', '3']
1,45
True
```

3.3.1 Formatage

Mettre une variable dans une chaîne de caractère

On utilise la méthode .format (ancienne syntaxe avec le %). On utilise des accolades.

On peut préciser: * le nom de l'argument (keyword argument) * le type d'écriture : entier (d), virgule fixe (f), notation scientifique * la précision

Exemple {name:8.3f} : * argument : name * virgule fixe * taille totale 8 * 3 chiffres après la virgule

[40]:
```
# Formatage
from math import pi
print('La valeur de pi est {:+.3f}'.format(pi))

c = 299792458
print('La vitesse de la lumière est c={c:.3e} m/s'.format(c=c))

# Format string
print(f'La valeur de pi est {pi:.3f}')
```

```
La valeur de pi est +3.142
La vitesse de la lumière est c=2.998e+08 m/s
La valeur de pi est 3.142
```

3.4 Les listes

- Elles peuvent contenir n'importe quel type de donnée
- Les indices commencent par 0
- Index négatif : par la fin (modulo la taille de la liste)
- iterateur : par exemple `range`.

[44]:
```
l = ['Pierre', 34, 3.1415]
#print(l[-1])
print(l.append(25))
```

```
None
```

La méthode .append() modifie la liste. C'est toujours la même liste (comme on rajoute une page dans un classeur).

[45]:
```
l.insert(1, print)
l
```

[45]: ['Pierre', <function print>, 34, 3.1415, 25]

[46]:
```
l.append(l)
l
```

[46]: ['Pierre', <function print>, 34, 3.1415, 25, […]]

3.5 Créer une liste à partir d'une autre liste

- Boucle `for`
- Liste comprehension []
- `list.append` : méthode de l'objet liste

[49]:
```
liste_initiale = [1, 34, 23, 2.]

liste_finale = []
for elm in liste_initiale:
#    print(elm)
    liste_finale.append(elm**2)
print(liste_finale)

[elm**2 for elm in liste_initiale]
```

```
[1, 1156, 529, 4.0]
```

[49]: [1, 1156, 529, 4.0]

3.6 Parcourir une liste

- `for`
- enumerate
- zip

[52]:
```
ma_liste = ['Dupont', 'Martin', 'Dubois']
for name in ma_liste:
    print(name)

for i, name in enumerate(ma_liste):
    print(f'Le nom numéro {i} est {name}')
```

```
Dupont
Martin
```

```
Dubois
Le nom numéro 0 est Dupont
Le nom numéro 1 est Martin
Le nom numéro 2 est Dubois
```

[53]:
```python
liste_age = [12, 35, 23]
for age, name in zip(liste_age, ma_liste):
    print(f"{name} a {age} ans.")
```

```
Dupont a 12 ans.
Martin a 35 ans.
Dubois a 23 ans.
```

4 Structures de contrôle

4.1 Boucles

- while
- for (voir les listes)

4.2 Tests

- if, elif, else

4.3 Fonctions

```
def nom_fonction(arg1, arg2, ...):
      ...
      return out1, out2
```

- Argument optionel, argument nommé
- Il peut y avoir plusieurs return au sein d'une fonction
- Documentation commentaire

[60]:
```python
def exponentielle(x, precision=1E-9):
    """ Calcule e**x

    Utilise le dl sum(x**n/n!)

    Arguments :
        x : nombre dont on calcule exp
        precision : precision du calcul
    """
    result = 0
    n = 1
    term = 1 # Initial value
    while abs(term)>precision :
        result = result + term
        term = term * x/n
```

```
        n = n+1
    return result

#print(2*exponentielle(2.1, 1E-6))
#print(exponentielle(2.1))
#print(exponentielle(2.1, precision=1E-6))
exponentielle(1, precision=1e-5)
```

[60]:
```
2.71827876984127
```

[62]:
```
# Ne pas faire
lt = 35 # largeur de la table
```

5 Les tableaux numpy

- numpy est la librairie qui permet de manipuler de larges tableaux de données
- Elle évite de devoir faire des boucles
- Elle contient des fonctions qui ne sont pas dans math
- En pratique, on n'utilise jamais le module math

5.1 Plusieurs utilisations :

- simulations numériques
- données expérimentales
- graph

5.2 Remarque:

Taille fixée à la création. Tous les éléments de même type et de type fini (int64 pour les entiers)

[64]:
```
from numpy import *
import numpy as np
```

[65]:
```
# Examples
a = np.arange(10)
np.sin(a)
```

[65]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

[69]:
```
# Il ne faut pas utiliser le module math avec des tableaux numpy
import math

#a = np.array([1, 5, 9])
#math.sin(a)
```

[70]:
```
# Comparaison de la vitesse entre une liste et un tableau
a = np.random.rand(1000000)

%timeit a**2

def carre(x):
    return [elm**2 for elm in x]
b = list(a)

%timeit carre(b)
```

```
634 µs ± 95 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
132 ms ± 209 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[71]:
```
# Boucle for avec un tableau
def carre_boucle(x):
    out = np.zeros(len(x))
    for i in range(len(x)):
        out[i] = x[i]**2
    return out

%timeit carre_boucle(b)
```

```
172 ms ± 1.83 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

6 Avantages (et inconvénients) des tableaux

- La taille et le type de donnée est fixé à la création du tableau

[73]:
```
a = np.array([1, 2, 4], dtype='complex')
a[1] = 3.14



a
```

[73]:
```
array([1.  +0.j, 3.14+0.j, 4.  +0.j])
```

[75]:
```
a = np.array([1, 2, 4])
a[0] = 12.424
a
```

[75]:
```
array([12,  2,  4])
```

[77]:
```
np.arange(10, dtype='float')
```

[77]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

7 Création d'un tableau

Il existe plusieurs fonctions pour créer un tableau.

- array : partir d'une liste
- zeros, ones, eye
- arange
- linspace, logspace
- np.random.rand, ...
- loadtxt
- load/save

Le type est déterminé automatiquement. On peut le forcer avec l'argument dtype

[79]:
```
linspace(start=0, stop=1, num=10)
```

[79]:
```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

[80]:
```
logspace(0, 2, 11)
```

[80]:
```
array([  1.        ,   1.58489319,   2.51188643,   3.98107171,
         6.30957344,  10.        ,  15.84893192,  25.11886432,
        39.81071706,  63.09573445, 100.        ])
```

[82]:
```
np.random.normal(loc=1, scale=.1, size=5)
```

[82]:
```
array([0.90658076, 1.11398766, 1.10048681, 0.98594407, 0.88126721])
```

[86]:
```
a = np.random.normal(loc=1, scale=.1, size=5)
np.savetxt('test.txt', a)

b = np.loadtxt('test.txt')
b
```

[86]:
```
array([1.15199874, 1.10609476, 1.06479221, 1.16211909, 0.94786923])
```

[87]:
```
a = np.random.normal(loc=1, scale=.1, size=1000000)
np.savetxt('test_long.txt', a)
```

[88]:
```
%timeit np.loadtxt('test_long.txt')
```

```
1.62 s ± 99 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[91]:
```
np.loadtxt?
```

[89]:
```
a = np.random.normal(loc=1, scale=.1, size=1000000)
np.save('test_long.npy', a)
```

[90]:
```
%timeit np.load('test_long.npy')
```

```
802 µs ± 31 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

8 Fonctions vectorisées

C'est une fonction qui calcule sur un tableau élément par élément

[92]:
```
x = np.linspace(-1, 1, 51, endpoint=False)*np.pi
```

[93]:
```
# Souvent il n'y a rien a faire
np.sin(x)

def ma_fonction(x):
    return np.sin(x)**2 + np.cos(x)**2
ma_fonction(x)
```

[93]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[95]:
```
# Sinon, vectorize
# Mais il existe des solutions pour éviter d'avoir à
# l'utiliser (c.f. prochaine partie)
def mafonction_simple(a):
#    print('Bonjour')
    if a>0:
        return a
    else:
        return -a

#mafonction_simple(np.linspace(-1, 1))
mafonction = np.vectorize(mafonction_simple)
mafonction(np.linspace(-1, 1))
# On peut utiliser un décorateur
```

[95]:
```
array([1.        , 0.95918367, 0.91836735, 0.87755102, 0.83673469,
       0.79591837, 0.75510204, 0.71428571, 0.67346939, 0.63265306,
       0.59183673, 0.55102041, 0.51020408, 0.46938776, 0.42857143,
       0.3877551 , 0.34693878, 0.30612245, 0.26530612, 0.2244898 ,
       0.18367347, 0.14285714, 0.10204082, 0.06122449, 0.02040816,
       0.02040816, 0.06122449, 0.10204082, 0.14285714, 0.18367347,
       0.2244898 , 0.26530612, 0.30612245, 0.34693878, 0.3877551 ,
       0.42857143, 0.46938776, 0.51020408, 0.55102041, 0.59183673,
       0.63265306, 0.67346939, 0.71428571, 0.75510204, 0.79591837,
       0.83673469, 0.87755102, 0.91836735, 0.95918367, 1.        ])
```

[96]:
```
# Il faut connaitre l'origine de cette erreur
if x>0:
    print('Bonjour')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-96-f039710a8284> in <module>
      1 # Il faut connaitre l'origine de cette erreur
----> 2 if x>0:
      3     print('Bonjour')

ValueError: The truth value of an array with more than one element is ambiguous
 ↪Use a.any() or a.all()
```

8.1 Fonctions utiles de numpy

- Fonction mathématique véctorisée: `sin`, `cos`, `tan`, `exp`, `log`, `arcsin`, `arccos`, `arctan`, `arctan2`, ...
- Fonction sur les tableaux : `mean`, `min`, `max`, `sum`, `prod`, `std`, `var`
- Tri : `sort`, `argsort`

De façon générale, toutes les opérations usuelles existent, il faut juste trouver le bonne fonction !

[98]:
```
a = np.random.rand(10)
a.max() # method
np.max(a) # fonction
```

[98]:
```
0.7744770094089056
```

[100]:
```
np.arctan2(-1, -1)
```

[100]:
```
-2.356194490192345
```

9 Indexer un tableau

- Slices
  - Tous les éléments sauf le premier
  - Sauf le dernier
  - Un élément sur deux
- Indexer avec un tableau d'entier
- Indexer avec un tableau de booléens

[ ]:
```
x = np.linspace(-1, 1, 51, endpoint=False)*np.pi
x[2:9]
```

[ ]:
```
x[4:10:2]
# x[start:stop:step], comme range ou arange
```

[ ]:
```
x = np.linspace(-1, 1, 51)
x
# tout sauf le dernier
x[:-1]
```

[ ]:
```
# Les deux derniers
x[-2:]
```

[ ]:
```

```

[ ]:
```
# La différence entre deux éléments consécutifs (dérivée numérique)
x[1:] - x[:-1]
```

[ ]:
```
# Indexer avec un tableau d'entier
x[np.array([1, 5, 10])]
```

[ ]:
```
# Par exemple : argsort
# Les trois éléments les plus petits
x = np.random.rand(10)
x
```

[ ]:
```
x[x.argsort()][:3]
```

[ ]:
```
# Avec un tableau de booléens
x = np.random.rand(5)
x
```

[ ]:
```
x[np.array([True, False, False, True, True])]
```

[ ]:
```
x[x>0.5]
```

[ ]:
```
def val_abs(x):
    res = np.zeros(len(x))
    res[x>0] = x[x>0]
    res[x<=0] = -x[x<=0]
    return res
```

[ ]:
```
x[:, 1]
```

10 Tracer des graphs

- Initialiser l'affichage
  - c'est automatique dans spyder (à vérifier…)
  - Commande magique : %matplotlib
  - Ou bien %matplotilb inline
  - Dans un script c'est plus compliqué (graph interactif ou pas, sortie pdf, …)

- il faut importer les fonctions pour tracer.

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
```

[ ]:
```
# Données
X = linspace(-2,2, 100)
Y = sin(X)**2*exp(-X**2)
Y_noise = Y + .1*(np.random.rand(len(X))-0.5)

# Graph
plt.figure(figsize=(12, 8))
plot(X,Y, label=u"Theory")
plot(X,Y_noise,'o', label=u"Experiment")
xlabel(r'Voltage [V]')
ylabel(r'$\zeta$ [m]')
title("Nonsense graph")
legend(loc='upper left')
grid(True)
savefig('ma_figure.pdf')
savefig('ma_figure.svg')
```

[ ]:
```

```

10.1 Quelques commandes graphiques

- `plot(X,Y)`
- `loglog(X,Y)`, `semilogx(X,Y)`, `semilogy(X,Y)`
- `errorbar(X,Y, xerr=sig_X, yerr=sig_Y, fmt='o')`
- `xlabel('blabla')`, `ylabel('blabla')`, `title('blabla')`
- `xlim((x_inf, x_sup))`, `ylim((y_inf, y_sup))` pour zoomer sur une partie du graph
- `grid(True)` pour tracer une grille
- `subplot(nx,ny,m)` pour faires plusieurs plots
- `imshow(image)` pour tracer une matrice en fausse couleur et `colorbar()` pour tracer l'échelle
- `text(x,y,s)`
- `savefig(nom_fichier)`. Pour sauver une figure. Le formet est déterminé par l'extension. Utiliser `pdf` ou `svg` si on veut modifier le fichier (avec Inkscape par exemple).

Le site web de matplotlib regorge d'exemples avec tout type de graph.

[ ]:
```

```