Informatique Pour Tous

Cours 7 – Les bibliothèques Numpy et Matplotlib

En sciences, nous sommes très souvent conduits à utiliser des tableaux de données numériques (issues de calculs ou d'expériences) et il est commode de pouvoir « visualiser » ces données sous forme de courbes (l'homme est un animal visuel).
Ainsi nous allons introduire les objets de type tableaux, `array`, afin de pouvoir stocker facilement les données, comme des couples de points, pour pouvoir ensuite tracer des courbes.

A / La bibliothèque Numpy

A. 1 – Les tableaux

L'objet de type tableau, `array`, peut être considéré comme une variante de l'objet liste, `list`, mais avec les différences et caractéristiques suivantes :

✔ Tous les éléments d'un tableau doivent être du même type, de préférence des nombres entiers (`int`), des réels (`float`), des complexes (`complex`) et des booléens (`bool`) (cf. les masques) afin d'améliorer l'efficacité du stockage des données et du calcul numérique. On dit que les tableaux sont des **conteneurs homogènes** (les listes sont des conteneurs non homogènes parce qu'elles peuvent contenir des éléments de différents types.)
✔ Le nombre d'éléments du tableau doit être connu avant de créer le tableau. On peut voir le tableau comme une liste dont la taille est fixée à l'avance.
✔ Les tableaux ne font pas partie de la distribution standard de Python. Il faut donc faire appel à une bibliothèque spécifique (on parle aussi de module ou package en anglais) appelée **Numerical Python** abrégée par `numpy`.
✔ Avec `numpy`, une large gamme d'opérations mathématiques est directement disponible sur les tableaux, ainsi les boucles `for` sur les éléments du tableau peuvent être rendues implicites, c'est « automatique » comme nous allons le voir sur des exemples. On parle de **vectorisation**.
✔ Un tableau avec un seul indice est appelé un vecteur (`vector`). Il est bien sûr possible de manipuler des tableaux à plusieurs indices comme nous le verrons plus tard.

Un des gros avantages de stocker des données dans un objet de type `array` plutôt que dans un objet de type `list` est que l'utilisation des tableaux permet **aux programmes de tourner plus rapidement**. Cela peut être crucial pour de nombreuses applications qui nécessitent des mathématiques avancées dans les sciences et l'industrie.

Voici quelques commandes importantes sur les tableaux :

```
>>> import numpy as np
# Permet d'importer la bibliothèque Numerical Python abrégé en numpy avec l'alias communément utilisé .np

>>> a=np.array(r)
# Convertit l'itérable r en un tableau (assigné à la variable a). Attention, il faut que l'itérable ne contienne que des éléments de même type (float, int, complex ou bool)

>>> a=np.zeros(n)
# Crée un tableau de n éléments tous égaux à zéro (0/False).

>>> a=np.ones(n)
# Crée un tableau de n éléments tous égaux à un (1/True).
```

```
>>> a=np.zeros_like(c)
# Crée un tableau d'éléments tous égaux à zéro, de type identique à celui du tableau c (float, int, complex ou bool) et de même longueur que ce dernier.

>>> a=np.ones_like(c)
# Crée un tableau d'éléments tous égaux à un, de type identique à celui du tableau c (float, int, complex ou bool) et de même longueur que ce dernier.

>>> a=np.linspace(p,q,n)
# COMMANDE TRES UTILE. Crée un tableau de n éléments uniformément distribués sur l'intervalle (p,q). Un élément i du tableau est accessible par la commande a[i] (comme pour les listes). Le pas est de (q-p)/(n-1).

>>> a=np.arange(p,q,pas)
# COMMANDE TRES UTILE. Crée un tableau de (p-q)/pas éléments uniformément distribués sur l'intervalle [p,q] avec un pas de pas.

>>> np.size(a)
# Renvoie le nombre d'éléments contenus dans le tableau a (un entier).
```

Voici quelques exemples :

```
import numpy as np

print()
a=np.zeros(10)
print(a)
print(a[2])
print(np.size(a))

print()
b=np.linspace(1,10,20)
print(b)
print(b[3])
print(np.size(b))
print((10-1)/(20-1))

print()
c=np.arange(1,10,0.5)
print(c)
print(c[3])
print(np.size(c))
print((10-1)/0.5)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
0.0
10

[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
2.4210526315789473
20
0.47368421052631576

[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5 9.  9.5]
2.5
18
18.0
```

A. 2 – Application à la génération de tableaux d'abscisses et d'ordonnées à partir d'une fonction mathématique usuelle

Approche 1 : SCALAIRE

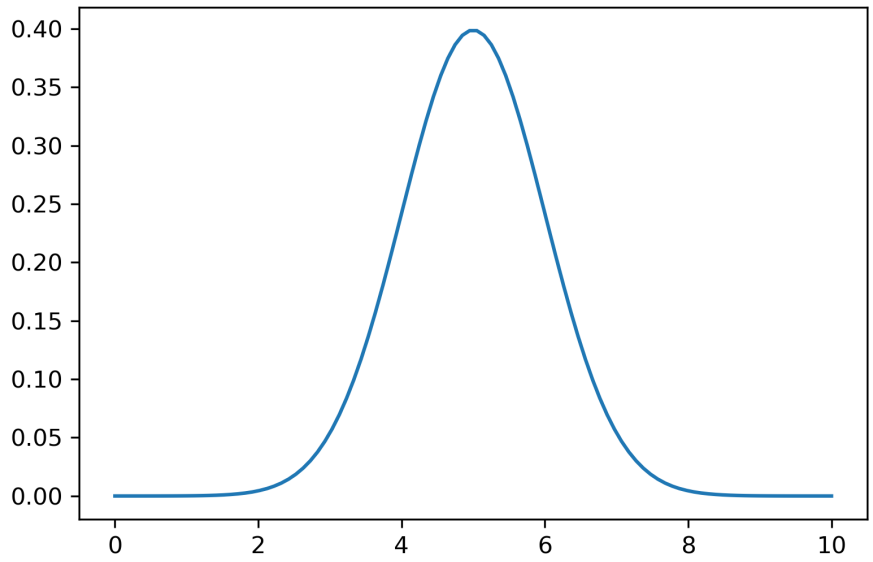
```
#importation des bibliothèques.
#from math import *
import numpy as np
import matplotlib.pyplot as plt

#définition de la fonction.
#ici oscillations amorties, régime pseudo-périodique.
#exp et sin appartiennent à la bibliothèques math.
#le paramètre d'entrée t est un float.
#la fonction retourne y, un float.
def f(t):
    y=(1/np.sqrt(2*np.pi))*np.exp(-(t-5)**2/2)
    return y

#création du tableau d'abcisses, 100 points
#répartis uniformement entre 0 et 10.
#création du tableau d'ordonnées rempli de 0,
#même taille que le tableau d'abcisses.
abscisse_t=np.linspace(0,10,100)
ordonnee_y=np.zeros(np.size(abscisse_t))

#avec la boucle for remplissage du tableau y
#élément par élément avec appel de la fonction f.
for i in range(np.size(abscisse_t)):
   ordonnee_y[i]=f(abscisse_t[i])

#génération de la courbe avec plot,
#cf. paragraphe suivant.
plt.plot(abscisse_t,ordonnee_y)
plt.savefig("fig1.png",dpi=300)
plt.show()
```



Approche 2 : VECTORISATION

```
#importation des bibliothèques.
from math import *
import numpy as np
import matplotlib.pyplot as plt

#définition de la fonction.
#ici oscillations amorties, régime pseudo-périodique.
#np.exp et np.sin appartiennent à la bibliothèques numpy.
#le paramètre d'entrée t est un tableau de float.
#la fonction retourne un tableau de float.
def f(t):
    y=4*np.exp(-0.5*t)*np.sin(3*t)
    return y

#création du tableau d'abscisses, 100 points
#répartis uniformement entre 0 et 10.
#création du tableau d'ordonnées rempli de 0,
#même taille que le tableau d'abscisses.
abscisse_t=np.linspace(0,10,100)
ordonnee_y=np.zeros(np.size(abscisse_t))

#on procède par vectorisation.
#on travaille directement avec les tableaux.
#plus besoin de boucle for, plus rapide.
ordonnee_y=f(abscisse_t)

#génération de la courbe avec plot,
#cf. paragraphe suivant.
plt.plot(abscisse_t,ordonnee_y)
plt.show()
```

B / La bibliothèque Matplotlib

Il est facile de tracer des courbes sous Python avec l'utilisation couplée de la bibliothèque `matplotlib` et de la bibliothèque `numpy` comme les exemples précédents le montrent. Afin de voir les commandes essentielles, nous allons reprendre l'exemple précédent afin d'améliorer l'aspect visuel du graphe par ajouts de labels des abscisses et des ordonnées, d'une légende etc… Je vous invite à vous référer au mémento Python qui résume les commandes essentielles disponibles de `matplotlib`.

```
>>> import matplotlib.pyplot as plt
# Permet d'importer la bibliothèque matplotlib avec l'alias communément utilisé.plt
```

Vous aurez l'occasion d'utiliser ces bibliothèques de nombreuses fois dans les cours de physique-chimie et de sciences de l'ingénieur en plus du cours d'informatique évidemment.

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

def f1(t):
    y=4*np.exp(-0.3*t)*np.sin(3*t)
    return y
def f2(t):
    y=4*np.exp(-0.3*t)
    return y

#création du tableau des abcisses:
#100 points répartis uniformément entre 0 et 10.
abscisse_t=np.linspace(0,10,100)
#création des tableaux des ordonnées,
#remplis de zéro et même taille que abscisse_t.
ordonnee_y1=np.zeros(np.size(abscisse_t))
ordonnee_y2=np.zeros(np.size(abscisse_t))
ordonnee_y3=np.zeros(np.size(abscisse_t))
#remplissage des tableaux d'ordonnées par VECTORISATION.
ordonnee_y1=f1(abscisse_t)
ordonnee_y2=f2(abscisse_t)
ordonnee_y3=-f2(abscisse_t)

#obtention des graphes plt.plot(tableau_abcisses,tableau_ordonnées,'options').
plt.plot(abscisse_t, ordonnee_y1, 'r-')
plt.plot(abscisse_t, ordonnee_y2, 'b+')
plt.plot(abscisse_t, ordonnee_y3, 'g--')
#labelisation des axes.
plt.xlabel('temps')
plt.ylabel('y')
#affichage d'une légende.
plt.legend(['exp(-0.3*t)*sin(3*t)','4*exp(-0.3*t)','-4*exp(-0.3*t)'])
#affichage d'un titre.
plt.title('Plusieurs courbes sur le même graphe')
#affichage d'une grille.
plt.grid()
#sauvegarder la figure.
plt.savefig("fig2.png",dpi=300)
#affichage des graphes à l'écran.
plt.show()
```