TD – 01

Exercices d'applications

Savoirs et compétences :

- ❒ *Alg – C17 : tris d'un tableau à une dimension de valeurs numériques (tri par insertion, tri rapide, tri fusion).*

Exercice 1 – Calcul de somme

Pour cet exercice, on prend $n = 1000000$. On pourra augmenter ou diminuer cette valeur en fonction de la machine utilisée.

1. Calculer $\sum_{i=0}^{n} i$ sans utiliser `numpy`.
2. Chronométrer le temps nécessaire pour le calcul précédent, par exemple en utilisant `time.clock()`.
3. Utiliser un tableau `numpy` et la méthode `sum` pour calculer à nouveau la somme proposée.
4. Comparer le temps de calcul avec la méthode précédente.

Correction
```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-
import numpy as np
from time import clock
# 1ère méthode : avec une boucle
t0 = clock()
a = range(1000001)
t1 = clock()
S = 0
for i in a:
    S += i
t2 = clock()
print('1ère méthode : {:6.6f} + {:6.6f} secondes'.format(t1-t0,t2-t1))

# 2ème méthode : avec une liste en compréhension
t0 = clock()
a = range(1000001)
t1 = clock()
S = sum(i for i in a)
t2 = clock()
print('2ème méthode : {:6.6f} + {:6.6f} secondes'.format(t1-t0,t2-t1))
```

Correction
```
# 3ème méthode : avec numpy
t0 = clock()
v = np.array(range(1000001))
t1 = clock()
S = v.sum() # ou np.sum(v)
t2 = clock()
print('3ème méthode : {:6.6f} + {:6.6f} secondes'.format(t1-t0,t2-t1))
```

Le calcul de la somme lui-même est extrêmement rapide avec numpy. La définition de l'itérateur range est extrêmement rapide (il n'est pas évalué lors de sa définition) La définition d'un tableau numpy est un peu lent, car il faut compter avec les problèmes d'allocation mémoire. Mais une fois qu'il est défini, on peut faire des calculs très rapides.

Exercice 2 – Produit de Wallis

On peut justifier que : $\pi = 2\prod_{n=1}^{+\infty} \frac{4n^2}{4n^2-1}$ appelé le *produit de Wallis*.

1. Écrire une fonction itérative, d'argument n, calculant : $2\prod_{i=1}^{n} \frac{4i^2}{4i^2-1}$.
2. Écrire une fonction utilisant un tableau `numpy`, effectuant le même calcul.
3. Comparer le temps de calcul de ces deux fonctions.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np
from time import clock

# 1ère méthode : fonction itérative
def wallis (n):
    p = 2
    for i in range(1, n+1):
        j = 4 * i * i
        p *= j / (j - 1)
    return p

t0 = clock()
print(wallis(1000000))
t1 = clock()
print('1ère méthode : {:6.6f} secondes'.format(t1-t0))
# 3.1415918681913633
# 1ère méthode : 0.313095 secondes

# 2ème méthode : avec un tableau numpy
def wallis_np (n):
    x = np.arange(1, n+1, dtype=np.float)
    x **= 2
    x *= 4
    y = x - 1
    x /= y
    return 2*np.prod(x)

t0 = clock()
print(wallis_np(1000000))
t1 = clock()
print('2ème méthode : {:6.6f} secondes'.format(t1-t0))
# 3.14159186819
# 2ème méthode : 0.020723 secondes

# On peut encore mieux faire :
def wallis_np (n):
    x = np.arange(1, n+1, dtype=np.float)
    return 2 * (1 / (1-1/(4*x**2)))
```

Exercice 3

1. Définir une matrice aléatoire `a` de taille 50×50.
2. Déterminer la valeur $\max_{i,j} |a_{i,j+1} - a_{i,j}|$.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

a = np.random.rand(5,5)
# matrice aléatoire d'éléments de [0,1]

b = a[:,1:] - a[:,:-1]
# matrice de tous les a_{i,j+1} - a_{i,j}

print(np.max(np.abs(b)))
```

```
# a_{i,j+1}-a_{i,j} est la discrêtisation de $\frac{\partial a}{\partial j}$
# Donc c'est la plus grande valeur absolue de la dérivée partielle par rapport àj
```

Exercice 4

1. Définir une matrice aléatoire de flottants `a` de taille 50×50.
2. Compter le nombre de valeurs inférieures à 0.5.
3. Remplacer toutes les valeurs inférieures à 0.5 par 0, et celles strictement supérieures à 0.5 par 1.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

a = np.random.rand(5,5)
print(a)
# matrice aléatoire d'éléments de [0,1]

# On utilise le fait que l'on peut sommer les booléens :
print(True+True)
# 2
print(False+False)
# 0

# On construit la matrice des booléens en comparant terme àterme a et .5
b = a <= .5
print(b)

# On somme tous les termes, ce qui donne le nombre de True
print(np.sum(b))

# On utilise un masque pour modifier les valeurs de a
a[a <= .5] = 0
a[a > .5] = 1
print(a)
```

Exercice 5

D'après exemple 3.15 p 28 « Algèbre linéaire », Robert C. Dalang, Amel Chaabouni.
On s'intéresse au système linéaire suivant :

$$\begin{cases} x_1 + 2x_2 + x_3 + x_4 = 0 \\ x_2 + 2x_3 + x_4 = 0 \\ x_1 + x_3 + 2x_4 = 1 \\ 2x_1 + x_2 + x_4 = 0 \end{cases}$$

1. Vérifier qu'il n'y a qu'une solution à ce système.
2. En utilisant `np.linalg.solve`, déterminer cette solution.
3. Vérifier le résultat obtenu en utilisant un produit matriciel.
4. Construire la matrice `m` de format 4×5, dont les colonnes sont successivement les colonnes de `a` et `b`.
5. Appliquer à `m` la méthode du pivot de Gauss pour résoudre « à la main » le système proposé.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

a = np.array([[1, 2, 1, 1],
              [0, 1, 2, 1],
              [1, 0, 1, 2],
              [2, 1, 0, 1]])
b = np.array([0, 0, 1., 0])

print('Le déterminant du système vaut {} <> 0 donc unique \
```

```
solution'.format(np.linalg.det(a)))
# Le déterminant du système vaut -4.0 <> 0 donc unique solution

x = np.linalg.solve(a,b)
print(x)
#[-0.5 0.  -0.5  1. ]

print('Vérification : on doit trouver le vecteur nul : \
{}'.format(a.dot(x) - b))
# Vérification : on doit trouver le vecteur nul : [ 0. 0. 0. 0.]

b.shape = (4,1)
m = np.concatenate((a,b), axis=1)
print(m)
# [[ 1. 2.  1.  1.  0.]
#  [ 0. 1.  2.  1.  0.]
#  [ 1. 0.  1.  2.  1.]
#  [ 2. 1.  0.  1.  0.]]

m[2,:] = m[2,:] - m[0,:]
m[3,:] = m[3,:] - 2 * m[0,:]
print(m)
# [[ 1. 2.  1.  1.  0.]
#  [ 0. 1.  2.  1.  0.]
#  [ 0. -2. 0.  1.  1.]
#  [ 0. -3. -2. -1. 0.]]

m[2,:] = m[2,:] + 2 * m[1,:]
m[3,:] = m[3,:] + 3 * m[1,:]
print(m)
# [[ 1. 2.  1.  1.  0.]
#  [ 0. 1.  2.  1.  0.]
#  [ 0. 0.  4.  3.  1.]
#  [ 0. 0.  4.  2.  0.]]

m[3,:] = m[3,:] - m[2,:]
print(m)
# [[ 1. 2.  1.  1.  0.]
#  [ 0. 1.  2.  1.  0.]
#  [ 0. 0.  4.  3.  1.]
#  [ 0. 0.  0. -1. -1.]]

m[2,:] = m[2,:] + 3 * m[3,:]
m[1,:] = m[1,:] + 1 * m[3,:]
m[0,:] = m[0,:] + 1 * m[3,:]
print(m)
# [[ 1. 2.  1.  0. -1.]
#  [ 0. 1.  2.  0. -1.]
#  [ 0. 0.  4.  0. -2.]
#  [ 0. 0.  0. -1. -1.]]

m[1,:] = m[1,:] - .5 * m[2,:]
# ici, il est impératif de travailler en flottants

m[0,:] = m[0,:] - .25 * m[2,:]
print(m)
# [[ 1.  2.  0.  0. -0.5]
#  [ 0.  1.  0.  0.  0. ]
#  [ 0.  0.  4.  0. -2. ]
#  [ 0.  0.  0. -1. -1. ]]

m[0,:] = m[0,:] - 2 * m[1,:]

m[2,:] /= 4
m[3,:] /= -1
print(m)
# [[ 1.  0.  0.  0. -0.5]
#  [ 0.  1.  0.  0.  0. ]
#  [ 0.  0.  1.  0. -0.5]
#  [-0. -0. -0.  1.  1. ]]

print('La solution est {}'.format(m[:,4]))
# La solution est [-0.5 0. -0.5 1. ]
```

Exercice 6

Créer une matrice 8×8, remplie de 0 et de 1 comme un échiquier.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np
# Solution 1
m = np.zeros((8, 8), dtype = np.bool)
m[1::2, ::2] = True
m[::2, 1::2] = True
print(m)
# [[False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]
#  [False True False True False True False True]
#  [ True False True False True False True False]]


# Solution 2
m = np.zeros(64, dtype=np.int)
m[::2] = 1
m.shape = 8, 8
print(m)
m.dtype = np.bool
print(m)
```

Exercice 7

1. Créer une matrice aléatoire de taille 5×15, constituées d'entiers et l'afficher.
2. Mettre à zéro tous les éléments de la première ligne de cette matrice.
3. Déterminer la moyenne des éléments de cette matrice.
4. Construire le vecteur dont les composantes sont les moyennes des lignes de la matrice.
5. Déterminer la moyenne des éléments de ce vecteur.

Correction

```
#! /usr/local/bin/python3.4
# -*- coding:utf-8 -*-

import numpy as np

m = np.random.randint(0, 10, (5, 15))
print(m)
# [[2 9 1 0 9 9 0 6 1 0 5 8 9 3 7]
#  [8 6 5 1 6 5 2 9 0 8 9 0 0 2 5]
#  [5 5 1 1 6 7 8 6 5 9 9 2 3 3 6]
#  [6 1 6 3 0 3 9 6 0 5 5 8 2 8 9]
#  [0 5 9 1 8 9 5 9 3 7 4 2 0 0 9]]

m[0,:] = 0
print(m)
# [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
#  [8 6 5 1 6 5 2 9 0 8 9 0 0 2 5]
#  [5 5 1 1 6 7 8 6 5 9 9 2 3 3 6]
#  [6 1 6 3 0 3 9 6 0 5 5 8 2 8 9]
#  [0 5 9 1 8 9 5 9 3 7 4 2 0 0 9]]

moyenne = m.mean()
print(moyenne)
# 3.78666666667

v = m.mean(axis=1)
print(v)
# [ 0.         4.4        5.06666667 4.73333333 4.73333333]
```

```
print(v.mean())
# 3.78666666667
```