Python et le calcul matriciel

Le module numpy contient les éléments indispensables à la modélisation des vecteurs, matrices et plus généralement des tableaux multidimensionnels. Pour cela, numpy fournit le type *ndarray*, qui, bien que très proche sur le plan syntaxique du type *list* , diffère de ce dernier sur plusieurs points importants :

- la taille des tableaux numpy est fixée au moment de la création et ne peut plus être modifiée par la suite [1] ;
- les tableaux numpy sont *homogènes*, c'est-à-dire constitués d'éléments de même type.

En contrepartie, l'accès aux éléments d'un tableau numpy est incomparablement plus rapide, ce qui justifie pleinement leur usage pour manipuler des matrices de grandes tailles.

Par la suite, nous supposerons avoir écrit au début de chacun des scripts de ce chapitre l'instruction :

```
import numpy as np
```

ce qui nous permettra de visualiser aisément les functions de ce module : elles seront préfixées par `.np`.

1.1 Création de tableaux

On utilise en général la fonction `array` pour former un tableau à partir de la liste de ses éléments (ou de la liste des listes pour une matrice bi-dimensionnelle). Par exemple, pour créer une matrice 3×4 à partir de ses éléments on écrira :

```
>>> a = np.array([[3, 6, 2, 0], [2, -3, 5, -1], [0, 6, 1, -1]])
>>> a
array([[ 3,  6,  2,  0],
       [ 2, -3,  5, -1],
       [ 0,  6,  1, -1]])
```

La matrice que nous venons de créer est une matrice à coefficients entiers car tous ses éléments sont des entiers ; si on change l'un des coefficients, le nouveau coefficient sera au préalable converti en entier, ce qui peut provoquer une confusion :

```
>>> a[0, 0] = 1.25
>>> a
array([[ 1,  6,  2,  0],
       [ 2, -3,  5, -1],
       [ 0,  6,  1, -1]])
```

Si la liste des éléments est hétérogène, certains seront automatiquement convertis. Par exemple, si la liste des coefficients contient des éléments de type *float* et de type *int*, ces derniers seront convertis en flottants :

```
>>> a = np.array([[3., 6, 2, 0], [2, -3, 5, -1], [0, 6, 1, -1]])
>>> a
array([[ 3.,  6.,  2.,  0.],
       [ 2., -3.,  5., -1.],
       [ 0.,  6.,  1., -1.]])
```

Aussi, pour éviter toute ambiguïté il est préférable de préciser lors de la création le type des éléments souhaités avec le paramètre optionnel `dtype` (pour *data type*) :

```
>>> b = np.array([1, 7, -1, 0, -2], dtype=float)
>>> b
array([ 1.,  7., -1.,  0., -2.])
```

On accède à un élément d'un tableau numpy exactement de la même façon que pour une liste : à partir de son indice.

```
>>> b[0]
1.0
```

Pour les tableaux multidimensionnels, outre la syntaxe usuelle `a[i][j]` il est aussi possible d'utiliser la syntaxe `a[i, j]` :

```
>>> a[0, 0]
3.0
```

Le slicing suit la même syntaxe que pour les listes Python. On retiendra surtout la syntaxe pour obtenir une vue d'une colonne ou d'une ligne d'une matrice :

```
>>> a[2]                    # 3e ligne de la matrice a
array([ 0.,  6.,  1., -1.])

>>> a[:, 2]                 # 3e colonne de la matrice a
array([ 2.,  5.,  1.])
```

EX 1
Taper ce code et dire ce que fait le programme

```
M = 2
N = 2

A = [[0 for i in range(N)] for j in range(M)]
B = [[0 for i in range(N)] for j in range(M)]
C = [[0 for i in range(N)] for j in range(M)]

for i in range(M):
    for j in range(N):
        A[i][j] = int(input("Saisir l'élement A[{0}][{1}] : ".format(i, j)))

for i in range(M):
    for j in range(N):
        B[i][j] = int(input("Saisir l'élement B[{0}][{1}] : ".format(i, j)))

for i in range(M):
    for j in range(N):
        C[i][j] = A[i][j]+B[i][j]

for i in range(M):
    for j in range(N):
        print(C[i][j], end=" , ")
    print("")
```

EXERCICE 2

```
M = 2
N = 2

A = [[0 for i in range(N)] for j in range(M)]
C = [[0 for i in range(N)] for j in range(M)]

num = int(input("Saisir un scalaire : "))

for i in range(M):
    for j in range(N):
        A[i][j] = int(input("Saisir l'élement A[{0}][{1}] : ".format(i, j)))

for i in range(M):
    for j in range(N):
        C[i][j] = num*A[i][j]

for i in range(M):
    for j in range(N):
        print(C[i][j], end=" , ")
    print("")
```