17

Tableaux à 2 dimensions, images

I. Les tableaux de Numpy

◇ On rencontre souvent des données qui se représentent naturellement par des tableaux à 2 dimensions (matrices, images). On pourrait utiliser des listes de listes mais le module **numpy** propose une structure de données adaptées : la structure *array*.

◇ Cette structure permet de regrouper dans un tableau des éléments de même type (en général *int*, *float* ou *complex*). Ce tableau peut avoir plusieurs dimensions mais on rencontrera uniquement deux cas : des tableaux à 1 dimension (analogue des vecteurs en mathématiques) ou 2 dimensions (analogue des matrices en mathématiques).

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

A :

i \ j	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	0	0	0

```
>>> (n,m) = A.shape
>>> print(n,m)
3 4
```

⚠ Dans un *array* (contrairement aux listes) tous les éléments doivent être du même type, en général *int* ou *float* ;

```
B = np.zeros((2,3))
print(B)
```

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

```
C = np.random.randint(0,6,(2,3))
print(C)
```

```
[[3 3 5]
 [3 4 1]]
```

II. Application : traitement d'images

◇ On dispose d'un tableau A de taille $n \times m$ représentant une image en niveaux de gris : chaque coefficient de A est un nombre entier compris entre 0 (noir) et 255 (blanc).

◇ Pour éclaircir A, on définit un nouveau tableau E avec $E[i,j] = \sqrt{A[i,j]}$. Pour que les coefficients de E restent entiers et compris entre 0 et 255 on pose en fait $E[i,j] = \lfloor 16\sqrt{A[i,j]} \rfloor$.

```
(n,m) = A.shape
E = np.zeros((n,m))
for i in range(0,n):      # 0<=i<=n-1
    for j in range(0,m): # 0<=j<=m-1
        E[i,j] = int(16*A[i,j]**0.5)
```

III. Calculs mathématiques sur les tableaux

◇ Le module NUMPY propose des fonctions mathématiques analogues à celles du module **math** mais qui s'appliquent aux tableaux coefficient par coefficient. Exemple :

```
A = np.array([[0,1],[2,4]])
E = np.sqrt(A)
print(E)
```

```
[[ 0.  1. ]
 [ 1.41421356  2. ]]
```

Le traitement d'image précédent pouvait donc s'écrire simplement :

```
E = np.floor(16*np.sqrt(A))
```

◇ On dispose ainsi des fonctions **np.exp**, **np.log**, **np.cos**, etc. On peut également réaliser des opérations mathématiques « coefficient par coefficient » :

```
B = A+1
print(B)
```

```
[[1 2]
 [3 5]]
```

```
C = A**2
print(C)
```

```
[[ 0  1]
 [ 4 16]]
```

```
D = 2*A
print(D)
```

```
[[0 2]
 [4 8]]
```