13

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 mathématique : vecteurs de $\mathbb{R}^n$ ou $\mathbb{C}^n$) ou 2 dimensions (analogue mathématiques : matrices de $\mathscr{M}_{np}(\mathbb{R})$ ou $\mathscr{M}_{np}(\mathbb{C})$).

◇ Construction et découpage :

```
>>> import numpy as np
>>> x=np.array([1,2,3])
>>> x[1]
2
>>> A=np.array([[1,2],[3,4]])
>>> A[1,1]
4
>>> A[1,:]
array([3, 4])
>>> A[:,1]
array([2, 4])
>>> B=np.zeros((2,3))
>>> np.random.randint(0,6,7)
array([0, 1, 4, 3, 5, 4, 1])
```

◇ Quelques exemples d'opérations :

```
>>> import numpy as np
>>> A=np.array([[1,2],[3,4]])
>>> np.floor(np.sqrt(A))
array([[ 1.,  1.],
       [ 1.,  2.]])
>>> A*A
array([[ 1,  4],
       [ 9, 16]])
>>> A**2
array([[ 1,  4],
       [ 9, 16]])
>>> A+1
array([[2, 3],
       [4, 5]])
```

Remarque. Le module **numpy** propose des versions spécifiques des fonctions usuelles qui peuvent travailler sur des tableaux (**np.sin**, **np.exp**, etc.). □

⚠
- Dans un *array* (contrairement aux listes) tous les éléments doivent être du même type, en général *int* ou *float* ;
- Les *array* de type *int* peuvent avoir un comportement étrange : on les évitera en utilisant une commande du type :

```
>>> import numpy as np
>>> A=np.array([[1,2],[3,4]],float)
```

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 tel que : $E[i,j] = \lfloor 16\sqrt{A[i,j]} \rfloor$:

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

◇ On bruite l'image en ajoutant des valeurs aléatoires à la matrice A. On fait en sorte que les coefficients de la nouvelle matrice restent compris entre 0 et 255 :

```
B=np.clip(A+np.random.randint(-25,26,(n,m)),0,255)
```

◇ Pour débruiter l'image B, on crée une nouvelle image D telle que $D[i,j]$ soit la moyenne des 9 valeurs de B situées autour de $B[i,j]$ (pour simplifier, on garde les pixels au bord de D noirs) :

```
D=np.zeros((n,m))
for i in range(1,n-1):
        for j in range(1,m-1):
                D[i,j]=np.mean(B[i-1:i+2,j-1:j+2])
```