TP N°16 : Le module incontournable du calcul scientifique : Numpy
Les matrices

On importera la bibliothèque Numpy au moyen de la commande : `import numpy as np`

Création des tableaux bi-dimensionnels

Création avec np.array

On définit souvent un tableau numpy à partir d'une lise de listes, en utilisant la fonction array du module numpy. La fonction array agit comme un mécanisme de conversion de type 'list' vers 'numpy.ndarray' qui est le type commun à tous les tableaux numpy.

```
a = np.array([[1, 2, 3],[2, 3, 4],[0, 1, 0]]) # créer une matrice à partie d'une liste de listes
print(type(a))  # <class 'numpy.ndarray'>
print(a.dtype) # int64
b = a.tolist() #créer un objet b de type list, copie de l'objet a de type ndarray
```

Création avec des fonctions spéciales

Découvrons quelques fonctions spéciales : np.ones, np.zeros, np.eye, np.diag,...
Rappelons que chez les booléens, False = 0 = None = « vide », et True = 1 = not None = « non vide » :

```
a = np.ones((3, 5)) # crée une matrice de 3 lignes × 5 colonnes, dtype = float (par défaut)
b = np.ones((3,5), dtype = np.int)
c = np.zeros((3, 5))
d = np.zeros((3, 5), dtype = np.bool)
e = np.eye(3) # Crée une matrice identité d'ordre n
g = np.eye(4,4)
h = np.diag([1,2,3])  # Crée une matrice diagonale
i = np.diag([1,2,3], k = 1) # essayer : -1, 2 (décalage de la diagonale)
A = np.ones((2,3))
B = np.zeros((2,3))
AB0 = np.concatenate((A,B), axis = 0) # crée une matrice par concaténation de B sous A
AB1 = np.concatenate((A,B), axis = 1) # crée une matrice par concaténation de B à droite de A
# np.column_stack :
v1 =np.array([1,2,3]); v2 =np.array([12,22,32]); v3 =np.array([13,23,33])
v = np.column_stack((v1,v2,v3)) # crée une matrice par empilement des vecteurs passés en arguments.
#Autres méthodes utiles
v.fill(1) # remplir le tableau v par une valeur constante.
a.fill(3.14) # remplir le tableau a par la valeur flottante 3.14
v.fill(3.14) # La valeur flottante 3.14 a été convertie en valeur entière 3 pour le remplissage
z = np.array([[1,2],[3,4]]*2) #tableau avec des répétitions
```

Tableaux répondant à une formule donnée

La fonction fromfunction permet de construire un tableau dont le terme général obéit à une formule donnée.

```
def f(i,j): return 10*i+j
t1 = np.fromfunction(f,(4,5)) # t1[i,j] = f(i,j)
t2 = np.fromfunction(f,(4,5), dtype = int) # On exige un tableau de type int
# Le même tableau peut être obtenu en convertissant une liste (liste de listes) "en compréhension"
t3 = np.array([[f(i,j) for j in range(5)] for i in range(4)])
```

Exemple : Matrice d'Hilbert, de terme général : $H[i,j] = \frac{1}{(i+j+1)}$ avec $i, j \geq 0$

```
t4 = np.fromfunction(lambda i,j: 1/(i+j+1),(4,4))
```

Exercice 1

Construire un objet de type numpy.ndarray

1. dont les termes sont les multiples de 3 compris entre 0 et 27 ;
2. de taille égale à 100 dont les extrémités sont 0 et 27 ;

Lecture de valeurs dans une matrice

Il est possible de sélectionner une sous matrice, en ne gardant que quelques lignes consécutives et/ou certaines colonnes consécutives. La spécification la plus générale d'une coupe d'un tableau M est :

```
M[ numero_ligne : numero_colonne ]
-----------------------------------------------------------------------------------------
M[ ligne_début(inclue) : ligne_fin(exclue) , colonne_début(inclue) : colonne_fin(exclue) ]
```

Si M est une matrice d'ordre $a * b$:

- `M[a,b]` : élément en position (a,b)
- `M[a] ou M[a,:]` : Vecteur ligne en position a
- `M[:,b]` : Vecteur colonne en position b
- `M[a:b,c:d]` : lignes de a à b-1, colonnes de c à d-1
- `M[:,:]` : une copie intégrale de M avec nouvelle référence
- `M[:a,:b]` : sous matrice les lignes du début jusqu'à a exclu ; les colonnes de début jusqu'à b exclu (Les a preimères lignes, les b premières colonnes).
- `M[::-1]` : On inverse l'ordre des lignes
- `M[:,::-1]` : On inverse l'ordre des colonnes
- `M[::2,::2]` : lignes et colonnes paires
- `M[1::2,1::2]` : lignes et colonnes impaires

```
c = np.array( range(24) )
c.shape = (3, 4, 2) ; print (c)
print(c[1,:,:])
print(c[1][0][2]) # troisième élément de la première colonne de la deuxième ligne du tableau c
```

Accès aux données dans un ordre quelconque :
Si m est une matrice, et si I est un tableau d'indices (éventuellement une liste ou un tuple), alors $m[I]$ renvoie le tableau (de même format que le tableau I) formé des éléments $m[i]$, où i est dans I.

```
m = np.arange(24).reshape(4,6); m
a = m[[3,1,0,1]] # on demande le tableau des lignes m[3] , m[1] , m[0] et m[1]
i = np.array([3,0,2]) # trois indices
m.take(i,axis=0) # lectures de lignes
m.take(i,axis=1) # lectures de colonnes
m.take(i) # lecture de m "aplati"
```

Écriture de valeurs dans une matrice

Exemples :

```
a = np.array(range(10))
a.shape = (2, 5) ; print(a)
a[0,0] = 7 # modifier un élément d'indice (0,0)
a[0,:] = np.zeros(5) # modifier la ligne d'indice 0
a[:,0] =np.ones(2) # modifier la colonne d'indice 0
b = np.concatenate((a,a), axis = 0) ;
b[2:4,2:4] = np.ones((2,2)) # modifier une sous-matrice
a[ a % 2 == 0 ] /= 2  ; print(a) # les éléments pairs de a sont divisés par 2
```

Quelques méthodes sur les tableaux

On peut utiliser les méthodes des objets (ou les fonctions associées du module `numpy`) : `a.max()`, `a.min()`, `a.sum()`, `a.prod()`, `a.mean()` (moyenne arithmétique), `np.apply_along_axis`.

```
a = np.random.randint(0, 10, (2,5)) ; print(a)
print(a.sum()) # ou encore np.sum(a)
a.sum(axis=0) # somme des lignes
```

```
a.sum(axis=1) # somme des colonnes
print(a.mean()) #moyenne arithmétique
print(a.mean(axis=0))
print(a.mean(axis=1))
m = np.arange(15).reshape(3,5); m
np.apply_along_axis(np.mean, 0, m) # moyenne lignes
np.apply_along_axis(np.mean, 1, m) # moyenne colonnes
print(a.max())
print(a.max(axis=0))
print(a.max(axis=1))
```

Exercice 2

1. Définir une matrice aléatoire a de flottants, 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.