NUMPY. QUELQUES MÉTHODES UTILES

Le module **numpy** permet de gérer les matrices numériques. On peut l'importer (par exemple) en écrivant `import numpy as np`. En voici quelques méthodes souvent utilisées.

1. Création d'un vecteur ligne (tableau unidimensionnel) :
   - syntaxe : `A = np.array(L)`
   - argument : une liste L $= [x_0, x_1, ..., x_{n-1}]$
   - exemple : `A = np.array`$([4, -2.3, 5.67])$ donne A $= (4, -2.3, 5.67)$
2. Création d'une matrice :
   - syntaxe : `A = np.array(L)`
   - argument : une liste de listes L $= [L_0, L_1, ..., L_{n-1}]$. Chaque sous-liste représente une ligne de la matrice.
   - exemple : `A = np.array(` $[[1, 2], [3, 5]]$) donne A $= \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}$.
3. La fonction `np.arange`. Elle joue le même rôle que la fonction `range` qui générait des listes, mais c'est elle qu'il faut utiliser pour les tableaux de numpy.
   - syntaxe : `np.arange(debut,fin,pas)` génère un tableau à une dimension formé d'entiers ou de flottants, commençant à **debut**, et finissant à **fin** (valeur exclue), avec un **pas**. Si vous n'indiquez pas la valeur de **pas**, Python considère implicitement que **pas** $= 1$.
   - exemples : `B = np.arange(1, 7)` : tableau formé des entiers de 1 à 6.
     `B = np.arange(1, 2, 0.1)` : tableau formé de flottants, B = [1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
   - Contrairement à la fonction range, les paramètres **debut**, **fin** et **pas** peuvent ici être des nombres flottants. Vous pouvez aussi avoir des valeurs négatives :
     `A = np.arange(-3.0, 4.0, 0.5)` donne A = [-3. -2.5 -2. -1.5 -1. -0.5 0. 0.5 1. 1.5 2. 2.5 3. 3.5].
4. `np.linspace` : c'est une fonction très utilisée car très pratique.
   - `A = np.linspace(debut,fin,N)` crée un tableau à une dimension formée de N nombres flottants régulièrement répartis entre **debut** et **fin** inclue (attention, c'est ici une exception à la convention générale de Python qui exclut d'habitude la dernière valeur. Ce n'est pas le cas ici et **fin** fait partie du tableau créé).
   - exemple : `A = np.linspace(1, 5, 10)` crée un tableau de 10 flottants entre 1 et 5.
5. `np.zeros( (n,p) )` (attention aux parenthèses !) crée une matrice de n lignes et p colonnes, dont les éléments sont 0.
6. `np.ones( (n,p) )` fonctionne de façon identique à `np.zeros` mais les éléments de la matrice générée sont tous égaux à 1.
7. Addition d'un scalaire : ajoute x à chaque élément de la matrice A et met le résultat dans B
   - syntaxe : `B = A + x`
   - exemple : A $= \begin{pmatrix} 1 & -3 \\ 4 & 5 \end{pmatrix}$ et `B = A + 2` donne B $= \begin{pmatrix} 3 & -1 \\ 7 & 8 \end{pmatrix}$.
8. Multiplication par un scalaire : multiplie chaque élément de la matrice A par le scalaire x et met le résultat dans B

- syntaxe : `B = A * x`
- exemple : A = $\begin{pmatrix} 1 & -3 \\ 4 & 5 \end{pmatrix}$ et `B = A * 2` donne B = $\begin{pmatrix} 2 & -6 \\ 8 & 10 \end{pmatrix}$.

9. Division par un scalaire non nul : divise chaque élément de la matrice A par le scalaire x et met le résultat dans B
   - syntaxe : `B = A / x`
   - exemple : A = $\begin{pmatrix} 1 & -3 \\ 4 & 5 \end{pmatrix}$ et `B = A/2` donne B = $\begin{pmatrix} 0.5 & -1.5 \\ 2 & 2.5 \end{pmatrix}$.
10. Produit de deux matrices
    - syntaxe : `C = A.dot(B)`
    - argument : une matrice numpy `C = A.dot(B)` représente le produit matriciel AB.
    - exemple : A = $\begin{pmatrix} 1 & -3 \\ 4 & 2 \end{pmatrix}$ B = $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ `C = A.dot(B)` donne C = $\begin{pmatrix} -5 \\ 8 \end{pmatrix}$
11. Si A est une matrice numpy, alors `A.size` est son nombre d'éléments.
12. Si A est une matrice numpy, alors `A.shape` renvoie un tuple (n,p) où n est le nombre de lignes de A et p son nombre de colonnes.
13. Si A est un tableau numpy unidimensionnel contenant N éléments et si $N = n \times p$, alors `A.reshape(n,p)` renvoie une matrice à n lignes et p colonnes.
    - exemple : A = `np.array`([1.2,3.4,2.1,6.7]) et `M = A.reshape`(2,2) donne $M = \begin{pmatrix} 1.2 & 3.4 \\ 2.1 & 6.7 \end{pmatrix}$.

      Autre exemple : A = `np.arrange`(1, 11) et M = `A.reshape`(2,5) donne une matrice 2 lignes, 5 colonnes de la forme $M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix}$.
14. Ce module contient aussi toutes les fonctions usuelles comme `np.cos`, `np.sin`, `np.exp`, `np.log`, etc ... Ces fonctions agissent directement sur tous les élément d'une matrice. Par exemple : `B = np.cos(A)` est la matrice dont tous les éléments sont les cosinus des éléments de la matrice A.
15. **Parcourt de tableaux numpy** :
    (a) Si A est un tableau numpy unidimensionnel, alors :
    - A[i] est l'élément de rang i du tableau A
    - A[debut : fin] est le sous-tableau contenant les élément dont le rang va de **debut** (inclu) à **fin** (exclu).

      Si vous omettez **debut** ou **fin**, cela donne :
    - A[: fin] qui est est le sous-tableau composé des éléments allant du rang 0 à fin - 1. Cette syntaxe est identique à A[0 : fin].
    - A[debut :] est le sous-tableau composé des éléments allant du rang debut jusqu'au dernier élément de A. Cette syntaxe est identique à A[debut : N] si A possède N éléments.

    (b) Si M est une matrice possédant n lignes et p colonnes, alors
    - M[i , j] est l'élément de M situé sur la ligne i et la colonne j. Faites toujours très attention : la première ligne ou première colonne porte le numéro 0.
    - Vous pouvez aussi extraire des sous-matrices à partir d'une matrice M grâce à la syntaxe :

      M[deb_ligne : fin_ligne , deb_col : fin_col]