Feuille de cours 8 (informatique) : Tableaux

La bibliothèque `numpy` de Python permet de manipuler des tableaux multidimensionnels. On se limitera à des tableaux à double entrée, correspondant mathématiquement aux *matrices*. Pour utiliser la bibliothèque `numpy` on commencera par l'importer via la commande :

```
import numpy as np
```

1 Premiers tableaux et opérations élémentaires

Les tableaux Python sont une structure de données permettant de stocker des variables *de même type*. Contrairement aux listes, tous les éléments d'un tableau doivent donc être du même type, on considèrera par exemple des tableaux de :

Pour définir une variable `A` contenant le tableau bidimensionnel suivant à n lignes et p colonnes

$$A = \begin{array}{|c|c|c|c|} \hline x_{11} & x_{12} & \dots & x_{1p} \\ \hline x_{21} & x_{22} & \dots & x_{2p} \\ \hline \vdots & \vdots & & \vdots \\ \hline x_{n1} & x_{n2} & \dots & x_{np} \\ \hline \end{array}$$

il faut utiliser la syntaxe suivante :

```
A = np.array([[x11, ... ,x1p],[x21, ... ,x2p], ... ,[xn1, ... ,xnp]])
```

Exemples : Créer les tableaux `A` et `B` correspondant aux matrices suivantes :

- $A = \begin{pmatrix} 1 & 4 & 3 \\ 2 & 0 & 6 \end{pmatrix}$
- $B = \begin{pmatrix} 5 & 2 \\ 1 & 3 \\ -1 & 2 \end{pmatrix}$

Remarques :

- **Attention** aux doubles crochets `[[...],[...],...,[...]]`. En Python, un tableau se présente en fait sous la forme d'une liste de listes. Toutefois, utiliser la bibliothèque `numpy` va nous offrir plus d'options que d'utiliser de simples listes de listes.
- Notez que dans cette syntaxe, on donne les coefficients d'un tableau *ligne par ligne* (et non colonne par colonne).

1.1 Tableaux de base

Il existe des commandes (à connaître) permettant de construire des tableaux simples rapidement :

- `np.eye(n)` renvoie la matrice identité de taille n.
  Par exemple : `np.eye(4)` vaut

- `np.zeros((n,p))` renvoie la matrice de taille $n \times p$ ne contenant que des 0.
  Par exemple : `np.zeros((2,3))` renvoie

- `np.ones((n,p))` renvoie la matrice de taille $n \times p$ ne contenant que des 1.
  Par exemple : `np.ones((3,2))` renvoie

- `np.diag(L)` où `L` est une liste de longueur n renvoie la matrice diagonale de taille $n \times n$ dont les coefficients diagonaux sont ceux de `L`.
  Par exemple, `np.diag([3,6,1,2])` renvoie

Remarque : Attention aux s et aux doubles parenthèses pour les fonctions `np.zeros` et `np.ones`.

1.2 Opérations

Les opérations de base `+,-,*,/` sur des tableaux sont réalisées **coefficients par coefficients**. Par exemple, si
`A = np.array([[3,1,0],[2,1,5]])` et `B = np.array([[4,1,1],[0,2,1]])`
alors :

- `2*A` contient

- `A+B` contient

- `A*B` contient

Remarques :

- Pour que ces opérations aient un sens, il faut que les tableaux `A` et `B` soient de même taille. Si ce n'est pas le cas alors Python affiche le message d'erreur suivant :

```
ValueError: operands could not be broadcast together with shapes (a,b) (c,d)
```

- **Attention**, la syntaxe `A*B` n'effectue donc **pas** le produit matriciel de `A` et `B` (pour réaliser ce produit, voir paragraphe 3.1).
- On peut aussi appliquer une fonction de la bibliothèque `numpy` à un tableau. Là encore la fonction s'applique alors coefficient par coefficient. Par exemple pour la matrice A ci-dessus, `np.exp(A)` renverra

Exercice 1
En utilisant les commandes précédentes, créer rapidement la matrice $A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$.

2 Numérotation et parcours

Pour récupérer le nombre de lignes et le nombre de colonnes d'un tableau `A`, on utilise les commandes suivantes :

```
np.size(A,0) # (0 pour les lignes)
np.size(A,1) # (1 pour les colonnes)
```

Remarque : Attention à ne pas oublier le deuxième argument 0 ou 1. Si cet argument n'est pas précisé, c'est-à-dire si on demande `np.size(A)`, alors Python renvoie le nombre total d'éléments de `A` c'est-à-dire le produit np si `A` a n lignes et p colonnes.

2.1 Accès aux coefficients

Si `A` est un tableau à n lignes et p colonnes alors ses coefficients sont numérotés

`A[i,j]` avec $i \in$ et $j \in$

Comme en mathématiques, le premier indice correspond au numéro de la ligne et le deuxième au numéro de la colonne.

Attention ! ! ! En mathématiques, pour une matrice $A \in \mathcal{M}_{n,p}(\mathbb{K})$ on utilise la notation $A_{i,j}$ avec $i \in [\![1, n]\!]$ et $j \in [\![1, p]\!]$, en Python la numérotation commence à 0, comme pour les listes !

Par exemple, si $A = \begin{pmatrix} 10 & 3 & 5 & 0 \\ 2 & 6 & -1 & -8 \\ 4 & 9 & -2 & 7 \end{pmatrix}$ alors :

- `A[2,1]` renvoie
- `A[1,3]` renvoie
- le coefficient 5 de A s'obtient par
- le coefficient 7 de A s'obtient par
- `A[2,4]` renvoie

Comme les listes, les tableaux sont des objets mutables : on peut modifier un coefficient du tableau placé en position (i, j). Il suffit pour cela d'écrire :

```
A[i,j] = nouvelle_valeur
```

Exercice 2
Que contient A après exécution du script suivant ?

```
A = np.array([[0,1,4],[2,1,5]])
A[0,1] = 7
A[1,1] = 9
```

Remarque : les commandes `A[i,:]` et `A[:,j]` permettent respectivement d'accéder aux listes correspondant à la i-ème ligne de `A` et à la j-ème colonne de `A`. On peut donc aussi modifier directement une ligne ou une colonne en entier.

2.2 Création par parcours à partir de la matrice nulle

Lorsqu'on veut créer une matrice qui ne s'obtient pas directement via les fonctions élémentaires décrites dans le paragraphe 1.1, la stratégie usuelle consiste à :

1. créer une matrice nulle A de la bonne taille $n \times p$, puis
2. à parcourir ses lignes et ses colonnes pour la remplir en modifiant les 0 qu'elle contient pour les remplacer par le coefficient souhaité.

En pratique on utilisera donc la syntaxe suivante :

1.
2.

Exercice 3
Créer les matrices A et B définies par les expressions ci-dessous :

1. $A \in \mathcal{M}_n(\mathbb{R})$ telle que pour tous $i, j \in [\![0, n-1]\!]$, `A[i,j]` $= i + j$.
2. $B \in \mathcal{M}_{n,p}(\mathbb{R})$ telle que pour tous $i, j \in [\![0, n-1]\!] \times [\![0, p-1]\!]$, `B[i,j]` $= ij$.

3 Exercices classiques et entraînement

3.1 Exercices classiques (questions de cours)

Exercice 4
Écrire une fonction `produit_mat` prenant en arguments deux matrices A et B et renvoyant leur produit matriciel. Votre fonction devra renvoyer un message d'erreur dans le cas où le produit matriciel n'est pas possible.
Remarque : bien sûr, cette fonction existe déjà en Python : on obtient le produit matriciel de `A` et `B` par la commande `np.dot(A,B)` (mais on s'interdit de l'utiliser ici).

Exercice 5
Écrire une fonction `transposer` prenant en argument une matrice A et renvoyant sa transposée.
Remarque : bien sûr, cette fonction existe déjà en Python : on obtient la transposée de `A` par la commande `np.transpose(A)` (mais on s'interdit de l'utiliser ici).

3.2 Entraînement

Exercice 6
Écrire une fonction `somme` prenant en argument une matrice `A` et renvoyant la somme de tous ses coefficients. Par exemple, si `A` $= \begin{pmatrix} 4 & 1 & 2 \\ 3 & 0 & 1 \end{pmatrix}$ alors `somme(A)` doit renvoyer $4+1+2+3+0+1 = 11$.

Exercice 7
Écrire une fonction `recherche` prenant en argument une matrice `A` et un nombre `x` et renvoyant `True` si `x` apparaît dans la matrice `A` et `False` sinon.

Exercice 8
Écrire une fonction `cree` prenant en argument un entier n et renvoyant la matrice $A_n \in \mathcal{M}_n(\mathbb{R})$ donnée par $A_n = \begin{pmatrix} 1 & n & n & n & \dots & n \\ n & 2 & n & n & \dots & n \\ n & n & 3 & n & \dots & n \\ \vdots & & & \ddots & & n \\ n & \dots & & n & n-1 & n \\ n & & \dots & & n & n \end{pmatrix}$. Par exemple, `cree(4)` doit renvoyer $A_4 = \begin{pmatrix} 1 & 4 & 4 & 4 \\ 4 & 2 & 4 & 4 \\ 4 & 4 & 3 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix}$.
On proposera deux versions de la fonction, l'une utilisant les opérations et fonctions élémentaires du paragraphe 1.1 ; l'autre réalisant la matrice A_n par un parcours à partir de la matrice initialement remplie de n.