INF 9 - Matrices avec Numpy

Le langage Python permet de travailler avec des tableaux multidimensionnels : dans le cas de tableau de dimension 2, cela correspond à une matrice.

Les extensions à charger pour traiter les problèmes d'algèbre linéaire en Python sont :

- `numpy` chargée avec la commande `import numpy as np`
- `numpy.linalg` qui contiendra des fonctions permettant notamment la résolution de systèmes linéaires.

1. CRÉATION DE MATRICES

Dans la première partie du cours, on se consacrera la création de matrices.

La première commande à connaitre est la commande `np.array` qui crée un tableau en le décrivant explicitement. Par exemple, que fait le code suivant ?

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(M)
```

Il crée et affiche la matrice dont les lignes successives dont les vecteurs explicitement donnés.

Les commandes `np.zeros`, `np.eye`, `np.ones` permettent de créer des matrices sur la base des matrices remarquables étudiées en cours.

La commande `np.zeros`, déjà étudiée, peut s'appeler avec comme argument un couple (m, n). On crée alors un tableau à n lignes et m colonnes rempli de zéros Ainsi la commande

```
M = np.zeros((3,4))
print(M)
```

affiche un tableau rempli de zéros à 3 lignes et 4 colonnes.

La commande `np.ones` fonctionne de la même façon mais crée une matrice remplie de 1.

La commande `np.eye` sert à créer des matrices identité. La matrice I_n est construite avec la commande

```
n=10 ##taille de la matrice que l'on peut modifier
I = np.eye(n)
```

On peut aussi taper

```
n=10
m=4 ##taille de la matrice que l'on peut modifier
I = np.eye(n,m)
```

pour construire une matrice dans $M_{n,m}(\mathbf{R})$ dont les seuls coefficients non nuls sont les diagonaux (et égaux à 1).

2. ACCÈS AUX COEFFICIENTS, MODIFICATION DE MATRICES

Commandes induites par les commandes de calcul classique. Les commandes suivantes, fonctionnant pour les réels et les entiers, sont encore valables pour les matrices. On avait déjà vu qu'elles restaient valables pour les vecteurs : c'est tout l'esprit de `numpy` de "vectoriser" un certain nombres d'opérations.

- les opérations arithmétiques de base : `+`, `-`, `*`, `/`, `**`. Les opérations sont effectuées **coefficient par coefficient**. En particulier **la commande `A*B` ne réalise pas de produit matriciel!**
- les opérations de comparaisons qui renvoient des **matrices de booléen** : les opérations `==`, `>`, `<`, `>=`, `<=`, `!=` réaliseront des comparaisons terme à terme pour créer la matrice des résultats. La comparaison à un nombre est autorisée : elle renvoie le résultat de la comparaison avec une matrice dans tous les coefficients sont ce nombre.

Accès, modification. Une fois une matrice créée :

- on a accès à la taille de la matrice par la commande
  `a,b= np.shape(M)`
  La commande affecte à `a` le nombre de lignes et à `b` le nombre de colonnes. C'est très intéressant si on veut ensuite faire un boucle sur les lignes ou sur les colonnes.
- on a accès directement à ses coefficients en tapant `M[i,j]`. (On rappelle que les lignes et les colonnes sont numérotées à partir de zéro. On accès aux lignes de la matrice en tapant `M[i]` et aux colonnes en tapant `M[:,j]`. Il existe aussi des commandes pour extraire toute sorte de sous-matrice.

- on peut modifier les coefficients ou les colonnes en réalisant une affectation à l'aide des commandes précédentes. Par exemple que fait la fonction suivante ?

```
def matrice(n):
    M = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            M[i,j]=i+j
    return(M)
```

Quelques opérations sur les matrices.

- Le **produit matriciel** est réalisé par la commande `np.dot(A,B)` qui renvoie le produit $A \times B$ s'il existe.
- La transposée d'une matrice est donnée par `np.transpose(A)`.

EXERCICES

Exercice 1 Avec Python, coder les matrices

$$A = \begin{pmatrix} 1 & 3 & 2 & 4 \\ -5 & 2 & -1 & 3 \\ 1 & 0 & 2 & -7 \end{pmatrix} \text{ et } B = \begin{pmatrix} -1 & -1 & -1 \\ 2 & -1 & 3 \\ 4 & 12 & 6 \\ 0 & -2 & -3 \end{pmatrix}$$

1. Calculer les produits AB et BA,
2. Calculer $(AB)^7$ et $(BA)^1 1$.
3. Calculer la matrice dont les coefficients sont les exponentielles de ceux de A,
4. Calculer la matrice M des cosinus des coefficients de $(BA)^{11}$.
5. Calculer $(BA)^2 M^3$.

Exercice 2 Écrire une fonction qui prend en entrée une matrice carrée et renvoie la somme de ses coefficients diagonaux. Si la matrice entrée n'est pas carrée, elle doit renvoyer un message d'erreur.

Vous pouvez programmer une deuxième fonction qui marchera aussi pour les matrices rectangulaires.

Exercice 3 Écrire un programme qui prend en entrée un entier $n > 0$ et construit la matrice

$$M = \begin{pmatrix} n & 2 & \cdots & 2 & \vdots \\ 2 & n & \ddots & 2 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 2 \\ \vdots & \cdots & & 2 & n \end{pmatrix}$$

1. avec des boucles for
2. sans aucune boucle for

Exercice 4 Construire en Python :

1. une fonction N qui prend en argument un entier naturel n non nul et qui renvoie la matrice de $M_n(\mathbf{R})$ avec des 1 sur la première colonne, la dernière colonne et la diagonale, des 0 ailleurs.
2. une fonction Z qui prend en argument un entier naturel n non nul et qui renvoie la matrice de $M_n(\mathbf{R})$ avec des 1 sur la première ligne, la dernière ligne et l'antidiagonale, des 0 ailleurs.

Exercice 5

1. Écrire une fonction qui prend en entrée deux réels a et b ainsi qu'un entier n et renvoie une matrice de $\mathrm{M}_n(\mathbf{R})$ dont les coefficients sont des nombres aléatoires uniformes entre a et b. (On utilisera `rd.random`).
2. Écrire une fonction qui renvoie une matrice symétrique dont les entées sont des v.a. similaires.
3. Simuler 100 matrices avec le procédé précédant et déterminer la moyenne des sommes de coefficients diagonaux obtenus. Commenter.