Python td 1 - Révisions sur les matrices

I. Définir une matrice

On importe les librairies `numpy` : `import numpy as np` et `numpy.linalg` : `import numpy.linalg as al`

1. Définition d'une matrice

Pour les matrices de petit format, la manière la plus simple est de les définir coefficient par coefficient, les éléments d'une même ligne étant entre crochets :

Tapez dans la console : A=np.array([[1,2,3],[4,5,6],[7,8,9]])

Puis tapez print(A) pour afficher cette matrice

Définissez ensuite la matrice ligne $L = \begin{pmatrix}1 & 4 & 5\end{pmatrix}$, puis la matrice colonne $C = \begin{pmatrix}1\\4\\5\end{pmatrix}$.

2. Opérations sur les matrices

On peut effectuer les opérations usuelles sur les matrices :

- $+, -$: somme ou différence de deux matrices de même taille;
- $*$: multiplication d'une matrice par un scalaire;
- `np.dot(A,B)` : produit des deux matrices A et B lorsque ce produit est bien défini ($A \times B$ existe si le nombre de ..... de A est égal au nombre de ..... de B);
- `np.transpose(A)` : transposée d'une matrice A.
- `al.matrix_power(A,n)` :
  calcule A^n

On peut également effectuer des opérations coefficient par coefficient

- $A * B$: si A et B ont les mêmes tailles, calcule le produit coefficient par coefficient
- A/B : si A et B ont la même taille et B n'a aucun coefficient nul, calcule le quotient coefficient par coefficient
- $A * *n$: élève chaque coefficient de A à la puissance n (attention, ceci N'EST PAS A^n !!!!)

On peut aussi faire les opérations suivantes :

- Application d'une fonction à un tableau coefficient par coefficient : `np.exp(A)` par exemple
- Additionner le même réel à tous les coefficients d'un tableau : `A+4` par exemple

Tapez les instructions suivantes et regardez ce qui se passe :

```
B=2*A
C=np.matrix_power(A,2)
D=A**2
E=np.cos(A)
F=np.matrix_power(A,2)+np.log(A)
```


II. Matrices prédéfinies

1. Matrices classiques

   - La matrice nulle à n lignes et p colonnes est définie par **A=np.zeros([n,p])**
   - La matrice remplie de 1, possédant n lignes, p colonnes, est définie par **J=np.ones([n,p])** :
   - La matrice unité I_n s'obtient par la commande **I=np.eye(n)**
   - On peut retrouver la taille d'une matrice (son nombre de lignes et de colonnes) par la commande **a,b =np.shape(A)**
   - L'instruction **A=np.zeros([n])** définit une matrice unidimensionnelle (avec un seul crochet) à n cases.
     Python distingue cette matrice de la matrice **np.zeros([n,1])**. Il peut être intéressant de travailler avec des matrices unidimensionnelles, notamment pour les tracés de courbes.

2. Construction d'une matrice "à pas"

   (a) **La commande** `np.linspace(a,b,n)` crée une matrice de premier terme a, de dernier terme b (avec $a < b$) et contenant n coefficients régulièrement espacés.
   Tapez par exemple `L1=np.linspace(0,10,11)` puis `L2=np.linspace(0,10,21)` et affichez les résultats correspondants.
   Il s'agit d'une matrice unidimensionnelle.

   (b) **La commande** `np.arange(a,b,p)` crée une matrice de premier terme a, contenant des coefficients régulièrement espacés avec le pas p, et n'atteignant pas b. Avec cette commande, redéfinir les matrices L_1 et L_2 ci-dessus.
   Il s'agit également de matrices unidimensionnelles.

III. Extraction de données d'un tableau

Attention, les lignes et colonnes des matrices en Python sont numérotées à partir de 0.

Par exemple si l'on reprend la matrice A=np.array([[1,2,3],[4,5,6],[7,8,9]])

$$A = \begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$$

alors la commande $a = A[0,0]$ renvoie $a = 1$, la commande $b = A[1,2]$ renvoie $b = 6$.

Plus généralement pour avoir le coefficient $a_{i,j}$ de la matrice $A = (a_{i,j})_{1\leq i,j\leq n}$, on devra taper `A[i-1,j-1]`.

Modifiez la matrice A pour obtenir la matrice

$$A = \begin{pmatrix}1 & 2 & 3\\4 & 1 & 6\\7 & 8 & 1\end{pmatrix}$$

Pour extraire la ième ligne du tableau A, on écrit : **L=A[i-1,:]**

Pour extraire la jème colonne du tableau A, on écrit : **C=A[:,j-1]**

IV. Exercices

Exercice 1
Définir la matrice A à 10 lignes et 11 colonnes de la forme suivante :

$$\begin{pmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

Exercice 2
Demander un entier n à l'utilisateur puis créer une matrice B de taille $n \times n$ de la forme :

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ 0 & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

Exercice 3
Quelle est la matrice affichée par le script ci-dessous (pour n quelconque) ?

```
C=np.ones([n,1])
L=[np.arange(1,n+1,1)] #crochets pour avoir un tableau à deux dimensions
A=np.dot(C,L)
print(A)
```

Calculer A^k où $k \in \mathbb{N}^*$.

V. Opérations sur les tableaux de nombres

Soit A une matrice, vue comme un tableau de nombres (les instructions ci-dessous n'utilisent pas les opérations matricielles).

`np.sum(A)` somme les cases de A;
`np.prod(A)` renvoie le produit des cases de A; (pas dans le programme officiel)
`np.min(A)` renvoie le minimum des cases de A;
`np.max(A)` renvoie le maximum des cases de A;
`np.mean(A)` renvoie la moyenne des cases de A;
`np.cumsum(A)` renvoie une matrice ligne égale à la somme cumulée des cases de A;
`np.cumprod(A)` renvoie une matrice ligne égale à le produit cumulé des cases de A; (pas dans le programme officiel)
`np.median(A)` renvoie la médiane des valeurs de A;
`np.var(A)` renvoie la variance des valeurs de A;
`np.std(A)` renvoie l'écart type des valeurs de A.

Nous nous servirons de ces instructions notamment en statistiques et en simulation d'expériences aléatoires. Ces instructions permettent également des calculs très rapides de certaines sommes ou produits.