Introduction aux matrices

Vos réponses aux questions devront être enregistrées dans le fichier TP1.py disponible sur l'ENT. Ce fichier est à déposer sur la zone de dépôt à la fin du TP.

Définition 1. Une matrice M de dimensions $n \times p$ est un tableau de nombres à n lignes et p colonnes.
Ces nombre sont appelés **coefficients** de la matrice.

Exemple 1. $M = \begin{pmatrix} 1 & 3 & 4 \\ -1 & 2 & 0 \\ 5 & -6 & 2 \end{pmatrix}$ est une matrice à 3 lignes et 3 colonnes.

Remarques 1.

1. Pour $1 \leq i \leq n$ et $1 \leq j \leq p$, on note $m_{i,j}$ le coefficient de la matrice M situé au croisement de la i-ème ligne et de la j-ième colonne.
   Que vaut $m_{2,3}$ pour la matrice M ci-dessus?
2. On note $\mathcal{M}_{n,p}(\mathbb{R})$ l'ensemble des matrices réelles $n \times p$.

Création de matrices avec Python

Les modules `numpy` et `numpy.linalg` permettent d'appeler les librairies dont nous aurons besoin. Modules que l'on importe avec les commandes :

```
import numpy as np
import numpy.linalg as alg
```

Pour définir une matrice, on utilise la fonction `array` du module `numpy`. Par exemple la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ peut se définir de la manière suivante via la console (notez bien le nombre de crochets) :

```
A = np.array([[1, 2, 3], [4, 5, 6]])
```

Les fonctions `zeros` et `ones` permettent de créer des matrices particulières remplies de 0 ou de 1.
Par exemple la **matrice nulle** $B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ peut se définir de la manière suivante via la console :

```
B = np.zeros((2,3))
```

La matrice $C = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$ par :

```
C = np.ones((3,2))
```

La fonction `diag` permet de créer une **matrice diagonale** dont seuls les termes de la diagonale sont éventuellement non nuls : la matrice $E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$ se définit par :

```
E = np.diag([1,2,3])
```

Enfin la fonction `eyes` permet de créer des **matrices identités**, c'est à dire des matrices diagonales, qui n'ont que des 1 sur la diagonale. On les note I_n où n est un entier correspondant à la taille de la matrice. La matrice $D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ par :

```
D = np.eye(4)
```

1. Définir les matrices suivantes.
   $M_1 = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 \end{pmatrix}$, $M_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$,
   $M_3 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $M_4 = \begin{pmatrix} 1 & 2 & 4 \\ -1 & 2 & 3 \\ 1 & 8 & 9 \end{pmatrix}$.

   L'attribut `shape` donne la taille d'une matrice : nombre de lignes, nombre de colonnes. Ainsi l'instruction

   ```
   A.shape
   ```

   Retourne

   ```
    (2, 3)
   ```

   On peut redimensionner une matrice, sans modifier ses termes, à l'aide de la méthode `reshape`.

   ```
   A.reshape((3, 2))
   ```

   Transforme la matrice A en $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$.

   Enfin la fonction `concatenate` permet de créer des matrices par blocs.

   Par exemple, si vous testez les instructions :

   ```
   A = np.ones((2,3))
   B = np.zeros((2,3))
    np.concatenate((A,B), axis=0)
    np.concatenate((A,B),axis=1)
   ```

   Vous devez obtenir :

   ```
   array([[ 1., 1., 1.],
   [ 1., 1., 1.],
   [ 0., 0., 0.],
   [ 0., 0., 0.]])
   ```

   Et

   ```
   array([[ 1., 1., 1., 0., 0., 0.],
   [ 1., 1., 1., 0., 0., 0.]])
   ```

L'accès à un terme de la matrice `A` se fait à l'aide de l'opération d'indexage `A[i, j]` où `i` désigne la ligne et `j` la colonne.

Attention, les indices commencent à zéro !

À l'aide d'intervalles, on peut également récupérer une partie d'une matrice : ligne, colonne, sous-matrice.

```
In : A[1, 0] # terme de la deuxième ligne, première colonne
Out : 3

In : A[0, :] # première ligne sous forme de tableau à 1 dimension
Out :
array([1, 2])

In : A[0, :].shape
Out : (2,)

In : A[0:1, :] # première ligne sous forme de matrice ligne
Out :
array([[1, 2]])

In : A[0:1, :].shape
Out : (1, 2)

In : A[ :, 1] # deuxième colonne sous forme de tableau à 1 dimension
Out : array([2, 4, 6])

In : A[ :, 1 :2] # deuxième colonne sous forme de matrice colonne
Out :
array([[2],
[4],
[6]])

In : A[1:3, 0 :2] # sous-matrice lignes 2 et 3, colonnes 1 et 2
Out :
array([[3, 4],
[5, 6]])
```

2. Définir la matrice $A = \begin{pmatrix} 2 & 2.3 & 2.4 \\ 9 & -8 & 9.81 \\ 22 & 1 & 0.145 \end{pmatrix}$
   (a) Afficher ses deuxièmes et troisièmes lignes.
   (b) Afficher le bloc composé des lignes 2 et 3 et des colonnes 2 et 3.
3. Construire les matrices suivantes à partir des matrices M_1, M_2, M_3, M_4 définies précédemment :
   $M_5 = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$ à partir de M_1.
   $M_6 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$ à partir M_1 et M_2.
   $M_7 = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 1 & 1 & 1. \end{pmatrix}$ à partir de M_1 et M_2.
   $M_8 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$ à partir M_3.
   $M_9 = \begin{pmatrix} 1 & 2 & 4 & 1 \\ -1 & 2 & 3 & 7 \\ 1 & 8 & 9 & 4 \end{pmatrix}$ à partir M_4.
   $M_{10} = \begin{pmatrix} 1 & 2 & 4 & 1 \\ -1 & 2 & 3 & 2 \\ 1 & 8 & 9 & 3 \end{pmatrix}$ à partir de M_3 et M_4.
   $M_{11} = \begin{pmatrix} 1 & 2 & 4 \\ -1 & 2 & 3 \\ 1 & 8 & 9 \\ 1 & 2 & 3 \end{pmatrix}$ à partir de M_3 et M_4.
4. Effectuer les modifications suivantes (en un minimum de commandes) :
   - M_{12} : Ajouter 2 au début de M_2.
   - M_{13} : Ajouter 8 en bas de M_3 et 9 en haut de M_3.
   - M_{14} : Supprimer la 2ème ligne de M_4. *On pourra utiliser l'instruction* `np.delete()`
   - M_{15} : Supprimer la 1ère et la 3ème cases de M_1.
   - M_{16} : Mettre 8 dans la 1ère et la 3ème cases de M_1.
   - M_{17} : Supprimer la 2ème colonne de M_4.