Matrices, tableaux et bibliothèque numpy

Le but de ce TP est de manipuler un nouveau type de variables (les tableaux) très pratique pour travailler avec des matrices.

On appelle matrice M de dimension $n \times p$ un tableau de nombres réels écrit sous la forme :

$$M = \left.\underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{pmatrix}}_{p\ colonnes}\right\} n \text{ lignes}$$

1. Représentation d'une matrice en python

1.1. Bibliothèque Numpy

Les bibliothèques sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi modules). Ce sont des « boîtes à outils » qui sont très utiles. À titre d'exemple, on a déjà utilisé la bibliothèque random

La bibliothèque n**umpy** permet d'effectuer des calculs sur des vecteurs ou des matrices, élément par élément, via un nouveau type d'objet appelé **array**.

Pour utiliser la bibliothèque n**umpy**, on la charge avec la commande : **import numpy**

```
In [6]: import numpy as np
```

remarque **as np** est une instruction qui permet de définir un nom raccourci pour n**umpy**

1.2. Définir un tableau : objets de type array

Les objets de type **array** correspondent à des tableaux à une ou plusieurs dimensions et permettent d'effectuer du calcul vectoriel.
La fonction **array()** convertit une liste en un objet de type **array**.

On saisit dans le shell :

```
In [13]: a=[1,2,3]

In [14]: b=np.array(a)

In [15]: b
```

Le contenu de la variable b est :

a)

À savoir
Un objet **array** ne contient que des données d'un **type identique.**
Il est possible de créer un objet **array** à partir d'une liste contenant des entiers et des chaînes de caractères, mais dans ce cas, toutes les valeurs seront comprises par n**umpy** comme des chaînes de caractères.
De même, il est possible de créer un objet **array** à partir d'une liste constituée d'entiers et de floats, mais toutes les valeurs seront alors comprises par n**umpy** comme des floats.

1.3. Définir une matrice : objet array à deux dimensions

Un objet **array** à **une dimension** permet de définir **un vecteur**.
Un objet **array** à **deux dimensions** permet de définir **une matrice**.

Pour construire des objets **array** à deux dimensions, il suffit de passer en argument une liste de listes à la fonction **array()** :

- On saisit :

```
In [21]: w=np.array([[1,2],[3,4],[5,6]])
```

  Le contenu de la variable w :

- Saisir une instruction créant un objet **array** représentant la matrice identité 3×3

1.4. Indices

Pour récupérer un ou plusieurs élément(s) d'un objet **array**, vous pouvez utiliser les indices ou les tranches, de la même manière qu'avec les listes.

- On donne :

```
In [24]: a=list(range(0,10))
In [25]: b=np.array(a)
In [26]: 
```

Que saisir en ligne In[26] pour récupérer :

- le 6ème élément de la variable b ?
- les 3 derniers ?
- les éléments de rang pair ?

- On donne :

```
In [29]: w=np.array([[1,2],[3,4],[5,6]])

In [30]: w
Out[30]:
array([[1, 2],
       [3, 4],
       [5, 6]])

In [31]: w[1]
```

Prévoir ce que retourne l'instruction w[1].

- On donne :

```
In [35]: w[1][0]
Out[35]: 3

In [36]: w[1][3]
---------------------------------------------------
-----------------------
IndexError                               Traceback
(most recent call last)
<ipython-input-36-a049def5e667> in <module>
----> 1 w[1][3]

IndexError: index 3 is out of bounds for axis 0 with
size 2

In [37]: |
```

  - Commenter la ligne Out[35] et l'erreur.
  - Comment extraire l'élément placé sur la i[ème] ligne et la j[ème] colonne d'un objet de type **`array`** ?

1.5. Copie d'arrays

- On donne

```
In [40]: b
Out[40]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [41]: c=b

In [42]: c[2]=25

In [43]: b
```

Quel est le contenu de la variable b ?

Afin d'éviter le problème, vous pouvez utiliser la fonction **np.array()** qui crée une nouvelle copie distincte de l'**array** initial.

- On donne :

```
In [44]: w=np.array([[1,2],[3,4],[5,6]])

In [45]: w
Out[45]:
array([[1, 2],
       [3, 4],
       [5, 6]])

In [46]: v=np.array(w)

In [47]: v[1][1]=10
```

Que contiennent les variables w et v ?

2. Premiers programmes

2.1. Transposition d'une matrice

a) La transposée d'une matrice carrée M de dimension $n \times n$: $M = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$

est la matrice notée ${}^tM = \begin{pmatrix} a_{1,1} & a_{2,1} & \cdots & a_{n,1} \\ a_{1,2} & a_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{n,n} \end{pmatrix}$.

Soit la matrice carrée $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$. Quelle est sa transposée ?

b) Soit la carrée matrice M de dimension $n \times n$: $M = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$

Quelle est sa transposée ?

c) On donne les méthodes suivantes :

- **.shape** : renvoie les dimensions d'un objet **array** sous forme (n,p)
  (n : nombre de lignes ; p : nombres de colonnes)
- **.reshape()** : renvoie un nouvel objet **array** avec les dimensions spécifiées

exemple

```
In [140]: v
Out[140]:
array([[1, 3],
       [3, 3],
       [3, 6]])

In [141]: v.shape
Out[141]: (3, 2)

In [142]: v.reshape(2,3)
Out[142]:
array([[1, 3, 3],
       [3, 3, 6]])
```

Remarque ligne Out[141] : on peut extraire l'entier 3 grâce à l'instruction v.shape[0]

```
In [144]: v.shape
Out[144]: (3, 2)

In [145]: v.shape[0]
Out[145]: 3
```

- À l'aide des méthodes ci-dessus, proposer une fonction **transpose** qui prend pour argument une matrice carrée de dimension 3×3 et qui renvoie sa transposée.
- Généraliser à une matrice carrée de dimension $n \times n$
- Généraliser à une matrice de dimension $n \times p$

2.2. Produit et puissance de matrices

a) Soient les matrices carrées suivantes : $A=\begin{pmatrix}-1 & 2 & 3\\ 4 & 0 & 6\\ -3 & 2 & 1\end{pmatrix}$ et $B=\begin{pmatrix}1 & 2 & 0\\ 1 & 3 & -3\\ 0 & -2 & -1\end{pmatrix}$

Effectuer à la main le produit AB.

b) Ci-dessous, compléter la ligne 10 du script suivant de telle façon que la fonction **mult** qui prend en argument deux matrices carrées de même dimension A et B retourne la matrice AB.

```
def mult(A,B):

    dim=A.shape[0]

    produit=np.zeros((dim,dim))

    for i in range(dim):
        for j in range(dim):
            for k in range(dim):
                produit[i][j]+=.......................

    return produit
```

À savoir

np.zeros((n,p)) renvoie un objet **array** à n lignes et p colonnes constitué de « 0 ».

Exemple

```
In [40]: P=np.zeros((2,3))

In [41]: P
Out[41]:
array([[0., 0., 0.],
       [0., 0., 0.]])
```

c) Coder et vérifier le script précédent à l'aide des résultats du a).

d) Proposer un script définissant la fonction puissance qui prend en paramètre une matrice carrée A ainsi qu'un nombre entier $n>1$ et qui renvoie la matrice A^n.

2.3. Matrices nilpotentes

On dit qu'une matrice carrée A de dimension $n \times n$ est nilpotente s'il existe un entier naturel $p \leqslant n$ tel que la matrice A^p soit nulle.

a) Quel est le rôle du script suivant ?

```
def c(A,B):
    dim=A.shape[0]
    out = True
    for i in range(dim):
        for j in range(dim):
            if A[i][j]!=B[i][j]:
                out = False
    return out
```

b) En utilisant la fonction **puissance** écrite à l'exercice précédent et la fonction **c**, proposer une fonction **nilpotente** permettant de tester si une matrice carrée A de dimension $n \times n$ est nilpotente.
Cette fonction prendra en argument une matrice carrée $n \times n$ et retournera True si la matrice est nilpotente et False sinon.