TP9

Matrices

La bibliothèque numpy permet de créer des tableaux (array) à une ligne, nous l'avons déjà vu, mais également des tableaux à plusieurs lignes et donc des matrices.

In [1]:
```
import numpy as np
```

Pour créer la matrice

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

on procède ainsi

In [2]:
```
A = np.array([[1,2,3], [4,5,6]]) # attention, deux niveaux de crochets
print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

Exercice 1

À l'aide du document de cours listant des commandes sur les matrices, répondre aux questions suivantes.

1. Définir la matrice $B = \begin{pmatrix} 0 & -1 \\ 4 & 7 \\ -8 & 2 \end{pmatrix}$.

In [3]:
```
B = np.array([[0,-1], [4,7], [-8,2]])
```

2. Afficher la transposée de B.

In [4]:
```
np.transpose(B)
```

Out[4]:
```
array([[ 0,  4, -8],
       [-1,  7,  2]])
```

3. Créer la matrice identité de taille 3, on l'appelera I.

In [5]:
```
I = np.eye(3)
print(I)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

4. Calculer IB avec in produit matriciel. Écrire un booléen qui vérifie que $IB = B$.

In [6]:
```
np.dot(I,B)
```

Out[6]:
```
array([[ 0., -1.],
       [ 4.,  7.],
       [-8.,  2.]])
```

In [7]:
```
np.dot(I,B) == B
```

Out[7]:
```
array([[ True,  True],
       [ True,  True],
       [ True,  True]])
```

5. Calculer AB et BA.

In [8]:
```
np.dot(A,B)
```

Out[8]:
```
array([[-16,  19],
       [-28,  43]])
```

In [9]:
```
np.dot(B,A)
```

Out[9]:
```
array([[ -4,  -5,  -6],
       [ 32,  43,  54],
       [  0,  -6, -12]])
```

Exercice 2

1. Définir $M = \begin{pmatrix} 4 & 2 & 0 \\ 1 & 0 & 1 \\ -1 & 2 & 1 \end{pmatrix}$ et $N = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ (pour N, on utilisera une commande efficace).

In [10]:
```
M = np.array([[4,2,0], [1,0,1], [-1,2,1]])
print(M)
```

```
[[ 4  2  0]
 [ 1  0  1]
 [-1  2  1]]
```

In [11]:
```
N = np.ones((3,3))
print(N)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

2. Calculer $3M - 2N$.

In [12]:
```
3*M-2*N
```

Out[12]:
```
array([[10.,  4., -2.],
       [ 1., -2.,  1.],
       [-5.,  4.,  1.]])
```

3. Calculer MN.

In [13]:
```
np.dot(M,N)
```

Out[13]:
```
array([[6., 6., 6.],
       [2., 2., 2.],
       [2., 2., 2.]])
```

4. Que fait l'instruction `M*N` ?

In [14]:
```
M*N # produit terme à terme
```

Out[14]:
```
array([[ 4.,  2.,  0.],
       [ 1.,  0.,  1.],
       [-1.,  2.,  1.]])
```

Exercice 3 - Puissances.

On reprend les matrices précédentes.

1. Calculer M^2 et M^3 avec np.dot.

In [15]:
```
np.dot(M,M)
```

Out[15]:
```
array([[18,  8,  2],
       [ 3,  4,  1],
       [-3,  0,  3]])
```

In [16]:
```
np.dot(M, np.dot(M,M))
```

Out[16]:
```
array([[ 78,  40,  10],
       [ 15,   8,   5],
       [-15,   0,   3]])
```

2. Que fait l'instruction `M**3` ?

In [17]:
```
M**3 # puissance terme à terme
```

Out[17]:
```
array([[64,  8,  0],
       [ 1,  0,  1],
       [-1,  8,  1]])
```

3. En utilisant la bibliothèque numpy.linalg (voir documentation), recalculer M^3.

In [18]:
```
import numpy.linalg as al

al.matrix_power(M,3)
```

Out[18]:
```
array([[ 78,  40,  10],
       [ 15,   8,   5],
       [-15,   0,   3]])
```

4. Écrire une fonction puisance(A,n) qui calcule A^n $(n \geqslant 1)$ pour une matrice A donnée, en utilisant uniquement np.dot.
5. Calculer M^{10} et vérifier avec al.matrix_power.

In [19]:
```
def puissance(A,n):
    P = A
    for k in range(n-1):
        P = np.dot(A,P)
    return P
```

In [20]:
```
puissance(M,10)
```

Out[20]:
```
array([[2387718, 1263344,  374450],
       [ 444447,  235480,   69997],
       [-444447, -234456,  -68973]])
```

In [21]:
```
al.matrix_power(M,10) == puissance(M,10)
```

Out[21]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

Exercice 4 - Inverse

On reprend les matrices précédentes.

1. Donner l'inverse de la matrice M (on admet qu'elle est inversible).

In [22]:
```
al.inv(M)
```

Out[22]:
```
array([[ 0.16666667,  0.16666667, -0.16666667],
       [ 0.16666667, -0.33333333,  0.33333333],
       [-0.16666667,  0.83333333,  0.16666667]])
```

2. Que vaut $M^{-1}M$?

In [23]:
```
np.dot(al.inv(M), M)
```

Out[23]:
```
array([[ 1.00000000e+00,  0.00000000e+00,  2.77555756e-17],
       [ 5.55111512e-17,  1.00000000e+00, -5.55111512e-17],
       [-5.55111512e-17,  0.00000000e+00,  1.00000000e+00]])
```

Il y a des erreurs dûes à des approximations.

Exercice 5 - Boucles imbriquées

1. Afficher le premier coefficient de la matrice M.

In [24]:
```
M[0,0]
```

Out[24]:
```
4
```

2. Afficher la deuxième ligne de M.

In [25]:
```
M[1,:]
```

Out[25]:
```
array([1, 0, 1])
```

3. Afficher la dernière colonne de M.

In [26]:
```
M[:,-1]
```

Out[26]:
```
array([0, 1, 1])
```

4. On souhaite définir la matrice $C = (c_{i,j})_{0 \leq i \leq 2, 0 \leq j \leq 3}$ telle que $c_{i,j} = i - j$ (on utilise ici la numérotation python). Compléter le programme.

In [27]:
```
# On définit C comme la matrice nulle de taille correcte
C = np.zeros((3,4))

# On va ensuite modifier chacun des cij avec deux boucles for imbriquées
for i in range(3):
    for j in range(4):
        C[i,j] = i-j
print(C)
```

```
[[ 0. -1. -2. -3.]
 [ 1.  0. -1. -2.]
 [ 2.  1.  0. -1.]]
```

5. Définir la matrice $D = (d_{i,j})_{1 \leq i \leq 4, 1 \leq j \leq 3}$ telle que $d_{i,j} = ij$ (attention, ici la numérotation commence à 1, comme en maths).

In [28]:
```
# On définit D comme la matrice nulle de taille correcte
D = np.zeros((4,3))

# On va ensuite modifier chacun des dij avec deux boucles for imbriquées
for i in range(4):
    for j in range(3):
        D[i,j] = (i+1)*(j+1)
print(D)
```

```
[[ 1.  2.  3.]
 [ 2.  4.  6.]
 [ 3.  6.  9.]
 [ 4.  8. 12.]]
```

6. Écrire une fonction qui calcule la somme des éléments d'une matrice (sans utiliser np.sum). Pour la vérification des résultats, on pourra utuliser np.sum.

In [29]:
```
def somme(A):
    (n,p) = np.shape(A)
    S = 0
    for i in range(n):
        for j in range(p):
            S = S + A[i,j]
    return S
```

In [30]:
```
somme(M)
```

Out[30]:
```
10
```

In [31]:
```
np.sum(M)
```

Out[31]:
```
10
```

In [ ]: