

Pour travailler avec des matrices en *Python*, on peut utiliser le module `numpy`, qu'on importe :

```
import numpy as np
```

I Écriture d'une matrice

1 Définition

Pour définir une matrice A , on crée un tableau `numpy` en donnant la liste des listes constituées par les lignes de la matrice A .

Exemple : Pour définir la matrice $A = \begin{pmatrix} 1 & 0 & -2 \\ -3 & 1 & 2 \end{pmatrix}$, constituée des listes $[1, 0, -2]$ et $[-3, 1, 2]$, on écrit :

```
A = np.array( [ [1, 0, -2] , [-3, 1, 2] ] )
```

Premières fonctions :

* `np.size(A)` : renvoie le nombre de coefficients de la matrice A .

```
>>> np.size(A) >>> 6
```

* `np.shape(A)` : renvoie un tuple correspondant à la taille de la matrice A .

```
>>> np.shape(A) >>> (2,3)
```

Remarques : Ainsi définie, une matrice est un objet informatique de type `numpy.ndarray`.

Si tous les coefficients sont des entiers, ils auront le type `numpy.int32` (format entier). Si au moins un coefficient est un flottant, alors tous les coefficients seront de type `numpy.float64` (format flottant).

2 Matrices particulières

On peut créer directement certaines matrices particulières grâce aux fonctions de `numpy`.

* `np.zeros((n,p))` : matrice nulle de taille (n, p) , de format flottant.

* `np.eye(n)` : matrice identité de taille n , de format flottant.

* `np.ones((n,p))` : matrice de taille (n, p) où les coefficients valent 1., de format flottant.

* `np.diag(L)` : matrice diagonale dont les coefficients sont donnés par les éléments de la liste L .

II Manipulation des matrices

1 Indexation et extraction

Remarque importante :

comme pour les listes, la numérotation des lignes et des colonnes des matrices commence à 0.

* `A[i,j]` : renvoie le coefficient situé sur la ligne i et la colonne j .

```
>>> A[0,2] >>> -2
```

* `A[i]` : renvoie la matrice-ligne d'indice i .

```
>>> A[1] >>> array([-3, 1, 2])
```

* `A[:,j]` : renvoie la transposée de la matrice-colonne d'indice j .

```
>>> A[:,0] >>> array([1, -3])
```

* `A[:i]` : renvoie la matrice formée par les i premières lignes de A .

* `A[i:]` : renvoie la matrice obtenue en supprimant les i premières lignes de A .

2 Modification des éléments dans une matrice

Comme les listes, une matrice est modifiable. On peut choisir de modifier :

* un coefficient : `>>> A[1,1]=0`

* une ligne : `>>> A[1]=[0,0,0]`

* une colonne : `>>> A[:,1]=[1,2]`

3 Opérations usuelles

- * addition : $C = A+B$
- * multiplication par une constante : $D = 2*A$
- * multiplication terme à terme (matrices de mêmes tailles) : $M = A*B$
- * produit matriciel (attention aux tailles) : $P = \text{np.dot}(A,B)$
- * transposition : $T = A.\text{transpose}()$
- * faire une copie indépendante : $E = A.\text{copy}()$
- * obtenir le rang d'une matrice : $r = \text{np.linalg.matrix_rank}(A)$
- * obtenir l'inverse d'une matrice inversible : $B = \text{np.linalg.inv}(A)$
- * obtenir la $n^{\text{ième}}$ puissance d'une matrice carrée : $P = \text{np.linalg.matrix_power}(A,n)$
- * obtenir la solution de $AX = B$ lorsque A est une matrice inversible : $X = \text{np.linalg.solve}(A,B)$

4 Autres commandes

Les matrices-numpy sont des objets itérables.

La commande `in` teste si un nombre ou une liste est contenu dans une matrice.

On pourra écrire `for a in A` : pour effectuer une boucle sur les éléments d'une matrice A (*attention* : une matrice numpy contient des matrice-lignes, et non des coefficients).

III Travail à réaliser

Exercice 1 : Créer les matrices : $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ $B = \begin{pmatrix} 4 & 4 & 4 & 4 & 2 \\ 4 & 4 & 4 & 4 & 4 \\ 4 & 3 & 4 & 4 & 4 \end{pmatrix}$ $C = \begin{pmatrix} -1 & 0 & 3 & 0 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

Exercice 2 : Écrire une fonction `somme(A,B)` qui prend en argument deux matrices A et B et qui renvoie la matrice somme C , lorsque les tailles le permettent, sans utiliser les opérations sur les matrices mais uniquement des boucles.

Exercice 3 : Écrire une fonction `produit(A,B)` qui prend en argument deux matrices A et B et qui renvoie la matrice produit C , lorsque les tailles le permettent, sans utiliser les opérations sur les matrices mais uniquement des boucles (sans utiliser la fonction `dot`).

Exercice 4 : Écrire une fonction `puissance(A,n)` qui prend en argument une matrice carrée A et un entier naturel n et qui renvoie la matrice A^n (on pourra se servir de la fonction `produit` ou de la fonction `dot`).

Exercice 5 : Écrire une fonction `transpose(A)` qui prend en argument une matrice A et qui renvoie la matrice transposée de A .

Exercice 6 : Écrire une fonction `remplace(A,x,y)` qui prend en arguments une matrice A et deux nombres x et y et qui remplace dans A toutes les occurrences de x par y .

Exercice 7 : Écrire une fonction `symetrie(A)` qui teste si une matrice est symétrique.

Exercice 8 : Écrire un programme :

- * qui demande à l'utilisateur les nombres n et p correspondant à la taille d'une matrice,
- * qui demande à l'utilisateur de saisir un à un les coefficients de cette matrice.

Exercice 9 :

1. Écrire une fonction `permuterLigne(A,i,j)` qui prend en arguments une matrice A et deux entiers naturels i et j et qui permute les lignes i et j de A lorsque cela est possible.
2. Écrire une fonction `multiplierLigne(A,i,m)` qui prend en arguments une matrice A , un entier naturel i et un flottant m et qui multiplie par m les coefficients de la ligne i dans la matrice A .
3. Écrire une fonction `combinaisonLineaire(A,i,j,m)` qui prend en arguments une matrice A , deux entiers naturels i et j , et un flottant m , et qui renvoie la matrice A dans laquelle la ligne L_i a été remplacée par $L_i + mL_j$.

Exercice 10 * : Dédurre de l'exercice 9 une fonction `Gauss(A)` qui échelonne une matrice.