TP # 12

Matrices en Python

Partie I : définition et manipulation

Rappels sur les vecteurs

En Python, on appelle "**vecteur**" un **tableau à une seule ligne** : `[x1,x2,...,xn]`.

Créer un vecteur "à la main"

Après avoir importé `numpy as np` : `V=np.array( [x1, ..., xn] )`

Quelques vecteurs particuliers

- `np.zeros(n)` crée un vecteur contenant n zéros.
- `np.ones(n)` crée un vecteur contenant n uns.

Définir une matrice

L'instruction `np.array` permet aussi de créer des **matrices**, c'est à dire des **tableaux à plusieurs lignes**. Pour cela, on lui donne, entre crochets, la liste des lignes de la matrice voulue (attention donc aux "doubles crochets" !)

Créer une matrice "à la main"

Après avoir importé `numpy as np` :

`A=np.array( [ [` $a_{1,1}, a_{1,2}, \dots, a_{1,p}$ `] , ..., [` $a_{n,1}, a_{n,2}, \dots, a_{n,p}$ `] ] )`

Exercice 1

1. Quelle matrice est créée par les instructions suivantes ? $A =$

```
>>> import numpy as np
>>> A = np.array([[4,5],[0,1],[3,2]])     >>> print(A)
```

2. Quelle instruction permet de créer la matrice $B = \begin{pmatrix} 3 & -1 & 2 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix}$?

Les instructions `np.zeros` et `np.ones` fonctionnent aussi pour des matrices : on doit alors leur donner la "taille" `(n,p)` (ou `[n,p]`) de la matrice (=nombre de lignes / colonnes).

Créer des matrices particulières

Après avoir importé `numpy as np` :

- `np.zeros( (n,p) )` crée une matrice de taille $n \times p$ contenant des 0.
- `np.ones( (n,p) )` crée une matrice de taille $n \times p$ contenant des 1.
- `np.eye(n)` crée la matrice identité I_n.

Exercice 2

Quelles matrices sont crées par les instructions suivantes ?

`np.zeros((2,3))` : ....

`np.ones((3,2))` : ....

`np.eye(3)` : ....

Somme de matrices, multiplication par une constante

Soient `A`$= (a_{i,j})$ et `B`$= (b_{i,j})$ des matrices de même taille. Soit `x` un nombre réel.

- `A + x` donne la matrice de coefficients $(a_{i,j} + x)$
- `x*A` donne la matrice de coefficients $(xa_{i,j})$ (c'est à dire xA)
- `A + B` donne la matrice de coefficients $(a_{i,j} + b_{i,j})$ (c'est à dire $A + B$)

On peut alors utiliser `np.ones`, `np.eye` pour créer rapidement certaines matrices.

Exercice 3

Constuire en une ligne, sans utiliser "`np.array`", les matrices suivantes :

1. $A = \begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$: `A =` ....

2. $B = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$: `B =` ....

3. $C = \begin{pmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{pmatrix}$: `C =` ....

En fait, comme pour les vecteurs, on peut effectuer des opérations diverses (`*` , `**` , `/` ...)

Autres opérations "coefficient par coefficient"

Soient `A`$= (a_{i,j})$ et `B`$= (b_{i,j})$ des matrices de même taille. Soit `k` un nombre réel.

- `A*B` donne la matrice de coefficients $(a_{i,j} \times b_{i,j})$

  (et non pas la matrice produit AB !)

- `A**k` donne la matrice de coefficients $(a_{i,j}^k)$

  (et non pas la puissance de matrice A^k !)

- `A/B` donne la matrice de coefficients $(\frac{a_{i,j}}{b_{i,j}})$
- Si f est une fonction, `f(A)` donne la matrice de coefficients $(f(a_{i,j}))$.

Exemple

Si, en Python, `A` = $\begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & 0 \end{pmatrix}$ et `B` = $\begin{pmatrix} 2 & -1 & 0 \\ 3 & 3 & 3 \end{pmatrix}$,

l'instruction `A*B` renvoie la matrice : $\begin{pmatrix} 2 & -2 & 0 \\ 0 & -3 & 0 \end{pmatrix}$.

On verra prochainement comment effectuer les "vrais" produits matriciels AB et A^k.

Extraction et modification de coefficients

Accéder à un coefficient / une ligne / une colonne

Si `A`$= (a_{i,j})$ est une matrice :

- `A[i,j]` est le coefficient $a_{i+1,j+1}$.
- `A[i,:]` est un vecteur (ligne) contenant la $(i+1)$-ème ligne de `A`.
- `A[:,j]` est un vecteur (ligne !) contenant la $(j+1)$-ème colonne de `A`.

On peut ainsi afficher ou même modifier les coefficients/lignes/colonnes de `A`.

Attention !

Comme d'habitude en Python, l'indexation des lignes/colonnes démarre à 0 !

Exercice 4

1. On définit en Python la matrice `A` = $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.

Que renvoient les instructions suivantes ?

`A[0,0]` : .... `A[0,1]` : .... `A[1,2]` : .... `A[2,0]` : ....

`print(A[2,:])` : ....

`print(A[:,0])` : ....

2. Deviner la matrice obtenue à la fin des instructions : $B =$

```
>>> B = np.zeros([2,3]) >>> B[0,0] = 1 >>> B[1,1] = -1 >>> B[0,2] = 1
```

3. Deviner la matrice obtenue à la fin des instructions : $C =$

```
>>> C = np.eye(3)  >>> C[1,:] = [1,2,3]
```

Taille d'une matrice

Si `A` est une matrice, l'instruction `a,b = np.shape(A)`

donne à `a` le nombre de lignes de `A`, à `b` le nombre de colonnes de `A`.

Exercice 5

?Ecrire une fonction `Tr` qui prend en entrée une matrice carrée $A \in \mathcal{M}_n(\mathbb{R})$ et renvoie la somme de ses coefficients diagonaux. On appelle ceci la *trace* de A.
(On commencera pas récupérer la valeur de "n"...)

```
import numpy as np

def Tr(A) :
```

Pour s'exercer...

Exercice 6
Triangle de Pascal.

On rappelle la formule de Pascal : pour tous $n, k \in \mathbb{N}$, $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$.

De plus, pour tout $n \in \mathbb{N}$, $\binom{n}{0} =$ ....

Ecrire une fonction `triangle_pascal` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie la matrice A contenant le triangle de Pascal jusqu'à la ligne n. Autrement dit :

$$\text{Pour tout } (i,j) \in [\![0,n]\!]^2, \quad \texttt{A[i,j]} = \binom{i}{j}$$

Pour ce faire, on pourra :

$\boxed{1}$ Définir une matrice `A` de la bonne taille, contenant uniquement des 0.

$\boxed{2}$ Remplir correctement la 0-ième colonne de A.
La 0=ième ligne est alors correctement remplie.

$\boxed{2}$ A partir de la ligne 1, utiliser la formule de Pascal pour définir `A[i,j]` à l'aide des coefficients de la ligne précédente. On écrira deux boucles `for` imbriquées.

Afficher ainsi `triangle_pascal(10)`.

Exercice 7
Pivot de Gauss.

1. Compléter le programme suivant pour que la fonction `echange(A,i,j)` renvoie la matrice Aà ? laquelle on a échangé les lignes i et j.

Par exemple, `echange(np.eye(3),1,2)` doit renvoyer : $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

```
import numpy as np
def echange(A,i,j) :

        L = A[i-1,:].copy()

        A[i-1, :] = ..............

        A[j-1,:] = ...............

        return(A)
```

2. Compléter le programme suivant pour que la fonction `operation(A,i,j,b)` renvoie la matrice A ? laquelle on a effectué l'opération $L_i \leftarrow L_i + bL_j$.

Par exemple, `operation(np.eye(3),2,1,2)` doit renvoyer : $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

On définira cette fonction ? la suite de la précédente.

```
def operation(A,i,j,b) :

        ......................................

        return(A)
```

3. Définir dans la console la matrice $A = \begin{pmatrix} 3 & 1 & -1 \\ 2 & 1 & -1 \\ 1 & 1 & 2 \end{pmatrix}$.

En utilisant des opérations `A=echange(A,i,j)` et `A=operation(A,i,j,b)` successivement, appliquer l'algorithme du pivot de Gauss pour transformer A en une matrice triangulaire supérieure. On affichera A entre deux instructions pour contr ?ler le résultat de l'opération effectuée.

Matrice obtenue à la fin du pivot de Gauss : ....

Pourquoi peut-on en déduire que A est inversible ?