TP # 13

Matrices en Python

Partie II : Calcul matriciel

Rappels sur les matrices

Créer une matrice "à la main"

Après avoir importé `numpy as np` :

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

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.

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`.

Remarque 1

Les opérations `A * B` et `A**k` se font "coefficient par coefficient" et ne correspondent pas du tout aux produits matriciels AB et A^k !

Opérations matricielles : produit, transposition, puissance, inverse

Produit matriciel et transposition

Après avoir importé `numpy as np` :

- `np.dot(A,B)` calcule le "vrai" produit matriciel AB (quand il a un sens !)
- `np.transpose(A)` renvoie la matrice transposée ${}^tA$.

Exercice 1

Définir dans la console :

```
>>> A = np.array([[1,1,2],[1,0,1],[0,0,1]])
>>> B = np.array([[0,0,-1],[1,1,0],[0,0,0]])
>>> X = np.array([[1],[1],[-1]])
```

Déterminer à l'aide de Python les matrices suivantes :

$A =$, $B =$, $X =$

$AB =$, $AX =$, ${}^tXB =$

Pour les puissances matricielles et le calcul de l'inverse, on a besoin d'une nouvelle bibliothèque d'"algèbre linéaire" (abrégé `al`) :

Importation de la bibliothèque `numpy.linalg`

Importation recommandée : `import numpy.linalg as al`

Puissances d'une matrice, rang, inverse résolution de système

- `al.matrix_power(A,n)` calcule la "vraie" puissance d'une matrice carrée A^n.
- `al.matrix_rank(A)` calcule le rang d'une matrice A (cf cours plus tard...)
- `al.inv(A)` calcule l'inverse A^{-1} d'une matrice carrée inversible A.
- `X = al.solve(A,Y)` renvoie l'unique solution X du système linéaire $AX = Y$ lorsque A est une matrice carrée inversible. Autrement dit, cela calcule $X = A^{-1}Y$. Attention : les inconnues `X` et le second membre `Y` sont donnés ici en vecteurs lignes (et non en colonne) !

Exercice 2

On considère la matrice $A = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$.

1. A l'aide de Python, calculer rapidement les puissances :

$A =$ $A^2 =$ $A^3 =$ $A^4 =$

Par exemple, pour A^3 on tape dans la console : .....

2. Conjecturer, pour tout $n \in \mathbb{N}$, l'expression de $A^n =$

4. Prévoir ainsi la valeur de A^{10} et vérifier avec Python : $A^{10} =$

Exercice 3

1. Définir en Python les matrices $B = \begin{pmatrix} -1 & 0 & 0 \\ -8 & 0 & -8 \\ 9 & 0 & 8 \end{pmatrix}$ et $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix}$.

3. A l'aide de Python, calculer la matrice $D = P^{-1}BP$.

```
>>> D =
```

$D =$

4. Pour tout $n \in \mathbb{N}$, compléter l'expression de $D^n =$

Remarque : On montrerait ensuite par récurrence que $\forall n \in \mathbb{N},\ B^n = PD^nP^{-1}$ et on en déduirait alors l'expression de B^n

Exercice 4

On admet que le système suivant est de Cramer.

$$\begin{cases} x - y + z = 1 \\ x + z = -1 \\ x + y + 2z = 3 \end{cases}$$

1. Déterminer son unique solution à l'aide de `al.solve`.

```
>>> import numpy.linalg as al
>>> A =
>>> Y =
>>> print( al.solve(A,Y) )
```

L'unique solution est : $(x, y, z) =$

2. Comparer avec le résultat obtenu en calculant $X = A^{-1}\begin{pmatrix} 1 \\ -1 \\ 3 \end{pmatrix}$ avec Python :

```
>>> Z = np.array( [ [1],[-1],[3] ] )
>>> X =
```

On obtient : $X =$

Exercice 5

BONUS : On donne le programme suivant :

```
def truc(n):
    x=np.arange(1,n+1) ; y=np.ones(n)
    return np.dot(x,np.transpose(y))
```

Déterminer, mathématiquement, l'expression de `truc`(n) en fonction de n.
Prévoir ainsi la valeur de `truc(100)`.