

## Séance 3. Matrices et systèmes linéaires.

---

### 1 Matrices avec Numpy

Le type favori de Numpy est `array`. Il est similaire à une liste sauf que tous les éléments doivent avoir le même type (`int`, `float`, `bool`...). Si  $A$  est un `array` numpy on peut retrouver le type de ses éléments avec la commande `A.dtype`. On utilisera ce type `array` pour représenter des matrices même s'il existe un type `matrix` sous Numpy. En effet ce dernier offre moins de possibilités que le type `array`. Néanmoins on peut passer de l'un à l'autre avec les commandes `mat` et `array`. On supposera que numpy est importé sous la forme, `import numpy as np`.

#### 1.1 Construction

Pour représenter une matrice sous forme d'un `array` on peut faire : `A=np.array([[2,3,4],[7,6,5],[8,9,0]])`. On remarquera que la matrice est écrite ligne par ligne. Pour obtenir le coefficient de  $A$  de position  $(i,j)$  on utilise `A[i,j]` ou encore `A[i][j]`. Avec l'exemple précédent `A[0,2]` et `A[0][2]` donneront tous les deux 4 car `A[0]` donne `array([2,3,4])`.

Pour récupérer la taille d'une matrice on utilise `A.shape`, tandis que `len(A)` donnera le nombre de lignes de  $A$ .

On peut, comme pour les listes, utiliser le *slicing* pour extraire des sous-tableaux. Par exemple `A[0:2]` donnera `array([[2,3,4],[7,6,5]])`.

Il existe des commandes pour créer les matrices de formes particulières :

- `np.zeros(n)` donne un vecteur nul horizontal de taille  $n$  et `np.zeros((n,p))` donne une matrice nulle de taille  $(n,p)$ .
- `np.eye(n)` ou `np.identity(n)` donnent la matrice carrée identité de taille  $n$ .
- `np.ones(n)` et `np.ones((n,p))` donnent un vecteur et une matrice remplis de 1.
- `np.diag(v)` donne une matrice diagonale dont la diagonale est constituée des coordonnées du vecteur  $v$ . `np.diag(v,k)` avec  $k \in \mathbb{Z}$  est une matrice avec une diagonale décalée de  $k$  crans remplie avec les coordonnées de  $v$ .
- `np.random.rand(n)` et `np.random.rand(n,p)` permettent de créer un vecteur et une matrice à coefficients aléatoirement choisis de manière uniforme sur  $[0,1]$ .

#### Exercice 1.

1. Écrire une fonction qui prend en argument le nombre de lignes et de colonnes et qui crée une matrice aléatoire dans laquelle chaque coefficient vaut 1 ou 0 avec probabilité  $\frac{1}{2}$ .
2. Écrire une fonction qui prend en argument une matrice comme décrite dans la question précédente et qui renvoie le booléen `True` si cette matrice contient au moins un « carré » de 4 coefficients valant 1 et `False` sinon.
3. Écrire une fonction qui prend en argument une matrice comme décrite dans la première question et qui renvoie le booléen `True` si cette matrice contient plus de 1 que de 0.

#### 1.2 Opérations

Attention! les opérations `+`, `*`, `**` n'agissent sur les `array` que terme à terme. Contrairement aux objets de type `matrix` sur lesquels ces opérateurs agissent au sens matriciel usuel.

Pour faire des produits matriciels entre deux `array`  $A$  et  $B$  on utilise la commande de Numpy `np.dot(A,B)`. Python indiquera un message d'erreur si les tailles des matrices ne permettent pas ce produit.

Pour la transposition, on utilise `A.transpose()`.

Certaines opérations classiques d'algèbre linéaire se situent dans le sous module `linalg` de Numpy. Par exemple, le calcul de l'inverse d'une matrice  $A$  se fera avec `np.linalg.inv(A)` et le calcul de son déterminant par `np.linalg.det(A)`. On peut aussi calculer les puissances d'une matrice : `np.linalg.matrix_power(A,n)` donnera le résultat de  $A^n$ . Pour le rang, on utilisera `np.linalg.matrix_rank(A)`.

### Exercice 2.

1. Déterminer le rang de la matrice  $A = \begin{pmatrix} 1 & \frac{1}{4} & 1 \\ 1 & \frac{1}{3} & 2 \\ 0 & 1 & 12 \end{pmatrix}$ . Mathématiquement, est-elle inversible?
2. Demander à Numpy de l'inverser. Moralité?

**Exercice 3.** On définit trois suites réelles par  $u_0 = 1, v_0 = 2, w_0 = 3$  et

$$\forall n \in \mathbb{N}, \begin{cases} u_{n+1} = 2u_n + 3v_n - w_n \\ v_{n+1} = -u_n + v_n - w_n \\ w_{n+1} = u_n - 3v_n - 2w_n \end{cases}$$

Calculer  $u_{100}, v_{100}$  et  $w_{100}$ .

## 2 Résolution d'un système linéaire avec Numpy

On va se restreindre aux systèmes de Cramer qui admettent donc une unique solution.

Si  $A$  est la matrice du système et  $Y$  la matrice colonne des seconds membres, on résout le système  $AX = Y$  à l'aide de la commande Numpy : `np.linalg.solve(A,Y)`.

### Exercice 4.

1. Résoudre le système suivant : 
$$\begin{cases} 2x + 2y - 3z = 2 \\ -2x - y - 3z = -5 \\ 6x + 4y + 4z = 16 \end{cases}$$
2. Que dire des deux systèmes suivants :

$$\begin{cases} x + \frac{1}{4}y + z = 1 \\ x + \frac{1}{3}y + 2z = 1 \\ y + 12z = 0 \end{cases} \quad \text{et} \quad \begin{cases} 4x + y + 4z = 4 \\ 3x + y + 6z = 3 \\ y + 12z = 0 \end{cases}$$

Résolvez-les avec Numpy...commentez.

## 3 Algorithme du pivot de Gauss

On veut mettre en place l'algorithme du pivot de Gauss pour un système linéaire de Cramer de la forme  $AX = Y$ . Le but est d'échelonner le système et donc la matrice  $A$  puis de remonter le système triangulaire obtenu en faisant des substitutions.

On sait qu'on va devoir faire des opérations élémentaires sur les lignes de  $A$  et  $Y$  comme des échanges  $L_i \leftrightarrow L_j$  ou encore des transvections de la forme  $L_i \leftarrow L_i + xL_j$ . Commençons par coder ces opérations élémentaires.

### Exercice 5.

1. Écrire une fonction qui prend en argument une matrice  $A$  et des entiers  $i, j$  et qui échange les lignes  $L_i$  et  $L_j$  de  $A$ . **Attention au phénomène d'aliasing.**
2. Écrire une fonction qui prend en argument une matrice  $A$ , des entiers  $i, j$  et un réel  $x$  et qui réalise la transvection  $L_i \leftarrow L_i + xL_j$ .

Ensuite à chaque étape, on va devoir choisir un « pivot ». En théorie, il suffit de le choisir non nul. Mais comme tester la nullité d'un réel codé en type `float` est assez périlleux à cause des erreurs d'arrondis, on va choisir pour pivot le coefficient qui a la plus grande valeur absolue : ça s'appelle la méthode du « pivot partiel ».

### Exercice 6.

Écrire une fonction qui prend en argument une matrice  $A = (a_{pq})_{(p,q) \in \llbracket 1, n \rrbracket^2}$  et un indice  $i$  et qui renvoie un indice  $j \geq i$  tel que  $|a_{ji}| = \max_{k \geq i} |a_{ki}|$

Il reste ensuite à mettre en place l'algorithme du pivot de Gauss dans son entier. Rappelons-en les étapes. *On numérote les  $n$  lignes de  $A$  de 0 à  $n - 1$  pour être dans un esprit pythonesque.*

1. *Mise sous forme triangulaire :*

- Pour  $i$  de 0 à  $n-2$ , trouver  $j$  entre  $i$  et  $n-1$  tel que  $|a_{ji}|$  soit maximal.
- Échanger les lignes  $i$  et  $j$  de  $A$  et  $Y$ .
- Faire disparaître la  $i$ -ième variable des lignes  $k$  pour  $k$  allant  $i+1$  à  $n-1$  en effectuant les transvections  
$$L_k \leftarrow L_k - \frac{a_{ki}}{a_{ii}} L_i \text{ sur } A \text{ et } Y.$$

2. *Remontée :*

Le système est alors sous forme triangulaire. Il reste la remontée. Le résultat est mis dans une matrice colonne  $X$  dont les coordonnées doivent vérifier :

$$x_i = \frac{1}{a_{ii}} \left( y_i - \sum_{k=i+1}^{n-1} a_{ik} x_k \right)$$

**Exercice 7.** Mettez en oeuvre l'algorithme du pivot de Gauss et testez-le sur les systèmes déjà résolus.