Résolution numérique des systèmes linéaires

TABLEAUX NUMPY – PIVOT DE GAUSS

Exercice 1 : Manipulation de tableaux Numpy

import numpy as np

Question 1 : Créer un tableau *Numpy* A contenant :

A=np.array([[2,2,-3],[-2,-1,-3],[6,4,4]], dtype=float)

Question 2 : Faire afficher la taille, puis la forme du tableau A.

```
print("taille de A : ", A.size)
print("taille de A : ", A.shape)
```

Question 3 : Les coefficients de la matrice sont notés a_{ij}. Tester une condition (par exemple $a_{ij} > 1$) à l'aide d'un masque.

```
print(A>1)
```

Question 4 : Extraire une partie de A, par exemple les coefficients a_{ij} pour $(i, j) \in \{0, 1\} \times \{1, 2\}$. Déterminer la taille et la forme de cette nouvelle matrice B.

```
B = A[ :2, 1 :]
```

Question 5 : Extraire la colonne du milieu. Déterminer la taille et la forme de cette nouvelle matrice C.

```
C = A[ : , [1] ]
print("C=", C)
print("taille de C : ", C.size)
print("forme de C : ", C.shape)
```

Question 6 : Créer un tableau D de forme 4×6 ne contenant que des zéros. Placer un bloc valant A en haut à gauche, et un autre en bas à droite :

```
D=np.zeros((4,6))
D[ :3, :3]=A
D[1 :, 3 :]=A
print(D)
```

Question 7 : Afficher $E = 2.A$, $F = A \times A$ (on veut le produit matriciel).

```
E=2*A
print("E=", E)

F=np.dot(A, A)
print("F=", F)
```

Question 8 : A l'aide de *np.eye*, calculer G = A + 2 I_3.

```
I3=np.eye(3)
G=A+2*I3
print(G)
```

Question 9 : Avec une commande de *np.linalg* adaptée, calculer l'inverse de A, notée H.

```
H=np.linalg.inv(A)
```

Question 10 : Tester le comportement de « I = A » et de « J = *np.copy*(A) » quand on change une valeur de A.

Exercice 2 : Résolution de systèmes linéaires

Question 11 : Résoudre A.X = Y à la main puis avec Python pour

$$A = \begin{pmatrix} 2 & 2 & -3 \\ -2 & -1 & -3 \\ 6 & 4 & 4 \end{pmatrix} \text{ et } Y = \begin{pmatrix} -14 \\ 21 \\ 4 \end{pmatrix}$$

Méthode 1 :

```
X  = np.dot(np.linalg.inv(A), Y)
```

Méthode 2 :

```
X=np.linalg.solve(A,Y)
```

Exercice 3 : Pivot de Gauss

Question 12 : Ecrire un algorithme de résolution d'un système linéaire inversible (ou de Cramer) par la méthode de Gauss avec recherche partielle du pivot.

- La recherche d'un pivot :

On commence par écrire une fonction « def chercher_pivot (M, i) » qui prend en paramètre une matrice M et l'indice d'une ligne de la matrice M et renvoie un indice de ligne j>=i tel que | Mji | soit maximal.

- Les échanges de lignes :

On écrit maintenant une fonction « def echange_lignes(M, i, j) » qui permet d'échanger la ième et la jème ligne de la matrice M.

- Les transvections

On écrit une fonction « def transvection(M, i, j, mu) » qui permet d'appliquer une transvection de la j ème ligne à la i ème ligne avec le scalaire mu.
Lj = Lj + mu*Li

- Recoller les morceaux :

Grâce à tous ces fonctions, l'écriture du programme Python qui fait la résolution d'un système linéaire devient comme prévu un simple exercice de traduction. On commence par faire une copie de la matrice fournie en entrée puis on applique l'algorithme décrit en dessus.

On écrit une fonction « def resolution(A, Y)» qui prend en paramètre la matrice A et la matrice Y et retourne le vecteur X contenant la solution du système.

```
def chercher_pivot(M, i):
    pivot = i
    for k in range(i+1, M.shape[0]):
        if abs(M[k][i])>abs(M[pivot][i]):
            pivot=k
    return pivot

def echange_lignes(M, i, j) :
    M[i], M[j] = M[j].copy(), M[i].copy()

def transvection(M, i, j , mu):
    M[j]= M[j] + mu*M[i]

def resolution(A, Y):
    nl = A.shape[0]
    # mise sous forme triangulaire
    for i in range(nl-1):
        pivot = chercher_pivot(A,i)
        if pivot > i :
            echange_lignes(A, i, pivot)
            echange_lignes(Y, i, pivot)
            for j in range(i+1, nl):
                mu = -A[j,i]/A[i,i]
                transvection(A, i, j, mu)
                transvection(Y, i, j, mu)
    # remonté de pivot
    X = np.zeros(np.shape(Y)[0])
    for i in range(len(X)-1, -1, -1):
        X[i]= (Y[i]- np.sum(A[i,i+1:]*X[i+1:])) /A[i,i]
    return X
```