TP n° 11 – Pivot de Gauss

I Rappels

L'objectif de l'algorithme du pivot de Gauss est de résoudre un système linéaire. Dans ce TP, nous nous contenterons de résoudre des systèmes de Cramer, autrement dit des systèmes à n équations, n inconnues qui ont une solution unique.

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ & \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n &= b_n \end{cases}$$

Matriciellement, on cherche à résoudre $AX = B$ avec A un tableau numpy $(n \times n)$, et B de taille $(n \times 1)$.
Les grandes étapes de l'algorithme sont les suivantes (les étapes sont différentes de celles présentées en cours de maths)

1. choix d'une ligne de référence L_i (on choisira celle avec le pivot maximal)
2. normalisation de la ligne de référence (on met la valeur du pivot à 1)
3. élimination des termes en dessous et au dessus du pivot
4. itérer ces étapes pour obtenir une matrice échelonnée réduite.

On va programmer des fonctions qui vont modifier les matrices A et B et renvoyer l'unique solution du système $AX = B$.

II Pseudo-code

Pour résoudre le système $AX = B$, on réduit la matrice augmentée $(A|B)$ du système avec le pseudocode suivant :

pour i allant de 1 à n : ♯ on parcourt toutes les lignes de A
  parmi les lignes $L_i, L_{i+1}, \cdots, L_n$, choix de la ligne de référence : L_{kmax}
  échange des lignes L_i et L_{kmax} ♯ on place la ligne de référence ♯ en haut
  $L_i \leftarrow \dfrac{1}{a_{i,i}} L_i$ ♯ on égalise le pivot à 1
  pour $k \neq i$: ♯ pour toutes les autres lignes
    $L_k \leftarrow L_k - a_{k,i} L_i$ ♯ on élimine les termes ♯ au dessus et en dessous du pivot

Remarque. Les modifications sur les lignes doivent être faites sur A et B.
A la fin de l'algorithme, le système étant de Cramer, A est la matrice identité et la matrice B contient la solution du système.

III Choix de la ligne de référence

Quand on fait la résolution à la main, on choisit la ligne de référence qui permet les calculs les plus simples. Ici, on ne peut évidemment pas appliquer cette stratégie.
Dans la résolution, on est amené à diviser par le pivot (autrement dit, le premier terme de la ligne). Pour éviter les erreurs d'arrondi, le meilleur pivot est celui qui est le plus grand en valeur absolue.

On a besoin d'une fonction `ligneRef` qui détermine la ligne de référence parmi les lignes $L_i, L_{i+1}, \cdots, L_n$.
On compare les pivots $a_{ii}, a_{i+1,i}, \cdots, a_{n,i}$ et la fonction renvoie le numéro de la ligne du meilleur pivot.
Par exemple, pour `A=np.array([[1,12,13],[-7,8,10],[0,9,7]])`
`ligneRef(A,0)` renvoie 1
`ligneRef(A,1)` renvoie 2.

Exercice 1.
1. Que va renvoyer `ligneRef(A,2)` ?
2. Coder la fonction `ligneRef`. Elle prend comme entrée la matrice A et le numéro i et renvoie k_{max}, le numéro de la ligne de référence.

IV Algorithme du pivot de Gauss

Exercice 2. Ecrire une fonction `echangeLignes` qui échange les lignes L_i et L_k d'une matrice A.
La fonction prend comme entrée la matrice A et le numéro des lignes i et k. On fera une fonction qui ne retourne rien, mais qui modifie A.

```
>>>A=np.array([[1,2,3],[-3,-3,1],[0,2,7]])
>>>echangeLignes(A,1,2)
>>>A
[[1,2,3],[0,2,7],[-3,-3,1]]
```

Exercice 3. Ecrire une fonction de transvection : $L_k \leftarrow L_k - \alpha L_i$. La fonction `transvection` prend comme entrée la matrice A, les numéros des lignes i, k et le coefficient α. Elle modifie la matrice A mais ne renvoie rien.

Exercice 4. Ecrire une fonction `Gauss` qui résout par la méthode du pivot de Gauss le système (supposé de Cramer) $AX = B$: la fonction prend comme entrée les deux matrices A et B et renvoie la solution X.

Exercice 5. Testez votre programme avec les matrices suivantes :
$A = \begin{pmatrix} 2.1 & 3 \\ 1 & 1 \end{pmatrix}$ et $B = \begin{pmatrix} 5.1 \\ 2 \end{pmatrix}$.
`Gauss(A,B)` renvoie `[1.0,1.0]`.

Exercice 6. Il existe une fonction implémentée dans Python qui permet de résoudre des systèmes linéaires : la fonction `linalg.solve` de la bibliothèque `scipy`. Testez-la sur des exemples. Obtient-on toujours le même résultat ?

Exercice 7. Résolvez **à la main** le système suivant :

$$\begin{cases} -\frac{1}{2}x + y + z = 1 \\ \frac{1}{3}x + 2y - 2z = 0 \\ \frac{1}{6}x - 3y + z = -1 \end{cases}$$

Appliquez la fonction `Gauss` pour résoudre ce système. Que se passe-t-il ? Pourquoi ?

Exercice 8. Résolvez **à la main** le système suivant :

$$\begin{cases} x + (1+10^{-30})y + z = 1 \\ x + y + z = 1 \\ y + z = 1 \end{cases}$$

Appliquez la fonction `Gauss` pour résoudre ce système. Que se passe-t-il ? Pourquoi ?

Exercice 9. Pour inverser une matrice A, on lui applique les opérations élémentaires jusqu'à obtenir la matrice identité. Si on applique ces opérations à I_n, la matrice finale est A^{-1}.
Ecrire une fonction qui renvoie l'inverse d'une matrice A.
On pourra comparer les résultats avec ceux obtenus avec la fonction `linalg.matrix_power(A,-1)` de la bibliothèque numpy.

Correction TP n° 11 – Pivot de Gauss

Solution 1.

```
def ligneRef(A,i):
    n=len(A)
    maxi = abs(A[i,i])
    kmax = i
    for k in range(i+1,n):
        if abs(A[k,i]) > maxi:
            maxi = abs(A[k,i])
            kmax = k
    return kmax
```

Solution 2. Méthode 1 :

```
import copy
def echangeLignes(A,i,k):
    A1=copy.deepcopy(A[i])
    A2=copy.deepcopy(A[k])
    A[k]=A1
    A[i]=A2
```

Méthode 2 :

```
def echangeLignes(A,i,k):
    n=len(A)
    for j in range(n):
        x = A[i,j]
        A[i,j] = A[k,j]
        A[k,j] = x
```

Solution 3.

```
def transvection(A,i,k,alpha):
    A[k]=A[k]-alpha*A[i]
```

Solution 4.

```
1  def pivotGauss(A,B):
2      n=len(A)
3      for i in range(n):
4          kmax = ligneRef(A,i)
5          pivot=A[kmax,i]
6          if pivot==0:
7              return('erreur')
8          # echange lignes i et kmax dans A et dans B
9          echangeLignes(A,i,kmax)
10         echangeLignes(B,i,kmax)
11         # normalisation de la ligne du pivot
12         for j in range(n):
13             A[i,j] = A[i,j]/pivot
14         B[i] = B[i]/pivot
15         # on elimine les termes au dessus et en dessous du pivot
16         for k in range(n):
17             if k != i:
18                 alpha=A[k,i]
19                 transvection(A,i,k,alpha)
20                 B[k]=B[k]-alpha*B[i]
21     return B
```

Solution 6. Aux erreurs d'arrondis près, on obtient la même solution. Mais donc `linalg.solve` ne fait pas strictement les mêmes calculs.

Solution 7. Le sytème n'est pas de Cramer : $L_1 + L_2 + L_3 = 0$: il existe une infinité de solution. Mais pour Python, $-\frac{1}{2} + \frac{1}{3} + \frac{1}{6}$ n'est pas exactement 0. L'algorithme renvoie un vecteur qui n'est pas solution.

Solution 8. C'est bien un système de Cramer qui admet une unique solution : $(0, 0, 1)$. Mais avec les erreurs d'arrondis, Python approxime $L_1 \approx L_2$. Il renvoie un message d'erreur car il divise par zéro.

Solution 9.

```
def inverse(A):
    n=len(A)
    #I est la matrice identite de taille n
    # toutes les operations effectuees sur A sont faites sur I
    I=np.eye(n)
    for i in range(n):
        kmax = ligneRef(A,i)
        pivot=A[kmax,i]
        if pivot==0:
            return('erreur')
        # echange lignes i et kmax dans A et dans B
        echangeLignes(A,i,kmax)
        echangeLignes(I,i,kmax)
        # normalisation de la ligne du pivot
        for j in range(n):
            A[i,j] = A[i,j]/pivot
            I[i,j] = I[i,j]/pivot
        # on elimine les termes au dessus et en dessous du pivot
        for k in range(n):
            if k != i:
                alpha=A[k,i]
                transvection(A,i,k,alpha)
                transvection(I,i,k,alpha)
    return I
```