TP – Pivot de Gauss

Pour résoudre le système AX = B supposé de Cramer (la matrice A est inversible), on construit la matrice AB en accolant la matrice A et la matrice B. Si A est de taille n×n et B de taille n×1, alors la matrice AB est de taille n×(n+1). Puis on effectue les opérations sur les lignes jusqu'à arriver à un système triangulaire. On remonte ensuite par substitutions pour obtenir la solution du système.

1. Ecrire une fonction *concatener(A,B)* prenant en argument une matrice carrée A de taille n et une matrice colonne B de taille n et renvoyant la matrice AB correspondante. On devra obtenir :

```
>>> concatener ([[2 ,3] ,[1 ,0]] ,[[4] ,[1]])
[[2, 3, 4], [1, 0, 1]]
```

➲ ***Corrigé***

```
def concatener(A,B):
    n=len(A[0])+len(B[0])
    L=[[0] * n for _ in range(len(A))]
    for i in range(len(A)):
        a=0
        for j in range(len(A[0])):
            L[i][j]=A[i][j]
            a+=1
        for k in range(len(B[0])):
            L[i][a]=B[i][k]
            a+=1
    return L
```

2. Ecrire une fonction *echanger(M,i,j)* qui échange dans la matrice M la ligne i et la ligne j. On devra obtenir :

```
>>> M=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
>>> echanger(M,0,2)
>>> M
[[9, 10, 11, 12], [5, 6, 7, 8], [1, 2, 3, 4]]
```

➲ ***Corrigé***

```
def echanger(M,i,j):
    for k in range(len(M[i])):
        M[i][k],M[j][k]=M[j][k],M[i][k]
```

3. Ecrire une fonction *ajouter(M,i,j ,a)* qui ajoute, à la ligne i, a fois la ligne j.

```
>>> M=[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
>>> ajouter(M,0,2,-2)
>>> M    [[-17, -18, -19, -20], [5, 6, 7, 8], [9, 10, 11, 12]]
```

➲ ***Corrigé***

```
def ajouter(M,i,j,a):
    for k in range(len(M[i])):
        M[i][k]+=M[j][k]*a
```

4. Ecrire une fonction *choixPivot(M,i)* qui cherche dans les éléments m_{ii}, $m_{(i+1),i}$,...,m_{ni} le premier pivot non nul et qui renvoie l'indice de la ligne correspondante. On doit obtenir le résultat suivant :

```
>>> M = [[1,3,2,4,1],[0,0,3,7,1],[0,1,5,1,0],[1,2,1,1,4]]
>>> choixPivot(M,1)
2
```

➲ ***Corrigé***

```
def choixPivot(M,j):
    maxi=j
    for k in range(j, len(M)):
        if M[k][j]>0:
            maxi=k
            break
    return maxi
```

Lorsque l'on s'occupe de la colonne d'indice i, l'algorithme du pivot de Gauss se déroule de la façon suivante :

- ***rechercher un pivot : on le trouve à la ligne i_0 ;***
- ***échanger les lignes i et i_0 ;***
- ***pour chaque valeur de k entre i + 1 et n, réaliser l'opération $L_k \leftarrow L_k - \frac{m_{ki}}{m_{ii}} L_i$***

Et on passe ensuite à la colonne suivante. On fait cela sur la matrice AB pour les colonnes de 1 à n.

5. Ecrire une fonction *trianguler(M)* qui réalise cette opération. On devrait obtenir le résultat suivant :

```
>>> M = [[1,3,2,4,3],[0,0,3,6,3],[0,1,5,3,4],[1,2,1,2,2]]
>>> trianguler(M)
>>> M
[[1, 3, 2, 4, 3], [0, 1, 5, 3, 4], [0, 0, 3, 6, 3], [0, 0, 0, -7, -1]]
```

➲ *Corrigé*

```
def trianguler(M):
    for i in range(len(M[0])-1):
        pivot=choixPivot(M,i)
        if pivot>i:
            echanger(M,i,pivot)
        for k in range(i+1,len(M)):
            c=M[k][i]/M[i][i]
            ajouter(M,k,i,-c)
    return M
```

À la suite de cet algorithme, on obtient un système triangulaire. Il faut ensuite remonter le système par substitution. Si $A^*X = B^*$ est un système triangulaire, on calcule $x_n = \frac{b_n^*}{a_{nn}^*}$. Puis lorsque l'on cherche à calculer x_{n-i}, on connait déjà $x_n, x_{n-1}, \ldots, x_{n-i+1}$ et on a

$$x_{n-i} = \frac{1}{a^*_{(n-i),(n-i)}} \left(b^*_{n-i} - a^*_{(n-i),(n-i+1)} x_{n-i+1} - \cdots - a^*_{(n-i),n} x_n \right)$$

6. Ecrire la fonction *remonter(M)* de remontée qui prend en argument une matrice de type AB avec A supposée triangulaire et renvoie le vecteur colonne solution X associé. On doit obtenir le résultat suivant :

```
>>> M = [[1,2,3],[0,1,1]]
>>> remonter(M)
[[1.0] , [1.0]]
```

➲ *Corrigé*

```
def remonter(M):
    L=[[0] for _ in range(len(M))]
    L[-1]=M[-1][-1]
    for k in range(len(M)-1,-1,-1):
        s=0
        for i in range(k+1,len(L)):
            s+=M[k][i]*L[i]
        L[k]=(M[k][-1]-s)/M[k][k]
    return L
```

7. Ecrire une fonction *PivotDeGauss(A,B)* qui prend en argument la matrice A carrée supposée inversible et la matrice colonne B et renvoie le vecteur solution X de l'équation AX = B. Par exemple, pour un système particulier on aura :

```
>>> PivotDeGauss ([[10 ,7 ,8 ,7] ,[7 ,5 ,6 ,5] ,[8 ,6 ,10 ,9] ,[5 ,7 ,9 ,10]] , [[32] ,[23] ,[33] ,[31]])
[[1.0] , [1.0] , [1.0] , [1.0]]
```

➲ *Corrigé*

```
def PivotDeGauss(A,B):
    M=concatener(A,B)
    trianguler(M)
    return remonter(M)
```

8. Essayer l'algorithme du pivot de Gauss sur le système suivant :

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

Le résultat est-il bien une solution ?

➲ ***Corrigé***

```
A=[[2,1],[1,1]]
B=[[1],[2]]
M=PivotDeGauss(A,B)
print(M)
So=np.linalg.solve(A,B)
print(So)
```

9. Modifier la fonction de recherche du pivot afin d'obtenir une stratégie de pivot partiel. On devrait obtenir le résultat suivant :

```
>>> M = [[1,3,2,4,1],[0,0,3,7,1],[0,1,5,1,0],[1,2,1,1,4]]
>>> choixPivotPartiel(M,1)
3
```

➲ ***Corrigé :***

```
def choixPivot(M,j):
    max=j
    for k in range(j, len(M)):
        if abs(M[k][j])>abs(M[max][j]):
            max=k
    return max
```