TD n°3,4,5 - METHODE DU PIVOT DE GAUSS

Contexte :

On considère un système linéaire de la forme $AX = B$ avec A matrice carrée de taille n et B vecteur colonne de taille n. La matrice A est supposée inversible donc le système admet une unique solution.

But :

Résolution de ce type de système linéaire par la méthode du pivot de Gauss-Jordan.

Principe :

1. On écrit la matrice augmentée M associée au système,
2. On échelonne cette matrice grâce à la méthode du Pivot de Gauss,
3. On résout le système triangulaire obtenu par remontée.

1. ECRITURE DE LA MATRICE ECHELONNEE

On rappelle que sous Python, une matrice est écrite comme une liste de liste.

Exemple : `M=[[4,3,-1,5],[-8,4,5,6],[1,6,-3,5]]`

`len(M)` donne le nombre de lignes n de la matrice.
`M[i]` renvoie la ligne i de M.
`M[i][j]` renvoie le coefficient d'indice (i, j) de M.
`0*len(M)` renvoie une liste nulle de taille n.

Il n'est pas essentiel de retenir par cœur mais utile quand même de savoir qu'il existe dans `Python` *une commande qui transpose une matrice :*

`M.T` *renvoie la transposée de la matrice* M.

Programmes Python : Voici un court programme permettant de passer du couple A, B de matrices du système à la matrice augmentée M. (vu sur CCP)

```
import numpy as np
def Augmente(A,B):
    n=len(A)
    m=len(B.T)
    M=np.concatenate((A,B.T),axis=1)
    return M
```

Voici un programme affichant la matrice augmentée ligne par ligne :

```
def affiche(M):
    for i in range(len(M)):
        print(M[i])
```

2. ALGORITHME DE REMONTEE

Le système étant triangulaire de la forme :

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ & a_{2,2} & \cdots & a_{2,n} \\ & (0) & \ddots & \vdots \\ & & & a_{n,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

On calcule les valeurs de $x_1, x_2, \ldots x_n$ par récurrence descendante et on a :

$$\begin{cases} x_n = \frac{y_n}{a_{n,n}} \\ \forall i \in \{n-1, \cdots, 1\}, \quad x_i = \frac{1}{a_{i,i}} \left(y_i - \sum_{k=i+1}^{n} a_{i,k} x_k \right) \end{cases}$$

Programme Python : **En pseudo-code**

```
Pour i descendant de n-1 à 0 faire :
    Pour j descendant de n-1 à i+1
        yi ← yi − ai,j xj
        Fin Pour
        xi ← yi / ai,i
    Fin Pour
```

En Python

```
def remonter(M):
    n=len(M) #on définit la taille de la matrice
    X=[0]*n  #on initialise le "vecteur" (liste) solution au vecteur nul
    X[n-1]=M[n-1][n]/M[n-1][n-1] #on calcule la dernière coordonnée
    for i in range(n-2,-1,-1): # Pour chaque coordonnée de la solution X
        X[i]=M[i][n] #Initialisation de la i-ème coordonnée de X
        for j in range(i+1,n): # Boucle pour soustraire la somme
            X[i]=X[i]-M[i][j]*X[j]
        X[i]=X[i]/M[i][i]  #division finale par M[i][i]
    return(X)
```

Complexité de l'algorithme :

Il y a une double boucle sur les indices i et j, soit $\frac{n(n-1)}{2}$ boucles

Dans chaque boucle on effectue 1 soustraction et 1 multiplication.
Enfin, il y a n divisions soit un total de :

$$2\frac{n(n-1)}{2} + n = n^2 \text{ opérations}$$ (*aux opérations initiales près*)

3. ALGORITHME DU PIVOT DE GAUSS

Programme Python : **En pseudo-code**

Pour j **de** 0 **à** $n-2$ **faire**
  Trouver i entre j et $n-1$ tel que $|a_{i,j}|$ soit maximal.
  Échanger L_i et L_j.
  Pour i **de** $j+1$ **à** $n-1$ **faire**
    $L_i \leftarrow L_i - \frac{a_{i,j}}{a_{j,j}} L_j$
  Fin Pour
Fin Pour

En Python

• Programme effectuant une transposition

```
def transposition(M,L1,L2) : #ligne1<-->ligne2
    X=M[L1]
    M[L1]=M[L2]
    M[L2]=X
```

• Programme effectuant une transvection (sur la matrice augmentée)

```
def transvection(M,L1,L2,k) : #ligne2<--ligne2 + k fois ligne1
        for j in range(0,len(M)+1):
            M[L2][j]=M[L2][j]+k*M[L1][j]
```

• Programme cherchant le pivot maximal (pour une colonne fixée)

```
def lignePivotMax(M,c):
    max=M[c][c]
    indiceMax=c
    for i in range(c,len(M)):
        if M[i][c]>max:
            max=M[i][c]
            indiceMax=i
    return(indiceMax)
```

(*il est plus efficace de chercher le pivot maximal que le premier pivot non nul*)

• Programme appliquant l'algorithme du pivot de Gauss (pivot maximal)

```
def triangulariser(M) :
    for j in range(0,len(M)-1):
        numLignePivot=lignePivotMax(M,j)
        transposition(M,j,numLignePivot)
        for i in range(j+1,len(M)):
            transvection(M,j,i,-M[i][j]/M[j][j])
    return(M)
```

Complexité de l'algorithme :

Pour la recherche d'un pivot maximal :

Il y a une double boucle sur les indices i et j, soit environ $\frac{n(n-1)}{2}$ boucles et donc autant de comparaisons.

Pour la méthode du Pivot de Gauss :

Il y a une double boucle sur les indices i et j, soit $\frac{n(n-1)}{2}$ boucles.

A chaque boucle on effectue une transvection, soit :

- Une division par $a_{j,j}$,
- Une soustraction entre lignes, donc n opérations,
- Une multiplication par $\frac{a_{i,j}}{a_{i,i}}$, donc n opérations.

Soit un nombre d'opérations au total de :

$$2n\ \frac{n(n-1)}{2} + \frac{n(n-1)}{2}\ n^2 \approx n^3 \text{ opérations}$$

4. PROGRAMME FINAL

Il suffit d'appliquer les deux programmes précédents à la suite.

Programme Python :

```
def pivotGauss(M):
    M=triangulariser(M)
    X=remonter(M)
    return(X)
```

Complexité de l'algorithme :

Il suffit de sommer les deux complexités obtenues.

On retiendra que la complexité totale est équivalente à n^3

Et que c'est la partie relative au pivot de Gauss qui porte la complexité la plus grande (n^3 contre n^2 pour la remontée).

BILAN : Que retenir pour les concours ?

On ne vous demandera pas d'implémenter l'algorithme en entier mais celui-ci vous est donné vous devez être capable d'identifier les différents sous-programmes ou compléter des morceaux manquants, ainsi que calculer sa complexité. Eventuellement modifier l'algorithme afin de s'adapter à certains types de matrices spécifiques en entrée (pour diminuer la complexité).