Informatique Chapitre 10 : pivot de Gauss

I. Introduction

La méthode du pivot de Gauss a été vue de façon théorique en cours de mathématiques. Elle permet de résoudre un système linéaire de n équations et p inconnues.
Cette méthode s'exprime bien sous forme d'algorithme car elle est basée sur l'exécution de tâches simples et répétitives.

Cette méthode de résolution va également illustrer les problèmes de précision propres au calcul numérique, les erreurs d'arrondis dues au codage en binaire pouvant induire des erreurs.

Dans ce cours nous allons étudier de façon pratique les cas particuliers simples de systèmes admettant une solution et une seule pour $n = p$ (on parlera de système de Cramer).

II. Principe de résolution

a. Système triangulaire

Il s'agit de systèmes d'équations très simples à résoudre qui se présentent tel l'exemple suivant :

$$\begin{cases} 2x + 2y - 3z = 2 \\ y - 6z = -3 \\ z = 4 \end{cases} \Leftrightarrow \begin{cases} 2x + 2y - 3z = 2 \\ y = 6z - 3 = 21 \\ z = 4 \end{cases} \Leftrightarrow \begin{cases} x = \frac{1}{2}(-2y + 3z + 2) = -14 \\ y = 21 \\ z = 4 \end{cases}$$

La résolution d'un système triangulaire sera la dernière étape de la résolution d'un système linéaire. On l'appelle souvent *phase de remontée* : on résout les équations de bas en haut, en substituant aux inconnues les valeurs trouvées dans les lignes inférieures.

Le système d'équations ci-dessus peut se mettre sous la forme matricielle suivante :

$$AX = Y \text{ avec } A = \begin{pmatrix} 2 & 2 & -3 \\ 0 & 1 & -6 \\ 0 & 0 & 1 \end{pmatrix}, X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ et } Y = \begin{pmatrix} 2 \\ -3 \\ 4 \end{pmatrix}.$$

On notera $a_{i,j}$ le coefficient de la matrice A se trouvant à l'intersection de la ligne i avec la colonne j. Le premier indice est toujours celui de la ligne, le second celui de la colonne.

b. Transvections

Pour réaliser la phase de remontée, il faut au préalable mettre le système sous forme triangulaire. Pour cela on réalise des *transvections*, c'est-à-dire des combinaisons linéaires des équations du système afin d'éliminer certaines inconnues.
Les *transvections* seront toujours appliquées de la même façon avec une méthode systématique (la méthode ne doit rien à l'intuition ni au hasard).

Soit par exemple un système de trois équations (e_1, e_2, e_3) à trois inconnues (x, y, z).
On note a (avec $a \neq 0$) le coefficient en x de la première équation e_1. On se sert de a comme *pivot* pour éliminer les occurrences de x dans e_2 et e_3. Si on note b et c les coefficients de x dans e_2 et e_3, le nouveau système constitué des équations : e_1, $e_2' = e_2 - \frac{b}{a}e_1$ et $e_3' = e_3 - \frac{c}{a}e_1$ est alors équivalent au premier et ne fait apparaître x que dans la première équation.
Soit d le coefficient y de e_2' (c'est le nouveau pivot) et f celui de y dans e_3', le système constitué des équations e_1, e_2' et $e_3' - \frac{f}{d}e_2'$ est équivalent au premier système mais est maintenant triangulaire.

Lors de la première étape, on ne touche pas à la première ligne. A la deuxième étape, on ne touche ni à la première ni à la deuxième ligne. Deux étapes ont suffi pour un système de dimension 3.
Pour simplifier les écritures, on note simplement $e_2 \leftarrow e_2 - \frac{b}{a}e_1$ et $e_3 \leftarrow e_3 - \frac{c}{a}e_1$ puis $e_3 \leftarrow e_3 - \frac{f}{d}e_2$.

Dans le cas où le pivot n'est pas là où on le veut (par exemple, le coefficient en x de la première équation est nul), on peut échanger l'ordre des lignes.

Exemple (les pivots successifs sont notés en gras) :

$$\begin{cases} \mathbf{2}x + 2y - 3z = 2 \\ -2x - y - 3z = -5 \\ 6x + 4y + 4z = 16 \end{cases} \begin{matrix} e_2 \leftarrow e_2 + e_1 \\ e_3 \leftarrow e_3 - 3e_1 \end{matrix} \Leftrightarrow \begin{cases} 2x + 2y - 3z = 2 \\ \mathbf{1}y - 6z = -3 \\ -2y + 13z = 10 \end{cases} \begin{matrix} \\ e_3 \leftarrow e_3 + 2e_2 \\ \\ \end{matrix} \Leftrightarrow \begin{cases} 2x + 2y - 3z = 2 \\ y - 6z = -3 \\ z = 4 \end{cases}$$

c. Problème de la comparaison à zéro

On a déjà vu les problèmes de précisions rencontrés en informatique quand on travaille en flottants (puisque tous les nombres ne sont pas représentables de façon exacte en base 2). Par exemple :

```
>>> 12*(1/3-1/4)-1
-2.220446049250313e-16
```

Alors que le résultat exact est : $12\left(\frac{4-3}{12}\right) - 1 = 0$.
Les calculs en flottants peuvent donc faire apparaître des termes extrêmement petits mais non nuls pourtant censés représenter 0. Cela peut conduire à prendre comme pivot une quantité très faible issue de l'accumulation d'approximations et donc à obtenir des quantités très grandes après division par le pivot. Le résultat obtenu sera alors faux.
Pour limiter ces erreurs pour un système de Cramer, on cherchera sur la colonne en cours le coefficient le plus élevé en valeur absolue comme pivot (ce qui n'est donc pas nécessaire pour un calcul à la main).

III. Algorithme

a. Mise sous forme triangulaire

Soit un système de Cramer de n équations et n inconnues. Pour éliminer des variables dans les équations successives il faut qu'après k étapes, pour tout i entre 1 et k, la $i^{ème}$ variable ait disparu de toutes les équations du système à partir de la $(i+1)^{ème}$ ligne du système.
Ainsi, après la $(n-1)^{ième}$ étape, le système sera bien sous forme triangulaire.

Dans le pseudo-code qui suit, on résout le système $AX = Y$.

La ligne L_i désigne les coefficients de la matrice A de la ligne i et les composantes du second membre Y qui est un vecteur (on rappelle que les indices des tableaux vont de 0 à $n-1$).

Pour $j \leftarrow 0$ A $n-2$
    Trouver i entre j et $n-1$ tel que $|a_{i,j}|$ soit maximale
    Echanger L_i et L_j
    Pour $k \leftarrow j+1$ A $n-1$
        $L_k \leftarrow L_k - \frac{a_{k,j}}{a_{j,j}} L_j$

Les lignes « Trouver… » et « Echanger… » seront à remplacer par le pseudo-code adéquat qui correspondra en fait à deux fonctions dont on parlera plus loin.

Les deux dernières lignes correspondront à une troisième fonction (Transvection).

On retient que rechercher i entre j et $n-1$ tel que $|a_{i,j}|$ soit maximale puis échanger deux lignes a deux objectifs :

- S'assurer que le coefficient en position (j, j) sera différent de 0 (c'est le pivot).
- Minimiser les erreurs numériques.

b. Remontée

Le système étant triangulaire, il n'y a plus qu'à exécuter la phase de remontée via des substitutions. Le résultat sera mis dans un tableau X. Il s'agit donc de calculer :

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

Soit en pseudo code :

Pour $i \leftarrow n-1$ A 0
    Pour $k \leftarrow i+1$ A $n-1$
        $y_i \leftarrow y_i - a_{i,k}.x_k$
    $x_i \leftarrow \frac{y_i}{a_{i,i}}$

c. Structuration

On peut donc envisager de scinder le programme en trois sous-parties :

- Définition d'une fonction *recherche_pivot* qui prend en compte la recherche du pivot. Elle doit indiquer à quelle ligne i se trouve le plus grand terme d'une colonne donnée j. Cette fonction prend en entrée une matrice A et un indice j. Elle doit renvoyer un indice $i \geq j$ tel que $|a_{i,j}|$ soit maximale.
- Définition d'une fonction *echange_lignes* permettant des échanges de lignes. Cette fonction prend en entrée la matrice A, l'indice i et l'indice j et ne renverra rien.
- Définition d'une fonction *transvection* permettant de faire les transvections. Cette fonction prend en entrée la matrice A, l'indice i, l'indice j et un coefficient μ correspondant au coefficient utilisé dans la transvection (elle permet de remplacer la ligne L_i par la ligne $L_i + \mu L_j$). Elle ne renverra rien.

d. Rappel de syntaxe Python

Une matrice correspond à une liste à 2 dimensions. Par exemple :

```
>>> A=[[2,2,-3],[-2,-1,-3],[6,4,4]]
>>> A[2][0]
6
```

e. Programme principal

Le programme principal contiendra donc :

- La mise sous forme triangulaire (à l'aide des trois fonctions précédentes).
- La phase de remontée.

Il pourra lui-même se mettre sous la forme d'une fonction admettant une matrice A et un vecteur Y comme arguments.

Le résultat (arrondi) sera par exemple le suivant :

```
>>> A=[[2,2,-3],[-2,-1,-3],[6,4,4]]
>>> Y=[[2],[-5],[16]]
>>> resolution(A,Y)
[-14.0, 21.0, 4.0]
```


IV. Pseudo codes

Il est important de tester le bon fonctionnement de chaque fonction au fur et à mesures de leurs écritures. On enregistrera toutes les fonctions dans le fichier *pivot_de_gauss.py*.

a. Recherche pivot

```
Fonction recherche(A,j) # j est le numéro de la colonne considérée
    n ← Longeur(A) # donne le nombre d'équations à résoudre et le nombre de colonnes de A
    imax ← j  # ligne du maximum provisoire
    max ← Abs(A[j][j]) # maximum provisoire sur la diagonale
        Pour k ← j+1 A n-1
        Si Abs(A[k][j]) > max
            max ← Abs(A[k][j])
            imax ← k
    Retourner imax
```

b. Echange lignes

```
Fonction echange (A,i,j) # i : ligne courante ; j : ligne contenant le pivot
    nc ← Longueur(A[0]) # nombre de colonnes. Pour le second membre Y on a nc = 1
    Pour k ← 0 A nc-1 # k numéro de colonne
        tmp ←  A[j][k]
        A[j][k] ← A[i][k]
        A[i][k] ← tmp
    # Ecrire A # pour vérifier le bon fonctionnement de la fonction
```

c. Transvection

```
Fonction transvection(A,i,j,mu) # remplace Li par Li+mu.Lj dans A
    nc ← Longueur(A[0]) # nombre de colonnes. Pour le second membre Y on a nc = 1.
    Pour k ← 0 A nc-1 # k numéro de colonne de la ligne i considérée. On peut commencer à  k = 0 car
                      les termes avant la colonne j (celle du pivot qui est sur la diagonale) sont nuls
        A[i][k] = A[i][k] + mu * A[j][k]
```

d. Programme principal : résolution

```
Fonction resolution(A,Y) # matrice et second membre
    n ← Longueur(A)
    Pour j ← 0 A n-1 # numéro de colonne
        imax ← recherche(A,j) # ligne du pivot
        echange(A,j,imax)
        echange(Y,j,imax) # ne pas oublier le second membre
        Pour k ← j+1 A n-1 # toutes les lignes sous le pivot qui se trouve toujours sur la diagonale
            mu ← -A[k][j]/A[j][j]
            transvection(A,k,j,mu)
            transvection(Y,k,j,mu)
    X = n*[0]
    Pour i ← n-1 A 0 # Python : for i in range(n-1,-1,-1):
        Pour k ← i+1 A n-1
            Y[i][0] ← Y[i][0] – A[i][k]*X[k]
        X[i] ← Arrondi(Y[i][0] / A[i][i])
    Retourner X
```

Mise sous forme triangulaire

Remontée

e. Mise en œuvre

On définit les matrices A et Y et on appelle la fonction resolution(A,Y) selon :
Ecrire « La solution du système AX = Y est » + resolution(A,Y).