Pivot de Gauss

PTSI Lycée Eiffel

22 mai 2020

Cette dernière partie de cours consacrée à l'agorithme du pivot de Gauss devrait logiquement se trouver dans le chapitre 4 d'analyse numérique, à la suite de l'étude de la résolution des équations différentielles par la méthode d'Euler, mais n'ayant plus les sources du document ayant permis de produire ledit chapitre 4, je ne peux pas y insérer facilement cette partie. Peu importe, le pivot de Gauss joue de toute façon un rôle à part parmi les algorithmes étudiés dans ce chapitre. Il permet comme vous le savez déjà de résoudre des systèmes d'équations linéaires (et aussi d'inverser des matrices, ce qui est un problème équivalent), ce qui peut logiquement lui valoir sa place parmi les autres techniques de « résolution d'équation » que nous avons déjà abordées. Mais, contrairement à la méthode de Newton ou à celle d'Euler, il n'est absolument pas question ici de faire une résolution **approchée** d'un système d'équations (et donc pas de faire à proprement parler de l'analyse numérique) mais d'effectuer une résolution **exacte** (quand on résout un système dont les solutions sont des nombres rationnels, on attend de notre machine qu'elle nous donne effectivement des valeurs exactes). Les seules limitations pratiques seront donc dues aux capacités de calcul de notre machine (et notamment au traitement des nombres décimaux par Python).

1 Présentation générale et propriétés de l'algorithme du pivot de Gauss.

Je ne vous ferai pas l'affront de vous décrire en détail un algorithme que vous connaissez par coeur et savez déjà appliquer « à la main », contentons-nous donc de préciser un peu le cadre dans lequel nous allons l'étudier ici : on suppose que notre algorithme utilisera comme données une matrice carrée de taille n (en pratique une liste de listes en Python si on ne veut pas utiliser de modules spécifiques pour faire du calcul matriciel) et une liste de n valeurs représentant le second membre de notre système à résoudre. On souhaite que le programme que nous allons écrire renvoie lui-même une liste de n valeurs constituant l'unique solution du système (dans l'ordre correspondant à celui imposé par les coefficients de la matrice du système), et éventuellement nous renvoie un message d'erreur dans le cas où le système n'est pas un système de Cramer (on pourrait imaginer préciser que le système n'a pas de solution, ou donner une base de l'espace des solutions dans le cas où il y en a une infinité, mais le programme sera largement assez compliqué sans ça).

Avant même de détailler le programme, penchons-nous sur les propriétés théoriques de ce célèbre algorithme. Eh bien, précisons-le immédiatement, le pivot de Gauss a beau être célèbre, ça n'en est pas moins un algorithme particulièrement pourri ! En particulier à cause des trois défauts suivants :

- le pivot est **lent**, il a une complexité cubique (rappelons que cela signifie que le temps d'exécution de l'algorithme sera proportionnel à n^3, où n est la taille des données manipulées, ici le nombre d'équations du système), ce qui rend difficile la résolution de systèmes contenant un grand nombre d'équations par cette méthode. Mais bon, on n'y peut tout simplement rien, il n'existe pas d'algorithme général de résolution de systèmes qui soit plus efficace que le pivot (sinon on vous aurait mis au courant depuis un moment). On arrive simplement à améliorer significativement les choses dans des cas très particuliers (matrice du système contenant une grande majorité de 0, par exemple). Ne soyez pas surpris en tout cas si votre ordinateur refuse de résoudre en un temps raisonnable un système de 100 équations à 100 inconnues (ce qui

représente de toute façon 10 000 coefficients à saisir, votre paresse vous évitera de tenter le coup).

- l'algorithme du pivot appliqué « bêtement » (c'est-à-dire en prenant à chaque étape le premier coefficient non nul disponible sur la colonne pour servir de pivot) conduit naturellement à un phénomène d'**explosion des coefficients**, c'est-à-dire que les calculs intermédiaires vont faire intervenir des nombres d'ordre de grandeur nettement plus grand que celui des coefficients initiaux, et ce même si les solutions du système ont elle-même un ordre de grandeur raisonnable. Ce phénomène perturbe forcément la précision des calculs effectués, mais on peut l'éviter en ajoutant à l'algorithme une étape de sélection du « meilleur pivot » qui limite cette explosion.
- l'algorithme du pivot est **numériquement instable**, ce qui signifie qu'une très légère modification des données peut résulter en un très grand écart au niveau des solutions du système (même principe que les phénomènes chaotiques en mathématiques, ce qu'on nomme souvent de façon imagée « l'effet papillon »). Je donne un exemple concret de ce phénomène juste en-dessous. Bien entendu, là aussi, de légères imprécisions dans les calculs peuvent se répercuter de façon dramatique sur les solutions affichées par notre programme, ce qui est gênant. Mais il n'y a pas d'autre solution que de faire des calculs les plus précis possibles.

Un exemple concret d'instabilité numérique. Prenez la matrice $H = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$, et résolvez le système matriciel $H \times \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. Vous obtenez une solution unique : $x = -4$, $y = 60$, $z = -180$ et $t = 140$. Très bien. Maintenant, modifiez un tout petit peu le second membre de votre système pour mettre à la place de $(1, 1, 1, 1)$ les quatre valeurs $(1.01, 0.99, 1.01, 0.99)$, on obtient alors pour unique solution $x = 1.16$, $y = 3$, $z = -43.8$ et $t = 51.8$, c'est-à-dire des valeurs assez éloignées de celles du système de départ alors qu'on a simplement un écart d'1% au niveau du second membre. Autre expérience, revenez au second membre initial avec 1 sur chaque ligne, mais modifier la matrice du système en arrondissant chaque coefficient au centième près (donc 0.33 à la place de $\frac{1}{3}$ ou 0.14 à la place de $\frac{1}{7}$), l'unique solution du système devient alors $x \simeq 5.6$, $y \simeq -31.4$, $z \simeq 21.7$ et $t \simeq 15.6$, là encore des valeurs très différentes de ce qu'on avait initialement. Il se trouve que la matrice H est spécifiquement construite pour amplifier de façon particulièrement notable cette instabilité numérique, mais on peut très bien constater le même genre de phénomène à partir de matrices extrêmement banales.

2 Implémentation en Python de l'algorithme.

L'algorithme du pivot de Gauss étant assez complexe à programmer, du moins à notre niveau, il représente un bon exemple des réflexes que doit rapidement acquérir un bon programmeur. Avant de se lancer dans l'écriture d'un programme qui va nécessiter quelques dizaines de lignes de code, on réfléchit bien évidemment à la structure globale, mais surtout on essaie dans la mesure du possible de découper le travail en tâches élémentaires qui seront effectuées par plusieurs sous-programmes distincts, le programme principal se contentant essentiellement de mettre bout à bout ces différents morceaux. Cette façon de procéder permet de rendre le programme plus lisible, de répartir le travail si on est plusieurs à travailler sur un projet informatique commun, et surtout de vérifier « étape par étape » la validité de la programmation (si on écrit 100 lignes de code d'un coup et qu'à la fin ça ne fait pas ce qu'on veut, bon courage pour retrouver l'erreur). Ici, on pourra procéder comme suit :

- écrire un programme **pivotoptimal(M,j)** qui prend comme argument une matrice M et un numéro de colonne j et qui renvoie le numéro de ligne où se trouve le pivot optimal dans

cette colonne, c'est-à-dire, parmi les coefficients de la matrice situés sous la diagonale, celui dont la valeur absolue est la plus grande (c'est le meilleur choix pour diminuer le phénomène d'explosion des coefficients).

- écrire un programme **echangeligne(M,i,j)** qui échange les lignes numéros i et j dans la matrice M.
- écrire un programme **combinaison(M,i,j,a)** qui effectue dans la matrice M une combinaison du type $L_i \leftarrow L_i + aL_j$, permettant d'annuler les coefficients sous la diagonale dans une colonne de la matrice.
- écrire un programme **remontesysteme(M,B)** prenant comme arguments une matrice triangulaire supérieure M et une liste « second membre » B et qui calcule l'unique solution du système $MX = B$ en « remontant » partir de la dernière équation, comme on le ferait à la main. Autrement dit, on calculera successivement (en partant de la fin) $x_n = \dfrac{b_n}{m_{n,n}}$, puis $x_{n-1} = \dfrac{b_{n-1} - m_{n-1,n}x_n}{m_{n-1,n-1}}$, et de façon plus générale $x_i = \dfrac{b_i - \sum_{j=i+1}^{n} m_{i,j}x_j}{m_{i,i}}$.

Il ne reste plus qu'à écrire les différents programmes :

```
def pivotoptimal(M,j) :
    i=j
    max=abs(M[j][j])
    for k in range(j+1,len(M)) :
        if abs(M[k][j])>max :
            i=k
            max=abs(M[k][j])
    return i

def echangeligne(M,i,j) :
    M[i],M[j]=M[j],M[i]
    return

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

def remontesysteme(M,b) :
    x=b[ :]
    x[-1]=float(b[-1])/M[-1][-1]
    for i in range(len(M)-2,-1,-1) :
        x[i]=(b[i]-sum([M[i][j]*x[j] for j in range(i+1,len(M))]))/M[i][i]
    return x
```

Il est temps d'écrire le programme principal, en faisant attention à bien modifier le sedonc membre à chaque échange de liste ou combinaison effectuée dans la matrice. On copiera par ailleurs la matrice initiale (pour la garder intacte) en utilisant une fonction spéficique.

```
from copy import deepcopy
def pivotgauss(M,b) :
    Mat=deepcopy(M)
    for j in range(len(M)) :
        i=pivotoptimal(Mat,j)
        echangeligne(Mat,i,j)
```

```
        b[i],b[j]=b[j],b[i]
        for k in range(j+1,len(M)) :
                a=-float(Mat[j][j])/Mat[j][k]
                combinaison(Mat,j,k,a)
                b[j]=b[j]+a*b[k]
return remontesysteme(Mat,b)
```