Méthode de Gauss

1 Introduction

1.1 Contexte

Des systèmes d'équations linéaires apparaissent dans de nombreux problèmes : analyse numérique pour résoudre une équation aux dérivées partielles (équation de la chaleur par exemple), optimisation linéaire, modélisation économique (analyse entrée-sortie par Wassily Leontief).

Bien que la plus ancienne utilisation connue vienne de Chine et date d'environ deux mille ans[1], la méthode de résolution usuelle d'un système linéaire porte le nom de *méthode (ou algorithme) du pivot de Gauss*. En effet, Carl Friedrich Gauss la présente sous sa forme moderne en 1810 dans un livre consacré à la trajectoire d'un astéroïde.

1.2 Mise en forme mathématique

Un système linéaire peut être mis sous forme matricielle. On est donc ramené à résoudre une équation de la forme $AX = B$ d'inconnue un vecteur colonne X de taille n où A est une matrice carrée inversible de taille n et B un vecteur colonne de taille n.

⇝ exercice 1

Comme A est inversible, la solution est bien sûr $X = A^{-1}B$ mais nous allons voir qu'en pratique on peut calculer X efficacement sans calculer A^{-1}, c'est tout le but de l'algorithme de Gauss.

On verra cependant à la fin que cette méthode de Gauss permet également de déterminer A^{-1} ou encore de calculer $\det(A)$.

2 La démarche algorithmique

2.1 Décomposition du problème

Commençons par remarquer que la résolution d'un système de la forme $AX = B$ est très facile si la matrice A est triangulaire supérieure : il suffit de partir de la dernière ligne puis de remonter ligne par ligne.

Exemple 1.

$$\begin{pmatrix}1&2&3\\0&4&5\\0&0&6\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}7\\8\\9\end{pmatrix} \iff \begin{cases}x+2y+3z=7\\4y+5z=8\\6z=9\end{cases} \iff \begin{cases}x+2y+3z=7\\4y+5z=8\\z=\frac{9}{6}\end{cases}$$

$$\iff \begin{cases}x+2y+3z=7\\y=\frac{1}{4}\left(8-5\times\frac{9}{6}\right)\\z=\frac{9}{6}\end{cases} \iff \begin{cases}x=9/4\\y=1/8\\z=3/2\end{cases}$$

On va donc chercher à rendre le système linéaire triangulaire. Pour cela, il faut remarquer que la solution n'est pas modifiée lorsqu'on effectue les deux opérations dites *élémentaires* suivantes sur les lignes de A et B :

— échange de deux lignes : $L_i \longleftrightarrow L_j$;

— modification d'une ligne par un multiple d'une autre : $L_i \leftarrow L_i + \alpha L_j$ (avec $j \neq i$).

⇝ exercice 2

Ainsi la méthode de Gauss va se faire en deux étapes :

① Transformation du système via des opérations élémentaires afin de le rendre triangulaire,

② Une fois cela fait, résolution du système triangulaire en partant de la dernière ligne.

1. Chapitre 8 du livre « Les Neuf Chapitres sur l'art mathématique » : `https://fr.wikipedia.org/wiki/Les_Neuf_Chapitres_sur_l%27art_math%C3%A9matique`

2.2 Mise sous forme triangulaire

Plaçons-nous quelque part au milieu de cette phase de mise sous forme triangulaire, disons à l'étape j, et supposons que la matrice soit à ce moment-là de la forme

$$\begin{pmatrix} a_{1,1} & & & \cdots & & a_{1,n} \\ 0 & a_{2,2} & & \cdots & & a_{2,n} \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & 0 & a_{j,j} & \cdots & a_{j,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & a_{n,j} & \cdots & a_{n,n} \end{pmatrix}.$$

L'idée la plus directe pour se rapprocher d'une matrice triangulaire à l'étape $j+1$ consiste à effectuer les opérations

$$(*) \quad L_i \leftarrow L_i - \frac{a_{i,j}}{a_{j,j}} L_j \text{ pour tout } j \in [\![j+1 \,;\, n]\!].$$

De cette façon, on fait apparaître des 0 sur la colonne j sous la diagonale.

Problème : le pivot, *i.e.* le coefficient $a_{j,j}$ peut avoir une valeur très proche de 0 (ce qui rend les calculs sensibles aux arrondis puisqu'il se trouve au dénominateur) voire peut être nul (et on sait tous qu'il ne faut pas diviser par zéro !).

Solution : on va choisir comme pivot non pas le coefficient $a_{j,j}$ mais le plus grand coefficient en valeur absolue parmi $a_{j,j}, \ldots, a_{n,j}$. Plus précisément, on va chercher l'indice $k \in [\![j \,;\, n]\!]$ tel que $|a_{k,j}|$ soit maximal puis, si $k \neq j$, on va échanger les lignes L_j et L_k. On appliquera alors l'idée précédente, cf $(*)$, avec cette valeur du pivot. Comme la matrice de départ est supposée inversible, on est certain que celui-ci est non nul et l'avoir choisi maximal permet de minimiser les problèmes liés aux arrondis.

On dit qu'on utilise la méthode de Gauss avec *pivot partiel*.

⇝ exercice 3

En répétant ce processus successivement sur toutes les colonnes, on obtiendra alors une matrice triangulaire. En pratique, il ne faut pas oublier que toutes les opérations effectuées sur la matrice A doivent l'être également sur le vecteur B pour conserver un système équivalent.

⇝ exercice 4

2.3 Résolution du système triangulaire

Comme vu dans l'exemple 1, il est très simple de résoudre un système triangulaire : il suffit de partir de la dernière ligne qui ne contient qu'une inconnue puis de remonter ligne par ligne.

Considérons le système $TX = C$ où T est une matrice triangulaire supérieure inversible, *i.e.*

$$\begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & \cdots & t_{1,n} \\ 0 & t_{2,2} & \cdots & \cdots & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & t_{n-1,n-1} & t_{n-1,n} \\ 0 & \cdots & \cdots & 0 & t_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} \iff \begin{cases} t_{1,1}x_1 + t_{1,2}x_2 + \ldots + t_{1,n}x_n = c_1 \\ \qquad\qquad\qquad\qquad\qquad \vdots \\ t_{n-1,n-1}x_{n-1} + t_{n-1,n}x_n = c_{n-1} \\ t_{n,n}x_n = c_n \end{cases}.$$

En partant de la dernière ligne, on obtient alors

$$x_n = \frac{c_n}{t_{n,n}} \quad \text{et} \quad \forall i \in [\![1 \,;\, n-1]\!],\ x_i = \frac{1}{t_{i,i}} \left(c_i - \sum_{k=i+1}^{n} t_{i,k} x_k \right),$$

en prenant soin de calculer x_{n-1} puis x_{n-2}, etc. jusqu'à x_1.

⇝ exercice 5

3 Applications

3.1 Calcul du déterminant

Pour calculer le déterminant d'une matrice grâce à l'algorithme de Gauss, il suffit de remarquer que :

— le déterminant d'une matrice triangulaire est égal au produit de ses termes diagonaux,
— un échange de deux lignes d'une matrice change le signe du déterminant,
— une opération élémentaire de la forme $L_i \leftarrow L_i + \alpha L_j$ avec $i \neq j$ ne modifie pas la valeur du déterminant.

Ainsi, comme la méthode de Gauss permet de rendre la matrice de départ triangulaire supérieure, il suffit de compter le nombre d'échanges de lignes effectués au cours du processus, on a alors le déterminant grâce à ce signe et les valeurs des coefficients diagonaux.

⇝ exercice 6

3.2 Calcul de l'inverse

Vous savez que l'algorithme de Gauss permet de calculer l'inverse d'une matrice A en effectuant les mêmes opérations sur la matrice identité. On s'arrête lorsque A a été mise sous forme *échelonnée réduite* (c'est-à-dire l'identité si A était inversible) et alors l'identité a été transformée en la matrice A^{-1}.

Précédemment, pour résoudre un système, on se contentait de rendre la matrice triangulaire supérieure. Pour l'inverser, on doit donc prolonger les calculs en mettant les termes diagonaux égaux à 1 et en faisant aussi apparaître des 0 partout au-dessus de la diagonale.

⇝ exercice 7

Avec une bibliothèque

Pour finir, même si ce n'est pas à savoir, il existe bien sûr des bibliothèques dans lesquelles toutes ces fonctions sont prédéfinies. Il s'agit du sous-module `linalg` (pour *Linear Algebra*) de `numpy`, ce dernier servant à définir des tableaux, type `array`, qui représente les matrices. En voici un exemple :

```
>>> import numpy as np  # Pour les tableaux = matrices
>>> import numpy.linalg as la  # Pour l'algèbre linéaire

# Création d'une matrice carrée et d'un vecteur colonne
>>> A = np.array([[1, 2, 3], [1, 2, 1], [3, 1, 1]])
>>> B = np.array([[4], [5], [6]])

# Pour résoudre le système AX = B :
>> la.solve(A, B)
array([[ 1.5], [ 2. ], [-0.5]])

# Pour calculer l'inverse de la matrice A :
>>> la.inv(A)
array([[-0.1, -0.1,  0.4],
       [-0.2,  0.8, -0.2],
       [ 0.5, -0.5,  0. ]])

# Pour calculer le déterminant de la matrice A :
>>> la.det(A)
-10.000000000000002
```

Nous reverrons ça en maths lorsque l'on préparera les oraux[2].

2. Pour les plus curieux, voir ce notebook : `https://capytale2.ac-paris.fr/web/c/127d-506161`

TD n° 9 : méthode de Gauss

Dans tout ce TD, on représentera une matrice carrée de taille n comme une liste de n listes de taille n (les lignes). De même on représentera un vecteur colonne comme une matrice à n lignes et une seule colonne, *i.e.* sous forme d'une liste de n listes de taille 1.

De plus, pour se conformer aux indices des listes en python, on numérotera les lignes et les colonnes de 0 à $n-1$ pour une matrice de taille n.

Exercice 1. *Création de matrices et de vecteurs*

1. Écrire une fonction `identite(n: int)` qui renvoie la matrice I_n. *On pourra commencer par initialiser un tableau de* 0 *de taille* $n \times n$.
2. Écrire une fonction `mat_alea(ligne: int, colonne: int, mini: float, maxi: float)` qui renvoie une matrice de taille `ligne`×`colonne` dont les coefficients sont choisis aléatoirement dans l'intervalle [`mini` ; `maxi`].
   Pour cela, on utilisera la fonction `uniform` *du module* `random` dont voici l'aide :
   ```
   >>> help(rd.uniform)
   uniform(a, b) method of random
   Get a random number in the range [a, b].
   ```
3. Donner la commande permettant d'obtenir une matrice aléatoire carrée de taille 3 dont les coefficients sont compris entre -10 et 10.
4. Même question pour un vecteur colonne de taille 4 avec des coefficients compris entre -5 et 5.

Exercice 2. *Opérations élémentaires*

1. Écrire deux fonctions `nb_lignes(A: [[float]])` et `nb_colonnes(A: [[float]])` qui renvoient respectivement le nombre de lignes et le nombre de colonnes de la matrice `A` donnée en argument.
2. Écrire une fonction `echange_lignes(A: [[float]], i: int, j: int)` qui modifie la matrice `A` en échangeant ses lignes `i` et `j`.
3. Écrire une fonction `combi_lignes(A: [[float]], i: int, j: int, alpha: float)` qui modifie la matrice `A` selon l'opération $L_i \leftarrow L_i + \alpha L_j$. On ajoutera une assertion pour s'assurer que $i \neq j$.
4. Donner la complexité de chacune de ces fonctions en fonction des dimensions de la matrice A.

Exercice 3. *Recherche du meilleur pivot*

Écrire une fonction `meilleur_pivot(A: [[float]], j: int)` qui prend en arguments une matrice `A` et un entier `j`, et qui renvoie l'indice de la ligne où se trouve le maximum en valeur absolue parmi les coefficients de la colonne `j` de `A` situés sous la diagonale, celle-ci comprise.
La valeur absolue est calculée grâce à la fonction `abs`.

Exercice 4. *Rendre la matrice triangulaire*

1. On considère le code suivant.
   ```
   1 L = [1, 2, 3]
   2 M = L
   3 M[0] = 5
   ```
   Après l'exécution de ces trois commandes, que vaut `L` ? que vaut `M` ?
2. Comment remédier à ce problème ?

On considère maintenant la fonction suivante.

```
1  from copy import deepcopy
2
3  def foo(A: [[float]], B: [[float]]) -> ([[float]], [[float]]):
4      # Pour ne pas modifier A et B, on les copie et on travaille sur
5      # ces copies ensuite.
6      T = deepcopy(A)
7      C = deepcopy(B)
8      n = nb_colonnes(T)
9      for j in range(n):
10         ind_pivot = meilleur_pivot(T, j)
11         val_pivot = T[ind_pivot][j]
12         echange_lignes(T, j, ind_pivot)
13         echange_lignes(C, j, ind_pivot)
14         for i in range(j+1, n):
15             coeff = -T[i][j] / val_pivot
16             combi_lignes(T, i, j, coeff)
17             combi_lignes(C, i, j, coeff)
18     return T, C
```

3. Expliquer les lignes 10 à 13.
4. Quelle est la forme de la matrice `T` avant et après de l'exécution de la boucle `for` de la ligne 14.
5. Quel est le but de cette fonction ?
6. Donner sa complexité en fonction de la taille de la matrice `A`.

Exercice 5. *Résolution d'un système triangulaire et d'un système quelconque*

1. Écrire une fonction `resol_tri(T: [[float]], C: [[float]])` qui prend en arguments une matrice triangulaire (supposée inversible) `T` et un vecteur `C`, et qui renvoie la solution du système $TX = C$.
2. En déduire une fonction `resol(A: [[float]], B: [[float]])` qui prend en arguments une matrice (supposée inversible) `A` et un vecteur `B`, et qui renvoie la solution du système $AX = B$ via la méthode du pivot de Gauss.

Exercice 6. *Calcul du déterminant*
En adaptant la fonction donnée dans l'exercice 4, écrire une fonction `det(A: [[float]])` qui renvoie la valeur du déterminant de la matrice donnée en argument.

Exercice 7. ★ *Inverser une matrice*
En s'inspirant du travail effectué dans l'exercice 4, écrire une fonction `inverse(A: [[float]])` qui renvoie la matrice A^{-1} (A est supposée inversible).

Pour finir, voici un extrait d'un sujet de concours qui abordait la méthode de Gauss.

Exercice 8. *Extrait de CCINP PSI 2022*
La résolution d'un système linéaire peut se faire à l'aide de l'algorithme du pivot partiel de Gauss.
La fonction `resolution` prend en arguments un tableau `R` de dimension $n \times n$ de type flottant et un vecteur à n éléments de type flottant. Cette fonction renvoie un vecteur à n éléments solution de l'équation $RX = Y$.
Les fonctions `combinaison` et `echanger_lignes` permettent respectivement de faire une combinaison linéaire entre deux lignes du système et d'échanger deux lignes du système.

```
1  def resolution(R: [[float]], Y: [float]) -> [float]:
2      syst = []
3      nb = len(R)
4      for k in range(nb):
5          ligne = []
6          for j in range(nb):
7              ligne.append(R[k][j])
8          ligne.append(Y[k])
9          syst.append(ligne)
10     # Mise sous forme forme triangulaire
11     for i in range(nb):
12         j = pivot(syst, i)
13         echanger_lignes(syst, i, j)
14
15         for k in range(i+1, nb):
16             # instruction à compléter
17             combinaison(syst, k, i, mu)
18     # Remontée
19     X = [0]*nb
20
21     # instructions à compléter
22
23     return X
```

1. Indiquer l'intérêt des lignes 2 à 9. Quelle est la taille du tableau `Syst` après l'exécution de ces lignes ?
2. À la ligne 12, la fonction `resolution` fait appel à la fonction `pivot` qui prend en arguments le tableau `syst` et un indice `i`. Expliquer en quoi il est pertinent d'appliquer cet algorithme avec le pivot le plus grand en valeur absolue. Écrire la fonction `pivot` répondant au problème.
3. La fonction `combinaison` prend en arguments un tableau `syst`, deux entiers `k` et `i` et un flottant `mu`. Cette fonction modifie la ligne `k` du tableau `syst` en réalisant l'opération :

$$\text{Ligne } \texttt{k} = \text{Ligne } \texttt{k} + \texttt{mu} \times \text{Ligne } \texttt{i}.$$

   Compléter la ligne 16 du code proposé en définissant la variable `mu`. Écrire une fonction `combinaison` qui réalise l'opération souhaitée.

Une fois le problème mis sous forme triangulaire, il reste à créer le vecteur des solutions `X`. C'est la phase de remontée.

4. Proposer les instructions permettant de construire le vecteur `X` à partir du système triangulaire.
5. En supposant une matrice A et un vecteur B définis, écrire l'instruction permettant de résoudre le système $AU = B$ et d'affecter le résultat dans une variable `U`.