T.P. numéro VI

Manipulation de matrices. Résolution de systèmes linéaires

1 Point de vue mathématique

1.1 Position du problème

Le calcul matriciel se révèle être un allié puissant dans de nombreux problèmes de physique. Parmi les applications de cet outil, on trouve la résolution de systèmes linéaires. Ce problème a été abordé lors du T.P. I à propos d'un système très simple de deux équations à deux inconnues. On se propose de généraliser le traitement à des systèmes linéaires de n équations à n inconnues. Soit le système suivant :

$$(S): \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 & \longleftarrow L_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 & \longleftarrow L_2 \\ \quad\vdots \quad\quad \vdots \quad\quad \cdots \quad\quad \vdots \quad = \vdots & \\ a_{n1}x_1 + a_{n2}x_n + \cdots + a_{nn}x_n = b_n & \longleftarrow L_n \end{cases}$$

où la solution recherché est un vecteur $(x_1, x_2, \cdots, x_n)$, les paramètres du problème étant les coefficients a_{ij} et b_i.
L'étude du système linéaire du T.P. I nous a montré qu'il fallait être prudent dans le choix de la méthode de traitement, en effet, nous ne sommes pas intéressés par le traitement d'un seul système, mais par la construction d'un programme permettant de résoudre un grand nombre de systèmes linéaires différents (un code permettant la résolution de n'importe quel système linéaire serait l'idéal). Nous recherchons donc une méthode robuste et performante dans un très grand nombre de cas.

1.2 Méthode du pivot de Gauss

La résolution d'un système linéaire consiste à transformer le système de départ en un système *équivalent* plus simple à résoudre. Les opérations suivantes transforment un système en un autre système équivalent :

– Échange de deux lignes $\quad L_i \longleftrightarrow L_j$
– Combinaison linéaire de deux lignes $\quad L_j \longrightarrow L_j + \lambda L_i$

La méthode du pivot de Gauss permet de résoudre le système linéaire de façon systèmatique et est donc très facilement adaptable à un traitement par l'ordinateur. Le but est d'obtenir un système équivalent sous forme triangulaire :

$$(S): \begin{cases} \alpha_{11}x_1 + \alpha_{12}x_2 + \cdots + \alpha_{1n}x_n = \beta_1 \\ \qquad\quad \alpha_{22}x_2 + \cdots + \alpha_{2n}x_n = \beta_2 \\ \qquad\qquad\quad \ddots \quad\quad \vdots \quad = \vdots \\ \qquad\qquad\qquad\quad \alpha_{nn}x_n = \beta_n \end{cases}$$

que l'on peut ensuite résoudre aisément (si le système n'est pas singulier) en « remontant » : on détermine à partir de la dernière équation l'inconnue $x_n = \beta_n/\alpha_{nn}$, puis successivement toutes les autres inconnues x_{n-1}, x_{n-2}, ... car connaissant les inconnues x_{i+1}, ..., x_n, on obtient x_i comme solution d'une simple équation linéaire à une inconnue.

Principe de la méthode Supposons que le système ait une forme triangulaire sur les $i-1$ premières lignes. Il reste donc le sous-système $(n-i+1)\times(n-i+1)$:

$$\begin{cases} a_{ii}x_i + \cdots + a_{in}x_n = b_{i+1} \\ \vdots \quad \cdots \quad \vdots = \vdots \\ a_{ni}x_i + \cdots + a_{nn}x_n = b_n \end{cases}$$

1. Par permutation de lignes, on fait en sorte que $a_{ii} \neq 0$: $L_i \longleftrightarrow L_j$ avec $a_{ji} \neq 0$.
   Si tous les a_{ji} sont nuls pour j allant de i à n, on a en fait un sous-système $(n-i)\times(n-i)$ pour les inconnues x_{i+1}, ... x_n.
2. On ajoute à toutes les lignes j pour j allant de $i+1$ à n la ligne i multipliée par $-a_{ji}/a_{ii}$: $L_j \longleftrightarrow L_j - (a_{ji}/a_{ii})L_i$.
   Ainsi on annule le coefficient devant x_i pour les lignes j allant de $i+1$ à n.

Après ces deux opérations on a obtenu un système sous forme triangulaire sur les i premières lignes et il reste un sous-système $(n-i)\times(n-i)$. Il suffit donc, pour mettre le système de départ sous forme triangulaire, de répéter ces deux étapes pour i allant de 1 à $n-1$.
Une fois sous forme triangulaire, on peut voir aisément si le système est singulier :
– si aucun des éléments diagonaux α_{ii} n'est nul, le système admet un vecteur solution unique ;
– sinon, le système est singulier.

2 Point de vue programmation

Le système de départ est représenté par la matrice de dimension $n\times(n+1)$

$$\begin{pmatrix} a_{11} & .... & a_{1n} & b_1 \\ a_{21} & ... & a_{2n} & b_2 \\ ... & ... & ... & ... \\ a_{n1} & ... & a_{nn} & b_n \end{pmatrix}$$

qui est stockée en mémoire sous la forme d'un tableau `mat[N][N+1]`. Le vecteur solution (éventuel) sera stocké dans le tableau `sol[N]`.
Remarque : En C, les tableaux passés en argument à des fonctions le sont automatiquement *par adresse* (cf. T.P. IV), c'est-à-dire que l'on peut modifier les valeurs de ses éléments dans la fonction appelée.

Cahier des charges La résolution du système s'effectue en trois étapes :

1. mise sous forme triangulaire ;
2. vérification de l'existence d'un vecteur solution unique ;
3. « remontée » du système

En pratique, le codage doit donc faire intervenir les fonctions de base suivantes définies par l'utilisateur :

Entrée-sortie

– EntreeSysteme() pour lire les paramètres du système ;
– AfficheSysteme() pour afficher le système à l'écran ;
– AfficheSolution () pour afficher le vecteur solution.

Opérations élémentaires sur les lignes

– EchangeLignes() pour échanger deux lignes de la matrice ;

– CombineLignes() pour faire la combinaison linéaire de deux lignes.

Méthode de Gauss

– Triangule () pour rendre la matrice triangulaire ;
– AdmetSolution() pour vérifier que l'on a affaire à un système admettant un vecteur solution unique ;
– Remonte() pour calculer les solutions à partir du système triangulé.

On se propose d'écrire un programme qui permette de résoudre les systèmes de dimension inférieure ou égale à 9. Ce programme sera écrit pas à pas en contrôlant chacune des étapes et appliqué, afin d'illustrer son bon fonctionnement, à la résolution du système 4×4 suivant :

$$\begin{cases} 2x_1 + 4x_2 + 2x_3 + x_4 = 3 \\ x_1 + 2x_2 + 4x_3 + 2x_4 = 1 \\ 3x_1 + 2x_2 + x_3 + 2x_4 = 0 \\ 4x_1 + x_2 + 3x_3 + 4x_4 = 2 \end{cases}$$

Exercice VI.1 *Entrée-sortie*

Télécharger le début de programme gauss.c .

1. Compléter les fonctions EntreeSysteme() et AfficheSysteme(). On fera afficher le système sous la forme :

```
Système :
|   2.0000*x1 +   4.0000*x2 +   2.0000*x3 +   1.0000*x4 =   3.0000
|   1.0000*x1 +   2.0000*x2 +   4.0000*x3 +   2.0000*x4 =   1.0000
|   3.0000*x1 +   2.0000*x2 +   1.0000*x3 +   2.0000*x4 =   0.0000
|   4.0000*x1 +   1.0000*x2 +   3.0000*x3 +   4.0000*x4 =   2.0000
```

2. Vérifier que l'affichage convienne aussi pour des systèmes d'autre dimension.

Remarque : Pour ne pas avoir à saisir les paramètres du système à chaque exécution du programme, on peut créer avec `emacs` un fichier `systeme` contenant :

```
4
2 4 2 1 3
1 2 4 2 1
3 2 1 2 0
4 1 3 4 2
```

Il suffira ensuite d'éxecuter le programme en tappant :

```
./a.out < systeme
```

Exercice VI.2 *Opérations élémentaires*

1. Ajouter au programme les fonctions EchangeLignes() et CombineLignes() avec leurs arguments.
2. Vérifier que le système obtenu en effectuant successivement la permutation $L_1 \longleftrightarrow L_4$ et la combinaison linéaire $L_4 \longrightarrow L_4 + (-2)L_2$ (en appelant les deux fonctions écrites au 1. dans le programme principal) est bien équivalent au système de départ.

Remarque : Il est possible de transférer dans un fichier `sortie` le résultat affiché à l'écran en exécutant :

```
./a.out < systeme > sortie
```

Exercice VI.3 *Triangularisation*

On exécutera le programme à chacune des étapes ci-dessous pour s'assurer d'obtenir les résultats escomptés.

1. Remplacer les appels aux fonctions EchangeLignes() et CombineLignes() du programme principal par un appel à la fonction Triangule ().
2. Écrire la fonction Triangule () en appliquant tout d'abord les opérations décrites dans le § « Principe de la méthode » pour $i = 1$:
   (a) déterminer la première ligne j ($j \geqslant i$) pour laquelle $a_{ij} \neq 0$;
   (b) effectuer la permutation des lignes i et j si nécessaire ;
   (c) réaliser les combinaisons linéaires pour annuler le coefficient en x_i de toutes les lignes j ($j \geqslant i$) si nécessaire.
3. Faire en sorte que la fonction Triangule () répète les opérations précédentes pour i allant de 1 à $n - 1$.

Exercice VI.4 *Résolution du système triangulaire*

1. Écrire la fonction AdmetSolution() qui doit retourner 1 si le système admet un vecteur solution unique et 0 dans le cas contraire.
2. Dans le programme principal, ajouter l'appel à la fonction AdmetSolution() et aux dernières fonctions Remonte() et AfficheSolution () si le système admet un vecteur solution unique.
   Afficher un message à l'écran si le système n'admet pas de vecteur solution unique.
3. Écrire la fonction Remonte() qui renvoie le vecteur solution dans le tableau `sol`.
   (a) Déterminer tout d'abord l'inconnue x_i à partir de la ligne L_i du système qui s'écrit :

$$\alpha_{ii}x_i + \sum_{j=i+1}^{n} \alpha_{ij}x_j = \beta_i$$

   en supposant toutes les inconnues x_j connues pour $j \geqslant i + 1$ et stockées dans le tableau `sol`.
   (b) Faire en sorte que la fonction Remonte() répète l'opération précédente pour i allant de n à 1.
4. Écrire la fonction AfficheSolution () et vérifier le résultat obtenu à l'exécution.
5. Vérifier le programme sur le système singulier :

$$\begin{cases} 2x_1 + 4x_2 = 4 \\ x_1 + 2x_2 = 1 \end{cases}$$

Listing des codes à télécharger

Code gauss.c

```c
#include <stdio.h>

#define N 10
int n;

void EntreeSysteme(double a[N][N+1]) {
  /* a compléter */
}

void AfficheSysteme(double a[N][N+1]) {
  /* a compléter */
}

int main() {
  double mat[N][N+1];
  double sol[N];
  int i;

  printf("Entrez la dimension du Système (<9)\n");
  scanf("%d",&n);
  for(i=1; i<=n; i++) sol[i] = 0;

  EntreeSysteme(mat);
  AfficheSysteme(mat);

}
```