UNIVERSITÉ Clermont Auvergne

Compte rendu : TP d'algorithmes numériques 1

HIAULT Lilian VALLET Baptiste

08 novembre 2019

Table des matières

Introduction

À l'occasion des travaux pratiques d'algorithmes numériques Hiault Lilian et Vallet Baptiste avons réalisé un programme en langage C qui permet de résoudre des systèmes linéaires grâce aux méthodes de Gauss et de Cholesky.

1 Rappel des méthodes de résolution d'équations linéaires

Les méthode de Gauss et de Cholesky permettent de résoudre des systèmes d'équation linéaires formé de plusieurs équations linéaires.

Par exemple :

$$\begin{cases} 2x + y = 5 \\ -x + 3y = 1 \end{cases}$$

On peut visualiser ce système d'inconnues x et y par des matrices de type $Ax = b$:

$$\begin{pmatrix} 2 & 1 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

Sous forme de matrice augmentée A :

$$A = \left(\begin{array}{cc|c} 2 & 1 & 5 \\ -1 & 3 & 1 \end{array}\right)$$

On utilise ces matrices pour résoudre les sytèmes d'équations linéaires.

1.1 Méthode de Gauss

1.1.1 Algorithme de Gauss

La méthode de Gauss permet de calculer des solutions exactes d'un système d'équation en un nombre d'itérations fini. Afin de résoudre un système, on triangularise la matrice. Pour cela on calcule le pivot de Gauss à chaque itération en transformant la premier coefficient non-nul de la ligne par un 1 grâce à des opérations élémentaires : échange de 2 lignes, multiplication d'une ligne par un scalaire non nul et ajout du multiple d'une ligne à une autre. Puis on utilise une suite d'opérations élémentaires sur les lignes suivantes jusqu'à obtenir un 0 sur toute la colonne sous chaque 1. On réitère cela pour chaque ligne jusqu'à obtenir une matrice échelonnée.

1.1.2 Exemple

À chaque étape on doit créer des 0 en dessous de la diagonale d'une matrice A jusqu'à obtenir une matrice A' échelonnée grâce à laquelle on pourra résoudre directement l'équation.

$$A = \left(\begin{array}{cc|c} 2 & 1 & 5 \\ -1 & 3 & 1 \end{array}\right)$$

$$A' = \left(\begin{array}{cc|c} 1 & \frac{1}{2} & \frac{5}{2} \\ 0 & 1 & 1 \end{array}\right)$$

On a donc :

$$\begin{cases} x + \frac{1}{2}y = \frac{5}{2} \\ y = 1 \end{cases} \iff \begin{cases} x = 2 \\ y = 1 \end{cases}$$

1.2 Méthode de Cholesky

1.2.1 Algorithme de Cholesky

On ne peut appliquer la méthode de Cholesky uniquement sur des matrices symétriques et définies positives. La méthode de Cholesky permet de décomposer une matrice A en deux matrices R et R^T tel que : $A = R^T R$ avec R une matrice triangulaire supérieure.

On a alors

$$R_{i,j} = \sqrt{a_{j,i} - \sum_{k=1}^{j-1} r_{j,k}^2}$$

1.2.2 Exemple

$$A = RR^T$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 \\ 1 & 5 & 14 & 14 \\ 1 & 5 & 14 & 15 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2 Présentation des programmes

2.1 Gauss

```
void gauss(double** tab, int taille,double* sys){
  int i, j, k, resteI; // i compte les lignes et j les colonnes
  i = 0;
  j = 0;
  k = 0;
  double pass = 0;
  // Trigonalisation
  while(j < taille){ // Changer de colonne
    i = k;
    resteI = 0;
    while(tab[i][j] == 0  && i < taille - 1){ // Changer de ligne dans une colonne
      i++;
    }
    if(tab[i][j] != 0){ // Si on trouve un pivot
      sys[i]=sys[i]/tab[i][j];
      diviseLigne(tab[i],taille,tab[i][j]);
      resteI = i + 1;
      while(resteI < taille){ // diviser chaque ligne
        sys[resteI] =sys[resteI]- (tab[resteI][j]/tab[i][j]) * sys[i];
        soustractionLigne(tab, taille, i, resteI, j);
        resteI++;
      }
      if (i != k){ // met le pivot en haut mais pas trop pour échelonner
        pass = sys[k];
        sys[k] = sys[i];
        sys[i] = pass ;
        echangeLigne(tab, taille, i, k);
      }
      k++;
    }
    else{
      //printf("Pas de pivot colonne %d\n", j+1);
    }
    j++;
  }
}
```

On résout un système du type $Ax = b$ où *tab* est la matrice A et *sys* est la matrice colonne b.

On utilise la fonction Gauss() une première fois pour trigonaliser la matrice. Puis, on applique renverseTab() et renverseSys() pour pouvoir réutiliser la fonction Gauss() et ainsi réduire au maximum la matrice. On réemploie renverseTab() et renverseSys() pour remettre le système à l'endroit. verifSol() vérifie enfin le nombre de solutions du système, et affiche le système simplifié.

2.2 Cholesky

```
void cholesky(double** tab,double** matR, int taille, double* sys) {
  double somme = 0;
  matR[0][0] = sqrt(tab[0][0]);
  for(int i =1; i<taille; i++){
    for(int j = 0; j<taille; j++){
      somme = 0;
      if(i==j){
        for(int k = 0; k<=j; k++){
          somme = somme + (matR[i][k]) * (matR[i][k]);
        }
        matR[i][i] = sqrt(tab[i][i] - somme);
      }
      else if(j>i){
        matR[i][j] = 0;
      }
      else{
        for(int k = 0; k<=j; k++){
          somme = somme + (matR[i][k]) * (matR[j][k]);
        }
        matR[i][j] = (1/matR[j][j]) * (tab[i][j] - somme);
      }
    }
  }
}
```

Cette fonction trouve la matrice R telle que $RR^T = A$. Elle applique les formules pour trouver les éléments de R en fonction de A et des éléments de R définis au fur et à mesure.

2.3 Autres fonctions et main

`void remplirTab(double** tab, int taille)`

Pour chaque position de la matrice (qui est rappelée), l'utilisateur entre une valeur.

`void afficheTab2D(double ** tab, int taille)`

Affiche une matrice carrée de taille "taille".

`void remplirSys(double* tab,int taille)`

Comme remplirTab mais pour un tableau à une dimension.

`void afficheSys(double* tab, int taille)`

Comme afficheTab2D mais pour un tableau à une dimension.

`void echangeLigne(double** tab, int taille, int nouv, int anc)`

Échange une ligne avec une autre d'un même tableau à deux dimensions en échangeant chaque variable avec celle de la même colonne mais d'une autre ligne grâce la variable *echange*.

`void diviseLigne(double* tab, int taille, double divi)`

Divise tous les membres d'une ligne par une valeur donnée.

`void renverseTab2D(double** tab,int taille)`

Cette fonction ne donne pas la transposée d'une matrice, elle inverse la position dans la ligne et elle inverse la position dans la colonne : i prend taille-i-1 et j prend taille -1-j. Cette fonction permet de faire repasser la matrice dans gauss pour la réduire au maximum.

`void renverseSys(double* sys, int taille)`

Inverse la position de chaque valeur.

`void verifSol(double** tab, double* sys,int taille)`

Cette fonction vérifie si le système admet une, plusieurs ou aucune solution. Pour ce faire, tant qu'il n'est pas insoluble, elle compte le nombre de coefficients nuls, si il la ligne est constituée exclusivement de zéros, alors on vérifie le résultat, si il est non nul, le système n'admet aucune solution, sinon, il y a une solution supplémentaire on passe à la prochaine ligne. Si il y a exactement un zéro, le nombre de solutions reste le même.

`void transposeTri(double** matR, int taille)`

C'est une fonction qui calcule la transposée d'une matrice triangulaire inférieure. Elle parcourt toute la matrice, les valeurs sont mises à leur place transposée puis annulées pour que le résultat soit une matrice triangulaire supérieure.

`void resoudCho(double** matR,int taille, double* sys, int transpose)`

Cette fonction résoud le système d'équation, si transposée = 0 alors la matrice entrée est L sinon c'est Lt.De plus elle crée un tableau x qui prend les nouvelles valeurs des résultats des lignes. La fonction résoud le système similairement à gauss, de haut en bas pour L et inversement pour Lt. Puis le tableau des résultats prend toutes ses nouvelles valeurs.

`int main()`

Création des tableaux dynamiquement par taille définie par l'utilisateur Utilisation de remplirTab et de remplirSys pour constituer le système à la main.

-Gauss : Utilisation de la fonction gauss une première fois pour trigonaliser la matrice puis renverseTab et renverseSys pour pouvoir réutiliser la fonction gauss et ainsi réduire au maximum la matrice, puis on réemploie renverseTab et renverseSys pour remettre le système à l'endroit. verifSol vérifie le nombre de solutions du système, et affiche le système simplifié.

-Cholesky : Utilisation de la fonction cholesky sur le système pour obtenir L, à partir duquel resoudCho détermine le y de L.y = b. Puis transposeTri détermine Lt, resoudCho calcule le système final.

3 Jeux d'essais

3.1 Résultats

Les matrices utilisées sont symétriques définies positives, elles peuvent donc êtres résolues par la méthode de Cholesky.

Taille de la matrice	Hilbert1	Hilbert2	Lehmer	Moler
10	5.6×10^{-4}	5.5×10^{-4}	5.1×10^{-4}	4.0×10^{-4}
100	2.9×10^{-2}	2.9×10^{-2}	2.4×10^{-2}	2.4×10^{-2}
1000	1.2×10^{1}	1.3×10^{1}	1.2×10^{1}	1.2×10^{1}

TABLE 1 – Tableau représentant la vitesse d'exécution en secondes de la méthode de Gauss en fonction de la taille de la matrice

Taille de la matrice	Hilbert1	Hilbert2	Lehmer	Moler
10	2.0×10^{-4}	1.9×10^{-4}	1.9×10^{-4}	2.0×10^{-4}
100	3.4×10^{-3}	2.9×10^{-3}	3.3×10^{-3}	3.0×10^{-3}
1000	2.0	2.0	2.1	2.1

TABLE 2 – Tableau représentant la vitesse d'exécution en secondes de la méthode de Cholesky en fonction de la taille de la matrice

3.2 Commentaire des jeux d'essais

Pour une matrice de taille 10, la méthode de Cholesky est 2.5 fois plus rapide que celle de Gauss (2.0×10^{-4}s contre 5.1×10^{-4}s).

Pour une matrice de taille 100, la méthode de Cholesky est 8.4 fois plus rapide que celle de Gauss (3.15×10^{-3}s contre 2.65×10^{-2}s).

Pour une matrice de taille 1000, la méthode de Cholesky est 10 fois plus rapide que celle de Gauss (2.1s contre 2.2×10^{1}s).

4 Conclusion générale sur les méthodes

La méthode de résolution d'équations linéaires de Cholesky est beaucoup plus rapide que celle de Gauss et plus la taille des équations à résoudre augmente, plus elle l'est.

Il semble donc plus intéressant d'utiliser la méthode de Cholesky. Toutefois, celle-ci ne fonctionne que sur des matrices symétriques définies positive alors que la méthode de Gauss fonctionne sur toutes les matrices. La méthode de Gauss est donc moins rapide mais plus générale : on peut toujours l'utiliser.