Algorithmes numériques – Rapport

Résolution de systèmes linéaires

Axel Delsol, Pierre-Loup Pissavy

Novembre 2013

Table des matières

Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

1 Structure globale du programme

Nous avons choisi de générer un programme principal avec menus, présenté sous la forme suivante :

```
MENU PRINCIPAL : RESOLUTION D'EQUATIONS LINEAIRES

1. Résolution par Gauss
3. Résolution par Cholesky
2. Résolution par Jacobi
4. Résolution par Gauss-Seidel
5. Résolution par Surrelaxation
6. Génération de matrices carrées
7. Jeux de test
0. Quitter
Votre choix :
```

FIGURE 1.1 – Apercu : Menu Principal

¡¡¡¡¡¡¡ HEAD Tous les fonctions de résolution font appel à des fonctions adaptées aux matrices, écrites dans le fichier `matrices.c`.

Chaque méthode de résolution reçoit les matrices (à éléments réels) générées auparavant (juste après le choix de la méthode dans le menu) en arguments.

=======

Chaque entrée du menu est codée dans un fichier source qui lui est propre (cf Figure **??**). Les fichiers headers correspondants contiennent les prototypes. Le fichier source `main.c` contient le menu principal.

Toutes les fonctions de résolution font appel à des fonctions intermédiaires, caractéristiques du calcul matriciel, écrites dans le fichier `matrices.c`.

```
gauss.c
gauss-seidel.c
generateur.c
jacobi.c
main.c
matrices.c
surrelaxation.c
cholesky.h
gauss.h
gauss-seidel.h
generateur.h
jacobi.h
matrices.h
surrelaxation.h
makefile
```

FIGURE 1.2 – Apercu : Arborescence

Les matrices sont générées juste après le choix de la méthode, avant d'être passées en arguments à la fonction de résolution.

Enfin, la compilation est gérée par un `makefile`. ¿¿¿¿¿¿¿ 8e40838088fc6d96da302cd16076aafc3c33c838

2 Méthodes directes

Les méthodes directes ont pour but d'obtenir des solutions exactes en simplifiant, étape par étape, le système donné sous la forme $A_{n,n} \cdot X_{n,1} = B_{n,1}$ où :

- A est une matrice donnée
- X est le vecteur solution
- B est un vecteur colonne donné

Pour ce faire, on utilise 2 étapes :

1. On décompose la matrice $A_{n,n}$ tel que $A = M \cdot N$ où $M_{n,n}$ est facile à inverser et $N_{n,n}$ est triangulaire.
2. On résoud les systèmes suivants :
   - On trouve $Y_{n,1}$ tel que $M \cdot Y = B$
   - On trouve enfin $X_{n,1}$ tel quel $N \cdot X = Y$

2.1 Méthode de Gauss

La méthode de Gauss permet de calculer une solution exacte en un nombre fini d'étapes.
On cherche la matrice N triangulaire supérieure telle que $A = M \cdot N$ avec M la matrice identité.
Remarque : La résolution du sysème $M \cdot Y = B$ est évidente puisque la matrice M est la matrice identité.
Critère d'application de l'algorithme :

- Les éléments diagonaux ne peuvent être nuls,
- Le déterminant ne doit pas être nul.

2.1.1 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include "matrices.h"

void gauss (double** a, double** b, int n)
{
  //initialisation
  int i, j, k;
  double pivot;

  //gauss
  for (k=0; k<n-1; k++)
  {
    for (i=k+1; i<n; i++)
    {
      pivot = a[i][k]/a[k][k];
      for (j=k+1; j<n; j++)
      {
        a[i][j]=a[i][j]-pivot*a[k][j];
      }
      b[i][0]=b[i][0]-pivot*b[k][0];
    }
    printf("\nMatrice :\n");
    afficherMatrice(a,n,n);
    printf("Vecteur :\n");
    afficherMatrice(b,n,1);
  }
```

```c

  //calcul et affichage resultat
  double** x=solveTriangulaireSup(a, b, n);
  printf("\nRésultat :\n");
  afficherMatrice(x, n, 1);

  //libération mémoire
  for (i=0;i<n;i++)
  {
    free(x[i]);
  }
  free(x);
}
```

FIGURE 2.1 – Code : gauss.c

2.1.2 Jeux de tests

Systèmes :

$$\begin{pmatrix} 2 & 4 & -2 & 0 \\ 1 & 2 & 0 & 1 \\ 3 & -1 & 1 & 2 \\ 0 & -1 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -6 \\ 0 \\ 8 \\ 6 \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 7 \\ 9 \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 3 & 9 & 27 \\ 2 & 4 & 8 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 14 \\ 120 \\ 50 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} 2 & 0 & 4 & -2 \\ 1 & 0 & 2 & 1 \\ 0 & 1 & 0 & 2 \\ 1 & 3 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 2 \\ 1 \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 3 & 9 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 2 \\ 9 \end{pmatrix} \tag{5}$$

2.2 Méthode de Cholesky

La méthode de Cholesky peut-être divisée en 2 étapes.

1. On décompose la matrice

2.2.1 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrices.h"

void cholesky (double ** a, double ** b, int n)
{
  int i, j, k;
  double **r = (double**) malloc(n*sizeof(double*));
  for (i=0; i<n; i++)
  {
    r[i]=(double*) malloc(n*sizeof(double));
  }

  /*Calcul de R*/
  for (k=0; k<n; k++)
  {
    for (i=k; i<n; i++)
    {
      double somme=0;
      if(k==i)
      {
        for (j=0; j<k; j++)
        {
          somme=somme+r[i][j]*r[i][j]; //somme des carrés
        }
        r[i][k]=sqrt(a[i][k]-somme);
      }
      else
      {
        for (j=0; j<k; j++)
        {
          somme=somme+r[i][j]*r[k][j];
        }
        r[i][k]=(1.0/r[k][k])*(a[i][k]-somme);
      }
    }
  }
  printf("Matrice R :\n");
  afficherMatrice(r,n,n);
  printf("Matrice Rt :\n");
  afficherMatrice(transpose(r,n),n,n);

  /*Résolution de Ry=b*/
  double** y=solveTriangulaireInf(r,b,n);

  /*Résolution de tRx=y*/
  double** x=solveTriangulaireSup(transpose(r,n),y,n);

  /*Affichage de x*/
  printf("\nVecteur résultat :\n");
  afficherMatrice(x,n,1);
}
```

FIGURE 2.2 – Code : cholesky.c

2.2.2 Jeux de tests

$$\begin{pmatrix} 64 & 40 & 24 \\ 40 & 29 & 17 \\ 24 & 17 & 19 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{6}$$

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ -1 \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 \\ 1 & 5 & 14 & 14 \\ 1 & 5 & 14 & 15 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 6 \end{pmatrix} \tag{8}$$

$$\begin{pmatrix} 1 & -2 & 0 \\ -2 & 8 & -6 \\ 0 & -6 & 25 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3/2 \\ 5/3 \\ 3/4 \end{pmatrix} \tag{9}$$

$$\begin{pmatrix} 4 & 0 & 12 & -6 \\ 0 & 1 & 2 & 1 \\ 12 & 2 & 49 & -4 \\ -6 & 1 & -4 & 51 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{10}$$

2.3 Comparaison

¡¡¡¡¡¡¡ HEAD

3 Méthodes itératives

3.1 Méthode de Jacobi

3.2 Métode de Gauss-Seidle

=======

4 Méthodes itératives

4.1 Méthode de Jacobi

4.1.1 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrices.h"

void jacobi (double** a, double** b, double** xInit, int n, double prec)
{
  int i, j, cpt;
  double residu=2*prec;
  double** xNext= xInit;
  cpt=0;
  while (residu>=prec)
  {
    for (i=0; i<n; i++)
    {
      double somme1=0, somme2=0;
      for (j=0; j<i; j++) //avant l'élément diagonal
      {
        somme1=somme1+a[i][j]*xNext[j][0];
      }
      for (j=i+1; j<n; j++)
      {
        somme2=somme2+a[i][j]*xNext[j][0]; //après l'élément diagonal
      }
      xNext[i][0]=(1/a[i][i])*(b[i][0]-somme1-somme2);
    }
    double** ax=produitMatriciel(a, xNext, n, n, 1); //ax
    double** axb=difference(ax, b, n, 1); //ax - b
    residu=norme(axb, n); //norme de (ax - b)
    cpt++;

    //affichage
    printf("\nVecteur àl'itération %d :\n", cpt);
    afficherMatrice(xNext, n, 1);

    //libération mémoire
    for (i=0;i<n;i++)
    {
      free(ax[i]);
      free(axb[i]);
      free(xNext[i]);
    }
    free(ax);
    free(axb);
    free(xNext);
  }
}
```

FIGURE 4.1 – Code : jacobi.c

4.1.2 Jeux de tests

4.2 Méthode de Gauss-Seidel

4.2.1 Programme

4.2.2 Jeux de tests

4.3 Méthode de Sur-relaxation

4.3.1 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrices.h"
#include "gauss-seidel.h"

void surrelaxation (double** a, double** b, double** xInit, int n, double prec, double ohm)
{
  if (ohm==1)
  {
    gaussseidel(a,b,xInit,n,prec);
  }
  else
  {
    int i, j, cpt;
    double residu=2*prec;
    cpt=0;
    while (residu>=prec)
    {
      for (i=0; i<n; i++)
      {
        double somme1=0, somme2=0;
        for (j=0; j<i; j++)
        {
          somme1=somme1+a[i][j]*xInit[j][0];
        }
        for (j=i+1; j<n; j++)
        {
          somme2=somme2+a[i][j]*xInit[j][0];
        }
        xInit[i][0]=(1-ohm)*xInit[i][0]+(ohm/a[i][i])*(b[i][0]-somme1-somme2);
      }
      double** ax=produitMatriciel(a, xInit, n, n, 1);
      double** axb=difference(ax, b, n, 1);
      residu=norme(axb, n);
      cpt++;

      //affichage
      printf("\nVecteur àl'itération %d :\n", cpt);
      afficherMatrice(xInit, n, 1);

      //libération mémoire
      for (i=0;i<n;i++)
      {
        free(ax[i]);
        free(axb[i]);
      }
      free(ax);
      free(axb);
    }
  }
}
```

FIGURE 4.2 – Code : surrelaxation.c

4.3.2 Jeux de tests

4.4 Comparaison

¿¿¿¿¿¿¿ 8e40838088fc6d96da302cd16076aafc3c33c838

Conclusion