Algorithmes numériques – Rapport

Résolution de systèmes linéaires

Axel Delsol, Pierre-Loup Pissavy

Novembre 2013

Table des matières

1 Préambule

1.1 Structure 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

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 1.2). 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`.

1.2 Compilation

1.3 Systèmes testés

Les matrices suivantes sont toutes symétriques définies positives à diagonale dominante. Nous avons choisi de prendre 10 systèmes pouvant être résolus par toutes les méthodes pour faciliter la comparaison des méthodes directes, itératives et directes-intermédiaires.

Systèmes :

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

$$\begin{pmatrix} 19 & 0 & 12 & -6 \\ 0 & 4 & 2 & 1 \\ 12 & 2 & 49 & -4 \\ -6 & 1 & -4 & 51 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} 5 & 1 & 1 & 1 \\ 1 & 13 & 5 & 5 \\ 1 & 5 & 49 & 14 \\ 1 & 5 & 14 & 51 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 6 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} 1 & 0.5 & 0.25 \\ 0.5 & 1 & 0 \\ 0.25 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 8 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 7 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 6 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix} \tag{5}$$

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

La résolution se fait en 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 résout 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 système $M \cdot Y = B$ 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);
```

```
27    }
28
29    //calcul et affichage resultat
30    double** x=solveTriangulaireSup(a, b, n);
31    printf("\nRésultat :\n");
32    afficherMatrice(x, n, 1);
33
34    //libération mémoire
35    for (i=0;i<n;i++)
36    {
37      free(x[i]);
38    }
39    free(x);
40  }
```

FIGURE 2.1 – Code : gauss.c

2.1.2 Jeux de tests

2.1.3 Résultats des tests

Système	Résultat obtenu	Résultat théorique	Ecart
(1)	$\begin{pmatrix} 0.63415 \\ -0.26829 \\ 0.17073 \\ -0.58537 \end{pmatrix}$	$\begin{pmatrix} \frac{26}{41} \\ -\frac{11}{41} \\ \frac{7}{41} \\ -\frac{24}{41} \end{pmatrix}$	0
(2)	$\begin{pmatrix} 0.06132 \\ 0.24599 \\ -0.00287 \\ 0.02177 \end{pmatrix}$	$\begin{pmatrix} \frac{3067}{50015} \\ \frac{7382}{30009} \\ -\frac{431}{150045} \\ \frac{1089}{50015} \end{pmatrix}$	0
(3)	$\begin{pmatrix} -0.04885 \\ 0.10345 \\ 0.04458 \\ 0.09623 \end{pmatrix}$	$\begin{pmatrix} -\frac{1181}{20928} \\ \frac{563}{6976} \\ \frac{121}{1308} \\ \frac{95}{872} \end{pmatrix}$	0
(4)	$\begin{pmatrix} -0.36364 \\ 2.18182 \\ 1.09091 \end{pmatrix}$	$\begin{pmatrix} -\frac{4}{11} \\ \frac{24}{11} \\ \frac{12}{11} \end{pmatrix}$	0
(5)	$\begin{pmatrix} 1.00000 \\ -0.81145 \\ 1.27048 \\ -0.46016 \\ 1.56790 \\ 0.05780 \\ 2.05345 \\ 0.88005 \\ 1.86675 \\ 2.06675 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -\frac{241}{297} \\ \frac{1132}{891} \\ -\frac{410}{891} \\ \frac{127}{81} \\ \frac{103}{1782} \\ \frac{4879}{2376} \\ \frac{697}{792} \\ \frac{22177}{11880} \\ \frac{24553}{11880} \end{pmatrix}$	0

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 Résultats des tests

2.3 Comparaison

3 Méthodes itératives

3.1 Méthode de Jacobi

3.1.1 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 3.1 – Code : jacobi.c

3.1.2 Jeux de tests

Système	Résultat obtenu	Résultat théorique	Ecart
(1)	0	$\begin{pmatrix} \frac{26}{41} \\ -\frac{11}{41} \\ \frac{7}{41} \\ -\frac{24}{41} \end{pmatrix}$	0
(2)	0	$\begin{pmatrix} \frac{3067}{50015} \\ \frac{7382}{30009} \\ -\frac{431}{150045} \\ \frac{1089}{50015} \end{pmatrix}$	0
(3)	0	$\begin{pmatrix} -\frac{1181}{20928} \\ \frac{563}{6976} \\ \frac{121}{1308} \\ \frac{95}{872} \end{pmatrix}$	0
(4)	0	$\begin{pmatrix} -\frac{4}{11} \\ \frac{24}{11} \\ \frac{12}{11} \end{pmatrix}$	0
(5)	0	$\begin{pmatrix} 1 \\ -\frac{241}{297} \\ \frac{1132}{891} \\ -\frac{410}{891} \\ \frac{127}{81} \\ \frac{103}{1782} \\ \frac{4879}{2376} \\ \frac{697}{792} \\ \frac{22177}{11880} \\ \frac{24553}{11880} \end{pmatrix}$	0

3.1.3 Résultats des tests

3.2 Méthode de Gauss-Seidel

3.2.1 Programme

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

void gaussseidel (double** a, double** b, double** xInit, int n, double prec)
{
  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/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 3.2 – Code : gauss–seidel.c

3.2.2 Résultats de tests

3.3 Méthode de Sur-relaxation

3.3.1 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 3.3 – Code : surrelaxation.c

3.3.2 Résultats de tests

3.4 Comparaison

Conclusion