

Algorithmes numériques – Rapport

Résolution de systèmes linéaires

Axel Delsol, Pierre-Loup Pissavy

Novembre 2013

# Table des matières

<b>1</b>	<b>Structure globale du programme</b>	<b>3</b>
<b>2</b>	<b>Méthodes directes</b>	<b>4</b>
2.1	Méthode de Gauss . . . . .	4
2.1.1	Programme . . . . .	4
2.1.2	Jeux de tests . . . . .	5
2.2	Méthode de Cholesky . . . . .	6
2.2.1	Programme . . . . .	6
2.2.2	Jeux de tests . . . . .	7
2.3	Comparaison . . . . .	8
<b>3</b>	<b>Méthodes itératives</b>	<b>9</b>
3.1	Méthode de Jacobi . . . . .	9
3.1.1	Programme . . . . .	9
3.1.2	Jeux de tests . . . . .	10
3.2	Méthode de Gauss-Seidel . . . . .	11
3.2.1	Programme . . . . .	11
3.2.2	Jeux de tests . . . . .	12
3.3	Méthode de Sur-relaxation . . . . .	13
3.3.1	Programme . . . . .	13
3.3.2	Jeux de tests . . . . .	14
3.4	Comparaison . . . . .	15

# Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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, consectetur 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 consectetur.

# 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 – Aperçu : Menu Principal

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 – Aperçu : 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`.

## 2 Méthodes directes

### 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. Critère d'application de l'algorithme :

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

#### 2.1.1 Programme

```
1 | #include <stdio.h>
2 | #include <stdlib.h>
3 | #include "matrices.h"
4 |
5 | void gauss (double** a, double** b, int n)
6 | {
7 |     //initialisation
8 |     int i, j, k;
9 |     double pivot;
10 |
11 |     //gauss
12 |     for (k=0; k<n-1; k++)
13 |     {
14 |         for (i=k+1; i<n; i++)
15 |         {
16 |             pivot = a[i][k]/a[k][k];
17 |             for (j=k+1; j<n; j++)
18 |             {
19 |                 a[i][j]=a[i][j]-pivot*a[k][j];
20 |             }
21 |             b[i][0]=b[i][0]-pivot*b[k][0];
22 |         }
23 |         printf("\nMatrice :\n");
24 |         afficherMatrice(a,n,n);
25 |         printf("Vecteur :\n");
26 |         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

Systèmes :

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

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

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

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

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

## 2.2 Méthode de Cholesky

### 2.2.1 Programme

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrices.h"
5
6  void cholesky (double ** a, double ** b, int n)
7  {
8      int i, j, k;
9      double **r = (double**) malloc(n*sizeof(double*));
10     for (i=0; i<n; i++)
11     {
12         r[i]=(double*) malloc(n*sizeof(double));
13     }
14
15     /*Calcul de R*/
16     for (k=0; k<n; k++)
17     {
18         for (i=k; i<n; i++)
19         {
20             double somme=0;
21             if(k==i)
22             {
23                 for (j=0; j<k; j++)
24                 {
25                     somme=somme+r[i][j]*r[i][j]; //somme des carrés
26                 }
27                 r[i][k]=sqrt(a[i][k]-somme);
28             }
29             else
30             {
31                 for (j=0; j<k; j++)
32                 {
33                     somme=somme+r[i][j]*r[k][j];
34                 }
35                 r[i][k]=(1.0/r[k][k])*(a[i][k]-somme);
36             }
37         }
38     }
39     printf("Matrice R :\n");
40     afficherMatrice(r,n,n);
41     printf("Matrice Rt :\n");
42     afficherMatrice(transpose(r,n),n,n);
43
44     /*Résolution de Ry=b*/
45     double** y=solveTriangulaireInf(r,b,n);
46
47     /*Résolution de tRx=y*/
48     double** x=solveTriangulaireSup(transpose(r,n),y,n);
49
50     /*Affichage de x*/
51     printf("\nVecteur résultat :\n");
52     afficherMatrice(x,n,1);
53 }
```

FIGURE 2.2 – Code : cholesky.c

### 2.2.2 Jeux de tests

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

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

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

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

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



## 2.3 Comparaison

## 3 Méthodes itératives

### 3.1 Méthode de Jacobi

#### 3.1.1 Programme

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrices.h"
5
6  void jacobi (double** a, double** b, double** xInit, int n, double prec)
7  {
8      int i, j, cpt;
9      double residu=2*prec;
10     double** xNext= xInit;
11     cpt=0;
12     while (residu>=prec)
13     {
14         for (i=0; i<n; i++)
15         {
16             double somme1=0, somme2=0;
17             for (j=0; j<i; j++) //avant l'élément diagonal
18             {
19                 somme1=somme1+a[i][j]*xNext[j][0];
20             }
21             for (j=i+1; j<n; j++)
22             {
23                 somme2=somme2+a[i][j]*xNext[j][0]; //après l'élément diagonal
24             }
25             xNext[i][0]=(1/a[i][i])*(b[i][0]-somme1-somme2);
26         }
27         double** ax=produitMatriciel(a, xNext, n, n, 1); //ax
28         double** axb=difference(ax, b, n, 1); //ax - b
29         residu=norme(axb, n); //norme de (ax - b)
30         cpt++;
31
32         //affichage
33         printf("\nVecteur à l'itération %d :\n", cpt);
34         afficherMatrice(xNext, n, 1);
35
36         //libération mémoire
37         for (i=0; i<n; i++)
38         {
39             free(ax[i]);
40             free(axb[i]);
41             free(xNext[i]);
42         }
43         free(ax);
44         free(axb);
45         free(xNext);
46     }
47 }
```

FIGURE 3.1 – Code : jacobi.c

### 3.1.2 Jeux de tests

## 3.2 Méthode de Gauss-Seidel

### 3.2.1 Programme

### 3.2.2 Jeux de tests

## 3.3 Méthode de Sur-relaxation

### 3.3.1 Programme

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "matrices.h"
5
6  void surrelaxation (double** a, double** b, double** xInit, int n, double prec, double ohm)
7  {
8      int i, j, cpt;
9      double residu=2*prec;
10     cpt=0;
11     while (residu>=prec)
12     {
13         for (i=0; i<n; i++)
14         {
15             double somme1=0, somme2=0;
16             for (j=0; j<i; j++)
17             {
18                 somme1=somme1+a[i][j]*xInit[j][0];
19             }
20             for (j=i+1; j<n; j++)
21             {
22                 somme2=somme2+a[i][j]*xInit[j][0];
23             }
24             xInit[i][0]=(1-ohm)*xInit[i][0]+(ohm/a[i][i])*(b[i][0]-somme1-somme2);
25         }
26         double** ax=produitMatriciel(a, xInit, n, n, 1);
27         double** axb=difference(ax, b, n, 1);
28         residu=norme(axb, n);
29         cpt++;
30
31         //affichage
32         printf("\nVecteur à l'itération %d :\n", cpt);
33         afficherMatrice(xInit, n, 1);
34
35         //libération mémoire
36         for (i=0; i<n; i++)
37         {
38             free(ax[i]);
39             free(axb[i]);
40         }
41         free(ax);
42         free(axb);
43     }
44 }
```

FIGURE 3.2 – Code : surrelaxation.c

### 3.3.2 Jeux de tests

### 3.4 Comparaison



## Conclusion