

Algorithmes numériques – Rapport
Résolution de systèmes linéaires

Axel Delsol, Pierre-Loup Pissavy

Novembre 2013

Table des matières

1	Préambule	2
1.1	Structure du programme	2
1.2	Compilation	3
1.3	Systèmes testés	4
2	Méthodes directes	5
2.1	Méthode de Gauss	5
2.1.1	Programme	5
2.1.2	Résultats des tests	7
2.2	Méthode de Cholesky	8
2.2.1	Programme	8
2.2.2	Résultats des tests	10
2.3	Comparaison	11
3	Méthodes itératives	12
3.1	Méthode de Jacobi	12
3.1.1	Programme	12
3.1.2	Résultats des tests	14
3.2	Méthode de Gauss-Seidel	15
3.2.1	Programme	15
3.2.2	Résultats de tests	16
3.3	Méthode de Sur-relaxation	17
3.3.1	Programme	17
3.3.2	Résultats de tests	18
3.4	Comparaison	19

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 – Aperçu : 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 – 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`.

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 :

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

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

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

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

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

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 que $N \cdot X = Y$

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

```
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 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

```
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
54     // libération mémoire
55     for (i=0; i<n; i++)
56     {
57         free(r[i]);
58     }
59     free(r);
```

60 || }

FIGURE 2.2 – Code : cholesky.c

2.2.2 Résultats des 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

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= (double**) malloc(n*sizeof(double*));
11     double** ax;
12     double** axb;
13     cpt=0;
14     for (i=0; i<n; i++)
15     {
16         xNext[i]= (double*) malloc(sizeof(double));
17         for (j=0; j<n; j++)
18         {
19             xNext[i][j]=xInit[i][j];
20         }
21     }
22     while (residu>=prec)
23     {
24         for (i=0; i<n; i++)
25         {
26             double somme1=0, somme2=0;
27             for (j=0; j<i; j++) //avant l'élément diagonal
28             {
29                 somme1=somme1+a[i][j]*xNext[j][0];
30             }
31             for (j=i+1; j<n; j++)
32             {
33                 somme2=somme2+a[i][j]*xNext[j][0]; //après l'élément diagonal
34             }
35             xNext[i][0]=(1/a[i][i])*(b[i][0]-somme1-somme2);
36         }
37         ax=produitMatriciel(a, xNext, n, n, 1); //ax
38         axb=difference(ax, b, n, 1); //ax - b
39         residu=norme(axb, n); //norme de (ax - b)
40         cpt++;
41
42         //affichage
43         printf("\nVecteur à l'itération %d :\n", cpt);
44         afficherMatrice(xNext, n, 1);
45     }
46
47     //libération mémoire
48     for (i=0; i<n; i++)
49     {
50         free(ax[i]);
51         free(axb[i]);
```

```
52 |     free(xNext[i]);  
53 | }  
54 | free(ax);  
55 | free(axb);  
56 | free(xNext);  
57 | }
```

FIGURE 3.1 – Code : jacobi.c

3.1.2 Résultats des tests

Système	Résultat obtenu	Résultat théorique	Ecart	Précision	Convergence
(1)	0	$\begin{pmatrix} \frac{26}{41} \\ -\frac{11}{41} \\ \frac{7}{41} \\ -\frac{24}{41} \end{pmatrix}$	0	0	0
(2)	0	$\begin{pmatrix} \frac{3067}{50015} \\ \frac{7382}{30009} \\ -\frac{431}{150045} \\ \frac{1089}{50015} \end{pmatrix}$	0	0	0
(3)	0	$\begin{pmatrix} -\frac{1181}{20928} \\ \frac{563}{6976} \\ \frac{121}{1308} \\ \frac{95}{872} \end{pmatrix}$	0	0	0
(4)	0	$\begin{pmatrix} -\frac{4}{11} \\ \frac{24}{11} \\ \frac{12}{11} \end{pmatrix}$	0	0	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	0	0

3.2 Méthode de Gauss-Seidel

3.2.1 Programme

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

FIGURE 3.2 – Code : gauss-seidel.c

3.2.2 Résultats de tests

Système	Résultat obtenu	Résultat théorique	Ecart	Précision	Convergence
(1)	0	$\begin{pmatrix} \frac{26}{41} \\ -\frac{11}{41} \\ \frac{7}{41} \\ -\frac{24}{41} \end{pmatrix}$	0	0	0
(2)	0	$\begin{pmatrix} \frac{3067}{50015} \\ \frac{7382}{30009} \\ -\frac{431}{150045} \\ \frac{1089}{50015} \end{pmatrix}$	0	0	0
(3)	0	$\begin{pmatrix} -\frac{1181}{20928} \\ \frac{563}{6976} \\ \frac{121}{1308} \\ \frac{95}{872} \end{pmatrix}$	0	0	0
(4)	0	$\begin{pmatrix} -\frac{4}{11} \\ \frac{24}{11} \\ \frac{12}{11} \end{pmatrix}$	0	0	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	0	0

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  #include "gauss-seidel.h"
6
7  void surrelaxation (double** a, double** b, double** xInit, int n, double prec, double ohm)
8  {
9      if (ohm==1)
10     {
11         gaussseidel(a,b,xInit,n,prec);
12     }
13     else
14     {
15         int i, j, cpt;
16         double residu=2*prec;
17         double** ax;
18         double** axb;
19         cpt=0;
20         while (residu>=prec)
21         {
22             for (i=0; i<n; i++)
23             {
24                 double somme1=0, somme2=0;
25                 for (j=0; j<i; j++)
26                 {
27                     somme1=somme1+a[i][j]*xInit[j][0];
28                 }
29                 for (j=i+1; j<n; j++)
30                 {
31                     somme2=somme2+a[i][j]*xInit[j][0];
32                 }
33                 xInit[i][0]=(1-ohm)*xInit[i][0]+(ohm/a[i][i])*(b[i][0]-somme1-somme2);
34             }
35             ax=produitMatriciel(a, xInit, n, n, 1);
36             axb=difference(ax, b, n, 1);
37             residu=norme(axb, n);
38             cpt++;
39
40             //affichage
41             printf("\nVecteur à l'itération %d :\n", cpt);
42             afficherMatrice(xInit, n, 1);
43         }
44         //libération mémoire
45         for (i=0;i<n;i++)
46         {
47             free(ax[i]);
48             free(axb[i]);
49         }
50         free(ax);
51         free(axb);
52     }
53 }
```

FIGURE 3.3 – Code : surrelaxation.c

3.3.2 Résultats de tests

Système	Résultat obtenu	Résultat théorique	Ecart	Précision	Convergence
(1)	0	$\begin{pmatrix} \frac{26}{41} \\ -\frac{11}{41} \\ \frac{7}{41} \\ -\frac{24}{41} \end{pmatrix}$	0	0	0
(2)	0	$\begin{pmatrix} \frac{3067}{50015} \\ \frac{7382}{30009} \\ -\frac{431}{150045} \\ \frac{1089}{50015} \end{pmatrix}$	0	0	0
(3)	0	$\begin{pmatrix} -\frac{1181}{20928} \\ \frac{563}{6976} \\ \frac{121}{1308} \\ \frac{95}{872} \end{pmatrix}$	0	0	0
(4)	0	$\begin{pmatrix} -\frac{4}{11} \\ \frac{24}{11} \\ \frac{12}{11} \end{pmatrix}$	0	0	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	0	0

3.4 Comparaison

Conclusion