

Application en Ingénierie et Programmation Numérique  
*"Rendu I - Méthodes Directes"*

VILLEDIEU Maxance et BESQUEUT Corentin

1 octobre 2023

# Table des matières

<b>I</b>	<b>Résolution de systèmes linéaires par des Méthodes Directes : Méthode de Gauss</b>	<b>2</b>
I.1	Détail de l'algorithme . . . . .	2
I.2	Le pivot de Gauss en pratique . . . . .	3
I.2.1	De manière générale . . . . .	3
I.2.2	Exercice . . . . .	4
I.3	Implémentation de l'algorithme de Gauss en passant par le système d'équations linéaires	5
I.3.1	Commentaires fonctionnels . . . . .	5
I.3.2	Code source . . . . .	5
I.3.3	Interactions Utilisateur/Console . . . . .	6
I.3.4	Exemples d'exécution . . . . .	8
I.4	Pivot de Gauss avec matrice augmentée . . . . .	11
I.4.1	Code source . . . . .	11
I.4.2	Commentaires du code . . . . .	11
I.4.3	Inputs / Outputs . . . . .	12
I.4.4	Exemples d'exécutions . . . . .	13
<b>II</b>	<b>Résolution de systèmes linéaires par des Méthodes Itératives</b>	<b>15</b>
II.1	Méthode de Gauss-Seidel . . . . .	15
II.1.1	Introduction à la méthode de Gauss-Seidel . . . . .	15
II.1.2	Mise en place des matrices pour la méthode de Gauss-Seidel . . . . .	15
II.1.3	Algorithme . . . . .	16
II.1.4	Résolution manuelle . . . . .	16
II.1.5	Implémentation . . . . .	18
II.1.6	Exemples d'exécution . . . . .	19

# Chapitre I

## Résolution de systèmes linéaires par des Méthodes Directes : Méthode de Gauss

Dans le cadre de ce premier TP, nous devons implémenter l'algorithme du *Pivot de Gauss* en utilisant le langage de programmation C.

*Afin de rendre ce document plus compréhensible et lisible, nous estimons que la présence de nos codes sources en clair est nécessaire.*

### I.1 Détail de l'algorithme

Soient deux matrices  $A \in \mathcal{M}_{m,m}$  et  $b \in \mathcal{M}_{m,1}$ .  
L'algorithme de Gauss se décrit ainsi :

```
Pour  $k = 1, \dots, n - 1$  Faire :  
    Pour  $i = k + 1, \dots, n$  Faire :  
        
$$\alpha_i^{(k)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$
  
        Pour  $j = k, \dots, n$  Faire :  
            
$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \alpha_i^{(k)} a_{kj}^{(k)}$$
  
        FIN Pour  $j$   
        
$$b_i^{(k+1)} = b_i^{(k)} - \alpha_i^{(k)} b_k^{(k)}$$
  
    FIN Pour  $i$   
FIN Pour  $k$ 
```

Une fois la matrice échelonnée par cet algorithme, on appliquera la formule suivante pour trouver les solutions du système :

$$x_n = \frac{b_n}{a_{n,n}}$$

et

$$\forall i = n-1, \dots, 1, x_i = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1+i}^n a_{ij} x_j \right)$$

La complexité temporelle de cet algorithme est cubique soit  $O(n^3)$  avec une complexité exacte de  $\frac{2n^3}{3}$ . Pour l'implémentation de cet algorithme, nous allons présenter deux façons de le conceptualiser avec une comparaison algorithmique des deux programmes.

## I.2 Le pivot de Gauss en pratique

### I.2.1 De manière générale

Soit  $A \in \mathcal{M}_{m,m}$  et  $B \in \mathcal{M}_{m,1}$  et  $x$  la matrice des inconnues. Considérons alors le système suivant  $Ax = b$ .

Ce système peut être représenté sous la forme d'une matrice augmentée  $M$  tel que :

$$M = \left( \begin{array}{cccc|c} a_{11} & \dots & a_{1m} & & b_1 \\ \vdots & & \ddots & & \vdots \\ a_{m1} & \dots & a_{mm} & & b_m \end{array} \right)$$

Après exécution du pivot de Gauss,  $M$  devient

$$M = \left( \begin{array}{ccccc|c} 1 & a_{12} & a_{13} & \dots & a_{1m} & b'_1 \\ 0 & 1 & a'_{23} & \dots & a'_{2m} & b'_2 \\ 0 & 0 & 1 & \dots & \vdots & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & \dots & 1 & b'_m \end{array} \right)$$

Une fois que tous les pivots sont placés, il suffira de reconstituer le système et de le remonter afin de déterminer les inconnues comme suit :

$$\text{Soit } A' = \left( \begin{array}{ccccc} 1 & a'_{12} & a'_{13} & \dots & a'_{1m} \\ 0 & 1 & a'_{23} & \dots & a'_{2m} \\ 0 & 0 & 1 & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 1 \end{array} \right) \text{ et } b = \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_m \end{pmatrix}$$

alors pour  $A'x = b$  on a donc  $x_n = \frac{b_n}{a_{n,n}}$  et  $x_i = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1+i}^n a_{ij} x_j \right), \forall i = n-1, \dots, 1$ .

*Nous remarquerons que l'implémentation du pivot de Gauss ne nécessitera pas de mettre nos pivots à*

### I.2.2 Exercice

Résoudre le système linéaire suivant :  $x + y + 2z = 3$   
 $x + 2y + z = 1$

$2x + y + z = 0$  On pose  $A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix}$ ,  $B = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$  et  $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

puis la matrice augmentée  $(A \mid B) = \left( \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 0 \end{array} \right)$

Commençons par échelonner la matrice augmentée à l'aide du pivot de Gauss :

$$\left( \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 0 \end{array} \right) \xrightarrow[L_2 - L_1 \rightarrow L_2]{-(L_3 + 2L_1) \rightarrow L_3} \left( \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -1 & -2 \\ 0 & 1 & 3 & 6 \end{array} \right) \xrightarrow{\frac{L_3 - L_2}{4} \rightarrow L_3} \left( \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 1 & 2 \end{array} \right)$$

Maintenant que notre matrice augmentée est échelonnée, nous pouvons déterminer les inconnues du système par substitution (en partant du bas) :

$$\left( \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -1 & -2 \\ 0 & 0 & 1 & 2 \end{array} \right) \xrightarrow[L_1 - 2L_3 \rightarrow L_1]{L_2 + L_3 \rightarrow L_2} \left( \begin{array}{ccc|c} 1 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array} \right) \xrightarrow{L_1 - L_2 \rightarrow L_1} \left( \begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array} \right)$$

Nous avons maintenant  $A = I_3$  et donc nous pouvons remplacer dans le système  $AX = B$ ,  $A$  par  $I_3$ , ce qui nous donne :

$$AX = B \iff I_3 X = B \iff \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \iff \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix}$$

Nous avons alors notre couple solution du système, qui est :

$$\begin{aligned} x &= -1 \\ y &= 0 \\ z &= 2 \end{aligned}$$

## I.3 Implémentation de l'algorithme de Gauss en passant par le système d'équations linéaires

Dans cette partie, vous trouverez quelques commentaires sur mon implémentation de l'élimination de Gauss, ainsi que le code de l'algorithme décrit en I.1. Il est à noter que cette première implémentation ne recourt pas à l'utilisation de la matrice augmentée. En effet, le programme fonctionne directement avec le système d'équations linéaires  $Ax = B$ . Vous trouverez également le code de la fonction qui, à un système matriciel échelonné, retourne un vecteur solution du système.

### I.3.1 Commentaires fonctionnels

#### Fonctions usuelles de manipulation de matrices

Ce code implémente diverses fonctions pour travailler avec des matrices à coefficients en nombre flottants.

- La fonction ***createMatrix*** alloue dynamiquement de la mémoire pour créer une matrice de nombres flottants avec un nombre spécifié de lignes et de colonnes.
- La fonction ***printMatrix*** affiche les éléments d'une matrice de nombres flottants.
- La fonction ***freeMatrix*** libère la mémoire allouée pour une matrice de nombres flottants.
- La fonction ***completeMatrix*** permet à l'utilisateur de saisir des valeurs pour remplir les éléments d'une matrice de nombres flottants.
- La fonction ***generateB*** génère un vecteur colonne  $B$  en fonction de la somme des éléments de chaque ligne de la matrice  $A$ .

#### Fonctions résolvant notre système linéaire $Ax = B$ à l'aide de l'algorithme de Gauss

Dans le cadre de notre résolution de systèmes d'équations linéaires, deux fonctions jouent un rôle clef dans ce code : la fonction ***gauss*** et la fonction ***resolution***.

- La fonction ***gauss*** joue un rôle important dans la préparation de la résolution de notre système d'équations linéaires. En effectuant l'élimination de Gauss sur la matrice  $A$ , elle la transforme en une matrice triangulaire supérieure. Cela signifie que les éléments sous la diagonale principale de la matrice deviennent tous des zéros, simplifiant ainsi la résolution du système. De plus, la fonction met également à jour la matrice  $B$  en conséquence, garantissant que notre système  $Ax = B$  reste équilibré.
- La fonction ***resolution***, quant à elle, prend en charge la résolution effective du système linéaire une fois que la matrice  $A$  a été triangulée par la fonction ***gauss***. Elle utilise la méthode de substitution pour calculer la solution et stocke le résultat dans le vecteur  $X$ . Cette étape finale permet d'obtenir les valeurs des variables inconnues du système, fournissant ainsi la solution recherchée pour le problème initial.

En combinant ces deux fonctions avec celles sus-citées en I.3.1, le code réalise un processus complet de résolution de systèmes d'équations linéaires de manière efficace et précise (aux erreurs d'arrondies près).

### I.3.2 Code source

Puisque nous sommes contraints de minimiser la présence de code dans ce rapport, nous ne présenterons pas ici les fonctions usuelles de manipulations de matrices suivantes : ***createMatrix***, ***printMatrix***, ***freeMatrix***, ***completeMatrix***, ***generateB***.

```

0  /*
1  *PERFORM GAUSSIAN ELIMINATION ON A Ax=B MATRIX SYSTEM OF LINEAR EQUATIONS
2  */
3
4  void gauss(float** matA, float** matb, int size){
5      for(int k=0; k<size-1; k++){
6          for(int i=k+1; i<size; i++){
7              float alpha=matA[i][k]/matA[k][k];
8              for(int j=k; j<size; j++){
9                  matA[i][j]=matA[i][j]-alpha*matA[k][j];
10             }
11             matb[i][0]=matb[i][0]-alpha*matb[k][0];
12         }
13     }
14 }
15
16 /*
17 *SOLVE A MATRIX SYSTEM OF LINEAR EQUATIONS USING BACKWARD SUBSTITUTION
18 */
19 void resolution(float** matA, float** matb, float** matx, int size){
20     matx[size-1][0]=matb[size-1][0]/matA[size-1][size-1];
21     for(int i=size-2; i>=0; i--){
22         float sum=0;
23         for(int j=i+1; j<size; j++){
24             sum+=matA[i][j]*matx[j][0];
25         }
26         matx[i][0]=(1/matA[i][i])*(matb[i][0]-sum);
27     }
28 }

```

### I.3.3 Interactions Utilisateur/Console

#### Entrées utilisateur

En premier lieu dans notre programme, nous avons besoin de spécifier le système  $Ax = B$  à l'ordinateur. Pour ce faire, nous allons dans l'ordre :

1. Allouer une matrice  $A$  en mémoire. Cette matrice verra sa taille définie par la première entrée utilisateur du programme (nous demanderons consécutivement le nombre de lignes, puis le nombre de colonnes de la matrice).
2. Définir les coefficients de la matrice  $A$ . Il s'agira de la deuxième entrée utilisateur de notre programme.  
Par définition de notre fonction *completeMatrix*, nous remplirons la matrice dans l'ordre suivant :  
 $a_{1,1}, a_{1,2}, \dots, a_{1,n}$ , puis  $a_{2,1}, \dots, a_{2,n}$ , jusque  $a_{n,1}, \dots, a_{n,n}$
3. Allouer une matrice  $B$  en mémoire. À noter que la taille de  $B$  est définie automatiquement en fonction de la taille de  $A$ . Nous avons  $A \in \mathcal{M}_{n,p} \Rightarrow B \in \mathcal{M}_{n,1}$ .
4. Définir les coefficients de la matrice  $B$ . Chaque coefficient prendra la valeur de la somme des éléments de la ligne respective de la matrice  $A$ .  
Nous avons donc :  
Soient  $A \in \mathcal{M}_{n,p}$  et  $B \in \mathcal{M}_{n,1}, \forall i \in \{1, n\}, b_{i,1} = \sum_{j=1}^p a_{i,j}$ .
5. Allouer une matrice  $X$  en mémoire. Cette matrice aura la même taille que la matrice  $B$ . Ces coefficients ne seront pas définis pour le moment.

En guise d'exemple, le système matriciel  $AX = B$  suivant :

$$\begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 13 \\ 2 \end{pmatrix} \quad (\text{I.1})$$

est représenté par l'entrée utilisateur :

Listing I.1 – User Input

```

0 Row count of matrix A : 3
1
2 Column count of matrix A : 3
3
4             FILL IN THE VALUE OF MATRIX A
5
6 Value for a_1,1: 3
7 Value for a_1,2: 0
8 Value for a_1,3: 4
9 Value for a_2,1: 7
10 Value for a_2,2: 4
11 Value for a_2,3: 2
12 Value for a_3,1: -1
13 Value for a_3,2: 1
14 Value for a_3,3: 2

```

Une fois toutes les matrices initialisées et complétées, nous pouvons attaquer la résolution du système par la triangularisation du système. Ceci fait, nous résolverons le système obtenu pour obtenir notre vecteur  $X$  solution.

### Affichage Console

Dès lors le système  $AX = B$  connu par l'ordinateur, ce dernier peut retrouver les valeurs de la matrice  $X$ . Voici l'affichage produit par notre programme en console :

Listing I.2 – Console Display of the Gauss elimination for the AX

```

0             A matrix
1
2 3.000000    0.000000    4.000000
3 7.000000    4.000000    2.000000
4 -1.000000    1.000000    2.000000
5
6             B matrix
7
8 7.000000
9 13.000000
10 2.000000
11
12             TRIANGULARIZATION
13             A Matrix
14
15 3.000000    0.000000    4.000000
16 0.000000    4.000000    -7.333333
17 0.000000    0.000000    5.166667
18
19             B Matrix
20
21 7.000000
22 -3.333332

```



```

23 5.166667
24
25          SOLVING
26          SOLUTION VECTOR X
27
28 1.000000
29 1.000000
30 1.000000

```

Il est à repérer que le programme affiche dans cet ordre :

- La **Matrice**  $A$
- La **Matrice**  $B$
- La **Matrice**  $A$  une fois triangulée supérieure
- La **Matrice**  $B$  une fois mise à jour en conséquence pour que le système reste équilibré
- La **Matrice**  $X$  solution du système

Remarque : le temps d'exécution de ce programme a été de 0.000237 secondes

### I.3.4 Exemples d'exécution

*Note* : Par souci de présentation, les coefficients des matrices sont ici arrondis pour une précision de  $10^{-3}$

Soient les matrices  $A$  données dans le TP et en annexe du document. On obtient respectivement les résultats suivants :

Listing I.3 –  $A_1X = B$  results

```

0          A matrix
1      3.000  0.000  4.000
2      7.000  4.000  2.000
3     -1.000  1.000  2.000
4
5          B matrix
6      7.000
7     13.000
8      2.000
9
10         TRIANGULARIZATION
11
12         A Matrix
13      3.000  0.000  4.000
14      0.000  4.000 -7.333
15      0.000  0.000  5.167
16
17         B Matrix
18      7.000
19     -3.333
20      5.167
21
22         SOLVING
23         SOLUTION VECTOR X
24      1.000
25      1.000
26      1.000
27
28     Temps d'exécution : 0.000181 secondes

```

Listing I.4 –  $A_2X = B$  results

```

0          A matrix
1     -3.000  3.000 -6.000
2     -4.000  7.000  8.000

```

```

3      5.000    7.000    -9.000
4
5                                     B matrix
6      -6.000
7      11.000
8      3.000
9
10                                    TRIANGULARIZATION
11
12                                    A Matrix
13      -3.000    3.000    -6.000
14      0.000    3.000    16.000
15      0.000    0.000    -83.000
16
17                                    B Matrix
18      -6.000
19      19.000
20      -83.000
21
22                                    SOLVING
23
24                                    SOLUTION VECTOR X
25      1.000
26      1.000
27      1.000
28
29      Temps d'execution : 0.000209 secondes

```

Listing I.5 –  $A_3X = B$  results

```

0                                     A matrix
1      4.000    1.000    1.000
2      2.000    -9.000    0.000
3      0.000    -8.000    6.000
4
5                                     B matrix
6      6.000
7      -7.000
8      -2.000
9
10                                    TRIANGULARIZATION
11
12                                    A Matrix
13      4.000    1.000    1.000
14      0.000    -9.500    -0.500
15      0.000    0.000    6.421
16
17                                    B Matrix
18      6.000
19      -10.000
20      6.421
21
22                                    SOLVING
23
24                                    SOLUTION VECTOR X
25      1.000
26      1.000
27      1.000
28
29      Temps d'execution : 0.000195 secondes

```

Listing I.6 –  $A_4X = B$  results

```

0                                     A matrix
1      7.000    6.000    9.000
2      4.000    5.000    -4.000
3      -7.000    -3.000    8.000
4

```

```

5                                     B matrix
6      22.000
7      5.000
8     -2.000
9
10                                TRIANGULARIZATION
11
12                                A Matrix
12      7.000    6.000    9.000
13      0.000    1.571   -9.143
14      0.000    0.000   34.455
15
16                                B Matrix
17      22.000
18     -7.571
19     34.455
20
21                                SOLVING
22
23                                SOLUTION VECTOR X
23      1.000
24      1.000
25      1.000
26
27      Temps d'execution : 0.000367 secondes

```

Listing I.7 –  $A_5X = B$  results

```

0                                     A matrix
1      1.000    0.500    0.250
2      0.500    1.000    0.000
3      0.250    0.000    1.000
4
5                                     B matrix
6      1.750
7      1.500
8      1.250
9
10                                TRIANGULARIZATION
11
12                                A Matrix
12      1.000    0.500    0.250
13      0.000    0.750   -0.125
14      0.000    0.000    0.917
15
16                                B Matrix
17      1.750
18      0.625
19      0.917
20
21                                SOLVING
22
23                                SOLUTION VECTOR X
23      1.000
24      1.000
25      1.000
26
27      Temps d'execution : 0.000206 secondes

```

Listing I.8 –  $A_6X = B$  results

```

0                                     A matrix
1      1.000    0.500    0.250    0.125    0.062    0.031
2      0.500    1.000    0.000    0.000    0.000    0.000
3      0.250    0.000    1.000    0.000    0.000    0.000
4      0.125    0.000    0.000    1.000    0.000    0.000
5      0.062    0.000    0.000    0.000    1.000    0.000
6      0.031    0.000    0.000    0.000    0.000    1.000

```

A matrix							
1.000	0.500	0.250	0.125	0.062	0.031	0.016	0.008
0.500	1.000	0.000	0.000	0.000	0.000	0.000	0.000
0.250	0.000	1.000	0.000	0.000	0.000	0.000	0.000
0.125	0.000	0.000	1.000	0.000	0.000	0.000	0.000
0.062	0.000	0.000	0.000	1.000	0.000	0.000	0.000
0.031	0.000	0.000	0.000	0.000	1.000	0.000	0.000
0.016	0.000	0.000	0.000	0.000	0.000	1.000	0.000
0.008	0.000	0.000	0.000	0.000	0.000	0.000	1.000
B matrix							
1.992							
1.500							
1.250							
1.125							
1.062							
1.031							
1.016							
1.008							
TRIANGULARIZATION							
A Matrix							
1.000	0.500	0.250	0.125	0.062	0.031	0.016	0.008
0.000	0.750	-0.125	-0.062	-0.031	-0.016	-0.008	-0.004
0.000	0.000	0.917	-0.042	-0.021	-0.010	-0.005	-0.003
0.000	0.000	0.000	0.977	-0.011	-0.006	-0.003	-0.001
0.000	0.000	0.000	0.000	0.994	-0.003	-0.001	-0.001
0.000	0.000	0.000	0.000	0.000	0.999	-0.001	-0.000
0.000	0.000	0.000	0.000	0.000	0.000	1.000	-0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
B Matrix							

```

32      1.992
33      0.504
34      0.836
35      0.956
36      0.989
37      0.997
38      0.999
39      1.000
40
41      SOLVING
42
43      SOLUTION VECTOR X
44
45      1.000
46      1.000
47      1.000
48      1.000
49      1.000
50      1.000
51
52      Temps d'execution : 0.000329 secondes

```

Listing I.10 –  $A_8X = B$  results

```

0
1      A matrix
2      1.000  0.500  0.250  0.125  0.062  0.031  0.016  0.008  0.004  0.002
3      0.500  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
4      0.250  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
5      0.125  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000
6      0.062  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000
7      0.031  0.000  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000
8      0.016  0.000  0.000  0.000  0.000  0.000  1.000  0.000  0.000  0.000
9      0.008  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000  0.000
10     0.004  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
11     0.002  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000
12
13     B matrix
14     1.998
15     1.500
16     1.250
17     1.125
18     1.062
19     1.031
20     1.016
21     1.008
22     1.004
23     1.002
24
25     TRIANGULARIZATION
26
27     A Matrix
28     1.000  0.500  0.250  0.125  0.062  0.031  0.016  0.008  0.004  0.002
29     0.000  0.750  -0.125  -0.062  -0.031  -0.016  -0.008  -0.004  -0.002  -0.001
30     0.000  0.000  0.917  -0.042  -0.021  -0.010  -0.005  -0.003  -0.001  -0.001
31     0.000  0.000  0.000  0.977  -0.011  -0.006  -0.003  -0.001  -0.001  -0.000
32     0.000  0.000  0.000  0.000  0.994  -0.003  -0.001  -0.001  -0.000  -0.000
33     0.000  0.000  0.000  0.000  0.000  0.999  -0.001  -0.000  -0.000  -0.000
34     0.000  0.000  0.000  0.000  0.000  0.000  1.000  -0.000  -0.000  -0.000
35     0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  -0.000  -0.000
36     0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  1.000
37
38     B Matrix
39     1.998
40     0.501
41     0.834
42     0.955
43     0.989
44     0.997
45     0.999
46     1.000
47     1.000
48
49     SOLVING
50
51     SOLUTION VECTOR X
52
53     1.000
54     1.000
55     1.000
56     1.000

```

```

56      1.000
57      1.000
58      1.000
59      1.000
60      1.000
61
62      Temps d'execution : 0.000360 secondes

```

Listing I.11 –  $A_9X = B$  results

```

0      A matrix
1      3.000  -1.000  0.000
2      -2.000  3.000  -1.000
3      0.000  -2.000  3.000
4
5      B matrix
6      2.000
7      0.000
8      1.000
9
10     TRIANGULARIZATION
11
12     A Matrix
13     3.000  -1.000  0.000
14     0.000  2.333  -1.000
15     0.000  0.000  2.143
16
17     B Matrix
18     2.000
19     1.333
20     2.143
21
22     SOLVING
23
24     SOLUTION VECTOR X
25     1.000
26     1.000
27     1.000
28
29     Temps d'execution : 0.000345 secondes

```

Listing I.12 –  $A_{10} = B$  results

```

0      A matrix
1      3.000  -1.000  0.000  0.000  0.000  0.000
2      -2.000  3.000  -1.000  0.000  0.000  0.000
3      0.000  -2.000  3.000  -1.000  0.000  0.000
4      0.000  0.000  -2.000  3.000  -1.000  0.000
5      0.000  0.000  0.000  -2.000  3.000  -1.000
6      0.000  0.000  0.000  0.000  -2.000  3.000
7
8      B matrix
9      2.000
10     0.000
11     0.000
12     0.000
13     0.000
14     1.000
15
16     TRIANGULARIZATION
17
18     A Matrix
19     3.000  -1.000  0.000  0.000  0.000  0.000
20     0.000  2.333  -1.000  0.000  0.000  0.000
21     0.000  0.000  2.143  -1.000  0.000  0.000
22     0.000  0.000  0.000  2.067  -1.000  0.000
23     0.000  0.000  0.000  0.000  2.032  -1.000
24     0.000  0.000  0.000  0.000  0.000  2.016

```

```

25                                     B Matrix
26      2.000
27      1.333
28      1.143
29      1.067
30      1.032
31      2.016
32
33                                     SOLVING
34
35                                     SOLUTION VECTOR X
36      1.000
37      1.000
38      1.000
39      1.000
40      1.000
41
42      Temps d'execution : 0.000407 secondes

```

Listing I.13 –  $A_{11}X = B$  results

```

0                                     A matrix
1      3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000
2      -2.000  3.000  -1.000  0.000  0.000  0.000  0.000  0.000
3      0.000  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000
4      0.000  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000
5      0.000  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000
6      0.000  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000
7      0.000  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000
8      0.000  0.000  0.000  0.000  0.000  0.000  -2.000  3.000
9
10                                    B matrix
11      2.000
12      0.000
13      0.000
14      0.000
15      0.000
16      0.000
17      0.000
18      1.000
19
20                                    TRIANGULARIZATION
21
22                                    A Matrix
23      3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000
24      0.000  2.333  -1.000  0.000  0.000  0.000  0.000  0.000
25      0.000  0.000  2.143  -1.000  0.000  0.000  0.000  0.000
26      0.000  0.000  0.000  2.067  -1.000  0.000  0.000  0.000
27      0.000  0.000  0.000  0.000  2.032  -1.000  0.000  0.000
28      0.000  0.000  0.000  0.000  0.000  2.016  -1.000  0.000
29      0.000  0.000  0.000  0.000  0.000  0.000  2.008  -1.000
30      0.000  0.000  0.000  0.000  0.000  0.000  0.000  2.004
31
32                                    B Matrix
33      2.000
34      1.333
35      1.143
36      1.067
37      1.032
38      1.016
39      1.008
40      2.004
41
42                                    SOLVING
43
44                                    SOLUTION VECTOR X
45      1.000
46      1.000
47      1.000
48      1.000
49      1.000
50      1.000
51
52      Temps d'execution : 0.000318 secondes

```

Listing I.14 –  $A_{12} = B$  results

```

0
1      3.000   -1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
2      -2.000   3.000   -1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
3      0.000   -2.000   3.000   -1.000   0.000   0.000   0.000   0.000   0.000   0.000
4      0.000   0.000   -2.000   3.000   -1.000   0.000   0.000   0.000   0.000   0.000
5      0.000   0.000   0.000   -2.000   3.000   -1.000   0.000   0.000   0.000   0.000
6      0.000   0.000   0.000   0.000   -2.000   3.000   -1.000   0.000   0.000   0.000
7      0.000   0.000   0.000   0.000   0.000   -2.000   3.000   -1.000   0.000   0.000
8      0.000   0.000   0.000   0.000   0.000   0.000   -2.000   3.000   -1.000   0.000
9      0.000   0.000   0.000   0.000   0.000   0.000   0.000   -2.000   3.000   -1.000
10     0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   -2.000   3.000
11
12
13      2.000
14      0.000
15      0.000
16      0.000
17      0.000
18      0.000
19      0.000
20      0.000
21      0.000
22      1.000
23
24      TRIANGULARIZATION
25
26      3.000   -1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
27      0.000   2.333   -1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
28      0.000   0.000   2.143   -1.000   0.000   0.000   0.000   0.000   0.000   0.000
29      0.000   0.000   0.000   2.067   -1.000   0.000   0.000   0.000   0.000   0.000
30      0.000   0.000   0.000   0.000   2.032   -1.000   0.000   0.000   0.000   0.000
31      0.000   0.000   0.000   0.000   0.000   2.016   -1.000   0.000   0.000   0.000
32      0.000   0.000   0.000   0.000   0.000   0.000   2.008   -1.000   0.000   0.000
33      0.000   0.000   0.000   0.000   0.000   0.000   0.000   2.004   -1.000   0.000
34      0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   2.002   -1.000
35      0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   2.001
36
37      B Matrix
38      2.000
39      1.333
40      1.143
41      1.067
42      1.032
43      1.016
44      1.008
45      1.004
46      1.002
47      2.001
48
49      SOLVING
50
51      SOLUTION VECTOR X
52      1.000
53      1.000
54      1.000
55      1.000
56      1.000
57      1.000
58      1.000
59      1.000
60      1.000
61
62      Temps d'execution : 0.000563 secondes

```

**Nous remarquerons** que sur le calcul de  $A_4$ , par exemple, nous tombons sur des valeurs extrêmement proches de 1. Ceci est provoqué à cause des erreurs d'arrondis provoquées par l'encodage des nombres flottants.



## I.4 Pivot de Gauss avec matrice augmentée

### I.4.1 Code source

Voici le code source de mon implémentation du pivot de Gauss via le passage par la matrice augmentée.

C'est-à-dire que dans mon implémentation, on fera usage de la concaténation des matrices.

```

0  /*
1  * BUILD AUGMENTED MATRIX
2  */
3  float **AugmentedMatrix(float **M1, float **M2, int m, int n) {
4      float **A = allocate(m, m + 1);
5      for (int i = 0; i < m; i++) {
6          for (int j = 0; j < n + 1; j++) {
7              (j != n) ? (A[i][j] = M1[i][j]) : (A[i][j] = M2[i][0]);
8          }
9      }
10     return A;
11 }
12 /*
13 * PERFORM GAUSS ALGORITHM ONLY ON AUGMENTED MATRIX
14 */
15 void gauss(float **A, int m, int p) {
16     if (m != p) {
17         puts("La matrice doit etre carree !");
18         return;
19     }
20     for (int k = 0; k <= m - 1; k++) {
21         for (int i = k + 1; i < m; i++) {
22             float pivot = A[i][k] / A[k][k];
23             for (int j = k; j <= m; j++) {
24                 A[i][j] = A[i][j] - pivot * A[k][j];
25             }
26         }
27     }
28 }
29 /*
30 * DETERMINE ALL UNKNOWN VARIABLES
31 */
32 float *findSolutions(float **A, int m) {
33     float *S = calloc(m, sizeof *S);
34     S[m - 1] = A[m - 1][m] / A[m - 1][m - 1];
35     for (int i = m - 1; i >= 0; i--) {
36         S[i] = A[i][m];
37         for (int j = i + 1; j < m; j++) {
38             S[i] -= A[i][j] * S[j];
39         }
40         S[i] = S[i] / A[i][i];
41     }
42     return S;
43 }
```

### I.4.2 Commentaires du code

Mon implémentation utilise strictement l'algorithme de Gauss rappelé précédemment avec seulement quelques changements d'indices puisque au lieu de travailler sur une matrice carrée et un vecteur colonne, mon programme utilise une matrice augmentée ayant  $m$  lignes et  $m + 1$  colonnes,  $m \in \mathbb{N}^*$ .

#### Détail des fonctions non conventionnelles :

Comme mentionné précédemment, je ne détaillerai pas les fonctions `gauss()` et `findSolutions()` puisque ces fonctions permettent seulement d'une part d'implémenter l'algorithme de Gauss et d'autre part à "remonter" la matrice échelonnée afin de récupérer les valeurs des inconnus.

**-float \*\*AugmentedMatrix(float \*\*M1, float \*\*M2, int m, int n) :** cette fonction permet de créer une matrice  $A \in \mathcal{M}_{m,m+1}$  à partir de la concaténation de  $M1 \in \mathcal{M}_{mm}$  et  $M2 \in \mathcal{M}_{m,1}$ . Soient  $a_{ij}$  les coefficients peuplant  $A$ ,  $b_{ij}$  les coefficients peuplant  $M1$  et  $c_{i0}$  les coefficients peuplant  $M2$ .

On obtient alors  $a_{ij} = b_{ij} \forall i \in \mathbb{N}_m, \forall j \in \mathbb{N}_m$  et  $a_{ij} = c_{i0}$  si  $j = m + 1$ .

Cette fonction renvoie alors  $A$ , la matrice de flottants créée dynamiquement.

Les fonctions *fillM()*, *printMatrix()*, *freeAll()* et *multiplication()* et *allocate()* sont quatre fonctions utilitaires qui permettent respectivement : de remplir une matrice, d'afficher convenablement une matrice, de libérer les matrices en mémoires, de définir la multiplication matricielle et enfin d'allouer de la mémoire pour déclarer les matrices (avec quelques légères sécurités permettant d'être sûr que les matrices sont bieninstanciées convenablement).

### I.4.3 Inputs / Outputs

Mon programme demande d'abord 4 entiers  $m, p, n, q$  qui correspondent aux dimensions de la première matrice  $A \in \mathcal{M}_{mp}$  et de la seconde matrice  $X \in \mathcal{M}_{nq}$ . Le but étant de résoudre le système  $AX = b$ , nous initialiserons  $X$  à 1. Ce choix de valeur permettra de contrôler la validité du programme, ainsi à la fin de ce dernier, si  $\forall x_i \neq 1, \forall i \in \mathbb{N}_n$ , on pourra affirmer que le programme est faux.

Sur les  $m \times p$  prochaines lignes, le programme demande les coefficients de  $A$ .

Sur les  $n \times q$  prochaines lignes, le programme demandera les coefficients du vecteur colonne  $X$ , que l'utilisateur initialisera à 1.

On peut alors automatiser les entrées en utilisant des fichiers.

Ainsi les matrices :  $M = \begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{pmatrix}$  et  $X = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

sont représentés par ce fichier d'entrées :

Listing I.15 – input.txt

```
0 3 3
1 3 1
2 3 0 4
3 7 4 2
4 -1 1 2
5 1 1 1
```

Pour ce qui est des résultats produits par mon programme, une fois injecté dans un fichier texte, une output "type" ressemble à ceci.

Listing I.16 – Gauss elimination with M and X matrix

```
0 PRINTING MATRIX FROM: 0x556d938672c0 LOCATION :
1 3.000000 0.000000 4.000000
2 7.000000 4.000000 2.000000
3 -1.000000 1.000000 2.000000
4 PRINTING MATRIX FROM: 0x556d93867340 LOCATION :
5 1.000000
6 1.000000
7 1.000000
8 PRINTING MATRIX FROM: 0x556d938673c0 LOCATION :
9 7.000000
10 13.000000
11 2.000000
12 PRINTING MATRIX FROM: 0x556d93867440 LOCATION :
13 3.000000 0.000000 4.000000 7.000000
14 0.000000 4.000000 -7.333333 -3.333332
15 0.000000 0.000000 5.166667 5.166667
16 SOLUTIONS
17 x0 = 1.000000
```

```

18 | x1 = 1.000000
19 | x2 = 1.000000
20 | RUNTIME: 0.000002 seconds

```

On remarquera que le programme affiche dans cet ordre :

- **La Matrice A**
- **La Matrice X**
- **La Matrice B trouvée avec les valeur de X**
- **La Matrice augmentée en triangle supérieur**
- **Les solutions**
- **Un timer permettant de contrôler le temps d'exécution approximatif de mon programme**

#### I.4.4 Exemples d'exécutions

$$\text{Soient } A_2 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & 8 \\ 5 & 7 & -9 \end{pmatrix}, A_4 = \begin{pmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{pmatrix}, A_6 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & 8 \\ 5 & 7 & -9 \end{pmatrix}$$

On obtient respectivement ces résultats :

Listing I.17 – Matrix 2 results

```

0 | PRINTING MATRIX FROM: 0x55f604fb32c0 LOCATION :
1 | -3.000000 3.000000 -6.000000
2 | -4.000000 7.000000 8.000000
3 | 5.000000 7.000000 -9.000000
4 | PRINTING MATRIX FROM: 0x55f604fb3340 LOCATION :
5 | 1.000000
6 | 1.000000
7 | 1.000000
8 | PRINTING MATRIX FROM: 0x55f604fb33c0 LOCATION :
9 | -6.000000
10 | 11.000000
11 | 3.000000
12 | PRINTING MATRIX FROM: 0x55f604fb3440 LOCATION :
13 | -3.000000 3.000000 -6.000000 -6.000000
14 | 0.000000 3.000000 16.000000 19.000000
15 | 0.000000 0.000000 -83.000000 -83.000000
16 | SOLUTIONS
17 | x0 = 1.000000
18 | x1 = 1.000000
19 | x2 = 1.000000
20 | RUNTIME: 0.000002 seconds

```

Listing I.18 – Matrix 4 results

```

0 | PRINTING MATRIX FROM: 0x55f7afd662c0 LOCATION :
1 | 7.000000 6.000000 9.000000
2 | 4.000000 5.000000 -4.000000
3 | -7.000000 -3.000000 8.000000
4 | PRINTING MATRIX FROM: 0x55f7afd66340 LOCATION :
5 | 1.000000
6 | 1.000000
7 | 1.000000
8 | PRINTING MATRIX FROM: 0x55f7afd663c0 LOCATION :
9 | 22.000000
10 | 5.000000
11 | -2.000000
12 | PRINTING MATRIX FROM: 0x55f7afd66440 LOCATION :
13 | 7.000000 6.000000 9.000000 22.000000
14 | 0.000000 1.571428 -9.142858 -7.571429
15 | 0.000000 0.000000 34.454552 34.454548

```

```

16 SOLUTIONS
17 x0 = 1.000001
18 x1 = 0.999999
19 x2 = 1.000000
20 RUNTIME: 0.000002 seconds

```

Listing I.19 – Matrix 6 results

```

0 PRINTING MATRIX FROM: 0x557fdaa552c0 LOCATION :
1 3.000000 -1.000000 0.000000
2 0.000000 3.000000 -1.000000
3 0.000000 -2.000000 3.000000
4 PRINTING MATRIX FROM: 0x557fdaa55340 LOCATION :
5 1.000000
6 1.000000
7 1.000000
8 PRINTING MATRIX FROM: 0x557fdaa553c0 LOCATION :
9 2.000000
10 2.000000
11 1.000000
12 PRINTING MATRIX FROM: 0x557fdaa55440 LOCATION :
13 3.000000 -1.000000 0.000000 2.000000
14 0.000000 3.000000 -1.000000 2.000000
15 0.000000 0.000000 2.333333 2.333333
16 SOLUTIONS
17 x0 = 1.000000
18 x1 = 1.000000
19 x2 = 1.000000
20 RUNTIME: 0.000002 seconds

```

**On apercevra** que sur le calcul de  $A_4$ , on obtient sur des valeurs extrêmement proches de 1. Ceci est provoqué par des erreurs d'arrondis issus par l'encodage des nombres flottants.

**Nous remarquerons** que l'implémentation utilisant la matrice augmentée est sensiblement meilleure en terme d'efficacité. En effet, le temps d'exécution est multiplié par 100 sur la première implémentation.

## Chapitre II

# Résolution de systèmes linéaires par des Méthodes Itératives

## II.1 Méthode de Gauss-Seidel

### II.1.1 Introduction à la méthode de Gauss-Seidel

La méthode de Gauss-Seidel est une méthode itérative pour résoudre les systèmes linéaires de la forme  $Ax = b$ , où  $A$  est une matrice carrée d'ordre  $n$  et  $x, b$  sont des vecteurs de  $\mathbb{R}^n$ . C'est une méthode qui génère une suite qui converge vers la solution de ce système lorsque celle-ci en a une et lorsque les conditions de convergence suivantes sont satisfaites (quels que soient le vecteur  $b$  et le point initial  $x^0$ ) :

- Si la matrice  $A$  est symétrique définie positive,
- Si la matrice  $A$  est à diagonale strictement dominante.

### II.1.2 Mise en place des matrices pour la méthode de Gauss-Seidel

Soit  $Ax = b$  le système linéaire à résoudre, où  $A \in \mathcal{M}_{n,n}$  et  $b \in \mathcal{M}_{n,1}$ . On cherche  $x \in \mathcal{M}_{n,1}$  solution du système. Dans un premier temps, on va écrire  $A$  sous la forme  $A = D - E - F$  où  $D$  est une matrice diagonale,  $E$  est une matrice triangulaire inférieure, et  $F$  est une matrice triangulaire supérieure.

On peut alors écrire :

$$Ax = b \quad (\text{II.1})$$

$$\Leftrightarrow (D - E - F)x = b \quad (\text{II.2})$$

$$\Leftrightarrow Dx = b - (E + F)x \quad (\text{II.3})$$

$$\Leftrightarrow x = D^{-1}[b - (E + F)x] \quad (\text{II.4})$$

On définit ensuite une suite de vecteurs  $(x^k)$  en choisissant un vecteur  $x^0$  et par la formule de récurrence :

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{k+1} - \sum_{j=i+1}^n a_{i,j} x_j^k \right) \quad (\text{II.5})$$

### II.1.3 Algorithme

Pour résoudre un système  $Ax = b$ , avec  $A \in \mathcal{M}_n$  et  $b \in \mathcal{M}_{n,1}$ , on s'appuie sur l'algorithme suivant en posant :

- un vecteur initial  $x^{(0)}$  choisi au préalable,

- l'erreur à l'itération  $k=0$  calculée par  $\varepsilon^{(0)} = \|Ax^{(0)} - b\|$ ,
- une variable  $k$  qui sera notre compteur d'itération.

```

0   $x^{(0)} = x_0 \in \mathcal{M}_{n,1}$ 
1   $\varepsilon^{(0)} = \varepsilon$  (erreur)
2   $k = 0$ 
3  Tant Que  $(\varepsilon^{(k)} > \varepsilon)$  faire :
4      Pour  $i = 1$  à  $n$  :
5           $x_i^{(k+1)} = \frac{1}{a_{i,i}} \left[ b_i - \left( \sum_{j=i+1}^n a_{i,j} x_j^{(k)} + \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} \right) \right]$  pour  $i = 1, \dots, n$ 
6           $\varepsilon^{(k+1)} = \|Ax^{(k+1)} - b\|$ 
7           $k = k + 1$ 
8  Fin Tant Que

```

### II.1.4 Résolution manuelle

Soit  $A = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & 3 \\ 3 & 7 & 8 \end{pmatrix} \in \mathcal{M}_3(\mathbb{R})$ , et  $b = \begin{pmatrix} 2 \\ 2 \\ 8 \end{pmatrix} \in \mathcal{M}_{3,1}(\mathbb{R})$

Calculons le vecteur  $x^{(1)}$  (vecteur  $x$  trouvé après 1 itération de l'algorithme) solution du système  $Ax = b$ ,

en prenant comme point initial  $x^{(0)} = (0, 0, 0)$  :

#### Résolution par le calcul itératif

Dans cette sous-partie, nous résolverons le système de la même manière que le fait l'algorithme sus-cité.

Pour obtenir le vecteur  $x^{(1)}$  (obtenu à l'itération  $k = 1$ ), il nous faut obtenir  $x_1^{(1)}, x_2^{(1)}, x_3^{(1)}$  par la formule suivante :

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left[ b_i - \left( \sum_{j=i+1}^n a_{i,j} x_j^{(k)} + \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} \right) \right] \text{ pour } i = 1, \dots, 3$$

Pour  $i = 1$  :

$$x_1^{(1)} = \frac{1}{a_{1,1}} \left[ b_1 - \left( \sum_{j=2}^3 a_{1,j} x_j^{(0)} + \sum_{j=1}^0 a_{1,j} x_j^{(1)} \right) \right] \quad (\text{II.6})$$

$$= \frac{1}{1} \left[ 2 - \left( a_{1,2} x_2^{(0)} + a_{1,3} x_3^{(0)} + 0 \right) \right] \quad (\text{II.7})$$

$$= 2 - 2 \times 0 - 3 \times 0 = 2 \quad (\text{II.8})$$

Pour  $i = 2$  :

$$x_2^{(1)} = \frac{1}{a_{2,2}} \left[ b_2 - \left( \sum_{j=3}^3 a_{2,j} x_j^{(0)} + \sum_{j=1}^1 a_{2,j} x_j^{(1)} \right) \right] \quad (\text{II.9})$$

$$= \frac{1}{3} \left[ 2 - \left( a_{2,3}x_3^{(0)} + a_{2,1}x_1^{(1)} \right) \right] \quad (\text{II.10})$$

$$= \frac{1}{3} \left( 2 - 3 \times 0 - 1 \times 2 \right) \quad (\text{II.11})$$

$$= \frac{1}{3} \times 0 = 0 \quad (\text{II.12})$$

Pour  $i = 3$  :

$$x_3^{(1)} = \frac{1}{a_{3,3}} \left[ b_3 - \left( \sum_{j=4}^3 a_{3,j}x_j^{(0)} + \sum_{j=1}^2 a_{3,j}x_j^{(1)} \right) \right] \quad (\text{II.13})$$

$$= \frac{1}{8} \left[ 8 - \left( 0 + a_{3,1}x_1^{(1)} + a_{3,2}x_2^{(1)} \right) \right] \quad (\text{II.14})$$

$$= \frac{1}{8} \left( 8 - 3 \times 2 - 7 \times 0 \right) \quad (\text{II.15})$$

$$= \frac{1}{8} \times 2 = \frac{1}{4} \quad (\text{II.16})$$

Conclusion :

Nous avons  $x_1^{(1)} = 2$ ,  $x_2^{(1)} = 0$ ,  $x_3^{(1)} = \frac{1}{4}$ . Et donc,  $x^{(1)} = \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ \frac{1}{4} \end{pmatrix}$

### Résolution par le calcul matriciel

Dans la section II.1.2, nous avons vu que l'on pouvait décomposer la matrice  $A$  par une matrice diagonale  $D$ , une matrice triangulaire inférieure  $E$ , et une matrice triangulaire supérieure  $F$ . Ceci fait, nous pouvons obtenir le vecteur  $x$  par la formule suivante :

$$x^{(k+1)} = D^{-1}[b - (E + F)x^{(k)}]$$

Nous avons alors :

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & 3 \\ 3 & 7 & 8 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 8 \end{pmatrix}}_D - \underbrace{\begin{pmatrix} 0 & -2 & -2 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix}}_E - \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -3 & -7 & 0 \end{pmatrix}}_F$$

Nous obtenons alors :

$$x^{(1)} = \begin{pmatrix} \frac{1}{1} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{8} \end{pmatrix} \left[ \begin{pmatrix} 2 \\ 2 \\ 8 \end{pmatrix} - \left( \begin{pmatrix} 0 & -2 & -2 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -3 & -7 & 0 \end{pmatrix} \right) \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \right] \quad (\text{II.17})$$

$$= \begin{pmatrix} \frac{1}{1} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{8} \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 8 \end{pmatrix} \quad (\text{II.18})$$

$$= \begin{pmatrix} 2 \\ \frac{2}{3} \\ 1 \end{pmatrix} \quad (\text{II.19})$$

*Remarque : Au fur et à mesure des itérations, le vecteur  $x$  donné par le calcul itératif effectué dans la partie II.1.4 se rapproche de la solution donnée par le précédent calcul. Il est alors normal que le vecteur trouvé au bout de la première itération soit différent du vecteur trouvé ci-dessus.*

## II.1.5 Implémentation

### Commentaires fonctionnels

*Note : L'implémentation qui suit utilise exactement les mêmes fonctions usuelles de manipulation de matrice que l'implémentation de l'algorithme de Gauss décrit dans la section I.3.1. De plus, dans cette implémentation, nous définirons une variable  $k$  qui sera notre compteur d'itération et qui permettra l'arrêt de notre code si la suite ne converge pas. Enfin, notre variable erreur  $\varepsilon$  sera mise à jour à chaque itération de la manière suivante :*

$$\varepsilon^{(k)} = p^{(k)} = \text{Max}_{i=1,\dots,n} |\bar{x}_i - \tilde{x}_i^k|$$

Nous détaillerons dans cette section uniquement les fonctions dites "non-usuelles" qui vont nous servir pour l'implémentation de l'algorithme de Gauss-Seidel. Il s'agit ici de la fonction de mise à jour de notre variable erreur et de la fonction implémentant l'algorithme de Gauss-Seidel.

#### Fonction *majEpsilon* :

La fonction **majEpsilon** permet de mettre à jour la variable d'erreur  $\varepsilon$  lors de l'exécution de notre algorithme. Grâce au vecteur  $x^{(k)}$  qui représente la solution actuelle de notre système d'équations, la fonction calcule la différence absolue entre chaque élément  $x_i^{(k)}$  et 1. Cela permet de mesurer à quel point les valeurs actuelles se rapprochent de 1, qui est notre valeur cible pour les solutions convergentes. La fonction conserve le maximum de ces différences absolues en tant que mesure d'erreur afin de mettre à jour notre variable  $\varepsilon$ . Cela permet de contrôler la précision de l'algorithme et de décider quand il a convergé de manière satisfaisante vers la solution recherchée. La fonction **majEpsilon** joue donc un rôle dans la détermination du critère d'arrêt de l'algorithme. Voici son implémentation en C :

```

0 float majEpsilon(float** matXk, int row){
1     float maxforEps=0;
2     for(int i=0; i<row; i++){
3         float soustr=fabs(1-matXk[i][0]);
4         printf("%f\n", matXk[i][0]);
5         if (soustr>maxforEps){
6             maxforEps=soustr;
7         }
8     }
9     return maxforEps;
10 }
```

#### Fonction *gaussSeidel* :

La fonction **gaussSeidel** implémente l'algorithme de Gauss-Seidel tel que décrit précédemment dans la section II.1.3. Par la programmation itérative, notre algorithme mettra à jour les solutions actuelles jusqu'à ce que l'erreur minimale définie soit atteinte ou que le nombre maximal d'itérations soit atteint. Voici son implémentation en C :

```

0 float** gaussSeidel(float **matA, float **matB, float **matXk, int row, int column,
1                     int nbIterMax)
2 {
3     //CREATING OUR SOLUTION VECTOR AT ITERATION k
4     float **matXkl=createMatrix(row, 1);
5
6     //INITIALIZING OUR ERROR VARIABLE AND ITERATION COUNTER
7     float epsilon=majEpsilon(matXk, row);
8     int iter=0;
9 }
```



```

10  while ((epsilon >= pow(10, -6)) && (iter < nbIterMax)) {
11      for (int i=0; i < row ; i++) {
12          float sumF=0;
13          float sumE=0;
14
15          //CALCULATION OF F
16          for (int j=i+1; j < row ; j++) {
17              sumF += matA[i][j] * matXk[j][0];
18          }
19
20          //CALCULATION OF E
21          for (int j=0; j < i ; j++) {
22              sumE += matA[i][j] * matXk1[j][0];
23          }
24
25          //CALCULATION OF ELEMENT  $X_i^{(k)}$ 
26          matXk1[i][0] = (matB[i][0] - sumF - sumE) / matA[i][i];
27
28      }
29
30      //UPDATING OUR SOLUTION VECTOR
31      for (int k=0; k < row ; k++) {
32          matXk[k][0] = matXk1[k][0];
33      }
34
35      //UPDATING OUR ERROR VARIABLE AND ITERATION COUNTER
36      epsilon = majEpsilon(matXk, row);
37      iter++;
38  }
39
40  //RETURN THE SOLUTION VECTOR
41  return matXk;
42  }

```

### II.1.6 Exemples d'exécution

Soient les matrices  $A$  données dans le TP et dans l'annexe du document (section ??). En résolvant le système  $Ax = b$  en question, nous obtenons respectivement les résultats suivants :

Listing II.1 –  $A_1X = B$  results

```

0  A matrix
1  3.000  0.000  4.000
2  7.000  4.000  2.000
3  -1.000  1.000  2.000
4  B matrix
5  7.000
6  13.000
7  2.000
8
9  SOLVING
10 SOLUTION VECTOR X
11 -nan
12 -nan
13 -nan
14 Temps d'exécution : 0.000380 secondes

```

Listing II.2 –  $A_2X = B$  results

```

0  A matrix
1  -3.000  3.000  -6.000
2  -4.000  7.000  8.000
3  5.000  7.000  -9.000

```

```

4          B matrix
5      -6.000
6      11.000
7      3.000
8          SOLVING
9      SOLUTION VECTOR X
10     -nan
11     -nan
12     -nan
13     Temps d'execution : 0.000328 secondes

```

Listing II.3 –  $A_3X = B$  results

```

0          A matrix
1      4.000    1.000    1.000
2      2.000   -9.000    0.000
3      0.000   -8.000    6.000
4          B matrix
5      6.000
6      -7.000
7      -2.000
8          SOLVING
9      SOLUTION VECTOR X
10     1.000
11     1.000
12     1.000
13     Temps d'execution : 0.000257 secondes

```

Listing II.4 –  $A_4X = B$  results

```

0          A matrix
1      7.000    6.000    9.000
2      4.000    5.000   -4.000
3     -7.000   -3.000    8.000
4          B matrix
5     22.000
6      5.000
7     -2.000
8          SOLVING
9      SOLUTION VECTOR X
10     1.000
11     1.000
12     1.000
13     Temps d'execution : 0.000267 secondes

```

Listing II.5 –  $A_5X = B$  results

```

0          A matrix
1      1.000    0.500    0.250
2      0.500    1.000    0.000
3      0.250    0.000    1.000
4          B matrix
5      1.750
6      1.500
7      1.250
8          SOLVING
9      SOLUTION VECTOR X
10     1.000
11     1.000
12     1.000
13     Temps d'execution : 0.000286 secondes

```

Listing II.6 –  $A_6X = B$  results

```

0
1      A matrix
2      1.000    0.500    0.250    0.125    0.062    0.031
3      0.500    1.000    0.000    0.000    0.000    0.000
4      0.250    0.000    1.000    0.000    0.000    0.000
5      0.125    0.000    0.000    1.000    0.000    0.000
6      0.062    0.000    0.000    0.000    1.000    0.000
7      0.031    0.000    0.000    0.000    0.000    1.000
8      B matrix
9      1.969
10     1.500
11     1.250
12     1.125
13     1.062
14     1.031
15
16     SOLVING
17     SOLUTION VECTOR X
18     1.000
19     1.000
20     1.000
21     1.000
22     1.000
23     1.000
24     Temps d'execution : 0.000305 secondes

```

Listing II.7 –  $A_7X = B$  results

```

0
1      A matrix
2      1.000    0.500    0.250    0.125    0.062    0.031    0.016    0.008
3      0.500    1.000    0.000    0.000    0.000    0.000    0.000    0.000
4      0.250    0.000    1.000    0.000    0.000    0.000    0.000    0.000
5      0.125    0.000    0.000    1.000    0.000    0.000    0.000    0.000
6      0.062    0.000    0.000    0.000    1.000    0.000    0.000    0.000
7      0.031    0.000    0.000    0.000    0.000    1.000    0.000    0.000
8      0.016    0.000    0.000    0.000    0.000    0.000    1.000    0.000
9      0.008    0.000    0.000    0.000    0.000    0.000    0.000    1.000
10     B matrix
11     1.992
12     1.500
13     1.250
14     1.125
15     1.062
16     1.031
17     1.016
18     1.008
19
20     SOLVING
21     SOLUTION VECTOR X
22     1.000
23     1.000
24     1.000
25     1.000
26     1.000
27     1.000
28     1.000
29     Temps d'execution : 0.000369 secondes

```

Listing II.8 –  $A_8X = B$  results

```

0
1      A matrix
2      1.000    0.500    0.250    0.125    0.062    0.031    0.016    0.008    0.004    0.002
3      0.500    1.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000
4      0.250    0.000    1.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000
5      0.125    0.000    0.000    1.000    0.000    0.000    0.000    0.000    0.000    0.000
6      0.062    0.000    0.000    0.000    1.000    0.000    0.000    0.000    0.000    0.000
7      0.031    0.000    0.000    0.000    0.000    1.000    0.000    0.000    0.000    0.000
8      0.016    0.000    0.000    0.000    0.000    0.000    1.000    0.000    0.000    0.000
9      0.008    0.000    0.000    0.000    0.000    0.000    0.000    1.000    0.000    0.000
10     0.004    0.000    0.000    0.000    0.000    0.000    0.000    0.000    1.000    0.000
11     0.002    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    1.000
12     B matrix
13     1.998
14     1.500

```

```

14 1.250
15 1.125
16 1.062
17 1.031
18 1.016
19 1.008
20 1.004
21 1.002
22
23 SOLVING
24 SOLUTION VECTOR X
25
26 1.000
27 1.000
28 1.000
29 1.000
30 1.000
31 1.000
32 1.000
33 1.000
34 Temps d'execution : 0.000351 secondes

```

Listing II.9 –  $A_9X = B$  results

```

0 A matrix
1 3.000 -1.000 0.000
2 -2.000 3.000 -1.000
3 0.000 -2.000 3.000
4 B matrix
5 2.000
6 0.000
7 1.000
8
9 SOLVING
10 SOLUTION VECTOR X
11 1.000
12 1.000
13 1.000
14 Temps d'execution : 0.000274 secondes

```

Listing II.10 –  $A_{10}X = B$  results

```

0 A matrix
1 3.000 -1.000 0.000 0.000 0.000 0.000
2 -2.000 3.000 -1.000 0.000 0.000 0.000
3 0.000 -2.000 3.000 -1.000 0.000 0.000
4 0.000 0.000 -2.000 3.000 -1.000 0.000
5 0.000 0.000 0.000 -2.000 3.000 -1.000
6 0.000 0.000 0.000 0.000 -2.000 3.000
7 B matrix
8 2.000
9 0.000
10 0.000
11 0.000
12 0.000
13 1.000
14
15 SOLVING
16 SOLUTION VECTOR X
17 1.000
18 1.000
19 1.000
20 1.000
21 1.000
22 Temps d'execution : 0.000315 secondes

```

Listing II.11 –  $A_{11}X = B$  results

```

0 A matrix

```

```

1  3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000
2  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000  0.000
3  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000
4  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000
5  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000
6  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000
7  0.000  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000
8  0.000  0.000  0.000  0.000  0.000  0.000  -2.000  3.000
9
10         B matrix
11
12  2.000
13  0.000
14  0.000
15  0.000
16  0.000
17  0.000
18
19         SOLVING
20         SOLUTION VECTOR X
21
22  1.000
23  1.000
24  1.000
25  1.000
26  1.000
27  1.000
28  Temps d'execution : 0.000341 secondes

```

Listing II.12 –  $A_1 2X = B$  results

```

0
1         A matrix
2  3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
3  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
4  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000  0.000  0.000
5  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000  0.000
6  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000  0.000
7  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000  0.000
8  0.000  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000  0.000
9  0.000  0.000  0.000  0.000  0.000  0.000  -2.000  3.000  -1.000  0.000
10 0.000  0.000  0.000  0.000  0.000  0.000  0.000  -2.000  3.000
11
12         B matrix
13
14  2.000
15  0.000
16  0.000
17  0.000
18  0.000
19  0.000
20  0.000
21  1.000
22
23         SOLVING
24         SOLUTION VECTOR X
25
26  1.000
27  1.000
28  1.000
29  1.000
30  1.000
31  1.000
32  1.000
33  1.000
34  Temps d'execution : 0.000479 secondes

```

report [french]babel [T1]fontenc [utf8]inputenc mathtools amssymb hyperref float amsthm listings  
 geometry setspace graphicx fancyhdr subcaption cleveref  
 Définition  
 hmargin=3cm,vmargin=2.5cm

## II.2 Méthode de Jacobi

Rappelons que la méthode de **Jacobi** est itérative et ne garantit pas toujours un résultat. La méthode est définie si  $A$  est définie positive.

L'algorithme permet de trouver un résultat si la matrice est dite à diagonale strictement dominante.

Autrement dit, Soit  $[a_{ij}]_{0 \leq i, j \leq n}$  les coefficients réels peuplant  $A \in \mathcal{M}_{n,n}(\mathbb{R})$ , alors si :

$\forall i, |a_{i,i}| < \sum_{i \neq j} |a_{ij}|$ , on a que Jacobi converge vers l'unique solution du système  $Ax = b$ .

### II.2.1 Principe de la méthode

On veut résoudre  $Ax = b$  avec  $A \in \mathcal{M}_{n,n}(\mathbb{R})$ ,  $n \in \mathbb{N}$ ,  $x$  la vecteur colonne contenant les inconnus et  $b$  le vecteur colonne des solution.

On pose  $D \in \mathcal{M}_{n,n}(\mathbb{R})$  la matrice contenant les coefficients  $[a_{i,j}]_{0 \leq i=j \leq n}$  de  $A$ .

On pose aussi  $E$  et  $F$  avec  $E$  la matrice triangulaire opposée inférieure de  $A$  et  $F$  la matrice supérieure opposée de  $A$ .

On obtient alors :

$$Ax = b \quad (\text{II.20})$$

$$(D - E - F)x = b \quad (\text{II.21})$$

$$Dx - (E + F)x = b \quad (\text{II.22})$$

$$x = D^{-1}(E + F)x + D^{-1}b \quad (\text{II.23})$$

$$x^{k+1} = D^{-1}(E + F)x^k + D^{-1}b \quad (\text{II.24})$$

Ce qui donne l'algorithme suivant :

Soit  $\epsilon$  L'erreur maximale, un point initial  $x^0$  et  $k = 0$

avec  $\epsilon^0 = \|Ax^0 - b\|$

On obtient :

0	Tant que $(\epsilon^{(k)} \leq \epsilon)$
1	$x_i^{k+1} = \frac{1}{a_{ii}}[b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}], i = 1, \dots, n$
2	$\epsilon^{k+1} = \ Ax^{k+1} - b\ $
3	$k = k + 1$
4	FIN JACOBI

**Remarque, on ajoutera aussi un nombre d'itérations maximum afin de ne pas être dans le cas d'une boucle infinies (si jacobi diverge alors l'erreur augmente).**

### II.2.2 Résolution manuelle

*Nous en détaillerons seulement une itération*

Soit  $A = \begin{pmatrix} 4 & 1 & 0 \\ -1 & 3 & 6 \\ -2 & -5 & -3 \end{pmatrix}$ ,  $b = \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix}$ ,  $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$  et  $x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

On a  $A = \underbrace{\begin{pmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{pmatrix}}_D - \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 5 & 0 \end{pmatrix}}_E - \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -6 \\ 0 & 0 & 0 \end{pmatrix}}_F$

On a donc  $x^{k+1} = D^{-1}[(E + F)x^k + b]$

Dans le cas présent on obtient alors :

$$x^1 = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix} \left[ \left[ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 5 & 0 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -6 \\ 0 & 0 & 0 \end{pmatrix} \right] \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix} \right]$$

$$x^1 = \begin{pmatrix} 2 \\ 1 \\ -4 \end{pmatrix}$$

et

$$\epsilon^{(1)} = \|Ax^{(1)} - b\|$$

$$\epsilon^{(1)} = \left\| \begin{pmatrix} 4 & 1 & 0 \\ -1 & 3 & 6 \\ -2 & -5 & -3 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ -4 \end{pmatrix} - \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix} \right\|$$

$$\epsilon^{(1)} = \left\| \begin{pmatrix} 1 \\ -26 \\ -5 \end{pmatrix} \right\|$$

$$\epsilon^{(1)} = \sqrt{1^2 + (-26)^2 + (-5)^2}$$

$$\epsilon^{(1)} = \sqrt{(702)}$$

### II.2.3 Implémentation

Pour l'implémentation de cette méthode, nous utiliserons  $\epsilon$  comme suit :

$$\epsilon^{(k)} = p^k = \text{Max}_{i=1,\dots,n} |\bar{x}_i - \tilde{x}_i^k|$$

Où  $\bar{x}_i$  est le résultat attendu et  $\tilde{x}_i^k$  est l'approximation trouvée à l'étape  $k$ .

De plus on utilisera aussi une limite d'occurrence, pour pouvoir gérer les matrices où **Jacobi** diverge.