

# Rapport de projet mathématiques : Méthodes numériques pour la résolution de systèmes linéaires

Alix ANNERAUD - Amandine BURCON - Myriem ABID

# Sommaire

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Théorie . . . . .	3
<b>2</b>	<b>Références</b>	<b>4</b>
2.1	Bibliographie . . . . .	4
2.2	Biographies . . . . .	4
2.2.1	Johann Carl Friedrich Gauss . . . . .	4
2.2.2	André-Louis Cholesky . . . . .	4
2.2.3	Jacobi . . . . .	4
2.2.4	Philipp von Seidel . . . . .	4
<b>3</b>	<b>Travail préliminaire</b>	<b>6</b>
3.1	Type de donnée . . . . .	6
3.2	Implémentation des matrices et vecteurs . . . . .	6
<b>4</b>	<b>Méthodes directes</b>	<b>9</b>
4.1	Méthode triangulaire inférieure . . . . .	9
4.2	Méthode triangulaire supérieure . . . . .	10
4.3	Élimination de Gauss . . . . .	11
4.4	Factorisation “LU” . . . . .	12
4.5	Factorisation de Cholesky . . . . .	13
<b>5</b>	<b>Méthodes itératives</b>	<b>15</b>
5.1	Méthode de Jacobi . . . . .	15
5.2	Méthode de Gauss-Seidel . . . . .	15
<b>6</b>	<b>Code complet</b>	<b>17</b>
<b>7</b>	<b>Conclusion</b>	<b>18</b>

# 1 Introduction

## 1.1 Motivation

Les systèmes d'équations sont utiles à un bon nombres de domaines, par exemple :

— Electricité : recherche

Que ce soit en thermodynamique avec l'évolution thermique d'un système, en mécanique du point, en chimie ave. Cependant, ici nous allons nous intéressé à un cas concret.

## 1.2 Théorie

Le but est de résoudre l'équation linéaire de matrice :  $AX = B$  dans  $\mathbb{R}^N$  sans avoir à calculer  $A^{-1}$ . Car calculer  $A^{-1}$  peut poser problème lorsque  $N$  devient grand. En effet, pour  $N = 20$ , il faudra 5 fois l'age de l'univers pour calculer  $A^{-1}$ .

Nous allons donc à l'aide de plusieurs méthodes, tenter de simplifier le problème en sous-problèmes plus simple à résoudre et procéder à des optimisations graduelles afin d'optimiser le cout en temps de calcul et l'espace mémoire utilisé pour le cacul.

## 2 Références

### 2.1 Bibliographie

### Références

### 2.2 Biographies

#### 2.2.1 Johann Carl Friedrich Gauss



Avant d'étudier la méthode de Gauss, il peut être pertinent d'établir une courte biographie de son auteur. Surnommé "prince des mathématiques" par ses pairs, ce mathématicien, né le 30 avril 1777, a contribué de bien des manières à développer non seulement les mathématiques mais également les méthodes en astrophysique et en électromagnétisme. Johann Carl Friedrich Gauss a pu se pencher sur des problèmes dits classiques (depuis l'antiquité) en adoptant des méthodes et raisonnements modernes. Il démontre alors le théorème fondamental de l'algèbre que l'on connaît aujourd'hui sous le nom du théorème de d'Alembert-Gauss, il dédie également un ouvrage à la théorie des nombres contenant plusieurs démonstrations qui révolutionnent l'arithmétique, on lui doit en partie la forme actuelle des nombres complexes. En astrophysique, il met au point la méthode des moindres carrés permettant de minimiser les incertitudes dues aux mesures ce qui lui permet de déterminer exactement la position de Ceres, une planète naine du système solaire. Avec la contribution de Wilhelm Weber, Gauss formule deux théorèmes essentiels en électromagnétisme réfutant l'existence de monopôle magnétique et établissant une relation entre le flux d'un champ électrique sur une surface fermée et la charge électrique totale à l'intérieur de cette surface. Cette liste est loin de résumer tous ses travaux, dont une partie a été publiée après son décès le 23 février 1855, ces derniers sont très nombreux et ont servi de base de recherche pour d'autres mathématiciens et physiciens après lui. Il est donc intéressant d'étudier la solution apportée par Gauss et de l'appliquer à notre problématique. L'élimination de Gauss est expliquée ci-après.

#### 2.2.2 André-Louis Cholesky

André-Louis Cholesky, également appelé René Cholesky, est un polytechnicien français qui s'est engagé dans l'armée suite à sa formation. Au cours de sa carrière militaire, on lui assigne différentes missions qui l'emmènent à plusieurs endroits où il effectue notamment des travaux de triangulation. Il participe à la Première Guerre mondiale où il est blessé et il meurt le 31 août 1918 des suites de ses blessures. Cependant, on le connaît plus pour ses contributions aux mathématiques que pour sa carrière de militaire. En effet, il est l'auteur d'un manuscrit intitulé "Sur la résolution numérique des systèmes d'équations linéaires". Cette méthode est en fait une nouvelle approche de la méthode des moindres carrés et sera publiée 6 ans après sa mort.

#### 2.2.3 Jacobi

#### 2.2.4 Philipp von Seidel

Philipp von Seidel est un mathématicien et physicien Allemand. Né le 24 octobre 1821, il étudie dans de nombreuses villes dû au travail de son père, puis suit des cours privés sous la tutelle de L.C. Schnürlein, ancien élève de Gauss, pendant un an. Il suit ensuite l'enseignement d'autres grands représentants de leur

discipline tels que Dirichlet, Encke, Jacobi ou encore Neumann dans trois universités différentes. En 1846, Seidel obtient son doctorat grâce à sa thèse : "Sur la meilleure forme des miroirs dans un télescope", puis publie une autre thèse six mois plus tard sur un sujet entièrement différent : "Études sur la convergence et divergence des fractions continues", ce qui lui permet de devenir professeur à l'université de Munich. Des problèmes de vue le forcent à prendre une retraite anticipée, et n'étant pas marié, ce sont sa sœur puis une veuve qui s'occupent de lui en fin de vie. Il meurt le 13 août 1896.

Seidel se concentre tout au long de sa carrière sur ses travaux en optique et en astronomie ainsi que sur l'analyse mathématique, et décompose les aberrations optiques du premier ordre en cinq équations, appelées "équations de Seidel". Il applique la théorie des probabilités à l'astronomie, et l'utilise également pour étudier la fréquence de certaines maladies ou le climat. Il est particulièrement connu pour la méthode de Gauss-Seidel de résolution d'équation numérique, que nous décrirons dans ce rapport.

## 3 Travail préliminaire

### 3.1 Type de donnée

Tout d'abord, en C, comme la plupart des langages de programmation, il est impossible d'exprimer l'intégralité des réels, entiers, complexes ... car il faudrait disposer d'une mémoire infinie pour représenter une infinité de nombres. Comme nous utilisons des nombres réels pour les vecteurs et matrices, il faut choisir parmi les types signés flottant du C qui sont :

Type	Taille	Portée	Précision maximale
float	32 bits / 4 octets	$1,2 \cdot 10^{-38}$ à $3,4 \cdot 10^{38}$	6 chiffre après la virgule
double	64 bits / 8 octets	$2,3 \cdot 10^{-308}$ à $1,7 \cdot 10^{308}$	15 chiffres après la virgule
long double	80 bits / 10 octets	$3,4 \cdot 10^{-4932}$ à $1,1 \cdot 10^{4932}$	19 chiffres après la virgule

Dans ce projet, le point qui nous intéresse le plus est celui de la performance de la résolution, cette dernière étant très exigeante en temps processeur. Ainsi, le choix du type est crucial. Au final, nous avons choisi le type "double" car il offre une précision et une portée amplement suffisante, sans compromettre les performances de nos algorithmes. En effet, contrairement au type « long double », le type « double » nécessite pas de temps processeur supplémentaire par rapport à un type « float » sur les ordinateurs modernes. C'est à dire, dont les processeur possèdent des registres de calculs flottant de 64 bits / 8 octets. Il est à noter que cette considération aurait été différente il y 10 - 20 ans car les processeurs d'antan ne supportaient ne possédaient que des registres de calcul flottant de 32 bits, 16 bits voir 8 bits (pour les plus anciens). De plus, bien que le type « double » occupe deux fois plus de mémoire que le type « float », ici l'impact sur la mémoire reste relativement limité comparé à la précision supplémentaire offerte.

### 3.2 Implémentation des matrices et vecteurs

Ensuite, les concepts de matrice et de vecteur en C ne sont pas directement implémenté dans le langage. Cependant, nous pouvons utiliser une liste pour les vecteurs et des listes imbriquées dans des listes (tableau à 2 dimension) pour les matrices.

Ainsi, pour l'allocation des vecteurs et des matrices, nous avons implémenté les fonctions suivantes :

```
double *Allocation_Vecteur(int Taille)
{
    // Allocation d'un espace de mémoire de taille = Taille Vecteur *
    // Taille du type (8 octets).
    double * Vecteur = (double *)malloc(Taille * sizeof(double));

    // Nettoyage du vecteur.
    Nettoyage_Vecteur(Vecteur, Taille);

    return Vecteur;
}

double **Allocation_Matrice(int Taille)
{
    // Allocation de la première dimension
    double **Matrice = (double **)malloc(Taille * sizeof(double));

    // Allocation de la deuxième dimension
    for (int i = 0; i < Taille; i++)
    {
        Matrice[i] = Allocation_Vecteur(Taille);
    }

    // Nettoyage de la matrice (remplissage par des 0)
    Nettoyage_Matrice(Matrice, Taille);
}
```

```

    return Matrice;
}

```

Ces fonctions appellent les fonctions “Nettoyage\_Vecteur” et “Nettoyage\_Matrice”. Ces fonctions viennent nettoyer les matrices et vecteurs en les remplissant par des 0 car la mémoire alloué en C peut ne pas être toujours propre (contenir des valeurs aléatoires) :

```

void Nettoyage_Vecteur(double *Vecteur, int Taille)
{
    // Itère parmi les éléments du vecteur
    for (int i = 0; i < Taille; i++)
    {
        // Remplissage par des 0
        Vecteur[i] = 0;
    }
}

```

```

void Nettoyer_Matrice(double **A, int Taille)
{
    // Itère parmi la première dimension
    for (int i = 0; i < Taille; i++)
    {
        // Nettoyage de la deuxième dimension
        Nettoyage_Vecteur(A[i], Taille);
    }
}

```

Ensuite, afin de rendre l’affichage des vecteurs et matrices lisible, nous avons implémenté les fonctions suivantes :

```

void Afficher_Vecteur(double *Vecteur, int Taille)
{
    // Saut de ligne.
    printf("\n");
    // Itère parmi les éléments du vecteur.
    for (int i = 0; i < Taille; i++)
    {
        // Affichage de la valeur suivi d'un saut de ligne.
        printf("%f\n", Vecteur[i]);
    }
}

```

```

void Afficher_Matrice(double **Matrice, int Taille)
{
    // Itère parmi la première dimension de la matrice.
    for (int i = 0; i < Taille; i++)
    {
        // Retour à la ligne.
        printf("\n");

        // Itère parmi la deuxième dimension de la matrice.
        for (int j = 0; j < Taille; j++)
        {
            // Affichage de la valeur avec un séparateur.
            printf("|_ %f_", Matrice[i][j]);
        }
    }
}

```

```
    }  
    // Retour à la ligne.  
    printf("|\\n");  
}  
}
```

Enfin, comme nous utilisons de l'allocation dynamique pour les vecteurs et matrices, contrairement à l'allocation statique, la libération de la mémoire (désallocation) doit être faite manuellement :



## 4 Méthodes directes

### 4.1 Méthode triangulaire inférieure

#### Introduction

Soit  $B$  et  $X$  dans  $\mathbb{R}^N$   $A \in \mathbb{R}^{N \times N}$  une matrice triangulaire inférieure. On cherche, pour  $A$  et  $B$  donné,  $X$  tel que :

$$AX = B$$

#### Méthode

Soit  $N = 3$ , on a donc :

$$\begin{aligned} AX &= B \\ \Rightarrow \begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \\ \Rightarrow \begin{pmatrix} a_{11}x_1 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{pmatrix} &= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \\ \Rightarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 \\ \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{33}}x_1 - \frac{a_{32}}{a_{33}}x_2 \end{pmatrix} \end{aligned}$$

On en déduit alors une formule générale pour  $X$  :

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}}, i = 1, \dots, N.$$

#### Code

```
double *Sol_Inf(double **a, double *b, int Taille)
{
    // Allocation du vecteur X.
    double *x = Allocation_Vecteur(Taille);

    // Calcul du premier terme de X.
    x[0] = b[0] / a[0][0];

    // Itère parmis les lignes de la matrice.
    for (int i = 1; i < Taille; i++)
    {
        // Calcul de la somme des a[i][j] * x[j]
        double Sum = 0;
        for (int j = 0; j < i; j++)
        {
            Sum = Sum + a[i][j] * x[j];
        }
        // Calcul du terme X[i].
        x[i] = (b[i] - Sum) / a[i][i];
    }
}
```

```

    return x;
}

```

### Exemple

Ainsi, pour  $A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 1 & 4 & -1 \end{pmatrix}$  et  $B = \begin{pmatrix} 1 \\ 8 \\ 10 \end{pmatrix}$ , l'algorithme nous retourne  $X = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$ , ce qui est correct.

### Conclusion

Ainsi, cette algorithme possède une complexité temporelle maximale quadratique ( $O(N^2)$ ) car nous avons deux boucles imbriquées les unes dans les autres qui dépendent plus ou moins de la taille  $N$ .

## 4.2 Méthode triangulaire supérieure

### Introduction

Soit  $B$  et  $X \in \mathbb{R}^N$ ,  $A \in \mathbb{R}^{N \times N}$  une matrice triangulaire supérieure. Comme pour la section précédente, on cherche pour  $A$  et  $B$  donné,  $X$  tel que :

$$AX = B$$

### Exemple

Soit  $N = 3$ , on a :

$$\begin{aligned}
 AX &= B \\
 \Rightarrow \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \Leftrightarrow \\
 \Rightarrow \begin{cases} a_{33}x_3 = b_3 \\ a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \end{cases} \\
 \Rightarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \begin{pmatrix} \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 \\ \frac{b_2}{a_{22}} - \frac{a_{23}}{a_{22}}x_3 \\ \frac{b_3}{a_{33}} \end{pmatrix} = \begin{pmatrix} \frac{b_1}{a_{11}} - \frac{a_{21}}{a_{22}}x_1 \\ \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 \\ \frac{b_3 - \sum_{k=1}^2 a_{3k}x_k}{a_{33}} \end{pmatrix}
 \end{aligned}$$

On en déduit alors une formule générale pour  $X$  :

$$x_i = \frac{b_i - \sum_{j=i+1}^N a_{ij}x_j}{a_{ii}}, i = N, \dots, 1$$

### Code

Ecrire algorithme ici

### Exemple

Ainsi, pour  $A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 8 \\ 0 & 0 & 5 \end{pmatrix}$  et  $B = \begin{pmatrix} 6 \\ 16 \\ 15 \end{pmatrix}$ , l'algorithme nous retourne  $X = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$ , ce qui est correct.

## Conclusion

De manière similaire à la précédente, cette algorithme possèdent une complexité quadratique ( $O(N^2)$ ).

## 4.3 Élimination de Gauss

### Introduction

Maintenant que nous disposons des fonctions pour résoudre  $AX = B$  avec  $A$  une matrice triangulaire inférieure ou supérieure carrée. On va maintenant chercher à résoudre la même équation mais lorsque  $A$  est une matrice carrée quelconque. Nous allons utiliser la méthode de Gauss pour transformer l'expression  $AX = B$  où  $A$  est une matrice carrée en  $UX = e$  avec  $U$  qui est une matrice triangulaire supérieure. Ainsi, il ne restera plus qu'à résoudre  $UX = e$  avec la fonction « Sol\_Sup » pour obtenir  $X$ .

### Méthode

Soit  $N = 3$ , tel que :

$$\Rightarrow \underbrace{\begin{pmatrix} 3 & 1 & 2 \\ 3 & 2 & 6 \\ 6 & 1 & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_X = \underbrace{\begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix}}_B \Leftrightarrow \underbrace{\begin{pmatrix} a & b & c \\ 0 & d & f \\ 0 & 0 & g \end{pmatrix}}_U \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_X = \underbrace{\begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}}_e$$

On a donc :

$$\begin{array}{l} (1) \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ 3x_1 + 2x_2 + 6x_3 \\ 6x_1 + x_2 - x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix} \Leftrightarrow \begin{pmatrix} ax_1 + bx_2 + cx_3 \\ dx_2 + fx_3 \\ gx_3 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} \\ (2) \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ 3x_1 + 2x_2 + 6x_3 \\ 6x_1 + x_2 - x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix} \Leftrightarrow \begin{pmatrix} ax_1 + bx_2 + cx_3 \\ dx_2 + fx_3 \\ gx_3 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} \\ (3) \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ 3x_1 + 2x_2 + 6x_3 \\ 6x_1 + x_2 - x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix} \Leftrightarrow \begin{pmatrix} ax_1 + bx_2 + cx_3 \\ dx_2 + fx_3 \\ gx_3 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} \end{array}$$

On va maintenant éliminer  $x_1$  des équations (2) et (3) en utilisant (1) :

$$\begin{array}{l} (1) \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3) - \frac{6}{3}(1) \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_2 - 5x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \\ (2') \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3) - \frac{6}{3}(1) \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_2 - 5x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \\ (3') \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3) - \frac{6}{3}(1) \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_2 - 5x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \end{array}$$

Puis, on va éliminer  $x_2$  de l'équation (3') en utilisant (2') :

$$\begin{array}{l} (1) \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3') - \frac{-1}{1}(2') \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix} \\ (2') \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3') - \frac{-1}{1}(2') \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix} \\ (3'') \quad \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ (2) - \frac{3}{3}(1) \\ (3') - \frac{-1}{1}(2') \end{pmatrix} = \begin{pmatrix} 3x_1 + x_2 + 2x_3 \\ x_2 + 4x_3 \\ -x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix} \end{array}$$

Si on factorise par  $X$ , on a donc bien  $U$  une matrice triangulaire supérieure carrée :

$$\underbrace{\begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & -1 \end{pmatrix}}_U \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_X = \underbrace{\begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}}_e$$

Il ne reste plus qu'à utiliser « Sol\_Sup » pour résoudre cette équation.

### Code

```
Code de mes fesses
void KILL_ME()
```

### Exemple

Ainsi, pour  $A = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 2 & 1 \\ 3 & -1 & 1 \end{pmatrix}$  et  $B = \begin{pmatrix} 5 \\ 5 \\ 6 \end{pmatrix}$ , l'algorithme nous retourne  $X = \begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix}$ , ce qui est correct.

## Conclusion

Cette méthode est de complexité  $O()$ , ce qui est relativement efficace. Cependant, à chaque changement de  $B$ , il faut recalculer totalement les matrices  $U$  et  $e$ , ce qui est coûteux en temps de calcul.

## 4.4 Factorisation “LU”

### Introduction

Afin d’éviter de recalculer  $U$  si  $B$  change, on exprime  $A$  de notre expression initiale :

$$AX = B$$

En cette expression :

$$A = LU$$

Où  $L$  est une matrice triangulaire supérieure carrée et  $U$  une matrice inférieure carrée. Cette opération sera réalisé par la fonction “LU”. On a donc :

$$LUX = B$$

On peut maintenant passer à la résolution qui va se faire en deux étapes. Tout d’abord, nous allons exprimer  $Y = UX$  tel que :

$$LY = B$$

$L$  étant une matrice triangulaire inférieure carrée, il suffit d’utiliser la fonction « Sol\_Sup » afin d’obtenir  $Y$ .

Enfin, comme  $UX = Y$  avec  $U$  une matrice supérieure carrée, il suffit d’utiliser « Sol\_Inf » pour obtenir  $X$ .

### Méthode

#### Code

```
void LU(double **L, double **A, int Taille)
{
    Nettoyer_Matrice(L, Taille, Taille);

    // Rempli les 1 en diagonale
    for (int i = 0; i < Taille; i++)
    {
        L[i][i] = 1;
    }

    // Calcule L et U
    for (int i = 0; i < Taille - 1; i++)
    {
        for (int k = i + 1; k < Taille; k++)
        {
            double C = A[k][i] / A[i][i];

            L[k][i] = C;

            for (int j = 0; j < Taille; j++)
            {
                A[k][j] = A[k][j] - (C * A[i][j]);
            }
        }
    }
}
```

```

    }
  }
}

```

### Exemple

Ainsi, pour  $A = \begin{pmatrix} 3 & 1 & 2 \\ 3 & 2 & 6 \\ 6 & 1 & -1 \end{pmatrix}$ , l'algorithme nous retourne  $L = \begin{pmatrix} 1 & 0 & 0 \\ 5 & 1 & 0 \\ 3 & 0,875 & 1 \end{pmatrix}$ .

Puis, pour  $B = \begin{pmatrix} 6 \\ 16 \\ 15 \end{pmatrix}$  et  $L$ , « Sol\_Inf » retourne  $Y = \begin{pmatrix} 5 \\ -20 \\ 8,5 \end{pmatrix}$ .

Enfin, pour  $A$  et  $Y$ , « Sol\_Inf » retourne  $X = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$ , ce qui est correct.

### Conclusion

Cette fonction est de complexité temporelle  $O()$ .

Cependant, dans le cas où  $A$  est symétrique, il est possible de procéder à d'autres optimisations et diminuer le temps de calcul.

## 4.5 Factorisation de Cholesky

### Méthode

Comme vu précédemment, nous avons réussi à optimiser notre résolution dans le cas où  $B$  change afin d'éviter de tout recalculer. Dans le cas où  $A$  est symétrique, il est également possible de procéder à d'autres optimisations :

On a donc la matrice  $A$  qui est :

— Symétrique : c'est à dire que  $A = A^T$ .

— Définie positive, c'est à dire quelle est positive et inversible, tel que  $\langle AY, Y \rangle > 0, \forall Y \in \mathbb{R}^N \setminus \{\vec{0}\}$

On peut alors exprimer  $A$  en fonction de  $L$  une matrice triangulaire inférieure carrée dont la diagonale est positive :

$$A = L \cdot L^T$$

Si on remplace dans notre expression initiale, on a :

$$AX = B \Leftrightarrow L \cdot L^T \cdot X = B$$

On peut passer maintenant à la résolution. De manière similaire à la méthode « LU », nous allons procéder en deux étapes.

— Tout d'abord, la résolution de l'expression  $LY = B$  où  $Y = L^T X$  avec la fonction « Sol\_Inf ».  $L$  étant une matrice triangulaire inférieure carrée.

— Enfin, la résolution de l'expression  $L^T X = Y$  avec la fonction « Sol\_Sup ».  $L^T$  étant également une matrice triangulaire inférieure carrée.

### Application

Soit  $N = 4$ ,  $L$  une matrice de taille  $N * N$  :

On créer une matrice  $L$  tel que  $L \cdot L^T = A$ , on a :

$$\begin{aligned}
& \underbrace{\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}}_L \underbrace{\begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{pmatrix}}_{L^T} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}}_A \\
& \Leftrightarrow \begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} & l_{11}l_{41} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{32} & l_{21}l_{41} + l_{22}l_{42} \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 & l_{31}l_{41} + l_{32}l_{42} + l_{33}l_{43} \\ l_{41}l_{11} & l_{41}l_{21} + l_{42}l_{22} & l_{41}l_{31} + l_{42}l_{32} + l_{43}l_{33} & l_{41}^2 + l_{42}^2 + l_{43}^2 + l_{44}^2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}
\end{aligned}$$

On a donc pour les termes en diagonale :

$$\begin{aligned}
& - l_{11}^2 = a_{11} \Rightarrow l_{11} = \sqrt{a_{11}} \\
& - l_{21}^2 + l_{22}^2 = a_{22} \Rightarrow l_{22} = \sqrt{a_{22} - l_{21}^2} \\
& - l_{31}^2 + l_{32}^2 + l_{33}^2 = a_{33} \Rightarrow l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} \\
& - l_{41}^2 + l_{42}^2 + l_{43}^2 + l_{44}^2 = a_{44} \Rightarrow l_{44} = \sqrt{a_{44} - l_{41}^2 - l_{42}^2 - l_{43}^2}
\end{aligned}$$

On constate que :  $l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$

On a également pour les autres termes :

$$\begin{aligned}
& - l_{21}l_{11} = a_{21} \Rightarrow l_{21} = \frac{a_{21}}{l_{11}} \\
& - l_{31}l_{11} = a_{31} \Rightarrow l_{31} = \frac{a_{31}}{l_{11}} \\
& - l_{41}l_{11} = a_{41} \Rightarrow l_{41} = \frac{a_{41}}{l_{11}}
\end{aligned}$$

On constate que  $l_{ij} = \frac{a_{ij}}{l_{jj}} - \sum_{k=1}^{j-1} l_{ik}l_{jk}$

## Algorithme

[Ecrire code](#)

## Conclusion

Lors de la factorisation, on effectue

## 5 Méthodes itératives

Contrairement aux méthodes directes précédents où l'on cherche directement le vecteur  $X$  pour résoudre  $AX = B$ . Ici, nous allons nous intéresser.

### 5.1 Méthode de Jacobi

#### Méthode

Soit  $B, X \in \mathbb{R}^N$  et  $A \in \mathbb{R}^{N \times N}$ , on cherche toujours à résoudre l'équation  $AX = B$ .

On peut décomposer  $A$  en 3 matrices tel que  $A = E + F + G$  :

—  $D$  la matrice contenant la diagonale de  $A$  :  $D = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_{nn} \end{pmatrix}$ .

—  $E$  la matrice contenant la partie inférieur de  $A$  (sans sa diagonale) :  $E = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ A_{1j} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ A_{ij} & \cdots & A_{(i-1)j} & 0 \end{pmatrix}$ .

—  $F$  la matrice contenant la partie supérieure de  $A$  (sans la diagonale) :  $F = \begin{pmatrix} 0 & A & \cdots & A_{ij} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & A \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$  e

L'équation initiale  $AX = B$  devient alors :  $(E + D + F)X = B$ , ce qui donne

$$\Rightarrow DX = b - (E + F)X$$

$$\Rightarrow X = D^{-1}(b - [E + F]X)$$

$$\begin{cases} X^0 & k = 0 \\ X^{k+1} = D^{-1}(b - [E + F]X^k) & k = 1, \dots, N \end{cases}$$

Ainsi, pour un  $X^0$  donné, on peut calculer  $X_i^{k+1} = \frac{b_i - \sum_{j=1, j \neq i}^N A_{ij} X_j^k}{A_{ii}}$  pour  $i = 1, \dots, N$ . OK

On peut alors calculer la norme : Norm =  $\|x_{k+1} - x_k\|_2 = \sqrt{\sum_{n=0}^{N-1} (x_{k+1}[n] - x_k[n])^2}$

On définit  $\varepsilon = 0,001$  et  $X_k = X_0$ .

#### Algorithme

### 5.2 Méthode de Gauss-Seidel

#### Méthode

Les coeffs du vecteur  $X(k+1)$  sont plus précis que ceux de  $X(k)$ . Lorsqu'on calcule  $X(k+1)$ , on calcule les coefficients de ce vecteur un par un : on peut donc utiliser les coefficients de ce vecteur déjà calculés plutôt que ceux du vecteur précédent. Cependant, en calculant  $X_i(k+1)$  on ne connaît les coef de  $X(k+1)$  que jusqu'à  $i-1$ , on utilise ceux de  $X(k)$  pour le reste de la somme. Donc somme séparée en deux.

## **Algorithme**

dd

## **Conclusion**

Plus précis mais plus complexe



## 6 Code complet

## 7 Conclusion

Ainsi, nous avons découvert plusieurs méthodes possibles pour la résolution d'une équation de matrice de la forme  $AX = B$ , sans calculer  $A^{-1}$ . Voici un récapitulatif des différentes méthodes :

Méthode	Complexité temporelle	Avantages	
Triangulaire inférieure		-	
Triangulaire supérieure			- Fonction
Élimination de Gauss	$O\left(\frac{2n^3}{3}\right)$	- Résolution d'un	
La factorisation LU	$O\left(\frac{n^3}{3}\right)$	Méthode optimisé lorsque $A$ reste constant.	
La factorisation de Cholesky	$O\left(\frac{n^3}{6}\right)$		- Fonctionne un
Méthode de Jacobi			
Méthode Gauss-Seidel			

Ainsi, on constate que les méthodes ci-dessus sont complémentaires car elles présentent chacune des avantages et des inconvénients. Ainsi, le choix de la méthode doit se faire en fonction :

- La forme de  $A$  et  $B$ , afin d'optimiser le temps de calcul et éventuellement l'empreinte mémoire
- La précision de la résolution

Ainsi, il en revient à l'utilisateur de choisir quel méthode convient le mieux. Nous aurions pu éventuellement rédiger un algorithme qui fait le choix automatiquement.

nous avons trouvé différentes méthodes pour résoudre l'équation de matrice :  $AX = B$  sans à calculer directement  $A^{-1}$ .