Republique Algerienne Democratic and popular

Ministry of Higher Education and Scientific Research

Dr. Yahia Farès University of Medea

Faculty of Sciences - Department of Mathematics and Computer Science



Memory

Presented for the attainment

of the

Master's Degree in Mathematics

Specialization: Operations Research

By

MOULEBBAT MERIEM

Theme

Finite-difference discretization of a one-dimensional diffusion equation with logistic linear reaction for bounded solutions

Defended before the jury composed of:

M_r	BIBI Hamza	M.C.B	UYF Medea	President
M_r	ALIANE Mohamed	M.C.B	UYF Medea	Supervisor
M_r	SEBAOUI Madjid	M.C.A	UYF Medea	Examiner
M_{me}	GUTTAIE Ghania	M.C.A	UYF Medea	Examiner

June 24, 2024

Acknowledgement

First and foremost, I would like to thank **Allah**, the Merciful and Compassionate, for giving me the strength and courage to complete this modest work. Additionally, it is a pleasure to thank everyone who contributed to the completion of this memory.

First of all. I would like to express my sincerest thanks to my supervisor, Dr **M. ALIANE**, for guiding me throughout this study and for his scientific and moral support.

Dr **H. BIBI**, M.C.B at Dr.Yahia Fares University of Medea, for honouring us by chairing the jury for this memory.

I would like to thank Dr **M. SEBAOUI**, M.C.A at Dr.Yahia Fares the University of Medea, for honouring me by being a member of the jury and for dedicating some of his valuable time to read and correct this memory.

I would like to thank Dr **G. GUTTAI**, M.C.A at Dr.Yahia Fares the University of Medea, for honouring me by being a member of the jury and for dedicating some of his valuable time to read and correct this memory.

I would like to express my special thanks to my dear parents for their support and encouragement, which enabled me to continue my studies.

Finally. I would like to thank everyone who helped, whether directly or indirectly, in completing this project.

Dedication

To my dearest father, "Mohammed", who taught me that success comes only through patience and perseverance, to the man who believes in my courage and reminds me of my strength and capability, to my solid ground and sturdy wall.

To the one whose prayers are the secret of my success, to the smile of life and the essence of existence, "my beloved mother".

To my sisters: "Touba" and "Anfel".

To everyone who has been a support and a pillar on this journey, to loyal friends, to those who stood by me in hardships and crises, to those who poured out their feelings and advice, to my family, I dedicate this work, by the grace of God Almighty, for all praise is due to Allah, the Lord of all worlds.

MOULEBBAT MERIEM

Abstract

This Thesis focuses on the application of the finite difference method, combined with the Lyapunov method, to solve linear partial differential equations (PDEs) with homogeneous Dirichlet boundary conditions. PDEs are fundamental in the modeling of many physical and engineering phenomena, but their analytical resolution is often complex, if not impossible. Thus, digital methods become essential.

We start with an introduction to the basic concepts of PDEs and homogeneous Dirichlet boundary conditions. Next, we detail the finite difference method, a technique that discretizes the continuous domain into a lattice of points and approximates the partial derivatives by finite differences. This method is known for its simplicity of implementation and its effectiveness for problems defined on regular domains.

In order to improve the stability and convergence of the finite difference method, we integrate the Lyapunov method. The Lyapunov method, formulated by the matrix equation AX+XB=C, is used to analyze the stability of solutions and optimize numerical schemes. This approach allows the discrete PDE problem to be reformulated into a Lyapunov equation, thereby facilitating numerical solution while ensuring desirable properties such as stability and precision.

We then develop the practical implementation of this combined approach, detailing the algorithms and steps required to solve linear PDEs. Case studies and numerical simulations are presented to illustrate the effectiveness of our method. These examples show how the finite difference method, enhanced by Lyapunov analysis, can efficiently solve typical problems while respecting the constraints of homogeneous Dirichlet boundary conditions.

Finally, we discuss the advantages and limitations of this combined approach. We also discuss future perspectives for the improvement and extension of these techniques, including the adaptation to nonlinear problems.

ملخص

تُركِّز هذه الأطروحة على تطبيق طريقة الفروقات المحدودة، بالتكامل مع طريقة ليابونوف، لحل المعادلات التفاضلية الجزئية الخطية $(PDEs)$ مع شروط الحدود ديريتشله المتجانسة. تُعد المعادلات التفاضلية الجزئية أساسية في نمذجة العديد من الظواهر الفيزيائية والهندسية، إلا أن حلها التحليلي غالباً ما يكون معقداً، إن لم يكن مستحيلاً لذا، تصبح الأساليب الرقمية ضرورية.

نبدأ بمقدمة عن المفاهيم الأساسية للمعادلات التفاضلية الجزئية وشروط الحدود ديريتشله المتجانسة. بعد ذلك، نشرح طريقة الفروقات المحدودة، وهي تقنية تقوم بتجزئة المجال المستمر إلى شبكة من النقاط وتُقَدِّر المشتقات الجزئية من خلال الفروقات المحدودة. تُعرف هذه الطريقة بسهولتها في التنفيذ وفعاليتها في المسائل المعرفة على المجالات المنتظمة.

من أجل تحسين استقرار وتقارب طريقة الفروقات المحدودة، ندمج طريقة ليابونوف، التي تصاغ بواسطة معادلة المصفوفة $AX + XB = C$، لتحليل استقرار الحلول وتحسين المخططات العددية. تُمكِّن هذه الطريقة من إعادة صياغة مشكلة المعادلة التفاضلية الجزئية المتقطعة إلى معادلة ليابونوف، مما يُسهِّل الحل العددي مع ضمان خصائص مرغوبة مثل الاستقرار والدقة.

نقوم بعد ذلك بتطوير التطبيق العملي لهذا النهج المتكامل، مُفَصِّلين الخوارزميات والخطوات اللازمة لحل المعادلات التفاضلية الجزئية الخطية. يتم تقديم دراسات حالة ومحاكاة عددية لتوضيح فعالية طريقتنا. تُظهر هذه الأمثلة كيف يمكن لطريقة الفروقات المحدودة، المحسّنة بتحليل ليابونوف، أن تحل المشكلات النموذجية بكفاءة مع احترام قيود شروط الحدود ديريتشله المتجانسة.

أخيراً، نناقش مزايا وقيود هذا النهج المتكامل. كما نناقش آفاق المستقبل لتحسين وتوسيع هذه التقنيات، بما في ذلك التكيف مع المشكلات غير الخطية.

List of Figures

1.1 Numerical methods for solving linear system $Gx = c$. 8
1.2 Comparison of CPU times for different methods for solving $Gx = c$. 24

2.1 Comparison of CPU Times for Different methods for solving $LX + XU = C$. 35

3.1 Comparison between analytical solution and numerical solution for $T = 1$ and $I = [-1, 1]$. 41
3.2 Comparison between analytical solution and numerical solution for $T = 1$ and $I = [-5, 5]$. 41
3.3 Comparison between analytical solution and numerical solution for $T = 2$ and $I = [-5, 5]$. 42
3.4 Comparison between analytical solution and numerical solution for $T = 5$ and $I = [-10, 10]$. 42

List of Tables

1.1 Comparison of CPU Times for Different methods for solving $Gx = c$ 24

2.1 Comparison of CPU Times for Different methods for solving $LX + XU = C$. 34

Contents

Introduction **1**

1 Numerical methods for solving linear system $Gx = c$ **4**

1.1 Introduction 4

1.2 Specific Cases for Matrix G 5

1.2.1 Identity Matrix 5

1.2.2 Diagonal Matrix 6

1.2.3 Upper Triangular System 6

1.2.4 Lower Triangular System 7

1.3 Direct Methods for Solving Linear Systems 8

1.3.1 Gauss Elimination Method 8

1.3.2 LU Factorization 10

1.3.3 Cholesky Decomposition 13

1.4 Iterative methods for solving linear systems 16

1.4.1 Jacobi method 18

1.4.2 Gauss-Seidel Method 21

1.5 Numerical Comparison 23

2 Lyapounouv Sylvester equation **26**

2.1 Introduction 26

2.2 Kronecker product 26

2.3 The Sylvester equation 28

2.3.1 Transformation by using the Kronecker Product 29

2.3.2 Algorithmic Complexity 30

2.4 Existence and Uniqueness of the Solution 30

2.5 Specific cases for lower and upper matrices 31

2.6 Numerical Comparison 34

3 Numerical solution of PDE using finite-difference method combined with Lyapunov equation 36

3.1 Statment of the problem 36

3.2 Discretization 36

3.3 Matrix representation 37

3.3.1 Model 1 38

3.3.2 Model 2 39

Conclusion 44

Bibliography 46

Introduction

In practical life, there is a pressing need to solve linear systems $Gx = c$, where G represents the coefficient matrix, x is the vector of unknowns, and c is the vector of known numbers. These systems are used in various engineering and scientific applications, such as calculating forces in mechanical structures, solving distribution and storage problems in computer science, and analyzing electrical circuits in electrical engineering.
Additionally, we encounter the Sylvester equation $AX + XB = C$, which involves an unknown matrix X, and three known matrices A, B, and C. This equation finds applications in a wide range of engineering and scientific fields, such as controlling dynamic systems, designing filters, and analyzing various engineering scenarios .
Finally, solving partial differential equations (PDEs) presents another challenge, where research suggests that using finite difference methods to solve these equations can be complex. However, the process can be simplified, and accurate and efficient results achieved using the Lyapunov equation [15]and [11]. These methods are used in various applications, such as fluid dynamics, heat transfer, and aerodynamics.
Therefore, this research focuses on developing and improving numerical methods to solve linear systems [5] , including Sylvester equations [7]and [8] , and solving partial differential equations using the Lyapunov equation, and integrating them with finite difference methods [12]and [16] .

The linear matrix equation $Gx = c$ is a common form of a linear system of equations. Historically, the study and solution of linear systems date back thousands of years and have significantly impacted the development of mathematics.

Ancient Origins: Linear systems appeared in the history of ancient mathematics. For example, the Chinese used methods similar to matrix operations in "The Nine Chapters on the Mathematical Art" (circa 200 BC). In this text, mathematical problems involving linear systems were solved using techniques akin to Gaussian elimination.

Development of Linear Algebra: In the 19th century, the concept of matrices and determinants was formalized, paving the way for modern methods to solve linear systems of equations [6]. Mathematicians such as Arthur Cayley and James Joseph Sylvester contributed to developing the theory of matrices, which is central to linear algebra.

Gaussian Elimination [1] named after Carl Friedrich Gauss, this method is a classic technique for solving systems of linear equations. It involves using simple operations to put a matrix in echelon form, allowing the linear system to be solved.

Modern Applications: Today, the equation $Gx = c$ is used in many fields, including engineering, physics, economics, computer science, and others. Software like MATLAB or NumPy (for Python) provides powerful tools for solving large-scale linear systems quickly.

Thus, the linear matrix equation $Gx = c$ has a long history and remains a fundamental part of applied mathematics, with applications that extend to virtually all aspects of science and engineering.

The Sylvester equation is a matrix equation that takes the following form: $AX + XB = C$ where A, B and C are given matrices and the goal is to find the unknown matrix X [10]. This equation is named after the mathematician James Joseph Sylvester, who made significant contributions to linear algebra and determinants in the 19th century. Sylvester greatly contributed to the development of linear algebra and laid down some fundamental concepts in this field. The Sylvester equation has broad applications in various fields, including mathematics, engineering, computer science and physics.

This memory is organized as follows, in the first chapter, we recall the basic concepts of linear systems $Gx = c$ [2] . We give a brief overview of the methods for solving these systems, starting with direct methods such as Cramer's formulas, Gauss elimination, LU factorization, and Cholesky decomposition [4], and moving on to iterative methods like Jacobi and Gauss-Seidel [5] and [3]. Various cases for the matrix G are examined, detailing each method and its applications [1] . We particularly focus on direct methods decomposition, and introduce iterative methods to provide a comprehensive understanding of the computational complexity of each method and when it is appropriate to use them. This detailed knowledge is aimed to be useful for students and professionals in mathematics and engineering.

Chapter two is organized as follows: In section 1, we introduce the problem of the Lyapunov-Sylvester equation and explain its significance [7]. In section 2 , we propose

an algorithm that utilizes the Kronecker product to transform the original Lyapunov-Sylvester equation into a standard linear system that can be efficiently solved [8]. In section 3, we elucidate the proposed algorithm through three examples of Lyapunov-Sylvester problems, where these examples are solved using three methods: S-LUS , S-KPLUS algorithm, in addition to LMM [9]. Finally, we compare the aforementioned methods through numerical simulation using the MATLAB programming language, where we provide a comprehensive comparison of CPU times for these algorithms, demonstrating their performance across a variety of matrix sizes.

Chapter third is structured as follows: In section 1, we present the problem statement and challenges, along with dividing the spatial and temporal domain and using finite-difference approximations [12]. Then, in section three, we review the finite-difference scheme that updates the solution at each grid point according to time steps [11]. Next, we present the boundary conditions imposed on the domain, ensuring compatibility with the initial conditions. Subsequently, we represent the problem in matrix form to enhance computational efficiency, focusing on implementation using MATLAB. We provide an example of a well-known problem where we compare the numerical solution with the exact solution [13] and [16]. Finally, we evaluate performance by analyzing execution times and comparing results for different grid sizes and time intervals [14].

Chapter 1

Numerical methods for solving linear system $Gx = c$

1.1 Introduction

In this chapter, we propose to study some methods for solving systems of n equations with n unknowns of the form:

$$\begin{cases} g_{11}x_1 + g_{12}x_2 + \ldots + g_{1n}x_n = c_1 \\ g_{21}x_1 + g_{22}x_2 + \ldots + g_{2n}x_n = c_2 \\ g_{31}x_1 + g_{32}x_2 + \ldots + g_{3n}x_n = c_3 \\ \vdots \\ g_{n1}x_1 + g_{n2}x_2 + \ldots + g_{nn}x_n = c_n, \end{cases} \tag{1.1}$$

let G be a matrix $\mathcal{M}_n(\mathbb{R})$ (the set of square matrices of order n with elements in $\mathbb{R}$), c be the vector with components $(c_1, c_2, ..., c_n)^T$ and x be the unknown vector with components $(x_1, x_2, ..., x_n)^T$. Solving the system (1.1) is then equivalent to solving the matrix equation $Gx = c$. To solve this type of system, the first idea that comes to mind is to use Cramer's formulas, given that we assume G is invertible, then $det(G) \neq 0$, therefore

$$x_i = \frac{1}{det(G)} \begin{bmatrix} g_{11} & \ldots & g_{1,i-1} & c_1 & g_{1,i+1} & \ldots & g_{1n} \\ g_{21} & \ldots & g_{2,i-1} & c_2 & a_{2,i+1} & \ldots & g_{2n} \\ \vdots & & & & & & \\ g_{n1} & \ldots & g_{n,i-1} & c_n & g_{n,i+1} & \ldots & g_{nn} \end{bmatrix}, i = 1 \ldots n, \tag{1.2}$$

as a result, solving the system (1.1) using these formulas requires a total of $T_{Cramer} \approx (n+1)^2 n! - 1$ elementary operations. For example, for $n = 5$, $T_{Cramer} \approx 4319$, and for

$n = 10$, $T_{Cramer} \approx 4.10^8$, in this case, it becomes very slow in terms of execution time (or computation)as soon as n exceeds 4, another method is to calculate the inverse matrix of G, that is:

$$G^{-1}Gx = G^{-1}c \text{ hence } x = G^{-1}c.$$

For this, *a* total of $T_{inverse} \approx n!(n^2 + n + 1) + 3n^2 - n$ elementary operations are used. For example, for $n = 5$, $T_{inverse} \approx 3790$, and for $n = 10$, $T_{inverse} \approx 4.10^8$. This method is not more advantageous than the first. We resort to numerical methods with acceptable computation times and an operation count of order n^3. These methods can be:

- Direct methods: used only if the number of equations is not too high ($n \leq 100$).
- Iterative methods: used as soon as the system size is large ($n \geq 100$), which does not prevent them from being useful for small systems as well.

Definition 1.1.1. *A direct method for solving the system (1.1) is a method that leads to the exact solution of the problem after a finite number of arithmetic operations based on the system's dimension. We will explore specific cases such as the diagonal matrix, identity matrix, lower triangular matrix, and upper triangular matrix.*

1.2 Specific Cases for Matrix G

1.2.1 Identity Matrix

$$\begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}, \tag{1.3}$$

- Calculate x_i for each i from 1 to n : $x_i = c_i$.

Example 1.2.1.

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } c = \begin{pmatrix} 6 \\ 9 \\ 12 \end{pmatrix},$$

we calculate $x_1 = 6$, $x_2 = 9$, $x_3 = 12$.

$$x = \begin{pmatrix} 6 \\ 9 \\ 12 \end{pmatrix}.$$

1.2.2 Diagonal Matrix

$$\begin{pmatrix} g_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & g_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}, \tag{1.4}$$

- Calculate x_i for each i from 1 to n : $x_i = \frac{c_i}{g_{ii}}$.

Example 1.2.2.

$$G = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \text{ and } c = \begin{pmatrix} 6 \\ 9 \\ 12 \end{pmatrix},$$

we calculate $x_1 = \frac{6}{2}$, $x_2 = \frac{9}{3}$, $x_3 = \frac{12}{4}$.

$$x = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}.$$

1.2.3 Upper Triangular System

If G is upper triangular matrix, the system (1.1) can be expressed in the following form:

$$\begin{pmatrix} g_{11} & g_{12} & \dots & \dots & \dots & g_{1n} \\ 0 & g_{22} & \dots & \dots & \dots & g_{2n} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & g_{ii} & \dots & g_{in} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & g_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{pmatrix}, \tag{1.5}$$

since we assume that G is invertible, then $det(G) \neq 0$, i.e. $g_{ii} \neq 0$ for all $i = 1, \dots, n$. The solution to the system $Gx = c$ is:

Algorithm 1: Upper Triangular System

```
input:G,c
output:x
for i = n:-1:1 do
    x(i) = (c(i) - G(i, i+1:n) * x(i+1:n)) / G(i, i);
end
```

The complexity of algorithm (1) is of order $O(n^2)$.

1.2.4 Lower Triangular System

If G is lower triangular, the system (1.1) can be expressed in the following form:

$$\begin{pmatrix} g_{11} & 0 & \dots & \dots & \dots & 0 \\ g_{21} & g_{22} & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ g_{i1} & \vdots & \ddots & g_{ii} & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ g_{n1} & g_{n2} & g_{n3} & \dots & \dots & g_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{pmatrix}, \tag{1.6}$$

since we assume that G is invertible, then $det(G) \neq 0$, i.e. $g_{ii} \neq 0$ for all $i = 1, \dots, n$. The solution to the system $Gx = c$ is:

Algorithm 2: Lower Triangular System

```
input:G,b
output:x
for i = n do
    x(i) = (c(i) - G(i, 1:i-1) * x(1:i-1)) / G(i, i);
end
```

The complexity of algorithm (2) is of order $O(n^2)$.

Final Note: In conclusion, it should be noted that if the matrix G does not follow one of the discussed specific structures (diagonal, identity, upper triangular, or lower triangular), solving the system of equations $Gx = c$ may become complex. In such cases, it is often preferable to use advanced numerical methods such as Gauss method, LU decomposition, or iterative methods like Jacobi or Gauss-Seidel.

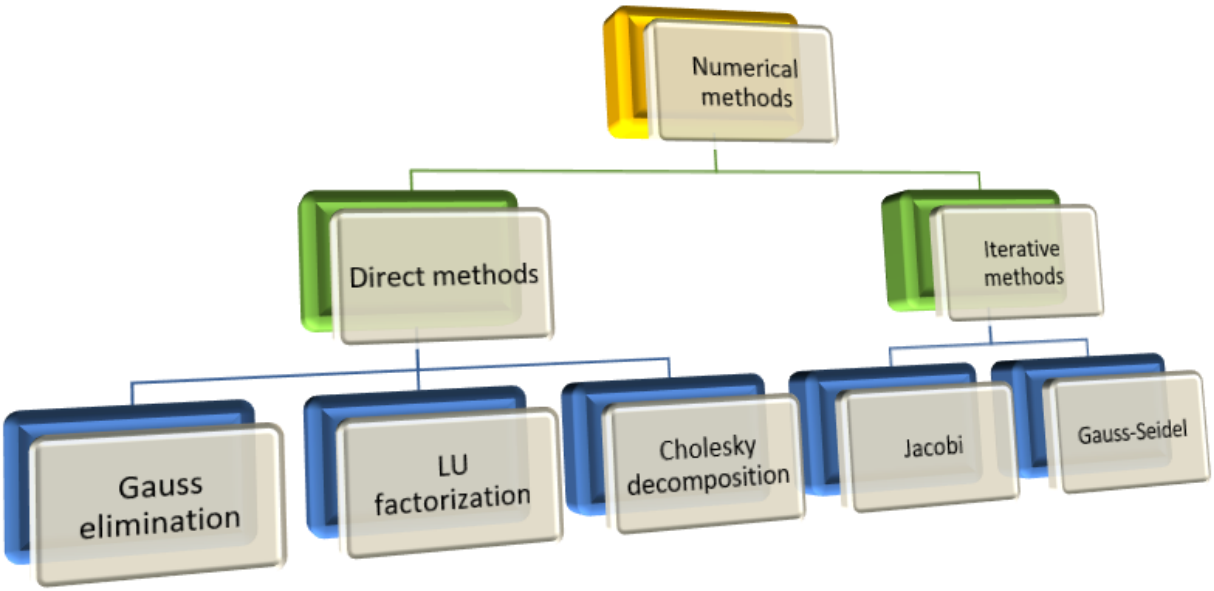


Figure 1.1: Numerical methods for solving linear system $Gx = c$.

1.3 Direct Methods for Solving Linear Systems

1.3.1 Gauss Elimination Method

Theorem 1.3.1 (*Gauss's Theorem*)**.** *For a square matrix G, there exists an invertible matrix M, such that the matrix $G' = MG$ becomes upper triangular. The Gauss method transforms the system (1.1) into an equivalent triangular system (1.3) using an elimination algorithm with only three elementary operations:*

- *Operation $(l_i \to \lambda l_i)$: Replaces line i with a multiple of itself.*
- *Operation $(l_i \to l_j)$: Intervene line i and line j .*
- *Operation $(l_i \to l_i + \lambda l_j)$: Replace line i the line i plus one multiple of line j.*

These three elementary operations are allowed, as they enable us to obtain a triangular system equivalent to the original system (1.1), meaning they have the same solution.

Note 1. *During elimination, it is possible for a pivot to become zero in the natural order of rows. For stability reasons, it is beneficial at each elimination step to choose the pivot such that $|g_{kk}^{(p)}|$ is as large as possible. This is known as the method of partial pivoting in Gauss.*

The Gauss elimination method involves eliminating all terms below the diagonal of matrix G, requiring a total of $T_{Gauss} = (4n^3 - 9n^2 + 7n)/6$ elementary operations. For example, for $n = 5$, $T_{Gauss} = 115$, and for $n = 10$, $T_{Gauss} = 805$. This demonstrates the significant improvement this method brings compared to other methods.

Note 2. *The Gauss method allows the computation of the determinant of matrix G using the formula*

$$det(G) = (-1)^p \prod_{k=1}^{n} g_{kk}^{(p)}.$$

where p is the number of permutations.

Algorithm 3: Gauss Elimination

```
function x = gauss(G, c)
clc; clear all
input:G
output:s,x
n = length(c);
for k = 1:n do
    for i = k+1:n do
        c(i) = c(i) - G(i, k)/G(k, k) * c(k);
        G(i, k:n) = G(i, k:n) - G(i, k)/G(k, k) * G(k, k:n);
    end
end
x = zeros(n, 1);
for i = n:-1:1 do
    x(i) = (c(i) - G(i, i+1:n) * x(i+1:n)) / G(i, i);
end
```

Example 1.3.1. *Perform Gaussian elimination on the following linear systems:*

$$\begin{pmatrix} 2 & 4 & 4 \\ 1 & 3 & 1 \\ 1 & 5 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -6 \end{pmatrix}, \tag{1.7}$$

we write the first system in the form of a table.

$$\begin{pmatrix} 2 & 4 & 4 & 2 \\ 1 & 3 & 1 & 1 \\ 1 & 5 & 6 & -6 \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3, \end{matrix} \tag{1.8}$$

we perform Gaussian elimination. Successively, we have:

$$\begin{pmatrix} 2 & 4 & 4 & 2 \\ 0 & 1 & -1 & 0 \\ 0 & 3 & 4 & -7 \end{pmatrix} \begin{matrix} \\ L_2' \leftarrow L_2 - 0,5L_1 \\ L_3' \leftarrow L_3 - 0,5L_1, \end{matrix} \tag{1.9}$$

$$\begin{pmatrix} 2 & 4 & 4 & 2 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 7 & 7 \end{pmatrix} \quad L_3'' \leftarrow L_3 - 3L_2', \tag{1.10}$$

we then obtain the following triangular system:

$$\begin{cases} 2x_1 + 4x_2 + 4x_3 = 2 \\ \quad x_2 - x_3 = 0 \\ \quad 7x_3 = -7, \end{cases} \tag{1.11}$$

using the back-substitution algorithm (3), we have successively $x_3 = -1$ *;* $x_2 = -1$ *;* $x_1 = 5$*, the Gaussian elimination above is said to be without permutation.*

1.3.2 LU Factorization

These methods involve decomposing the matrix G by expressing it as $G = LU$, where L is a lower triangular matrix, and U is an upper triangular matrix. The system (1.1) can then be written as:

$$Gx = c \Leftrightarrow L(Ux) = c,$$

which can be decomposed into two triangular systems:

$$\underbrace{\begin{pmatrix} g_{11} & g_{12} & \dots & \dots & g_{1n} \\ g_{21} & g_{22} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & \dots & g_{nn} \end{pmatrix}}_{G=LU} = \underbrace{\begin{pmatrix} l_{11} & 0 & \dots & \dots & 0 \\ l_{21} & l_{22} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 1 & u_{12} & \dots & \dots & u_{1n} \\ 0 & 1 & \dots & \dots & u_{2n} \\ \vdots & \ddots & \ddots & \dots & u_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}}_{U}. \tag{1.12}$$

A necessary and sufficient condition for an invertible matrix G to be decomposed into the form $G = LU$ is that $det(G_k) \neq 0$ for all $k = 1, ..., n$ where $G_k = (g_{ij})_{i,j=1}^{k}$ (i.e. all minors are non-zero).

Note 3. *It should be emphasized that the LU decomposition is not unique, so arbitrary choices must be made beforehand. The two most popular choices consist of either imposing that the matrix U has ones on the diagonal (Crout's decomposition) or that the matrix L has ones on the diagonal (Doolittle's decomposition).*
The determinant of matrix G can be calculated easily; indeed,

$$det(G) = det(L) \times det(U).$$

For the Doolittle method, since $det(L) = 1$, we have

$$det(G) = det(U) = \prod_{k=1}^{n} u_{kk}.$$

For the Crout method, since $det(U) = 1$, we have

$$det(G) = det(L) = \prod_{k=1}^{n} l_{kk}.$$

Example 1.3.2. *Let's consider the matrix:*

$$G = \begin{pmatrix} 4 & 8 & 8 \\ 4 & 5 & 6 \\ 2 & 1 & 3 \end{pmatrix}, det(G) = -12.$$

Solving equation (1.12)*, which results in after substitution:*

$$LU = \begin{pmatrix} l_{11} & u_{12}l_{11} & u_{13}l_{11} \\ l_{21} & u_{12}l_{21} + l_{22} & u_{13}l_{21} + u_{23}l_{22} \\ l_{31} & u_{12}l_{31} + l_{32} & u_{13}l_{31} + u_{23}l_{32} + l_{33} \end{pmatrix},$$

then

$$\begin{cases} l_{11} = 4 \\ l_{21} = 4 \\ l_{31} = 2 \\ u_{12}l_{11} = 8 \\ u_{13}l_{11} = 8 \\ u_{12}l_{21} + l_{22} = 5 \\ u_{12}l_{31} + l_{32} = 1 \\ u_{13}l_{21} + u_{23}l_{22} = 6 \\ u_{13}l_{31} + u_{23}l_{32} + l_{33} = 3 \end{cases} \Leftrightarrow \begin{cases} l_{11} = 4 \\ l_{21} = 4 \\ l_{31} = 2 \\ u_{12} = 2 \\ u_{13} = 2 \\ l_{22} = -3 \\ l_{32} = -3 \\ u_{23} = \frac{4}{3} \\ l_{33} = 3 \end{cases},$$

thus, the LU decomposition of matrix G is:

$$\underbrace{\begin{pmatrix} 4 & 8 & 8 \\ 4 & 5 & 6 \\ 2 & 1 & 3 \end{pmatrix}}_{G} = \underbrace{\begin{pmatrix} 4 & 0 & 0 \\ 4 & -3 & 0 \\ 2 & -3 & 3 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 1 & 2 & 2 \\ 0 & 1 & \frac{4}{3} \\ 0 & 0 & 1 \end{pmatrix}}_{U}.$$

Algorithm 4: LU Factorization

```
function factorisation(G,c)
input:G,c
output:x
n=length(c);
L=zeros(n);
U=eye(n);
for k=1:n do
    for i=k:n do
        L(i,k)=G(i,k)-L(i,1:k-1)*U(1:k-1,k);
    end
    for i=k+1:n do
        U(k,i)=(G(k,i)-L(k,1:k-1)*U(1:k-1,i))/L(k,k);
    end
end
y=zeros(n,1);
for i=1:n do
    y(i)=(c(i)-L(i,1:i-1)*y(1:i-1))/L(i,i);
end
x=zeros(n,1);
for i=n:-1:1 do
    x(i)=(y(i)-U(i,i+1:n)*x(i+1:n))/U(i,i);
end
```

1.3.3 Cholesky Decomposition

This method involves factorizing a symmetric positive definite matrix G into a product of two matrices, such that $G = LL^T$, where L is a lower triangular real matrix. The system (1.1) can then be written as:

$$Gx = c \Leftrightarrow L(L^T x) = c,$$

which can be decomposed into two triangular systems:

$$\underbrace{\begin{pmatrix} g_{11} & g_{12} & \dots & \dots & g_{1n} \\ g_{21} & g_{22} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & \dots & g_{nn} \end{pmatrix}}_{G=LL^T} = \underbrace{\begin{pmatrix} l_{11} & 0 & \dots & \dots & 0 \\ l_{21} & l_{22} & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} l_{11} & l_{21} & \dots & \dots & l_{n1} \\ 0 & l_{22} & \dots & \dots & l_{n2} \\ \vdots & \ddots & \ddots & \dots & l_{n3} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & l_{nn} \end{pmatrix}}_{L^T}. \tag{1.13}$$

A necessary and sufficient condition for an invertible matrix G to be decomposed into the form $G = LL^T$.

Note 4. *The Cholesky decomposition indeed offers an advantage for calculating the determinant. Since Lis triangular, we have:*

$$\det(G) = \det(L^T L) = \det(L) \times \det(L^T) = \det(L)^2 = \left(\prod_{i=1}^{dim\ L} l_{ii} \right)^2.$$

Note 5. *The Cholesky method utilizes:*

$$\begin{cases} \frac{1}{6}(n^3 + 6n^2 - 7n) & \textit{additions and subtractions} \\ n^2 + \frac{3}{2}n & \textit{divisions} \\ n & \textit{square roots.} \end{cases}$$

Given a total of $T_{Cholesky} = \frac{1}{6}(2n^3 + 15n^2 + n)$ *elementary operationsm, by example for* $n = 5$, $T_{Cholesky} = 105$ *and for* $n = 10$, $T_{Cholesky} = 585$.

Example 1.3.3. *Let's consider the matrix:*

$$G = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 5 \end{pmatrix}, det(G) = 16.$$

Solving equation (1.13)*, which results in after substitution:*

$$LL^T = \begin{pmatrix} l_{11}^2 & l_{21}l_{11} & l_{31}l_{11} \\ l_{11}l_{21} & l_{21}^2 l_{22}^2 & l_{31}l_{21} + l_{32}l_{22} \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{pmatrix},$$

then

$$\begin{cases} l_{11}^2 = 4 \\ l_{11}l_{21} = 2 \\ l_{11}l_{31} = 2 \\ l_{21}^2 l_{22}^2 = 2 \\ l_{21}l_{31} + l_{22}l_{32} = 1 \\ l_{31}^2 + l_{32}^2 + l_{33}^2 = 5 \end{cases} \Leftrightarrow \begin{cases} l_{11} = 2 \\ l_{21} = 1 \\ l_{31} = 1 \\ l_{22} = 1 \\ l_{32} = 0 \\ l_{33} = 2 \end{cases},$$

thus, the LL^T *decomposition of matrix* G *is:*

$$\underbrace{\begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 5 \end{pmatrix}}_{G} = \underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}}_{L^T}.$$

Algorithm 5: Cholesky Decomposition

```
function cholesky
input:G,c
output:x
n=length(c);
L=zeros(n);
for i=1:n do
    L(i,i) = sqrt(g(i,i) − sum(L(i,1:i−1).^2));
    for j=i+1:n do
        L(j,i) = (g(i,j) − sum(L(i,1:i−1).*L(j,1:i−1)))/L(i,i);
    end
end
y=zeros(n,1);
for i=1:n do
    y(i) = (c(i) − L(i,1:i−1)*y(1:i−1))/L(i,i);
end
U = L';
x = zeros(n,1);
for i=n:-1:1 do
    x(i) = (y(i) − U(i,i+1:n)*x(i+1:n))/U(i,i);
end
```

1.4 Iterative methods for solving linear systems

In this section, we consider the system (1.1), which is a system of linear equations of order n. We have seen that to numerically solve this system, it suffices to proceed with Gaussian elimination with or without pivot change, or alternatively, to perform an *LUdecomposition* of G (or LL^T if G is symmetric positive definite), then solve two triangular systems. In the case where the matrix G is full, the number of operations required to implement these algorithms is of the order of n^3, which can be enormous when n is large. Under these conditions, it becomes much more convenient to seek the roots of this system using approximate numerical methods called iterative methods. The principle

of these methods is to construct a sequence of vectors $x^{(0)}, x^{(1)}, ..., x^{(n)}$, defined by:

$$\begin{cases} x^{(0)} \text{ initial vector} \\ x^{(n+1)} = Bx^{(n)} + b, \end{cases} \tag{1.14}$$

where B is an $n \times n$ matrix called the iteration matrix and b is a vector in $\mathbb{R}^n$ such that:

$$\lim_{n\to\infty} \|x - x^{(n)}\| = 0.$$

In this case, we say that the sequence $\{x^{(n)}\}$ converges towards the exact solution of system (1.1), and we can write: $x = Bx + b$. Therefore, this equation is of the fixed-point type. Note that the indices i of x^i are placed above the symbol x to avoid confusion with the i-th component x_i.

- The practical use of these methods requires not only convergence but also that this convergence is not too slow.

The decomposition methods are a prime example of iterative methods. Let's assume we have decomposed G into the difference of two square matrices of order N, K and M, i.e., $G = K - M$. If K is a regular matrix, we can transform equation $Gx = c$ as follows:

$$(K - M)x = c \to Kx = Mx + c.$$

In this case, we can take $B = K^{-1}M$ and $b = K^{-1}c$. Thus, equation (1.14) becomes:

$$\begin{cases} x^{(0)} \text{ given initial value} \\ x^{(n+1)} = K^{-1}Mx^{(n)} + K^{-1}c \quad . \end{cases} \tag{1.15}$$

If K is diagonal or triangular. To achieve this, we write the matrix G in the form:

$$G = D - E - F, \tag{1.16}$$

where D is the diagonal matrix formed by the elements $(g_{ii})_{1\leq i\leq n}$, $-E$ is the matrix formed by the sub-diagonal elements of G, so $-E$ can be represented as follows:

$$-E = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ g_{21} & 0 & 0 & \cdots & 0 \\ g_{31} & g_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & \cdots & 0 \end{pmatrix}. \tag{1.17}$$

$-F$ is the matrix formed by the elements above the diagonal of G, so $-F$ can be represented as follows:

$$-F = \begin{pmatrix} 0 & g_{12} & g_{13} & \cdots & g_{1n} \\ 0 & 0 & g_{23} & \cdots & g_{2n} \\ 0 & 0 & 0 & \cdots & g_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{1.18}$$

Schematically, we have:

$$G = D - E - F = \begin{pmatrix} g_{11} & g_{12} & g_{13} & \cdots & g_{1n} \\ g_{21} & g_{22} & g_{23} & \cdots & g_{2n} \\ g_{31} & g_{32} & g_{33} & \cdots & g_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & \cdots & g_{nn} \end{pmatrix}. \tag{1.19}$$

If we assume that $g_{ii} \neq 0$ for all $i = 1, 2, \ldots, n$, then D and $(D - E)$ are regular, and we can choose, for example, $K = D$ (Jacobi method) or $K = D - E$ (Gauss-Seidel method).

1.4.1 Jacobi method

Let's go back to the relations (1.15) and (1.19). If we define $K = D$ and $M = E + F$, we can express the system (1.1) as follows:

$$g_{ii}x_i = c_i - \sum_{j<i} g_{ij}x_j - \sum_{j>i} g_{ij}x_j; \quad \text{for all } i = 1, 2, \ldots, n.$$

So, we can express the relation (1.15) component by component as follows:

$$x_i^{(n+1)} = \frac{1}{g_{ii}} \left(c_i - \sum_{j=1; j \neq i}^{n} g_{ij} x_j^{(n)} \right); \quad \text{for all } i = 1, 2, \ldots, n. \tag{1.20}$$

This relation allows us to explicitly compute the components $x_i^{(n+1)}$ for all $i = 1, 2, \ldots, n$ of the vector $x^{(n)}$. The matrix J defined by:

$$J = K^{-1}M = D^{-1}(E + F), \tag{1.21}$$

is called the Jacobi matrix. The estimation of the error due to the Jacobi method satisfies:

$$\|x - x^{(n)}\| \leq \frac{\|J\|_k}{1 - \|J\|_k} \|x^{(1)} - x^{(0)}\|_k,$$

where $\| \cdot \|_k$ denotes the matrix norm. Returning to the initial system, here is the convergence condition:

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |g_{ij}| < |g_{ii}| \quad \forall i = 1, \ldots, n.$$

Theorem 1.4.1. *A sufficient condition for convergence is for the matrix G of the system $Gx = c$ to be strongly diagonally dominant.*

Example 1.4.1. *Apply the Jacobi method to*

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20. \end{cases} \tag{1.22}$$

Let's first check for convergence. The matrix is diagonally dominant because:

$$g_{11} = 4 > 0.24 + |-0.08| = 0.32$$

$$g_{22} = 3 > 0.09 + |-0.15| = 0.24$$

$$g_{33} = 4 > 0.04 + |-0.08| = 0.12.$$

Solution To begin, rewrite the system

$$\begin{cases} x_1 = 2 - 0.06x_2 - 0.02x_3 \\ x_2 = 3 - 0.03x_1 + 0.05x_3 \\ x_3 = 5 - 0.01x_1 + 0.02x_2. \end{cases} \tag{1.23}$$

We can choose $x^{(0)} = (2, 3, 5)^T$. *The first approximation is*

$$\begin{cases} x_1^{(1)} &= 2 - 0.06 \times 3 - 0.02 \times 5 = 1.92 \\ x_2^{(1)} &= 3 - 0.03 \times 2 + 0.05 \times 5 = 3.19 \\ x_3^{(1)} &= 5 - 0.01 \times 2 + 0.02 \times 3 = 5.04. \end{cases}$$

Continue iteration, we obtain

$$x^{(2)} = (1.91, 3.19, 5.04)^T$$

$$x^{(3)} = (1.91, 3.20, 5.05)^T,$$

Algorithm 6: Jacobi

```
function jacobi(G,c)
input:G,c
n=length(c);
tol=0.001;
x0=[0;0;0];
nmax=2;
x=zeros(n,1);
for i=1:n do
    SommeI=sum(abs(G(i,1:i-1)))+sum(abs(G(i,i+1:n));
    if abs(G(i,i))>SommeI then
        disp('the matrix is diagonally dominant')
    else
        disp('the matrix is not diagonally dominant')
    end
end
for niter=1:nmax do
    for i=1:n do
        j=[1:i-1,i+1:n]; i=1:n ;
        x(i)=(c(i)-G(i,j)*x0(j))/G(i,i);
    end
    if norm(x-x0)<tol then
        return
    end
    x0=x;
end
disp('no convergence')
```

1.4.2 Gauss-Seidel Method

By defining $K = D - E$ and $M = F$, the system $(1, 1)$ takes the form:

$$(D - E)x^{(n+1)} = Fx^{(n)} + c, \tag{1.24}$$

which is a lower triangular system for the unknown $x^{(n+1)}$. Therefore, we can write:

$$g_{ii}x_i^{(n+1)} + \sum_{j=1}^{i-1} g_{ij}x_j^{(n+1)} = c_i - \sum_{j=i+1}^{N} g_{ij}x_j^{(n)} \text{ , for all i = 1, 2, ..., n.}$$

Consequently, equation (1.15) will be written component by component as follows:

$$x_i^{(n+1)} = \frac{1}{g_{ii}}\left(c_i - \sum_{j<i} g_{ij}x_j^{(n+1)} - \sum_{j>i} g_{ij}x_j^{(n)}\right), \quad 1 \leq i \leq n. \tag{1.25}$$

Note 6. *A priori, the Gauss − Seidel method should be more efficient than the Jacobi method because it takes into account, as it progresses, the already calculated values of* $x_i^{(n+1)}$.

Example 1.4.2. *Let's return to the first system given in equation (1.22), the matrix is diagonally dominant, check for convergence. Let's put the system in reduced form, as shown in (1.23).*
We can choose $x^{(0)} = (2, 3, 5)^T$. *The first approximation is:*

$$\begin{cases} x_1^{(1)} &= 2 - 0.06 \times 3 - 0.02 \times 5 = 1.72 \\ x_2^{(1)} &= 3 - 0.03 \times 1.72 + 0.05 \times 5 = 3.19 \\ x_3^{(1)} &= 5 - 0.01 \times 1.72 + 0.02 \times 3.19 = 5.04. \end{cases}$$

For the second iteration, we get:

$$\begin{cases} x_1^{(2)} &= 2 - 0.06 \times 3.19 - 0.02 \times 5.04 = 1.70 \\ x_2^{(2)} &= 3 - 0.03 \times 1.70 + 0.05 \times 5.04 = 3.20 \\ x_3^{(2)} &= 5 - 0.01 \times 1.70 + 0.02 \times 3.20 = 5.04. \end{cases}$$

Continue iteration, we obtain

$$x^{(3)} = (0.28, 3.24, 5.06)^T$$

$$x^{(4)} = (1.70, 3.20, 5.04)^T$$

$$x^{(5)} = (1.70, 3.20, 5.04)^T.$$

Algorithm 7: Gauss Seidel

```
1  Gauss seidel(G,c)
2  input:G,c
3  output:x
4  n=length(c);
5  tol=0.1;
6  x0=[0;0;0;0];
7  nmax=100;
8  x=zeros(n,1);
9  for i=1:n do
10     SommeI=sum(abs(G(i,1:i-1)))+sum(abs(G(i,i+1:n)));
11     if abs(G(i,i))>SommeI then
12         disp('the matrix is diagonally dominant') ;
13     else
14         disp('the matrix is not diagonally dominant') ;
15     end
16 end
17 for niter=1:nmax do
18     for i=1:n do
19         x(i)=(c(i)-G(i,i+1:n)*x0(i+1:n)-G(i,1:i-1)*x(1:i-1))/G(i,i);
20     end
21     if norm(x-x0)<tol then
22         return
23     end
24 end
25 x0=x;
26 disp('no convergence')
```

1.5 Numerical Comparison

In this section we will compare the various Direct and Iterative numerical methods that we have previously addressed as described:

Table 1.1: Comparison of CPU Times for Different methods for solving $Gx = c$

N	CPU_{Gauss}	CPU_{LU}	CPU_{Jacobi}	$CPU_{Gauss-Seidel}$
10	0,0093	0,0083	9.89×10^{-4}	8.12×10^{-5}
50	0,0141	0,0197	0.0019	8.72×10^{-4}
100	0,1180	0,0709	0.0055	8.59×10^{-4}
200	0,2753	0,2663	0.0157	0.0034
400	0.4171	0.5756	0.0289	0.0041
600	1.0116	1.2511	0.0579	0.0060
800	2.5920	2.6345	0.0949	0.0112
1000	6,1300	4,6818	0.1549	0.0151

We solved this linear program for various values of N using both Iterative and Direct Methods in MATLAB 2021b on a PC equipped with a Core i5 processor running at 2.50 GHz and 16 GB of RAM. The CPU execution times for the Gauss elimination algorithm, LU factorization, as well as Jacobi and Gauss-Seidel methods, depending on the size of the matrix N, as shown in Table (1.1).

The table results are plotted in Figure (3.4).

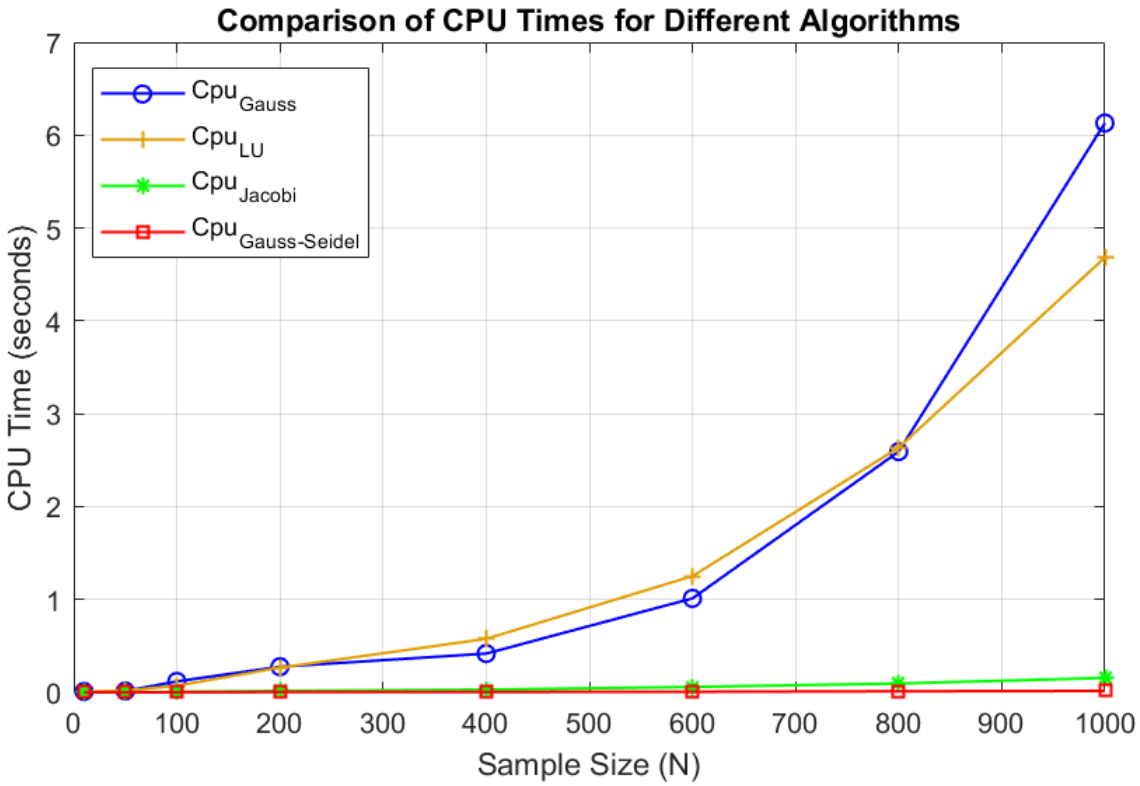


Figure 1.2: Comparison of CPU times for different methods for solving $Gx = c$.

From of the figure (3.4), we deduce When $N \leq 100$, it is preferable to use direct

methods because the time difference is very close, while the results of direct methods are accurate unlike indirect methods. When $N > 100$, indirect methods are faster than direct methods. We note that :

- In the case of $N > 800$, the LU decomposition method is better than the Gauss method.
- The Gauss-Seidel method converges more closely than Jacobi.

Chapter

2 Lyapounouv Sylvester equation

2.1 Introduction

The Lyapunov-Sylvester equation is fundamental in analyzing and stabilizing dynamic systems, widely used in PDE and control theory [9]. This chapter aims to explore this equation and its solution using the Kronecker product, which simplifies operations in high-dimensional spaces. We will define the Kronecker product, study the Lyapunov-Sylvester equation, and transform it into a standard linear system using the Kronecker product, with examples and analysis of computational complexity [8] and [7]. Finally, we will discuss the conditions for the existence and uniqueness of solutions, and their importance in engineering and scientific applications.

2.2 Kronecker product

Definition 2.2.1. *Let $A = (a_{i,j})_{\substack{1\leq i\leq n\\ 1\leq j\leq m}}$ and $B = (b_{i,j})_{\substack{1\leq i\leq p\\ 1\leq j\leq q}}$ be two matrices of sizes $m \times n$ and $p \times q$ respectively. The Kronecker product, or tensor product, of A and B, denoted $A \otimes B$, is the matrix of size $mp \times nq$ given by:*

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}. \tag{2.1}$$

Example 2.2.1. *Consider the two matrices A and B given by*

$$A = \begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 3 \end{pmatrix}, \; B = \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix}.$$

Their Kronecker product is explicitly given as follows:

$$A \otimes B = \begin{pmatrix} 2. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} & 1. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} \\ 0. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} & 2. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} \\ 1. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} & 3. \begin{pmatrix} 2 & 5 \\ 0 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 4 & 10 & 2 & 5 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 4 & 10 \\ 0 & 0 & 0 & 2 \\ 2 & 5 & 6 & 15 \\ 0 & 1 & 0 & 3 \end{pmatrix}.$$

Properties 2.2.1. *Let A, B, C be three matrices of the same size, we have the following properties:*

- *Distributivity: The Kronecker product is distributive with respect to addition.*

$$(\alpha A + \beta B) \otimes C = \alpha(A \otimes C) + \beta(B \otimes C),$$

$$A \otimes (\alpha B + \beta C) = \alpha(A \otimes B) + \beta(A \otimes C).$$

- *Associativity:* $(A \otimes B) \otimes C = A \otimes (B \otimes C)$.
- *Rank: The rank of the Kronecker product is the product of the ranks of the involved matrices. In other words, if A is a matrix of rank r_1 and B is a matrix of rank r_2, then the Kronecker product $A \otimes B$ is a matrix of rank $r_1 \times r_2$.*
- *Transposition:* $(A \otimes B)^T = A^T \otimes B^T$.

Definition 2.2.2. *Let C be a matrix of size $m \times n$, denoted by $c_1, \ldots c_n$ representing the columns of A The vectorization of C denoted $vec(C)$, is the column vector of size $mn \times 1$ This is expressed as follows:*

$$vec(C) = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}. \tag{2.2}$$

2.3 The Sylvester equation

Let $\mathbb{K}$ be a commutative field. The Lyapunov-Sylvester equation is defined as follows :

$$\sum_{i=1}^{n} A_i X B_i = C,$$

where A and B are two square matrices of size $m \times m$ and $n \times n$ respectively, and $X, C \in \mathcal{M}_{m,n}(\mathbb{K})$.

2.3.0.1 Special Cases

Let $A \in \mathcal{M}_m(\mathbb{K})$ and $B \in \mathcal{M}_n(\mathbb{K})$ be two square matrices of size $m \times m$ and $n \times n$ respectively, and $X, C \in \mathcal{M}_{m,n}(\mathbb{K})$.

- The Sylvester equation is the matrix equation of the form

$$AX + XB = C. \tag{2.3}$$

- The Lyapunov equation is also defined by (if $B = A^T$)

$$AX + XA^T = C. \tag{2.4}$$

- The non linear matrix equation of the form

$$AX - XB^T = C + XDX. \tag{2.5}$$

  Is called the Riccati equation. The problem is to determine matrix X that satisfy the above equations

2.3.1 Transformation by using the Kronecker Product

In this paragraph, we explain how to transform equation (2.3) through the Kronecker product into a standard equation of the form $Gx = c$.

We begin by illustrating an example in $\mathcal{M}_2(\mathbb{K})$: Consider the matrices A,B,C and X given by

$$A = \begin{pmatrix} a_1 & a_3 \\ a_2 & a_4 \end{pmatrix}, B = \begin{pmatrix} b_1 & b_3 \\ b_2 & b_4 \end{pmatrix}, C = \begin{pmatrix} c_1 & c_3 \\ c_2 & c_4 \end{pmatrix} \text{ and } X = \begin{pmatrix} x_1 & x_3 \\ x_2 & x_4 \end{pmatrix}.$$

Solving equation (2.3), which results in after substitution:

$$\begin{pmatrix} a_1x_1 + a_3x_2 - b_1x_1 - b_2x_3 & a_1x_3 + a_3x_4 - b_3x_1 - b_4x_3 \\ a_2x_1 + a_4x_2 - b_1x_2 - b_2x_4 & a_2x_3 + a_4x_4 - b_3x_2 - b_4x_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_3 \\ c_2 & c_4 \end{pmatrix}.$$

By identification, we obtain the linear system:

$$\begin{cases} (a_1 - b_1)x_1 + a_3x_2 - b_2x_3 = c_1 \\ a_2x_1 + (a_4 - b_1)x_2 - b_2x_4 = c_2 \\ b_3x_1 + (a_1 - b_4)x_3 + a_3x_4 = c_3 \\ b_3x_2 + a_2x_3 + (a_4 - b_4)x_4 = c_4, \end{cases}$$

what is given in matrix form is:

$$\underbrace{\begin{pmatrix} a_1 - b_1 & a_3 & -b_2 & 0 \\ a_2 & a_4 - b_1 & 0 & -b_2 \\ -b_3 & 0 & a_1 - b_4 & a_3 \\ 0 & b_3 & a_2 & a_4 - b_4 \end{pmatrix}}_{G} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}}_{c},$$

we observe that the matrix of this system can be decomposed as follows:

$$\begin{pmatrix} \begin{pmatrix} a_1 & a_3 \\ a_2 & a_4 \end{pmatrix} - b_1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & -b_2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ -b_3 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} a_1 & a_3 \\ a_2 & a_4 \end{pmatrix} - b_4 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix}.$$

This matrix, denoted by G, is expressed using the Kronecker product in the form:

$$G = (I_n \otimes A - B^T \otimes I_m). \tag{2.6}$$

Now, we provide a generalization of this transformation. Let $A = (a_{i,j})_{\substack{1\leq i\leq m \\ 1\leq j\leq m}}$ and $B = (b_{i,j})_{\substack{1\leq i\leq n \\ 1\leq j\leq n}}$. If we denote $vec(X)$ as the column vector obtained from matrix X by juxtaposing the column vectors of matrix X, then the equation $AX + XB = C$ is transformed into a standard linear system:

$$G\text{vec}(X) = \text{vec}(C). \tag{2.7}$$

In this case, if the matrix G is invertible, then the equation has a unique solution.

2.3.2 Algorithmic Complexity

We are interested in the complexity of computing the solution to equation (2.3) using the Kronecker product.

- The two operations $I_n \otimes A$ and $B^T \otimes I_m$ require $O(n^2m^2)$ operations.
- Their addition does not affect the complexity, and we have $O(n^2m^2)$ operations.
- Inverting the matrix G requires $O((n^2m^2)^3)$ operations.
- Multiplying G^{-1} by the column vector $vec(C)$ requires $O(n^2m^2)$ operations.
- Reorganizing the elements to obtain matrix X requires $O(n^2m^2)$ operations.

Thus, the total complexity of solving the Sylvester equation using the Kronecker product is generally dominated by $O((n^2m^2)^3)$. For the case of equation (2.4), the system G reduces to a matrix product $G = (I_n \otimes A + A^T \otimes I_n)$.

2.4 Existence and Uniqueness of the Solution

Let E be a vector space over a field $\mathbb{K}$, and $f : E \to E$ an endomorphism. The spectrum of f, denoted by $spec(f)$, is the set of eigenvalues of f, or equivalently, the set of $\lambda \in \mathbb{K}$ such that $f - \lambda \cdot \text{id}_E$ is not invertible. Let M be the square matrix of order n associated with f. The characteristic polynomial of M is denoted by $P(X) = \det(MX - \lambda I_n)$.

Theorem 2.4.1 (Cayley-Hamilton). *If M is a square matrix of order n and $\rho(X)$ is its characteristic polynomial, then $P(M) = p_0 I + p_1 M + \ldots + p_{n-1} M^{n-1} + p_n M^n = 0$, where p_i are the coefficients of $P(X)$.*

Theorem 2.4.2 (Spectral Theorem). *Let M be a matrix over a vector space $\mathbb{E}$ of dimension n over $\mathbb{K}$. Then, the spectrum of the polynomial operator of this matrix, denoted by $q(M)$, is the image of the spectrum of M under q as follows: $spec(q(M)) = q(spec(M))$.*

Theorem 2.4.3. *Let $A \in \mathcal{M}_m(\mathbb{C})$ and $B \in \mathcal{M}_n(\mathbb{C})$, the equation (2.3) has a unique solution $X \in \mathcal{M}_{n,m}(\mathbb{C})$ for any solution $C \in \mathcal{M}_{n,m}(\mathbb{C})$ if and only if A and B do not share any eigenvalues. This can be expressed as:*

$$spec(A) \cap spec(B) = \varnothing.$$

2.5 Specific cases for lower and upper matrices

In this section, we aim to find actual solutions to the equation 2.3. First, we will develop a method to solve the specific case:

$$LX + XU = C, \tag{2.8}$$

subsequently, we will integrate this method with the techniques discussed in the previous chapter to achieve the desired solutions.

To find an algorithm that allows solving the equation (2.8), for $L \in \mathcal{M}_{n,n}$ in the form (1.6) and $U \in \mathcal{M}_{m,m}$ in the form (1.5), we will start with matrices of rank three and then generalize the method for all possible values of n.

$$\underbrace{\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}}_{LX} + \underbrace{\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}}_{XU} = \underbrace{\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}}_{C},$$

solving equation (2.8), which results in after substitution:

$$for j = 1$$

$$\begin{cases} (l_{11}+u_{11})x_{11}=c_{11} \\ l_{21}x_{11}+(l_{22}+u_{11})x_{21}=c_{21} \\ l_{31}x_{11}+l_{32}x_{21}+(l_{33}+u_{11})x_{31}=c_{31} \end{cases} \Leftrightarrow \begin{cases} x_{11}=c_{11}/(l_{11}+u_{11}) \\ x_{21}=(c_{21}-l_{21}x_{11})/(l_{22}+u_{11}) \\ x_{31}=(c_{31}-l_{31}x_{11}-l_{32}x_{21})/(l_{33}+u_{11}), \end{cases}$$

$$for j=2$$

$$\begin{cases} u_{12}x_{11}+(l_{11}+u_{22})x_{12}=c_{12} \\ l_{21}x_{12}+u_{12}x_{21}+(l_{22}+u_{22})x_{22}=c_{22} \\ l_{31}x_{12}+l_{32}x_{22}+u_{12}x_{31}+(l_{33}+u_{22})x_{32}=c_{32} \end{cases} \Leftrightarrow \begin{cases} x_{12}=(c_{12}-u_{12}x_{11})/(l_{11}+u_{22}) \\ x_{22}=(c_{22}-l_{21}x_{12}-u_{12}x_{21})/(l_{22}+u_{22}) \\ x_{32}=(c_{32}-s_1)/(l_{33}+u_{22}), \end{cases}$$

$$for j=3$$

$$\begin{cases} u_{13}x_{11}+u_{23}x_{12}+(l_{11}+u_{33})x_{13}=c_{13} \\ l_{21}x_{13}+u_{13}x_{21}+u_{23}x_{22}+(l_{22}+u_{33})x_{23}=c_{23} \\ l_{31}x_{13}+l_{32}x_{23}+u_{13}x_{21}+u_{23}x_{32}+(l_{33}+u_{33})x_{33}=c_{33} \end{cases} \Leftrightarrow \begin{cases} x_{13}=(c_{13}-s_2)/(l_{11}+u_{33}) \\ x_{23}=(c_{23}-s_3)/(l_{22}+u_{33}) \\ x_{33}=(c_{33}-s_4/(l_{33}+u_{33}), \end{cases}$$

where $s_1=l_{31}x_{12}+l_{32}x_{22}+u_{12}x_{31}$, $s_2=u_{13}x_{11}+u_{23}x_{12}$, $s_3=l_{21}x_{13}+u_{13}x_{21}+u_{23}x_{22}$ and $s_4=l_{31}x_{13}+l_{32}x_{23}+u_{13}x_{21}+u_{23}x_{32}$.

Note 7. *When using expression (2.7) where G (2.6) to solve equation (2.8) , we deduce that regardless of n, G is a lower triangular matrix. This is a special case that is easy to solve as shown in algorithm (2).*

The different algorithms for solving the problem (2.8) are given:

- Sylvester for LU systems (S-LUS).
- Sylvester–Kronecker Product for LU systems (S-KPLUS).
- Lyap method of MATLAB (LMM).

The first method relies on expanding the equation (2.8) and finding the relationship between its indices $1 \leq i, j \leq n$.

Algorithm 8: S-LUS

```
function lyap
input:L,U,C;
output:X;
n=length(L);
X=zeros(n,n);
for j=1:n do
    for i=1:n do
        X(i,j)=(C(i,j)-L(i,1:i-1)*X(1:i-1,j)-X(1:j-1,j)'*U(i,1:j-1))/(L(i,i)+U(j,j));
    end
end
disp(X)
```

The second method relies on transforming the equation (2.8) into $Gx = c$ using the Kronecker product, and then solving it using one of the methods mentioned in the first chapter.

Algorithm 9: S-KPLUS

```
function Gx=c
input:L,U,C;
output:X;
n=length(L);
X=zeros(n,n);
I=eye(n,n);
G=kron(I,L)+kron(U',I);
c=reshape(C,[ ],1);
x=inv(G)*c;
Z=reshape(x,[ ],n);
```

Finally, we used the Lyapunov results presented in the requirement.

Algorithm 10: LMM

```
function lyap
input:L,U,C;
output:X;
n=length(L);
X=zeros(n,n);
X=lyap(L,U,-C);
```

2.6 Numerical Comparison

Table 2.1: Comparison of CPU Times for Different methods for solving $LX + XU = C$.

N	CPU_{S-LUS}	$CPU_{S-KPLUS}$	CPU_{LMM}
5	0.007	0.009	0.031
10	0.005	0.008	0.031
15	0.014	0.038	0.053
20	0.024	0.015	0.035
40	0.019	0.306	0.038
60	0.033	2.340	0.027
80	0.042	11.605	0.034
100	0.121	42.127	0.054
120	0.114	/	0.034
140	0.148	/	0.035
160	0.222	/	0.035
180	0.250	/	0.036
200	0.299	/	0.037
800	7.683	/	0.269
1000	14.074	/	Does not converge
1500	48.919	/	Does not converge
2000	119.512	/	Does not converge

We solved this linear program for various values of N using different methods in MATLAB 2021b on a PC equipped with a Core i5 processor running at 2.50 GHz and 16 GB of RAM. The CPU execution times for differents methods S-LUS, S-KPLUS and LMM, depending on the size of the matrix N, as shown in Table (2.1).

The table results are plotted in Figure (2.1).

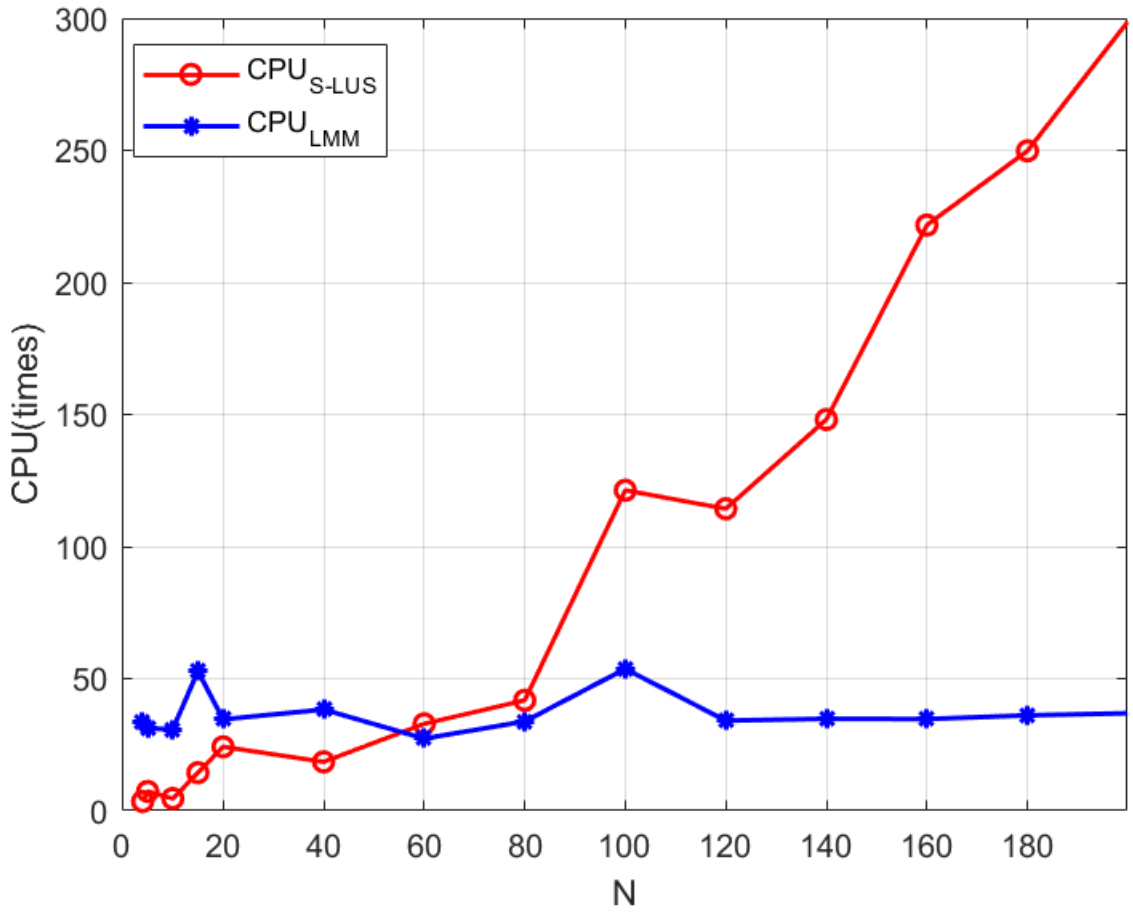


Figure 2.1: Comparison of CPU Times for Different methods for solving $LX + XU = C$.

From the graph of figure (2.1) and table (2.1), we deduce that:

- Method S-KPLUS is significantly slower compared to methods S-LUS and LMM.
- Method LMM is faster than S-LUS in the range $60 \leq N \leq 100$, but it becomes ineffective when $N > 100$.
- Method S-LUS is fast and effective for all natural numbers N.

Chapter

3 Numerical solution of PDE using finite-difference method combined with Lyapunov equation

3.1 Statment of the problem

We consider the following linear PDE problem

$$\begin{cases} \frac{\partial u}{\partial t}(t,x) = \frac{\partial^2 u}{\partial x^2}(t,x) + f(u), \quad 0 \leq t \leq T, \quad a \leq x \leq b \\ u(0,x) = \varphi(x) \\ u(t,a) = u(t,b) = 0. \end{cases} \tag{3.1}$$

3.2 Discretization

For computational reasons, we restrict our attention to a bounded interval $I = [a,b] \in \mathbb{R}$ and, obviously, $a < b$. We fix uniform partition of I of the form

$$a = x_0 < x_1 < \ldots < x_N = b, \tag{3.2}$$

where N are positive integers, and each of the partitions has a norm given by $\triangle x = (b-a)/N$. So we fix again a uniform partition of the interval $[0,T]$ of the form

$$0 = t_0 < t_1 < \ldots < t_K = T, \tag{3.3}$$

with these conventions, we will approximate numerically the solutions of (3.1) in $I \times [0, T]$. We will represent the computational approximation of the exact value of $u(t_k, x_m)$ by $u_{k,n}$. Clearly, appropriate initial and boundary conditions must be imposed upon the problem in order to compute new approximations of the solution. In our case, we will consider discrete initial profiles of the form

$$u_{0,n} = \phi(x_n),$$

where $\phi : I \to \mathbb{R}$ is a suitable function which may be, for instance, the expression of an analytical solution at the time $t = 0$. Additionally, we consider discrete Dirichlet boundary conditions of the form

$$u_{k,0} = \psi_a(t_k),$$

$$u_{k,N} = \psi_b(t_k),$$

for practical purposes, we may think of as given by an exact solution of the problem under investigation at the boundary. The various derivative operators of u with respect to time and space in the continuous problem will be replaced within the interior of the grid by approximations, resulting in second-order accuracy. The time derivative will be substituted by

$$\frac{\partial u}{\partial t}(t, x) \simeq \frac{u_{k+1,j} - u_{k,j}}{l}.$$

We approximate the Laplacian operator we obtain

$$\frac{\partial^2 u}{\partial x^2}(t, x) \simeq \frac{u_{k,j+1} + u_{k,j-1} - 2u_{k,j}}{h^2}.$$

Let $\alpha = \frac{l}{h^2}$, we obtain from utilizing these discrete approximations

$$u_{k+1,j} = \alpha u_{k,j+1} + (1 + l - 2\alpha)u_{k,j} + \alpha u_{k,j-1}, \forall j = 1, 2, ..., N. \tag{3.4}$$

3.3 Matrix representation

Throughout, we will follow the nomenclature introduced in the previous section, and let P denote the positive, integer number $(N + 1)$. For the sake of convenience, we will present our numerical method in vector form.

3.3.1 Model 1

For every $k = 1, 2, ..., K$, let u_k be the P-dimensional, real vector of the approximate solution at the time t_k, i.e. let

$$u_k = \Big(u_{k,0}, u_{k,1}, \ldots, u_{k,N}\Big).$$

On the other hand, for every $k = 1, 2, ..., K-1$, let b^{k+1} be the P-dimensional, real vector containing the information of the solutions on the boundary at the time t_{k+1},

for all $j = 1, 2, ..., N$ we obtain the following system:

$$\begin{cases} u_{k+1,1} = & \alpha u_{k,0} + (1-2\alpha)u_{k,1} + \alpha u_{k,2} \\ u_{k+1,2} = & \alpha u_{k,1} + (1-2\alpha)u_{k,2} + \alpha u_{k,3} \\ u_{k+1,3} = & \alpha u_{k,2} + (1-2\alpha)u_{k,3} + \alpha u_{k,4} \\ \vdots & \vdots \\ u_{k+1,N} = & \alpha u_{k,N-1} + (1-2\alpha)u_{k,N} + \alpha u_{k,N+1}, \end{cases} \tag{3.5}$$

this system is written in the following matrix form :

$$\underbrace{\begin{pmatrix} u_{k+1,1} \\ u_{k+1,2} \\ \vdots \\ u_{k+1,N} \end{pmatrix}}_{u^{k+1}} = \underbrace{\begin{pmatrix} (1-2\alpha) & \alpha & \vdots & 0 \\ \alpha & (1-2\alpha) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \alpha \\ 0 & \ldots & \alpha & (1-2\alpha) \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u_{k,1} \\ u_{k,2} \\ \vdots \\ u_{k,N} \end{pmatrix}}_{u^k} + \underbrace{\begin{pmatrix} \alpha u_{k,0} \\ 0 \\ \vdots \\ 0 \\ \alpha u_{k,N+1} \end{pmatrix}}_{b}, \tag{3.6}$$

$$\begin{cases} u_{k+1} = Au_k + b, \ k = 0, ..., K-1 \\ u_0 = \text{initial condition.} \end{cases} \tag{3.7}$$

3.3.2 Model 2

Let I be the identity matrix of size $(N+1)\times(N+1)$, let I_0 be the diagonal matrix of the same size that I given by

$$I_0 = \begin{pmatrix} 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & 1 & \ddots & \dots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \dots & \dots & \ddots & 0 & 1 & 0 \\ 0 & \dots & \dots & \dots & 0 & 0 & 0 \end{pmatrix},$$

and define the block matrix B of size $P\times P$ by

$$B = \begin{pmatrix} I & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & I & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & I & \ddots & \dots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \dots & \dots & \ddots & 0 & I & 0 \\ 0 & \dots & \dots & \dots & 0 & 0 & I \end{pmatrix}. \tag{3.8}$$

In this last identity, all the zeros represent null matrices of sizes $(N+1)\times(N+1)$.

Let

$$T = \begin{pmatrix} (1-2\alpha) & -\alpha & \vdots & 0 \\ -\alpha & -(1-2\alpha) & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\alpha \\ 0 & \dots & -\alpha & -(1-2\alpha) \end{pmatrix}.$$

For every $m = 1, 2, ..., M-1$, $n = 1, 2, ..., N-1$ and $k = 1, 2, ..., K-1$, let

$$u_{k+1,N} - (1-2\alpha)u_{k,N} - \alpha u_{k,N+1} - \alpha u_{k,N-1} = 0, \tag{3.9}$$

let A_m^{k+1} be the square matrix of size $(N+1)\times(N+1)$ defined by

$$u_k = \Big(u_{1,1}, u_{1,2}, \ldots, u_{1,N}, u_{2,1}, u_{2,2}, \ldots, u_{2,N}, \ldots, u_{i,1}, u_{i,2}, \ldots, u_{i,N}, \ldots, u_{k,1}, u_{k,2}, \ldots, u_{k,N}\Big).$$

$$\underbrace{\begin{pmatrix} I & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\ T & I & 0 & \ldots & \ldots & \ldots & 0 \\ 0 & T & I & \ddots & \ldots & \ldots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ldots & \ldots & \ddots & T & I & 0 \\ 0 & \ldots & \ldots & \ldots & 0 & T & I \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_i \\ \vdots \\ u_k \end{pmatrix}}_{u} = \underbrace{\begin{pmatrix} \alpha u_{k,0} \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ \alpha u_{k,N+1} \end{pmatrix}}_{b}, \tag{3.10}$$

we will assess the convergence rates of the finite difference discretization by comparing the numerically computed solution to the exact solution. Our goal is to numerically solve the following problem.

Example 3.3.1. *Using several examples, we aim to offer numerical implementations to validate the theoretical findings previously discussed*

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} = \dfrac{\partial^2 u(t,x)}{\partial x^2} + f(u(t,x)), (t,x) \in [0,T]\times I, \\ u(0,x) = \sin(\pi x),\ x \in I, \\ u(t,a) = u(t,b) = 0,\ t \in [0,T], \end{cases}$$

where $f(u(t,x)) = u(t,x)$.

The exact solution of the problem (1)

$$u_{ex}(t,x) = \exp(-(\pi^2-1)t)\sin(\pi x).$$

We developed an implementation with MATLAB that solve the problem 1, with the function Lyap. This function finds the solution of the system $AX + XB = C$.

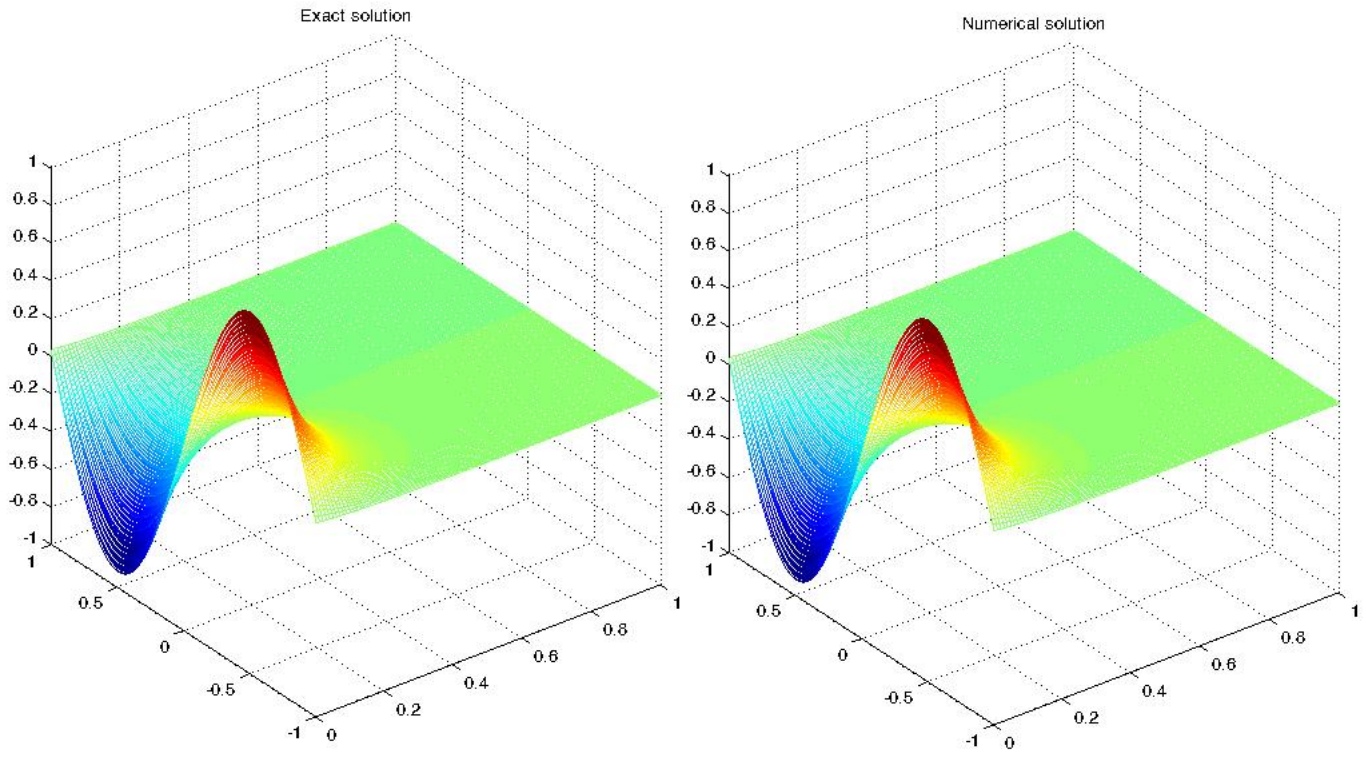


Figure 3.1: Comparison between analytical solution and numerical solution for $T = 1$ and $I = [-1, 1]$.

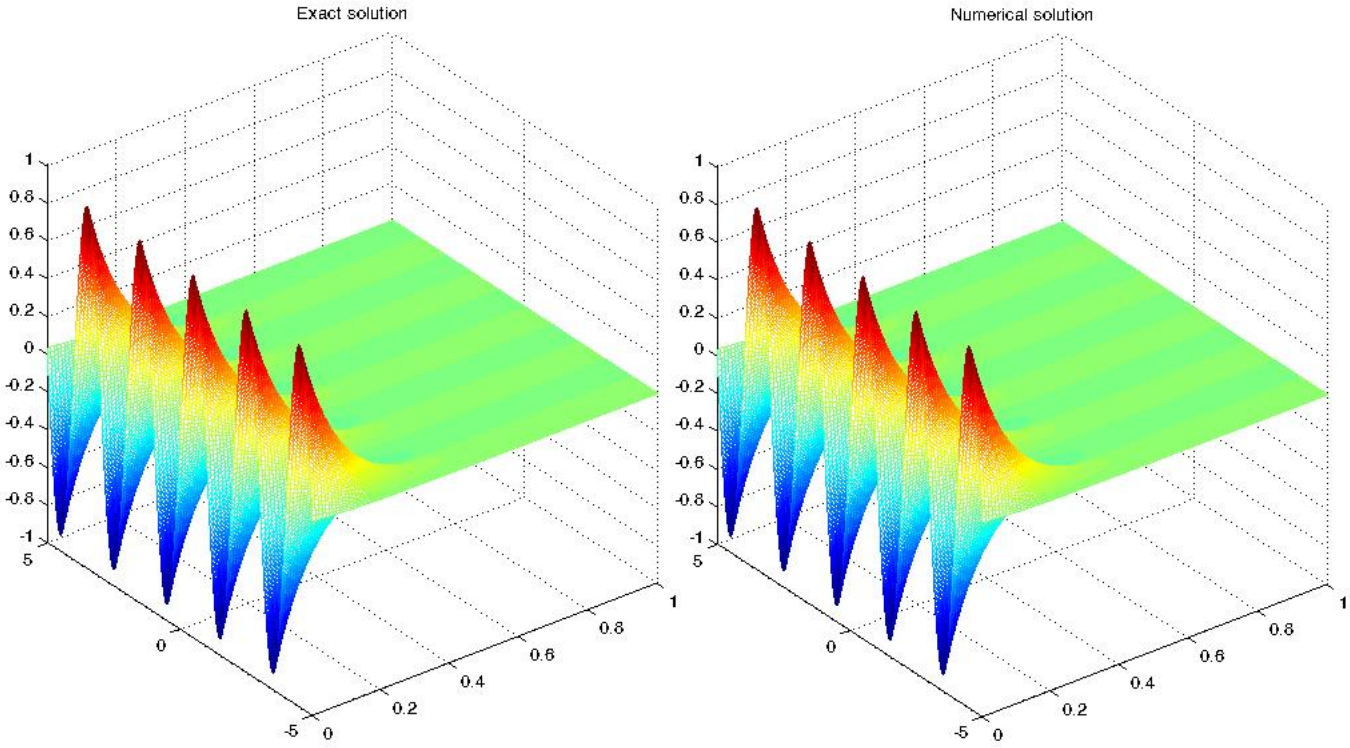


Figure 3.2: Comparison between analytical solution and numerical solution for $T = 1$ and $I = [-5, 5]$.

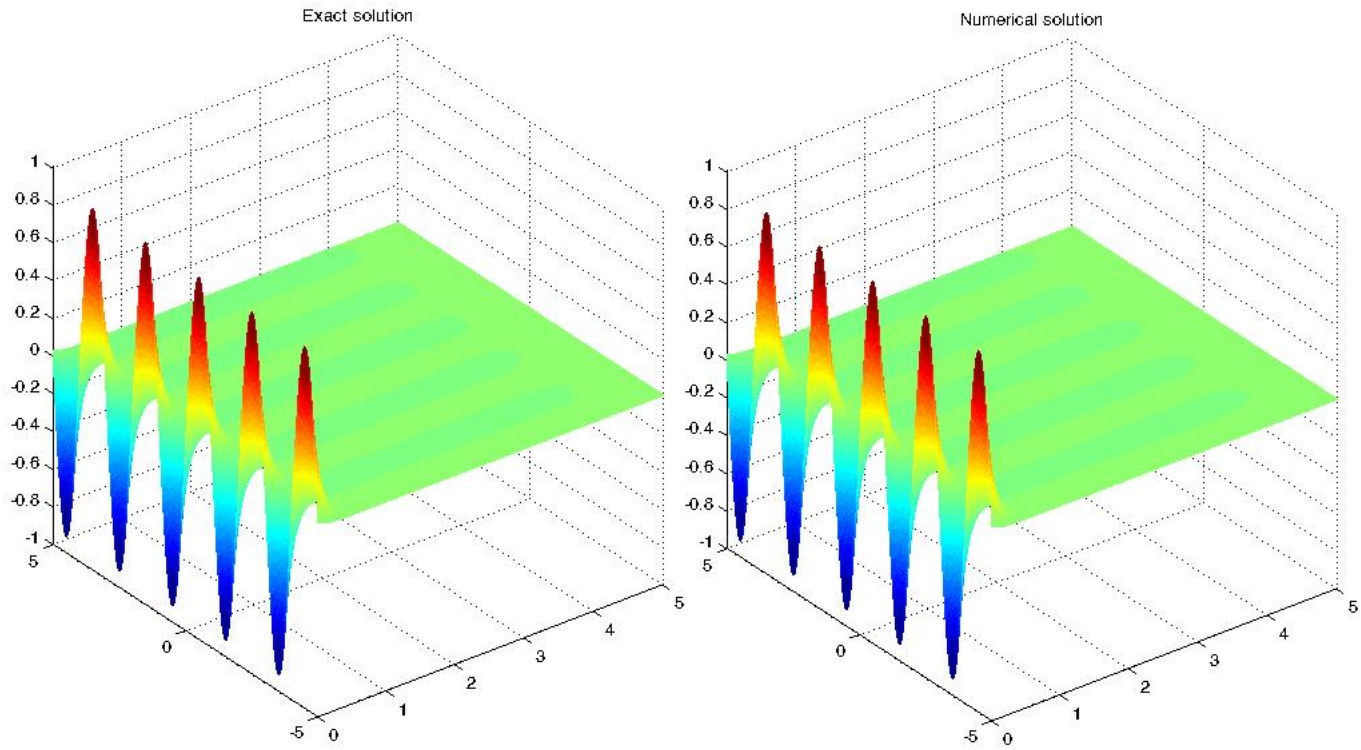


Figure 3.3: Comparison between analytical solution and numerical solution for $T = 2$ and $I = [-5, 5]$.

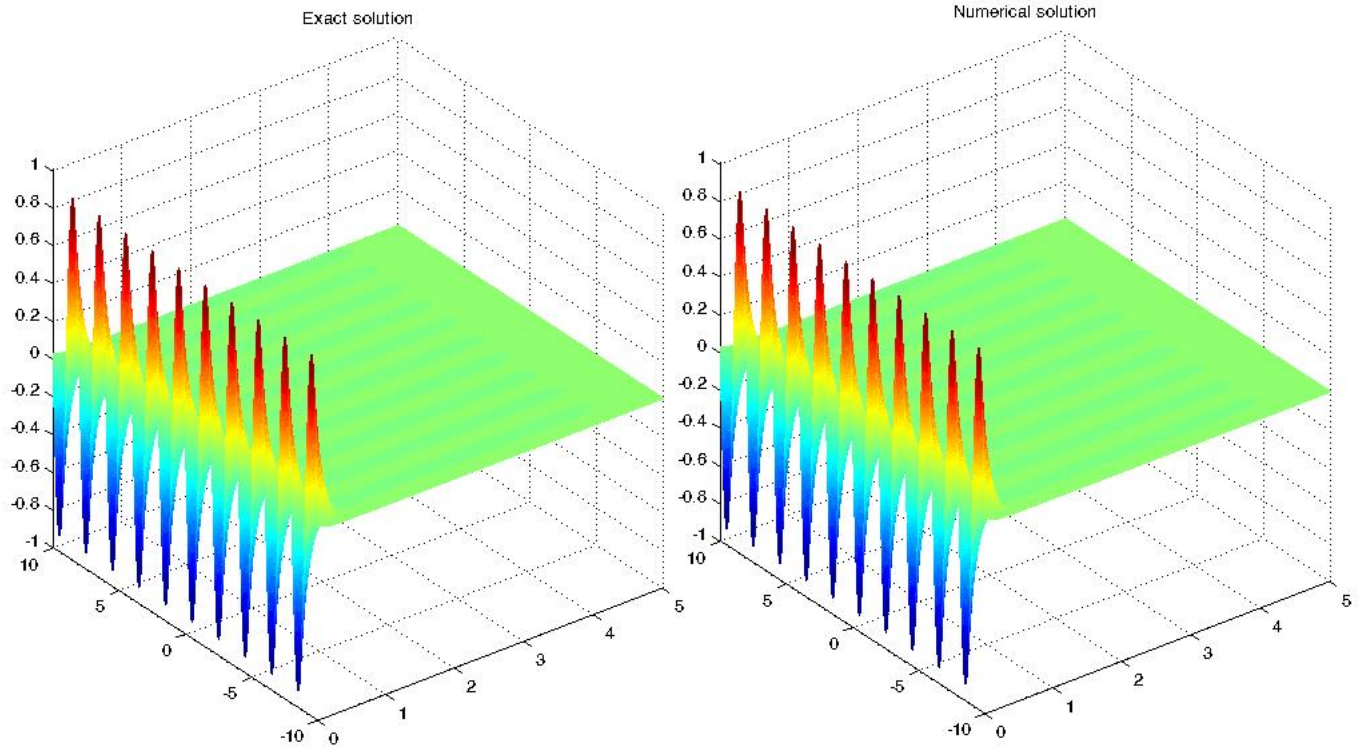


Figure 3.4: Comparison between analytical solution and numerical solution for $T = 5$ and $I = [-10, 10]$.

We fix $h = 0.01$ and $l = 0.01$, we fix $T = 1, 5, 10$ and $I = [-1, 1], [-5, 5]$ and

$[-10, 10]$. Moreover, when comparing numerical simulations to the exact solution, we observe that the Lyapunov method developed in our work has proved more efficient in termes of convergence and precision.

Conclusion

This work focuses on the application of the finite difference method, combined with the Lyapunov method, to solve linear partial differential equations (PDEs) with homogeneous Dirichlet boundary conditions. PDEs are fundamental in the modeling of many physical and engineering phenomena, but their analytical resolution is often complex, if not impossible. Thus, numerical methods become essential.

We start with an introduction to the basic concepts of PDEs and homogeneous Dirichlet boundary conditions. Next, we detail the finite difference method, a technique that discretizes the continuous domain into a lattice of points and approximates the partial derivatives by finite differences. This method is known for its simplicity of implementation and its effectiveness for problems defined on regular domains.

In order to improve the stability and convergence of the finite difference method, we integrate the Lyapunov method. The Lyapunov method, formulated by the matrix equation AX+XB=C, is used to analyze the stability of solutions and optimize numerical schemes. This approach allows the discrete PDE problem to be reformulated into a Lyapunov equation, thereby facilitating numerical solution while ensuring desirable properties such as stability and precision.

We then develop the practical implementation of this combined approach, detailing the algorithms and steps required to solve linear PDEs. Case studies and numerical simulations are presented to illustrate the effectiveness of our method. These examples show how the finite difference method, enhanced by Lyapunov analysis, can efficiently solve typical problems while respecting the constraints of homogeneous Dirichlet boundary conditions.

Finally, we discuss the advantages and limitations of this combined approach. The integration of the finite difference method with the Lyapunov method offers a robust framework for solving linear PDEs, providing improved stability and accuracy. Future perspectives include the adaptation of these techniques to nonlinear problems and complex

geometries, which could further extend the applicability and effectiveness of the methods presented in this dissertation.

Bibliography

[1] **M. GILLI**, Methodes numeriques, Département d'économétrie Université de Genéve, 25 mars 2006.

[2] **R. Herbin**, Analyse numérique Systémes Linéaires, Pour L3, Université d'Aix-Marseille, 8 septembre 2016.

[3] **F. ZEMANI**, Polycopié Programmation et méthode numérique sous MATLAB, Université des Sciences et de la Technologie d'Oran.

[4] **R. TLEMSANI**, Méthodes Numériques, Licence L2, Université des Sciences et de la Boudiafe USTOMB.

[5] **C. KELLY**, Iterative methods for linear and nonlinear equations, SIAM, Philadelphia, 1995.

[6] **G. MARCHUK**, Methods of numerical mathematics, Springer-Verlag, New York, Heidelberg, and Berlin, 1982.

[7] **A. BOURAI & R. HALES**, La résolution des équations de Lyapunov-Sylvester sur un corps quelconque, Mémoire de Master, Université des Sciences et de la Technologie Houari Boumediene, 13/07 2023.

[8] **A. Bezia & M. Aliane** , Solving Nonlinear Reaction-Diffusion Equations: A Comparative Study of Numerical Methods, Bezia A. (NHSM), 26 Novembre 2023.

[9] **P. Kirrinnis**, Fast algorithms for the Sylvester equation AX- XBT= C, Theoretical Computer Science 259.1-2 (2001): 623-638.

[10] **F. Gerrish, & A. Ward**, Sylvester's matrix equation and Roth's removal rule, The Mathematical Gazette 82.495 (1998): 423-430.

[11] **H. Bloch & T. Lenne**, Analyse numérique et simulation de systèmes de réaction-diffusion, Mémoire de Master , université de lille, 28 mai 2017.

[12] **J. Macias-Diaz**, A bounded finite-difference discretization of a two-dimensional diffusion equation with with logistic nonlinear reaction, Universidad Autonoma de Aguascalientes Mexico, 1 August 2011.

[13] **S. Déscombes**, Magali Ribot, Convergence of the Peaceman-Rachford approximation for reaction-diffussion systems, Numer. Math, 2003.

[14] **D. Furihata & S. Koide**, Nonlinear and Linear Conservative Finite Difference Schemes for Regularized Long Wave Equation,Japan J. Indust. Appl. Math., 26 (2009).

[15] **D. Furihata & M. Mori**, A stable finite difference scheme for the Cahn–Hilliard equation based on the Lyapunov functional, ZAMM Z. angew. Math. Mech., 76 (1996), 405–406.

[16] **S. Kim**, Numerical Methods for Partial Differential Equations, Department of Mathematics and Statistics Mississippi State University, MS 39762 USA, December 11, 2023.