Hassiba Benbouali University of Chlef
(UHBC)
Faculty of Exact Sciences And Informatics
Department of Mathematics

Numerical Analysis II

Intended for students of
Second Year LMD Mathematics Degree (L2)

Mr. MEZOUAGHI Abdelheq
Email: **a.mezouaghi@univ-chlef.dz**

May 7, 2025

General introduction

Numerical analysis forms the cornerstone of modern scientific computing, enabling the solution of complex mathematical problems where analytical methods fail. This discipline is crucial for physics simulations, engineering design, financial modeling, and artificial intelligence applications. Our course material equips second-year LMD mathematics students with fundamental computational techniques, beginning with linear system solutions (Gaussian elimination, iterative methods) in Chapter 1. Chapter 2 covers eigenvalue problems, essential for stability analysis and machine learning algorithms. First-order differential equations (Chapter 3) model dynamic systems in biology and economics, solved numerically via Euler and Runge-Kutta methods. The final chapter tackles nonlinear algebraic systems using Newton-Raphson iterations, vital for optimization and control theory.

Through algorithmic rigor and MATLAB examples, we emphasize error analysis and computational efficiency. Progressive exercises develop both theoretical understanding and practical implementation skills, preparing students for research and industry challenges in applied mathematics.

Contents

Chapter 1

Solving linear systems

Introduction

Linear systems of equations arise in numerous scientific and engineering applications, from circuit analysis to computational fluid dynamics. Solving these systems efficiently is crucial for accurate and practical results. Methods for solving linear systems can be broadly classified into two categories:

Direct Methods

These provide exact solutions (up to machine precision) in a finite number of steps. Examples include:

- Gaussian elimination
- LU decomposition ($A = LU$, where L is lower triangular and U is upper triangular)
- Cholesky factorization (for symmetric positive-definite matrices, $A = LL^T$)

Numerical (Iterative) Methods

These approximate the solution through successive iterations, making them preferable for large, sparse systems where direct methods are computationally expensive. Common techniques include:

- Jacobi method
- Gauss-Seidel method
- Relaxation method

The choice between direct and iterative methods depends on factors such as matrix size ($n \times n$), sparsity, and conditioning (e.g., condition number $\kappa(A)$). This discussion explores key algorithms from both approaches, their advantages, and their limitations.

1.1 Review of Linear Algebra Concepts

[Vector Space] A vector space over a field $\mathbb{K}$ (where $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$) is a non-empty set V equipped with:

- an internal operation + called addition,
- an external operation : called scalar multiplication,

satisfying the following properties:

1. $(V, +)$ is a commutative group;
2. The scalar multiplication satisfies:

$$\forall \lambda \in \mathbb{K}, \forall v, w \in V, \quad \lambda(v + w) = \lambda v + \lambda w,$$

$$\forall \lambda, \mu \in \mathbb{K}, \forall v \in V, \quad (\lambda + \mu)v = \lambda v + \mu v, \quad (\lambda\mu)v = \lambda(\mu v), \quad 1 \cdot v = v,$$

where 1 is the identity element of $\mathbb{K}$.

The elements of V are called vectors, and those of $\mathbb{K}$ are scalars.

[Norm] Let E be a vector space over $\mathbb{K}$. A norm $N(\cdot)$ (or $\|\cdot\|$) on E is a function from E to $\mathbb{R}^+$ satisfying:

1. $\|\lambda v\| = |\lambda| \cdot \|v\|$ for all $\lambda \in \mathbb{K}, v \in E$;
2. $\|u + v\| \leq \|u\| + \|v\|$ for all $u, v \in E$;
3. $\|v\| = 0 \Rightarrow v = 0$.

In this case, the pair $(E, \|\cdot\|)$ is called a normed space. If only the first two properties are satisfied, the function is called a semi-norm.

Remark. Two norms N and N' are said to be equivalent if there exist constants $c_1 > 0$ and $c_2 > 0$ such that:

$$\forall u \in E, \quad c_1 N(u) \leq N'(u) \leq c_2 N(u).$$

If E is finite-dimensional, then all norms on E are equivalent.

Basic Concepts on Matrices

[Matrix] A matrix with n rows and p columns over a field $\mathbb{K}$ is a table of elements of $\mathbb{K}$ with n rows and p columns. The element in row i and column j is denoted a_{ij}. The matrix A is written:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{np} \end{pmatrix} \quad \text{or} \quad A = (a_{ij})_{1 \leq i \leq n, 1 \leq j \leq p}.$$

The set of all such matrices is denoted $M_{n \times p}(\mathbb{K})$.

1.1.1 Special Matrices

Square Matrix. When $p = n$, the matrix A is said to be square of order n:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

Diagonal Matrix. A square matrix where all off-diagonal elements are zero, i.e., $a_{ij} = 0$ if $i \neq j$:

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

The identity matrix of order n, denoted I_n, is the diagonal matrix with $a_{ii} = 1$ for all $1 \leq i \leq n$.

Triangular Matrix. A square matrix with all elements below the diagonal equal to zero is called upper triangular:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix}$$

Similarly, a lower triangular matrix satisfies $a_{ij} = 0$ if $i < j$:

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & a_{nn} \end{pmatrix}$$

Similar Matrices. Let A and B be square matrices of the same order, with B invertible. The matrices A and $B^{-1}AB$ are called similar, and the transformation $A \mapsto B^{-1}AB$ is called a similarity. If B is unitary, then A and $B^{-1}AB$ are said to be unitarily similar.

Symmetric Matrix. A square matrix A is symmetric if $A^T = A$ and skew-symmetric if $A = -A^T$. That is, $a_{ij} = a_{ji}$ for all i, j.

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Orthogonal Matrix. A square matrix $A \in \mathbb{R}^{n \times n}$ is orthogonal if $A^T A = AA^T = I$, i.e., $A^{-1} = A^T$.

Remark. If A is symmetric and B is any matrix, then BAB^T and $B^T AB$ are symmetric.

Positive Definite Matrix. A square matrix A is positive definite if:

$$\forall x \in \mathbb{C}^n,\ x \neq 0 \Rightarrow x^T Ax > 0.$$

If the strict inequality is replaced by $\geq$, then A is said to be positive semi-definite.

Let $A = \begin{pmatrix} 2 & \beta & \\ -2 & -\beta & 2 \end{pmatrix}$ with $\beta \neq -1$. Then A is positive definite but not symmetric.

For all non-zero vectors $X = (x_1, x_2)^T \in \mathbb{R}^2$, we have:

$$(AX, X) = 2(x_1^2 + x_2^2 - x_1 x_2) > 0.$$

[Matrix Norm] Let A be a square matrix with complex coefficients. The matrix norm subordinate to a vector norm is defined by:

$$\|A\| = \sup_{X \neq 0} \frac{\|AX\|}{\|X\|}.$$

That is, $\|\cdot\|$ is the smallest constant $C > 0$ such that:

$$\|AX\| \leq C\|X\|, \quad \forall X \in \mathbb{K}^n.$$

Let $\|\cdot\|_p$, $1 \leq p < \infty$ be the vector norm defined by:

$$\forall X = (x_1, \ldots, x_n)^T, \quad \|X\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

(the case $p = 2$ corresponds to the Hermitian norm).

Let $\|X\|_1 = \max_{1 \leq i \leq n} |x_i|$ and similarly denote by $\|\cdot\|_1$ and $\|\cdot\|_p$ the induced matrix norms.

[Spectral Radius] Let A be a square matrix with complex coefficients. The spectral radius $\rho(A)$ is the largest modulus of the eigenvalues of A. That is:

$$\forall A \in M_n(\mathbb{C}),\ \forall \lambda \in \mathbb{C}, \quad \rho(\lambda A) = |\lambda| \rho(A), \quad \rho(A^k) = \rho(A)^k.$$

Let $\|\cdot\|$ be any matrix norm on $M_n(\mathbb{C})$. Then for all $A \in M_n(\mathbb{C})$:

$$\rho(A) \leq \|A\|.$$

Conversely, for every matrix A and every $\varepsilon > 0$, there exists a subordinate norm $\|\cdot\|$ such that:

$$\|A\| \leq \rho(A) + \varepsilon.$$

Let $A \in M_n(\mathbb{C})$. Then:

$$\lim_{k \to \infty} A^k = 0 \quad \Leftrightarrow \quad \rho(A) < 1.$$

Let B be a square matrix such that $\rho(B) < 1$. Then $I - B$ and $I + B$ are invertible, and:

$$(I - B)^{-1} = I + B + B^2 + \cdots$$

1.1.2 Determinant of a Matrix

The determinant of a square matrix A is the scalar defined by:

$$\det(A) = \sum_{\sigma \in \mathcal{P}} \operatorname{sign}(\sigma) a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)},$$

where $\mathcal{P}$ is the set of all permutations of $(1, \ldots, n)$, and $\operatorname{sign}(\sigma)$ is $+1$ (resp. -1) if σ is an even (resp. odd) permutation.

Eigenvalues and Eigenvectors

Let A be an $n \times n$ matrix. A vector $x \in \mathbb{C}^n$ ($x \neq 0$) is called an **eigenvector** of A if there exists a scalar $\lambda \in \mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$) such that:

$$Ax = \lambda x.$$

The scalar λ is called an **eigenvalue** of A. We say that x is an eigenvector associated with the eigenvalue λ.

Properties

1. If x is an eigenvector of A, then Ax is colinear with x.
2. If x is an eigenvector associated with the eigenvalue λ, then λ is unique for x.
3. The set of eigenvalues of A is called the **spectrum** of A, denoted $\sigma(A)$.
4. If $\lambda = 0$ is an eigenvalue of A, then A is not invertible, i.e., $\det(A) = 0$.
5. The equation $Ax = \lambda x$ can be rewritten as:
$$(\lambda I - A)x = 0.$$
6. If λ is an eigenvalue of A, then this equation has a non-zero solution $x \neq 0$, which implies:
$$\det(\lambda I - A) = 0.$$

The equation $\det(\lambda I - A) = 0$ is called the **characteristic equation** of A, and the polynomial $P(\lambda) = \det(\lambda I - A)$ is called the **characteristic polynomial** of A.

If A is of order n, then $P(\lambda)$ is a degree n polynomial with leading coefficient 1, i.e.,

$$P(\lambda) = \lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_0.$$

Hence, A has at most n distinct eigenvalues.

Let

$$A = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}.$$

Compute the eigenvalues of A:

$$P(\lambda) = \det(\lambda I - A) = \det \begin{bmatrix} \lambda - 1 & -3 \\ -4 & \lambda - 2 \end{bmatrix} = \lambda^2 - 3\lambda - 10.$$

The roots are $\lambda_1 = 5$ and $\lambda_2 = -2$.

1.1.3 Computing Eigenvectors

Let A be a square matrix and λ an eigenvalue. The **eigenspace** associated with λ is the set of non-zero vectors x such that:

$$(A - \lambda I)x = 0.$$

It is denoted $E_\lambda = \{x \in \mathbb{C}^n \mid (A - \lambda I)x = 0\}$.

Let

$$A = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \quad \text{with eigenvalues } \lambda_1 = 5, \lambda_2 = -2.$$

1. Eigenvectors for $\lambda_1 = 5$:

$$A - 5I = \begin{bmatrix} -4 & 3 \\ 4 & -3 \end{bmatrix}, \quad \text{Solve } (A - 5I)x = 0.$$

System:

$$-4x_1 + 3x_2 = 0 \Rightarrow x_1 = \frac{3}{4}x_2.$$

So,

$$E_5 = \text{span}\left\{ \begin{bmatrix} \frac{3}{4} \\ 1 \end{bmatrix} \right\}.$$

2. Eigenvectors for $\lambda_2 = -2$:

$$A + 2I = \begin{bmatrix} 3 & 3 \\ 4 & 4 \end{bmatrix}, \quad \text{Solve } (A + 2I)x = 0.$$

System:

$$3x_1 + 3x_2 = 0 \Rightarrow x_2 = -x_1.$$

So,

$$E_{-2} = \text{span}\left\{ \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}.$$

1.1.4 Properties of Eigenvalues and Eigenvectors

If A is a triangular matrix, then its eigenvalues are the diagonal entries of A.

If A is a diagonal matrix, then its eigenvalues are the diagonal entries.

Let A be an invertible $n \times n$ matrix. If λ is an eigenvalue of A, then λ^{-1} is an eigenvalue of A^{-1}.

Let A be a square matrix and $k \in \mathbb{Z}$. If λ is an eigenvalue of A with corresponding eigenvector x, then λ^k is an eigenvalue of A^k with the same eigenvector x (provided A is invertible if $k < 0$).

Let

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

This is a symmetric matrix. Compute the characteristic polynomial:

$$P(\lambda) = \det(\lambda I - A) = \lambda^3 - 6\lambda^2 + 10\lambda - 4.$$

We find $\lambda = 2$ is a root:

$$P(\lambda) = (\lambda - 2)(\lambda^2 + 4\lambda - 2).$$

Then the eigenvalues are:

$$\lambda_1 = 2, \quad \lambda_2 = 2 - \sqrt{2}, \quad \lambda_3 = 2 + \sqrt{2}.$$

Corresponding eigenspaces:

$$E_{\lambda_1} = \text{span}\left\{\begin{bmatrix}-1\\0\\1\end{bmatrix}\right\}, \quad E_{\lambda_2} = \text{span}\left\{\begin{bmatrix}1\\\sqrt{2}\\1\end{bmatrix}\right\}, \quad E_{\lambda_3} = \text{span}\left\{\begin{bmatrix}\frac{1}{2}\\-\frac{\sqrt{2}}{2}\\\frac{1}{2}\end{bmatrix}\right\}.$$

A matrix $A \in \mathcal{M}_n(\mathbb{K})$ is said to be **diagonalizable** if there exists an invertible matrix $P \in \mathcal{M}_n(\mathbb{K})$ such that:

$$P^{-1}AP = D = \text{diag}(\lambda_1, \ldots, \lambda_n).$$

The columns of P are eigenvectors of A forming a basis of $\mathbb{C}^n$.

Let A be a symmetric matrix ($A^T = A$), then A has n linearly independent eigenvectors. Furthermore, these eigenvectors can be chosen to form an orthonormal basis. Hence, any real symmetric matrix is diagonalizable.

If A is a symmetric matrix with real entries, then all of its eigenvalues are real.

Moreover, we have:

$$\sum_{i=1}^{n} \lambda_i = \text{tr}(A), \qquad \prod_{i=1}^{n} \lambda_i = \det(A),$$

which means that diagonalization preserves both the trace and the determinant.

If A is a positive definite matrix, then all diagonal elements of A are positive.

Properties:

1. A matrix A is **positive definite** if and only if all of its eigenvalues are positive.

2. Two similar matrices have the same spectrum and the same characteristic polynomial.

   Indeed, it is easy to verify that if (λ, X) is an eigenpair of A, then $(\lambda, B^{-1}X)$ is an eigenpair of $B^{-1}AB$, since:

$$(B^{-1}AB)(B^{-1}X) = B^{-1}AX = \lambda B^{-1}X.$$

1.2 Direct Methods

The principle of direct methods essentially consists in transforming the system, after a finite number of manipulations, into a linear system that is easier to solve (often triangular). These "manipulations" are often successive linear combinations of the equations leading to the solution in a finite number of operations. From a matrix point of view, this often amounts to multiplying the system by matrices $M_1, \ldots, M_k$ such that the final system

$$MAX = MB$$

with $M = M_k \cdots M_1$, is easy to invert. In this case, the matrix $T = MA$ is often triangular.

1.2.1 Solving a System with a Triangular Matrix

The simplest linear systems to solve are obviously those where the matrix A is diagonal (and invertible). In this case, the solution is given by:

$$\forall i \leq n, \quad x_i = \frac{b_i}{a_{ii}}$$

More generally, triangular systems are also easy to solve. For example, if the matrix A is upper triangular, then system (2.1) becomes:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{nn}x_n &= b_n \end{aligned}$$

From the last equation, we get:

$$x_n = \frac{b_n}{a_{nn}}$$

The next-to-last equation gives:

$$x_{n-1} = \frac{1}{a_{n-1,n-1}} \left(b_{n-1} - a_{n-1,n}x_n\right)$$

By successively going up the system, we find:

$$\forall k \leq n-1, \quad x_k = \frac{1}{a_{kk}} \left(b_k - a_{k,k+1}x_{k+1} - \cdots - a_{kn}x_n\right)$$

In this formula, computing x_k from $x_{k+1}, \ldots, x_n$ requires $(n-k)$ multiplications and one division. Hence, the total cost is:

$$n \text{ divisions and } \sum_{k=1}^{n}(n-k) = \sum_{j=0}^{n-1} j = \frac{n(n-1)}{2} \text{ multiplications}$$

1.2.2 Gaussian Elimination

Gaussian elimination consists in transforming the matrix A into an upper triangular matrix. For this, we construct $[A|B]$ and transform it into $[A'|B']$, where A' is upper triangular.

Let $A^{(1)} = A$ and $B^{(1)} = B$.

Step 1: If $a_{11}^{(1)} \neq 0$ (otherwise, swap with another row), perform:

- Keep row 1: $L_1^{(2)} \leftarrow L_1^{(1)}$
- For $i = 2$ to n:

$$L_i^{(2)} \leftarrow L_i^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} L_1^{(1)}$$

We obtain a new matrix $[A^{(2)}|B^{(2)}]$ with zero entries below the pivot.
Step 2: Proceed similarly with $A^{(2)}$, assuming $a_{22}^{(2)} \neq 0$. Apply:

$$\begin{aligned} L_1^{(3)} &\leftarrow L_1^{(2)} \\ L_2^{(3)} &\leftarrow L_2^{(2)} \\ L_i^{(3)} &\leftarrow L_i^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} L_2^{(2)} \quad \text{for } i = 3, \ldots, n \end{aligned}$$

Repeat until step p.
At the end, we obtain the system:

$$\begin{aligned} a_{11}^{(1)} x_1 + a_{12}^{(1)} x_2 + \cdots + a_{1n}^{(1)} x_n &= b_1^{(1)} \\ a_{22}^{(2)} x_2 + \cdots + a_{2n}^{(2)} x_n &= b_2^{(2)} \\ &\vdots \\ a_{nn}^{(n)} x_n &= b_n^{(n)} \end{aligned}$$

This is a triangular system, which can be solved using back-substitution:

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}}, \quad x_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j \right) \quad \text{for } i = n-1, \ldots, 1$$

Remark: Gaussian elimination requires approximately $\frac{2}{3}n^3$ operations.
Example: Solve the system:

$$\begin{cases} 2x_1 + x_2 + 4x_4 = 2 \\ -4x_1 - 2x_2 + 3x_3 - 7x_4 = -9 \\ 4x_1 + x_2 - 2x_3 + 8x_4 = 2 \\ -3x_2 - 12x_3 - x_4 = 2 \end{cases}$$

Using Gaussian elimination, we obtain the triangular system:

$$\begin{cases} 2x_1 + x_2 + 4x_4 = 2 \\ -3x_2 - 12x_3 - x_4 = 2 \\ 2x_3 + \frac{1}{3}x_4 = -5 \\ \frac{1}{2}x_4 = \frac{5}{2} \end{cases} \Rightarrow x_4 = 5, \quad x_3 = -\frac{10}{3}, \quad x_2 = 11, \quad x_1 = -\frac{29}{2}$$

1.2.3 Gauss-Jordan Method

The Gauss-Jordan method consists in transforming the matrix A into the identity matrix. We transform $[A|B]$ into $[I|B']$, where I is the identity matrix. Then:

$$AX = B \Rightarrow IX = B' \Rightarrow X = B'$$

Let $A^{(1)} = A$ and $B^{(1)} = B$.
Step 1: If $a_{11}^{(1)} \neq 0$, perform:

- Normalize row 1: $L_1^{(2)} \leftarrow \frac{L_1^{(1)}}{a_{11}^{(1)}}$
- For $i = 2$ to n:

$$L_i^{(2)} \leftarrow L_i^{(1)} - a_{i1}^{(1)} L_1^{(2)}$$

Step 2: If $a_{22}^{(2)} \neq 0$, perform:

- $L_2^{(3)} \leftarrow \frac{L_2^{(2)}}{a_{22}^{(2)}}$
- For all $i \neq 2$: $L_i^{(3)} \leftarrow L_i^{(2)} - a_{i2}^{(2)} L_2^{(3)}$

And so on for step k:

- Normalize L_k: $L_k^{(k+1)} \leftarrow \frac{L_k^{(k)}}{a_{kk}^{(k)}}$
- For all $i \neq k$: $L_i^{(k+1)} \leftarrow L_i^{(k)} - a_{ik}^{(k)} L_k^{(k+1)}$

We then have:

$$\left[\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & a_{1;k+1}^{(k+1)} \cdots a_{1n}^{(k+1)} \\ & & & & b_1^{(k+1)} \\ 0 & 1 & \cdots & 0 & a_{2;k+1}^{(k+1)} \cdots a_{2n}^{(k+1)} \\ & & & & b_2^{(k+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & & & & \\ 0 & 0 & \cdots & 1 & a_{k;k+1}^{(k+1)} \cdots a_{kn}^{(k+1)} \\ & & & & b_k^{(k+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & & & & \\ 0 & 0 & \cdots & 0 & a_{n;k+1}^{(k+1)} \cdots a_{nn}^{(k+1)} \\ & & & & b_n^{(k+1)} \end{array}\right]$$

Remark. The Gauss-Jordan method requires n^3 elementary operations. It is slower than the Gaussian method, except that in this case, the matrix A is directly inverted.

Example 2.2 Consider the system:

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ 4x_1 + 3x_2 - x_3 = 6 \\ 3x_1 + 5x_2 + 3x_3 = 4 \end{cases}$$

1. Write the system in the form $AX = B$. 2. Find the solution $X = (x_1, x_2, x_3)^T$. 3. Find the inverse of matrix A.

Solution:

1. The matrix form is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix} \Rightarrow AX = B$$

2. We compute the determinant:

$$\det(A) = \begin{vmatrix} 3 & -1 \\ 5 & 3 \end{vmatrix} - \begin{vmatrix} 4 & -1 \\ 3 & 3 \end{vmatrix} + \begin{vmatrix} 4 & 3 \\ 3 & 5 \end{vmatrix} = 10 \neq 0$$

So the system has a unique solution. Applying Gauss-Jordan transformations:
Step 1: Since $a_{11}^{(1)} = 1 \neq 0$, with:

$$L_2 \leftarrow L_2 - 4L_1, \quad L_3 \leftarrow L_3 - 3L_1$$

we obtain:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & -1 & -5 & 2 \\ 0 & 2 & 0 & 1 \end{array}\right]$$

Step 2: $a_{22}^{(2)} = -1 \neq 0$, with:

$$L_3 \leftarrow L_3 + 2L_2, \quad L_1 \leftarrow L_1 - L_2$$

we obtain:

$$\left[\begin{array}{ccc|c} 1 & 0 & -4 & 3 \\ 0 & 1 & 5 & -2 \\ 0 & 0 & -10 & 5 \end{array}\right]$$

Step 3: $a_{33}^{(3)} = -10 \neq 0$, with:

$$L_3 \leftarrow \frac{1}{-10} L_3, \quad \text{then eliminate above entries}$$

we get:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & -\frac{1}{2} \end{array}\right]$$

Thus $X = (1, \frac{1}{2}, -\frac{1}{2})^T$, and:

$$A^{-1} = \frac{1}{10} \begin{bmatrix} 14 & 2 & -4 \\ -15 & 0 & 5 \\ 11 & -2 & -1 \end{bmatrix}$$

1.2.4 LU Decomposition of A

The method consists in decomposing the matrix A as $A = LU$, where L is a lower triangular unit matrix and U is an upper triangular matrix.

To solve the system:

$$AX = B \Leftrightarrow LUX = B \Leftrightarrow (LY = B \text{ and } UX = Y)$$

Thus, solving $AX = B$ is reduced to solving two triangular systems $LY = B$ and $UX = Y$.

The method: Matrices are computed alternately row by row using the following partition:

$$L = U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ \ell_{21} & u_{22} & \cdots & u_{2n} \\ \ell_{31} & \ell_{32} & u_{33} & \cdots \\ \vdots & \vdots & \ddots & \ddots \\ \ell_{n1} & \ell_{n2} & \cdots & u_{nn} \end{bmatrix}$$

Step 1: $i = 1$, set $u_{1j} = a_{1j}$ for $j = 1, \ldots, n$, and:

$$\ell_{i1} = \frac{a_{i1}}{u_{11}}, \quad i = 2, \ldots, n$$

Step s: $i = s$,

$$u_{sj} = a_{sj} - \sum_{k=1}^{s-1} \ell_{sk} u_{kj}, \quad j = s, \ldots, n$$

$$\ell_{is} = \frac{1}{u_{ss}} \left(a_{is} - \sum_{k=1}^{s-1} \ell_{ik} u_{ks} \right), \quad i = s+1, \ldots, n$$

The resulting matrices are:

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \ell_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Example 2.3 Solve by LU decomposition:

$$\begin{cases} 3x_1 - x_2 = 5 \\ -2x_1 + x_2 + x_3 = 0 \\ 2x_1 - x_2 + 4x_3 = 15 \end{cases} \Rightarrow A = \begin{bmatrix} 3 & -1 & 0 \\ -2 & 1 & 1 \\ 2 & -1 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 \\ 0 \\ 15 \end{bmatrix}$$

Decompose $A = LU$. By computing:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2/3 & 1 & 0 \\ 2/3 & -1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 0 \\ 0 & 1/3 & 1 \\ 0 & 0 & 5 \end{bmatrix}$$

So $AX = B \Leftrightarrow LUX = B$
Solving $LY = B$ gives $Y = (5, 10/3, 25/3)^T$, then $UX = Y$ yields $X = (10/3, 5, 5/3)^T$.
Example 2.4

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ 4x_1 + 3x_2 - x_3 = 6 \\ 3x_1 + 5x_2 + 3x_3 = 4 \end{cases} \Rightarrow A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

LU decomposition yields:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix}$$

Then $LY = B$ gives $Y = (1, 2, 5)^T$ and $UX = Y$ yields $X = (1, 1/2, -1/2)^T$

1.2.5 Cholesky Factorization

In the case where the matrix is symmetric, the memory required to solve a linear system can be reduced.

Since the matrix is symmetric, we can store only the lower triangular part along with its main diagonal. We then use the **Cholesky factorization**, which consists in decomposing A as:

$$A = LL^T$$

where L is a lower triangular matrix. The upper triangular matrix U of the LU decomposition is thus replaced by the transpose of L, reducing the memory footprint.

To solve a linear system, we proceed in two steps:

1. Solve the lower triangular system $L\vec{y} = \vec{b}$
2. Then solve the upper triangular system $L^T\vec{x} = \vec{y}$

We follow the same logic as in the LU decomposition but now applied to a symmetric matrix:

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

By performing the matrix multiplication, we obtain:

$$\begin{aligned} a_{11} = l_{11}^2 \quad &\Rightarrow \quad l_{11} = \sqrt{a_{11}} \\ a_{21} = l_{21}l_{11} \quad &\Rightarrow \quad l_{21} = \frac{a_{21}}{l_{11}} \\ a_{31} = l_{31}l_{11} \quad &\Rightarrow \quad l_{31} = \frac{a_{31}}{l_{11}} \end{aligned}$$

This completes the first column of L. Note that $l_{11} \neq 0$ (the pivot must be non-zero). We then compute the second column:

$$\begin{aligned} a_{22} = l_{21}^2 + l_{22}^2 \quad &\Rightarrow \quad l_{22} = \sqrt{a_{22} - l_{21}^2} \\ a_{32} = l_{31}l_{21} + l_{32}l_{22} \quad &\Rightarrow \quad l_{32} = \frac{a_{32} - l_{31}l_{21}}{l_{22}} \end{aligned}$$

1.3 Iterative Methods

The principle of iterative methods is to construct a sequence $(X^{(k)})_{k\geq 0}$ that is easy to compute and such that:

$$\lim_{k\to\infty} X^{(k)} = X$$

where X is the solution of the linear system:

$$AX = B \tag{2.3}$$

A usual way to construct such a sequence is to decompose:

$$A = S - T$$

with S easily invertible. Then:

$$SX^{(k+1)} = TX^{(k)} + B, \quad X^{(0)} \text{ given}$$

or equivalently:

$$X^{(k+1)} = MX^{(k)} + C, \quad M = S^{-1}T, \quad C = S^{-1}B$$

If the sequence converges, its limit is the solution to (2.3).

The method requires:

1. Matrix S is easily invertible. 2. The sequence $(X^{(k)})$ converges for any initial $X^{(0)}$.

Condition (1) typically means solving $SY = B_0$ is cheaper than solving $AX = B$, e.g., if S is diagonal or triangular and A is full.

A method is convergent if condition (2) holds; otherwise, it is divergent.

To study convergence, define the error:

$$E^{(k)} = X^{(k)} - X$$

Then subtracting:

$$E^{(k+1)} = ME^{(k)}$$

1.3.1 Jacobi Method

Consider the linear system:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n
\end{aligned}$$

We assume that all diagonal elements are non-zero ($a_{ii} \neq 0, \ \forall i$). As with any iterative method, we start with an initial approximation of the solution, denoted as:

$$\begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix}$$

The Jacobi algorithm consists in isolating the diagonal term in each equation:

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{a_{11}}\left(b_1 - \sum_{j=2}^{n} a_{1j}x_j^{(k)}\right)\\
x_2^{(k+1)} &= \frac{1}{a_{22}}\left(b_2 - \sum_{\substack{j=1\\ j\neq 2}}^{n} a_{2j}x_j^{(k)}\right)\\
x_3^{(k+1)} &= \frac{1}{a_{33}}\left(b_3 - \sum_{\substack{j=1\\ j\neq 3}}^{n} a_{3j}x_j^{(k)}\right)\\
&\vdots\\
x_n^{(k+1)} &= \frac{1}{a_{nn}}\left(b_n - \sum_{j=1}^{n-1} a_{nj}x_j^{(k)}\right)
\end{aligned}
$$

More generally, we write:

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

Example Consider the system:

$$
\begin{aligned}
3x_1 + x_2 - x_3 &= 2\\
x_1 + 5x_2 + 2x_3 &= 17\\
2x_1 - x_2 - 6x_3 &= -18
\end{aligned}
$$

The Jacobi method gives:

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{3}(2 - x_2^{(k)} + x_3^{(k)})\\
x_2^{(k+1)} &= \frac{1}{5}(17 - x_1^{(k)} - 2x_3^{(k)})\\
x_3^{(k+1)} &= -\frac{1}{6}(-18 - 2x_1^{(k)} + x_2^{(k)})
\end{aligned}
$$

Starting from the initial guess $\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$:

- First iteration:

  $$x_1^{(1)} = \frac{1}{3}(2-0+0) = \frac{2}{3}, \quad x_2^{(1)} = \frac{1}{5}(17-0-0) = \frac{17}{5}, \quad x_3^{(1)} = \frac{-1}{6}(-18-0+0) = 3$$

- Second iteration:

  $$x_1^{(2)} = \frac{1}{3}\left(2 - \frac{17}{5} + 3\right) = \frac{8}{15}, \quad x_2^{(2)} = \frac{1}{5}\left(17 - \frac{2}{3} - 6\right) = \frac{31}{15}, x_3^{(2)} = \frac{-1}{6}\left(-18 - 2\cdot\frac{2}{3} + \frac{17}{5}\right) \approx 2.6$$

The iterations produce the following table:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.000000	0.000000	0.000000
1	0.666667	3.400000	3.000000
2	0.533333	2.066667	2.655556
3	0.862963	2.231111	2.833333
4	0.867407	2.094074	2.915802
5	0.940576	2.060198	2.940123
6	0.959975	2.035835	2.970159
7	0.978108	2.019941	2.980686
8	0.986915	2.012104	2.989379
9	0.992425	2.006865	2.993621
10	0.995585	2.004067	2.996331

The solution converges to:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Example 4.36. The following system cannot be directly solved using Jacobi since $a_{11} = 0$:

$$\begin{aligned} 0x_1 + 3x_2 + x_3 &= 7 \\ 5x_1 + x_2 - 2x_3 &= 15 \\ 3x_1 - 4x_2 + 8x_3 &= 9 \end{aligned}$$

We swap the first two rows:

$$\begin{aligned} 5x_1 + x_2 - 2x_3 &= 15 \\ 0x_1 + 3x_2 + x_3 &= 7 \\ 3x_1 - 4x_2 + 8x_3 &= 9 \end{aligned}$$

The Jacobi method gives:

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{5}(15 - x_2^{(k)} + 2x_3^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{3}(7 - x_3^{(k)}) \\ x_3^{(k+1)} &= \frac{1}{8}(9 - 3x_1^{(k)} + 4x_2^{(k)}) \end{aligned}$$

Starting from the initial guess $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, we obtain:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.000000	0.000000	0.000000
1	3.000000	2.333333	1.125000
2	2.983333	1.958333	1.166667
3	3.075000	1.944444	0.985417
4	3.005278	2.004861	0.944097
5	2.976667	2.018634	1.000451
6	2.996454	1.999850	1.018067
7	3.007257	1.993978	1.001255
8	3.001706	1.999582	0.994268
9	2.997791	2.001911	0.999151
10	2.999278	2.000283	1.001784
11	3.000657	1.999405	1.000412
12	3.000284	1.999863	0.999456
13	2.999810	2.000181	0.999825
14	2.999894	2.000058	1.000162
15	3.000053	1.999946	1.000069

Convergence toward:

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Matrix formulation. The Jacobi method can be written in matrix form. We decompose the matrix A as:

$$A = D + L + U$$

where:

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Then the system $A\vec{x} = \vec{b}$ becomes:

$$\begin{aligned} (D + L + U)\vec{x} &= \vec{b} \\ D\vec{x} &= -(L + U)\vec{x} + \vec{b} \\ \vec{x} &= -D^{-1}(L + U)\vec{x} + D^{-1}\vec{b} \end{aligned}$$

Define:

$$T_J = -D^{-1}(L + U), \quad \vec{c}_J = D^{-1}\vec{b}$$

Thus:

$$\vec{x}^{(k+1)} = T_J\vec{x}^{(k)} + \vec{c}_J$$

The Jacobi method is therefore a fixed-point method of the form:

$$\vec{g}(\vec{x}) = T_J\vec{x} + \vec{c}_J$$

Convergence depends on the spectral radius $\rho(T_J)$. If $\rho(T_J) < 1$, the method converges. A matrix A is said to be **strictly diagonally dominant** if:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad \forall i$$

1.3.2 Gauss-Seidel Method

The Gauss-Seidel method is an improved variant of the Jacobi method. To fully understand its principle, it is sufficient to reconsider the Jacobi method and observe how it might be enhanced. In the general case, the Jacobi method is written as:

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

which can also be expressed as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \tag{4.29}$$

The Gauss-Seidel method is based on the simple observation that the calculation of $x_2^{(k+1)}$ requires the use of $x_1^{(k)}, x_3^{(k)}, \ldots, x_n^{(k)}$ from the previous iteration. However, at iteration $k+1$, during the computation of $x_2^{(k+1)}$, we already have a better approximation of x_1 than $x_1^{(k)}$, namely $x_1^{(k+1)}$. Similarly, when calculating $x_3^{(k+1)}$, one can use $x_1^{(k+1)}$ and $x_2^{(k+1)}$, which have already been computed. More generally, for the calculation of $x_i^{(k+1)}$, one can use $x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_{i-1}^{(k+1)}$ already computed and $x_{i+1}^{(k)}, x_{i+2}^{(k)}, \ldots, x_n^{(k)}$ from the previous iteration. This leads to the following expression:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \tag{4.30}$$

Using the notation introduced for the Jacobi method, the Gauss-Seidel method in matrix form is:

$$\vec{x}^{(k+1)} = D^{-1}(\vec{b} - T_I \vec{x}^{(k+1)} - T_S \vec{x}^{(k)})$$

or equivalently:

$$(T_I + D)\vec{x}^{(k+1)} = \vec{b} - T_S \vec{x}^{(k)}$$

and finally:

$$\vec{x}^{(k+1)} = -(T_I + D)^{-1} T_S \vec{x}^{(k)} + (T_I + D)^{-1} \vec{b} = T_{GS} \vec{x}^{(k)} + \vec{c}_{GS}$$

Example 4.41 Consider the linear system:

$$\begin{cases} 3x_1 + x_2 - x_3 = 2 \\ x_1 + 5x_2 + 2x_3 = 17 \\ 2x_1 - x_2 - 6x_3 = -18 \end{cases}$$

The Gauss-Seidel method gives:

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{3}(2 - x_2^{(k)} + x_3^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{5}(17 - x_1^{(k+1)} - 2x_3^{(k)}) \\ x_3^{(k+1)} &= -\frac{1}{6}(-18 - 2x_1^{(k+1)} + x_2^{(k+1)}) \end{aligned}$$

Note the index differences from the Jacobi method. Starting from:

$$\vec{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

we obtain:

$$\begin{aligned} x_1^{(1)} &= \frac{1}{3}(2 - 0 + 0) = \frac{2}{3} \\ x_2^{(1)} &= \frac{1}{5}\left(17 - \frac{2}{3} - 0\right) = \frac{49}{15} \\ x_3^{(1)} &= -\frac{1}{6}(-18 - 2 \cdot \frac{2}{3} + \frac{49}{15}) = \frac{241}{90} \end{aligned}$$

At the second iteration:

$$\begin{aligned} x_1^{(2)} &= \frac{1}{3}\left(2 - \frac{49}{15} + \frac{241}{90}\right) \approx 0.4703704 \\ x_2^{(2)} &= \frac{1}{5}\left(17 - 0.4703704 - 2 \cdot \frac{241}{90}\right) \approx 2.234815 \\ x_3^{(2)} &= -\frac{1}{6}(-18 - 2 \cdot 0.4703704 + 2.234815) \approx 2.784321 \end{aligned}$$

and so on.

Gauss-Seidel Iterations

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
1	0.6666667	3.266667	2.677778	6	0.9914991	2.005729	2.996212
2	0.4703704	2.234815	2.784321	7	0.9968277	2.002150	2.998584
3	0.8498354	2.116305	2.930561	8	0.9988115	2.000804	2.999470
4	0.9380855	2.040158	2.972669	9	0.9995553	2.000301	2.999802
5	0.9775034	2.015432	2.989929	10	0.9998335	2.000113	2.999926

We observe that, for the same number of iterations, the approximate solution obtained with the Gauss-Seidel method is more accurate. The Gauss-Seidel method generally converges faster than the Jacobi method, though not always.

1.3.3 Relaxation Method

Let $A \in \mathbb{R}^{n\times n}$, $X \in \mathbb{R}^n$, and $B \in \mathbb{R}^n$. The SOR method is an iterative technique to solve the system:

$$AX = B$$

The matrix A is decomposed as:

$$A = D - L - U$$

where:

- D is the diagonal of A,
- $-L$ is the strict lower triangular part,
- $-U$ is the strict upper triangular part.

Given a relaxation parameter $\omega \in (0, 2)$, the SOR iteration formula is:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

for $i = 1, 2, \ldots, n$.

Example

Solve the system using the SOR method:

$$\begin{cases} 4x_1 - x_2 = 3 \\ -2x_1 + 5x_2 = -3 \end{cases}$$

Write in matrix form:

$$A = \begin{bmatrix} 4 & -1 \\ -2 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 3 \\ -3 \end{bmatrix}$$

Let $\omega = 1.25$, and start with an initial guess:

$$X^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

First iteration (k = 0):

$$x_1^{(1)} = (1-\omega)x_1^{(0)} + \frac{\omega}{4}(3 + x_2^{(0)}) = 0 + \frac{1.25}{4}(3) = 0.9375$$

$$x_2^{(1)} = (1-\omega)x_2^{(0)} + \frac{\omega}{5}(-3 + 2x_1^{(1)}) = 0 + \frac{1.25}{5}(-3 + 2\cdot 0.9375) = -0.21875$$

So,

$$X^{(1)} = \begin{bmatrix} 0.9375 \\ -0.21875 \end{bmatrix}$$

Continue iterating until convergence, i.e., when $\|X^{(k+1)} - X^{(k)}\| < \varepsilon$ for some tolerance ε.

1.4 Convergence of Iterative Methods and error estimation

Let us denote:

$$E(k) = X(k) - X$$

Then:

$$E(k) = M^k E(0) = M^k (X(0) - X)$$

Hence:

$$\|E(k)\| \leq \|M^k\| \cdot \|X(0) - X\|$$

where $\|\cdot\|$ denotes both a vector norm and the associated matrix norm subordinate to this vector norm.

Thus, a sufficient condition for $E(k) \to 0$ is:

$$\lim_{k \to +\infty} M^k = 0$$

which is equivalent to saying that the matrix M is convergent. This is known to be equivalent to:

$$\rho(M) < 1$$

We summarize this in the following theorem:

Theorem 2.5: Let M be a square matrix, C a column vector, and X a solution to the system:

$$X = MX + C$$

A necessary and sufficient condition for the recurrence relation:

$$X^{(k+1)} = MX^{(k)} + C, \quad X^{(0)} \text{ given}$$

to converge for any $X^{(0)} \in \mathbb{C}^n$ is:

$$\rho(M) < 1$$

In this case:

$$\|X^{(k)} - X\| \leq \|M^k\| \cdot \|X^{(0)} - X\|$$

for any vector norm $\|\cdot\|$ (the matrix norm being subordinate to this vector norm).

We now specify some possible choices for the matrix S above. Each of these choices corresponds to a specific method.

We begin by decomposing the matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ as:

$$A = D - L - U$$

where:

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}, \quad -L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}, \quad -U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}$$

We assume throughout that D is invertible.

Jacobi Method

This corresponds to the choice $S = D$ and the iterative scheme:

$$DX^{(k+1)} = (L+U)X^{(k)} + B, \quad X^{(0)} \text{ given}$$

Then:

$$X^{(k+1)} = D^{-1}(L+U)X^{(k)} + D^{-1}B$$

and:

$$M_J = D^{-1}(L+U)$$

is called the Jacobi matrix.

Gauss-Seidel Method

This corresponds to the choice $S = D - L$, i.e., the scheme:

$$X^{(k+1)} = (D-L)^{-1}UX^{(k)} + (D-L)^{-1}B, \quad X^{(0)} \text{ given}$$

Then:

$$M_G = (D-L)^{-1}U$$

which is called the Gauss-Seidel matrix.

Example 2.6: Consider the system:

$$\begin{cases} 4x_1 + x_2 = 3 \\ x_1 - 2x_2 = -15 \end{cases}$$

Solve it using both Jacobi and Gauss-Seidel iterative processes.

1.5 Exercices

Exercise 2.1: Solve the following linear system using the Gauss method:

$$\begin{aligned} 3x_1 + 17x_2 + 10x_3 &= -10 \\ 2x_1 + 4x_2 - 2x_3 &= 12 \\ 6x_1 + 18x_2 - 12x_3 &= 60 \end{aligned}$$

Exercise 2.2: Solve using the Gauss-Jordan method the linear system $AX = B$, where:

$$B = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \quad A = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

Exercise 2.3: Perform an LU factorization of the matrix:

$$A = \begin{bmatrix} 3 & 5 \\ 6 & 7 \end{bmatrix}$$

Exercise 2.4: Consider the linear system:

$$(S) : \begin{cases} -x_1 + 3x_2 + x_4 = 4 \\ 3x_1 + 5x_2 + 8x_3 - 3x_4 = -1 \\ -2x_1 - 6x_2 + 3x_3 + 2x_4 = -2 \\ -x_2 + 2x_3 + x_4 = 1 \end{cases}$$

1. Write the system in matrix form $AX = B$. 2. Solve the system using the Gauss-Jordan algorithm and deduce the inverse of the matrix A.

Exercise 2.5: Write the Jacobi and Gauss-Seidel methods for the system $AX = B$, where:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Exercise 2.6: Given the linear system:

$$\begin{cases} x_1 + x_2 + x_3 = 14 \\ 3x_1 + 9x_2 + 27x_3 = 120 \\ 2x_1 + 4x_2 + 8x_3 = 50 \end{cases}$$

1. Write it in matrix form $AX = B$. 2. Give the L and U matrices from LU factorization of A. 3. Solve the system.

Exercise 2.7: Consider the system:

$$\begin{cases} 3x_1 + x_2 - x_3 = 2 \\ x_1 + 5x_2 + 2x_3 = 17 \\ 2x_1 - x_2 - 6x_3 = -18 \end{cases}$$

1. Write in matrix form $AX = B$ and give matrices D, L, U such that $A = D - L - U$. 2. Determine the Gauss-Seidel matrix and show that $\rho(M_J) < 1$. 3. Starting from

$X_0 = (0.8333, 2.6333, 2.8389)^T$, compute an approximation to the solution using Gauss-Seidel to within 10^{-1}.

Exercise 2.8: Consider the system:

$$\begin{cases} 5x_1 + x_2 - 2x_3 = 15 \\ 3x_2 + x_3 = 7 \\ 3x_1 - 4x_2 + 8x_3 = 9 \end{cases}$$

1. Write in matrix form $AX = B$ and give matrices D, L, U such that $A = D - L - U$. 2. Determine the Jacobi matrix and show that $\rho(M) < 1$. 3. Compute an approximation to the solution using Jacobi to within 10^{-1}.

Solution: 1. The system:

$$\begin{cases} 5x_1 + x_2 - 2x_3 = 15 \\ 3x_2 + x_3 = 7 \\ 3x_1 - 4x_2 + 8x_3 = 9 \end{cases}$$

is written as $AX = B$ with:

$$A = \begin{bmatrix} 5 & 1 & -2 \\ 0 & 3 & 1 \\ 3 & -4 & 8 \end{bmatrix} = D - L - U$$

with:

$$D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -3 & 4 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -1 & 2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

2. The Jacobi method solves $AX = B$ via:

$$X^{(k+1)} = MX^{(k)} + C, \quad M_J = D^{-1}(L + U), \quad C = D^{-1}B$$

We have:

$$M_J = \begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/8 \end{bmatrix} \begin{bmatrix} 0 & -1 & 2 \\ 0 & 0 & -1 \\ -3 & 4 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1/5 & 2/5 \\ 0 & 0 & -1/3 \\ -3/8 & 1/2 & 0 \end{bmatrix}$$

Since $\|M_J\| = 1/2 < 1$, and $\rho(M_J) < \|M_J\| < 1$, then $\rho(M_J) < 1$.

3. The Jacobi algorithm converges and is written:

$$X^{(k+1)} = MX^{(k)} + C, \quad C = D^{-1}B = \begin{bmatrix} 3 \\ 7/3 \\ 9/8 \end{bmatrix}$$

Giving:

$$\begin{cases} x_1^{(k+1)} = -\frac{1}{5}x_2^{(k)} + \frac{2}{5}x_3^{(k)} + 3 \\ x_2^{(k+1)} = -\frac{1}{3}x_3^{(k)} + \frac{7}{3} \\ x_3^{(k+1)} = -\frac{3}{8}x_1^{(k)} + \frac{1}{2}x_2^{(k)} + \frac{9}{8} \end{cases}$$

Starting from $X^{(0)} = (0, 0, 0)^T$, we get:

$$X^{(1)} = \begin{bmatrix} 3.000000 \\ 2.333333 \\ 1.125000 \end{bmatrix} \Rightarrow \|X^{(1)} - X^{(0)}\| = 3.000000 > 10^{-1}$$

Similarly, we have:

$$\begin{cases} x_1^{(2)} = 2.983333 \\ x_2^{(2)} = 1.958333 \\ x_3^{(2)} = 1.166667 \end{cases}$$

Hence,

$$X^{(2)} - X^{(1)} = (-0.016667,\ -0.375000,\ 0.041667)^T$$

and thus

$$\|X^{(2)} - X^{(1)}\| = 0.375 > 10^{-1}$$

The next iteration gives:

$$\begin{cases} x_1^{(3)} = 3.075000 \\ x_2^{(3)} = 1.944444 \\ x_3^{(3)} = 0.985417 \end{cases}$$

Hence,

$$X^{(3)} - X^{(2)} = (0.091667,\ -0.013889,\ -0.18125)^T$$

and

$$\|X^{(3)} - X^{(2)}\| = 0.18125 > 10^{-1}$$

Likewise, we have:

$$\begin{cases} x_1^{(4)} = 3.005278 \\ x_2^{(4)} = 2.004861 \\ x_3^{(4)} = 0.944097 \end{cases}$$

So,

$$X^{(4)} - X^{(3)} = (-0.069722,\ 0.060417,\ -0.04132)^T$$

and

$$\|X^{(4)} - X^{(3)}\| = 0.069722 < 10^{-1}$$

We conclude that:

$$X \approx X^{(4)}$$

Chapter 2

Computation of Eigenvalues and Eigenvectors

Introduction

In the general case, it is not easy to compute the eigenvalues of a matrix A, since the entries of A are not always exact values. For example, in a physical or chemical problem, there are always perturbations in the measurement of data, regardless of the precision of the instruments used. For this reason, direct methods for computing eigenvalues are often very complex.

Since computing the eigenvalues consists in determining the roots of the characteristic polynomial, it is not easy to calculate them exactly. For instance, starting from matrices of order strictly greater than 4, there is no general method to compute the exact roots of the characteristic polynomial.

Example Let the matrix A be defined by:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{bmatrix}$$

The characteristic polynomial is:

$$P(\lambda) = -\lambda^3 + 3\lambda^2 + 26\lambda + 20$$

We observe that it is not possible to compute the zeros of P using a general closed-form method. By applying the Intermediate Value Theorem, we can find three roots approximately located in the following intervals:

$$\beta \in [-1, -0.5], \quad \gamma \in [-3.5, -3], \quad \delta \in [7, 7.2]$$

Using the bisection method (dichotomy), we obtain approximate solutions:

$$\beta \approx 0.88679, \quad \gamma \approx -3.1879, \quad \delta \approx 7.0747$$

2.1 Direct Methods for Eigenvalue Computation

Principle: Direct methods compute eigenvalues **in a finite number of steps** (unlike iterative approaches). These are suitable for small to medium-sized matrices but may be computationally expensive for large systems.

2.1.1 Overview of Direct Eigenvalue Computation Methods:

1. **Characteristic Polynomial:** Eigenvalues are obtained by solving $\det(A-\lambda I) = 0$.

   Example 1:

   For $A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$, solving $\lambda^2 - 7\lambda + 10 = 0$ yields $\lambda = 5, 2$.

   Example 2:

   For $A = \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$:

   $$\det(A - \lambda I) = \begin{vmatrix} 3-\lambda & 1 \\ 2 & 2-\lambda \end{vmatrix} = \lambda^2 - 5\lambda + 4 = 0$$

   Eigenvalues: $\lambda = 4, 1$.

   Example 3:

   For $B = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}$:

   $$\det(B - \lambda I) = (3-\lambda)(\lambda^2 - 4\lambda + 3) = 0$$

   Eigenvalues: $\lambda = 3, 1, 3$.

2. **QR Decomposition (One-Step):** Factorize $A = QR$, then compute $A' = RQ$. Diagonal entries approximate eigenvalues. *Example:* $A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ transforms to $A' = \begin{pmatrix} 3 & 0 \\ 0 & -1 \end{pmatrix}$ after one QR step.

   QR Decomposition of a 3x3 Matrix

   Consider the matrix:

   $$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

   Step 1: Gram-Schmidt Orthogonalization

   (a) **First column (q1):**

   $$\mathbf{a}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{q}_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

(b) **Second column (q2):**

$$\mathbf{a}_2 = \begin{pmatrix}1\\0\\1\end{pmatrix}, \quad \mathbf{u}_2 = \mathbf{a}_2 - (\mathbf{q}_1^\top \mathbf{a}_2)\mathbf{q}_1 = \begin{pmatrix}1\\0\\1\end{pmatrix} - \frac{1}{2}\begin{pmatrix}1\\1\\0\end{pmatrix} = \frac{1}{2}\begin{pmatrix}1\\-1\\2\end{pmatrix}$$

$$\mathbf{q}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} = \frac{1}{\sqrt{6}}\begin{pmatrix}1\\-1\\2\end{pmatrix}$$

(c) **Third column (q3):**

$$\mathbf{a}_3 = \begin{pmatrix}0\\1\\1\end{pmatrix}, \quad \mathbf{u}_3 = \mathbf{a}_3 - (\mathbf{q}_1^\top \mathbf{a}_3)\mathbf{q}_1 - (\mathbf{q}_2^\top \mathbf{a}_3)\mathbf{q}_2$$

$$= \begin{pmatrix}0\\1\\1\end{pmatrix} - \frac{1}{2}\begin{pmatrix}1\\1\\0\end{pmatrix} - \frac{1}{6}\begin{pmatrix}1\\-1\\2\end{pmatrix} = \frac{1}{3}\begin{pmatrix}-2\\2\\2\end{pmatrix}$$

$$\mathbf{q}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} = \frac{1}{\sqrt{12}}\begin{pmatrix}-2\\2\\2\end{pmatrix} = \frac{1}{\sqrt{3}}\begin{pmatrix}-1\\1\\1\end{pmatrix}$$

Step 2: Construct Q and R Matrices

$$Q = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}}\\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}}\end{pmatrix}, \quad R = Q^\top A = \begin{pmatrix}\sqrt{2} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}}\\ 0 & 0 & \frac{2}{\sqrt{3}}\end{pmatrix}$$

Step 3: Verify the Decomposition

$$QR = \begin{pmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\end{pmatrix} = A \quad \checkmark$$

QR Algorithm for Eigenvalues

Initialize $A_0 \leftarrow A$ $k = 1$ to `max_iterations` a. Compute shift: $\mu \leftarrow A_{k-1}(n,n)$ *(Wilkinson shift)*b. QR decomposition: $Q_k, R_k \leftarrow$ `qr_decomp`$(A_{k-1} - \mu I)$c. Update matrix: $A_k \leftarrow R_k Q_k + \mu I$d. If $\|$`offdiag`$(A_k)\| < \varepsilon$: **break** loope. `diag`(A_k) *(approximate eigenvalues)*

After 3 iterations:

$$A_3 = \begin{pmatrix}2.000 & 0.000 & 0.000\\ 0.000 & -0.414 & 0.000\\ 0.000 & 0.000 & 1.414\end{pmatrix}$$

Eigenvalues

$$\lambda_1 = 2.000, \quad \lambda_2 = -0.414, \quad \lambda_3 = 1.414$$

(d) **Schur Decomposition:** For any matrix A, $A = QTQ^*$ where T is upper triangular (eigenvalues on diagonal).

(e) **Jacobi Method (Symmetric Matrices):** Applies Givens rotations to diagonalize A. *Example:* $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ becomes $\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$ after one rotation.

Note: These methods are rarely used for $n > 4$ due to high computational cost ($O(n^3)$). Hybrid approaches (e.g., QR with shifts) are preferred for larger matrices.

2.2 Power Iteration Method for Dominant Eigenvalue

This method approximates the dominant eigenvalue, as we shall see. Certain conditions are necessary for the application of this method.

Let A be a real square matrix of order n, with eigenvalues ordered such that:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|. \tag{2.1}$$

Here, λ_1 is the dominant eigenvalue of A. Let x_1 be its associated normalized eigenvector. If the eigenvectors of A are linearly independent, then λ_1 and x_1 can be approximated by the following iterative process called the power method.

Algorithm

Given an arbitrary initial vector $x^{(0)} \in \mathbb{C}^n$, let

$$y^{(0)} = \frac{x^{(0)}}{\|x^{(0)}\|}.$$

Then, for $k = 1, 2, \ldots$ compute:

$$x^{(k)} = Ay^{(k-1)}, \quad y^{(k)} = \frac{x^{(k)}}{\|x^{(k)}\|}, \quad \lambda^{(k)} = (y^{(k)})^T Ay^{(k)}.$$

We obtain for $k \geq 1$:

$$y^{(k)} = \alpha^{(k)} A^k y^{(0)}, \tag{2.2}$$

where

$$\alpha^{(k)} = \prod_{i=1}^{k} \|x^{(i)}\|^{-1}. \tag{2.3}$$

The name "power method" is justified by the presence of powers of A in the process.

The iteration stops when the following convergence criterion is satisfied:

$$|\lambda^{(k)} - \lambda^{(k-1)}| < \varepsilon |\lambda^{(k)}|.$$

2.2.1 Convergence Analysis

Since the eigenvectors $\{x_1, x_2, \ldots, x_n\}$ of A are linearly independent, they form a basis for $\mathbb{C}^n$. Hence, the initial vector can be written as:

$$x^{(0)} = \sum_{i=1}^{n} \beta_i x_i, \quad y^{(0)} = \frac{x^{(0)}}{\|x^{(0)}\|} = \alpha^{(0)} \sum_{i=1}^{n} \beta_i x_i.$$

Then,

$$x^{(1)} = Ay^{(0)} = \alpha^{(0)} \sum_{i=1}^{n} \beta_i \lambda_i x_i,$$

$$y^{(1)} = \alpha^{(1)} \sum_{i=1}^{n} \beta_i \lambda_i x_i,$$

and by induction:

$$y^{(k)} = \alpha^{(k)} \sum_{i=1}^{n} \beta_i \lambda_i^k x_i = \lambda_1^k \alpha^{(k)} \left(\beta_1 x_1 + \sum_{i=2}^{n} \beta_i \left(\frac{\lambda_i}{\lambda_1} \right)^k x_i \right).$$

Since $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$ for $i = 2, \ldots, n$, we conclude that:

$$\lim_{k \to \infty} y^{(k)} \propto x_1, \quad \text{if } \beta_1 \neq 0.$$

In practice, $\beta_1 \neq 0$ is not guaranteed since x_1 is unknown, but rounding errors generally introduce a component along x_1, which is beneficial here. The error in approximating x_1 by $y^{(k)}$ is proportional to:

$$\|y^{(k)} - x_1\| \sim \left| \frac{\lambda_2}{\lambda_1} \right|^k.$$

Therefore,

$$\lambda^{(k)} \to \lambda_1 \text{ as } k \to \infty.$$

In the case where A is symmetric, we can even show that:

$$|\lambda^{(k)} - \lambda_1| \leq C(A) \left| \frac{\lambda_2}{\lambda_1} \right|^{2k}, \tag{2.4}$$

where $C(A)$ is a constant depending on A.

Example

Consider:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Using the power method, the dominant eigenvalue is $\lambda_{\max} = 2 + \sqrt{2} \approx 3.4142$. We compare this with computed values to estimate the error $|\lambda^{(k)} - \lambda_{\max}|$.

MATLAB Code

```
A = [2 -1 0; -1 2 -1; 0 -1 2];
x0 = [1; 0; 0];
y = x0 / norm(x0);
val1 = y' * A * y;
eps = 1e-6;
n = 0;

for i = 1:20
    x0 = A * y;
    y = x0 / norm(x0);
    val = y' * A * y;
    if abs(val - val1) < eps * abs(val)
        fprintf('Eigenvalue found: %f\n', val);
        break;
    else
        val1 = val;
        n = n + 1;
    end
end

z = max(eig(A)); % For comparison
```

After 13 iterations, we obtain $\lambda \approx 3.4142$.

2.3 Inverse Power Iteration Method for Smallest magnitude Eigenvalue

This method applies the power method to A^{-1} (assuming A is non-singular). The eigenvalues of A^{-1} are the reciprocals of those of A:

$$\frac{1}{|\lambda_1|} < \frac{1}{|\lambda_2|} \leq \cdots \leq \frac{1}{|\lambda_n|}.$$

Hence, the dominant eigenvalue of A^{-1} corresponds to the smallest in magnitude eigenvalue λ_n of A. This leads to the inverse power method.

Algorithm

Given an initial vector $x^{(0)} \in \mathbb{C}^n$, define:

$$y^{(0)} = \frac{x^{(0)}}{\|x^{(0)}\|}.$$

Then for $k = 1, 2, \ldots$:

$$x^{(k)} = A^{-1}y^{(k-1)}, \quad y^{(k)} = \frac{x^{(k)}}{\|x^{(k)}\|}, \quad \mu^{(k)} = (y^{(k)})^T A^{-1} y^{(k)}.$$

Assuming $1/\lambda_n$ is dominant, then:

$$\lim_{k\to\infty} \mu^{(k)} = \frac{1}{\lambda_n}, \quad \text{so } \lambda_n = \left(\lim_{k\to\infty} \mu^{(k)}\right)^{-1}.$$

An error estimate similar to the power method holds:

$$\left|\mu^{(k)} - \frac{1}{\lambda_n}\right| \leq C(A) \left|\frac{\lambda_n}{\lambda_{n-1}}\right|^{2k}.$$

Example of Inverse Power Method (MATLAB Code)

We consider again the matrix:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

The following MATLAB code applies the inverse power method:

```
A = [2 -1 0; -1 2 -1; 0 -1 2];
x0 = [1; 0; 0];
no = norm(x0);
y = x0 / no;

val1 = y' * inv(A) * y;
eps = 1e-6;
n = 0;

for i = 1:20
    x0 = A \ y;          % Solving A*x0 = y instead of computing inv(A)
    no = norm(x0);
    y = x0 / no;
    val = y' * inv(A) * y;

    if abs(val - val1) < eps * abs(val)
        fprintf('Eigenvalue found\n');
        break;
    else
        val1 = val;
        n = n + 1;
    end

    eigenvalue = 1 / val1;
end
```

After $n = 6$ iterations, the inverse power method yields the smallest eigenvalue:

$$\lambda = 0.5858.$$

2.4 Householder Method

For any vector $u \in \mathbb{R}^n$, we define the matrix:

$$H = I - \frac{2uu^T}{\|u\|_2^2} = I - \frac{uu^T}{\phi}, \quad \text{with } \phi = \begin{cases} \frac{1}{2}\|u\|_2^2, & \text{if } u \neq 0 \\ 1, & \text{if } u = 0 \end{cases}$$

The matrix H is called the **Householder matrix**, and the vector u is the **Householder vector**. It is easy to verify that Householder matrices are:

- Symmetric: $H = H^T$
- Orthogonal: $H^T H = I$
- Dependent only on the direction of u

Example Let $u = (-1, -2)^T$, then:

$$\|u\|_2^2 = 5, \quad \phi = \frac{1}{2} \cdot 5 = \frac{5}{2}$$

$$H = I - \frac{uu^T}{\phi} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 2/5 & -4/5 \\ -4/5 & 8/5 \end{bmatrix} = \begin{bmatrix} 3/5 & 4/5 \\ 4/5 & -3/5 \end{bmatrix}$$

QR Factorization

Let $A = (a_{ij})$ be any real square matrix. The **QR factorization** consists in finding an orthogonal matrix Q and an upper triangular matrix R such that:

$$A = QR$$

Using Householder matrices, we build a sequence of $n - 1$ Householder matrices such that:

$$H_{n-1} \cdots H_2 H_1 A = R$$

We construct H_1 such that it transforms the first column a_1 of A into a multiple of e_1, i.e., zeroing all components $a_{21}, a_{31}, \ldots, a_{n1}$. We seek:

$$H_1 a_1 = \left(I - \frac{u_1 u_1^T}{\phi_1} \right) a_1 = \|a_1\|_2 e_1 = \begin{bmatrix} r_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad |r_{11}| = \|a_1\|_2$$

$$u_1 = a_1 - r_{11} e_1 = \begin{bmatrix} a_{11} - r_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}$$

To reduce numerical errors, the sign of r_{11} is chosen as $\text{sign}(r_{11}) = -\text{sign}(a_{11})$.

After applying H_1, we get:

$$A_2 = H_1 A = \begin{bmatrix} r_{11} & a_{12}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}$$

The second step uses H_2 that leaves the first row and column unchanged. So we take u_2 orthogonal to e_1, i.e., $u_2(1) = 0$. Define:

$$a = \begin{bmatrix} a_{12}^{(2)} \\ a_{22}^{(2)} \\ \vdots \\ a_{n2}^{(2)} \end{bmatrix}, \quad b = \begin{bmatrix} a_{12}^{(2)} \\ r_{22} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad u = a - b = \begin{bmatrix} 0 \\ a_{22}^{(2)} - r_{22} \\ a_{32}^{(2)} \\ \vdots \\ a_{n2}^{(2)} \end{bmatrix}$$

with $r_{22} = \sqrt{(a_{22}^{(2)})^2 + \cdots + (a_{n2}^{(2)})^2}$

After $n-1$ steps:

$$H_{n-1} \cdots H_1 A = R, \quad Q^T = H_{n-1} \cdots H_1, \quad \Rightarrow \quad A = QR$$

Example Let:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & -1 & 1 \\ 2 & 0 & 1 \end{bmatrix}$$

Step 1:

$$\|a_1\|_2 = \sqrt{1^2 + 2^2 + 2^2} = 3, \quad u_1 = \begin{bmatrix} 1+3 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 2 \end{bmatrix}, \quad \phi_1 = \frac{1}{2} \cdot (16 + 4 + 4) = 12$$

$$H_1 = I_3 - \frac{u_1 u_1^T}{\phi_1} = \begin{bmatrix} -1/3 & -2/3 & -2/3 \\ -2/3 & 2/3 & -1/3 \\ -2/3 & -1/3 & 2/3 \end{bmatrix} \Rightarrow H_1 A = \begin{bmatrix} -3 & -1/3 & -8/3 \\ 0 & -8/3 & -7/3 \\ 0 & -5/3 & -7/3 \end{bmatrix}$$

Step 2:

$$r_{22} = \sqrt{64/9 + 25/9} = \sqrt{89/9}, \quad u_2 = \begin{bmatrix} 0 \\ -8/3 - \sqrt{89/9} \\ -5/3 \end{bmatrix} \approx \begin{bmatrix} 0 \\ -5.81 \\ -1.66 \end{bmatrix}, \quad \phi_2 = \frac{1}{2} \cdot (5.81^2 + 1.66^2) \approx 18$$

$$H_2 H_1 = \left(I - \frac{u_2 u_2^T}{\phi_2} \right) H_1 = \begin{bmatrix} -0.33 & -0.66 & -0.66 \\ 0.91 & -0.38 & -0.07 \\ -0.21 & -0.63 & 0.74 \end{bmatrix} \Rightarrow R = H_2 H_1 A = \begin{bmatrix} -3 & -0.33 & -2.66 \\ 0 & 3.14 & 3.21 \\ 0 & 0 & -0.74 \end{bmatrix}$$

$$Q = H_1 H_2 = \begin{bmatrix} -0.33 & 0.91 & -0.21 \\ -0.66 & -0.38 & -0.63 \\ -0.66 & -0.07 & 0.74 \end{bmatrix}$$

QR Algorithm

Algorithm:

(a) Set $A^{(1)} = A$, $K = 1$

(b) While $\max_{i \neq j} |a_{ij}^{(K)}| > \epsilon$

i. Compute $Q^{(K)}, R^{(K)}$ such that $A^{(K)} = Q^{(K)} R^{(K)}$

ii. Set $A^{(K+1)} = R^{(K)} Q^{(K)}$

iii. Increment K

Principle: For any invertible real matrix A, there exists a unique QR decomposition:

$$A = QR, \quad Q \text{ orthogonal, } R \text{ upper triangular}$$

At each step:

$$A^{(k)} = R^{(k-1)} Q^{(k-1)}, \quad \text{then } A^{(k)} = Q^{(k)} R^{(k)}$$

For large K, $A^{(K)} \to$ upper triangular matrix with eigenvalues λ_i on the diagonal.

If A is symmetric, $A^{(K)}$ converges to a diagonal matrix.

Example Let:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & -1 & 1 \\ 2 & 0 & 1 \end{bmatrix}$$

MATLAB code:

```
A = [1 3 4; 2 -1 1; 2 0 1];
for i = 1:170
    [Q, R] = qr(A);
    A = R * Q;
end
valp = diag(A)
anti = eig(A)
```

At iteration $K = 32$, the matrix becomes:

$$A^{(32)} = \begin{bmatrix} 4.5955 & 1.9505 & -1.3869 \\ 0 & -3.1049 & -0.5215 \\ 0 & 0 & -0.4906 \end{bmatrix}$$

Thus, the eigenvalues of A are:

$$\lambda_1 = 4.5955, \quad \lambda_2 = -3.1049, \quad \lambda_3 = -0.4906$$

These match the values obtained via direct computation.

2.5 Eigenvector Computation Methods

Power Iteration Method

Algorithm

Choose random $\mathbf{x}_0$ with $\|\mathbf{x}_0\| = 1$ $k = 1, 2, \ldots$ until convergence $\mathbf{y}_k \leftarrow A\mathbf{x}_{k-1}$ $\mathbf{x}_k \leftarrow \mathbf{y}_k / \|\mathbf{y}_k\|_2$ $\lambda_k \leftarrow \mathbf{x}_k^\top A \mathbf{x}_k$ $\mathbf{x}_k$ (dominant eigenvector)

Example

For $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$:

- Iteration 1: $\mathbf{x}_1 = \begin{pmatrix} 0.8944 \\ 0.4472 \end{pmatrix}$
- Iteration 2: $\mathbf{x}_2 = \begin{pmatrix} 0.9487 \\ 0.3162 \end{pmatrix}$
- Converges to $\mathbf{v}_1 = \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$

Inverse Iteration Method

Algorithm

Choose random $\mathbf{x}_0$ with $\|\mathbf{x}_0\| = 1$ $k = 1, 2, \ldots$ Solve $A\mathbf{y}_k = \mathbf{x}_{k-1}$ $\mathbf{x}_k \leftarrow \mathbf{y}_k / \|\mathbf{y}_k\|_2$ $\mathbf{x}_k$ (eigenvector for smallest eigenvalue)

Example

Same matrix A:

- Converges to $\mathbf{v}_2 = \begin{pmatrix} 0.7071 \\ -0.7071 \end{pmatrix}$
- Corresponding eigenvalue: $\lambda_2 = 1$

QR Algorithm for All Eigenvectors

Procedure

(a) Compute Schur decomposition $A = QTQ^\top$

(b) For each λ_i (diagonal entries of T):

- Solve $(T - \lambda_i I)\mathbf{v}_i = \mathbf{0}$
- Compute $\mathbf{u}_i = Q\mathbf{v}_i$

Example

For $A = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$:

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

$$T = \begin{pmatrix} 3 & 0 \\ 0 & 5 \end{pmatrix}$$

$$\text{Eigenvectors} : \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Chapter 3

Numerical Resolution of First-Order Ordinary Differential Equations

3.1 Introduction

First-order ordinary differential equations (ODEs) play a crucial role in modeling dynamic systems across physics, engineering, and biology. While analytical solutions are preferable, many practical problems lack closed-form solutions, necessitating numerical approaches. This chapter explores key techniques for approximating solutions to first-order ODEs, including the Euler method, TaylorâĂŹs method, and Runge-Kutta methods, which transform continuous problems into discrete solvable forms. We examine the derivation, implementation, and error analysis of these methods, emphasizing their stability and convergence properties.

Let $I = [a, b] \subset \mathbb{R}$ and let f be a given function

$$f : I \times \mathbb{R} \to \mathbb{R}, \quad (t, y) \mapsto f(t, y).$$

Let y be a differentiable function from $\mathbb{R}$ to $\mathbb{R}$. A first-order differential equation is defined as:

$$\frac{dy(t)}{dt} = f(t, y(t)) \tag{3.1}$$

We say that y is a solution of this differential equation on $[a, b]$ if it satisfies relation (3.1) for all $t \in [a, b]$.

A Cauchy problem, or initial value problem, consists of the differential equation along with the initial condition $y(t_o) = y_o$, where y_o is a given number. The problem becomes:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_o) = y_o \end{cases} \tag{3.2}$$

Remark: Equation (3.2) is said to be of the first order because only the first derivative of $y(t)$ appears. If second-order derivatives were present, it would be a second-order equation, and so on.

Example: Solve the problem:

$$\begin{cases} y'(t) = t \\ y(0) = 1 \end{cases} \tag{3.3}$$

Solution: Integrating, we get:

$$y(t) = \frac{t^2}{2} + C.$$

Applying the initial condition $y(0) = 1$, we find $C = 1$, hence the particular solution:

$$y(t) = \frac{t^2}{2} + 1.$$

Example: Solve the problem:

$$\begin{cases} y'(t) = ty(t) \\ y(1) = 2 \end{cases} \tag{3.4}$$

Solution: We solve by separation of variables:

$$\frac{dy}{y} = t\,dt \Rightarrow \ln y(t) = \frac{t^2}{2} + C \Rightarrow y(t) = Ce^{t^2/2}.$$

Applying $y(1) = 2$, we find $C = 2e^{-1/2}$. The solution is:

$$y(t) = 2e^{(t^2-1)/2}.$$

Example: Solve the problem:

$$\begin{cases} y'(t) = 3\sqrt[3]{y(t)} \quad \text{for } t > 0 \\ y(0) = 0 \end{cases} \tag{3.5}$$

Solution: One can verify that the functions $y(t) = 0$ and $y(t) = \left(\frac{8t^3}{27}\right)$ are all solutions.

Remark: This example shows that the Cauchy problem (3.2) does not always have a unique solution.

Example: Solve the problem:

$$\begin{cases} y'(t) = y^3(t) \quad \text{for } t > 0 \\ y(0) = 1 \end{cases} \tag{3.6}$$

Solution: One verifies that for $t \in [0, 1/2]$,

$$y(t) = \frac{1}{\sqrt{1-2t}}.$$

Remark: The solution blows up as $t \to 1/2$, i.e.,

$$\lim_{t \to 1/2} y(t) = +\infty.$$

Remark: These examples show that the mathematical study of the existence and uniqueness of solutions to the Cauchy problem (3.2) can be delicate.

Let f be a function defined on $[a, b] \times \mathbb{R}$. If there exists a constant $L > 0$ such that:

$$|f(t, u) - f(t, v)| \leq L|u - v| \quad \forall u, v \in \mathbb{R}, \forall t \in [a, b],$$

then f is said to be Lipschitz continuous (or L-Lipschitz) with respect to y.

If f is continuous on $[a, b] \times \mathbb{R}$ and L-Lipschitz with respect to y, then for any initial condition $y_0 \in \mathbb{R}$, the Cauchy problem (3.2) admits a unique solution on $[a, b]$.

Example: Show that the following problem has a solution

$$\begin{cases} y'(t) = \sin(y(t)) + e^{-t^2/2} \\ y(0) = 1 \end{cases} \tag{3.7}$$

Solution: We verify:

$$|f(t, u) - f(t, v)| = |\sin u - \sin v| = \left| \int_v^u \cos \xi \, d\xi \right| \leq |u - v|,$$

so f is Lipschitz in y. By Theorem 3.6, the problem has a unique global solution.

Remark: In this example, even though there is a unique global solution, no closed-form expression for $y(t)$ can be given. Numerical methods are therefore necessary.

3.2 Euler's Method

Euler's method is the simplest numerical technique for solving ordinary differential equations. It has a nice geometric interpretation and is easy to implement, although it is rarely used due to its low accuracy.

EulerâĂŹs method provides approximations y_i of $y(t_i)$ for $i = 0, \ldots, N$.

3.2.1 Algorithm: Euler's Method

1. Given a step size h, an initial condition (t_0, y_0), and a maximum number of iterations N.

2. For $0 \leq i \leq N$:

$$y_{i+1} = y_i + hf(t_i, y_i),$$
$$t_{i+1} = t_i + h.$$

Record t_{i+1} and y_{i+1}.

3. Stop.

3.2.2 Geometric Interpretation

Consider again the problem (3.2) with the initial condition $y(t_0) = y_0$. We aim to approximate the solution at $t = t_1 = t_0 + h$.

From the differential equation:

$$y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0),$$

we follow the line through (t_0, y_0) with slope $f(t_0, y_0)$, whose equation is:

$$d_0(t) = y_0 + f(t_0, y_0)(t - t_0).$$

At t_1, we get:

$$d_0(t_1) = y_0 + hf(t_0, y_0) = y_1,$$

an approximation of $y(t_1)$. Generally, $y(t_1) \neq y_1$, and the error propagates in future iterations.

To compute y_2, we use the slope at (t_1, y_1), approximating:

$$y'(t_1) \approx f(t_1, y_1).$$

Then the tangent line is:

$$d_1(t) = y_1 + f(t_1, y_1)(t - t_1),$$

yielding:

$$y_2 \approx d_1(t_2) = y_1 + hf(t_1, y_1).$$

Remark: This development highlights an important feature of numerical methods for ODEs: the error introduced at each step propagates and affects future steps. In general, the error $|y(t_i) - y_i|$ increases slightly with i.

3.2.3 Definition

A differential equation solving method is called a one-step method if it takes the form

$$y_{n+1} = y_n + h\,\phi(t_n, y_n) \tag{3.8}$$

where ϕ is an arbitrary function. Such a relation is called a difference equation. The method is one-step if, to obtain the solution at $t = t_{n+1}$, one only needs the numerical solution at time t_n. Multistep methods, on the other hand, require the numerical solutions at times $t_{n-1}, t_{n-2}, t_{n-3}, \ldots$.

Euler's method is of course a one-step method, with:

$$\phi(t, y) = f(t, y)$$

3.2.4 Definition

The local truncation error at point $t = t_n$ is defined by

$$\tau_{n+1}(h) = \frac{y(t_{n+1}) - y(t_n)}{h} - \phi(t_n, y(t_n))$$

The local truncation error measures how well the analytical solution satisfies the difference equation (3.8).

3.2.5 Theorem

Assume the function $f(t, y)$ is continuous in both variables and Lipschitz continuous with respect to y, uniformly in t, and that $y \in C^2[a, b]$. Let $M_2 = \max_{t\in[a,b]} |y''(t)|$. Then we have the estimate:

$$|y_n - y(t_n)| \leq \left(e^{L(b-a)} - 1\right) \frac{M_2}{2L} h \tag{3.10}$$

3.2.6 Example

Consider the problem $y'(t) = y(t)+t$, with initial condition $y(0) = 1$. Approximate the solution at $t = 1$ using Euler's method by dividing the interval into 10 equal parts. Compare to the exact solution.

Solution: Note that the exact solution is $y(t) = 2e^t - t - 1$.

Euler's Method MATLAB Code:

```
clc
clear all
h = 0.1;
t0 = 0;
y0 = 1;
y = y0;
t = t0;
for i = 1 : 10
    yi = y + h * (y + t);
    t = t + h;
    fi = 2 * exp(t) - t - 1;
    ri = abs(yi - fi);
    y = yi;
end
y
```

The results are summarized in the following table:

i	y_i	$y(t_i)$	$\lvert y(t_i) - y_i \rvert$
0	1.00000000	1.00000000	0
1	1.10000000	1.11034184	0.01034184
2	1.22000000	1.24280552	0.02280552
3	1.36200000	1.39971762	0.03771762
4	1.52820000	1.58364940	0.05544940
5	1.72102000	1.79744254	0.07642254
6	1.94312200	2.04423760	0.10111560
7	2.19743420	2.32750541	0.13007121
8	2.48717762	2.65108186	0.16390424
9	2.81589538	3.01920622	0.20331084
10	3.18748492	3.43656366	0.24907874

3.2.7 Definition

Any numerical method approximating $y(t_n)$ by y_n such that the error satisfies:

$$|y(t_n) - y_n| \leq kh^p \quad \text{with } k > 0$$

is said to be of order p.

Remark: From equation (3.10), EulerâĂŹs method is of order 1 because $|y(t_n) - y_n| \leq kh$.

Note: Do not confuse the order of a differential equation with the order of a numerical method used to solve it.

3.3 Taylor Method of Order 2

Taylor expansion allows for a generalization of Euler's method. Instead of approximating the curve $y(t)$ on $[t_i, t_{i+1}]$ by a straight line, it is now approximated by a parabola.

3.3.1 Algorithm: Taylor Method of Order 2

(a) Given a step h, initial condition (t_0, y_0), and a maximum number of iterations N.

(b) For $0 \leq i \leq N$:

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + \frac{\partial f}{\partial y}(t_i, y_i) f(t_i, y_i)\right)$$

$$t_{i+1} = t_i + h$$

(c) Stop.

Remark: In this algorithm, errors also propagate from one iteration to the next.

3.3.2 Theorem

Assume $f(t,y) \in C^2([a,b] \times \mathbb{R})$, f is L_0-Lipschitz, and

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f$$

is L_1-Lipschitz with respect to y, uniformly in t. Let $M_3 = \max_{t \in [a,b]} |y^{(3)}(t)|$. Then,

$$|y_n - y(t_n)| \leq \left(e^{L_0 + \frac{(b-a)}{2} L_1} (b-a) - 1 \right) \frac{M_3}{6L_0} h^2 \tag{3.11}$$

Remark: This confirms that the Taylor method is of order 2.

3.3.3 Example

Use again the equation $y'(t) = y(t) + t$ with initial condition $y(0) = 1$. Compare the result obtained using the Taylor method to that obtained with Euler's method.

The results are:

i	y_i	$y(t_i)$	$\lvert y(t_i) - y_i \rvert$
0	1.00000000	1.00000000	0
1	1.10000000	1.11034184	3.42×10^{-4}
2	1.24205000	1.24280552	7.56×10^{-4}
3	1.39846525	1.39971762	0.00125237
4	1.58180410	1.58364940	0.00184529
5	1.79489353	1.79744254	0.00254901
6	2.04085735	2.04423760	0.00338025
7	2.32314737	2.32750541	0.00435804
8	2.64557785	2.65108186	0.00550401
9	3.01236352	3.01920622	0.00684270
10	3.42816169	3.43656366	0.00840196

Remark

It is possible to obtain even more accurate Taylor methods by continuing the Taylor expansion to higher-order terms. This requires evaluating increasingly high-order derivatives of the function $f(t, y(t))$, which involves additional computation of partial derivatives of f:

$$\frac{\partial^2 f}{\partial t^2}, \quad \frac{\partial^2 f}{\partial y^2}, \quad \frac{\partial^2 f}{\partial t \partial y}, \ldots, \quad \frac{\partial^{i+j} f}{\partial t^i \partial y^j}$$

For this reason, such methods are difficult to use. However, it is possible to bypass this difficulty using so-called Runge-Kutta methods.

3.4 Runge-Kutta Methods

3.4.1 Second-Order Runge-Kutta Methods

Recall the second-order Taylor expansion:

$$y(t_{n+1}) = y(t_n)+hf(t_n, y(t_n))+\frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right)+O(h^2) \tag{3.4.1}$$

We seek an equivalent expression of the same order $O(h^2)$, without partial derivatives:

$$y(t_{n+1}) = y(t_n) + a_1 hf(t_n, y(t_n)) + a_2 hf(t_n + a_3 h, y(t_n) + a_4 h)$$

Using a Taylor expansion of $f(t_n + a_3 h, y(t_n) + a_4 h)$, we get:

$$f(t_n + a_3 h, y(t_n) + a_4 h) = f(t_n, y(t_n)) + a_3 h\frac{\partial f}{\partial t} + a_4 h\frac{\partial f}{\partial y} + O(h^2)$$

Substituting back:

$$y(t_{n+1}) = y(t_n) + (a_1 + a_2)hf(t_n, y(t_n)) + a_2 a_3 h^2\frac{\partial f}{\partial t} + a_2 a_4 h^2\frac{\partial f}{\partial y} + O(h^2)$$

Comparing with (3.4.1), we obtain the system:

$$\begin{cases} a_1 + a_2 = 1 \\ a_2 a_3 = \frac{1}{2} \\ a_2 a_4 = \frac{1}{2} \end{cases}$$

This system has more unknowns than equations, allowing multiple variants.

Modified Euler Method: Choosing $a_1 = a_2 = \frac{1}{2},\ a_3 = 1,\ a_4 = f(t_n, y(t_n))$, which satisfies the system:

Algorithm 1 Modified Euler Method

(a) Given step size h, initial condition (t_0, y_0), and max iterations N

(b) For $0 \leq i \leq N$

- $\widehat{y} = y_n + hf(t_n, y_n)$
- $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, \widehat{y}))$
- $t_{n+1} = t_n + h$

(c) Output t_{n+1}, y_{n+1}

Example Solve $y'(t) = y(t) + t$, with $y(0) = 1$, using step $h = 0.1$. Two iterations yield:

$$\widehat{y} = 1 + 0.1(1 + 0) = 1.1$$

$$y_1 = 1 + 0.05((1 + 0) + (1.1 + 0.1)) = 1.11$$

$$\widehat{y} = 1.11 + 0.1(1.11 + 0.1) = 1.231$$

$$y_2 = 1.11 + 0.05((1.11 + 0.1) + (1.231 + 0.2)) = 1.24205$$

Midpoint Method Variant: Choosing $a_1 = 0, a_2 = 1, a_3 = \frac{1}{2}, a_4 = \frac{1}{2}f(t_n, y_n)$:

Algorithm 2 Midpoint Method

(a) Given step h, initial condition (t_0, y_0), and iterations N

(b) For $0 \leq i \leq N$

- $k_1 = hf(t_n, y_n)$
- $y_{n+1} = y_n + hf(t_n + h/2, y_n + k_1/2)$
- $t_{n+1} = t_n + h$

(c) Output t_{n+1}, y_{n+1}

3.4.2 Fourth-Order Runge-Kutta Method

Extending the Taylor expansion to order 5 leads to a system with 8 equations and 10 unknowns. The classic Runge-Kutta method of order 4 results:

Algorithm 3 Runge-Kutta Method of Order 4

(a) Given step h, initial condition (t_0, y_0), and iterations N

(b) For $0 \leq i \leq N$

- $k_1 = hf(t_n, y_n)$
- $k_2 = hf(t_n + h/2, y_n + k_1/2)$
- $k_3 = hf(t_n + h/2, y_n + k_2/2)$
- $k_4 = hf(t_n + h, y_n + k_3)$
- $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
- $t_{n+1} = t_n + h$

(c) Output t_{n+1}, y_{n+1}

Example For $y'(t) = y(t) + t$, with $y(0) = 1$, run 10 iterations and compare with the exact solution $y(t) = 2e^t - t - 1$. The absolute error is shown in the table:

t_i	y_i (RK4)	Exact $y(t_i)$	$\lvert y(t_i) - y_i \rvert$
0.0	1.00000000	1.00000000	0
0.1	1.11034167	1.11034184	1.69×10^{-7}
0.2	1.24280514	1.24280552	3.75×10^{-7}
0.3	1.39971699	1.39971762	6.21×10^{-7}
0.4	1.58364848	1.58364940	9.15×10^{-7}
0.5	1.79744128	1.79744254	1.26×10^{-6}
0.6	2.04423592	2.04423760	1.68×10^{-6}
0.7	2.32750325	2.32750541	2.16×10^{-6}
0.8	2.65107913	2.65108186	2.73×10^{-6}
0.9	3.01920283	3.01920622	3.39×10^{-6}
1.0	3.43655949	3.43656370	4.17×10^{-6}

3.5 Solved Exercices

Exercise 1

Consider the problem:

$$\begin{cases} y'(t) = -y(t) + t + 1 \\ y(0) = 1 \end{cases}$$

Using the Euler method with step size $h = 0.1$, compute the successive approximations $y(0.1), y(0.2), y(0.3), \ldots, y(1.0)$, and compare the numerical solution with the exact one. What happens to the error?

Solution:

We begin by finding the analytical solution of the equation:

$$y'(t) = -y(t) + t + 1$$

The general solution is given by:

$$y(t) = y_h(t) + y_p(t)$$

where $y_h(t)$ is the general solution of the homogeneous equation $y'(t) + y(t) = 0$, and $y_p(t)$ is a particular solution of the full equation.

It is easy to see that:

$$y_h(t) = Ce^{-t}, \quad y_p(t) = t$$

Thus, the exact solution is:

$$y(t) = e^{-t} + t$$

Now applying the Euler method:

$$y_{i+1} = y_i + hf(t_i, y_i), \quad y_0 = y(t_0) = 1$$

with $h = 0.1$ and $f(t_i, y_i) = -y + t + 1$, we compute the approximations.

We get the following table:

t_i	Exact $y(t_i)$	Euler y_i	$\lvert y(t_i) - y_i \rvert$
0.0	1.0000	1.0000	0.0000
0.1	1.0048	1.0000	0.0084
0.2	1.0187	1.0100	0.0087
0.3	1.0408	1.0290	0.0118
0.4	1.0703	1.0561	0.0142
0.5	1.1065	1.0905	0.0160
0.6	1.1488	1.1315	0.0173
0.7	1.1966	1.1784	0.0182
0.8	1.2493	1.2406	0.0087
0.9	1.3066	1.2965	0.0101
1.0	1.3679	1.3568	0.0111

We observe that increasing the number of points influences the accumulation of error.

Exercise 2

Consider the problem:

$$\begin{cases} y'(t) = ty(t) \\ y(1) = 2 \end{cases}$$

(a) Compute the explicit solution.

(b) Approximate $y(2)$ using the second-order Taylor method with step sizes $h = 0.5, 0.25, 0.125$.

Solution:

1. The equation $y' = ty$ is separable. The exact solution is:

$$y(t) = 2e^{\frac{t^2-1}{2}}$$

2. Recall the second-order Taylor approximation:

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + \frac{\partial f}{\partial y}(t_i, y_i)f(t_i, y_i)\right)$$

Given $f(t, y) = ty$, we compute:

$$\frac{\partial f}{\partial t} = y, \quad \frac{\partial f}{\partial y} = t$$

$$\Rightarrow y_{i+1} = y_i + ht_iy_i + \frac{h^2}{2}y_i(1 + 2t_i^2)$$

Implementing this in MATLAB gives: - For $h = 0.5$: $y(2) \approx 9.140625$ - For $h = 0.25$: $y(2) \approx 9.948075$ - For $h = 0.125$: $y(2) \approx 9.821823$

Exact value: $y(2) = 2e^{\frac{3}{2}} \approx 8.963378$

Errors:

$$\varepsilon(0.5) = |9.140625 - 8.963378| = 0.177247$$

$$\varepsilon(0.25) = |9.948075 - 8.963378| = 0.984697$$

$$\varepsilon(0.125) = |9.821823 - 8.963378| = 0.858445$$

Exercise 3

Consider the differential equation:

$$\begin{cases} y'(t) = y(t) - \dfrac{2t}{y(t)} \\ y(0) = 1 \end{cases}$$

We want to approximate the solution at $t = 0.4$.

(a) Use the Runge-Kutta method of order 2 with $h = 0.2$.

(b) Use the Runge-Kutta method of order 4 with $h = 0.2$.

(c) Verify that $y(t) = \sqrt{2t+1}$ is the exact solution and deduce the exact value at $t = 0.4$.

Results: - RK2: $y = [1.000000, 1.186667, 1.348312]$ - RK4: $y = [1.000000, 1.183229, 1.341667]$ - Exact: $y(0.4) = \sqrt{2(0.4)+1} = \sqrt{1.8} \approx 1.341641$

Exercise 4

Consider the differential equation:

$$y'(t) = -2ty(t), \quad y(0) = 1$$

(a) Find the exact solution.

(b) Compute $y(0.1), y(0.2), y(0.3), y(0.4), y(0.5)$ using Runge-Kutta method of order 4 with $h = 0.1$, and compare to the exact solution.

Solution: The equation is separable:

$$\frac{dy}{y} = -2tdt \Rightarrow \ln y = -t^2 + C \Rightarrow y(t) = e^{-t^2}$$

Then apply the RK4 method using the code from previous exercises with $f(t, y) = -2ty$ and $h = 0.1$, we obtain the following results table:

t_i	$y(t_i)$	y_i	$\lvert y(t_i) - y_i \rvert$
0.0	1.000000000	1.000000000	≈ 0
0.1	0.990049833	0.990049834	$1.0 \times 10^{-7} \approx 0.004158$
0.2	0.960789435	0.960789439	$1.0 \times 10^{-7} \approx 0.039167$
0.3	0.913931174	0.913931185	$1.0 \times 10^{-7} \approx 0.112526$
0.4	0.852143772	0.852143789	$1.0 \times 10^{-7} \approx 0.164987$
0.5	0.778800781	0.778800783	$1.0 \times 10^{-7} \approx 0.025277$

Exercise 5

Consider the Cauchy problem:

$$\begin{cases} y'(t) = t^2 + y(t), & t \in (0, 1] \\ y(0) = 1 \end{cases}$$

(a) Find the exact solution.

(b) Apply Euler's method with step size $h = 0.1$ to evaluate the solution at $t = 0.3$, then compare it to the exact solution.

Exercise 6

Consider the Cauchy problem:

$$\begin{cases} y'(t) = 2t - y(t), & t \in (0, 1] \\ y(0) = 1 \end{cases}$$

(a) Give the analytical solution.

(b) Apply Euler's method with step size $h = 0.1$, and give the numerical solution at $t = 0.3$, accurate to 10^{-3}.

(c) Provide the theoretical error of Euler's method and compare it to the actual error. Comment on the result.

Solution:

- The equation is a first-order linear differential equation. The general solution is:
$$y(t) = 3e^{-t} + 2t - 2$$
- Using Euler's method:
$$y(0.1) \approx 0.9$$
$$y(0.2) \approx 0.83$$
$$y(0.3) \approx 0.787$$
- Theoretical error:
$$|y_n - y(t_n)| \leq \frac{(e^{L(b-a)} - 1)}{2L} M_2 h$$
where L satisfies:
$$|f(t, u) - f(t, v)| \leq L|u - v|, \quad \forall u, v \in \mathbb{R},\ t \in [a, b]$$

Exercise 7

Consider the Cauchy problem:

$$\begin{cases} y'(t) = -y(t) + e^{-t} + e^{t}, & t \in (0, 1] \\ y(0) = \frac{1}{2} \end{cases}$$

(a) Show that the exact solution is $y(t) = \frac{1}{2}e^{t} + te^{-t}$.

(b) Apply Euler's method with step size $h = 0.1$, then compute the numerical solution at $t = 0.4$ with accuracy up to 10^{-3}.

(c) Apply the modified Euler method with step size $h = 0.2$, then compute the numerical solution at $t = 0.4$ with accuracy up to 10^{-3}.

(d) Compare both numerical solutions to the exact value $y(0.4) \approx 1.0140$, and comment.

Result:

- Euler's method gives $y(0.4) \approx 1.0323$, error $\varepsilon_E = 0.0183$
- Modified Euler gives $y(0.4) \approx 1.0141$, error $\varepsilon_{EM} = 0.0001$

Exercise 8

Consider the Cauchy problem:

$$\begin{cases} y'(t) = -2ty(t) + e^{t-t^2}, & t \in [0, 1] \\ y(0) = 1 \end{cases}$$

(a) Verify that the exact solution is $y(t) = e^{t-t^2}$.

(b) Using step size $h = 0.2$, approximate $y(0.2)$ using Euler's method and compute the absolute error ε_E.

(c) With the same step size $h = 0.2$, compute an approximation of $y(0.2)$ using the Runge-Kutta method of order 4, and compute the error ε_R.

(d) Compare the errors and comment.

Result:

$$\begin{aligned} \text{Exact: } & y(0.2) = e^{0.2-0.04} \approx 1.17350 \\ \text{Euler: } & y_E = 1.2 \Rightarrow \varepsilon_E = |1.2 - 1.17350| = 0.02649 \\ \text{RK4: } & y_R = 1.17350 \Rightarrow \varepsilon_R = 0.00001 \end{aligned}$$

Hence, the Runge-Kutta algorithm is more accurate than Euler's method.

Exercise 9

Consider the Cauchy problem:

$$\begin{cases} y'(t) = y(t) + e^{2t}, \\ y(0) = 2 \end{cases}$$

The analytical solution is $y(t) = e^t + e^{2t}$.

1. Modified Euler's Method:

Given $h = 0.05$, perform 2 iterations of the **modified Euler's method** and compute the error in y_2 by comparing the result with the analytical solution $y(0.1)$.

2. Runge-Kutta 4th-Order Method:

Given $h = 0.1$, perform 1 iteration of the **Runge-Kutta 4th-order method** and compute the error in y_1 by comparing the result with the analytical solution $y(0.1)$.

3. Results Analysis:

Comment on the results obtained in parts 1 and 2, comparing their accuracy and computational efficiency.

Solution

1. Modified Euler Method

We want to compute an approximation of $y(0.1)$ using 2 iterations with step size $h = 0.05$, initial value $y_0 = 2$, and the differential equation:

$$f(t, y) = y + e^{2t}$$

Algorithm Reminder:

$$\hat{y} = y_n + hf(t_n, y_n)$$
$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, \hat{y})\right)$$

Step 1 - Compute y_1:

$$\hat{y} = y_0 + 0.05 \cdot (2 + e^0) = 2 + 0.05 \cdot 3 = 2.15$$
$$f(0.05, \hat{y}) = 2.15 + e^{0.1} \approx 2.15 + 1.105171 = 3.255171$$
$$y_1 = 2 + \frac{0.05}{2}(3 + 3.255171) = 2 + 0.025 \cdot 6.255171 \approx 2.156379$$

Step 2 - Compute y_2:

$$\hat{y} = y_1 + 0.05 \cdot (2.156379 + e^{0.1}) = 2.156379 + 0.05 \cdot 3.26155 \approx 2.319457$$
$$f(0.1, \hat{y}) = 2.319457 + e^{0.2} \approx 2.319457 + 1.221403 = 3.540860$$
$$y_2 = 2.156379 + \frac{0.05}{2}(3.26155 + 3.540860) = 2.156379 + 0.025 \cdot 6.80241 \approx 2.326439$$

Thus, the numerical value is $y_2 \approx 2.326439$.
Analytical solution: $y(0.1) = e^{0.1} + e^{0.2} \approx 1.105171 + 1.221403 = 2.326574$.
Error: $|y_{\text{EM}} - y(0.1)| = |2.326439 - 2.326574| = 0.000135$

2. Runge-Kutta Method (Order 4)

One iteration with $h = 0.1$, $y_0 = 2$

Algorithm Reminder:

$$k_1 = hf(t_0, y_0) = 0.1(2 + e^0) = 0.3$$
$$k_2 = hf(t_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}) = 0.1(2.15 + e^{0.1}) = 0.325517$$
$$k_3 = hf(t_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}) = 0.1(2.162758 + e^{0.1}) = 0.326793$$
$$k_4 = hf(t_0 + h, y_0 + k_3) = 0.1(2.326793 + e^{0.2}) = 0.354820$$
$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$= 2 + \frac{1}{6}(0.3 + 2 \cdot 0.325517 + 2 \cdot 0.326793 + 0.354820)$$
$$= 2 + \frac{1.95944}{6} = 2.326573$$

Error: $|y_{\text{RK4}} - y(0.1)| = |2.326573 - 2.326574| = 0.000001$

3. Conclusion

Although the RK4 method used a larger step size $h = 0.1$, it yielded a more accurate result than the modified Euler method with $h = 0.05$. This confirms the superior precision of RK4 over the modified Euler method in solving this type of differential equation.

Chapter 4

Nonlinear Algebraic Systems Resolution

4.1 Introduction

Nonlinear systems of algebraic equations appear in various scientific disciplines. These equations do not generally have closed-form solutions, requiring iterative numerical methods. We study the solution of systems:

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{x} \in \mathbb{R}^n$$

where $\mathbf{F} = (f_1, f_2, \ldots, f_n)^T$ is a vector-valued function. We present fixed-point methods, Newton's method, convergence theory, and several solved problems.

4.2 Fixed-Point Iteration Method

4.2.1 Theory and Derivation

If we can write the system as $\mathbf{x} = \mathbf{G}(\mathbf{x})$, then we use the iteration:

$$\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$$

Convergence is guaranteed under the contraction condition: $\|\mathbf{G}'(\mathbf{x})\| < 1$.

4.2.2 Example 1

$$\begin{cases} x = \cos(y) \\ y = \sin(x) \end{cases} \Rightarrow \mathbf{G}(x, y) = [\cos(y), \sin(x)]$$

Start with $(x^{(0)}, y^{(0)}) = (0.5, 0.5)$. Iterate:

$$(x^{(1)}, y^{(1)}) = (\cos(0.5), \sin(0.5)) \approx (0.8776, 0.4794)$$

Continue until $\|x^{(k+1)} - x^{(k)}\| < \epsilon$.

4.3 Newton's Method for Nonlinear Systems

4.3.1 Theory

For $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, NewtonÃćâĆňâĎćs update rule is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J^{-1}(\mathbf{x}^{(k)})\mathbf{F}(\mathbf{x}^{(k)})$$

4.3.2 Jacobian Matrix

The Jacobian matrix $J(\mathbf{x})$ is defined as:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

4.3.3 Example 2

Solve:

$$\begin{cases} x^2 + y^2 - 1 = 0 \\ x - y = 0 \end{cases}$$

Define:

$$\mathbf{F}(x, y) = \begin{bmatrix} x^2 + y^2 - 1 \\ x - y \end{bmatrix}, \quad J(x, y) = \begin{bmatrix} 2x & 2y \\ 1 & -1 \end{bmatrix}$$

Starting from $(x_0, y_0) = (0.7, 0.7)$, compute:

$$\mathbf{F}(0.7, 0.7) = \begin{bmatrix} 0.98 - 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.02 \\ 0 \end{bmatrix}$$

$$J(0.7, 0.7) = \begin{bmatrix} 1.4 & 1.4 \\ 1 & -1 \end{bmatrix}$$

Solve $J\Delta x = -\mathbf{F}$ to update.

4.4 Convergence Analysis

4.4.1 Fixed-Point

Let $\mathbf{G}$ be Lipschitz continuous with constant $L < 1$. Then the iteration converges to the fixed point.

4.4.2 Newton Method

If $\mathbf{F}$ is C^2 and $J(\mathbf{x}^*)$ nonsingular at the root, then convergence is quadratic:

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq C\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2$$

4.5 Solved Exercises

Exercise 1: System with square roots

$$\begin{cases} x = \sqrt{3-y} \\ y = \sqrt{3-x} \end{cases}$$

Use $\mathbf{G}(x, y) = [\sqrt{3-y}, \sqrt{3-x}]$. Starting from $(1, 1)$:

$$(x_1, y_1) = (\sqrt{2}, \sqrt{2}) \approx (1.4142, 1.4142)$$

Continue until convergence.

Exercise 2: Nonlinear System - Newton's Method

$$\begin{cases} x^2 + y^2 - 4 = 0 \\ e^x + y - 1 = 0 \end{cases}$$

Define $\mathbf{F}(x, y) = [x^2 + y^2 - 4, e^x + y - 1]$

Jacobian:

$$J(x, y) = \begin{bmatrix} 2x & 2y \\ e^x & 1 \end{bmatrix}$$

Start at $(1, 1)$, compute:

$$\mathbf{F}(1, 1) = [1^2 + 1^2 - 4, e^1 + 1 - 1] = [-2, e]$$

$$J(1, 1) = \begin{bmatrix} 2 & 2 \\ e & 1 \end{bmatrix}$$

Solve:

$$J\Delta x = -\mathbf{F} \Rightarrow \Delta x = \begin{bmatrix} \delta x \\ \delta y \end{bmatrix}, \quad x_{new} = x + \delta x$$

Repeat until convergence.

Exercise 3: System in 3D

$$\begin{cases} x + y + z = 1 \\ x^2 + y^2 + z^2 = 3 \\ xyz = 1 \end{cases}$$

Use Newton method. Let:

$$\mathbf{F}(x, y, z) = \begin{bmatrix} x + y + z - 1 \\ x^2 + y^2 + z^2 - 3 \\ xyz - 1 \end{bmatrix}$$

Jacobian:

$$J = \begin{bmatrix} 1 & 1 & 1 \\ 2x & 2y & 2z \\ yz & xz & xy \end{bmatrix}$$

Choose initial guess $(1, 0, 0)$ (not valid), modify to $(1, 1, 1)$, proceed with iterations.

4.6 Conclusion

We discussed numerical approaches to nonlinear systems using fixed-point and Newton's methods. Newton's method is robust and efficient but requires Jacobian calculation. Fixed-point methods are simpler but slower.

General Conclusion

This Numerical Analysis 2 course provided second-year mathematics students with essential tools for solving applied and theoretical problems.
It was structured into four main chapters, each focused on a specific class of numerical problems.
Chapter one dealt with linear systems, introducing direct methods (Gaussian elimination, LU decomposition) and iterative methods (Jacobi, Gauss-Seidel).
These methods are foundational for applications in engineering, physics, and statistics.
Chapter two focused on eigenvalue and eigenvector computation, crucial for stability analysis, quantum mechanics, and data science.
We covered the power method, inverse iteration, and the QR algorithm.
Chapter three addressed the numerical solution of first-order ordinary differential equations (ODEs).
Euler's method, Runge-Kutta methods, and predictor-corrector schemes were presented.
We emphasized analysis of stability, consistency, and convergence of these methods.
Chapter four explored nonlinear algebraic systems.
We studied fixed-point iteration and Newton's method, including Jacobian computation and convergence behavior.
The course emphasized not only algorithm implementation but also their mathematical analysis.
Each method was illustrated with examples and corrected exercises.
Students developed both theoretical understanding and computational skills.
By the end of the course, they can solve complex problems not solvable analytically.
They are also able to implement algorithms in software and evaluate their efficiency.
The course bridges the gap between mathematical theory and computational practice.
It empowers students with critical tools for academic and professional success.
Numerical methods taught here have broad scientific and engineering applications.
This course fosters confidence and competence in tackling advanced numerical problems.

Bibliography

[1] K. Atkinson and W. Han, *Elementary Numerical Analysis*, 3rd edition, Wiley, 2003.

[2] C. H. Bruneau, *Introduction a l'analyse numerique*, Presses Universitaires de Bordeaux, 2012.

[3] R. L. Burden and J. D. Faires, *Numerical Analysis*, 10th edition, Brooks/Cole, Cengage Learning, 2015.

[4] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 7th edition, McGraw-Hill, 2015.

[5] G. Cohen, *Analyse numÃlrique*, Dunod, Paris, 2002.

[6] J. J. Damelincourt, *Cours d'analyse numerique avec applications*, Dunod, Paris, 2001.

[7] T. Gallouet, R. Herbin, *Analyse numerique*, Editions Ellipses, 1996.

[8] B. Jourdain, *Methodes numeriques pour les equations differentielles*, Editions de l'Ecole Polytechnique, 2010.

[9] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, Springer, 2007.

[10] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.