Ministry of Higher Education and Scientific Research
Mohamed-Boudiaf University of Science and Technology of Oran- USTOMB

Faculty of Mathematics and Computer Science
Department of Computer Science

Numerical Methods Course

Dr. Asmaa Ourdighi.

Online Course for Students via *Moodle*
Octobre 2024

Numerical Methods Course

Aim of the course

Numerical analysis is fundamentally intertwined with computer science. The advent of high-performance computing has revolutionized the field, enabling the solution of complex mathematical problems that were previously intractable. Numerical methods are at the heart of computational software, from scientific simulations to data analysis tools. Computer scientists design efficient algorithms and data structures to implement these methods, while numerical analysts develop the mathematical foundations. This interdisciplinary collaboration has led to groundbreaking advancements in fields such as artificial intelligence, machine learning, and data science, where numerical techniques are essential for tasks like training neural networks, processing large datasets, and solving optimization problems.

Content of the Material:

Chapter 1 Basics of Numerical Analysis and Scientific Computing

1.1.Motivations

> **Notation:** "*Numerical analysis is the study of algorithms for the problems of continuous mathematics.*" — Lloyd N. Trefethen

In the realm of mathematics, we frequently encounter continuous problems. However, computers are limited to discrete representations. For example, computers can only approximate irrational numbers like π or $\sqrt{2}$. Additionally, they use approximations for basic mathematical functions such as sine, cosine, etc. Numerical analysis bridges this gap, offering a rigorous framework for translating continuous mathematical problems into discrete problems that computer can handle. It is at the heart of many scientific and technological advances. Numerical analysis and computer science are inextricably linked. Advances in computing have empowered numerical methods to tackle increasingly complex problems. Computer scientists design efficient algorithms to implement these methods, while mathematicians provide the theoretical underpinnings. This synergistic relationship is indispensable in fields like artificial intelligence and data science, where numerical techniques are ubiquitous.

1.2.Floating-Point Arithmetic and Rounding Errors

Floating-point arithmetic is a numerical representation used in computers to approximate real numbers. Given that computers operate with finite memory, they cannot represent most real numbers exactly. Instead, they represent them using a finite number of bits, which leads to the following key concepts:: Representation of Floating-Point Numbers and Rounding Errors

1.2.1 Representation of Numbers in Machine

1.2.1.1 Introduction

In this section, we will introduce the concepts of mantissa, exponent, and how numbers are represented on a calculator or computer.

Base 10 is the natural base we work with and the one found in calculators. A decimal number, or decimal, has several different representations by simply changing the position of the decimal point and adding a power of 10 at the end of the number's representation. The part to the left of the decimal point is the integer part, and the part to the right before the exponent is called the mantissa. For example, the number x = 1234.5678 has several representations:

$$x = 1234{,}5678 = 1234{,}5678\,.\,10^{0} = 1{,}2345678.\,10^{3} = 0{,}0012345678.\,10^{6} \quad (1)$$

In each representation in Eq. (1):

- The *"mantissa"* is the significant part of the number, located to the left of the power of 10. In our example, it can be 1234,5678, 1,2345678, or 0,0012345678.
- The *"exponent"* is the power of 10 that shifts the decimal point: 0, 3, or 6.

Why this representation?

It offers great flexibility to represent very large or very small numbers, while using a fixed number of digits.

1.2.1.2 Representation of a Number in Machine: Floating-Point Numbers

The binary system is the foundation of computer arithmetic. In this system, numbers are represented using only two digits: 0 and 1.

Let's take the decimal number 39 and 3,625. In binary, it is represented as 100111 and 11.101, respectively in Eq. (2) and Eq. (3).

$$39 = 32 + 4 + 2 + 1 = 2^5 + 2^2 + 2^1 + 2^0 = (100111)_2 \quad (2)$$

$$3{,}625 = 2^1 + 2^0 + 2^{-1} + 2^{-3} = (11.101)_2 = (1.1101)_2 \quad (3)$$

In general, any real number *x* can be represented in a base *b* (*b = 10 for a calculator, b = 2 for a computer*) by: its sign "+ or –", the "mantissa *m"* (*also called significand*), the "base *b"*, and an "exponent *e"*. The mantissa is usually normalized to have a leading non-zero digit. By varying e, the decimal point is made to '*float*'. However, due to the finite nature of computer memory, only a finite subset of real numbers can be exactly represented. Consequently, a machine-level real number or floating-point number is subject to rounding errors, which can lead to loss of precision in certain calculations.

$$\check{X} = \pm m.b^e$$
$$m = D, D \ldots . D \quad and\ e = D \ldots D,\ \ where\ D \in \{0,1, \ldots, b-1\} \quad (4)$$

Approximate representations of π are: (0.031,2), (3.142,0), (0.003,3). It can be observed that these representations do not yield the same level of precision. To ensure uniqueness and optimal precision, a normalized mantissa is employed, where the leading digit before the radix point is non-zero. Normalized machine numbers adhere to this convention. In base 2, the initial bit of the mantissa is invariably 1, and thus, it is omitted to save a bit. The exponent is constrained to a finite range, L <= e <= U (typically L < 0 and U > 0). Consequently, the system is defined by four integral parameters: the base b (usually 2), the precision t (number of digits in the mantissa), and the minimum and maximum exponents, L and U, respectively.

Example: Let's take the number π (Pi) and represent it in base 2 with a 5-bit mantissa and an exponent ranging from -2 to 2.
Choice of an approximation for π: We will use the approximation 3.1416.
Normalization of the mantissa: To normalize, we shift the decimal point so that there is a 1 before the decimal. Thus, 3.1416 becomes 1.1001×2^1.
Binary representation:
The mantissa (without the leading 1) is: 1001
The exponent is: 1 (which corresponds to 2^1 in our example)
Therefore, the floating-point representation of π in our system is $(1.1001,1)$.
To summarize, the number π is represented by:
Sign: + (positive)
Mantissa: 1.1001 (normalized)
Exponent: 1
Base: 2

Mathematical computations involve real numbers z drawn from a continuous interval $\in]-\infty,+\infty[$. However, the finite precision of computers necessitates approximations for most real numbers. For instance, $\frac{1}{3}, \sqrt{2}, \; and \; \pi$, with its infinite decimal expansion, cannot be exactly represented in a machine. Even the simplest calculations then become approximate. Practical experience shows that this limited set of representable numbers is largely sufficient for calculations on a computer; the numbers used in computations are machine numbers $\check{x} \in]\check{x}_{min},..,\check{x}_{max}[$. Hence, any real number x must be mapped to a machine number $\check{x}$ before it can be processed by a computer.

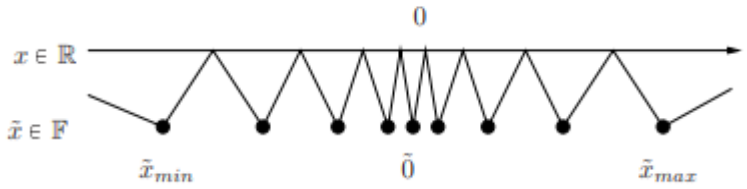

Figure 1. *Representation of real number $x \in R$ by machine numbers $\check{x} \in F$*

Definitions

Definition1: Machine precision, it is described by the machine number ε, ε is the smallest machine number such that $1 + \varepsilon > 1$ on the machine. It is the distance between the integer 1 and the closest number $\check{x} \in F$, which is greater than 1.

Definition2: The IEEE 754 standard is a widely adopted framework for representing floating-point numbers in computer systems. It defines formats for both single-precision and double-precision numbers, ensuring consistency and accuracy in numerical computations across different computing platforms.

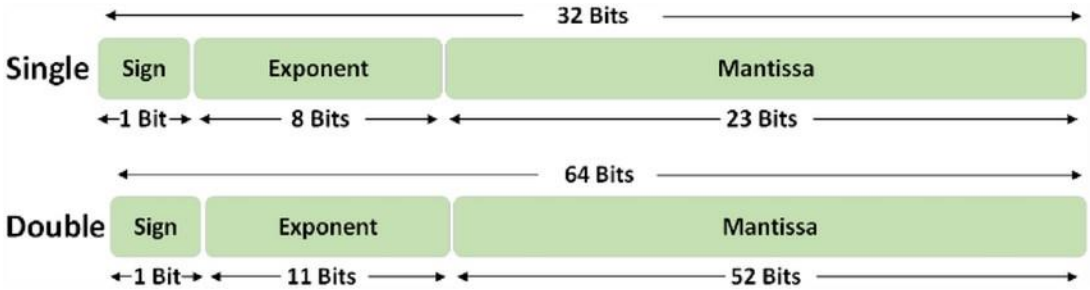

Figure 2. *Representation of The IEEE 754 standard for both single-precision and double-precision numbers*

The IEEE 754 standard defines a framework for representing floating-point numbers in computing, ensuring consistency and accuracy across platforms. It includes two primary formats: single precision (32 bits) and double precision (64 bits). In both formats, the representation is divided into three main components: the sign bit, which indicates whether the number is positive or negative; the exponent, which is biased to allow for both positive and negative values; and the mantissa, which represents the significant digits of the number in a normalized form. The bias for single precision is 127, while for double precision, it is 1023. For single precision, the format consists of 1 sign bit, 8 exponent bits, and 23 mantissa bits, while double precision uses 1 sign bit, 11 exponent bits, and 52 mantissa bits. This structure allows for efficient computation and standardized handling of special values, such as zero, infinity, and NaN (Not a Number), facilitating reliable numerical operations in various applications.

Example: Here's the representation of -6.75 in both single-precision and double-precision IEEE 754 formats.

1. Single Precision (32 bits)

Step 1: Convert to Binary
- *Absolute value*: 6.75 in binary is 110.11.

Step 2: Normalize
- *Normalize to:* 1011×2^2

Step 3: Sign Bit
- *Sign (S):* 1 (negative)

Step 4: Exponent
- Actual exponent: 2
- Biased exponent: $2 + 127 = 129$
- In binary: 10000001

Step 5: Mantissa
- Mantissa (without leading 1): 10110000000000000000000 (23 bits)

Final Representation

Putting it all together:
| 1 | 10000001 | 10110000000000000000000 |

2. Double Precision (64 bits)

Step 1: Convert to Binary
- *Absolute value*: 6.75 in binary is 110.11.

Step 2: Normalize
- *Normalize to:* 1011×2^2

Step 3: Sign Bit
- *Sign (S):* 1 (negative)

Step 4: Exponent
- Actual exponent: 2

- Biased exponent: $2 + 1023 = 1025$
- In binary: 10000001001 (11 bits)

Step 5: Mantissa

- Mantissa (without leading 1):
  1011000000000000000000000000000000000000000000000000 (52 bits)

Final Representation

Putting it all together:
| 1 | 10000001001 | 1011000000000000000000000000000000000000000000000000 |

1.2.2 Rounding Errors

Rounding errors occur when numbers are approximated to fit within the finite precision of a computer's numerical representation. These errors arise because many real numbers cannot be represented exactly in a binary format, leading to discrepancies between the true value and its computed representation. When performing arithmetic operations, results may require more precision than can be accommodated, necessitating rounding to fit the format. As a result, rounding errors can accumulate in iterative calculations, potentially leading to significant inaccuracies in the final output.

1.2.2.1 Absolute Error

The absolute error measures the difference between the exact value and the approximate value obtained through computation. It is defined as:

$$Absolute\ Error = \left|\text{Exact Value} - \text{Approximate Value}\right| \quad (5)$$

For example, if the exact value of a number is 3.14159 and the computed value is 3.14, the absolute error is: |3.14159−3.14|=0.00159

1.2.2.2 Relative Error

Relative error provides a measure of the absolute error in relation to the size of the exact value, allowing for a better understanding of the error's significance in context. It is calculated as:

$$\text{Relative Error} = \frac{\left|\text{Exact Value} - \text{Approximate Value}\right|}{\left|\text{Approximate Value}\right|} \quad (5)$$

Using the previous example, the relative error for the values 3.14159 (exact) and 3.14 (approximate) would be:

$$\text{Relative Error} = \frac{|0.00159|}{|3.14159|} \approx 0.000506 \quad (6)$$

Rounding errors can significantly impact the accuracy of numerical computations due to limited precision in representing real numbers. Absolute error quantifies the direct difference between exact and approximate values, while relative error contextualizes this difference by comparing it to the size of the exact value. Understanding both types of errors is crucial for assessing the reliability of numerical results in various applications.

1.3. Stability and Error Analysis of Numerical Methods and Problem Conditioning

To develop reliable and accurate numerical algorithms, it is essential to integrate error analysis, stability analysis, and considerations of problem conditioning:

1.3.1 Algorithm Selection

Algorithm selection in numerical analysis is a multi-faceted process that involves understanding the problem, analyzing errors, evaluating stability and complexity, and considering problem conditioning. By systematically evaluating these factors, one can choose the most appropriate numerical method that balances accuracy, efficiency, and reliability, ensuring that the chosen approach yields trustworthy results in practice. This careful selection process is essential for successfully solving complex mathematical problems across various scientific and engineering disciplines. Here's a detailed breakdown of the factors to consider when selecting an algorithm.

1.3.1.1 Understanding the Problem

Before selecting an appropriate algorithm, it is crucial to thoroughly understand the nature of the problem at hand. First, identify the "***type of problem"*** you are dealing with, whether it involves linear systems, nonlinear equations, optimization tasks, differential equations, or integration. Each type requires specific algorithms that leverage their unique characteristics and properties. Additionally, consider the "***dimensionality"*** of the problem, as the number of variables can significantly impact algorithm performance. Some algorithms may excel in lower-dimensional spaces but encounter challenges as dimensionality increases—a phenomenon often referred to as the "curse of dimensionality." This understanding lays the groundwork for making informed choices about which numerical methods will yield the most accurate and efficient results for your specific context.

1.3.1.2 Error Analysis

Understanding the error characteristics of different algorithms is vital for selecting the most suitable method for a given problem. "*Truncation error*" refers to the error introduced when an algorithm approximates a solution, which can be assessed based on the method's step size *h*. For example, numerical differentiation methods exhibit truncation errors that depend on the step size, typically expressed as Eq. (7):

$$E_{\text{trunc}} \sim O(h^p) \tag{7}$$

Where p denotes the order of the method.

In addition to truncation errors, it is essential to analyze "*round-off error"*, which results from the limitations of numerical precision during calculations. Algorithms requiring numerous arithmetic operations, particularly iterative methods, can accumulate significant round-off errors.

To achieve a comprehensive understanding of overall accuracy, one must consider the "*total error"*, which can be expressed as Eq. (8):

$$E_{\text{total}} = E_{\text{trunc}} + E_{round-off} \quad (8)$$

Recognizing how these error components interact and combine is crucial for selecting an algorithm that maintains a manageable total error, ensuring reliable and accurate numerical results.

1.3.1.3 Stability Analysis

The stability of an algorithm is a critical factor that indicates how errors propagate during computations. Stable algorithms are designed to avoid significant amplification of errors throughout the calculation process.

For instance, direct methods like Gaussian elimination are typically stable; however, they can encounter large truncation errors, particularly when applied to ill-conditioned problems. In contrast, iterative methods, such as the Jacobi method, may be unstable, especially if their convergence heavily relies on the initial guess or the conditioning of the problem. The "***condition number"*** of a matrix A, denoted as κ(A), serves as a measure of sensitivity to input variations and numerical errors. It is defined as Eq. (9)

$$\kappa(\mathrm{A}) = \|A\|\|A^{-1}\| \quad (9)$$

A high condition number indicates potential instability, suggesting that small changes in the input can lead to large variations in the output. Therefore, selecting algorithms that effectively mitigate the effects of high condition numbers is essential for ensuring reliable numerical results, particularly in sensitive computational scenarios.

1.3.1.4 Complexity and Efficiency

When selecting an algorithm, it's important to evaluate both its computational complexity and efficiency. "***Time complexity"*** measures how the execution time increases with the size of the input. For instance, solving a system of linear equations using Gaussian elimination has a time complexity of $O(n^3)$, which can be computationally expensive for large systems.

In contrast, iterative methods like the Conjugate Gradient algorithm often offer improved performance, particularly for large sparse systems, as they can converge faster and require fewer operations. Additionally, "***space complexity"*** should be assessed to understand the memory requirements of the algorithm. Some methods necessitate extra space for storing intermediate results, which can pose challenges in resource-constrained environments. Balancing time and space complexity is crucial for selecting efficient algorithms that meet the demands of specific problems while optimizing resource usage.

1.3.1.5 Specific Characteristics of Algorithms

When selecting an algorithm, it is crucial to consider its "***convergence behavior***, as different methods converge at different rates. For example, Newton's method converges quadratically near the solution, while the Bisection method exhibits linear convergence, making it slower but more robust for certain problems. Additionally, the "***robustness"*** of an algorithm is essential; methods sensitive to minor input changes may not be suitable for real-world applications. "***Adaptivity"*** is another key feature; some algorithms can dynamically adjust their parameters based on problem characteristics, such as adaptive step-size methods in numerical integration, which modify the step size

according to estimated errors. Evaluating these characteristics helps ensure the selection of algorithms that are efficient, reliable, and resilient under various conditions.

1.3.1.6 Problem Conditioning

Understanding the conditioning of a problem is crucial for algorithm selection, particularly in differentiating between *"**well-conditioned"*** and *"**ill-conditioned"*** problems. Ill-conditioned problems are sensitive to small input perturbations, which can lead to significant output variations, complicating the numerical solution. In such cases, it is often beneficial to use regularization techniques or select more robust algorithms that minimize error propagation. These strategies help stabilize the solution, ensuring that minor changes in input do not cause disproportionately large errors in the results. Recognizing the problem's conditioning enables informed decisions about which algorithms will provide reliable and accurate solutions.

1.3.1.7 Availability of Libraries and Tools

When choosing an algorithm, it's essential to consider the availability of implementation resources, such as libraries and tools that facilitate application. Libraries like NumPy and SciPy, along with specialized numerical solvers, can save significant time and effort by offering pre-implemented functions and optimized algorithms. Additionally, strong community support and comprehensive documentation are invaluable; they help users troubleshoot issues and provide practical insights into algorithm usage. This support enhances the implementation of numerical methods, ensuring effective application while minimizing potential challenges.

1.3.2 Refinement and Adaptation

Numerical methods can often be refined based on error and stability assessments. For instance, adaptive algorithms can adjust parameters like step size dynamically based on estimated errors.

1.3.3 Validation and Testing

Rigorous testing and validation against known solutions or benchmarks can help verify that the combined considerations of error, stability, and conditioning lead to reliable numerical results.

Error analysis and stability are vital for ensuring the reliability and accuracy of numerical algorithms. A comprehensive understanding of these concepts, combined with an awareness of problem conditioning, allows developers to create robust numerical methods suitable for a wide range of applications. By systematically addressing these aspects, we can enhance the performance and trustworthiness of numerical computations in science and engineering.

Chapter 2 Direct Methods for Solving Linear Systems

Definitions

Definition 1:
A *linear equation* is an equation that can be expressed in the form:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

Where $a_1, a_2, \dots, a_n$ are coefficients, $x_1, x_2, \dots, x_n$ are variables, and b is a constant.

Definition 2:
A *linear system of equations* consists of two or more linear equations involving the same set of variables.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

where $x_1, x_2, \dots, x_n$ are the unknowns, $a_{11}, a_{12}, \dots, a_{nn}$ are the coefficients of the system, and $b_1, b_2, \dots, b_n$ are the constant terms.
There are three possible behaviors of a linear system: infinitely many solutions, a single unique solution $\left(x_1, x_2, \dots, x_n\right)^T$, and no solution.
Understanding the possible behaviors of linear systems—infinitely many solutions, a single unique solution, and no solution—is crucial in both theoretical and practical contexts. Each behavior has distinct implications for how we approach solving systems of equations and understanding the relationships between variables. By identifying the nature of a linear system, we can apply the appropriate methods for analysis and solution.

Definition 3:
Triangular and *diagonal systems* are special cases of linear systems that offer the advantage of being very easy to solve. They form an important foundation for understanding and implementing more general methods for solving linear systems.

1) *Upper triangular system*, each equation has the form:

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

   Here, all coefficients below the main diagonal are zero ($\forall i, j = 1..n, i > j : a_{ij} = 0$).

2) *Lower triangular system*, the arrangement is inverted:

$$\begin{cases} a_{1n}x_n = b_1 \\ a_{2(n-1)}x_{n-1} + a_{2n}x_n = b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

   Where ($\forall i, j = 1..n, i < j : a_{ij} = 0$).
3) In a *diagonal system* of linear equations, the arrangement of the coefficients is such that all non-zero elements lie on the main diagonal of the coefficient matrix.
4) $$\begin{cases} a_{11}x_1 = b_1 \\ a_{22}x_2 = b_2 \\ \dots\dots\dots\dots \\ a_{nn}x_n = b_n \end{cases}$$
   Where ($\forall i, j = 1..n, i \neq j : a_{ij} = 0 \; and \; a_{ii} \neq 0$).

2.1 Remarks on Solving Triangular Systems

One of the fundamental principles in linear algebra is that any system of linear equations can be transformed into a triangular system. This is typically achieved through a process called ***Gaussian elimination*** or ***LU decomposition***. By manipulating the equations using row operations such as swapping rows, multiplying rows by non-zero constants, and adding multiples of one row to another one can systematically eliminate variables to achieve either an upper or lower triangular form.

2.1.1 Solving Upper Triangular System

To solve an upper triangular system, we use **back substitution**. The idea is to start from the last equation and substitute backward. The process is:

1) Solve for x_n:

$$x_n = \frac{b_n}{a_{nn}} \tag{10}$$

2) Substitute x_n into the previous-to-First equation and Solve for $\mathrm{i} = \mathrm{n} - 1 \ldots 1, x_i$:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \tag{11}$$

Example:

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 7 \\ 4x_2 + 2x_3 = 10 \\ 5x_3 = 15 \end{cases} \rightarrow \begin{cases} 2x_1 + 3x_2 + x_3 = 7 \\ 4x_2 + 2x_3 = 10 \\ \boldsymbol{x_3 = \frac{15}{5} = 3} \end{cases} \rightarrow \begin{cases} 2x_1 + 3x_2 + x_3 = 7 \\ \boldsymbol{x_2 = \frac{10 - 2(3)}{4} = 1} \\ \boldsymbol{x_3 = 3} \end{cases} \rightarrow \begin{cases} \boldsymbol{x_1 = \frac{7 - 3(1) - (3)}{2} = \frac{1}{2}} \\ \boldsymbol{x_2 = 1} \\ \boldsymbol{x_3 = 3} \end{cases}$$

The solution x is $\left(\frac{1}{2}, 1, 3\right)^T$

2.1.2 Solving Lower Triangular System

To solve a lower triangular system, we use forward substitution. The steps are as follows:

1) Solve for x_1:

$$x_1 = \frac{b_1}{a_{11}} \tag{12}$$

2) Substitute x_1 into the second-to-last equation and Solve for $\mathrm{i} = 2 \ldots \mathrm{n}, x_i$:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j}{a_{ii}} \tag{13}$$

Example:

$$\begin{cases} 2x_1 = 6 \\ 4x_1 + 3x_2 = 18 \\ 5x_1 - 2x_2 + 3x_3 = 7 \end{cases} \rightarrow \begin{cases} \boldsymbol{x_1 = \frac{6}{2} = 3} \\ 4x_1 + 3x_2 = 18 \\ 5x_1 - 2x_2 + 3x_3 = 7 \end{cases} \rightarrow \begin{cases} \boldsymbol{x_1 = 3} \\ \boldsymbol{x_2 = \frac{18 - 4(3)}{3} = 2} \\ 5x_1 - 2x_2 + 3x_3 = 7 \end{cases} \rightarrow \begin{cases} \boldsymbol{x_1 = 3} \\ \boldsymbol{x_2 = 2} \\ \boldsymbol{x_3 = \frac{7 - 5(3) + 2(2)}{3} = -\frac{4}{3}} \end{cases}$$

The solution x is $\left(3{,}2, -\frac{4}{3}\right)^T$

Triangular systems of equations provide a structured and efficient framework for solving linear equations. By transforming any system into a triangular form, we can leverage back substitution for upper triangular systems and forward substitution for lower triangular systems. Understanding these concepts is crucial for advanced numerical methods and applications in various fields, including engineering and computer science.

2.2 Gaussian Elimination Method

Gaussian elimination is a powerful and systematic method for solving systems of linear equations. The primary goal of this technique is to transform a rectangular system represented as AX= B into an equivalent upper triangular form, denoted as UX= C. In this representation, A is the coefficient matrix, X is the vector of variables, and B is the constant vector. The matrix U is an upper triangular matrix, and C is a modified constant vector resulting from the elimination process.

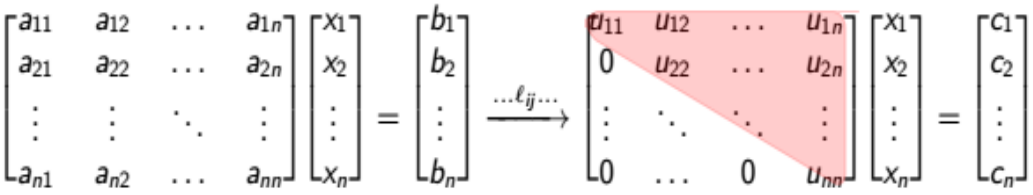

Figure 3. *Gaussian Elimination Method*

In summary, the Gaussian elimination process consists of two main steps: triangularization and resolution. First, we express a system of linear equations in the matrix form AX= B, combining the coefficient matrix A and constants vector B into an augmented matrix [A|B] The goal of triangularization is to transform this matrix into row echelon form, where all entries below the main diagonal are zeros. If a pivot is zero, we handle this by swapping with a non-zero row below it. After achieving row echelon form, the resolution step involves back substitution: starting from the last equation, we express each variable in terms of the others and substitute back up to find the values of all unknowns.

1. **Step 1: Triangularization**

1) **Writing the System in Matrix Form:** To begin, we express the system of linear equations in the form:$\boldsymbol{AX = B}$

$$A = \begin{pmatrix} a_{11} a_{12} \dots a_{1n} \\ a_{21} a_{22} \dots a_{2n} \\ \vdots \quad \vdots \quad \vdots \\ a_{n2} a_{n2} \cdots a_{nn} \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad (14)$$

Where:A is the coefficient matrix, X is the vector of unknowns, B is the constants vector.

2) **Formulating the Augmented Matrix**

Combine the matrix A and vector B into an augmented matrix denoted as $[A|B]$:

$$[A|B] = \left[\begin{array}{ccc|c} a_{11} a_{12} \dots a_{1n} & b_1 \\ a_{21} a_{22} \dots a_{2n} & b_2 \\ \vdots \quad \vdots \quad \vdots & \vdots \\ a_{n2} a_{n2} \cdots a_{nn} & b_n \end{array}\right] \quad (15)$$

Transforming to Row Echelon Form

The objective is to convert the augmented matrix into an upper triangular form (row echelon form), where all entries below the main diagonal are zeros.

$$[A|B] = \left[\begin{array}{ccc|c} a_{11} a_{12} \dots a_{1n} & b_1 & L_1 \\ a_{21} a_{22} \dots a_{2n} & b_2 & L_2 \\ \vdots \quad \vdots \quad \vdots & \vdots & \vdots \\ a_{n2} a_{n2} \cdots a_{nn} & b_n & L_n \end{array}\right] \quad (16)$$

3) Using the Elimination Formula

For $i = 1..(n-1)$ each pivot a_{ii}, replace the entries below it using the formula:

$$For\ j = i+1 \dots n,\ \ L_j = \ L_j - \frac{a_{ji}}{a_{ii}} \times \ L_i \quad (17)$$

Where:

- L_j is the j^{th}row,
- $L_i$is the i^{th}row (the pivot row),
- a_{ji} is the element below the pivot to reduce it to zero at position (j, i)
- a_{ii} is the pivot element at position (i, i).

4) Handling Zero Pivots

- **Identifying a Zero Pivot**

If the pivot element a_{ii} is zero, we cannot proceed with elimination directly, as division by zero is undefined.

- **Row Swapping**

To resolve this issue, look for a non-zero entry in the same column below the current pivot row. If found, swap the current row with the row containing the non-zero entry: $L_i \leftrightarrow L_k$
Where L_k is the row with the non-zero element: $i < K \leq n,\ a_{ki} \neq 0$.
After swapping, continue using the elimination formula as before.

5) Repeat the elimination process for each column until the matrix is in row echelon form

2. Step 2: Resolution

After triangularization, we obtain the following matrix system:

$$U = \begin{pmatrix} u_{11} u_{12} & \dots & u_{1n} \\ 0 \quad a_{22} & \dots & u_{2n} \\ \vdots \quad \vdots & \ddots & \vdots \\ 0 \quad 0 & \cdots & u_{nn} \end{pmatrix}, C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$$

Next, we transform the new system UX=C into a system of linear equations to solve for X, which will be the same solution as for the original system AX=B.

$$\begin{cases} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n = c_1 \\ \qquad u_{22}x_2 + \cdots + u_{2n}x_n = c_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad\qquad\qquad u_{nn}x_n = c_n \end{cases} \tag{18}$$

1) Back Substitution

- Solve for x_n:

$$x_n = \frac{c_n}{u_{nn}} \tag{19}$$

- Substitute x_n into the previous-to-First equation and Solve for $\mathrm{i} = \mathrm{n} - 1 \ldots 1, x_i$:

$$x_i = \frac{c_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}} \tag{20}$$

By transforming the system UX=C into a system of linear equations and solving through back substitution, you can effectively find the solution vector X, similar to how you would have solved AX=B. This process maintains the same structure and approach, ensuring consistency in solving linear systems.

Example:

Consider the following system of equations:

$$\begin{cases} 2x + 3y + z = 1 \\ 4x + y + 2z = 2 \\ -2x + 5y + 2z = 3 \end{cases}$$

Step 1: Write the Augmented Matrix

First, we form the augmented matrix [A|B]:

$$\left[\begin{array}{ccc|c} 2 & 3 & 1 & 1 \\ 4 & 1 & 2 & 2 \\ -2 & 5 & 2 & 3 \end{array}\right] \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

Step 2: Triangularization

For i=1 to (n-1)=2

1. $i = 1$ **pivot** $a_{11} = 2 \neq 0$

Eliminate the first column below the pivot by following Eq. (17):

$$L_j = L_j - \frac{a_{ji}}{a_{ii}} \times L_i$$

- For $L_2 = L_2 - \frac{4}{2}L_1 = L_2 - 2L_1$
- For $L_3 = L_3 - \frac{(-2)}{2}L_1 = L_3 + L_1$

The calculations for L_2 and L_3 are as follows:

$$L_2 = (4\ 1\ 2|2) - 2(2\ 3\ 1|1) = (4\ 1\ 2|2) - (4\ 6\ 2|2) = (0\ -5\ 0|0)$$

$L_3 = (-2\ \ 5\ 2|3) + (\ 2\ 3\ 1|1) = (\ 0\ 8\ 3|4)$

The augmented matrix is now:

$$\left[\begin{array}{ccc|c} 2 & 3 & 1 & 1 \\ 0 & -5 & 0 & 0 \\ 0 & 8 & 3 & 4 \end{array}\right] \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

2. $i = 2$ **pivot** $a_{22} = -5 \neq 0$

Eliminate the first column below the next pivot

- For $L_3 = L_3 - \frac{8}{(-5)} L_1 = L_3 + L_1$

$$L_3 = (0\ -8\ \ 3|4) + \frac{8}{5}(0\ -5\ \ 0|0) = (0\ \ 0\ \ 3|4)$$

Finally, the augmented matrix is now:

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

Step 3: Back Substitution

From the final augmented matrix, we can write the equations:

$$\begin{cases} 2x + 3y + z = 1 \\ -5y = 0 \\ 3z = 4 \end{cases} \rightarrow \begin{cases} x = -\frac{1}{6} \\ y = 0 \\ z = \frac{4}{3} \end{cases}$$

Final Solution

The solution to the system AX=B is: $x = \left(-\frac{1}{6}, 0, \frac{4}{3}\right)^T$

Exercise

Consider the following system of linear equations represented by:

$$\begin{cases} 2y + z = 4 \\ x + 3y + 2z = 5 \\ 2x + y + z = 3 \end{cases}$$

1. Form the augmented matrix [A|B] and perform Gaussian elimination.
2. Then, deduce the determinant of A.
   NB: $\mathbf{det}(\boldsymbol{A}) = (-\mathbf{1})^{\boldsymbol{p}} . \prod_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{u_{ii}}$
   Where *p* is the number of permutations performed during the triangularization of *A* and U is an upper triangular matrix.

2.3 Matrix Interpretation of Gaussian Elimination: LU Factorization

LU Factorization is a method used to decompose a given matrix A into two matrices: a lower triangular matrix L and an upper triangular matrix U. This decomposition is particularly useful for solving systems of linear equations, calculating determinants, and performing matrix inversions efficiently (as shown in Figure.4).

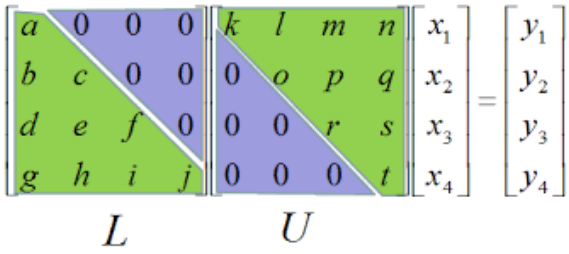

Figure 4. *Representation of LU Factorization*

For a matrix A to be decomposed into its LU factorization, it must satisfy several conditions: First, A must be a square matrix. Second, it must be non-singular, meaning that its determinant is non-zero (det(A)≠0). Lastly, if zero pivots are encountered during the elimination process, the use of permutation matrices may be necessary to enable the factorization. These conditions collectively ensure that the LU factorization can be performed without issues.

Steps to Obtain LU Factorization from Gaussian Elimination:

1) **Initial Matrix**: Consider a square matrix A of size n×n.
2) **Form of the Decomposition**: We seek to express:

   A= LU

$$A = \begin{pmatrix} a_{11} a_{12} \dots a_{1n} \\ a_{21} a_{22} \dots a_{2n} \\ \vdots \quad \vdots \quad \vdots \\ a_{n2} a_{n2} \cdots a_{nn} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & \dots 0 \\ l_{21} & 1 & \dots 0 \\ \vdots & \vdots & \vdots \\ l_{n2} & l_{n2} & \cdots 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} u_{11} u_{12} \dots u_{1n} \\ 0 \quad u_{22} \dots u_{2n} \\ \vdots \quad \vdots \quad \vdots \\ 0 \quad 0 \quad \cdots \quad u_{nn} \end{pmatrix}}_{U} \tag{21}$$

   Where L is a lower triangular matrix with ones on the diagonal and U is an upper triangular matrix.
3) **Gaussian Elimination**:
   Apply Gaussian elimination to transform A into an upper triangular matrix U.
   At each step of elimination, as you eliminate the entries below the diagonal of A, note the multipliers used to zero out the elements in the current column. These multipliers will form the elements of L.
4) **Constructing L and U**:
   For each non-zero element a_{ij} used for elimination, record the multiplier $m_{ij} = \frac{a_{ij}}{a_{jj}}$ and place this multiplier in the position $(i, j)$of matrix L.
   The matrix U will be the resulting matrix after all elimination steps.

When we have a linear system represented by the matrix equation: AX=B And we have a LU decomposition of matrix A, meaning A = LU, we can substitute LU for A in the original equation, as shown in Eq. (22):

$$AX = B \rightarrow (LU)X = B \rightarrow L(UX) = B \rightarrow LY = B$$

$$where \begin{cases} LY = B & is\ the\ first\ system\ to\ solve \\ UX = Y & is\ the\ second\ system\ to\ solve \end{cases} \quad (22)$$

By introducing a new variable Y, we can break down the problem into two simpler triangular systems:

1. To solve for Y, we start by defining Y=UX. This transforms our equation into LY=B. Given that L is a lower triangular matrix, we can apply forward substitution to efficiently solve for Y.
2. After obtaining Y, we proceed to solve the equation UX= Y. Since U is an upper triangular matrix, we can efficiently find X using backward substitution.

By factoring A into LU, we transform a potentially complex linear system into two simpler triangular systems, which can be solved efficiently using forward and backward substitution.

Example:

Consider the following matrix A:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix}$$

$$A = LU = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l12 & 1 & 0 \\ l13 & l32 & 1 \end{bmatrix} * \begin{bmatrix} u11 & u12 & u13 \\ 0 & u22 & u23 \\ 0 & 0 & u33 \end{bmatrix}$$

Using Gaussian elimination on A

For i=1 to (n-1) =2

- **Calculate i^{th} column of L**
- **Calculate i^{th} row of U**

1) For L: $i = 1\ and\ k = 2..n,\ \ l_{ki} = \frac{a_{ki}^{(i-1)}}{a_{ii}^{(i-1)}}$

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ l12 = \dfrac{a^{(0)}{}_{12}}{a^{(0)}{}_{11}} = \dfrac{4}{1} = 4 & 1 & 0 \\ l13 = \dfrac{a^{(0)}{}_{13}}{a^{(0)}{}_{11}} = \dfrac{3}{1} = 3 & l32 & 1 \end{bmatrix}$$

For U: Eliminate Entries Below the Pivot i=1: $a_{11} = 1 \neq 0$

$$For\ j = i + 1 \ldots n,\ \ L_j = \ L_j - \frac{a_{ji}}{a_{ii}} \times \ L_i$$

$$A^{(0)} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix} \begin{matrix} L_2 = L_2 - \frac{4}{1} L_1 \\ L_3 = L_{32} - \frac{3}{1} L_1 \end{matrix} \rightarrow A^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 2 & 0 \end{bmatrix}$$

2) For L: $i = 2\ and\ k = 2..n,\ \ l_{ki} = \frac{a_{ki}^{(k-1)}}{a_{ii}^{(k-1)}}$

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & l32 = \frac{a^{(1)}{}_{32}}{a^{(1)}{}_{22}} = \frac{2}{-1} = -2 & 1 \end{bmatrix}$$

Eliminate Entries Below the Pivot 2: $a_{22} = -1 \neq 0$

$$A^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 2 & 0 \end{bmatrix} L_3 = L_{32} - \frac{2}{(-1)} L_1 = L_3 + 2L_1 \rightarrow A^{(2)} \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix} = U$$

The obtained matrix is the upper triangular matrix U

$$U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix}$$

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

Exercise

Consider the following system of linear equations::

$$\begin{cases} 2x + 3y + z = 1 \\ 4x + 7y + 2z = 2 \\ 6x + 18y + 5z = 3 \end{cases}$$

1. Perform LU decomposition of the coefficient matrix A
2. Solve the system using the LU decomposition method.
3. Then, deduce the determinant of A.

Chapter 3 Iterative Methods for Solving Linear Systems

3.1 General Considerations

Direct and iterative methods offer two distinct approaches to solving systems of linear equations. Direct methods, such as Gaussian elimination or LU decomposition, compute the exact solution in a finite number of steps but can be computationally expensive and memory-intensive for large systems. In contrast, iterative methods, like the Jacobi method or conjugate gradient, approximate the solution iteratively, refining an initial guess. While they do not guarantee an exact solution in a finite number of iterations, they are often more efficient in terms of computational time and memory usage, especially for sparse systems. The choice between a direct and iterative method depends on the size of the system, the structure of the matrix, the desired accuracy, and the available computational resources.

Definition 1:

An iterative method for solving the linear system Ax=b is a systematic approach that generates a sequence $X^{(k)}$ $where\ k \in IN$. Each iterate $X^{(k)}$ is computed based on the previous iterates $X^{(0)}$,…, $X^{(k-1)}$, with the goal of converging to the solution X of the linear system. Typically, the construction of the sequence follows a recurrence relation of the form shown in Equation (23):

$$X^{(k)} = \boldsymbol{B}X^{(k-1)} + C \tag{23}$$

In this equation, $\boldsymbol{B}$ represents the iteration matrix derived from A, and C is a vector that depends on b. This formulation allows for flexibility in how the iterates are generated. By examining the limit as k approaches infinity ($k \to +\infty$), we observe that the solution X must satisfy Eq.(24):

$$X = BX + C \tag{24}$$

This reveals an important relationship between the solution and the matrices involved. Since we know that $X = A^{-1}b$, it follows that C can be expressed in Eq. (25) as:

$$C = (Id - b)\, A^{-1}b \tag{25}$$

This leads to the conclusion that the iterative method is fundamentally defined by the iteration matrix $\boldsymbol{B}$. Thus, understanding the structure and properties of $\boldsymbol{B}$ is crucial for analyzing the convergence and efficiency of the iterative method in solving the linear system $AX = b$.

Definition 2:

Splitting, A general technique for constructing the matrix $\boldsymbol{B}$ is based on a decomposition (or splitting) of the matrix A in the form:

$$A = P - N \tag{26}$$

Where P is an invertible matrix that is easy to invert, such as a diagonal or triangular matrix. The matrix P is referred to as the conditioning matrix. This approach is fundamental in iterative methods because it simplifies the process of finding solutions to linear systems.

$$PX^{(k+1)} = NX^{(k)} + b \rightarrow X^{(k+1)} = P^{-1}NX^{(k)} + P^{-1}b$$
$$where\ B = P^{-1}N\ and\ C = P^{-1}b \quad (27)$$

In this context, the matrix P plays a critical role as it dictates the stability and convergence properties of the iterative algorithm. The idea is to isolate the more easily manageable part of the matrix A (represented by P) from the more complex part (represented by N). By rearranging A in this way, we can define an iterative algorithm that can be expressed in a recurrence relation, allowing us to compute the next iterate based on the previous one.

The iterative algorithm can then be written as Eq. '23) ($X^{(k)} = \boldsymbol{B}X^{(k-1)} + C$).

where $\boldsymbol{B}$ is derived from the decomposition, and C incorporates the vector b. This formulation allows us to systematically update our estimates of the solution X in each iteration. The flexibility of choosing P allows for various strategies to optimize convergence, depending on the characteristics of the specific linear system being solved.

Overall, the splitting technique is a powerful tool in numerical linear algebra, providing a structured method to analyze and implement iterative methods effectively.

In practice, the most common splittings are based on the representation:

$$A = D - E - F \quad (28)$$

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} \qquad -E = \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n2} & a_{n2} & \cdots & 0 \end{pmatrix} \qquad -F = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

Where D is the diagonal part of A, E is the strictly lower triangular part (with zeros on the diagonal), and F is the strictly upper triangular part (also with zeros on the diagonal) as shown in Eq. (28).

3.2 Jacobi and Relaxation Methods

3.2.1 Jacobi Method

The Jacobi method is an iterative technique for solving systems of linear equations of the form $AX = b$. This method is particularly useful for large and sparse matrices, where direct methods may be inefficient. There are two primary approaches to implementing the Jacobi method: the classical format based on explicit equations and the alternative format utilizing matrix splitting.

Both approaches have their advantages and applications, and understanding them provides a comprehensive view of how the Jacobi method operates in practice. In this course, we will delve into both methods, starting with the classical formulation before exploring the benefits of the splitting technique.

3.2.1.1 Equations-Based Formula

To implement the corresponding algorithm, consider the following system of linear equations in Eq. (29). This method involves directly manipulating the equations to derive the iterative formula.

$$AX = b \rightarrow \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (29)$$

Therefore, by isolating x_1 from the first equation, x_2 from the second equation, ..., and x_n from the n^{th} equation, we obtain the expression given by Eq. (30). Each variable is isolated and expressed in terms of the other variables, leading to a straightforward computation in each iteration.

$$\begin{cases} x_1 = \frac{(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n)}{a_{11}} \\ x_2 = \frac{(b_2 - a_{21}x_1 - \cdots - a_{2n}x_n)}{a_{22}} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n = \frac{(b_n - a_{n1}x_1 - \cdots - a_{n(n-1)}x_{n-1})}{a_{nn}} \end{cases} \rightarrow \forall i = 1..n, x_i = \frac{1}{a_{ii}}(b_i - \sum_{\substack{i=1 \\ i \neq j}}^{n} a_{ij}x_i) \quad (30)$$

Derivation of the Iterative Formula

Starting from a given initial vector $X^{(0)}$, we construct the sequence of vectors $(X^{(k)})_{k \geq 0}$, defined by Eq. (31).

$$\forall i = 1 \ldots n, \; {x_i}^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{\substack{i=1 \\ i \neq j}}^{n} a_{ij}{x_i}^{(k)}) \quad (31)$$

Thus, by juxtaposing Eq. (31) with Eq. (23): ($X^{(k)} = \boldsymbol{B}X^{(k-1)} + C$), we obtain the iteration matrix $\boldsymbol{B_j}$ for the Jacobi method and a vector C in Eq. (32)

$$\boldsymbol{B_j} = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} \ldots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 \quad \ldots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \\ -\frac{a_{1n}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} \cdots & \vdots \\ & & 0 \end{pmatrix} \quad and \quad C = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix} \quad (32)$$

3.2.1.2 Splitting-Based Formula

This technique involves decomposing the matrix A into its components: specifically the diagonal part D, the strictly lower triangular part E, and the strictly upper triangular part F. This decomposition simplifies the iteration process and enhances the clarity of the algorithm, as shown in Eq. (28).

Derivation of the Iterative Formula

Given the equation $AX = b$, we can rewrite it using our decomposition:

$$DX = b + (E + F)X \quad (33)$$

Rearranging this leads to Eq. ’34):

$$X = D^{-1}(b + (E + F)X) \quad (34)$$

To derive the iterative formula, we isolate X in Eq. (35).

$$X^{(k)} = D^{-1}(b + (E + F)X^{(k-1)}) \quad (35)$$

This formula indicates that the new iterate $X^{(k)}$ is computed based on the previous iterate $X^{(k-1)}$and the contributions from the off-diagonal elements, represented by E and F.

By substituting Eq. (35) into Eq. (23), we obtain the iteration matrix for the Jacobi method, as given in Eq. (36).

$$\boldsymbol{B_j} = D^{-1}(E+F) \qquad and \qquad C = D^{-1}b \tag{36}$$

3.2.1.3 Iterative Algorithm Steps

The iterative process of the Jacobi method begins with the initialization step, where an initial guess for the solution is selected. This initial guess is often a zero vector, although any reasonable approximation can be used. The choice of initial values can influence the speed of convergence, but the method is generally robust enough to work with various starting points.
Once the initial guess is established, the process moves into the iteration phase. For each iteration, a new estimate is calculated based on the previous iterate. This step refines the approximation of the solution, incorporating the contributions from other variables in the system.
After computing the new estimate, it is essential to perform a convergence check to determine if the iterative process should continue. This is typically done by evaluating a stopping criterion, such as checking whether the difference between consecutive estimates is less than a predetermined small tolerance value. If this difference is sufficiently small, it indicates that the solution has stabilized, and the method can be terminated. This systematic approach ensures that the Jacobi method converges efficiently toward the true solution of the linear system.

1. **Initialization**: Choose an initial guess $X^{(0)}$ (often a zero vector).
2. **Iteration**: For each iteration k:

Calculate $X^{(k)}$using the formula

$$X^{(k)} = \boldsymbol{B_j}X^{(k-1)} + C$$

3. **Convergence Check**: Determine if the method has converged by evaluating the stopping criterion, such as:

$$\left\|X^{(k)} - X^{(k-1)}\right\| < \varepsilon$$

where ε epsilonϵ is a small tolerance value.

Example:

Consider the following system of equations:

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

1) Equations-Based Formula

$$\begin{cases} x_1 = \dfrac{(b_1 - a_{12}x_2 - a_{13}x_3)}{a_{11}} \\ x_2 = \dfrac{(b_2 - a_{21}x_1 - a_{23}x_3)}{a_{22}} \\ x_3 = \dfrac{(b_3 - a_{31}x_1 - a_{32}x_2)}{a_{11}} \end{cases} \rightarrow \begin{cases} x_1 = \dfrac{(2 - x_2 + x_3)}{3} \\ x_2 = \dfrac{(17 - x_1 - 2x_3)}{5} \\ x_3 = \dfrac{(-18 - 2x_1 + x_2)}{6} \end{cases}$$

Iterative Formula based on equation

$$\begin{cases} {x_1}^{(k+1)} = \dfrac{\left(2 - {x_2}^{(k)} + {x_3}^{(k)}\right)}{3} \\ {x_2}^{(k+1)} = \dfrac{\left(17 - {x_1}^{(k)} - 2{x_3}^{(k)}\right)}{5} \\ {x_3}^{(k+1)} = \dfrac{(-18 - 2{x_1}^{(k)} + {x_2}^{(k)})}{6} \end{cases}$$

$$\boldsymbol{B_j} = \begin{pmatrix} 0 & -1/3 & 1/3 \\ -1/5 & 0 & -2/5 \\ -1/3 & 1/6 & 0 \end{pmatrix} \qquad and \qquad C = \begin{pmatrix} 2/3 \\ 17/5 \\ -3 \end{pmatrix}$$

The iterative process of the Jacobi method:

$$\text{If } X^{(0)} = \begin{pmatrix} \frac{2}{3} \\ \frac{17}{5} \\ -3 \end{pmatrix} \rightarrow \begin{cases} {x_1}^{(1)} = \frac{\left(2 - {x_2}^{(0)} + {x_3}^{(0)}\right)}{3} \\ {x_2}^{(1)} = \frac{\left(17 - {x_1}^{(0)} - 2{x_3}^{(0)}\right)}{5} \\ {x_3}^{(1)} = \frac{(-18 - 2{x_1}^{(0)} + {x_2}^{(0)})}{6} \end{cases} \rightarrow \begin{cases} {x_1}^{(1)} = \frac{\left(2 - (\frac{17}{5}) + (-3)\right)}{3} = 0{,}5333 \\ {x_2}^{(1)} = \frac{\left(17 - (\frac{2}{3}) - 2(-3)\right)}{5} = 2{,}0666 \\ {x_3}^{(1)} = \frac{(-18 - 2(\frac{2}{3}) + (\frac{17}{5})}{6} = 2{,}6555 \end{cases}$$

NB: The result is computed with 4 significant digits (or 3 exact decimal places). So, probably, ε =0,5.10^{-3}.

$$\left\|X^{(1)} - X^{(0)}\right\| < 0.5.10^{-3} \rightarrow \left\|X^{(1)} - X^{(0)}\right\| = \left\| \begin{matrix} 0.5333 - \frac{2}{3} \\ 2.0666 - \frac{17}{5} \\ 2.6555 - 3 \end{matrix} \right\| = \left\| \begin{matrix} 0.5333 - \frac{2}{3} \\ 2.0666 - \frac{17}{5} \\ 2.6555 - 3 \end{matrix} \right\| = \left\| \begin{matrix} 1.1999 \\ 1.3334 \\ 0.3445 \end{matrix} \right\|$$

$$\left\| \begin{matrix} 1.1999 \\ 1.3334 \\ 0.3445 \end{matrix} \right\| > \begin{pmatrix} 0.5.10^{-3} \\ 0.5.10^{-3} \\ 0.5.10^{-3} \end{pmatrix}$$ So we should continue to process and calculate $X^{(1)}$.

2) Splitting-Based Formula

In our Sytem AX=b:

$$A = \begin{pmatrix} 3 & 1 & -1 \\ 1 & 5 & 2 \\ 2 & -1 & -6 \end{pmatrix} \qquad and \qquad b = \begin{pmatrix} 2 \\ 17 \\ -18 \end{pmatrix}$$

A=D-E-F

$$D=\begin{pmatrix}3 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & -6\end{pmatrix} \qquad E=\begin{pmatrix}0 & 0 & 0\\ -1 & 0 & 0\\ -2 & 1 & 0\end{pmatrix} \qquad F=\begin{pmatrix}0 & -1 & 1\\ 0 & 0 & -2\\ 0 & 0 & 0\end{pmatrix}$$

Calculate $\boldsymbol{B_j} = D^{-1}(E+F)\ and\ C = D^{-1}b$

$$D^{-1}=\begin{pmatrix}1/3 & 0 & 0\\ 0 & 1/5 & 0\\ 0 & 0 & -1/6\end{pmatrix}$$ Calculate

$Calculate\ D^{-1} using\ Gauss\ jordan\ method\ or\ D^{-1} = \frac{1}{\det(D)} \times adj(D)$

$$(E+F)=\begin{pmatrix}0 & -1 & 1\\ -1 & 0 & -2\\ -2 & 1 & 0\end{pmatrix}$$

$$\boldsymbol{B_j} = D^{-1}(E+F) \rightarrow \boldsymbol{B_j} = \begin{pmatrix}\frac{1}{3} & 0 & 0\\ 0 & \frac{1}{5} & 0\\ 0 & 0 & -\frac{1}{6}\end{pmatrix}\begin{pmatrix}0 & -1 & 1\\ -1 & 0 & -2\\ -2 & 1 & 0\end{pmatrix} = \begin{pmatrix}0 & -1/3 & 1/3\\ -1/5 & 0 & -2/5\\ -1/3 & 1/6 & 0\end{pmatrix}$$

$$C = D^{-1}b = \begin{pmatrix}\frac{1}{3} & 0 & 0\\ 0 & \frac{1}{5} & 0\\ 0 & 0 & -\frac{1}{6}\end{pmatrix}\begin{pmatrix}2\\ 17\\ -18\end{pmatrix} = \begin{pmatrix}2/3\\ 17/5\\ -3\end{pmatrix}$$

$$\boldsymbol{B_j} = \begin{pmatrix}0 & -1/3 & 1/3\\ -1/5 & 0 & -2/5\\ -1/3 & 1/6 & 0\end{pmatrix} \quad \text{and} \quad C=\begin{pmatrix}2/3\\ 17/5\\ -3\end{pmatrix}$$

Iterative Formula based on matrix:

$$\boldsymbol{X^{(k+1)}} = \begin{pmatrix}0 & -1/3 & 1/3\\ -1/5 & 0 & -2/5\\ -1/3 & 1/6 & 0\end{pmatrix}\boldsymbol{X^{(k)}} + \begin{pmatrix}2/3\\ 17/5\\ -3\end{pmatrix}$$

Exercise: Consider the following matrix

$$A=\begin{pmatrix}4 & -1 & 0\\ -1 & 2 & 0\\ 0 & 0 & 3\end{pmatrix}\ and\ B=\begin{pmatrix}2\\ 5\\ 1\end{pmatrix}$$

How many iterations for jacobi method needed to get an accuracy within 10^{-2}.

3.2.2 Relaxation Method

The relaxation method refers to a general family of iterative techniques that gradually refine the solution to a system of equations. It involves adjusting the current estimate of the solution by combining it with some computed corrections.

The idea is to improve convergence by introducing a relaxation factor (usually denoted as ω). In its simplest form, the relaxation method updates the solution by:

$$\begin{cases} X^{(0)} \in IR^n \\ x_i^{(k+1)} = x_i^{(k)} + \omega\left(\frac{b_i - - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}}\right) \end{cases} \quad (37)$$

The relaxation parameter, usually denoted as ω\omegaω, determines the step size of the update. It lies in the range $0 < \omega < 1$

3.3 Gauss-Seidel and Successive Relaxation Methods

3.3.1 Gauss-Seidel Method

The Gauss-Seidel method is a widely-used iterative technique for solving systems of linear equations, particularly advantageous for large-scale and sparse matrices. One of its key benefits is its simplicity, making it accessible for a variety of applications. Additionally, it often converges faster than the Jacobi method, providing quicker solutions, especially in scenarios involving sparse matrices where many elements are zero, thus minimizing computational overhead. However, while generally reliable, the method's convergence is not guaranteed for all systems, and it can be sensitive to the choice of initial guess. Understanding these advantages and limitations helps in effectively utilizing Gauss-Seidel, especially in conjunction with techniques like Successive Over-Relaxation and in parallel computing environments.

Typically, the construction of the detailed equation sequence follows a recurrence relation of the form shown in Equation (38):

$$\begin{cases} X^{(0)} \in IR^n \\ x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}\right) \end{cases} \quad (38)$$

In this equation, $x_i^{(k+1)}$ represents the updated value of the variable $x_i$in the $(k+1)^{th}$iteration. The term $b_i$is the corresponding constant from the system of equations, while a_{ij} are the coefficients from the matrix A. The first summation accounts for the contributions of already updated variables, and the second summation includes the contributions from previous iteration values, ensuring that the most current data is utilized. This iterative approach allows for progressive refinement of the solution, showcasing the method's efficiency, particularly in handling large, sparse systems.

Using this approach, we can represent the iterative update in Eq. (27) as: $X^{(k+1)} = P^{-1}NX^{(k)} + P^{-1}b \;\; where\; B = P^{-1}N \; and \;\; C = P^{-1}b$. It consists of choosing the simplest splitting with P = D-E et N = F. By replacing in the equation Eq. (27), we obtain Eq. (39).

$$X^{(k+1)} = (D-E)^{-1}FX^{(k)} + (D-E)^{-1}b$$

$$\boldsymbol{B_{GS}} = (D-E)^{-1}F \qquad and \qquad C = (D-E)^{-1}b \tag{39}$$

Example:

Consider the same system of equations:

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

Based on A=D-E-F

$$D = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -6 \end{pmatrix} \qquad E = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -2 & 1 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & -1 & 1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

Calculate $\boldsymbol{B_{SOR}} = (D-E)^{-1}\left(\left(\frac{1-\omega}{\omega}\right)D + F\right) \qquad and \qquad C = (D-E)^{-1}b$

$$(D-E) = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -6 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 \\ 1 & 5 & 0 \\ 2 & -1 & -6 \end{pmatrix}$$

$$(D-E)^{-1} = \begin{pmatrix} 0.333 & 0 & 0 \\ -0.066 & 0.2 & 0 \\ 0.122 & -0.033 & -0.166 \end{pmatrix}$$

$Calculate\ (D-E)^{-1}\ \ using\ Gauss\ jordan\ method\ or\ (D-E)^{-1} = \frac{1}{\det\,(D-E)} \times adj((D-E))$

$$\boldsymbol{B_j} = (D-E)^{-1}F \rightarrow \boldsymbol{B_j} = \begin{pmatrix} 0.333 & 0 & 0 \\ -0.066 & 0.2 & 0 \\ 0.122 & -0.033 & -0.166 \end{pmatrix}\begin{pmatrix} 0. & -1 & 1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & -0.333 & 0.333 \\ 0 & 0.066 & -0.466 \\ 0 & -0.122 & 0.188 \end{pmatrix}$$

$$C = (D-E)^{-1}b = \begin{pmatrix} 0.333 & 0 & 0 \\ -0.066 & 0.2 & 0 \\ 0.122 & -0.033 & -0.166 \end{pmatrix}\begin{pmatrix} 2 \\ 17 \\ -18 \end{pmatrix} = \begin{pmatrix} 0.666 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$\boldsymbol{B_j} = \begin{pmatrix} 0 & -0.333 & 0.333 \\ 0 & 0.066 & -0.466 \\ 0 & -0.122 & 0.188 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 0.666 \\ 3.208 \\ 2.671 \end{pmatrix}$$

Gauss Seidal Iterative Formula based on matrix:

$$\boldsymbol{X^{(k+1)}} = \begin{pmatrix} 0 & -0.333 & 0.333 \\ 0 & 0.066 & -0.466 \\ 0 & -0.122 & 0.188 \end{pmatrix}\boldsymbol{X^{(k)}} + \begin{pmatrix} 0.666 \\ 3.208 \\ 2.671 \end{pmatrix}$$

3.3.2 Successive Relaxation Method (SOR)

Relaxation methods are iterative techniques used to solve systems of linear equations, particularly in numerical analysis and computational mathematics. In this method, we slightly modify the previous method by introducing a parameter w, the relaxation coefficient. This parameter is generally constant. Relaxation in the Jacobi method typically does not provide any significant gains (see sub-section 3.2.2. However, when applied to the Gauss-Seidel method (see sub-section 3.3.1), it improves the speed of convergence. The update formula becomes:

$$\begin{cases} X^{(0)} \in IR^n \\ x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}(b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}) \end{cases} \quad (40)$$

In Eq. (40):

- $x_i^{(k+1)}$ is the updated value for the i^{th} variable.
- $x_j^{(k+1)}$ are the most recently updated values for indices $j < i$ and $x_i^{(k)}$ are the previous values for indices $j > i$.

The idea is that if the "*correction*" applied to a component is going in the "right direction," we benefit from increasing it by multiplying by a factor greater than 1 (ω>1\omega > 1ω>1: over-relaxation). Conversely, if there is a risk of diverging or oscillating, it is better to dampen the correction by multiplying by a factor less than 1 (ω<1\omega < 1ω<1: under-relaxation). A necessary but not sufficient condition for the convergence of these methods is that the parameter ω lies between 0 and 2.

Given the equation $AX = b$ we can rewrite it using our decomposition into the components D , E, and F ($where\ A = D - E - F\ see\ Section$ 3.1):

$$X^{(k+1)} = \left(\frac{D}{\omega} - E\right)^{-1}\left(\left(\frac{1-\omega}{\omega}\right)D + F\right)X^{(k)} + \left(\frac{D}{\omega} - E\right)^{-1} b \quad (41)$$

By substituting Eq. (41) into Eq. (23), we obtain the iteration matrix for the SOR method, as given in Eq. (42).

$$\boldsymbol{B_{SOR}} = \left(\frac{D}{\omega} - E\right)^{-1}\left(\left(\frac{1-\omega}{\omega}\right)D + F\right) \qquad and \qquad C = \left(\frac{D}{\omega} - E\right)^{-1} b \quad (42)$$

The iterative process for the SOR method will remain the same as described in subsection 3.2.1.3.

Example:

Consider the following system of equations and $= 1.1$:

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

Based on A=D-E-F

$$D = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -6 \end{pmatrix} \qquad E = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -2 & 1 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & -1 & 1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

Calculate $\boldsymbol{B_{SOR}} = \left(\frac{D}{\omega} - E\right)^{-1}\left(\left(\frac{1-\omega}{\omega}\right)D + F\right) \qquad and \qquad C = \left(\frac{D}{\omega} - E\right)^{-1} b$

$$\left(\frac{D}{\omega} - E\right) = \begin{pmatrix} \frac{3}{1.1} & 0 & 0 \\ 0 & \frac{5}{1.1} & 0 \\ 0 & 0 & -\frac{6}{1.1} \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ -2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2{,}72 & 0 & 0 \\ 1 & 4{,}54 & 0 \\ 2 & -1 & -5{,}45 \end{pmatrix}$$

$$\left(\frac{D}{\omega} - E\right)^{-1} = \begin{pmatrix} 0.37 & 0 & 0 \\ -0.08 & 0.22 & 0 \\ 0.15 & -0.04 & -0.18 \end{pmatrix}$$

$Calculate\ \left(\frac{D}{\omega} - E\right)^{-1}\ using\ Gauss\ jordan\ method\ or\ \left(\frac{D}{\omega} - E\right)^{-1} = \frac{1}{\det\left(\frac{D}{\omega} - E\right)} \times adj(\left(\frac{D}{\omega} - E\right))$

$$\left(\left(\frac{1-\omega}{\omega}\right)D + F\right) = \left(\left(\frac{1-1.1}{1.1}\right)D + F\right) = \left((-0{,}09)D + F\right) = \begin{pmatrix} -0.27. & -1 & 1 \\ 0 & -0.45 & -2 \\ 0 & 0 & 0.54 \end{pmatrix}$$

$$\boldsymbol{B_j} = \left(\frac{D}{\omega} - E\right)^{-1}\left(\left(\frac{1-\omega}{\omega}\right)D + F\right) \rightarrow \boldsymbol{B_j} = \begin{pmatrix} 0.37 & 0 & 0 \\ -0.08 & 0.22 & 0 \\ 0.15 & -0.04 & -0.18 \end{pmatrix}\begin{pmatrix} -0.27. & -1 & 1 \\ 0 & -0.45 & -2 \\ 0 & 0 & 0.54 \end{pmatrix}$$
$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix}$$

$$C = \left(\frac{D}{\omega} - E\right)^{-1} b = \begin{pmatrix} 0.37 & 0 & 0 \\ -0.08 & 0.22 & 0 \\ 0.15 & -0.04 & -0.18 \end{pmatrix}\begin{pmatrix} 2 \\ 17 \\ -18 \end{pmatrix} = \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$\boldsymbol{B_j} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

Iterative Formula based on matrix:

$$\boldsymbol{X^{(k+1)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(k)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$\boldsymbol{If\ X^{(0)}} = \begin{pmatrix} 0.00 \\ 0.00 \\ 0.00 \end{pmatrix}$ and In order to calculate the solution X using the Successive Over-Relaxation (SOR) method up to the 5th iteration, I will perform the following steps:

$$\boldsymbol{X^{(1)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(0)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \begin{pmatrix} 0.00 \\ 0.00 \\ 0.00 \end{pmatrix} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} = \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} \rightarrow \boldsymbol{X^{(1)}} = \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$\boldsymbol{X^{(2)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(1)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} = \begin{pmatrix} 0.407 \\ 2.18 \\ 2.73 \end{pmatrix} \rightarrow \boldsymbol{X^{(2)}} \approx \begin{pmatrix} 0.407 \\ 2.18 \\ 2.73 \end{pmatrix}$$

$$\boldsymbol{X^{(3)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(2)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \begin{pmatrix} 0.407 \\ 2.18 \\ 2.73 \end{pmatrix} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} = \begin{pmatrix} 0.90 \\ 2.21 \\ 2.91 \end{pmatrix} \rightarrow \boldsymbol{X^{(3)}} \approx \begin{pmatrix} 0.90 \\ 2.21 \\ 2.91 \end{pmatrix}$$

$$\boldsymbol{X^{(4)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(3)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \begin{pmatrix} -2.14 \\ 3.96 \\ -5.33 \end{pmatrix} + \begin{pmatrix} 0.90 \\ 2.21 \\ 2.91 \end{pmatrix} = \begin{pmatrix} 0.92 \\ 2.12 \\ 2.91 \end{pmatrix} \rightarrow \boldsymbol{X^{(4)}} \approx \begin{pmatrix} 0.92 \\ 2.12 \\ 2.91 \end{pmatrix}$$

$$\boldsymbol{X^{(5)}} = \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \boldsymbol{X^{(4)}} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix}$$

$$= \begin{pmatrix} -0.09 & -0.37 & 0.37 \\ 0.02 & 0.02 & -0.52 \\ -0.04 & -0.13 & 0.13 \end{pmatrix} \begin{pmatrix} 0.92 \\ 2.12 \\ 2.91 \end{pmatrix} + \begin{pmatrix} 0.74 \\ 3.58 \\ 2.86 \end{pmatrix} = \begin{pmatrix} 0.94 \\ 2.12 \\ 2.92 \end{pmatrix} \rightarrow \boldsymbol{X^{(5)}} \approx \begin{pmatrix} 0.94 \\ 2.12 \\ 2.92 \end{pmatrix}$$

Thus, the result for the fifth iteration is:

$$\boldsymbol{X^{(5)}} \approx \begin{pmatrix} 0.94 \\ 2.12 \\ 2.92 \end{pmatrix} \textit{ so the solution } X \rightarrow \begin{pmatrix} 1.00 \\ 2.00 \\ 3.00 \end{pmatrix}$$

3.4 Remarks on the Implementation of Iterative Methods

When implementing iterative methods for solving linear systems, several factors can significantly impact their efficiency and effectiveness. Here are some key considerations:

1. Initial Guess:

- **Choice:** The initial guess can significantly influence convergence speed. A good initial guess can accelerate convergence, while a poor one may lead to slow convergence or even divergence.
- **Strategies:** Common strategies include using a zero vector, averaging previous solutions, or leveraging prior knowledge about the system.

1) Zero Vector:

- **Example**: For a system $Ax = b$, starting with $x = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ is a common default. This can work well for many problems but may not be optimal for every system.

2) Averaging Previous Solutions:

This strategy can be effective in iterative algorithms where prior estimates are available. For instance, if previous solutions from a related problem or earlier iterations are known, averaging them can provide a more refined initial guess.

Example: If past solutions were $x^{(k-1)} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ and $x^{(k-2)} = \begin{pmatrix} 0.5 \\ 1.5 \\ 2.5 \end{pmatrix}$, an average can be $x^{(0)} = \frac{1}{2}(x^{(k-1)} + x^{(k-2)}) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

3) Leveraging Prior Knowledge:

If there is existing knowledge about the system or the expected solution range, this can guide the choice of an initial guess. For example, in physical systems modeled by differential equations, parameters often lie within known bounds.

Example: If a temperature distribution in a rod is known to stabilize around a certain value based on physical properties, starting near that temperature can lead to faster convergence.

In summary, the initial guess in iterative methods is crucial for convergence speed and overall effectiveness. By employing strategies such as using a zero vector, averaging previous solutions, or leveraging prior knowledge, one can enhance the chances of rapid convergence to the desired solution.

2. Convergence Criteria:

Defining clear criteria for determining convergence is essential in iterative methods. Convergence criteria help establish when the solution has sufficiently approximated the true answer, allowing the algorithm to terminate.

Common Convergence Criteria

1) Difference Between Successive Iterates:

The most straightforward method involves comparing the difference between successive estimates of the solution. If the difference falls below a predefined tolerance level, the process can be considered converged.

Example: For a vector x, the criterion can be expressed as: $\left\|x^{(k+1)} - x^{(k)}\right\| < \varepsilon$ where ε is a small positive number (e.g., 10^{-6}). If the norm of the difference is smaller than ε, the algorithm stops.

2) **Residual Norm**:

Another common approach is to check the residual of the equation $Ax = b$. If the norm of the residual $r = b - Ax$ is sufficiently small, the solution is deemed converged.

Example: If $\|r\| < \delta$, where δ is another small tolerance (e.g., 10^{-6}), the solution is considered acceptable.

Adaptive Convergence Criteria

Adaptive criteria involve adjusting the convergence tolerance based on the current progress of the solution. This approach can improve efficiency by allowing the algorithm to adapt to the problem's dynamics.

Benefits of Adaptive Criteria

1) **Dynamic Adjustment**:

Instead of a fixed tolerance, the tolerance can be adjusted based on how quickly the solution is approaching convergence. For example, as the iterates get closer to the solution, the tolerance could become stricter.

Example: If the norm of the difference $\left\|x^{(k+1)} - x^{(k)}\right\|$ is decreasing rapidly, the algorithm could tighten the tolerance from 10^{-6} to 10^{-8}.

2) **Performance Improvement**:

Adaptive criteria can lead to faster convergence in some cases, especially for complex or ill-conditioned problems where a static tolerance might not be appropriate.

Example: In an optimization problem, if the function value decreases significantly in one iteration, the tolerance could be relaxed temporarily, allowing for faster exploration of the solution space.

Clearly defining convergence criteria is fundamental in iterative methods, ensuring that the solution process is both effective and efficient. By implementing adaptive criteria, one can tailor the convergence process to the specific dynamics of the problem, enhancing performance and reducing unnecessary computations.

3. Method Selection:

When selecting an iterative method for solving linear systems, the properties of the matrix play a crucial role in determining which algorithm will be most effective. Different methods have strengths and weaknesses depending on the characteristics of the matrix involved.

Key Matrix Properties

1) **Diagonally Dominant Matrices**:

A matrix A is diagonally dominant if for each row, the absolute value of the diagonal entry is greater than or equal to the sum of the absolute values of the other entries in that row.

Method Suitability: Methods like Gauss-Seidel and Jacobi perform well with diagonally dominant matrices due to their guaranteed convergence.

Example: For a matrix $A = \begin{pmatrix} 10 & 1 & -5 \\ 4 & -7 & 2 \\ 1 & 1 & 3 \end{pmatrix}$

each row satisfies the diagonal dominance condition, making it suitable for these methods.

2) **Symmetric Positive Definite Matrices**:

A matrix is symmetric positive definite if it is symmetric ($A = A^T$) and all its eigenvalues are positive.

Method Suitability: Iterative methods like Conjugate Gradient are specifically designed for symmetric positive definite matrices, providing efficient convergence.

Example: A matrix like $A = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}$

is symmetric positive definite, making it ideal for methods like Conjugate Gradient.

3) **Sparse Matrices**:

Many large systems are represented by sparse matrices, which contain a significant number of zero elements.

Method Suitability: Iterative methods such as Krylov subspace methods (e.g., GMRES) are effective for sparse systems, as they can take advantage of the matrix's sparsity to reduce computational costs.

Example: A sparse matrix: $A = \begin{pmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix}$ is well-suited for these methods.

Hybrid Approaches

Combining different iterative methods can leverage their strengths and mitigate their weaknesses, often resulting in improved convergence rates and more robust performance.

Examples of Hybrid Approaches

1) **Using Smoothing Techniques**:

   Combine a direct method (like LU decomposition) with an iterative method. For example, one might use LU to obtain a preliminary solution and then apply SOR to refine it.

   Example: Start with $X^{(0)}$from LU decomposition and then iteratively improve it using SOR.

2) **Multi-Grid Methods**:

   Multi-grid methods involve using different grid levels (coarse to fine) to accelerate convergence. They can effectively reduce errors at multiple scales.

   Example: In solving PDEs (Partial Differential Equation), one can use a coarse grid to solve for global features and then refine the solution on finer grids.

3) **Preconditioning**:

   Preconditioning involves transforming the original problem into a more favorable form before applying an iterative method. This can improve convergence rates significantly.

   Example: Applying an incomplete LU decomposition as a preconditioner for methods like Conjugate Gradient.

4) **Adaptive Strategy**:

   Employ an adaptive method that switches between different iterative techniques based on the convergence behavior observed during iterations.

   Example: Start with Jacobi for initial convergence and switch to Gauss-Seidel for refinement once closer to the solution.

Choosing an appropriate iterative method based on matrix properties is crucial for efficient problem-solving. Additionally, exploring hybrid approaches can capitalize on the strengths of various methods, leading to faster convergence and enhanced robustness. By understanding both the nature of the matrix and the available methods, one can optimize the solution process effectively.

4. Relaxation Parameters:

In iterative methods like Successive Over-Relaxation (SOR), the relaxation parameter ω\omegaω plays a crucial role in optimizing convergence.

1) **Tuning ω**

   Careful tuning of ω can significantly impact convergence speed. For instance, a value around 1.25 often works well for diagonally dominant matrices, enhancing convergence

compared to standard methods. Conversely, setting ω too high or too low can lead to divergence or excessively slow convergence. For example, using $\omega = 0.5$ may result in prolonged iterations, while $\omega = 3$ could cause oscillations.

2) Dynamic Adjustment of ω

Adapting ω during the iteration process can further enhance efficiency. As the solution approaches the true value, adjusting ω can help maintain optimal convergence. For example, starting with $\omega = 1.25$ and gradually reducing it to 1.1 as the iterates stabilize allows for fine-tuning. Implementing a feedback mechanism that monitors convergence rates and adjusts ω\omegaω accordingly can lead to better performance, such as increasing ω when improvements are sluggish.

By carefully tuning and dynamically adjusting the relaxation parameter ω, one can significantly improve the performance of iterative methods like SOR, ensuring faster and more reliable convergence.

5. Matrix Properties:

1) Condition Number

The condition number of a matrix significantly affects the convergence of iterative methods. A poorly conditioned matrix can lead to slow convergence and numerical instability. For example, a matrix with a high condition number (e.g., 1,000) may cause small changes in the input to produce large variations in the output, making it challenging to converge to an accurate solution.

2) Sparsity

Leveraging the sparsity of a matrix is crucial for optimizing computational cost and memory usage. Sparse matrices, which contain a significant number of zero elements, can be represented efficiently, reducing both storage requirements and computational complexity. For instance, using specialized algorithms like Conjugate Gradient can take advantage of matrix sparsity, leading to faster solutions for large-scale problems**.**

In summary, understanding the condition number is essential for anticipating convergence behavior, while utilizing sparsity can enhance the efficiency of iterative methods, ultimately leading to quicker and more resource-efficient computations.

6. Numerical Stability:

1) Rounding Errors

Rounding errors are a critical concern in iterative methods, as they can accumulate during calculations and impact the accuracy of the final solution. For example, in a sequence of iterative updates, small errors introduced at each step can compound, leading to significant deviations from the true solution. This is especially problematic in poorly conditioned matrices, where precision is crucial.

2) Preconditioning

Preconditioning techniques can significantly improve the condition number of a matrix, thereby accelerating convergence. By transforming the original system into a more

favorable form, preconditioners make it easier for iterative methods to find a solution. For instance, applying an incomplete LU decomposition as a preconditioner can lead to faster convergence rates in methods like Conjugate Gradient, particularly for large, sparse systems.

In conclusion, being aware of rounding errors is essential for maintaining solution accuracy, while employing preconditioning techniques can enhance the efficiency of iterative methods by improving the condition number of the matrix and facilitating quicker convergence.

7. Parallelization:

Parallelizing iterative methods can significantly enhance performance by utilizing modern multi-core and distributed computing architectures. For example, in the Jacobi method, each element of the solution can be updated simultaneously, allowing for independent calculations across processors. This parallel approach is particularly beneficial for large-scale problems, where sequential processing would be inefficient.

Similarly, in methods like Conjugate Gradient, the computation of inner products and matrix-vector multiplications can be executed in parallel, leading to faster convergence and better resource utilization.

8. Implementation and Testing:

1) Language and Libraries

Selecting an appropriate programming language and leveraging optimized libraries is crucial for efficient implementation of iterative methods. For example, languages like Python or C++ offer powerful libraries such as NumPy and Eigen, respectively, which provide highly optimized functions for matrix operations. Utilizing these libraries can significantly enhance performance and reduce development time.

2) Testing

Thoroughly testing the implementation with various test cases is essential to ensure accuracy and identify potential issues. For instance, testing the algorithm on known solutions, edge cases, and larger problem sizes can help verify correctness and robustness. This process ensures that the method performs well across different scenarios and maintains stability under various conditions.

In summary, choosing the right programming language and libraries, along with rigorous testing, is vital for the effective implementation of iterative methods. These practices help optimize performance and ensure the reliability of the solution.

By carefully considering these factors and tailoring your implementation accordingly, you can enhance the performance and reliability of iterative methods for solving linear systems.

QCM

Here are multiple-choice questions (MCQs) based on previous section:

Question 1:

What is the impact of a good initial guess in iterative methods for solving linear systems?
A) It has no impact on convergence speed.
B) It can lead to slow convergence or divergence.
C) It can significantly accelerate convergence.
D) It only affects the final solution accuracy.
Answer: C) It can significantly accelerate convergence.

Question 2:

Which of the following is a common strategy for selecting an initial guess in iterative methods?
A) Using the identity matrix.
B) Averaging previous solutions.
C) Randomly generating values.
D) Always using zero as the initial guess.
Answer: B) Averaging previous solutions.

Question 3:

What does a matrix need to be for the Jacobi and Gauss-Seidel methods to guarantee convergence?
A) It must be sparse.
B) It must be symmetric.
C) It must be diagonally dominant or symmetric positive definite.
D) It must have a low condition number.
Answer: C) It must be diagonally dominant or symmetric positive definite.

Question 4:

How can the relaxation parameter ω\omegaω in Successive Over-Relaxation (SOR) be optimized?
A) It should always be set to 1.
B) It should be increased continuously throughout the iterations.
C) It can be tuned carefully to enhance convergence speed.
D) It is irrelevant to the convergence process.
Answer: C) It can be tuned carefully to enhance convergence speed.

Question 5:

What is the purpose of preconditioning techniques in iterative methods?
A) To increase the number of iterations needed for convergence.
B) To transform the problem into a more favorable form and improve the condition number of the matrix.
C) To simplify the matrix to a diagonal form.
D) To eliminate the need for convergence criteria.
Answer: B) To transform the problem into a more favorable form and improve the condition number of the matrix.

3.5 Convergence of Jacobi and Gauss-Seidel Methods

Definition 1

A square matrix $A \in IR_{nxn}$ is said to be ***diagonally dominant*** if, for each row i, the absolute value of the diagonal entry is greater than or equal to the sum of the absolute values of the other entries in that row. Mathematically, this can be expressed in Eq. (43) as:

$$\forall i = 1..n, |a_{ii}| \geq \sum_{j \neq i, j=1..n} |a_{ij}| \qquad (43)$$

for all i If the inequality is strict for at least one row, the matrix is called SDD, ***strictly diagonally dominant***.

Example:

$$A = \begin{pmatrix} 5 & 1 & -1 \\ 2 & 3 & 0 \\ 3 & -1 & -7 \end{pmatrix} \rightarrow \begin{cases} |a_{11}| = 5 & > & |a_{12}| + |a_{13}| = 1 + 1 = 2 \\ |a_{22}| = 3 & > & |a_{21}| + |a_{23}| = 2 + 0 = 2 \rightarrow \\ |a_{33}| = 7 & > & |a_{31}| + |a_{32}| = 3 + 1 = 2 \end{cases} \quad So\ A\ is\ (SDD)$$

Definition 2

The matrix $A \in IR_{nxn}$is symmetric, meaning $A = A^T$ this implies that for $\forall i, j = 1..n, a_{ij} = a_{ji}$

Example:

$$A = \begin{pmatrix} 5 & 2 & -1 \\ 2 & 3 & 0 \\ -1 & 0 & -7 \end{pmatrix} \rightarrow A\ is\ symetric\ \forall i, j = 1..3, a_{ij} = a_{ji}$$

Definition 3

Let $A \in IR_{nxn}$ be a matrix. The principal minors of order k of this matrix are the determinants of the truncated matrices $(a_{ij})_{1 \leq i,j \leq k}$ for $k$ranging from 1 to n.

$$A = \begin{pmatrix} a_{11} a_{12} \dots a_{1n} \\ a_{21} a_{22} \dots a_{2n} \\ \vdots \quad \vdots \quad \vdots \\ a_{n2} a_{n2} \cdots a_{nn} \end{pmatrix} \rightarrow \begin{cases} D_1 = a_{11} \\ D_2 = det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ \vdots \\ D_n = det \begin{pmatrix} a_{11} a_{12} \dots a_{1n} \\ a_{21} a_{22} \dots a_{2n} \\ \vdots \quad \vdots \quad \vdots \\ a_{n2} a_{n2} \cdots a_{nn} \end{pmatrix} \end{cases} \rightarrow \forall i = 1..n, D_i > 0 \qquad (44)$$

Where $D_i$are *Leading principal minors***.** These are the determinants of the square submatrices located in the upper left corner of A.

If all the leading principal minors of A are strictly positive, the matrix is said to be **positive definite**

Example:

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

$$\rightarrow \begin{cases} D_1 = a_{11} = 2 > 0 \\ D_2 = det\begin{pmatrix} 2 & 0 \\ 0 & -1 \end{pmatrix} = -\mathbf{2} < \mathbf{0} \\ D_3 = det\begin{pmatrix} 2 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -2 \end{pmatrix} = 2\begin{vmatrix} -1 & 1 \\ 0 & -2 \end{vmatrix} - 0\begin{vmatrix} 0 & 1 \\ 1 & -2 \end{vmatrix} + 12\begin{vmatrix} 0 & -1 \\ 1 & 0 \end{vmatrix} = 4 + 1 = 5 > 0 \end{cases}$$

Since not all leading principal minors are positive. $D_2 = -2 < 0$, the matrix A is not positive definite.

Definition 4

Matrix norms $\|A\|$ provide a way to measure the size or magnitude of a matrix A. They are defined in terms of the elements of the matrix, denoted as a_{ij}.

1) **Max Norm (Infinity Norm)**:

$$\|A\|_\infty = max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| \tag{45}$$

This norm takes the maximum absolute row sum of the matrix.

2) **1-Norm**:

$$\|A\|_1 = max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}| \tag{46}$$

This norm takes the maximum absolute column sum of the matrix.

Example:

$$A = \begin{pmatrix} 2 & 0 & -3 \\ 0 & -1 & 1 \\ 1 & 5 & -2 \end{pmatrix}$$

$$\|A\|_\infty = max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}| = max_{1 \le i \le 3}(|2| + |-3|, |-1| + |1|, |1| + |5| + |-2|)$$
$$= max_{1 \le i \le 3} \begin{pmatrix} 5 \\ 2 \\ 7 \end{pmatrix} \rightarrow \|A\|_\infty = 7$$

$$\|A\|_1 = max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}| = max_{1 \le i \le 3}(|2| + |1|, |-1| + |5|, |-3| + |1| + |-2|)$$
$$= max_{1 \le j \le 3}(3,6,6) \rightarrow \|A\|_1 = 6$$

Definition 5

The spectral radius of a matrix A is defined as the maximum absolute value of its eigenvalues λ_i. It is denoted in Eq. (47) as:

$$\rho(A) = max|\lambda_i| \tag{47}$$

where λ_i are the eigenvalues of the matrix A. The spectral radius provides important information about the stability and behavior of the matrix, particularly in applications related to dynamical systems and numerical analysis.

Example

Let's consider a simple 2x2 matrix:

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

To find the spectral radius, we first need to compute the eigenvalues of the matrix A.

1) Characteristic Polynomial: The eigenvalues are found by solving the characteristic equation given by: $det(A - \lambda I) = 0$.

   Where I is the identity matrix. For our matrix A:

   $$A - \lambda I = \begin{pmatrix} 4-\lambda & 2 \\ 1 & 3-\lambda \end{pmatrix}$$

2) Determinant Calculation**:**

   $$det(A - \lambda I) = (4-\lambda)(3-\lambda) - (2)(1) = \lambda^2 - 7\lambda + 10 = 0$$

3) Solving the Quadratic Equation:

   Using the quadratic formula $\lambda = \frac{-b \pm \sqrt{b^2-4ac}}{2a} = \frac{-7 \pm \sqrt{49-40}}{2} = \frac{-7 \pm 3}{2}$

   This gives us: $\lambda_1 = 5 \quad and \quad \lambda_2 = 2$

4) Calculating the Spectral Radius:

   Now, we compute the spectral radius: $\rho(A) = max|\lambda_i| = \rho(A) = \max(|5|, |2|) = 5$

3.5.1 Sufficient Conditions for Convergence of Iterative Methods

Iterative methods are widely used for solving linear systems, especially when dealing with large matrices. Among these methods, the Gauss-Seidel method and relaxation methods are particularly noteworthy. Their convergence properties can be influenced by the characteristics of the coefficient matrix A. In this section, we will discuss the sufficient conditions for the convergence of these methods, particularly focusing on the case when A is a strictly symmetric positive definite matrix.

3.5.1.1 Convergence of the Jacobi Method

The Jacobi method is another iterative approach used to solve the equation $AX = b$. Unlike the Gauss-Seidel method, which updates the solution using the latest values, the Jacobi method updates all components simultaneously using values from the previous iteration.

Sufficient Condition for Convergence: If A is strictly **symmetric positive definite**, the Jacobi method converges for any initial guess $X^{(0)}$. The convergence can be attributed to the following factors:

- Similar to the Gauss-Seidel method, the spectral radius $\rho(\boldsymbol{B_j})$ of the iteration matrix $\boldsymbol{B_j}$ associated with the Jacobi method is less than 1 ($\rho(\boldsymbol{B_j}) < 1$). This guarantees that the error decreases with each iteration.

3.5.1.2 Convergence of the Gauss-Seidel Method

The **Gauss-Seidel method** is an iterative approach that updates each component of x based on the most recent values, leveraging previously computed results.

Sufficient Condition for Convergence: If A is strictly symmetric positive definite, the Gauss-Seidel method converges for any initial guess $X^{(0)}$. This is due to:

- The spectral radius $\rho(\boldsymbol{B_{GS}})$ of the iteration matrix $\boldsymbol{B_{GS}}$ being less than 1 ($\rho(\boldsymbol{B_{GS}}) < 1$), ensuring that the error diminishes with each iteration.

3.5.1.3 Convergence of the Relaxation Method

The **relaxation method** is a generalization of the Gauss-Seidel method that introduces a relaxation factor ω.

Sufficient Condition for Convergence: For the relaxation method to converge, the relaxation factor ω must satisfy: $0<\omega<2$

When A is strictly symmetric positive definite, choosing ω\omegaω within the range (0,2) enhances the convergence speed. Specifically:

- When $\omega=1$, the method reduces to Gauss-Seidel.
- When $0<\omega<1$, the method may converge more slowly but still guarantees convergence.
- When $1<\omega<2$, the method often converges more quickly than standard Gauss-Seidel.

In summary, the conditions for convergence of the Jacobi method, Gauss-Seidel method, and the relaxation method are closely linked to the properties of the coefficient matrix A. Specifically, if A is strictly symmetric positive definite, all three methods will converge. For the relaxation method, selecting an appropriate relaxation factor ω (where $0<\omega<2$) is crucial for achieving optimal convergence rates. These insights are fundamental in numerical analysis and provide a strong basis for the effective implementation of iterative methods in practical applications.

Example:

To compute successive approximations of the solution of a system using the Jacobi and Gauss-Seidel methods, let's define a system of linear equations as follows:

$$A = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \; and \; b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{cases} ax + by = 1 \\ bx + ay = 1 \end{cases} \rightarrow \begin{cases} x^{k+1} = \frac{1}{a}(1 - by^k) \\ y^{k+1} = \frac{1}{a}(1 - bx^k) \end{cases}$$

Study of Convergence

1) Let's start with the **sufficient conditions**.

- $\forall i = 1..2, |a_{ii}| \geq \sum_{j \neq i, j=1..n} |a_{ij}| \rightarrow \begin{cases} |a| > |b| \\ |a| > |b| \end{cases}$

  the matrix A is said to be (**SDD) *strictly diagonally dominant*** if $|a| > |b|$

The Jacobi and Gauss-Seidel methods converges if $|a| > |b|$.

- $A = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \; and \; A^T = \begin{pmatrix} a & b \\ b & a \end{pmatrix}$

  1) Since $A = A^T$, A is **symmetric**.
  2) the matrix A is said to be **positive definite** if all the leading principal minors of A are strictly positive $\forall i = 1..2, D_i > 0$.
  1. $D_1 = a_{11} = a > 0$
  2. $D_2 = \det A = \begin{vmatrix} a & b \\ b & a \end{vmatrix} = a^2 - b^2 > 0 \rightarrow \; |a| > |b|$

The Gauss-Seidel method converges if $|a| > |b|$.

2) Let's start with the **Necessary & Sufficient Conditions**

$$D = \begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}, E = \begin{pmatrix} 0 & 0 \\ -b & 0 \end{pmatrix}, F = \begin{pmatrix} 0 & -b \\ 0 & 0 \end{pmatrix}$$

Jacobi Method

$$\boldsymbol{B_j} = D^{-1}(E + F) \; and \; C = D^{-1}b$$

$$D^{-1} = \begin{pmatrix} 1/a & 0 \\ 0 & 1/a \end{pmatrix} \; and \; (E + F) = \begin{pmatrix} 0 & -b \\ -b & 0 \end{pmatrix}$$

$$\boldsymbol{B_j} = \begin{pmatrix} 0 & -b/a \\ -b/a & 0 \end{pmatrix} \; and \; C = \begin{pmatrix} 1/a \\ 1/a \end{pmatrix}$$

$$X^{(K+1)} = D^{-1}(E + F)\, X^{(K)} + D^{-1}b$$

$$det(\boldsymbol{B_j} - \lambda I) = \det\begin{pmatrix} -\lambda & -\frac{b}{a} \\ -\frac{b}{a} & -\lambda \end{pmatrix} = \left(\lambda - \frac{b}{a}\right)\left(\lambda + \frac{b}{a}\right) = 0$$

$$det(\boldsymbol{B_j} - \lambda I) = 0 \rightarrow \lambda_1 = \frac{b}{a}\ and\ \lambda_2 = -\frac{b}{a}$$

The Jacobi method converges when $\rho(\boldsymbol{B_j}) = max|\lambda_i| < 1$

$$\rho(\boldsymbol{B_j}) = \max(|\lambda_1|, |\lambda_2|) = \left|\frac{b}{a}\right| < 1 \rightarrow |b| < |a|$$

The Jacobi method converges if $|\boldsymbol{a}| > |\boldsymbol{b}|$.

Gauss Seidel Method

$\boldsymbol{B_{GS}} = (D - E)^{-1}(F)\ and\ \ C = (D - E)^{-1}b$

$(D - E)^{-1} = \begin{pmatrix} 1/a & 0 \\ -b/a^2 & 1/a \end{pmatrix}\ and\ F = \begin{pmatrix} 0 & -b \\ 0 & 0 \end{pmatrix}$

$\boldsymbol{B_{GS}} = \begin{pmatrix} 0 & -b/a \\ 0 & b^2/a^2 \end{pmatrix}$

$X^{(K+1)} = (D - E)^{-1}(F)\, X^{(K)} + (D - E)^{-1}b$

$$det(\boldsymbol{B_{GS}} - \lambda I) = \det\begin{pmatrix} -\lambda & -b/a \\ 0 & b^2/a^2 - \lambda \end{pmatrix} = (-\lambda)\left(\frac{b^2}{a^2} - \lambda\right) = 0$$

$$det(\boldsymbol{B_{GS}} - \lambda I) = 0 \rightarrow \lambda_1 = 0\ and\ \lambda_2 = \frac{b^2}{a^2}$$

The Jacobi method converges when $\rho(\boldsymbol{B_{GS}}) = max|\lambda_i| < \left|\frac{b^2}{a^2}\right|$

$$\rho(\boldsymbol{B_{GS}}) = \max(|\lambda_1|, |\lambda_2|) = \left|\frac{b^2}{a^2}\right| < 1 \rightarrow |b| < |a|$$

The Gauss Seidel method converges if $|\boldsymbol{a}| > |\boldsymbol{b}|$.

Chapter 4 Computation of Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are fundamental concepts in linear algebra with numerous applications in science and engineering. This course section will cover the localization of eigenvalues and the power method for their computation.

4.1 Localization of Eigenvalues

Definition

Eigenvalues are scalar values associated with a square matrix A that satisfy the Eq. (48):

$$A\mathbf{x} = \boldsymbol{\lambda}\mathbf{x} \tag{48}$$

Where $\boldsymbol{\lambda}$ is an eigenvalue and x is the corresponding eigenvector.

The equation states that when the matrix A acts on the vector x, the output is a scaled version of x. In other words, A transforms x by merely stretching or compressing it without changing its direction.

Example;

$A = \begin{pmatrix} 3 & 0 \\ 8 & -1 \end{pmatrix}$ and x=$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$

$$Ax = \begin{pmatrix} 3 & 0 \\ 8 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix} = 3 \times \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Thus, the eigenvalue $\boldsymbol{\lambda = 3}$ and the associated eigenvector is $x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

4.1.1 Finding Eigenvalues: Analytical calculation

To find the eigenvalues of a matrix A, we rearrange the equation (48) into the following form:

$$\mathrm{Ax} - \lambda \mathrm{x} = 0 \tag{49}$$

This can be rewritten as:

$$(A - \lambda I)x = 0 \tag{50}$$

where I is the identity matrix of the same size A. For non-trivial solutions (i.e., x≠0), the determinant of $(A - \lambda I)$must be zero:

$$\boldsymbol{det(A - \lambda I) = 0} \tag{51}$$

he equation $\boldsymbol{det(A-\lambda I)=0}$ is known as the characteristic equation of the matrix A The solutions to this polynomial equation give the eigenvalues λ1,λ2,…,λn .

Example

Consider the matrix:

$$A=\begin{pmatrix}10 & 0 & 0\\ 1 & -3 & -7\\ 0 & 2 & 6\end{pmatrix}$$

To find the Eigenvalues and eigenvectors, we first need to compute the eigenvalues of the matrix A.

1) Characteristic Polynomial: The eigenvalues are found by solving the characteristic equation given by: $det(A-\lambda I)=0$.

   Where I is the identity matrix. For our matrix A:

$$A=\begin{pmatrix}10 & 0 & 0\\ 1 & -3 & -7\\ 0 & 2 & 6\end{pmatrix} \quad and\ I=\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

$$A-\lambda I=\begin{pmatrix}10-\lambda & 0 & 0\\ 1 & -3-\lambda & -7\\ 0 & 2 & 6-\lambda\end{pmatrix}$$

2) Determinant Calculation**:**

$$det(A-\lambda I)=\det\begin{pmatrix}10-\lambda & 0 & 0\\ 1 & -3-\lambda & -7\\ 0 & 2 & 6-\lambda\end{pmatrix}=(10-\lambda)[(-3-\lambda)(6-\lambda)+14]$$
$$=(\lambda-10)[\lambda^2-3\lambda-4]=(\lambda-10)(\lambda+1)(\lambda-4)$$

3) Solving the Quadratic Equation: $(\lambda-10)(\lambda+1)(\lambda-4)=0$

   This gives us: $\lambda_1=10\ ,\ \lambda_2=-1\ and\quad \lambda_3=4$

   The Eigenvalues are $\boldsymbol{\lambda_1=10}$, $\boldsymbol{\lambda_2=-1\ and}$ $\boldsymbol{\lambda_3=4}$

4) Calculating the eigenvectors:

   Now, we compute the eigenvectors:

   For $\boldsymbol{\lambda_1=10}$

$$Ax=\lambda_1 x\rightarrow\begin{pmatrix}10 & 0 & 0\\ 1 & -3 & -7\\ 0 & 2 & 6\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}=10\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}\rightarrow\begin{cases}10x_1=10x_1\\ x_1-3x_2-7x_3=10x_2\\ 2x_2+6x_3=10x_3\end{cases}$$
$$\rightarrow\begin{cases}x_1=1\\ x_2=2/33\\ x_3=1/33\end{cases}\rightarrow X_1=\begin{pmatrix}1\\ 2/33\\ 1/33\end{pmatrix}$$

For $\lambda_2 = -1$

$$Ax = \lambda_2 x \rightarrow \begin{pmatrix} 10 & 0 & 0 \\ 1 & -3 & -7 \\ 0 & 2 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = -1\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{cases} 10x_1 = -x_1 \\ x_1 - 3x_2 - 7x_3 = -x_2 \\ 2x_2 + 6x_3 = -x_3 \end{cases}$$

$$\rightarrow \begin{cases} x_1 = 0 \\ x_2 = 1 \\ x_3 = -2/7 \end{cases} \rightarrow X_2 = \begin{pmatrix} 0 \\ 1 \\ -2/7 \end{pmatrix}$$

For $\lambda_3 = 4$

$$Ax = \lambda_3 x \rightarrow \begin{pmatrix} 10 & 0 & 0 \\ 1 & -3 & -7 \\ 0 & 2 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 4\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{cases} 10x_1 = 4x_1 \\ x_1 - 3x_2 - 7x_3 = 4x_2 \\ 2x_2 + 6x_3 = 4x_3 \end{cases}$$

$$\rightarrow \begin{cases} x_1 = 0 \\ x_2 = -1 \\ x_3 = 1 \end{cases} \rightarrow X_3 = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$$

The Eigenvectors are $X_1 = \begin{pmatrix} 1 \\ 2/33 \\ 1/33 \end{pmatrix}, X_2 = \begin{pmatrix} 0 \\ 1 \\ -2/7 \end{pmatrix}$ ***and*** $X_3 = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$

Unfortunately, this approach is generally not recommended, as there's no reliable and quick way to find the roots of a polynomial with a degree higher than 4. Thus, we investigate numerical methods for solving the eigenvalue problem

4.1.2 Localization Techniques

Localization methods help to determine the approximate locations of eigenvalues without computing them directly. Some common techniques include:

4.1.2.1 Gershgorin Circle Theorem

The Gershgorin Circle Theorem states that every eigenvalue of a matrix $A = [a_{ij}]$ lies within at least one of the Gershgorin disks defined by Eq. (52):

$$D_i = \{z \in C: |z - a_{ij}| \leq \sum_{j \neq i} |a_{ij}|\} \quad (52)$$

This means that for each row i we can draw a circle in the complex plane centered at $a_{ij}$with a radius equal to the sum of the absolute values of the other entries in the row.

The Gerschgorin theorem states that all the eigenvalues of a matrix A belong to the union of n disks. If the Gerschgorin disks are all disjoint, then each one contains exactly one eigenvalue.

Notations

1) All the eigenvalues of A are located within the union of the disks.
2) The i^{th} disk is defined with center a_{ij} and radius r_i is given by Eq. (53):

$$r_i = \sum_{j\neq i}^{n} |\, a_{ij}|$$ (53)

The Gerschgorin theorem provides a valuable way to localize eigenvalues of a matrix. Each Gerschgorin disk can be visualized in the complex plane, centered at the diagonal entry a_{ij} of the matrix and extending outwards by the radius r_i.

For example, if you have a matrix:

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

the disks are calculated as follows:

1. **For i=1**

   - Center: : $b_{11} = 1$

     The Gerschgorin disks according to the rows:

     Radius: $r_1 = |0| + |-3| = 3$

     Disk: $D_1 \rightarrow (1,3)$

     The Gerschgorin disks according to the columns

     Radius: $r_1 = |2| + |1| = 3$

     Disk: $D_1 \rightarrow (1,3)$

2. **For i=2**:

   Center: $b_{22} = 3$

   The Gerschgorin disks according to the rows:

   Radius: : $r_2 = |2| + |1| = 3$

   Disk: $D_2 \rightarrow (3,3)$

   The Gerschgorin disks according to the columns:

   Radius: : $r_2 = |0| + |0| = 0$

   Disk: $D_2 \rightarrow (3,0)$

3. **For i=3**

   Center: $b_{33} = -2$

The Gerschgorin disks according to the rows:

Radius: : $r_3 = |1| + |0| = 1$

Disk: $D_3 \rightarrow (-2,1)$

The Gerschgorin disks according to the columns:

Radius: : $r_3 = |-3| + |1| = 4$

Disk: $D_3 \rightarrow (-2,4)$

Knowing that B and B^T share the same eigenvalues, we choose the smallest radius for each Gerschgorin disk. This provides the following localization of the spectrum.

$$D_1 \rightarrow (1,3)\ , \qquad D_2 \rightarrow (3,0) \qquad and \quad D_3 \rightarrow (-2,1)$$

Figure 4 shows the localization of the real eigenvalues and complex eigenvalues.

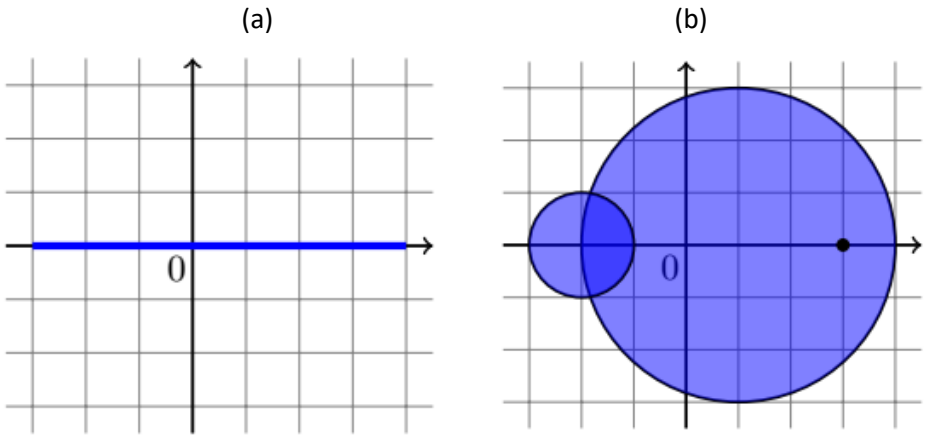

Figure.4 *Localization of the real eigenvalues (a) and complex eigenvalues (b).*

By employing analytical calculations, we were able to accurately determine the eigenvalues, which were then used to verify localization.

Determinant Calculation**:**

$$\text{Det}(B - \lambda I) = \det\begin{bmatrix} 1-\lambda & 0 & -3 \\ 2 & 3-\lambda & 1 \\ 1 & 0 & -2-\lambda \end{bmatrix}$$

$$= (1-\lambda)\big[\big((3-\lambda)(-2-\lambda) - 0\big)\big] + 0 - [3(3-\lambda)]$$

$$= (1-\lambda)[(\lambda-3)(\lambda-2)] + 3(\lambda-3)]$$

$$= (\lambda-3)[-(\lambda-1)(\lambda-2) + 3]$$

$=(\lambda-3)[-(\lambda^2-3\lambda+2)+3]$

$=-(\lambda-3)[\lambda^2+\lambda+1]$

Solving the Quadratic Equation:

$-(\lambda-3)[\lambda^2+\lambda+1]=0 \rightarrow\ \lambda-3=0\ \ or\ \ \ \lambda^2+\lambda+1=0$

The solutions are $\lambda_1=3,\ \lambda_2=-\frac{1}{2}+\frac{\sqrt{3}}{2}i, and\ \lambda_3=-\frac{1}{2}-\frac{\sqrt{3}}{2}$i . Figure 4 (b) confirms the localization of the eigenvalues..

Exercise: Consider the following matrix

$$A=\begin{pmatrix} 30 & 1 & 2 & 3 \\ 4 & 15 & -4 & -2 \\ -1 & 0 & 3 & 5 \\ -3 & 5 & 0 & -1 \end{pmatrix}$$

1) Calculate the Gershgorin disks.
2) Draw the Gershgorin disks in the complex plane.
3) Determine the possible localization of the eigenvalues of matrix A based on the Gershgorin disks.

In addition to the Gershgorin Circle Theorem, there are several other localization methods for determining eigenvalues. ***Interval Analysis*** is one such method; for instance, using ***Sturm's theorem***, we can ascertain the number of eigenvalues within a specified interval by analyzing the roots of the characteristic polynomial. This helps in refining the intervals that contain the eigenvalues.

Furthermore, various ***numerical methods*** can be utilized to obtain eigenvalues with greater precision after initial localization. Techniques like the ***QR algorithm*** offer a reliable way to compute eigenvalues more accurately, enhancing our understanding of the spectral properties of matrices.

4.2 Power Method

The Power Method is one of the simplest techniques for calculating the eigenvalues of a matrix A. This iterative method is particularly useful for finding the eigenvalue with the largest absolute value, known as the dominant eigenvalue, along with its corresponding eigenvector.

Theorem

Let A be a square matrix of size $n \times n$ that has N eigenvalues. The eigenvalues are defined as the values $\lambda_1, \lambda_2\ \ldots,\ \lambda_n$ that satisfy the characteristic equation Eq. (54):

$$\left|\boldsymbol{\lambda_1}\right| > |\boldsymbol{\lambda_2}| \geq\ \ldots \geq |\boldsymbol{\lambda_n}| \tag{54}$$

An eigenvalue λ_1 of a matrix A is said to be **dominant** if its absolute value is greater than the absolute values of all other eigenvalues of A.

Let $X^{(0)}$be a suitably chosen vector, then the sequences of vectors in Eq. (55)

$$\left\{X^{(k)} = \left[x_1^{(k)}, x_2^{(k)} \dots, x_n^{(k)}\right]\right\} \tag{55}$$

And the sequence of scalars C_k generated in Eq. (56) where Any vector X in R^n can be expressed as:$\sum_{i=1}^{n} c_i x_i$.

$$x^{(1)} = Ax^{(0)} = \sum_{i=1}^{n} c_i A x_i = \sum_{i=1}^{n} c_i \lambda_i x_i \tag{56}$$

$$x^{(k)} = A^k x^{(0)} = \sum_{i=1}^{n} c_i (\lambda_i)^k x_i = \lambda_1^k \left[c_i x_i + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_2 \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n\right]$$

Converges respectively to the dominant eigenvector $\boldsymbol{v_1}$ and the eigenvalue λ_1.

NB. The dominant eigenvalue plays a crucial role in various numerical methods and applications, particularly in iterative methods for solving systems of linear equations or finding eigenvalues themselves. The dominant eigenvalue significantly impacts various aspects of numerical analysis and system behavior. In iterative methods like the Power Method, it governs the convergence rate, allowing the method to quickly approach the dominant eigenvalue and its associated eigenvector, while other eigenvalues influence the process to a lesser extent. Furthermore, in dynamic systems, the magnitude of the dominant eigenvalue serves as an indicator of stability: if it is less than 1, the system is stable, whereas if it exceeds 1, the system becomes unstable. Additionally, the dominant eigenvalue provides valuable insights into the long-term behavior of processes modeled by matrices, including applications in population dynamics, Markov chains, and iterative algorithms.

Iterative Process

The method proceeds as follows:

1) **Initialization**: Choose an initial vector $X^{(0)}$ (often randomly).
2) **Iteration**: For k=0,1,2,…

   $x^{(1)} = Ax^{(0)} = \sum_{i=1}^{n} c_i A x_i = \sum_{i=1}^{n} c_i \lambda_i x_i$

   Here, c_i are the coefficients corresponding to the eigenvalues λi and eigenvectors x_i.

3) **Normalization**: To avoid overflow or underflow, normalize $x^{(1)}$:

   $$Y^{(k+1)} = c^{(k+1)} X^{(k)}$$

   where $c^{(k+1)} = \max_{1 \le i \le N}\left\{\left|x_i^{(k)}\right|\right\}$, This ensures that the components of the vector $Y^{(k+1)}$ are scaled to maintain numerical stability.

4) **Update the Iterative Vector**:

   $$X^{(k+1)} = \frac{1}{c^{(k+1)}} X^{(k)}$$

Here, ${c_i}^{(k+1)} = {x_j}^{(k)}$ is the coefficient that contributes to the eigenvector approximation.

5) **Convergence Check**: Estimate the dominant eigenvalue using the Rayleigh quotient:

$$\lambda \approx \frac{X^{(k)T} A X^{(k)}}{X^{(k)T} X^{(k)}}$$

Repeat until the change in $X^{(k)}$ is smaller than a predefined tolerance.

Example:

Applying the Power Method to Matrix A

Given the matrix A=$\begin{bmatrix} 10 & 0 & 0 \\ 1 & -3 & -7 \\ 0 & 2 & 6 \end{bmatrix}$ and the initial vector $x^{(0)}$=$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

we will apply the Power Method to find the dominant eigenvalue and its corresponding eigenvector.

Iterative Process

1. **First Iteration**:

   A$x^{(0)}$=$\begin{bmatrix} 10 & 0 & 0 \\ 1 & -3 & -7 \\ 0 & 2 & 6 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}=\begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix}$

   Normalization: Calculate the maximum element for normalization:

   $$c^{(1)} = \max_{1 \le i \le N} \{|{x_i}^{(1)}|\} = 10$$

   Normalize $x^{(1)}$:

   $$Y^{(1)} = \frac{X^{(1)}}{c^{(1)}} = \frac{1}{10}\begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0{,}1 \\ 0 \end{bmatrix}$$

   Update**:**

   $$X^{(1)} = Y^{(1)}.c^{(1)} = \begin{bmatrix} 1 \\ 0{,}1 \\ 0 \end{bmatrix}.10 = \begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix}$$

2. **Second Iteration**:

   $x^{(2)}$ =A$x^{(1)}$=$\begin{bmatrix} 10 & 0 & 0 \\ 1 & -3 & -7 \\ 0 & 2 & 6 \end{bmatrix}\begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix}=\begin{bmatrix} 100 \\ -20 \\ 12 \end{bmatrix}$

$$c^{(2)} = \max_{1 \le i \le N} \{|{x_i}^{(1)}|\} = 100$$

Normalize $x^{(2)}$:

$$Y^{(2)} = \frac{X^{(2)}}{c^{(2)}} = \frac{1}{100}\begin{bmatrix}100\\-20\\12\end{bmatrix} = \begin{bmatrix}1\\-0{,}2\\0.12\end{bmatrix}$$

Update**:**

$$X^{(2)} = Y^{(2)}.c^{(2)} = \begin{bmatrix}1\\-0{,}2\\0.12\end{bmatrix}.100 = \begin{bmatrix}100\\-20\\12\end{bmatrix}$$

3. Third Iteration:

$$x^{(3)} = \mathrm{A}x^{(2)} = \begin{bmatrix}10 & 0 & 0\\1 & -3 & -7\\0 & 2 & 6\end{bmatrix}\begin{bmatrix}100\\-20\\12\end{bmatrix} = \begin{bmatrix}1000\\-140\\72\end{bmatrix}$$

$$c^{(3)} = \max_{1\le i\le N}\left\{\left|x_i{}^{(1)}\right|\right\} = 1000$$

Normalize $x^{(2)}$:

$$Y^{(3)} = \frac{X^{(3)}}{c^{(3)}} = \frac{1}{1000}\begin{bmatrix}1000\\-140\\72\end{bmatrix} = \begin{bmatrix}1\\-0{,}14\\0.072\end{bmatrix}$$

Update**:**

$$X^{(2)} = Y^{(2)}.c^{(2)} = \begin{bmatrix}1\\-0{,}14\\0.072\end{bmatrix}.1000 = \begin{bmatrix}1000\\-140\\72\end{bmatrix}$$

Rayleigh Quotient

Finally, we can estimate the dominant eigenvalue using the Rayleigh quotient:

$$\lambda \approx \frac{X^{(2)T}AX^{(2)}}{X^{(2)T}X^{(k2)}} = \frac{[100 \quad -20 \quad 12]\begin{bmatrix}1000\\-140\\72\end{bmatrix}}{[100 \quad -20 \quad 12]\begin{bmatrix}100\\-20\\12\end{bmatrix}} = \frac{103664}{10444} \approx 9.91$$

The Power Method has shown that the dominant eigenvalue of matrix A is approximately $\lambda \approx 9.91$, with an associated eigenvector that converges through the iterative process.

Exercise:

Consider the following matri x and Initial Vector

$$A=\begin{pmatrix}5 & 4 & 2\\ 2 & 3 & 1\\ 1 & 2 & 3\end{pmatrix} \text{ and } X^{(0)}=\begin{pmatrix}1\\ 1\\ 1\end{pmatrix}$$

1) Apply the Power Method to find the dominant eigenvalue and its corresponding eigenvector of matrix A.
2) Perform two iterations of the Power Method and calculate the Rayleigh quotient to estimate the dominant eigenvalue.

Chapter 5 Matrix Analysis

5.1 Vector Spaces

Definition

A vector space V over a field F is a mathematical structure composed of a set of elements called vectors, which adhere to specific properties. These properties ensure that vector operations are well-defined and consistent. Below are the key properties that characterize a vector space:

- **Closure under Addition**: For any two vectors $u, v \in V$, their sum $u + v$ must also be in V. This property ensures that adding vectors together results in another vector within the same space.
- **Closure under Scalar Multiplication**: For any vector $u \in V$ and any scalar $c \in F$, the product cu must also be in V. This indicates that multiplying a vector by a scalar does not take it outside the vector space.
- **Associativity of Addition**: Vector addition is associative, meaning that for any vectors $u, v, w \in V$:

  $$(u + v) + w = u + (v + w) \quad (57)$$

  This property ensures that the grouping of vectors during addition does not affect the outcome.

- **Commutativity of Addition**: Addition of vectors is commutative, so for any vectors $u, v \in V$:

  $$u + v = v + u \quad (58)$$

  This means the order in which vectors are added does not change the result.

- **Identity Element of Addition**: There exists a zero vector $0 \in V$ such that for any vector $u \in V$:

  $$u + 0 = u \quad (59)$$

  The zero vector serves as the additive identity.

- **Inverse Elements of Addition**: For every vector $u \in V$, there exists a vector $-u \in V$ such that:

  $$u + (-u) = 0 \quad (60)$$

  This property guarantees that for every vector, there is a corresponding *"opposite"* vector that sums to the zero vector.

- **Distributive Property**: Scalar multiplication distributes over vector addition and scalar addition:

  $$c(u + v) = cu + cv$$

$$(c + d)u = cu + du \quad (61)$$

This means that scaling a sum of vectors is equivalent to scaling each vector and then adding.

- **Associativity of Scalar Multiplication**: Scalar multiplication is associative, so for any scalars $c, d \in F$ and any vector $u \in V$:

$$c(du) = (cd)u \quad (62)$$

This ensures that the order of scalar multiplication does not affect the result.

- **Identity Element of Scalar Multiplication**: The scalar multiplication by the identity scalar (1) yields the vector itself:

$$1u = u \quad (63)$$

This property confirms that multiplying a vector by 1 leaves it unchanged.

Example

Consider the vector space R^2 over the field of real numbers R. Elements of this vector space are vectors of the form $\binom{x}{y}$where $x, y \in R$.

- **Closure under Addition**: If $u = \binom{x_1}{y_1}$ and $v = \binom{x_2}{y_2}$, then $u + v = \binom{x_1 + x_2}{y_1 + y_2}$ in R^2.
- **Closure under Scalar Multiplication**: For $c \in R$ and $u = \binom{x}{y}$, the product $cu = \binom{cx}{cy}$ remains in R^2.

This foundational structure provides the basis for many areas in mathematics, including linear algebra, functional analysis, and beyond.

5.2 Matrices

Definition

A matrix is a rectangular array of numbers arranged in rows and columns. It is commonly used to represent linear transformations or systems of linear equations. Each element in the matrix corresponds to a specific position defined by its row and column indices.

Notation

A matrix A of size $n \times m$ has m rows and n columns, denoted as:

$$A = \begin{pmatrix} a_{11} a_{12} \dots a_{1m} \\ a_{21} a_{22} \dots a_{2m} \\ \vdots \quad \vdots \quad \vdots \\ a_{n1} a_{n2} \cdots a_{nm} \end{pmatrix} \quad (64)$$

Here, a_{ij} represents the element located in the i row and j column of matrix A.

Matrices are fundamental in various mathematical fields, including linear algebra, computer science, and statistics, where they facilitate operations such as addition, multiplication, and finding determinants and eigenvalues.

5.2.1 Matrix Operations

5.2.1.1 Addition

Definition

Two matrices $A \in R_{nxm}$ and $B \in R_{nxm}$ can be added if they have the same dimensions: The addition of two matrices A and B results in a new matrix C, where each element c_{ij} is calculated by adding the corresponding elements of A and B:

$$C = A + B = \left[c_{ij} = a_{ij} + b_{ij}\right]_{n\times m}$$

$$= \begin{pmatrix} a_{11} + b_{11}\, a_{12} + b_{12} \ldots a_{1m} + b_{1m} \\ a_{21} + b_{21}\, a_{22} + b_{22} \ldots a_{2m} + b_{2m} \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \\ a_{n1} + b_{n1} a_{n2} + b_{n2} \cdots\ a_{nm} + b_{nm} \end{pmatrix} \tag{65}$$

- **Example:**

1) If we have: $A = \begin{pmatrix} 1 & -3 \\ -2 & -1 \end{pmatrix}$ $and\ B = \begin{pmatrix} -5 & 3 \\ 7 & 2 \end{pmatrix}$

Then the sum $C = A + B$ is:

$$C = A + B = \begin{pmatrix} 1-5 & -3+3 \\ -2+7 & -1+2 \end{pmatrix} = \begin{pmatrix} -4 & 0 \\ 5 & 1 \end{pmatrix}$$

2) If we have $A = \begin{pmatrix} 1 & -3 & 0 \\ 0 & -1 & 5 \end{pmatrix}$ $and\ B = \begin{pmatrix} 2 & 7 \\ 1 & 0 \end{pmatrix}$

Matrix A is of size 2×3 and matrix B is of size 2×2. Since their dimensions do not match, we cannot add them directly.

5.2.1.2 Scalar Multiplication

Definition

Scalar multiplication involves multiplying each element of a matrix A by a scalar c.

If A is a matrix of size $n \times m$ given by:

$$A = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{1m} \\ a_{21}\, a_{22} \ldots a_{2m} \\ \vdots \quad \vdots \quad\ \vdots \\ a_{n1} a_{n2} \cdots\ a_{nm} \end{pmatrix} \tag{66}$$

then the result of multiplying A by the scalar$c$ is another matrix cAdefined as:

$$c.A = \begin{pmatrix} c.a_{11} & c.a_{12}\dots & c.a_{1m} \\ c.a_{21} & c.a_{22}\dots & c.a_{2m} \\ \vdots & \vdots & \vdots \\ c.a_{n1} & c.a_{n2}\cdots & c.a_{nm} \end{pmatrix} \quad (67)$$

Notation

Properties of Scalar Multiplication:

1) Distributive Property: $c(A+B) = cA + cB$
2) Associative Property: $c(dA) = (cd)A$
3) Identity Element: $1A = A$

Example

Let $A = \begin{pmatrix} 1 & -5 \\ 0 & -1 \end{pmatrix}$ and let $c = 3$.

Then,

$$cA = 3\begin{pmatrix} 1 & -5 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 3\times 1 & 3\times(-5) \\ 3\times 0 & 3\times(-1) \end{pmatrix} = \begin{pmatrix} 3 & -15 \\ 0 & -3 \end{pmatrix} \quad .$$

This demonstrates how each element of the matrix A is multiplied by the scalar c.

5.2.1.3 Matrix Multiplication

Definition

The product of two matrices AB is defined when the number of columns in matrix A is equal to the number of rows in matrix B. Specifically, if A is an $m \times n$ matrix and B is an $n \times p$ matrix, then the resulting product AB will be an $m \times p$ matrix.

Notation

- Let A be an $m \times n$ matrix:

$$A = \begin{pmatrix} a_{11} & a_{12}\dots & a_{1n} \\ a_{21} & a_{22}\dots & a_{2n} \\ \vdots & \vdots & \vdots \\ a_{m1} & a_{m2}\cdots & a_{mn} \end{pmatrix} \quad (68)$$

- Let B be an $n \times p$ matrix:

$$B = \begin{pmatrix} b_{11} & b_{12}\dots & b_{1p} \\ b_{21} & b_{22}\dots & b_{2p} \\ \vdots & \vdots & \vdots \\ b_{n1} & b_{n2}\cdots & b_{np} \end{pmatrix} \quad (69)$$

Calculation of the Product AB

The product AB is calculated by taking the dot product of the rows of A with the columns of B. The element in the i^{th} row and j^{th} column of the resulting matrix $C = AB$ is given by:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad (68)$$

where:

- c_{ij} is the entry in the i^{th} row and j^{th}column of the matrix C.
- a_{ik} is the entry from the i^{th} row of A.
- b_{kj} is the entry from the j^{th} column of B.

Notation

Properties of Matrix Multiplication

1) Non-Commutativity: In general, $AB \neq BA$.
2) Associativity: $(AB)C = A(BC)$.
3) Distributive: $A(B + C) = AB + AC$.

Example

Let:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \; and \; B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

The product AB is calculated as follows: $AB = \begin{pmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$

Matrix multiplication is a fundamental operation in linear algebra, with wide-ranging applications in various fields including computer science, engineering, and economics. Understanding the conditions for multiplication, the calculation method, and properties is essential for further exploration of linear transformations and systems of equations.

5.2.2 Relationships between Linear Mappings and Matrices

Definition

Every linear mapping $T: R^n \rightarrow R^m$ can be represented by a matrix A such that:

$$T(x) = Ax \quad (69)$$

for all $x \in R^n$. Here, A is an $m \times n$ matrix where the action of T on a vector x corresponds to the matrix multiplication of A and x.

5.2.2.1 Representation of the Matrix

If T is defined by its effect on the standard basis vectors $e_1, e_2, \ldots, e_n$ of R^n the columns of the matrix A are given by:

$$A = (T(e_1)\ T(e_2)\ \ldots\ T(e_n)) \tag{70}$$

where each $T(e_2)$ is an m-dimensional vector.

Notation

The properties are :

1) Linearity: The matrix A captures the linearity of the mapping T, meaning:
   $T(c_1x_1 + c_2x_2) = c_1T(x_1) + c_2T(x_2)$ for any scalars c_1, c_2 and vectors $x_1, x_2 \in R^n$.

2) Composition: If $T_1: R^n \to R^m$ and $T_2: R^n \to R^m$ are linear mappings with matrices A_1 and A_2, respectively, then the composition $T_2 \circ T_1$ can be represented by the matrix product A_2A_1.

Understanding the relationship between linear mappings and matrices allows for the translation of abstract linear transformations into concrete matrix operations, facilitating analysis and computation in various mathematical and applied contexts.

Example

Let $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$ and $A = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Calculation: $T(x) = Ax = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

5.2.3 Inverse of a Matrix

The inverse A^{-1} of a matrix A satisfies the following conditions:

$$AA^{-1} = I\ and\ A^{-1}A = I \tag{71}$$

$$A^{-1} = \frac{1}{\det\ (A)} \times adj(A)$$

where I is the identity matrix and $adj(A) is$ Adjugate.

The adjugate (or adjoint) of a matrix A, is the transpose of the cofactor matrix of A. It is calculated by:

- Finding the cofactor for each element of A.
- Transposing the resulting matrix of cofactors.

Example

For

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}$$

Finding A^{-1} involves calculating the determinant and the adjugate. we will follow these steps:

1. Calculate the Determinant of A.
2. Find the Adjugate of A.
3. Use the formula for the inverse.

$$\det(A) = 1\begin{vmatrix} 1 & 4 \\ 6 & 0 \end{vmatrix} - 2\begin{vmatrix} 0 & 4 \\ 5 & 0 \end{vmatrix} + 3\begin{vmatrix} 0 & 1 \\ 5 & 6 \end{vmatrix} = -24 + 40 - 15 = 1$$

Calculate the Cofactor Matrix

To find the cofactor C_{ij}, we calculate the determinant of the 2×2 matrix obtained by deleting the i row and j column and apply the sign based on the position.

$$C_{ij} = (-1)^{i+j} det(i, j)$$

1. **Cofactor** C_{11} (delete row 1, column 1):

$$C_{11} = det\begin{pmatrix} 1 & 4 \\ 6 & 0 \end{pmatrix} = -24$$

2. **Cofactor** C_{12} (delete row 1, column 2):

$$C_{12} = -det\begin{pmatrix} 0 & 4 \\ 5 & 0 \end{pmatrix} = 20$$

3. **Cofactor** C_{13} (delete row 1, column 3):

$$C_{13} = det\begin{pmatrix} 0 & 1 \\ 5 & 6 \end{pmatrix} = -16$$

4. **Cofactor** C_{21} (delete row 2, column 1):

$$C_{21} = -det\begin{pmatrix} 2 & 3 \\ 6 & 0 \end{pmatrix} = 18$$

5. **Cofactor** C_{22} (delete row 2, column 2):

$$C_{22} = det\begin{pmatrix} 1 & 4 \\ 6 & 0 \end{pmatrix} = -15.$$

6. **Cofactor** C_{23} (delete row 2, column 3):

$$C_{23} = -det\begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} = 4$$

7. **Cofactor** C_{31} (delete row 3, column 1):

$$C_{31} = det\begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} = 5$$

8. **Cofactor** C_{32} (delete row 3, column 2):

$$C_{32} = -det\begin{pmatrix} 1 & 3 \\ 0 & 4 \end{pmatrix} = -4$$

9. **Cofactor** C_{33} (delete row 3, column 3):

$$C_{33} = -det\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} = 1$$

. Putting it all together, the cofactor matrix , and Taking the transpose of the cofactor matrix :

$$C = \begin{pmatrix} -24 & 20 & -5 \\ 18 & -15 & 4 \\ 5 & -4 & 1 \end{pmatrix} \rightarrow \boldsymbol{adj(A)} = \boldsymbol{C^T} = \begin{pmatrix} \boldsymbol{-24} & \boldsymbol{18} & \boldsymbol{5} \\ \boldsymbol{20} & \boldsymbol{-15} & \boldsymbol{-4} \\ \boldsymbol{-5} & \boldsymbol{4} & \boldsymbol{1} \end{pmatrix}$$

Using the formula for the inverse:

$A^{-1} = \frac{1}{\det(A)} \times adj(A)$

Since $det(A) = 1$, The inverse of the matrix A is:

$$A^{-1} = \begin{pmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{pmatrix}$$

5.2.4 Trace and Determinant of a Matrix

Definition 1

The trace of a square matrix A, denoted $tr(A)$, is defined as the sum of the diagonal elements:

$$\boldsymbol{tr(A) = a_{11} + a_{22} + \cdots + a_{nn}} \quad (72)$$

Properties of the Trace

1) Linearity: $tr(A + B) = tr(A) + tr(B)$ for any two square matrices A and B of the same size.
2) Scalar Multiplication**:** $tr(cA) = ctr(A)$ for any scalar c.
3) Transpose $tr(A^T) = tr(A)$.

Definition 2

The determinant of a square matrix A, denoted $det(A)$, is a scalar value that provides important properties regarding the matrix, including whether it is invertible.

Properties of the Determinant

1) Multiplicative: $det(AB) = det(A).det(B)$ for any two square matrices A and B.
2) Invertibility: A matrix A is invertible if and only if $\det(A) \neq 0$.
3) Effect of Row Operations:

- Swapping two rows multiplies the determinant by -1.
- Multiplying a row by a scalar c multiplies the determinant by C.
- Adding a multiple of one row to another does not change the determinant.

5.2.5 Eigenvalues and Eigenvectors

Definition

For a square matrix A, a non-zero vector v is called an eigenvector and the corresponding scalar λ is called an eigenvalue if *(see chapter 4*):

$$Av = \lambda v \tag{73}$$

5.2.5.1 Finding Eigenvalues

To find the eigenvalues of A, solve the characteristic equation:

$$det(A - \lambda I) = 0 \tag{74}$$

where I is the identity matrix of the same size as A.

5.2.5.2 Finding Eigenvectors

Once the eigenvalues are determined, substitute each eigenvalue λ back into the equation $(A - \lambda I)v = 0$ to find the corresponding eigenvectors.

Properties

1) Sum of Eigenvalues: The sum of the eigenvalues of A equals the trace of A.
2) Product of Eigenvalues**:** The product of the eigenvalues equals the determinant of A.

5.2.6 Similar Matrices

Definition

Two square matrices A and B are said to be similar if there exists an invertible matrix P such that:

$$B = P^{-1}AP \tag{75}$$

Properties of Similar Matrices

1) Same Eigenvalues**:** Similar matrices share the same eigenvalues.
2) Same Determinant and Trace: If A and B are similar, then $det(A) = det(B) \; and \; tr(A) = tr(B)$).
3) Invariant under Similarity**:** Many properties of matrices, such as rank and characteristic polynomial, are invariant under similarity.

5.2.7 Some Special Matrices

Definition 1

A diagonal matrix is a square matrix in which all elements outside the main diagonal are zero. It can be represented as:

$$D = \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & d_n \end{pmatrix}$$

where $d_1, d_2, .. d_n$ are the diagonal entries.

Properties

1) **Eigenvalues:** The eigenvalues of a diagonal matrix are simply its diagonal entries:

   Eigenvalues=$d_1, d_2, .. d_n$

2) **Determinant:** The determinant of a diagonal matrix is the product of its diagonal entries:

   $\det(A) = d_1 \times d_2 \times, .\times d_n$

3) **Inverse:** If $d_i \neq 0$ for all i, the inverse of a diagonal matrix D is also a diagonal matrix:

$$D^{-1} = \begin{pmatrix} 1/d_1 & 0 & \dots & 0 \\ 0 & 1/d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1/d_n \end{pmatrix}$$

Definition 2

A matrix Q is orthogonal if its transpose is equal to its inverse:

$$Q^T Q = I$$

Properties

1) Orthonormal Vectors: The columns (and rows) of Q are orthonormal vectors. This means:

   - Each column vector has a length of 1.
   - Any two different columns are orthogonal to each other.

2) **Determinant:** The determinant of an orthogonal matrix is either +1 or −1:

$$det(Q) = \pm 1$$

3) **Inverse:** The inverse of an orthogonal matrix is its transpose:

$$Q^{-1} = Q$$

Definition 3

A matrix A is **symmetric** if it is equal to its transpose:

$$A = A^T$$

Properties

1) Real Eigenvalues: All eigenvalues of a symmetric matrix are real.
2) Diagonalization: There exists an orthogonal matrix Q such that:

$$A = Q\Lambda Q^T$$

where Λ is a diagonal matrix containing the eigenvalues of A.

3) Quadratic Form: For any vector x: $x^T A\ x$ is a vreal number.

5.3 Norms and Inner Products

Norms and inner products are fundamental concepts in linear algebra that provide a way to measure the size of vectors and the angle between them. These tools are essential for various applications in mathematics, physics, and engineering.

5.3.1 Definitions

Vector Norms

A norm is a function that assigns a non-negative length or size to vectors in a vector space. It is denoted as $\| x \|$\for a vector x.

Properties of Norms

For any vector x and scalar c:

1) Non-negativity: $\| x \| \geq 0$ and $\| x \| = 0$ if and only if $x = 0$.
2) Scalar multiplication**:** $\| cx \| = | c | \| x \|$.
3) Triangle inequality: $\| x + y \| \leq \| x \| + \| y \|$.

Common Norms

1. **1-Norm (Manhattan Norm)**: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
2. **2-Norm (Euclidean Norm)**: $\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

3. **Infinity Norm**: $\|x\|_\infty = \max_i |x_i|$

5.3.2 Inner Products and Vector Norms

Inner Product

An inner product is a generalization of the dot product that allows us to define angles and lengths in a vector space. For two vectors $x, y \in R^n$, the inner product is denoted as $\langle x, y \rangle$.

Properties of Inner Products

1) Conjugate symmetry: $\langle x, y \rangle = \overline{\langle y, x \rangle}$
2) Linearity**:** $\langle cx + z, y \rangle = c\langle x, y \rangle + \langle z, y \rangle$
3) Positive definiteness**:** $\langle x, x \rangle \geq 0$ and $\langle x, x \rangle = 0$ if x=0.

Relationship Between Norms and Inner Products

The norm of a vector can be derived from the inner product:

$$\|x\| = \sqrt{(x, x)}$$

Examples of Inner Products

1) Standard Inner Product: $\langle x, y \rangle = \sum_{i=1}^{n} x_i \, y_i$
2) Weighted Inner Product**:** $\langle x, y \rangle = \sum_{i=1}^{n} w_i x_i \, y_i$ where wi>0w_i > 0wi>0 are weights.

5.3.3 Matrix Norms

Definition

A matrix norm is a function that assigns a non-negative size to matrices, analogous to vector norms. It is denoted as $\| A \|$ for a matrix A.

Properties of Matrix Norms

For any matrices A and B of appropriate dimensions and scalar c:

1) Non-negativity: $\| A \| \geq 0$ and $\| A \| = 0$ if and only if A is the zero matrix.
2) Scalar multiplication**:** $\| cA \| = | c | \| A \|$.
3) Triangle inequality: $\| A + B \| \leq \| A \| + \| B \|$.

Common Matrix Norms

1. **Frobenius Norm**:

   $$\| A \|_F = \sqrt{\sum_{i,j} |a_{ij}|^2} = \sqrt{tr(A^* A)}$$

   where $A^* *$ is the conjugate transpose of A.

2. **1-Norm**:

$$\| A \|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$$

This norm is the maximum absolute column sum of the matrix.

3. **Infinity Norm**:

$$\| A \|_\infty = \max_{1 \le j \le m} \sum_{i=1}^{n} |a_{ij}|$$

This norm is the maximum absolute row sum of the matrix.

4. **2-Norm (Spectral Norm)**:

$$\| A \|_2 = \sigma_{max}$$

where σ_{max} is the largest singular value of A.

Applications of Norms and Inner Products

- **Stability Analysis**: In control theory, norms are used to analyze the stability of systems.
- **Optimization**: In machine learning, norms help in regularization techniques to prevent overfitting.
- **Numerical Analysis**: Norms measure error and convergence rates of numerical methods.

QCM

Here are multiple-choice questions (MCQs) based on previous section:

Question 1:

1. Which of the following is NOT a property of a vector space?
A) Closure under addition
B) Existence of a zero vector
C) Commutativity of multiplication
D) Closure under scalar multiplication
Answer: C) Commutativity of multiplication

Question 2:

What is the result of the matrix multiplication $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$?.
A) $\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
B) $\begin{pmatrix} 31 & 52 \\ 15 & 24 \end{pmatrix}$
C) $\begin{pmatrix} 14 & 12 \\ 11 & 3 \end{pmatrix}$
D) $\begin{pmatrix} 26 & 7 \\ 19 & 32 \end{pmatrix}$
Answer: $\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$.

Question 3:

If A is a matrix representing a linear mapping, which of the following statements is true?
A) The matrix has the same dimensions as the vector space it maps to.
B) The rank of the matrix is always equal to its dimension.
C) Every linear mapping can be represented by a matrix.
D) The inverse of a matrix always represents a linear mapping.
Answer: C) Every linear mapping can be represented by a matrix.

Question 4:

What is the determinant of the matrix $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$?
A) -2.
B) 2.
C) 4.
D) 1.
Answer: A) -2.

Question 5:

Which of the following is a property of inner products in vector spaces?
A) It is commutative but not associative.
B) It can produce a scalar result.
C) It requires at least three vectors.
D) It can be negative.
Answer: B) It can produce a scalar result.

References

Timothy Sauer, "*Numerical Analysis*", Publisher: PearsonEdition, November 26, 2011 ISBN-0321783670 (2nd edition).

Francois Cuvelier, "*Analyse numérique élémentaire",* course note, Université Paris XIII, 2022.

Jaan Kiusalaas*, "Numerical Methods in Engineering with Python",* Publisher: Cambridge University Press. (2nd edition).

Richard L. Burden and J. Douglas Faires, "*Numerical Analysis*", Publisher: Cengage Learning, Publication: January 1, 2015 (Edition10)

Steven C. Chapra and Raymond P. Canale, "*Numerical Methods for Engineers*", Publisher: McGraw Hill, Publication: March 3, 2020 (Edition 8)

Lloyd N. Trefethen and David Bau III, "*Numerical Linear Algebra"*, Publisher: SIAM-Society for Industrial and Applied Mathematics, 1997.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, "*Numerical Recipes: The Art of Scientific Computing*", Publisher: Cambridge University Press, Publication: date6 September 2007. (Edition 3)