Numerical Linear Algebra - Notes - v0.2.0-dev

260236

November 2024

Preface

Every theory section in these notes has been taken from the sources:

- Course slides. [1]

About:

GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on numerical linear algebra. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

Contents

1 Preliminaries **4**
1.1 Notation 4
1.2 Matrix Operations 5
1.3 Basic matrix decomposition 7
1.4 Determinants 9
1.5 Sparse matrices 10
1.5.1 Storage schemes 10

2 Iterative methods for linear systems of equations **14**
2.1 Why not use the direct methods? 14
2.2 Linear iterative methods 16
2.2.1 Definition 16
2.2.2 Jacobi method 19
2.2.3 Gauss-Seidel method 20
2.2.4 Convergence of Jacobi and Gauss-Seidel methods 21
2.2.5 The stationary Richardson method 23
2.3 Stopping Criteria 26

Index **29**

1 Preliminaries

This section introduces some of the basic topics used throughout the course.

1.1 Notation

We try to use the same notation for anything.

- **Vectors**. With $\mathbb{R}$ is a set of real numbers (scalars) and $\mathbb{R}^n$ is a space of column vectors with n real elements.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

  Vectors with all zeros and all ones:

$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

- **Matrices**. With $\mathbb{R}^{m \times n}$ is a space of $m \times n$ matrices with real elements:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

  Identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_n \end{bmatrix}$$

  Where $\mathbf{e}_i$, $i = 1, 2, \ldots, n$ are the canonical vectors.

$$\mathbf{e}_i = \begin{bmatrix} 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \end{bmatrix}^T$$

  Where 1 is the i-th entry.

1.2 Matrix Operations

Some basic matrix operations:

- **Inner products**. If $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ then:

$$\mathbf{x}^T\mathbf{y} = \sum_{i=1,\ldots,n} x_i y_i$$

For real vectors, the commutative property is true:

$$\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x}$$

Furthermore, the vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are **orthogonal** if:

$$\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x} = \mathbf{0}$$

And finally, some useful properties of matrix multiplication:

1. Multiplication by the *identity* changes nothing.

$$A \in \mathbb{R}^{n\times m} \;\Rightarrow\; \mathbf{I}_n A = A = A\mathbf{I}_m$$

2. Associativity:

$$A\,(BC) = (AB)\,C$$

3. Distributive:

$$A\,(B + D) = AB + AD$$

4. No commutativity:

$$AB \neq BA$$

5. Transpose of product:

$$(AB)^T = B^T A^T$$

- **Matrix powers**. For $A \in \mathbb{R}^{n\times n}$ with $A \neq \mathbf{0}$:

$$A^0 = \mathbf{I}_n \qquad A^k = \underbrace{A \cdots A}_{k \text{ times}} = AA^{k-1} \qquad k \geq 1$$

Furthermore, $A \in \mathbb{R}^{n\times n}$ is:

  - **Idempotent** (projector) $A^2 = A$
  - **Nilpotent** $A^k = \mathbf{0}$ for some integer $k \geq 1$

- **Inverse**. For $A \in \mathbb{R}^{n\times n}$ is **non-singular** (**invertible**), if exists A^{-1} with:

$$AA^{-1} = \mathbf{I}_n = A^{-1}A \tag{1}$$

Inverse and transposition are interchangeable:

$$A^{-T} \triangleq \left(A^T\right)^{-1} = \left(A^{-1}\right)^T$$

Furthermore, an inverse of a product for a matrix $A \in \mathbb{R}^{n\times n}$ can be expressed as:

$$(AB)^{-1} = B^{-1}A^{-1}$$

Finally, remark that if $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^n$ and $A\mathbf{x} = 0$, then A is **singular**.

- **Orthogonal matrices**. Given a matrix $A \in \mathbb{R}^{n \times n}$ that is *invertible*, the matrix A is said to be **orthogonal** if:

$$A^{-1} = A^T \Rightarrow A^T A = \mathbf{I}_n = A A^T$$

- **Triangular matrices**. There are two types of triangular matrices:

  1. **Upper triangular matrix**:

$$\mathbf{U} = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{n,n} \end{bmatrix}$$

     $\mathbf{U}$ is **non-singular** if and only if $u_{ii} \neq 0$ for $i = 1, \ldots, n$.

  2. **Lower triangular matrix**:

$$\mathbf{L} = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix}$$

     $\mathbf{L}$ is **non-singular** if and only if $l_{ii} \neq 0$ for $i = 1, \ldots, n$.

- **Unitary triangular matrices**. Are matrices similar to the lower and upper matrices, but they have the main diagonal composed of ones.

  1. **Unitary upper triangular matrix**:

$$\mathbf{U} = \begin{bmatrix} 1 & u_{1,2} & \cdots & u_{1,n} \\ 0 & 1 & \cdots & u_{2,n} \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

  2. **Unitary lower triangular matrix**:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix}$$

1.3 Basic matrix decomposition

In the Numerical Linear Algebra course, we will use three main decomposition:

- **LU factorization with (partial) pivoting**. If $A \in \mathbb{R}^{n \times n}$ is a non-singular matrix, then:
$$PA = LU$$
Where:
  - P is a permutation matrix
  - L is an unit lower triangular matrix
  - U is an upper triangular matrix

  Note that the linear system solution:
$$A\mathbf{x} = \mathbf{b}$$
  Can be solved directly by calculation:
$$PA = LU$$
  This way the complexity is equal to $O\left(n^3\right)$. So a smarter way to reduce complexity is to use the *divide et impera* (or *divide and conquer*) technique. Then solve the system:
$$\begin{cases} L\mathbf{y} = P\mathbf{b} & \rightarrow \text{ unit lower triangular system, complexity } O\left(n^2\right) \\ U\mathbf{x} = \mathbf{y} & \rightarrow \text{ upper triangular system, complexity } O\left(n^2\right) \end{cases}$$

- **Cholesky decomposition**. If $A \in \mathbb{R}^{n \times n}$ is a symmetric[1] and positive definite[2], then:
$$A = L^T L$$
  Where L is a lower triangular matrix (with positive entries on the diagonal). Also note that the linear system solution:
$$A\mathbf{x} = \mathbf{b}$$
  Can be solved directly by calculation:
$$A = L^T L$$
  This way the complexity is equal to $O\left(n^3\right)$. So a smarter way to reduce complexity is to use the *divide et impera* (or *divide and conquer*) technique. Then solve the system:
$$\begin{cases} L^T\mathbf{y} = \mathbf{b} & \rightarrow \text{ lower triangular system, complexity } O\left(n^2\right) \\ L\mathbf{x} = \mathbf{y} & \rightarrow \text{ upper triangular system, complexity } O\left(n^2\right) \end{cases}$$

[1] $A^T = A$
[2] $\mathbf{z}^T A\mathbf{z} > 0 \qquad \forall \mathbf{z} \neq 0$

- **QR decomposition**. If $A \in \mathbb{R}^{n \times n}$ is a non-singular matrix, then:

$$A = QR$$

Where:

  - Q is an orthogonal matrix
  - R is an upper triangular

Note that the linear system solution:

$$A\mathbf{x} = \mathbf{b}$$

Can be solved directly by calculation:

$$A = QR$$

This way the complexity is equal to $O\left(n^3\right)$. So a smarter way to reduce complexity is to use the *divide et impera* (or *divide and conquer*) technique. Then:

1. Multiply $\mathbf{c} = Q^T\mathbf{b}$, complexity $O\left(n^2\right)$
2. Solve the lower triangular system $R\mathbf{x} = \mathbf{c}$, complexity $O\left(n^2\right)$

1.4 Determinants

We will assume that the determinant topic is well known. However, in the following enumerated list there are some useful properties about the determinant of a matrix:

1. If a general matrix $T \in \mathbb{R}^{n \times n}$ is upper- or lower-triangular, then the determinant is computed as:

$$\det(T) = \prod_{i=1}^{n} t_{i,i}$$

2. Let $A, B \in \mathbb{R}^{n \times n}$, then is true:

$$\det(AB) = \det(A) \cdot \det(B)$$

3. Let $A \in \mathbb{R}^{n \times n}$, then is true:

$$\det\left(A^T\right) = \det(A)$$

4. Let $A \in \mathbb{R}^{n \times n}$, then is true:

$$\det(A) \neq 0 \iff A \text{ is non-singular}$$

5. **Computation**. Let $A \in \mathbb{R}^{n \times n}$ be non-singular, then:
   (a) Factor $PA = LU$
   (b) $\det(A) = \pm \det(U) = \pm u_{1,1} \dots u_{n,n}$

1.5 Sparse matrices

A **sparse matrix** is a matrix in which most of the elements are zero; roughly speaking, given $A \in \mathbb{R}^{n \times n}$, the number of non-zero entries of A (denoted $\text{nnz}(A)$) is $O(n)$, we say that A is **sparse**.

Sparse matrices are so important because when we try to solve:

$$A\mathbf{x} = \mathbf{b}$$

The A matrix is often sparse, especially when it comes from the discretization of partial differential equations.

Finally, note that the iterative methods (explained in the next section) only use a sparse matrix A in the context of the matrix-vector product. Then we only need to provide the matrix-vector product to the computer.

1.5.1 Storage schemes

Unfortunately, storing a sparse matrix is a waste of memory. Instead of storing a dense array (with many zeros), the main idea is to **store only the non-zero entries, plus their locations**.

This technique allows to save data storage because it will be from $O(n^2)$ to $O(\text{nnz})$.

The most common sparse storage types are:

- **Coordinate format (COO)**. The data structure consists of three arrays (of length $\text{nnz}(A)$):
  - `AA`: all the values of the non-zero elements of A in any order.
  - `JR`: integer array containing their row indices.
  - `JC`: integer array containing their column indices.

  For **example**:

$$A = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

$$\texttt{AA} = [12.\ \ 9.\ \ 7.\ \ 5.\ \ 1.\ \ 2.\ \ 11.\ \ 3.\ \ 6.\ \ 4.\ \ 8.\ \ 10.]$$

$$\texttt{JR} = [\ 5\ \ 3\ \ 3\ \ 2\ \ 1\ \ 1\ \ 4\ \ 2\ \ 3\ \ 2\ \ 3\ \ 4\ ]$$

$$\texttt{JC} = [\ 5\ \ 5\ \ 3\ \ 4\ \ 1\ \ 4\ \ 4\ \ 1\ \ 1\ \ 2\ \ 4\ \ 3\ ]$$

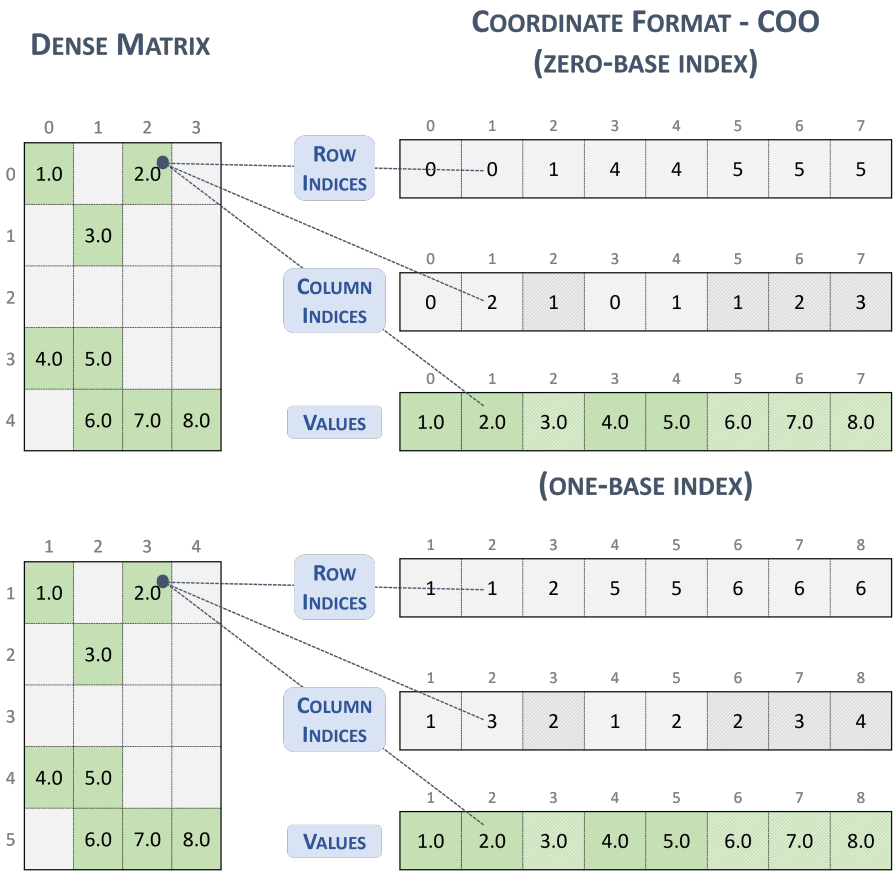

Figure 1: Graphical representation of the coordinate format (COO) technique. From the figure we can see the representation of the `AA` array, called *values*, the `JR`, called *row indices*, and finally the `JC`, called *column indices*. The algorithm is very simple. The figures are taken from the NVIDIA Performance Libraries Sparse, which is part of the NVIDIA Performance Libraries.

- **Coordinate Compressed Sparse Row format (CSR)**. If the elements of A are listed by row, the array `JC` might be replaced by an array that points to the beginning of each row.
  - `AA`: all the values of the non-zero elements of A, stored row by row from $1, \ldots, n$.
  - `JA`: contains the column indices.
  - `IA`: contains the pointers to the beginning of each row in the arrays A and `JA`. Thus `IA`(i) contains the position in the arrays `AA` and `JA` where the i-th row starts. The length of `IA` is $n+1$, with `IA`$(n+1)$ containing the number $A(1) + \mathrm{nnz}(A)$. Remember that n is the number of rows.

For **example**:

$$A = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

$$\texttt{AA} = [1.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.]$$

$$\texttt{JA} = [1\ \ 4\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\,]$$

$$\texttt{IA} = [1\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\,]$$

To retrieve each position of the matrix, the algorithm is quite simple. Consider the `IA` arrays.

1. We start at position one of the array, then the value 1:

   $$\texttt{AA} = [1.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.]$$
   $$\texttt{JA} = [1\ \ 4\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\,]$$
   $$\texttt{IA} = [\textcircled{1}\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\,]$$

2. We use the value one to see the first (index one) position of the array JA, and the value is 1:

   $$\texttt{AA} = [1.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.]$$
   $$\texttt{JA} = [\textcircled{1}\ \ 4\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\,]$$
   $$\texttt{IA} = [1\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\,]$$

3. But with the same index of `IA`, you also check the array `AA`, which has a value of 1:

   $$\texttt{AA} = [\textcircled{1}.\ \ 2.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.]$$
   $$\texttt{JA} = [1\ \ 4\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\,]$$
   $$\texttt{IA} = [1\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\,]$$

4. Now we can check the next row of the matrix. So we check the array `IA` at position 2 and get the value 3. But be careful! From 1 (the previously calculated value) to 3 (the value just taken) there is the value 2 in between. So we can assume that the value 2 is also in the first row.

   $$\texttt{AA} = [1.\ \ \textcircled{2}.\ \ 3.\ \ 4.\ \ 5.\ \ 6.\ \ 7.\ \ 8.\ \ 9.\ \ 10.\ \ 11.\ \ 12.]$$
   $$\texttt{JA} = [1\ \ \textcircled{4}\ \ 1\ \ 2\ \ 4\ \ 1\ \ 3\ \ 4\ \ 5\ \ 3\ \ 4\ \ 5\,]$$
   $$\texttt{IA} = [1\ \ 3\ \ 6\ \ 10\ \ 12\ \ 13\,]$$

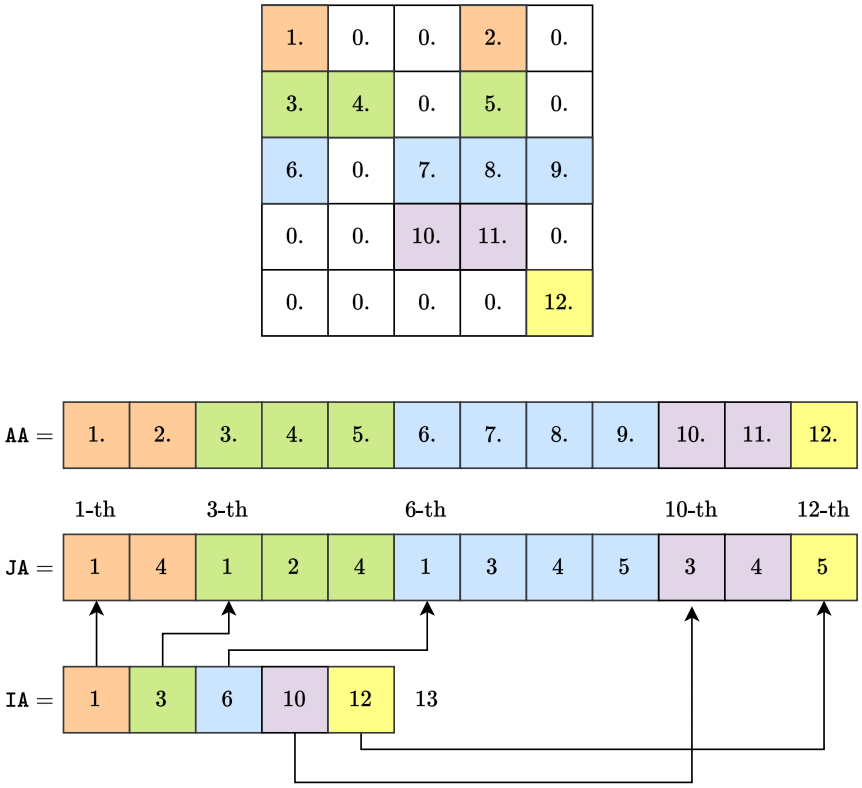

Figure 2: View an illustration of the CRS technique using colors to improve readability.

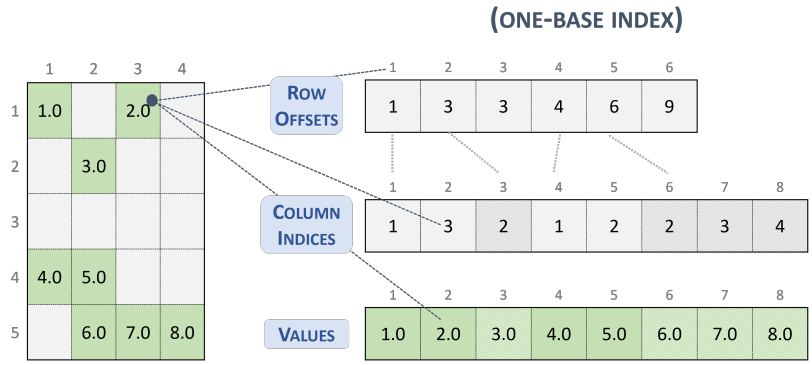

Figure 3: Graphical representation of the coordinate compressed sparse row (CSR) technique. From the figure we can see the representation of the `AA` array, called *values*, the `IA`, called *row offset*, and finally the `JA`, called *column indices*. It's interesting to see how the empty line case is handled. It copies the previous value of the array. The figures are taken from the NVIDIA Performance Libraries Sparse, which is part of the NVIDIA Performance Libraries.

2 Iterative methods for linear systems of equations

2.1 Why not use the direct methods?

Let us considering the following linear system of equations:

$$Ax = b$$

Where $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ and $\det(A) \neq 0$. In general, direct methods are **not very suitable whenever**:

- n **is large**. Typically, the average cost of direct methods scales as n^3, except in selected cases. As a trivial example, if peak performance is 1 PetaFLOPS (10^{15} floating point operations per second), then

$$n = 10^7 \rightarrow \approx 10^6 \text{ seconds} \approx 11 \text{ days}$$

- **Matrix** A **is sparse**. Direct methods suffer from the *fill-in* phenomenon[3] (see later). Unfortunately, sparse matrices are very popular in many application problems and we cannot consider them.

Definition 1: Sparse Matrix

Let $A \in \mathbb{R}^{n \times n}$ we say that A is **sparse** the number of non-zero elements (abbreviated as $\text{nnz}(A)$) is approximately equal to the number of rows/columns n, i.e. $\text{nnz}(A) \sim n$.

What is an iterative method?

It is clear that iterative methods are usually better than direct methods. An **iterative method** is a **mathematical procedure that uses an initial value to generate a sequence of improving approximate solutions to a class of problems**, where the i-th approximation (called an "*iteration*") is derived from the previous ones.

More precisely, we introduce a sequence $\mathbf{x}^{(k)}$ of vectors determined by a recursive relation that identifies the method.

$$\mathbf{x}^{(0)} \rightarrow \mathbf{x}^{(1)} \rightarrow \cdots \rightarrow \mathbf{x}^{(k)} \rightarrow \mathbf{x}^{(k+1)} \rightarrow \cdots$$

To "*initialize*" the iterative process, it is necessary to provide a starting point (*initial vector*, also called *initial guess*) $\mathbf{x}^{(0)}$, e.g. based on physical/engineering applications.

[3] The fill-in of a matrix are those entries that change from an initial zero to a non-zero value during the execution of an algorithm. To reduce the memory requirements and the number of arithmetic operations used during an algorithm, it is useful to minimize the fill-in.

After initialization, the core of the process should, sooner or later, produce a result. It is a very complex and long topic, but in general it refers to the process by which an iterative algorithm approaches a fixed point or a solution to a problem after several iterations. An **iterative method must satisfy the convergence property**:

$$\lim_{k \to +\infty} \mathbf{x}^{(k)} = \mathbf{x} \tag{2}$$

It is important to note that the **convergence does not depend on the choice of the initial vector** $x^{(0)}$.

From the property 2, it should be clear that **convergence is guaranteed only after an ∞ number of iterations**. From a practical point of view, we need to stop the iteration process after a finite number of iterations when we are *sufficiently close* to the solution.

In addition to the *problem of convergence* and "*when should we stop our convergence method*", we have to deal with the *numerical error* inevitably introduced by our method.

These topics will be explained and faced in the following pages.

2.2 Linear iterative methods

2.2.1 Definition

In general, we consider linear iterative methods of the following form:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f} \qquad k \geq 0$$

Where $B \in \mathbb{R}^{n\times n}$, $\mathbf{f} \in \mathbb{R}^n$ and the matrix B is called **iteration matrix**. The choice of the iteration matrix and $\mathbf{f}$ uniquely identifies the method.

The question is now automatic. **How to choose** an intelligent iteration matrix and $\mathbf{f}$? There are two main factors to consider:

- **Consistency**. This is a necessary condition, but not sufficient to guarantee the convergence. If $\mathbf{x}^{(k)}$ es the exact solution $\mathbf{x}$, then $\mathbf{x}^{(k+1)}$ is again equal to $\mathbf{x}$ (no update if the exact solution is found):

  $$\mathbf{x} = B\mathbf{x} + \mathbf{f} \longrightarrow \mathbf{f} = (I - B)\,\mathbf{x} = (I - B)\,A^{-1}\mathbf{b}$$

  The former identity gives a relationship between B and $\mathbf{f}$ as a function of the data.

- **Convergence**. To study the convergence we need the error and the spectral radius:

  - **Error**. Let us introduce the error at step $(k+1)$:

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

    And an appropriate vector norm, such as the Euclidean norm $||\cdot||$. Then we have:

    $$\begin{aligned}
    \left|\left|\mathbf{e}^{(k+1)}\right|\right| &= \left|\left|\mathbf{x} - \mathbf{x}^{(k+1)}\right|\right| \\
    &= \left|\left|\mathbf{x} - \left(B\mathbf{x}^{(k)} + \mathbf{f}\right)\right|\right| \\
    &= \left|\left|\mathbf{x} - B\mathbf{x}^{(k)} - \mathbf{f}\right|\right| \\
    &= \left|\left|\mathbf{x} - B\mathbf{x}^{(k)} - (I - B)\,\mathbf{x}\right|\right| \\
    &= \left|\left|\mathbf{x} - B\mathbf{x}^{(k)} - I\mathbf{x} + B\mathbf{x}\right|\right| \\
    &= \left|\left|\mathbf{x} - B\mathbf{x}^{(k)} - \mathbf{x} + B\mathbf{x}\right|\right| \\
    &= \left|\left|-B\mathbf{x}^{(k)} + B\mathbf{x}\right|\right| \\
    &= \left|\left|B\left(\mathbf{x} - \mathbf{x}^{(k)}\right)\right|\right| \\
    &= \left|\left|B\mathbf{e}^{(k)}\right|\right| \\
    &\leq ||B|| \cdot \left|\left|\mathbf{e}^{(k)}\right|\right|
    \end{aligned}$$

    Note that $||B||$ is the matrix norm induced by the vector norm $||\cdot||$.

Using recursion, we get:

$$\begin{aligned}
\left|\left|\mathbf{e}^{(k+1)}\right|\right| &\leq ||B|| \cdot \left|\left|\mathbf{e}^{(k)}\right|\right| \\
&\leq ||B|| \cdot ||B|| \cdot \left|\left|\mathbf{e}^{(k-1)}\right|\right| \\
&\leq ||B|| \cdot ||B|| \cdot ||B|| \cdot \left|\left|\mathbf{e}^{(k-2)}\right|\right| \\
&\leq \dots \\
&\leq ||B||^{(k+1)} \cdot \left|\left|\mathbf{e}^{(0)}\right|\right| \\
\lim_{k\to\infty} \left|\left|\mathbf{e}^{(k+1)}\right|\right| &\leq \left(\lim_{k\to\infty} ||B||^{(k+1)}\right) \cdot \left|\left|\mathbf{e}^{(0)}\right|\right|
\end{aligned}$$

And here is the key. The **sufficient condition for convergence is to choose a matrix B that has the norm less than** 1:

$$||B|| < 1 \Longrightarrow \lim_{k\to\infty} \left|\left|\mathbf{e}^{(k+1)}\right|\right| = 0$$

We recall that the *Euclidean norm* (commonly used) of a matrix is calculated by taking the square root of the sum of the absolute squares of its elements. Let A be a matrix of size $m \times n$, the Euclidean norm:

$$||A||_2 \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$

– **Spectral radius**. The spectral radius of a matrix is the **largest absolute value of its eigenvalues**. We define:

$$\rho(B) = \max_j |\lambda_j(B)|$$

Where $\lambda_j(B)$ are the eigenvalues of B.
Why is the spectral radius useful? Well, if the matrix B is symmetric positive definite (SPD)[4], then the spectral radius is equal to the Euclidean norm of the matrix.

$$B \text{ is SPD } \Rightarrow ||B||_2 = \rho(B) \wedge \rho(B) < 1 \iff \text{method convergences}$$

And this is a very big help to us for many reasons.

* **Balance and Predictability**. When the norm is equal to the spectral, it means that the influence of the matrix is well distributed. In other words, this uniformity can help make our iterative methods more predictable, reducing the possibility of non-convergence.
* **Efficiency**. It avoids scenarios where the matrix might have hidden large entries affecting convergence or stability.

[4] **SPD (Symmetric Positive Definite)** is a matrix:
* Symmetric: $A = A^T$
* Positive Definite: $x^T AX > 0$, $\forall x \in \mathbb{R}^n \setminus \{0\}$

Let $C \in \mathbb{R}^{n\times n}$ then the spectral radius of a matrix is equal to the infimum (lower bound) of its matrix norm:

$$\rho(C) = \inf\{||C|| \ \ \forall \text{ induced matrix norm } ||\cdot||\} \tag{3}$$

It follows from this property that:

$$\rho(B) \leq ||B|| \qquad \forall \text{induced matrix norm } ||\cdot|| \tag{4}$$

Note that thanks to 4 we can observe that if:

$$\exists\, ||\cdot|| \text{ such that } ||B|| < 1 \implies \rho(B) < 1$$

The convergence of the method is guaranteed by the following theorem.

Theorem 1 (**necessary and sufficient condition for convergence**). *A* ***consistent*** *iterative method with iteration matrix B converges if and only if $\rho(B) < 1$.*

2.2.2 Jacobi method

Let the problem of solve $Ax = b$, where A is a square matrix, x is the vector of unknowns, and b is the result vector.

We start from the i-th line of the linear system:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \;\rightarrow\; a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$$

Formally the solution x_i for each i si given by:

$$x_i = \frac{b_i - \sum_{j \neq i} a_{ij}x_j}{a_{ii}} \tag{5}$$

Obviously the previous identity cannot be used in practice because we do not know x_j, for $j \neq i$. And here is the **magic idea** of Jacobi: we could think of introducing an iterative method (Jacobi) that **updates** $x_i^{(k+1)}$ **step** $k+1$ **using the other** $x_j^{(k)}$ **obtained in the previous step** k.

$$x_i = \frac{b_i - \sum_{j \neq i} a_{ij}x_j}{a_{ii}} \xrightarrow{\text{as } x_j \text{ is not well known}} x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}}{a_{ii}} \tag{6}$$

Where $\forall i = 1, \ldots, n$.

Algorithm

1. **Start with an initial guess** $x^{(0)}$, also zero.
2. **Update each component** $x_i^{(k+1)}$ using the equation 6.
3. **Repeat until the changes are less than a specified tolerance** or we haven't found the exact solution (in practice very difficult, almost impossible).

How much does it cost?

It depends on the matrix used:

- **Dense matrix** (bad choice). Each iteration costs $\approx n^2$ operations, so the Jacobi method is competitive if the number of iteration is less than n.
- **Sparse matrix** (good choice). Each iteration costs only $\approx n$ operations.

Can it be parallelized?

The parallelization of the Jacobi method is actually **one of its main advantages** on modern computers. Each update of x_i depends only on the previous values of the other x_j, not on the current iteration values. This independence makes it easy to distribute the work across multiple processors.

2.2.3 Gauss-Seidel method

Given the Jacobi method, the Gauss Seidel method is similar, but with one clever difference: it uses the latest available values during iterations.

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}} \tag{7}$$

At iteration $(k+1)$, let's consider the computation of $x_i^{(k+1)}$. we observer that for $j < i$ (with $i \geq 2$), $x_j^{(k+1)}$ is known (we have already calculated it). We can therefore think of using the quantities at step $(k+1)$ if $j < i$ and, as in the Jacobi method, those at the previous step k if $j > i$.

Algorithm

1. **Start with an initial guess** $x^{(0)}$, also zero.
2. **Iteration**. For each row i from 1 to n calculate the value of the equation 7.
3. **Repeat until the changes are less than a specified tolerance**.

How much does it cost?

The cost is comparable to the Jacobi method explained on page 19.

Can it be parallelized?

Unlike the Jacobi method, the Gauss-Seidel method relies on the most recent updates within the same iteration. This sequential dependency **makes it more difficult to parallelize, as each update depends on the previous ones**.

While it's harder to parallelize due to its inherent sequential nature, we can still achieve some degree of parallelism with clever strategies such as red-black ordering. This makes the Gauss-Seidel method less straightforward to parallelize than Jacobi, but not impossible.

2.2.4 Convergence of Jacobi and Gauss-Seidel methods

Let be a general matrix A, and :

- D the **diagonal part** of A
- $-E$ **lower triangular part** of A
- $-F$ **upper triangular part** of A

$$A = \begin{bmatrix} \ddots & & -F \\ & D & \\ -E & & \ddots \end{bmatrix}$$

The previous Jacobi and Gauss-Seidel methods can be rewritten as:

- Jacobi:
  - Method:
  $$D\mathbf{x}^{(k+1)} = (E+F)\,\mathbf{x}^{(k)} + \mathbf{b}$$
  - Iteration matrix:
  $$B_J = D^{-1}(E+F) = D^{-1}(D-A) = I - D^{-1}A$$
- Gauss-Seidel
  - Method:
  $$(D-E)\,\mathbf{x}^{(k+1)} = F\mathbf{x}^{k} + \mathbf{b}$$
  - Iteration matrix:
  $$B_{GS} = (D-E)^{-1}F$$

We present a theorem which gives us the **sufficient condition for convergence** of the Jacobi and Gauss-Seidel methods.

Theorem 2 (**sufficient condition for convergence of Jacobi and Gauss–Seidel**). *The following conditions are sufficient for convergence:*

- *If a matrix A is **strictly diagonally dominant by rows**:*
  $$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \qquad i = 1, \ldots, n$$
  Then Jacobi and Gauss-Seidel converge.
- *If a matrix A is **strictly diagonally dominant by columns**:*
  $$|a_{ii}| > \sum_{j \neq i} |a_{ji}| \qquad i = 1, \ldots, n$$
  Then Jacobi and Gauss-Seidel converge.
- *If a matrix A is SPD (symmetric positive and definite), then the Gauss-Seidel method is convergent.*

- *If a matrix A is tridiagonal*[5]*, then the square spectral value of the Jacobi iteration matrix is equal to the spectral value of the Gauss-Seidel iteration matrix.*

$$\rho^2 (B_J) = \rho (B_{GS})$$

[5] A matrix is **tridiagonal** when it has non-zero elements only on the main diagonal, the diagonal above the main diagonal, and the diagonal below the main diagonal.

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

2.2.5 The stationary Richardson method

The stationary Richardson method is a way of refining a guess for solving the general problem $Ax = b$. We **start with an initial guess for the solution**, then we **keep adjusting that guess based on how far it is from the actual answer**. The **adjustments depend on a parameter we choose**, which can speed up or slow down how quickly we get to the right answer. We **keep doing this until our guess is close enough to the actual solution**.

Mathematically, given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$, the stationary Richardson method is based on the following recursive update:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha \cdot \underbrace{\left(\mathbf{b} - A\mathbf{x}^{(k)}\right)}_{\text{residual } \mathbf{r}^{(k)}} \tag{8}$$

The idea is to update the numerical solution by adding a quantity proportional to the residual. Indeed, it is expected that if the residual is *large* (*small*), the solution at step k should be corrected *much* (*little*). Where α is a weighted version of the residual.

- Iteration matrix B_α:
$$B_\alpha = I - \alpha A$$
- $\mathbf{f}$:
$$\mathbf{f} = \alpha \mathbf{b}$$

We now ask ourselves **which value of the parameter** α, among those that **guarantee convergence**, **maximizes the speed of convergence**. We introduce the following A-induced norm where A is SPD:

$$||\mathbf{z}||_A = \sqrt{\sum_{i,j=1}^{n} a_{ij} z_i z_j} \iff ||\mathbf{z}||_A = \sqrt{(A\mathbf{z}, \mathbf{z})} = \sqrt{\mathbf{z}^T A \mathbf{z}}$$

We look for $0 < \alpha_{\text{opt}} < \dfrac{2}{\lambda_{\max(A)}}$ such that $\rho(B_\alpha)$ is minimum. That is:

$$\alpha_{\text{opt}} = \underset{0<\alpha<\frac{2}{\lambda_{\max(A)}}}{\operatorname{argmin}} \left\{ \max_i |1 - \alpha \lambda_i(A)| \right\}$$

To understand which α to choose, we plot the problem. On the *x-axis* are the values of α and on the *y-axis* is the spectral radius equal to $|1 - \alpha\lambda_i(A)|$, with $i = 1, \ldots, n$.

In the figure 4 we can see that the upper bound of the spectral radius is equal to 1 (no convergence). Each line represents the possible value of the spectral radius for different values of α. In **green** we see the **spectral radius equal to** $\rho(B_\alpha)$; it is important because its intersection with the upper bound of ρ represents the right bound of the interval where the **values of** α **guarantee convergence**. It can also be seen by the **red arrow**. The **lowest point of the curve is where the spectral radius is minimized, indicating the best** α **for convergence**.

In other words, the optimal value is given by the intersection between the curves:

$$|1 - \alpha\lambda_1(A)| \cap |1 - \alpha\lambda_n(A)|$$

That gives us the perfect formula:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min}(A) + \lambda_{\max}(A)} \tag{9}$$

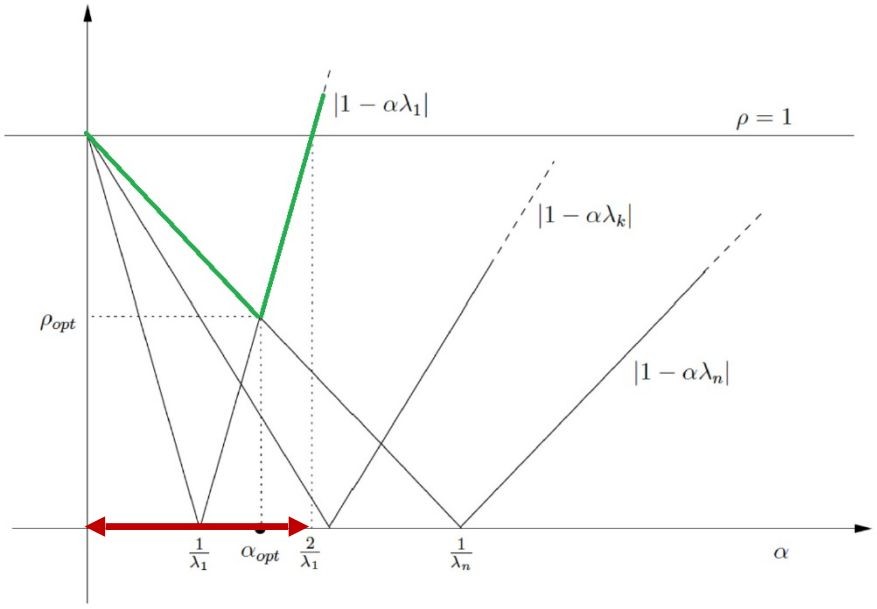

Figure 4: Graphical representation of the optimal alpha to choose in the stationary Richardson method.

If A is SPD, the eigenvalues of A (real and positive) are:

$$\lambda_{\max}(A) = \lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_n(A) = \lambda_{\min}(A) > 0$$

Theorem 3. *Let A be a symmetric and positive definite matrix. The* ***stationary Richardson method is convergent if and only if:***

$$0 < \alpha < \frac{2}{\lambda_{\max}(A)} \tag{10}$$

Since there is a strong correlation between the optimal α and the optimal spectral radius, we can obtain

$$\begin{aligned} \rho_{\text{opt}} &= \rho\left(B_{\alpha_{\text{opt}}}\right) \\ &= -1 + \alpha_{\text{opt}}\lambda_{\max}(A) \\ &= 1 - \alpha_{\text{opt}}\lambda_{\min}(A) \\ &= \frac{\lambda_{\max}(A) - \lambda_{\min}(A)}{\lambda_{\max}(A) + \lambda_{\min}(A)} \end{aligned}$$

Finally, since A is SPD, we have the Euclidean norm equal to the maximum eigenvalue of A: $||A||_2 = \lambda_{\max}(A)$. Moreover, $\lambda_i\left(A^{-1}\right) = \frac{1}{\lambda_i(A)}$, $i = 1, \ldots, n$:

$$\rho_{\text{opt}} = \frac{K(A) - 1}{K(A) + 1} \tag{11}$$

Algorithm

1. **Start with an initial guess** $x^{(0)}$ and **select a parameter** α.
2. **Iteration**. For each k calculate the value of the equation 8.
3. **Repeat until the changes are less than a specified tolerance**.

How much does it cost?

The cost of each iteration depends by type of matrix:

- **Dense matrix**: the cost of each iteration is about n^2 **operations**, where n is the number of unknowns in the linear system.
- **Sparse matrix**: the cost of each iteration is only about n **operations**.

Can it be parallelized?

The stationary Richardson method is not as easily parallelizable as the Jacobi method. Richardson uses the entire solution vector from the previous iteration in each step. This dependency makes it **more difficult to parallelize**.

2.3 Stopping Criteria

A practical test is needed to determine when to stop the iteration. The **main idea** is that we stop iterations when:

$$\frac{\left|\left|\mathbf{x}-\mathbf{x}^{(\mathbf{k})}\right|\right|}{\left|\left|\mathbf{x}^{(k)}\right|\right|} \leq \varepsilon$$

Where ε is a **user defined tolerance**. Meanwhile, the error (left side of the equation) is unknown! There are two criteria we can use to replace it:

- **Residual-based stopping criteria**. It looks at the *residual*, which is the difference between the current solution and the one obtained by reapplying the method's equation:

  $$r^{(k)} = b - Ax^{(k)}$$

  This residual gets smaller as the solution gets closer to the exact answer. When it's small enough, the iteration stops. This approach works because the residual essentially tracks the behaviour of the error. When the residual is small, the error is usually small.

  From a mathematical point of view:

  $$\frac{\left|\left|\mathbf{x}-\mathbf{x}^{(\mathbf{k})}\right|\right|}{\left|\left|\mathbf{x}^{(k)}\right|\right|} \leq K\left(A\right)\frac{\left|\left|\mathbf{r}^{(k)}\right|\right|}{||\mathbf{b}||} \Longrightarrow \frac{\left|\left|\mathbf{r}^{(k)}\right|\right|}{||\mathbf{b}||} \leq \varepsilon$$

  Where $K\left(A\right)$ is the **condition number** of A. It is a measure of **how sensitive the solution of a system of linear equations is to errors in the data or errors in the solution process**.

  - A **low condition number** (close to 1) means that the matrix is well conditioned, and **small errors in the data will cause only small errors in the solution**.
  - A **high condition number** indicates that the matrix is poorly conditioned, and even **small errors in the data can lead to large errors in the solution**.

  To reduce the condition number and the error, we need to use a preconditioner on the main matrix A. So instead of solving the general problem $Ax = b$ directly, we choose a preconditioner P and solve $P^{-1}Ax = P^{-1}b$:

  $$\frac{\left|\left|\mathbf{x}-\mathbf{x}^{(\mathbf{k})}\right|\right|}{\left|\left|\mathbf{x}^{(k)}\right|\right|} \leq K\left(P^{-1}A\right)\frac{\left|\left|\mathbf{z}^{(k)}\right|\right|}{||\mathbf{b}||} \Longrightarrow \frac{\left|\left|\mathbf{z}^{(k)}\right|\right|}{||\mathbf{b}||} \leq \varepsilon \qquad \mathbf{z}^{(k)} = P^{-1}\mathbf{r}^{(\mathbf{k})}$$

- **Distance between consecutive iterates criteria**. It looks at **how much the current iterate (solution) changes compared to the previous one. When this difference becomes small enough, it's a signal that the method is converging and can be stopped**.

  Mathematically, define:

  $$\delta^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \Longrightarrow \left|\left|\delta^{(k)}\right|\right| \leq \varepsilon \Longrightarrow \left|\left|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right|\right|$$

With some manipulation, we can also demonstrate the relation between the true error and $\delta^{(k)}$:

$$\left|\left|\mathbf{e}^{(k)}\right|\right| \leq \frac{1}{1-\rho\left(B\right)} \cdot \left|\left|\delta^{(k)}\right|\right|$$

Indeed:

$$\begin{aligned} \left|\left|\mathbf{e}^{(k)}\right|\right| &= \left|\left|\mathbf{x}-\mathbf{x}^{(k)}\right|\right| \\ &= \left|\left|\mathbf{x}-\mathbf{x}^{(k+1)}+\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right|\right| \\ &= \left|\left|\mathbf{e}^{(k+1)}+\delta^{(k)}\right|\right| \\ &\leq \rho\left(B\right)\cdot\left|\left|\mathbf{e}^{(k)}\right|\right|+\left|\left|\delta^{(k)}\right|\right| \end{aligned}$$

References

[1] Antonietti Paola Francesca. Numerical Linear Algebra. Slides from the HPC-E master's degree course on Politecnico di Milano, 2024.

Index

C

Convergence property 15
Coordinate Compressed Sparse Row format (CSR) 11
Coordinate format (COO) 10

D

Distance between consecutive iterates criteria 26

I

Idempotent Matrices 5
Invertible Matrices 5
Iterative Method 14

L

Lower triangular matrix 6

M

Matrices Multiplication 5
Matrix Associativity Property 5
Matrix Distributive Property 5

N

Nilpotent Matrices 5
Non-singular Matrices 5

O

Orthogonal Matrices 6
Orthogonal Vectors 5

R

Residual-based stopping criteria 26

S

Singular Matrices 5
Sparse Matrix 10, 14
SPD (Symmetric Positive Definite) 17

T

Transpose product between matrices 5
Tridiagonal matrix 22

U

Unitary lower triangular matrix 6
Unitary upper triangular matrix 6
Upper triangular matrix 6