Numerical Linear Algebra - Notes - v0.1.0

260236

September 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

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:

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

  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:

$$A\left(BC\right) = \left(AB\right)C$$

  3. Distributive:

$$A\left(B + D\right) = AB + AD$$

  4. <u>No</u> commutativity:

$$AB \neq BA$$

  5. Transpose of product:

$$\left(AB\right)^T = B^T A^T$$

- **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:

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

  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:

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

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

  1. **Upper triangular matrix**:

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

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

  2. **Lower triangular matrix**:

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

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

- **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**:

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

  2. **Unitary lower triangular matrix**:

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

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 $O\left(n^2\right)$ 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**:

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

$$\begin{aligned} \texttt{AA} &= [12. \quad 9. \quad 7. \quad 5. \quad 1. \quad 2. \quad 11. \quad 3. \quad 6. \quad 4. \quad 8. \quad 10.] \\ \texttt{JR} &= [\,5 \quad 3 \quad 3 \quad 2 \quad 1 \quad 1 \quad 4 \quad 2 \quad 3 \quad 2 \quad 3 \quad 4\,] \\ \texttt{JC} &= [\,5 \quad 5 \quad 3 \quad 4 \quad 1 \quad 4 \quad 4 \quad 1 \quad 1 \quad 2 \quad 4 \quad 3\,] \end{aligned}$$

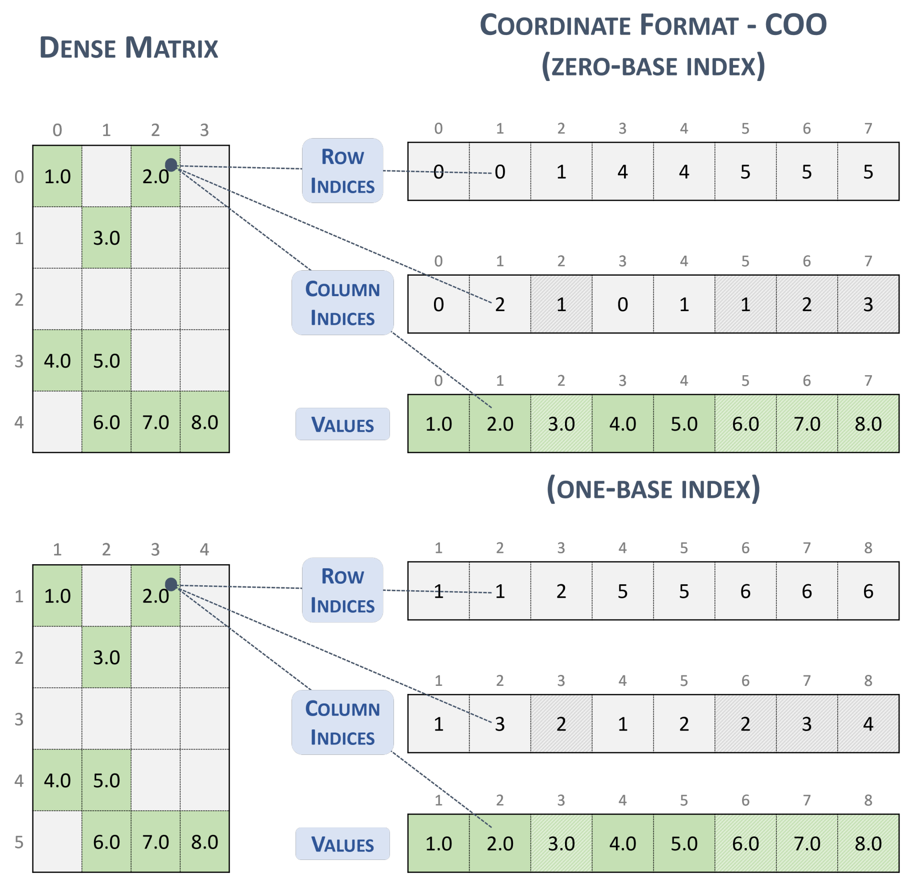

Figure 1: Graphical representation of the coordinate format (COO) technique. From the figure we can see the representation of the `AA` array, called *values*, the `JR`, called *row indices*, and finally the `JC`, called *column indices*. The algorithm is very simple. The figures are taken from the NVIDIA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

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

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

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:

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

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

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

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

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

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.

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

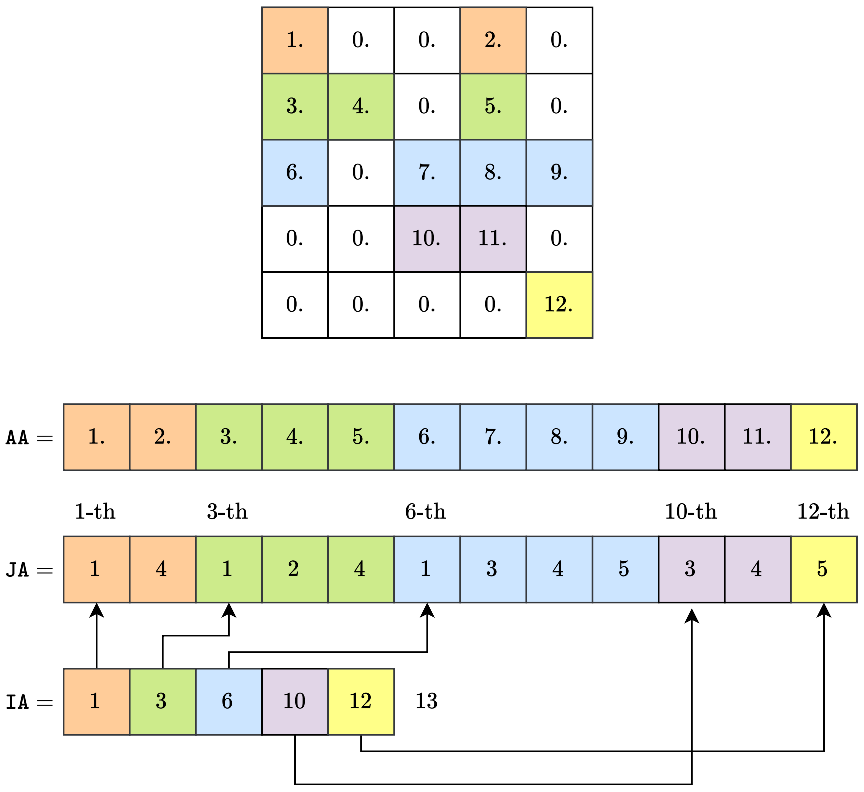

Figure 2: View an illustration of the CRS technique using colors to improve readability.

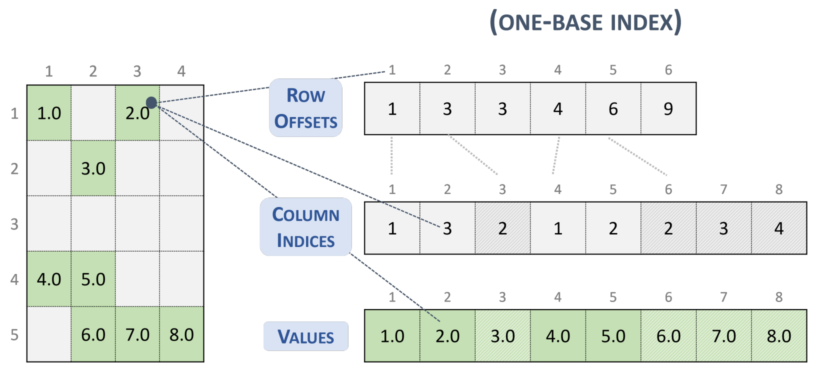

Figure 3: Graphical representation of the coordinate compressed sparse row (CSR) technique. From the figure we can see the representation of the `AA` array, called *values*, the `IA`, called *row offset*, and finally the `JA`, called *column indices*. It's interesting to see how the empty line case is 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.

References

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

Index