MECHANICAL ENGINEERING

Data-Driven Methods for Engineers (MECH0107)

Dr Lama Hamadeh
Office 429 |Roberts Building
Mechanical Engineering Department
(l.hamadeh@ucl.ac.uk)

Dr Llewellyn Morse
Office 503D |Roberts Building
Mechanical Engineering Department
(l.morse@ucl.ac.uk)

2025 - 2026

Contents

1 Introduction

In the past, data on which science and engineering are based was scarce and frequently obtained by experiments proposed to verify a given hypothesis. Each experiment yielded only minimal data. Today, data is abundant and abundantly collected in each single experiment at a minimal cost. Data-driven modeling and scientific discovery is a change of paradigm in how many problems, both in science and engineering, are addressed. These methods have the additional advantage of testing correlations between different variables and observations, learning unforeseen patterns in nature, and allowing us to discover new scientific laws. For instance, some scientific fields have been using artificial intelligence for some time due to the inherent difficulty in obtaining laws and equations to describe some phenomena [1].

One of the aims of this module is to help you grow from a T-shaped professional into a π-shaped professional, as shown in Fig. (1). A π-shaped professional combines deep expertise in two distinct but complementary domains: engineering knowledge and computational/data-driven knowledge. The two "legs" of the π represent this depth, where the engineering expertise covers the understanding of physical systems, design principles, and manufacturing processes, while computational expertise includes programming, algorithms, artificial intelligence, and data analysis. The horizontal bar of the π reflects broad skills such as communication, problem-solving, and multidisciplinary thinking, which enable integration between the two fields. This combination is increasingly vital because modern engineering depends heavily on computational tools like simulations, optimization, and AI-driven design, while computational methods themselves require a grounding in physical principles to ensure realistic and safe outcomes. Innovation often emerges at the intersection of these disciplines, where professionals who "speak both languages" can bridge teams and accelerate progress. Moreover, possessing depth in two domains and breadth across disciplines makes such individuals more adaptable to rapid technological changes, ensuring their skills remain relevant in the evolving landscape of research, industry, and innovation.

In this module, we aim to provide an introductory course on the key data-driven methods used to analyze, examine, and predict patterns of engineering problems. The main learning outcomes of this module are:

1. **Identify** the foundational concepts of artificial intelligence, machine learning, and data-driven modeling and their applications in science and engineering systems.

2. **Develop** and refine various supervised and unsupervised algorithms and appreciate their underlying mathematical backgrounds.

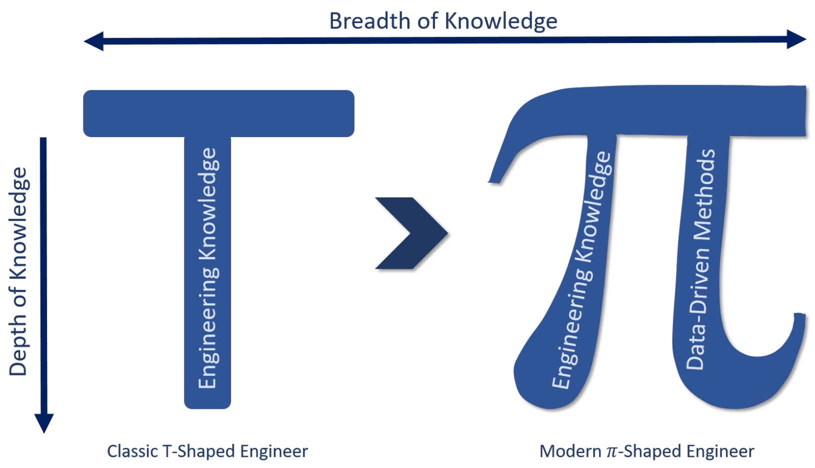


Figure 1: Comparison between t-shaped talent and pi-shaped talent.

3. **Recognise** the value of data, **know** how to ask and answer data-driven questions, and **examine** the reliability and robustness of data-driven models.
4. **Extract** features and patterns from data and discover new knowledge from it.
5. **Identify** the need to use neural networks and deep learning algorithms in some applications, **compare** their efficiency with machine learning algorithms, and **interpret** their predictions.

1.1 Mode of Study

This module will be delivered entirely via in-person teaching. The teaching consists of two types of sessions:

- **Lectures**. There are 9 lectures in total, with one 2-hour lecture per week. Please always check your timetable to confirm the time and the venue of your session. For the lecture to go as smoothly for everyone as possible, please bring your laptop with your preferred software downloaded on it to the lecture so you can code along if it's required. Additionally, it is highly preferable to read the lecture before you come into the session. This ensures that you have a good idea about the lecture topic, and hence it increases your engagement and your overall understanding.
- **Tutorials**. There are 9 tutorials for this module, with one 2-hour tutorial per week. Please always check your timetable to confirm the time and the venue of

your session. During the tutorial session, you will be given a list of questions that relate to the lecture topic for that week. You need to spend the two hours trying to solve them with the help of your peers and the available PGTAs who will be present during the entirety of the session to help and guide you through the thought process of the questions. Please always make sure to ask them if you have any problems/questions.

The assessment of this module has two pieces of coursework: 40% coursework that will be published 4 weeks after the start of this module, and 60% coursework that will be published after the final lecture ends. Please always check the Moodle page for any announcements, instructions, and important deadlines.

1.2 Approaching a Data-Driven Engineering Problem

Taking a data-driven approach refers to the systematic collection, management, analysis, interpretation, and application of data. Any data-driven project mainly comprises three phases: data analysis, data management, and data visualization. When approaching problem-solving in a data-driven project, these steps are to follow:

1. **Define the problem**. Clearly articulate the problem statement, objectives, and desired outcomes.
2. **Break the problem into smaller pieces**. Use a logic tree (also known as a decision tree) to delve into the problem's complexity and uncertainty. Determine what data is necessary to solve each piece and whether that data is readily available.
3. **Data collection** Identify relevant data sources and collect the necessary data to address the problem.
4. **Data pre-processing**. Clean, transform, and prepare the data for analysis. This may involve handling missing values, outliers, and other data quality issues.
5. **Exploratory data analysis (EDA)**. Explore the data to gain insights, and identify patterns, correlations, and relationships between variables.
6. **Model selection**. Choose appropriate data-driven algorithms or statistical models based on the problem and data characteristics.
7. **Implement the chosen model**. More on how to do that will be explained in the following lectures.

8. **Monitoring and maintenance**. Continuously monitor the model's performance in production, retraining or updating it as needed to ensure it remains effective over time.

1.3 Programming Language & Software

The coding material which is associated with the topics of this module, will all be written using the Python programming language. The reason behind choosing Python is that it has libraries for data loading, visualization, statistics, natural language processing, image processing, and more. This vast ecosystem of tools, packages, and libraries addresses a wide-ranging number of programming scenarios and provides scientists and engineers with a large array of general- and special-purpose functionality. In addition, Python has an easy-to-learn structure, runs on multiple operating systems, is license-free, and has excellent online documentation. It has data types that allow you to express complex operations in a very concise manner. This surely makes Python stand out from all the other programming languages. In this module, we will be using Python 3 since all new standard library improvements are only available by default in Python 3. Python 3 is also easier for newcomers to learn, and several aspects of the core language are more consistent than those in Python 2.

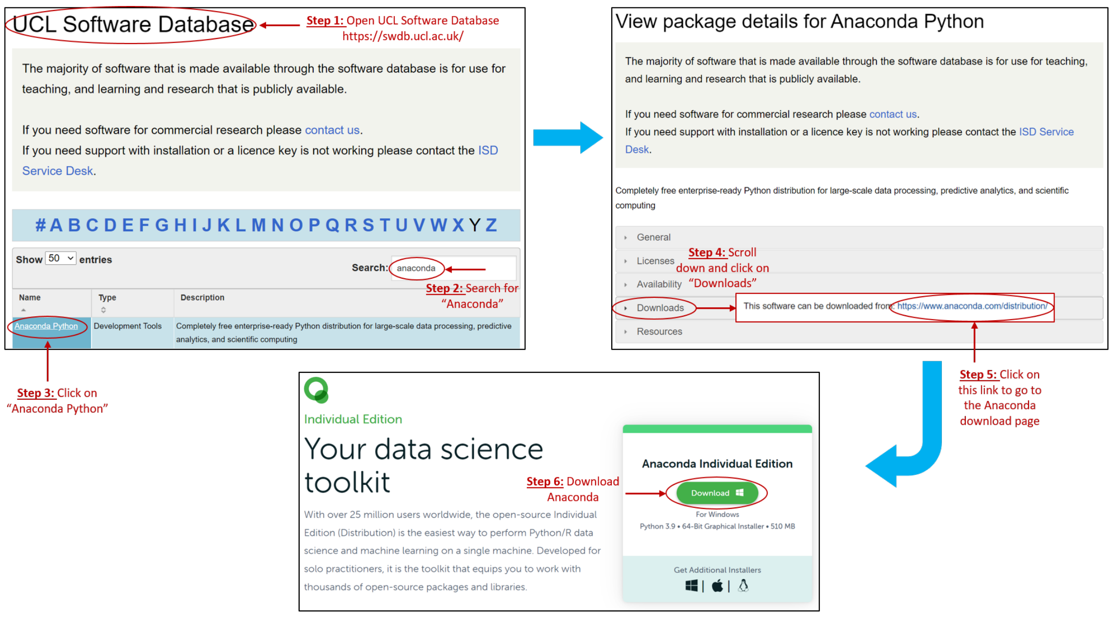
1.3.1 Anaconda Distribution & Jupyter Notebook

As mentioned before, Python has a sea of libraries to perform and tackle several scientific computing scenarios. But instead of installing hundreds of packages manually one at a time, we will be using a Python distribution. A distribution consists of the core Python packages and several hundred modules, all working seamlessly together. All of this is available through a single download. Several Python-integrated development environments (IDEs) allow one to write, test, and debug Python software, e.g., IDLE, Microsoft Visual Studio, PyCharm, PyDev, Thonny, etc. We will use the one in this module called Anaconda distribution. One of Anaconda's very useful development environments is Jupyter Notebook. Jupyter Notebook is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text, and multimedia resources in a single document. Computational notebooks have been around for decades, but Jupyter has exploded in popularity over the past couple of years due to its flexibility and easy-to-use interface. You can use Jupyter Notebooks for nearly all sorts of science and engineering tasks including data cleaning and transformation, numerical simulation, exploratory data analysis, data visualization, statistical modeling, machine learning, deep learning, and much more [2].

1.3.2 Download Anaconda

Anaconda can be easily downloaded onto your computer/laptop either from its official website using this link: `https://www.anaconda.com/download`, or using the UCL software database following the next steps, as shown in Fig(2):

1. Open UCL Software Database using this link: `https://swdb.ucl.ac.uk/`, and then enter your UCL username and password.
2. In the "Search" box, type "Anaconda" and then hit "enter" to look for the package in the dataset.
3. The search should bring back one result: "Anaconda Python". Click on it to view its details.
4. The main page of the package details has several horizontal tabs that provide different tools, e.g., general, licenses, availability, etc. Scroll down and click on "Downloads".
5. The "Downloads" tab has a link that directs the user to the Anaconda website.
6. Locate your download and install Anaconda.


UCL Software Database

The majority of software that is made available through the software database is for use for teaching, and learning and research that is publicly available.

If you need software for commercial research please contact us.
If you need support with installation or a licence key is not working please contact the ISD Service Desk.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Show 50 entries Search: anaconda

Name	Type	Description
Anaconda Python	Development Tools	Completely free enterprise-ready Python distribution for large-scale data processing, predictive analytics, and scientific computing

View package details for Anaconda Python

The majority of software that is made available through the software database is for use for teaching, and learning and research that is publicly available.

If you need software for commercial research please contact us.
If you need support with installation or a licence key is not working please contact the ISD Service Desk.

Completely free enterprise-ready Python distribution for large-scale data processing, predictive analytics, and scientific computing

- General
- Licenses
- Availability
- Downloads: This software can be downloaded from: https://www.anaconda.com/distribution/
- Resources

Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Anaconda Individual Edition

For Windows
Python 3.9 • 64-Bit Graphical Installer • 510 MB

Step 1: Open UCL Software Database https://swdb.ucl.ac.uk/
Step 2: Search for "Anaconda"
Step 3: Click on "Anaconda Python"
Step 4: Scroll down and click on "Downloads"
Step 5: Click on this link to go to the Anaconda download page
Step 6: Download Anaconda
Download
Get Additional Installers

Figure 2: Download Anaconda Distribution.

1.4 Foundational Data Types

When working with data, it's important to recognize that not all values are the same type, and the type determines how we can store, process, and analyze the information. In data analysis and machine learning, variables are often grouped into three main types:

1. **Continuous**. Continuous data has a quantitative nature and represents measurements on a numeric scale and can take on an infinite number of possible values within a range. For example, temperature in a heat exchanger (e.g., 73.4 °C), stress in a steel beam (e.g., 250 MPa), flow rate of water in a pipe (e.g., 3.56 m^3/s), voltage across a circuit component (e.g., 5.12 V), and rotational speed of a turbine (e.g., 3500 rpm). This type of data is often visualized with line graphs, histograms, or scatter plots.

2. **Ordinal**. Ordinal data has a qualitative but ordered nature and describes values that have a meaningful order or ranking, but the differences between levels are not consistent or precisely measurable. For example, material quality grades (Grade A, Grade B, Grade C), surface finish rating in manufacturing (Coarse < Medium < Fine < Superfine), priority level in maintenance scheduling (Urgent < High < Normal < Low), and corrosion severity classification (Mild < Moderate < Severe).

3. **Categorical**. Categorical data has a qualitative but unordered nature and represents information sorted into distinct groups or labels with no inherent order. For example, type of welding process (MIG, TIG, Stick, Flux-Cored), bridge design type (Suspension, Arch, Beam, Truss), material type (Steel, Aluminum, Composite, Polymer), failure mode category (Fatigue, Creep, Fracture, Wear), and control system type (PID, Fuzzy Logic, Adaptive, Model Predictive).

In data analysis, recognizing data types ensures the use of appropriate statistical tests, prevents incorrect assumptions, and guides preprocessing steps such as normalization, discretization, or grouping. In visualization, it influences chart selection—continuous data might be shown with histograms or scatter plots, ordinal data with ordered bar charts, and categorical data with unordered bar or pie charts. In machine learning, data type awareness is essential for feature engineering, choosing suitable algorithms, and optimizing model performance, since some models can handle categorical variables directly while others cannot. Ultimately, a deep understanding of data types allows analysts and engineers to extract meaningful insights, avoid analytical pitfalls, and design models that reflect the true nature of the underlying data.

1.5 Basics of Linear Algebra

Linear algebra is the branch of mathematics that focuses on linear equations. It is often applied to the science and engineering fields, specifically machine learning. Linear algebra is also central to almost all areas of mathematics like geometry and functional analysis. The concepts of linear algebra are crucial for understanding the theory behind machine learning, especially deep learning. They give you better intuition for how algorithms work under the hood, which enables you to make better decisions. In linear algebra, data is represented by linear equations, which are presented in the form of matrices and vectors. Therefore, you are mostly dealing with matrices and vectors rather than with scalars. Fig(3) shows the difference between scalars, vectors, matrices, and tensors. It can be seen that scalars are zero-dimensional, i.e., they appear as one number, as shown in Fig(3, a). Vectors, on the other hand, are one-dimensional, and they can be either column vectors with dimensions $1 \times n$ where n is the number of columns, as shown in Fig(3, b), or row vectors with dimensions $m \times 1$ where m is the number of rows, as shown in Fig(3, c). Matrices expand the idea of vectors and they have a two-dimensional structure with dimensions $m \times n$, as shown in Fig(3, d). Tensors, as shown in Fig(3, e), are often used to represent multi-dimensional data, such as color images, volumetric data, or time series data.

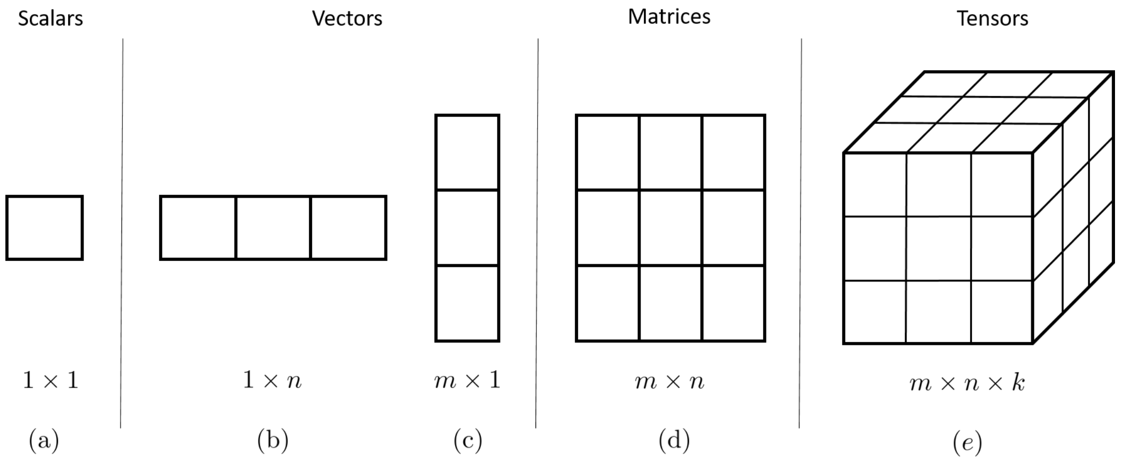


Figure 3: Dimensions of scalars in (a), vectors in (b) and (c), matrices in (d), and tensors in (e).

In the following, we will go through the main basic concepts of linear algebra that will repeatedly appear throughout the topics of this module.

1.5.1 Vectors

A vector is an object that has both a magnitude and a direction. Geometrically, we can picture a vector as a directed line segment, whose length is the magnitude of the vector and with an arrow indicating the direction. The direction of the vector is from its tail to its head, as shown in Fig(4,a). Two vectors are the same if they have the same magnitude and direction. This means that if we take a vector and translate it to a new position (without rotating it), then the vector we obtain at the end of this process is the same vector we had initially. Two examples of vectors are those that represent force and velocity. Both force and velocity are in a particular direction. The magnitude of the vector would indicate the strength of the force or the speed associated with the velocity.

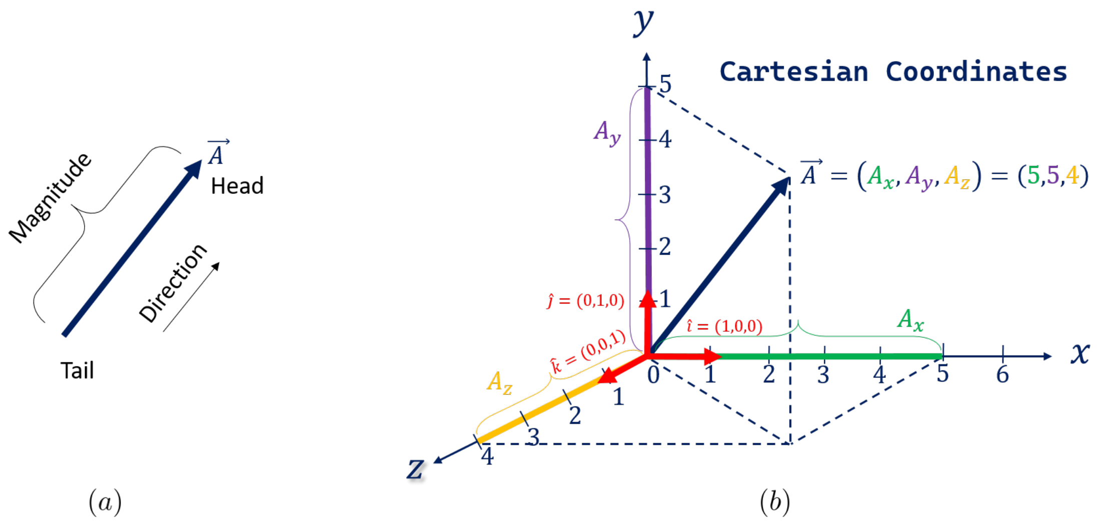


Figure 4: (a) Vectors features. (b) Vectors in Cartesian Coordinates

We denote vectors using boldface as in **A** or **B**. Especially when writing by hand where one cannot easily write in boldface, people sometimes denote vectors using arrows as in $\overrightarrow{A}$ or $\overrightarrow{B}$. In the 3D Cartesian Coordinates, as shown in Fig(4,b), vectors are represented by three components in three dimensions. These components correspond to the points where the projections of this vector hit the three axes. For example, vector **A** is represented in the Cartesian Coordinates in its three components; A_x on axis x, A_y on axis y, and A_z on axis z, and we write: $\mathbf{A} = (A_x, A_y, A_z) = (5, 5, 4)$. Vectors can also be represented by the unit vectors, i.e., $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$. Unit vectors, sometimes referred to as direction vectors are usually determined to form the base of a vector space. They are represented by the symbol $\hat{()}$ which is called a cap or hat. Every vector in the space can be expressed as a linear combination of unit vectors. For example, vector **A** in Fig(4,b) can be written as: $\mathbf{A} = A_x\hat{\imath} + A_y\hat{\jmath} + A_z\hat{\jmath}$. In the following, we will cover the main topics related to vectors.

1.5.1.1 Magnitude of a Vector

The vector magnitude is calculated as follows:

$$||\mathbf{A}|| = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

The magnitude is a scalar quality that corresponds to the length of the vector. As a special case, unit vectors have a magnitude of one.

1.5.1.2 Dot Product

The dot product, also called the scalar product, is a scalar quantity that tells us how much of a vector **A** is applied in the direction of vector **B**. The dot product can also help us measure the angle θ formed by a pair of vectors, as shown in Fig(5). We denote the dot product as $\mathbf{A} \cdot \mathbf{B}$. The dot product is calculated as follows:

$$\mathbf{A} \cdot \mathbf{B} = ||\mathbf{A}|| \cdot ||\mathbf{B}|| \cdot \cos\theta \tag{1}$$

where $||\mathbf{A}||$ and $||\mathbf{B}||$ are the magnitudes of both vectors, and θ is the angle between them. As shown in Fig(5), the sign of the dot product is dictated by the angle between these two vectors. If the angle is acute, then the dot product is positive. If the angle is obtuse, then the dot product is negative. However, if the angle is right, then we have a zero dot product. In this case, both vectors are said to be perpendicular or orthogonal. For example, unit vectors are orthogonal vectors as the angle between each pair of them is a right angle. It is worth noting here that if two vectors are perpendicular (orthogonal) and have magnitudes of one, then they are referred to as orthonormal vectors, e.g., unit vectors. Moreover, if the angle is 0^0 or 180^0 then both vectors are said to be parallel to each other and the corresponding dot product is either in its maximum or minimum value, respectively.
The dot product can also be evaluated using the components of each vector, i.e.,

$$\mathbf{A} \cdot \mathbf{B} = A_x \cdot B_x + A_y \cdot B_y + A_z \cdot B_z \tag{2}$$

Both Methods in Eq.(2) and Eq.(1) are equivalent.

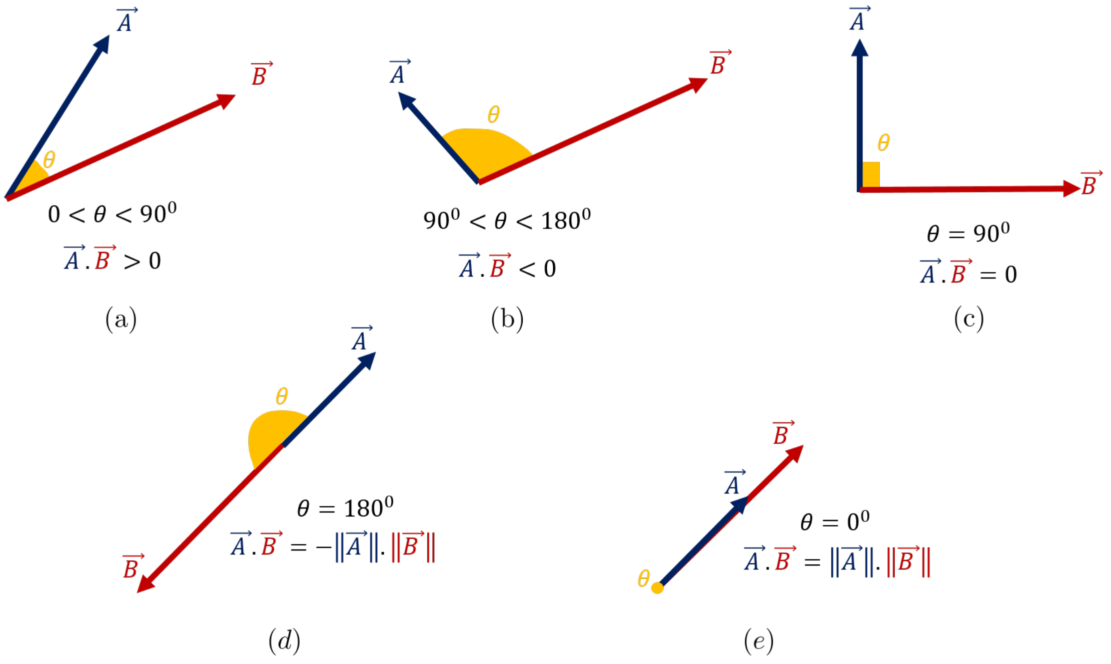


Figure 5: (a) Positive dot product with acute angle between **A** and **B**. (b) negative dot product with an obtuse angle between **A** and **B**. (c) zero dot product with a right angle between **A** and **B**. (d) maximum dot product for $\theta = 0^0$, and (e) minimum dot product for $\theta = 180^0$.

1.5.1.3 Cross Product

The cross product also called the vector product, computes vector **C** which is perpendicular to the plane that consists of both vectors **A** and **B**. The cross product can also help us measure the angle θ formed by a pair of vectors, as shown in Fig(6). We denote the cross product as $\mathbf{A} \times \mathbf{B}$. The cross product is calculated as follows:

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = ||\mathbf{A}|| \cdot ||\mathbf{B}|| \cdot \sin\theta\hat{n} \tag{3}$$

where $||\mathbf{A}||$ and $||\mathbf{B}||$ are the magnitudes of both vectors, θ is the angle between them, and $\hat{n}$ is the unit vector perpendicular to the plane containing **A** and **B**. The angle θ formed between both vectors **A** and **B** dictates the magnitude and the direction of the resulting vector **C**. As shown in Fig(7), when the angle θ between both vectors is either 0^0 or 180^0, the resulting vector **C** has a magnitude of 0. However, if the angle θ is equal to 90^0, then vector **C** has the maximum magnitude with a direction that follows the right-hand rule. Furthermore, if the angle θ is equal to 270^0, then vector **C** has the minimum magnitude with an opposite direction to the previous case.

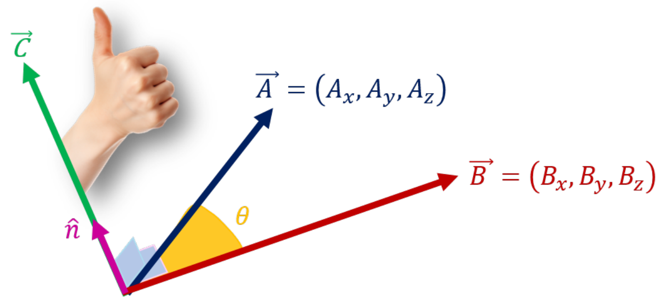


Figure 6: The result of the cross product between vectors **A** and **B** with an angle θ between them is another vector **C** that is perpendicular to the plane consisting **A** and **B**. The direction of vector **C** follows the right-hand rule.

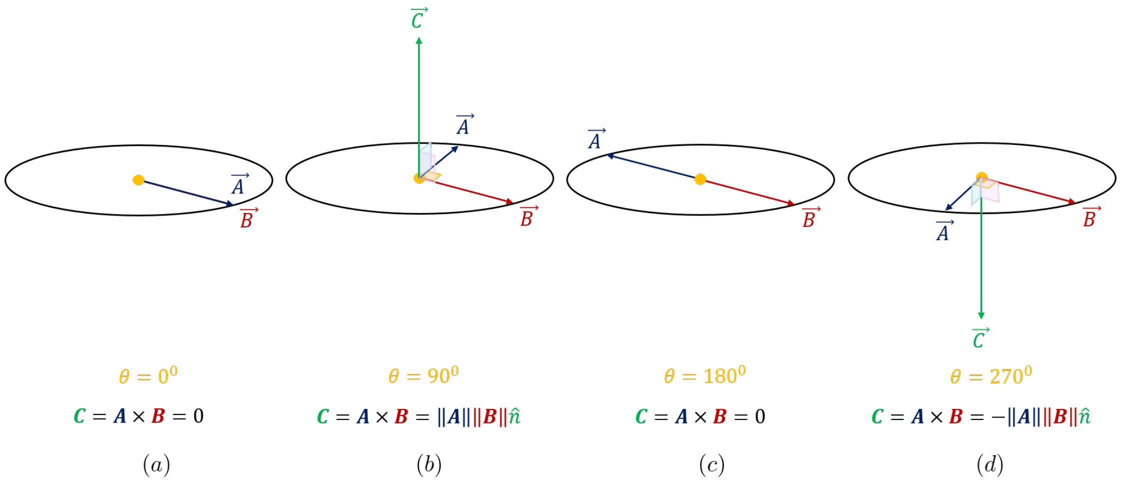


Figure 7: Special cases of the cross product between vectors **A** and **B**. (a) and (c) correspond to the cases where the magnitude of the resulting vector **C** is zero. However, (b) and (d) correspond to the maximum and minimum magnitudes, respectively, of the resulting vector **C**.

The cross product can also be evaluated using the components of each vector, i.e.,

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} = \begin{vmatrix} A_y & A_z \\ B_y & B_z \end{vmatrix} \hat{i} - \begin{vmatrix} A_x & A_z \\ B_x & B_z \end{vmatrix} \hat{j} + \begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix} \hat{k} \tag{4}$$

Both Methods in Eq.(4) and Eq.(3) are equivalent.

1.5.1.4 Vector Norms

Vector Norms are defined as a set of functions that take a vector as an input and output a positive value against it, as shown in Fig(8). We can obtain different lengths for the same vector depending on the type of function we use to calculate the magnitude. All norm functions originate from a standard equation of Norm, known as the p-norm. We obtain a different norm function for different parameter values p (p should be a real number greater than or equal to 1). The generalized equation, however, is shown below:

$$\underbrace{\| \mathbf{x} \|_p}_{p\text{-Norm}} = \left(\sum_i^n |\mathbf{x}_i|^p \right)^{\frac{1}{p}} = (|\mathbf{x}_1|^p + |\mathbf{x}_2|^p + \cdots + |\mathbf{x}_n|^p)^{\frac{1}{p}} \tag{5}$$

This takes an n-dimensional vector $\mathbf{x}$ and raises each element to its p-th power. Then, we sum all the obtained elements and take the p-th root to get the p-norm of the vector, also known as its magnitude. Now, with different parameter values p, we will obtain a different norm function.

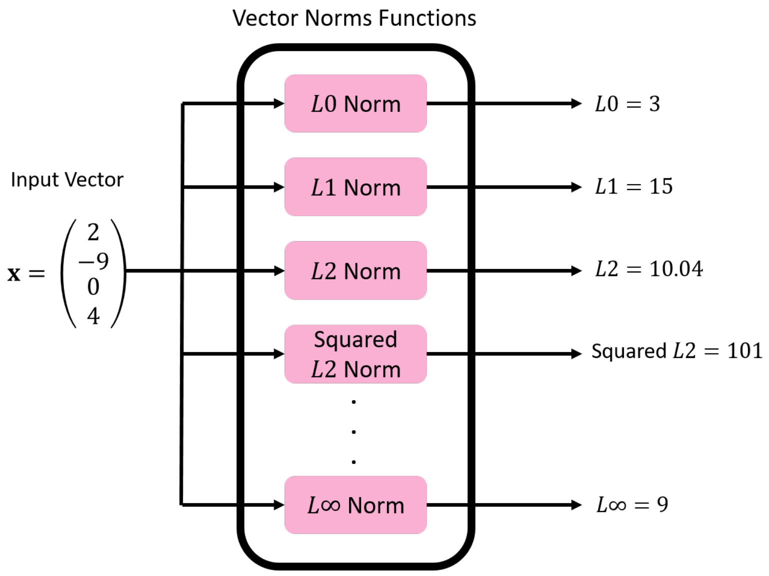


Figure 8: The most common vector norm functions calculated from vector$\mathbf{x}$.

Let's discuss them one by one below.

1. ***L*0 Norm.**
   Although $p = 0$ lies outside the domain of the p-norm function, substituting $p = 0$ in the above equation gives us the individual vector elements raised to the power 0, which is 1 (provided the number is not zero). Furthermore,

we also have a p-th root in the equation, which is not defined for $p = 0$. To handle this, the standard way of defining the $L0$ norm is to count the number of non-zero elements in the given vector. For example, the $L0$ norm for a vector $\mathbf{x} = [2 \quad 3 \quad 4 \quad 0 \quad 1]$, is equal to $L0 = 4$.

2. **$L1$ Norm**
Substituting $p = 1$ in the standard equation of p-norm, Eq.(5), we get the following:

$$\underbrace{\| \mathbf{x} \|_1}_{L1\text{-Norm}} = \left(\sum_i^n |\mathbf{x}_i| \right) = (|\mathbf{x}_1| + |\mathbf{x}_2| + \cdots + |\mathbf{x}_n|) \tag{6}$$

The $L1$ norm is also known as the Manhattan Distance or Taxicab norm. It can be seen from Eq.(6) that the $L1$ norm is the sum of the absolute value of the entries in the vector. The $L1$ norm is also referred to as the *Mean Absolute Error*.

3. **$L2$ Norm**
Of all norm functions, the most common and important is the $L2$ Norm. Substituting $p = 2$ in the standard equation of p-norm, Eq.(5), we get the following equation for the $L2$ Norm:

$$\underbrace{\| \mathbf{x} \|_2}_{L2\text{-Norm}} = \left(\sum_i^n |\mathbf{x}_i|^2 \right)^{\frac{1}{2}} = \left(|\mathbf{x}_1|^2 + |\mathbf{x}_2|^2 + \cdots + |\mathbf{x}_n|^2\right)^{\frac{1}{2}} \tag{7}$$

The above equation is often referred to as the *Root Mean Squared Error* when used to compute the error. L2 norm measures the distance from the origin, also known as Euclidean distance and that's why it is also known as the Euclidean norm.

4. **Squared $L2$ Norm**
As the name indicates, the squared L2 Norm is the same as the L2 Norm but squared.

$$\underbrace{\| \mathbf{x} \|_2^2}_{\text{Squared } L2\text{-Norm}} = \left(\sum_i^n |\mathbf{x}_i|^2 \right) = \left(|\mathbf{x}_1|^2 + |\mathbf{x}_2|^2 + \cdots + |\mathbf{x}_n|^2\right) \tag{8}$$

The above equation is often referred to as the *Mean Squared Error* when used to compute the error in machine learning.

5. **$L\infty$ Norm**
   This norm is also referred to as Max Norm. As infinity is an abstract concept in Mathematics, we can't just substitute $p = \infty$ in the standard p-norm equation. However, we can study the function's behavior as p approaches infinity using limits. As a result, the max norm returns the absolute value of the largest magnitude element. For example, the max norm of a vector $\mathbf{x} = [2 \quad 3 \quad 7 \quad 0 \quad -9]$, is equal to $L\infty = 9$.

1.5.2 Matrices

Matrices, as shown in Fig(3,c), are a two-dimensional set of numbers or symbols distributed in a rectangular shape in vertical and horizontal lines so that their elements are arranged in rows and columns, i.e., $m \times n$ where m is the number of rows and n is the number of columns. Boldface capital letters represent matrices, and lowercase letters with subscripts represent individual numbers in the matrices, i.e.,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \ddots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \tag{9}$$

In machine learning and data analysis, matrices are used to represent datasets, where each row corresponds to an observation or sample, and each column represents a feature or attribute of that sample. This structured representation makes it convenient to apply mathematical operations and transformations to the data. The main topics related to matrices will be briefly covered in the following.

1.5.2.1 Matrices Operations

The matrix operations help us to combine two or more matrices, to form a single matrix. They form the cornerstone of linear algebra, facilitating calculations in engineering, physics, and computer science. These operations, including addition, subtraction, multiplication, and inversion, enable the manipulation of numerical data in a structured format. Understanding their application is essential for solving complex mathematical problems efficiently. Next, the main operations that are commonly applied on matrices are covered.

1. **Addition**
   If **A** and **B** are matrices of the same size, i.e.,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

then they can be added.

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{bmatrix}$$

Matrix addition is commutative and associative.

2. **Subtraction**
   If **A** and **B** are matrices of the same size, i.e.,

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad \mathbf{B}=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

   then one can be subtracted from the other.

$$\mathbf{A}-\mathbf{B}=\begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} \\ a_{21}-b_{21} & a_{22}-b_{22} \end{bmatrix}$$

   Matrix subtraction is neither commutative nor associative.

3. **multiplication**
   If **A** is a matrix of size $m \times n$ and **B** is a matrix of size $n \times p$. For example,

$$\underbrace{\mathbf{A}}_{3\times 2}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \quad \text{and} \quad \underbrace{\mathbf{B}}_{2\times 3}=\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

   then the product **AB** is a matrix of size $m \times p$. Therefore, it is clear that for the multiplication to be doable, the number of columns in the first matrix must be equal to the number of rows in the second matrix. That is, the inner dimensions must be the same.

$$\underbrace{\mathbf{A}\cdot\mathbf{B}}_{3\times 3}=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} & a_{11}b_{13}+a_{12}b_{23} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} & a_{21}b_{13}+a_{22}b_{23} \\ a_{31}b_{11}+a_{32}b_{21} & a_{31}b_{12}+a_{32}b_{22} & a_{31}b_{13}+a_{32}b_{23} \end{bmatrix}$$

   Matrix multiplication is not commutative.

4. **Transpose**
   The matrix resulting from a given matrix after changing or reversing its rows to columns and columns to rows is called the transpose of a matrix. If **A** is a matrix of size $m \times n$, then its transpose $\mathbf{A}^T$ is a matrix of size $n \times m$. For example,

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \Longrightarrow \quad \mathbf{A}^T=\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}$$

5. **Determinant**
The determinant of a square matrix is a scalar attribute that plays a pivotal role in linear algebra. It is a measure that reflects several key properties of a matrix, including whether it is invertible or singular. Represented as $\det(\mathbf{A})$ or $|\mathbf{A}|$, the determinant is crucial in various mathematical operations, such as solving systems of linear equations, analyzing matrix properties, and understanding the impact of linear transformations on geometric figures. It is a tool that helps to determine the behavior of a matrix in the context of vector spaces and transformations. Geometrically we could interpret the determinant of a matrix as the area (for 2×2 matrices) or the volume (for 3×3 and so on...) between all the vectors. For negative determinants, this essentially means that the orientation of the axes relative to one another has changed. Determinants have several properties:

   (a) The determinant is a real number, it is not a matrix.
   (b) The determinant can be a negative number.
   (c) The determinant only exists for square matrices, i.e., $n \times n$.
   (d) The inverse of a matrix will exist only if the determinant is not zero. When a matrix has a zero determinant, we say the matrix is singular.

For a 2×2 matrix, a determinant is calculated as follows:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \Longrightarrow \quad \det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

For 3×3 matrices, calculating determinants requires more arithmetic steps. To explain these steps, let $\mathbf{A}$ be a 3×3 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

To find its determinant, we follow the next steps:

   (a) Step 1: Find the matrix minor. Replace every entry with its minor. Given an entry in a 3×3 matrix, cross out its entire row and column, and take the determinant of the 2×2 matrix that remains (this is called the minor).

For matrix **A**, this gives us

$$\text{Minor}(\mathbf{A}) = \begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ \\ \begin{vmatrix} a_{21} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ \\ \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

(b) Step 2: Find the matrix of cofactors. Change the signs of some of the minors, according to the pattern:

$$\begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix}$$

thus creating what's called the matrix of cofactors.

(c) Step 3: Calculate the determinant. To evaluate the determinant of a 3×3 matrix we choose any row or column of the original matrix **A** - this will contain three elements. We then find three products by multiplying each element in the row or column chosen by its cofactor. Finally, we sum these three products to find the value of the determinant. Note that it does not matter which row or column we choose, we will always get the same value for the determinant.

6. **Inverse**
The inverse of a matrix is another matrix operation, which on multiplication with the given matrix gives the identity matrix. For a matrix **A**, its inverse is $\mathbf{A}^{-1}$, and $\mathbf{A}\mathbf{A}^{-1} = I$, where I is the identity matrix. If **A** is a square matrix, then its inverse $\mathbf{A}^{-1}$ is a matrix of the same size. The general formula for the inverse of a matrix of order 2×2 is equal to the adjoint of a matrix divided by the determinant of a matrix, i.e.,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \Longrightarrow \mathbf{A}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} = \frac{1}{\det(\mathbf{A})}\text{Adj}(\mathbf{A})$$

for matrices of higher-order, e.g., 3×3 matrices, there are three main methods for finding inverses: the method of cofactors, Gauss-Jordan elimination, and the Cayley-Hamilton theorem. Here, we will go step-by-step with the method of cofactors (the other two methods will not be covered here). For a 3×3 matrix $\mathbf{A}$,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The steps of the method of cofactors can be listed as follows:

(a) Step 1: Find the matrix minor. Replace every entry with its minor. Given an entry in a 3×3 matrix, cross out its entire row and column, and take the determinant of the 2×2 matrix that remains (this is called the minor). For matrix $\mathbf{A}$, this gives us

$$\text{Minor}(\mathbf{A}) = \begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

(b) Step 2: Find the matrix of cofactors. Change the signs of some of the minors, according to the pattern:

$$\begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix}$$

thus creating what's called the matrix of cofactors.

(c) Step 3: Find the matrix adjoint. Find the adjoint matrix by taking the transpose of the resulting matrix,i.e., $\text{Adj}(\mathbf{A})$.

(d) Step 4: Calculate the inverse. Finally, divide the adjoint matrix with the determinant of the matrix to compute its inverse, i.e., $\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\text{Adj}(\mathbf{A})$.

The inverse of the matrix exists only if the determinant of the matrix is a non-zero value. The matrix whose determinant is non-zero and for which the inverse matrix can be calculated is called an invertible matrix.

1.5.2.2 Main Types of Matrices

Different types of Matrices and their forms are used for solving numerous problems. In the following, we will go through the ten main types of matrices that are commonly used in engineering data analysis problems.

- **Square Matrix.**
  A square matrix is a matrix that has an equal number of rows and columns, i.e., the matrix's dimension is $m \times m$ where m is the number of columns which is the same as the rows.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 6 \\ 5 & 0 & -1 \\ 2 & 6 & 3 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & 4 \\ 5 & 10 \end{bmatrix} \tag{10}$$

- **Rectangular Matrix.**
  A rectangular matrix is a matrix in which the number of rows is NOT equal to the number of columns, i.e., the matrix's dimension is $m \times n$ where m is the number of rows and n is the number of columns.

$$\mathbf{C} = \begin{bmatrix} 1 & 2 & 6 \\ 5 & 0 & -1 \\ 2 & 6 & 3 \\ 6 & 8 & 0 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 1 & 4 & 5 & 2 & 8 \\ 5 & 10 & 3 & 7 & 9 \\ 3 & 1 & 0 & -1 & 4 \end{bmatrix} \tag{11}$$

- **Zero Matrix.**
  A zero matrix is a matrix that has all its elements equal to zero. Since a zero matrix contains only zeros as its elements, it is also called a null matrix.

$$\mathbf{E} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{F} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

- **Identity Matrix.**
  An identity matrix is a square matrix in which all the elements of principal diagonals are one, and all other elements are zeros.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

- **Upper Triangular Matrix.**
  A square matrix is called upper triangular if all the entries below the main diagonal are zero.

$$\mathbf{J} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix} \tag{14}$$

- **Lower Triangular Matrix.**
  A square matrix is called lower triangular if all the entries above the main diagonal are zero.
  $$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 0 \end{bmatrix} \tag{15}$$

- **Symmetric Matrix.**
  A symmetric matrix is a square matrix that is equal to its transpose.
  $$\mathbf{L} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & 0 \\ -1 & 0 & 5 \end{bmatrix} = \mathbf{L}^T \tag{16}$$

- **Anti-Symmetric Matrix.**
  An anti-symmetric matrix, also called a skew-symmetric matrix, is a square matrix that is equal to the negative of its transpose. Note that this requires the diagonal elements to be equal to zero.
  $$\mathbf{M} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 3 \\ -2 & -3 & 0 \end{bmatrix} = -\mathbf{M}^T \tag{17}$$

- **Diagonal Matrix.**
  A diagonal matrix is a square matrix in which every element except the principal diagonal elements is zero.
  $$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \tag{18}$$

- **Orthogonal/Orthonormal Matrix.**
  An orthogonal matrix is a square matrix where the product of this matrix with its transpose is equal to the identity matrix. On the other hand, an orthonormal matrix is a matrix in which the column vectors form an orthonormal set. An orthonormal set means that each column vector has a length of one and is orthogonal to all of the other column vectors.
  $$\mathbf{O} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \implies \mathbf{O}^T\mathbf{O} = \mathbf{O}\mathbf{O}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

1.5.2.3 EigenDecomposition

Eigendecomposition is an operation that breaks a matrix down into its eigenvalues and eigenvectors to help you better understand its non-obvious and universal properties. This operation can prove useful since it allows certain matrix operations to be easier to perform and it also tells us important facts about the matrix itself [3]. For example, a matrix is only singular if any of its eigenvalues are zero. Before talking about how to calculate the eigenvalues and the eigenvectors, let's try to understand the reasoning behind eigendecomposition, which will give us a more intuitive understanding.

Multiplying a matrix by a vector can also be interpreted as a linear transformation. In most cases, this transformation will change the direction of the vector. Let's assume, we have a matrix $\mathbf{A}$ and a vector $\mathbf{v}$, which we can multiply:

$$\mathbf{Av} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

This linear transformation can be visualized as shown in Fig(9,a). It can be seen that the matrix-vector multiplication changed the vector's direction. But what does this have to do with eigendecomposition? It turns out that when performing eigendecomposition, we are looking for a vector, i.e., an eigenvector, whose direction will not be changed by a matrix-vector multiplication. Only its magnitude will either be scaled up, as in Fig(9,b), or down, as in Fig(9,c). Therefore, the effect of the matrix on the vector is the same as the effect of a scalar on the vector. Therefore, we can say that the vector $\mathbf{v}$ in Fig(9,a) is not an eigenvector to the matrix $\mathbf{A}$ since the linear transformation $\mathbf{Av}$ changes its direction. On the other hand, vector $\mathbf{v}$ in Fig(9,b) and Fig(9,c) are indeed eigenvectors since their resulting vectors $\mathbf{Av}$ maintain their directions. This effect can be described, more formally, by the fundamental eigenvalue equation:

$$\mathbf{Av} = \lambda \mathbf{v} \tag{20}$$

where $\mathbf{A}$ is the square matrix to be eigendecomposed, $\mathbf{v}$ are the eigenvectors, and λ are the eigenvalues. After rearranging and factoring the vector $\mathbf{v}$ out, we get the following equation:

$$(\mathbf{A} - \lambda \mathbf{I})\, \mathbf{v} = 0$$

Now, we have arrived at the core idea of eigendecomposition. To find a non-trivial solution to the equation above, we first need to discover the scalars λ that shift the matrix $\mathbf{A}$ just enough to make sure a matrix-vector multiplication equals zero, sending the vector $\mathbf{v}$ in its null-space. Thinking about it geometrically, we are looking for a matrix that squishes space into a lower dimension with an area or

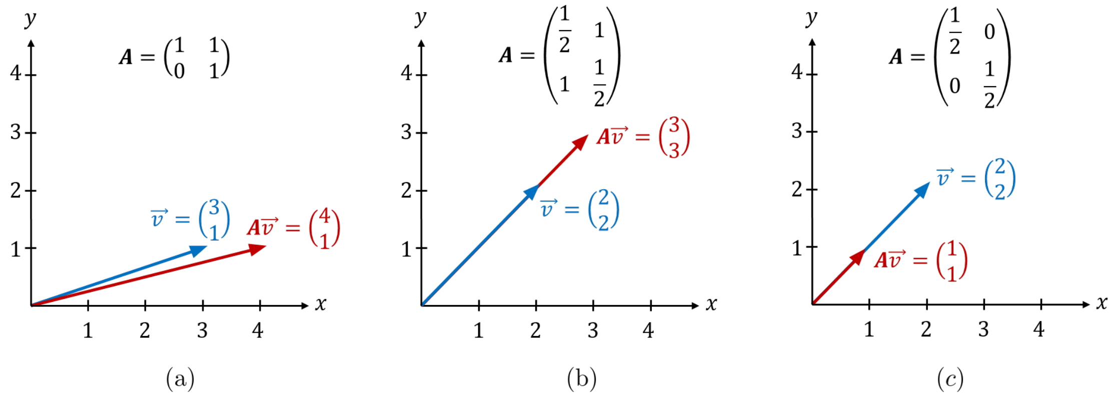


Figure 9: (a) Vector **v** is not an eigenvector to the matrix **A** since the resulting linear transformation **Av** changes its direction. (b) and (c) **v** are eigenvectors to matrices **A** since the resulting vectors **Av** maintains their directions.

volume of zero. We can achieve this squishing effect when the matrix determinant equals zero.

$$\det\left(\mathbf{A} - \lambda \mathbf{I}\right)\mathbf{v} = 0 \tag{21}$$

The scalars λ we have to discover are called the eigenvalues, which unlock the calculation of the eigenvectors. We can envision the eigenvalues as some kind of keys unlocking the matrix to get access to the eigenvectors. Let's explain how to compute both the eigenvalues and their associated eigenvectors by an example. Imagine, we have a 2×2 matrix:

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

and we want to compute the eigenvalues. By using Eq.(21), we can calculate the characteristic polynomial and solve for the eigenvalues as follows:

$$\left| \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} - \lambda \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \right| = 0$$

$$\begin{vmatrix} 3-\lambda & 1 \\ 0 & 2-\lambda \end{vmatrix} = 0$$

Solving this determinant for λ gives:

$$(3-\lambda)(2-\lambda) = 0 \quad \Longrightarrow \quad \lambda_1 = 3 \qquad \text{and} \qquad \lambda_2 = 2$$

The values λ_1 and λ_2 are the eigenvalues. This means that the associated eigenvectors have a magnitude of three and two, respectively. Now, we can unlock the eigenvectors.

Eigenvectors describe the directions of a matrix and are invariant to rotations. Meaning the eigenvectors we are looking for will not change their direction. To compute the eigenvectors, we use the following equation:

$$(\mathbf{A} - \lambda_i \mathbf{I})\,\mathbf{v}_i = 0 \tag{22}$$

Based on the fundamental eigenvalue equation, we simply pluck in the *ith*-eigenvalue and retrieve the *ith*-eigenvector from the null space of the matrix. Let's continue our example from before and use the already discovered eigenvalues $\lambda_1 = 3$ and $\lambda_2 = 2$, starting with $\lambda_1 = 3$:

$$\left(\begin{bmatrix}3 & 1\\0 & 2\end{bmatrix} - \lambda_1 \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)\begin{bmatrix}x_1\\y_1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

$$\begin{bmatrix}3-3 & 1\\0 & 2-3\end{bmatrix}\begin{bmatrix}x_1\\y_1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix}0 & 1\\0 & -1\end{bmatrix}\begin{bmatrix}x_1\\y_1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

It is clear from the multiplication above that one of the solutions both x_1 and y_1 satisfies the two linear systems of equations are $x = 1$ and $y = 0$. Therefore, the eigenvector associated with the first eigenvalue $\lambda_1 = 3$ can be written as:

$$\mathbf{v}_1 = \begin{bmatrix}x_1\\y_1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$$

Taking the same steps for the second eigenvalue, i.e., $\lambda_2 = 2$ gives:

$$\left(\begin{bmatrix}3 & 1\\0 & 2\end{bmatrix} - \lambda_2 \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)\begin{bmatrix}x_2\\y_2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

$$\begin{bmatrix}3-2 & 1\\0 & 2-2\end{bmatrix}\begin{bmatrix}x_2\\y_2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}\begin{bmatrix}x_2\\y_2\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

One of the solutions that satisfies the above linear system of equations is when $x_2 = 1$ and $y_2 = -1$. Therefore, the eigenvector associated with the second eigenvalue $\lambda_2 = 2$ can be written as:

$$\mathbf{v}_2 = \begin{bmatrix}x_2\\y_2\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$$

It is only through a slight abuse of language that we can talk about the eigenvector associated with one given eigenvalue. Strictly speaking, there is an infinity of eigenvectors associated with each eigenvalue of a matrix. Because any scalar multiple of an eigenvector is still an eigenvector, there is, in fact, an (infinite) family of eigenvectors for each eigenvalue, that are proportional to each other.

1.5.3 Tensors

A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array, as shown in Fig(3,e) and Fig(10). In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a tensor. Therefore, we can say that a vector is a one-dimensional or first-order tensor and a matrix is a two-dimensional or second-order tensor.

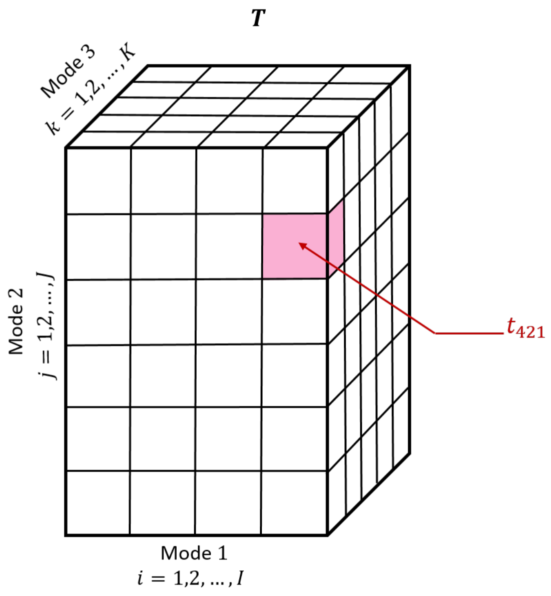
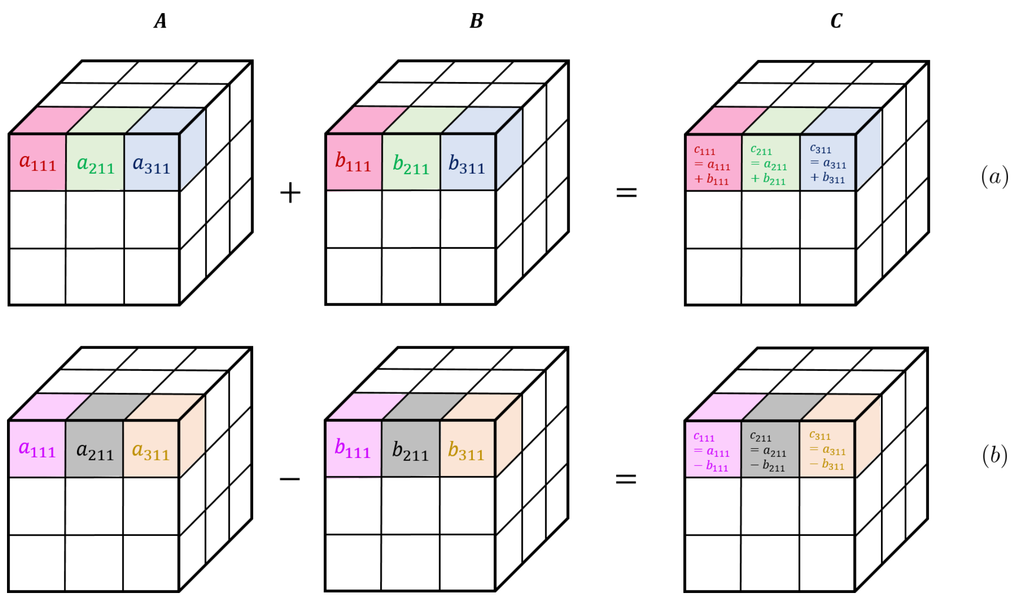


Figure 10: Tensor structure and elements.

Tensors are an important tool in many areas of mathematics and physics, including differential geometry, mechanics, quantum field theory, and machine learning, to name just a few examples. They provide a powerful and flexible way to represent complex relationships and structures in a way that is amenable to mathematical analysis and computation. For instance, different time snapshots (columns) of a matrix may include measurements as diverse as temperature, pressure, concentration of a substance, etc. Vectorising this data generally does not make sense. Ultimately, what is desired is to preserve the various data structures and types in their own, independent directions [4]. In particular, tensors play a critical role in machine learning and deep learning, where they are used to represent and manipulate large datasets, such as images, audio, and text, and to define complex neural network architectures. Tensor notation is much like matrix notation with a capital letter, **T**, representing a tensor, and lowercase letters with subscript integers representing scalar values within the tensor, t_{ijk}. The main arithmetic operations applied on scalars, vectors, and matrices can also be applied to tensors. For example, the element-wise addition of two tensors with the same dimensions results in a new tensor with the same dimensions where each scalar value is the element-wise addition of the scalars in the parent tensors, as shown in Fig(11). The same applied to subtraction.



A
B
C
a_{111} a_{211} a_{311}
+
b_{111} b_{211} b_{311}
=
$c_{111} = a_{111} + b_{111}$
$c_{211} = a_{211} + b_{211}$
$c_{311} = a_{311} + b_{311}$
(a)
a_{111} a_{211} a_{311}
−
b_{111} b_{211} b_{311}
=
$c_{111} = a_{111} - b_{111}$
$c_{211} = a_{211} - b_{211}$
$c_{311} = a_{311} - b_{311}$
(b)

Figure 11: Tensor element-wise addition in (a) and subtraction in (b).

The element-wise multiplication of one tensor from another with the same dimensions results in a new tensor with the same dimensions where each scalar value is the element-wise multiplication of the scalars in the parent tensors. Fig(12) shows the process of multiplying tensor **A** and tensor **B** and the resulting elements of the first index on the third dimension.

1.5.4 System of Linear Equations

A linear equation is an algebraic equation that contains variables (or unknowns) that have an exponent (raised to a power) that is no higher than one. It is also known as a one-degree equation. A linear system of equations is a set of n linear equations in m variables:

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

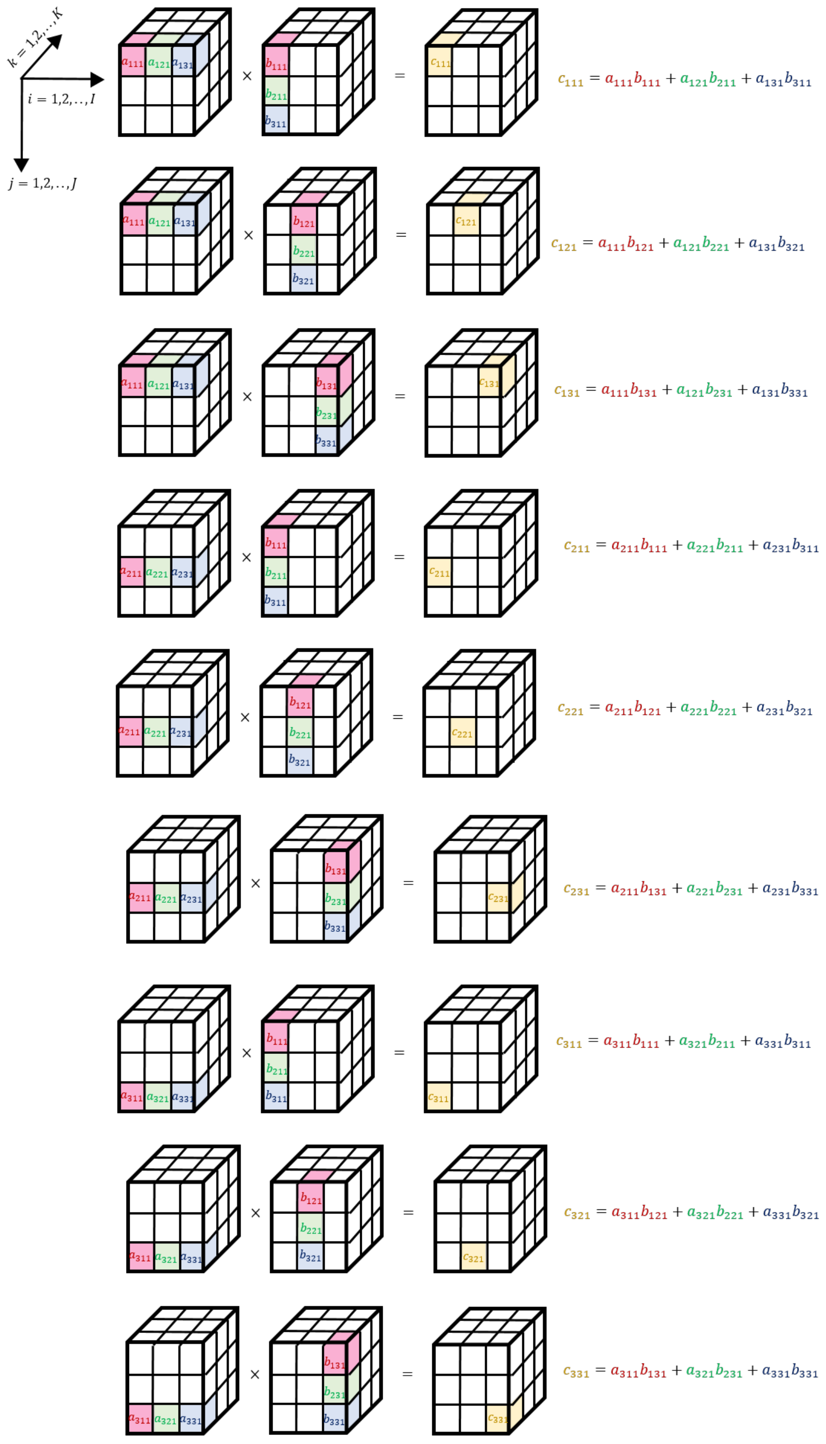

$k = 1,2,..,K$
$i = 1,2,..,I$
$j = 1,2,..,J$
$\times$
$=$
$c_{111} = a_{111}b_{111} + a_{121}b_{211} + a_{131}b_{311}$
$c_{121} = a_{111}b_{121} + a_{121}b_{221} + a_{131}b_{321}$
$c_{131} = a_{111}b_{131} + a_{121}b_{231} + a_{131}b_{331}$
$c_{211} = a_{211}b_{111} + a_{221}b_{211} + a_{231}b_{311}$
$c_{221} = a_{211}b_{121} + a_{221}b_{221} + a_{231}b_{321}$
$c_{231} = a_{211}b_{131} + a_{221}b_{231} + a_{231}b_{331}$
$c_{311} = a_{311}b_{111} + a_{321}b_{211} + a_{331}b_{311}$
$c_{321} = a_{311}b_{121} + a_{321}b_{221} + a_{331}b_{321}$
$c_{331} = a_{311}b_{131} + a_{321}b_{231} + a_{331}b_{331}$

Figure 12: Tensor element-wise multiplication.

This Linear system can be represented in a matrix format as follows:

$$\mathbf{Ax} = \mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{23}$$

where $\mathbf{A}$ is the matrix of coefficients, $\mathbf{x}$ is the column vector of variables (unknowns), and $\mathbf{b}$ is the column vector of constants. The solution of linear systems is one of the most basic aspects of computational science, In many applications, the solution technique often gives rise to a system of linear equations which need to be solved as efficiently as possible [5]. A system of linear equations can have either: one unique solution, infinitely many solutions, or no solutions.

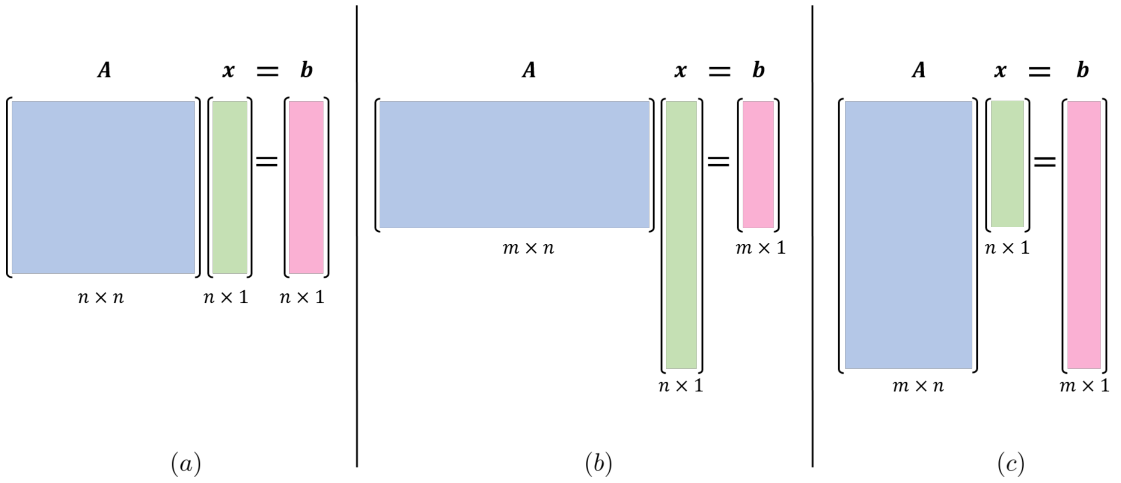


Figure 13: Linear system of equations. In (a), a determined system with only one unique solution. In (b), an underdetermined system, i.e., $m < n$, with an infinite number of solutions. In (c), an overdetermined system, i.e., $m > n$, with no solutions.

- In general, if we have n variables (or unknowns), we need n linearly independent equations (or constraints) to find a unique solution. This system is known as a determined system of linear equations and has only one unique solution, as shown in Fig(13, a).
- If we have more variables (or unknowns) than equations (or constraints), i.e., $m < n$, the system is said to be underdetermined. This system has an infinite number of possible solutions, as shown in Fig(13, b).

- If we have more equations (or constraints) than variables(or unknowns), i.e., $m > n$, then the system will, in general, have no solution. Such a system is said to be overdetermined or inconsistent. Although there may be no actual solution, there are often points in the space of variables that are "almost solutions" in a sense that can be made mathematically rigorous. Least squares analysis is one common approach to finding such approximate solutions, as shown in Fig(13, c).

The main algorithms usually used to solve such systems are covered in the following.

1.5.5 Direct Solution Methods for $\mathbf{Ax} = \mathbf{b}$

A central concern in almost any computational strategy is a fast and efficient computational method for achieving a solution of a large system of equations $\mathbf{Ax} = \mathbf{b}$. In trying to render a computationally tractable, it is crucial to minimize the operations it takes to solve such a system. There are a variety of direct methods for solving $\mathbf{Ax} = \mathbf{b}$: Gaussian elimination, LU decomposition, and inverting the matrix $\mathbf{A}$ [5].

1.5.5.1 Gaussian Elimination

The standard beginning to discussions of solution techniques for $\mathbf{Ax} = \mathbf{b}$ involved Gaussian elimination. We will consider a very simple example of a 3×3 system to understand the operation count and numerical procedure involved in this technique. As a specific example, consider the three equations and three unknowns:

$$x_1 + x_2 + x_3 = 1 \tag{24a}$$
$$x_1 + 2x_2 + 4x_3 = -1 \tag{24b}$$
$$x_1 + 3x_2 + 9x_3 = 1 \tag{24c}$$

In matrix algebra form, we can rewrite this as $\mathbf{Ax} = \mathbf{b}$ with:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

The Gaussian elimination procedure begins with the construction of the augmented matrix:

$$
\begin{aligned}
[\mathbf{A}|\mathbf{b}] &= \left[\begin{array}{ccc|c} \underline{\mathbf{1}} & 1 & 1 & 1 \\ 1 & 2 & 4 & -1 \\ 1 & 3 & 9 & 1 \end{array}\right] \quad \begin{array}{l} R_1 \\ R_2 \\ R_3 \end{array} \\
&= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & -2 \\ 1 & 3 & 9 & 1 \end{array}\right] \quad \begin{array}{l} \\ R_2 - R_1 \\ \\ \end{array} \\
&= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & \underline{\mathbf{1}} & 3 & -2 \\ 0 & 2 & 4 & 0 \end{array}\right] \quad \begin{array}{l} \\ \\ R_3 - R_1 \end{array} \\
&= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & -2 \\ 0 & 1 & 4 & 0 \end{array}\right] \quad \begin{array}{l} \\ \\ R_3 \div 2 \end{array} \\
&= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & -2 \\ 0 & 0 & 1 & 2 \end{array}\right] \quad \begin{array}{l} \\ \\ R_3 - R_2 \end{array}
\end{aligned}
$$

where we have underlined and bolded the pivot of the augmented matrix. The "pivot" or "pivot element" is an element on the left-hand side of a matrix that you want the elements above and below to be zero. It's clear from this example that having lots of zeros is helpful for computing the solutions. Back-substituting then gives the solution:

$$x_3 = 2 \quad \Longrightarrow \quad x_3 = 2 \tag{25a}$$

$$x_2 + 3x_3 = -2 \quad \Longrightarrow \quad x_2 = -8 \tag{25b}$$

$$x_1 + x_2 + x_3 = 1 \quad \Longrightarrow \quad x_1 = 7 \tag{25c}$$

This procedure can be carried out for any matrix **A** which is non-singular, i.e., $\det(\mathbf{A}) \neq 0$. In this algorithm, we simply need to avoid these singular matrices and occasionally shift the rows to avoid a zero pivot. Provided we do this, it will always yield a unique answer.
In scientific computing, the fact that this algorithm works is secondary to the concern of the time required in generating a solution. The operation count for the Gaussian elimination can easily be estimated from the algorithmic procedure for an $n \times n$ matrix:

- Movement down the n pivots.
- For each pivot, perform n additions/subtractions across the columns.

- For each pivot, perform the addition/subtraction down the n rows.

In total, this results in a scheme whose arithmetic operations count is $O(n^3)$.

1.5.5.2 Inverse of a Matrix $\mathbf{A}^{-1}$

If we have one linear equation:

$$ax = b$$

in which the unknown is x and a and b are constants and $a \neq 0$, then $x = \frac{b}{a} = a^{-1}b$. What happens if we have more than one equation and more than one unknown? It turns out that the algebraic solution $x = a^{-1}b$ which is used for a single equation can also be applied to solve a system of linear equations but in this case, instead of taking the inverse of a constant, we take the inverse of a matrix. Therefore, if we have a system of n linear equations, then the vector of the unknowns can be computed as follows:

$$\mathbf{Ax} = \mathbf{b} \qquad \Longrightarrow \qquad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

where $\mathbf{A}^{-1}$ is the inverse of the $n \times n$ matrix.

Let's take a simple example of a 2×2 linear system of equations, as follows:

$$\begin{aligned} 3x_1 + 8x_2 &= 5 \\ 4x_1 + 11x_2 &= 7 \end{aligned}$$

This system can be written in a matrix format as:

$$\mathbf{Ax} = \mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} 3 & 8 \\ 4 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

To solve this system using the inverse of a matrix, we first need to calculate $\mathbf{A}^{-1}$. Calculating the inverse of a 2×2 matrix was covered in section (1.5.2), then we can write:

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{\begin{vmatrix} 3 & 8 \\ 4 & 11 \end{vmatrix}} \begin{bmatrix} 11 & -8 \\ -4 & 3 \end{bmatrix} = \begin{bmatrix} 11 & -8 \\ -4 & 3 \end{bmatrix}$$

Now we are ready to solve. Multiplying the inverse with the right-hand side vector gives:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 11 & -8 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Hence we can say that the solutions of this system are $x_1 = -1$ and $x_2 = 1$. The above steps can be applied to higher dimensions $n \times n$ of square matrices.

1.5.5.3 LU Decomposition

The motivation for an **LU** decomposition is based on the observation that systems of equations involving triangular coefficient matrices are easier to deal with. Indeed, the whole point of Gaussian elimination is to replace the coefficient matrix with one that is triangular. The **LU** decomposition is another approach designed to exploit triangular systems. **LU** decomposition of a matrix is the factorization of a given square matrix **A** into two triangular matrices, one upper triangular matrix **U** and one lower triangular matrix **L**, such that the product of these two matrices gives the original matrix $\mathbf{A} = \mathbf{LU}$. Generally, to solve a given system of linear equations using the **LU** decomposition algorithm, these steps usually are followed:

1. Factor matrix **A** into two triangular matrices; upper **U** and lower **L**, such that:

$$\mathbf{A} = \mathbf{LU}$$

2. The matrix format of the linear system can be written as:

$$\mathbf{Ax} = \mathbf{b} \qquad \mathbf{L}\left(\mathbf{Ux}\right) = \mathbf{b}$$

3. Let a new vector **Y**, such that:

$$\mathbf{Ux} = \mathbf{Y}$$

4. Solve for the vector **Y** in $\mathbf{LY} = \mathbf{b}$.

5. With **Y** now a known vector, solve for the vector **x** in $\mathbf{Ux} = \mathbf{Y}$.

Let's have a simple example of a 3×3 linear system of equations to clarify how this algorithm works. Suppose we have the system:

$$\begin{aligned} x_1 + 2x_2 + 4x_3 &= 1 \\ 3x_1 + 8x_2 + 14x_3 &= 2 \\ 2x_1 + 6x_2 + 13x_3 &= 3 \end{aligned}$$

This system of equations can be written in its matrix format as follows:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 8 & 14 \\ 2 & 6 & 13 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The first step here is to factor matrix **A** into an upper and lower triangular matrices;

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 8 & 14 \\ 2 & 6 & 13 \end{bmatrix} = \mathbf{LU} \quad \text{where} \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

It can be seen that a constraint on the diagonal can be put on either of the triangular matrices to simplify the entire operation. Here, the constraint is imposed on matrix **L**. Multiplying out **LU** and setting the answer equal to **A** gives:

$$\begin{bmatrix} U_{11} & U_{12} & U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + U_{22} & L_{21}U_{13} + U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & L_{31}U_{13} + L_{32}U_{23} + U_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 8 & 14 \\ 2 & 6 & 13 \end{bmatrix}$$

Now we use this to find the entries in **L** and **U**. from the top row, we can see easily that:

$$U_{11} = 1 \qquad U_{12} = 2 \qquad U_{13} = 4$$

Going to the second row, and after inserting the values of U_{11},U_{11}, and U_{11},

$$\begin{aligned} L_{21}U_{11} = 3 \quad &\Longrightarrow \quad L_{21} \times 1 = 3 \quad \Longrightarrow \quad L_{21} = 3 \\ L_{21}U_{12} + U_{22} = 8 \quad &\Longrightarrow \quad 3 \times 2 + U_{22} = 8 \quad \Longrightarrow \quad U_{22} = 2 \\ L_{21}U_{13} + U_{23} = 14 \quad &\Longrightarrow \quad 3 \times 4 + U_{23} = 14 \quad \Longrightarrow \quad U_{23} = 2 \end{aligned}$$

Notice how, at each step, the equation being considered has only one unknown in it, and other quantities that we have already found. This pattern continues on the last row:

$$\begin{aligned} L_{31}U_{11} = 2 \quad &\Longrightarrow \quad L_{31} \times 1 = 2 \quad \Longrightarrow L_{31} = 2 \\ L_{31}U_{12} + L_{32}U_{22} = 6 \quad &\Longrightarrow \quad 2 \times 2 + L_{32} \times 2 = 6 \quad \Longrightarrow \quad L_{32} = 1 \\ L_{31}U_{13} + L_{32}U_{23} + U_{33} = 13 \quad &\Longrightarrow \quad 2 \times 4 + 1 \times 2 + U_{33} = 13 \quad \Longrightarrow \quad U_{33} = 3 \end{aligned}$$

Putting all these results back into the general format of the factorized matrix gives:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 8 & 14 \\ 2 & 6 & 13 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{bmatrix}$$

After factorizing matrix $\mathbf{A}$ into two triangular matrices, the second step in this algorithm is to write the matrix format of the linear system as:

$$\mathbf{Ax}=\mathbf{b} \quad\Longrightarrow\quad \mathbf{L}\left(\mathbf{Ux}\right)=\mathbf{b} \quad\Longrightarrow\quad \begin{bmatrix}1&0&0\\3&1&0\\2&1&1\end{bmatrix}\left(\begin{bmatrix}1&2&4\\0&2&2\\0&0&3\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}\right)=\begin{bmatrix}1\\2\\3\end{bmatrix}$$

The third step here is to propose a new vector $\mathbf{Y}$ so that:

$$\mathbf{Ux}=\mathbf{Y} \quad\Longrightarrow\quad \begin{bmatrix}1&2&4\\0&2&2\\0&0&3\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix}$$

Once vector $\mathbf{Y}$ is known, which is the fourth step, the linear system s can be solved, i.e., the unknowns are computed. To solve for the vector $\mathbf{Y}$, we compute:

$$\mathbf{LY}=\mathbf{b} \quad\Longrightarrow\quad \mathbf{Y}=\mathbf{L}^{-1}\mathbf{b}=\begin{bmatrix}1&0&0\\3&1&0\\2&1&1\end{bmatrix}^{-1}\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}1&0&0\\-3&1&0\\1&-1&1\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}1\\-1\\2\end{bmatrix}$$

The fifth and final step in this algorithm, is to calculate the vector $\mathbf{x}$ since vector $\mathbf{Y}$ is now known. To do that we write:

$$\begin{aligned}\mathbf{Ux}=\mathbf{Y} \quad\Longrightarrow\quad \mathbf{x}=\mathbf{U}^{-1}\mathbf{Y}&=\begin{bmatrix}1&2&4\\0&2&2\\0&0&3\end{bmatrix}^{-1}\begin{bmatrix}1\\-1\\2\end{bmatrix}\\&=\begin{bmatrix}1&-1&-0.6\\0&0.5&-0.3\\0&0&0.3\end{bmatrix}\begin{bmatrix}1\\-1\\2\end{bmatrix}\\&=\begin{bmatrix}0.6\\-1.16\\0.6\end{bmatrix}=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\mathbf{x}\end{aligned}$$

1.5.6 Iterative Solution Methods for $\mathbf{Ax}=\mathbf{b}$

So far, we have discussed some of the main methods for solving systems of linear equations. These methods are direct methods, in the sense that they yield exact solutions (assuming infinite precision!). Another class of methods for solving linear systems consists of approximating solutions using iterative methods. When $\mathbf{A}$ is a large sparse matrix and when methods such as Gaussian elimination require too much time or too much space, you can solve the linear system using iterative methods, which enable you to trade-off between the run time of the calculation and the precision of the solution. Iteration ceases when the error is less than a user-supplied threshold. The final error depends on how many iterations one does as well as on the properties of the method and the linear system. Our overall goal is to develop methods that decrease the error by a large amount at each iteration and do as little work per iteration as possible. In this section, we will discuss two of the most basic iterative methods:

1. Jacobi's method.
2. Gauss-Seidel's method

The advantage of these methods is that they need to store very little, and are often much faster than the direct methods.

1.5.6.1 Jacobi's method

Perhaps the simplest iterative method for solving $\mathbf{Ax} = \mathbf{b}$ is Jacobi's Method. The jacobian method or Jacobi method is one of the iterative methods for approximating the solution of a system of n linear equations in n variables. The Jacobi iterative method is considered an iterative algorithm that is used for determining the solutions for the system of linear equations in numerical linear algebra, which is diagonally dominant [6]. Let's first explain the term "diagonally dominant". Suppose the coefficient matrix $\mathbf{A}$ of a system of linear equations has the form:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

In mathematics, a square matrix is said to be diagonally dominant if, for every row of the matrix, the magnitude/absolute value of the diagonal entry in a row is larger than or equal to the sum of the magnitudes/absolute values of all the other (non-diagonal) entries in that row. This means the following:

$$\mathbf{A} = \begin{bmatrix} \underline{\mathbf{a_{11}}} & a_{12} & a_{13} \\ a_{21} & \underline{\mathbf{a_{22}}} & a_{23} \\ a_{31} & a_{32} & \underline{\mathbf{a_{33}}} \end{bmatrix} \begin{matrix} \longrightarrow & |a_{11}| >= |a_{12}| + |a_{13}| \\ \longrightarrow & |a_{22}| >= |a_{21}| + |a_{23}| \\ \longrightarrow & |a_{33}| >= |a_{31}| + |a_{32}| \end{matrix}$$

Here, the underlined bolded elements are the diagonal elements that dominate the magnitude of their rows. It's worth noting that every row in the matrix $\mathbf{A}$ should satisfy this condition. This is a necessary condition for the Jacobi method to converge. For example:

$$\mathbf{A} = \underbrace{\begin{bmatrix} 4 & -1 & 1 \\ 4 & -8 & 1 \\ -2 & 1 & 5 \end{bmatrix}}_{\text{Diagonally Dominant}} \qquad \mathbf{A} = \underbrace{\begin{bmatrix} -2 & 1 & 5 \\ 4 & -8 & 1 \\ 4 & -1 & 1 \end{bmatrix}}_{\text{Not Diagonally Dominant}}$$

The steps of the Jacobi Methods can be summarized as follows:

1. **Step 1:** Write the system of linear equation in the matrix format of $\mathbf{Ax} = \mathbf{b}$.
2. **Step 2:** Make sure that matrix $\mathbf{A}$ is a diagonally dominant.
3. **Step 3:** Solve for one unknown from each of the equations, e.g., $x_{i+1} = \text{fun}(y_i, z_i)$, where i is the iteration counter.
4. **Step 4:** Initialize each of the unknowns with guess values, i.e., $x_0, y_0, \cdots, z_0$.
5. **Step 5:** Set a maximum number of iterations or a convergence tolerance.
6. **Step 6:** Start iterating!
7. **Step 7:** Stop when the answer "converges" or a maximum number of iterations has been reached.

Let's have a simple example to clarify the algorithm. Suppose we have the following system of linear equations:

$$4x - y + z = 7 \tag{26a}$$
$$4x - 8y + z = -21 \tag{26b}$$
$$-2x + y + 5z = 15 \tag{26c}$$

The matrix format of this system is:

$$\mathbf{Ax} = \mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} 4 & -1 & 1 \\ 4 & -8 & 1 \\ -2 & 1 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ -21 \\ 15 \end{bmatrix}$$

Looking at the matrix $\mathbf{A}$, it can be calculated easily and shown that it is diagonally dominant and hence we can proceed to implement Jacob's method. The next step is to solve for one unknown from each equation, so this gives:

$$x = \frac{7 + y - z}{4} \qquad y = \frac{21 + 4x + z}{8} \qquad z = \frac{15 + 2x - y}{5}$$

If i is the iteration counter, the previous equations become:

$$x_{i+1} = \frac{7 + y_i - z_i}{4} \tag{27}$$
$$y_{i+1} = \frac{21 + 4x_i + z_i}{8} \tag{28}$$
$$z_{i+1} = \frac{15 + 2x_i - y_i}{5} \tag{29}$$

It can be seen here that the values calculated of x, y, and z at each iteration do not change until the next iteration starts. Fig(14,a) shows the solutions of this system of equations after just 5 iterations starting from complete random guesses.

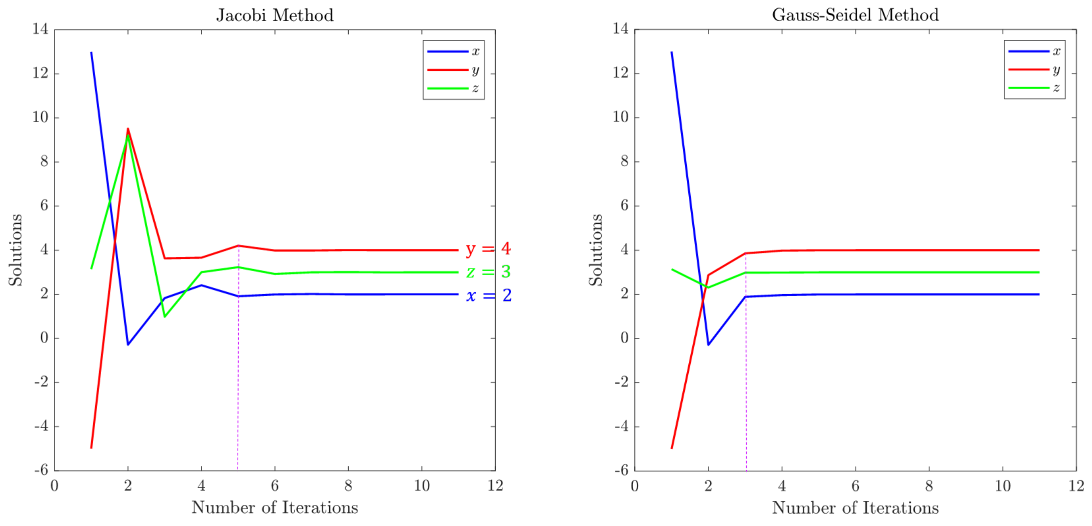
1.5.6.2 Gauss-Seidel's method

The Gauss-Seidel method is an improvement over the Jacobi method. With the Jacobi method, the values obtained in the *ith* iteration remain unchanged until the entire *ith* iteration has been calculated. With the Gauss-Seidel method, we use the new values as soon as they are known. This reduces the number of iterations and hence the Gauss-Seidel method converges much faster to the solution. The steps that this algorithm follows are very similar to the ones listed previously for the Jacobi method. The only difference lies in step 3. In this step, we need to make sure to use the new values of the already calculated unknowns in previous iterations [7]. For example, using Gauss-Seidel method on Eq.(29) gives:

$$x_{i+1} = \frac{7 + y_i - z_i}{4} \tag{30}$$
$$y_{i+1} = \frac{21 + 4x_{i+1} + z_i}{8} \tag{31}$$
$$z_{i+1} = \frac{15 + 2x_{i+1} - y_{i+1}}{5} \tag{32}$$

It can be seen here that to calculate the value of y_{i+1} in the next iteration, the new value of x_{i+1} is used as it has already been evaluated in the first equation. The same applied to calculating z_{i+1}. Solving this system using the Gauss-Seidel method is shown in Fig(14,b) where it shows that the method converges to the solution faster than the Jacobi method as it only requires 3 iterations to converge.
Question to think about!

If you change the order of only two equations in the linear system of Eq.(ref26), what do you think would happen to the solution whether you used the Jacobi method or Gauss-Seidel's method? Think about it!



(a) (b)

Figure 14: Iterative solution for the system of linear equation in Eq.(26) using Jacobi Method in (a) and Gauss-Seidel's method in (b).

1.5.7 Gradient (Steepest) Descent for $\mathbf{Ax} = \mathbf{b}$

The Jacobi and Gauss-Seidel iteration schemes are only two potential iteration schemes that can be developed for solving $\mathbf{Ax} = \mathbf{b}$. However, when it comes to high-performance computing and solving extremely large systems of equations, more efficient algorithms are needed to overcome the computational costs of the methods discussed so far [5]. In this section, the gradient descent, or steepest descent, algorithm is covered [8].

Gradient descent is an iterative optimization algorithm that numerically estimates the local minimum of a function, i.e., finding its lowest values, and ultimately solves the linear system of equations $\mathbf{Ax} = \mathbf{b}$. It serves as a standard method for training machine learning models and neural networks, aiming to reduce prediction errors relative to actual values. To understand the mathematics of this method, we need to go through its algorithmic steps.

1. **Write the system of linear equations in its matrix format $\mathbf{A} = \mathbf{xb}$.** For example, for a simple 2×2 system, this gives:

$$\mathbf{Ax} = \mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

2. **Convert the linear system into its quadratic form**. Generally, the quadratic form of a linear system of equations of n dimensions is given by:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \tag{33}$$

   where $\mathbf{A}$ is a square and diagonally dominant matrix, and $\mathbf{x}$ and $\mathbf{b}$ are vectors. Note that the quadratic form produces a scalar value $f(\mathbf{x})$.

3. **Compute the gradient vector of $f(\mathbf{x})$**. The term "gradient" is typically used in vector calculus for functions of scalar fields with several inputs and a single output. The gradient is a fancy word for derivative, or the rate of change of a function. It's a vector quantity that points in the direction of the greatest increase of the function $f(\mathbf{x})$ and is zero at a local maximum or local minimum. The gradient of the quadratic formula is defined as:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\mathbf{x}) \\ \frac{\partial}{\partial x_2} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\mathbf{x}) \end{bmatrix} \tag{34}$$

   The gradient of Eq.(33) can e easily computed as:

$$\nabla f(\mathbf{x}) = \frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b}$$

   If $\mathbf{A}$ is symmetric, then $\mathbf{A}^T = \mathbf{A}$, this means:

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$$

   The objective here is to construct an iterative process to find the critical point of the gradient so that $\nabla f(\mathbf{x}) = 0$ and hence $f(\mathbf{x})$ is minimized and $\mathbf{A}\mathbf{x} = \mathbf{b}$ is thus be achieved.

4. **Start the iteration process with an arbitrary point $\mathbf{x}^0$.**

5. **Construct the basis of iterations**. Recall from calculus that the gradient, $\nabla f(\mathbf{x})$, is the direction in which the function $f(\mathbf{x})$ is most rapidly increasing, and $-\nabla f(\mathbf{x})$ is the direction of the steepest descent. Thus, if we want to minimize $f(\mathbf{x})$, we might think of guessing at $\mathbf{x}$, evaluating the gradient, and taking a step in the opposite direction until the function stops decreasing. Then

we can repeat the process. At the initial guess point, the gradient $\nabla f(\mathbf{x})$ is computed. This gives the direction of steepest descent towards the minimum point of $f(\mathbf{x})$, i.e., the minimum is located in the direction given by $-\nabla f(\mathbf{x})$. The geometry of the steepest descent suggests the construction of an algorithm whereby the next point in the iteration is picked by following the steepest descent so that [4]:

$$\mathbf{x}_{k+1}(\tau) = \mathbf{x}_k - \tau \nabla f(\mathbf{x}) \tag{35}$$

where the parameter τ dictates how far to move along the gradient descent curve.

6. **Compute the step size τ.** One hard thing about the gradient descent method is adjusting the step size τ, a big τ can cause the loss of convergence while a small one yields a high cost of computational resources. In the gradient descent algorithm, it is crucial to determine when the bottom is reached so that the algorithm is always going downhill in an optimal way. This requires the determination of the correct value of τ in the algorithm. When solving linear equation problems, this task becomes much easier as the difficulty can be overcome by simplistic maths, which gives the exact value of τ. To compute the value of τ, consider the construction of a new function:

$$F(\tau) = f(\mathbf{x}_{k+1}(\tau))$$

which must be minimized now as a function of τ. This is accomplished by computing $\partial F/\partial \tau = 0$ using the chain rule. Thus, one finds:

$$\frac{\partial F}{\partial \tau} = -\nabla f(\mathbf{x}_{k+1}) \nabla f(\mathbf{x}_k) = 0$$

The geometrical interpretation of this result is the following: $\nabla f(\mathbf{x}_k)$ is the gradient direction of the current iteration point and $\nabla f(\mathbf{x}_{k+1})$ is the gradient direction of the future point, thus τ is chosen so that the two gradient directions are orthogonal.

To illustrate these steps in practice, let's consider the simple 2×2 system of linear equations:

$$\begin{aligned} 2x + y &= 0 \\ -x + 6y &= 0 \end{aligned}$$

This system can be written in the matrix format as follows:

$$\mathbf{A} = \mathbf{x}\mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} 2 & 1 \\ -1 & 6 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

To begin with the gradient decent algorithm, this system should be converted into its quadratic form:

$$\begin{aligned}
f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \\
&= \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 2 & 1 \\ -1 & 6 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 2x + y \\ -x + 6y \end{bmatrix} - 0 \\
&= x^2 + 3y^2
\end{aligned}$$

Let's now compute the gradient of this scalar function:

$$\begin{aligned}
\nabla f(x,y) &= \frac{\partial f}{\partial x}\,\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\,\hat{\mathbf{y}} \\
&= 2x\,\hat{\mathbf{x}} + 6y\,\hat{\mathbf{y}} \\
&= \begin{bmatrix} 2x \\ 6y \end{bmatrix}
\end{aligned}$$

The next step now is to build the iteration process. This starts with computing Eq.(35) as follows:

$$\begin{aligned}
\mathbf{x}_{k+1}(\tau) &= \mathbf{x}_k - \tau\nabla f(\mathbf{x}) \\
&= \begin{bmatrix} x \\ y \end{bmatrix} - \tau\begin{bmatrix} 2x \\ 6y \end{bmatrix} \\
&= (1-2\tau)x\,\hat{\mathbf{x}} + (1-6\tau)y\,\hat{\mathbf{y}}
\end{aligned}$$

This expression is used to compute:

$$\begin{aligned}
F(\tau) &= f(\mathbf{x}_{k+1}(\tau)) \\
&= [(1-2\tau)x]^2 + 3\,[(1-6\tau)y]^2 \\
&= (1-2\tau)^2x^2 + 3(1-6\tau)^2y^2
\end{aligned}$$

Setting $F'(\tau) = 0$ then gives:

$$\begin{aligned}
\frac{\partial F}{\partial \tau} &= \frac{\partial}{\partial \tau}\left[(1-2\tau)^2x^2 + 3(1-6\tau)^2y^2\right] \\
&= 8x^2\tau - 4x^2 + 216y^2\tau - 36y^2 \\
&= 0
\end{aligned}$$

Solving this equation for τ gives:

$$\tau = \frac{x^2 + 9y^2}{2x^2 + 54y^2}$$

This is the optimal descent step length. Note that the length of τ is updated as the algorithm progresses. This gives us all the information necessary to perform the steepest descent search for the minimum of the given function. Fig(15) shows the function $f(x, y)$ and its contoured lines. The red points refer to the system's solution as the algorithm progresses. The algorithm starts with a random guess of $x = 2.5$ and $y = 2$ and it keeps iterating taking certain step sizes until it reaches the surface's minimum, after ten iterations, which solves our system of linear equations which is $x \approx 0$ and $y \approx 0$.

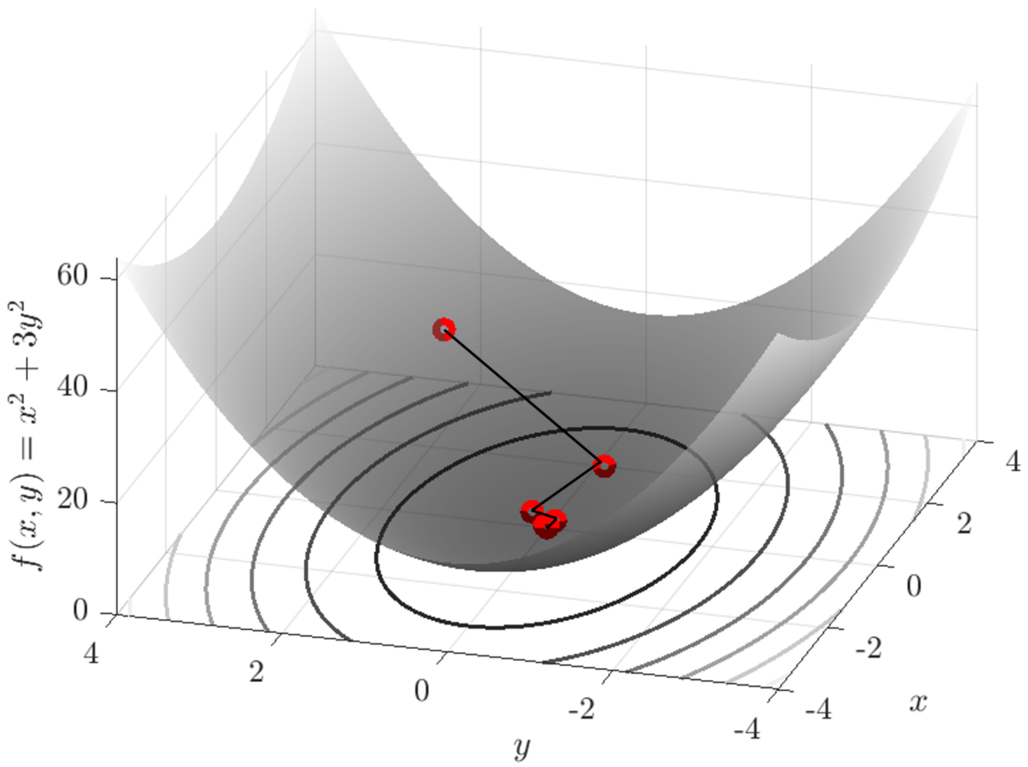


Figure 15: Gradient descent algorithm. The quadratic form surface of two variables $f(x, y) = x^2 + 3y^2$ is plotted along with its contour lines. The red points show the algorithm's iterations starting from a random guess and it keeps iterating until it finds the surface minimum, i.e., the solution of the linear system of equations.

1.6 Numerical Basic Statistics

Since the theme of this module is to mainly work with data and extract useful information from it, it would be unrealistic not to mention the basic statistical tools that are usually used for such tasks. In statistics, a data set can be well described by the following four fundamental quantities: **mean**, **median**, **mode**, and **standard deviation**. These four statistical quantities give us enough information to characterize the distribution of our data set.

1.6.1 Mean

The mean of a data set is the sum of all numbers in the data set divided by the number of points in the set. Although data points fall above, below, or on the mean, it can be considered a good estimate for predicting subsequent data points. If the collected dataset has N points, then its mean is defined/calculated in the following manner:

$$\overline{x} = \sum_{i=1}^{N} \frac{x_i}{N} \tag{36}$$

1.6.2 Median

The median of a data set is the middle value in a set of numbers listed in increasing order. Generally, the median is the middle value of a set of data containing an odd number of values, or the average of the two middle values of a set of data with an even number of values.

1.6.3 Mode

The mode of a set of data is the value that occurs most frequently.

1.6.4 Standard Deviation

The standard deviation describes how the numbers in the data set are distributed around the mean. Data sets with a small standard deviation have tightly grouped, precise data. Data sets with large standard deviations have data spread out over a wide range of values. The standard deviation is defined as follows:

$$\sigma = \sqrt{\sum_{i=1}^{N} \frac{(x_i - \overline{x})^2}{N}} \tag{37}$$

For example, suppose we have two datasets, as shown in Fig(16). Notice how both Data Sets have the same mean, median, and mode, which tells us the data points in each set are centered around the mean value of $\overline{x} = 9.8$. However, the standard deviations are quite different. The large standard deviation in Data Set 2 tells us there must be outliers in the data set which increases the distribution, whereas, the relatively small standard deviation in the first data set tells us the numbers in Set 1 are clustered closely together.

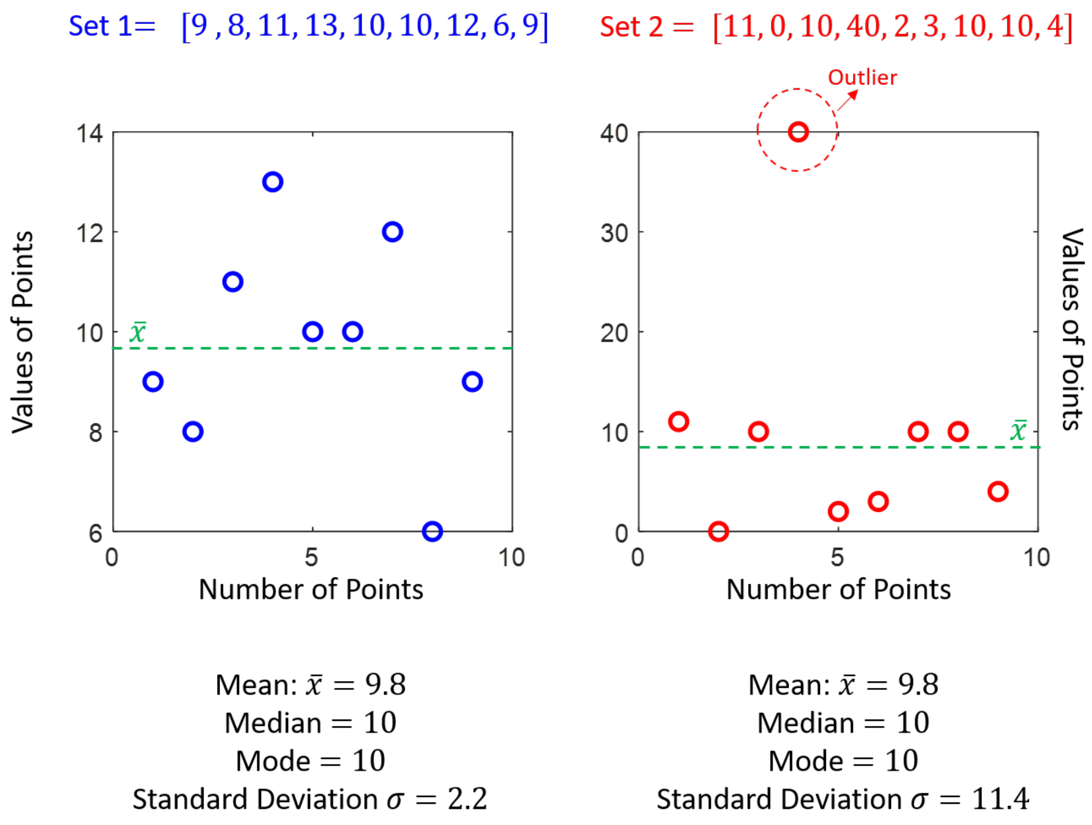


Figure 16: Basic statistics.