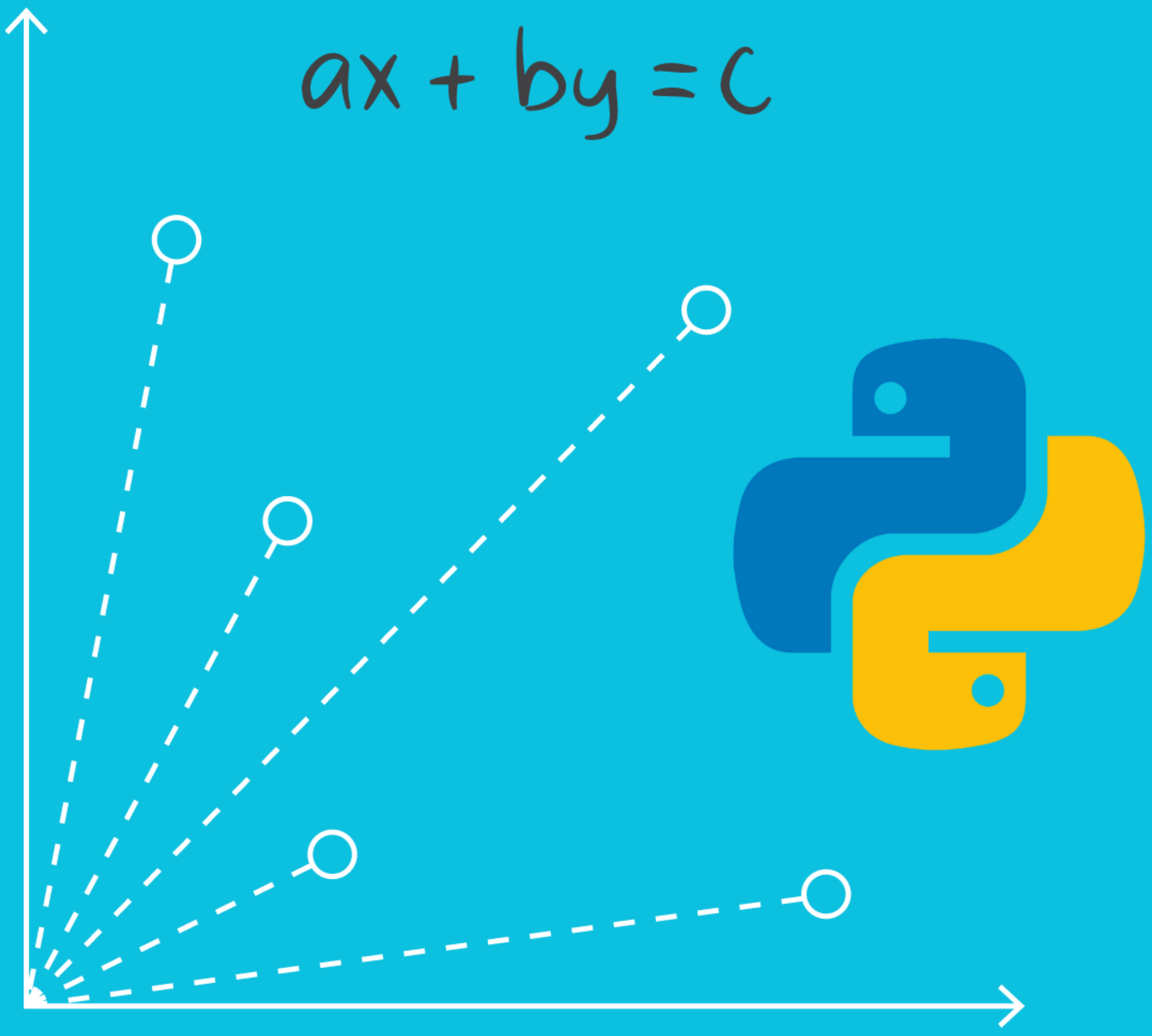
Eco Industry
Oriented University

LINEAR ALGEBRA

with Python



Writer:

Bakti Siregar, M.Sc., CDS.

First Edition

Linear Algebra

Bakti Siregar, M.Sc., CDS

Table of contents

Linear Algebra is a branch of mathematics that plays a fundamental role in various fields, ranging from physics and engineering to economics and computer science. In recent decades, the advancements in technology and data science have further emphasized the importance of Linear Algebra, particularly in finance, business, and machine learning. This book is designed to bridge the understanding of the basic theories of Linear Algebra with its applications in modern contexts, where data analysis and decision optimization are increasingly essential for strategic decision-making.

In the world of finance and business, Linear Algebra plays a key role in portfolio analysis, risk management, and in modeling and predicting market trends. Understanding vectors, matrices, and linear transformations is crucial for solving various problems, such as investment optimization, stock price forecasting, and regression analysis.

On the other hand, machine learning heavily relies on Linear Algebra to build efficient learning algorithms, especially in processing large-scale data. Concepts such as matrix decomposition, eigenvalues, and quadratic optimization methods enable the development of robust machine learning models that can be applied in various fields, from pattern recognition to big data management.

Preface

Linear Algebra serves as a foundational tool for advanced analytical methods in finance, business, and machine learning. This book aims to connect theoretical concepts with practical applications, helping readers harness the power of Linear Algebra in real-world scenarios.

Each chapter introduces key topics such as vectors, matrices, systems of linear equations, linear transformations, eigenvalues, and Singular Value Decomposition (SVD). Real-world examples and case studies demonstrate how these concepts can solve complex problems and drive strategic decision-making. Practical exercises at the end of each chapter encourage hands-on learning, and guidance on using R and Python will enable readers to implement these techniques effectively.

I extend my gratitude to everyone who supported this project, especially my family, friends, colleagues, and students. I hope this book serves as a valuable resource for students and professionals alike, inspiring you to explore the impact of Linear Algebra in today's data-driven world.

Advantages of This Book

This book offers a **practical and applied** approach, where the discussion not only focuses on the theory of Linear Algebra but also on its application in the real world, especially in the fields of **finance, business,** and **machine learning**. Each concept is accompanied by relevant case studies, such as portfolio optimization, risk management, and data analysis. The material is organized **systematically**, from basic to advanced topics, allowing readers to understand each section thoroughly before moving on to more complex concepts. This is particularly beneficial for those new to Linear Algebra or those wishing to deepen their understanding.

Additionally, this book emphasizes the **applications of Linear Algebra in the digital age**, especially in **data science** and **machine learning**, which have become essential in various industries. Readers will also be introduced to supporting software such as **R** and **Python**, which are used for simulations, calculations in Linear Algebra, and data analysis, providing practical skills that are highly valuable in the workforce.

Another advantage of this book is the inclusion of **case studies** and **real examples** in each chapter, which help readers directly see how Linear Algebra concepts can be applied to solve real problems. Examples include regression analysis in finance and image and text processing in machine learning. This book is also **suitable for a wide**

range of readers, from students to professionals, with explanations that are easy to understand, clear illustrations, and examples that effectively clarify abstract concepts.

This book strikes a balance between **theory and practice**, allowing readers to grasp Linear Algebra concepts while also learning how to apply them in real-world scenarios. Furthermore, it employs an **interdisciplinary approach**, linking Linear Algebra with fields such as economics, business, and technology, thus providing broader insights into the contributions of Linear Algebra across various industrial sectors.

About the Author

Bakti Siregar, M.Sc., CDS

Bakti serves as a lecturer in the Data Science Program at ITSB. He earned his Master's degree from the Department of Applied Mathematics at National Sun Yat Sen University, Taiwan. In addition to teaching, Bakti also works as a freelance Data Scientist for leading companies such as JNE, Samora Group, Pertamina, and PT. Green City Traffic. He has a special enthusiasm for working on projects (teaching) in the fields of Big Data Analytics, Machine Learning, Optimization, and Time Series Analysis in finance and investment. His main expertise lies in statistical programming languages such as R Studio and Python. He is also experienced in implementing database systems such as MySQL/NoSQL for data management and proficient in using Big Data tools like Spark and Hadoop. Some of his projects can be found at the following links: Rpubs, GitHub, Website, and Kaggle.

Acknowledgments

With heartfelt gratitude, I would like to thank everyone who contributed to the preparation of this book. I extend special thanks to my beloved family for their support, patience, and understanding throughout the writing process. I also thank my colleagues and academic peers for their valuable feedback and constructive criticism, which have helped refine this book. I greatly appreciate the contributions of the students who provided inspiration and motivation, as well as the publishing team for their professionalism in assisting with the publication process. I hope this book can benefit readers and contribute to the advancement of knowledge in finance, business, and machine learning.

Feedback & Suggestions

Constructive suggestions regarding aspects that need to be added or clarified are greatly appreciated, as they can help improve the quality of this book. Additionally, if there are specific topics that you find relevant and would like to explore further, please feel free to share those ideas. Feedback from readers will not only enrich the learning experience but also assist the author in crafting better works in the future.

Readers/users who wish to provide feedback and suggestions are invited to do so through the contact information below:

- dsciencelabs@outlook.com
- siregarbakti@gmail.com
- siregarbakti@itsb.ac.id

Chapter 1

Introduction

"Linear algebra is the key to understanding higher mathematics, as it provides a unified way of handling systems of equations, transformations, and more."
– Lay (2012)

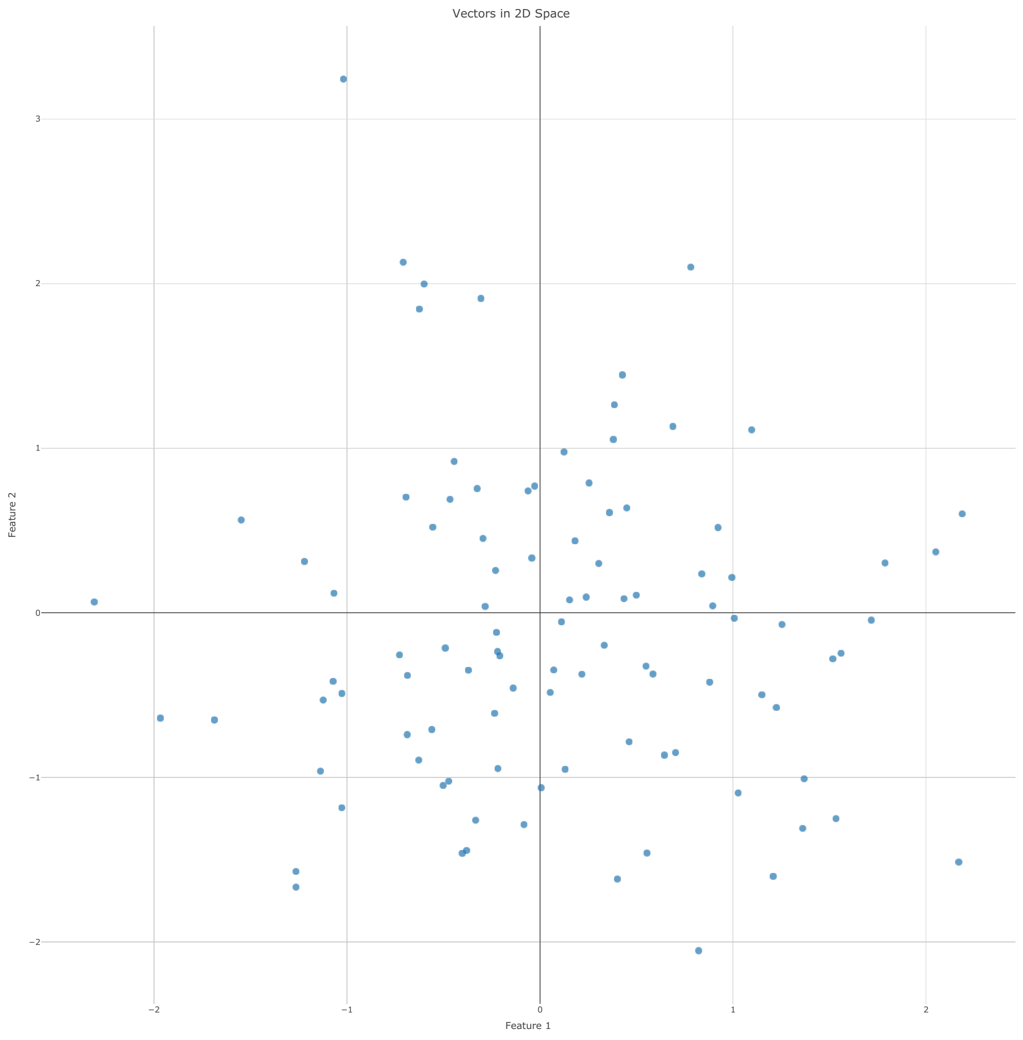
Linear algebra is a core mathematical discipline that serves as a critical foundation in data science. Numerous techniques in data analysis and machine learning rely on linear algebra concepts such as matrices, vectors, and linear transformations. In data science, linear algebra enables us to efficiently handle large datasets and apply various algorithms used for modeling and analysis.

1.1 Key Concepts in Linear Algebra:

1.1.1 Vectors and Vector Spaces

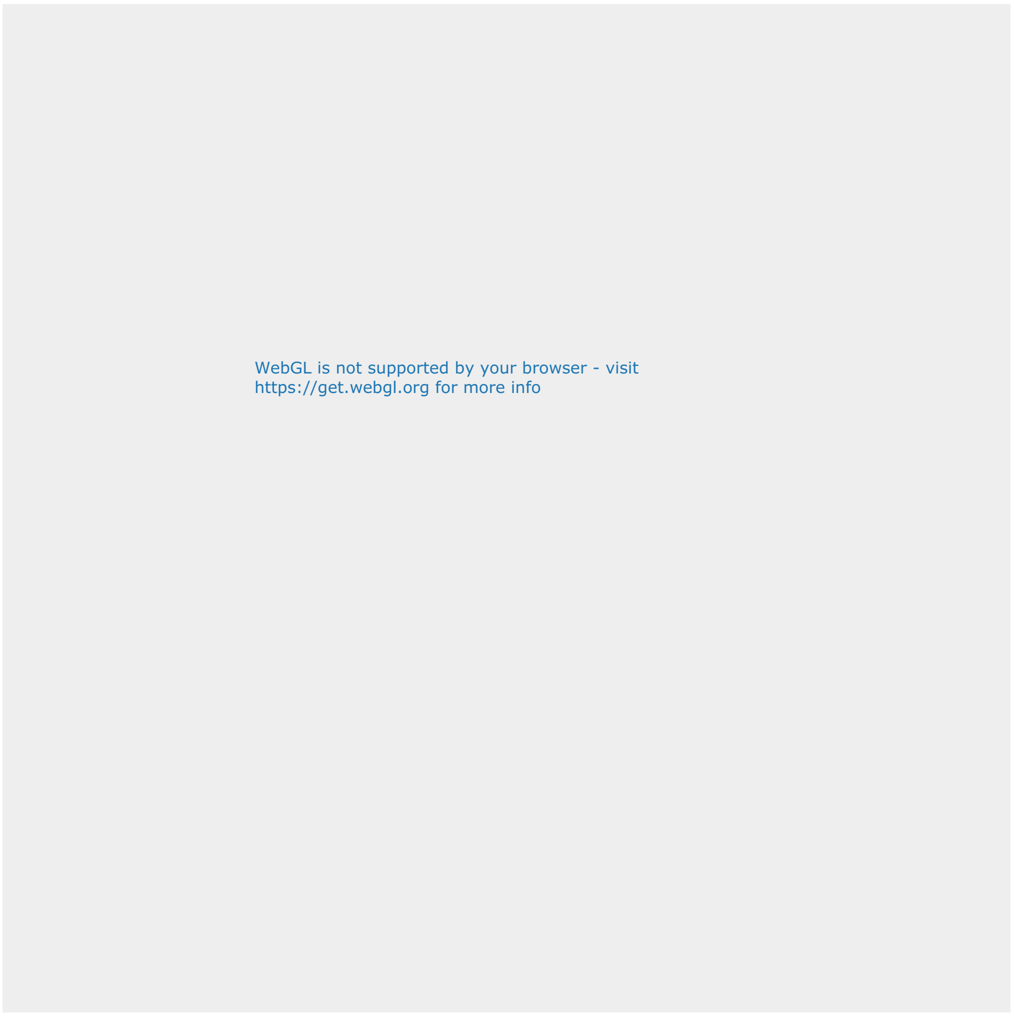
- Vectors are frequently used to represent features or attributes in datasets. For instance, each data point in a dataset can be viewed as a vector in an n-dimensional space, where each dimension corresponds to a specific feature.
- Vector spaces allow us to work with sets of vectors in a more abstract manner, facilitating methods like Principal Component Analysis (PCA) and dimensionality reduction.

In this section, we illustrate vectors and vector spaces by plotting a scatter plot of points in a 2D space.



1.1.2 Matrices and Matrix Operations

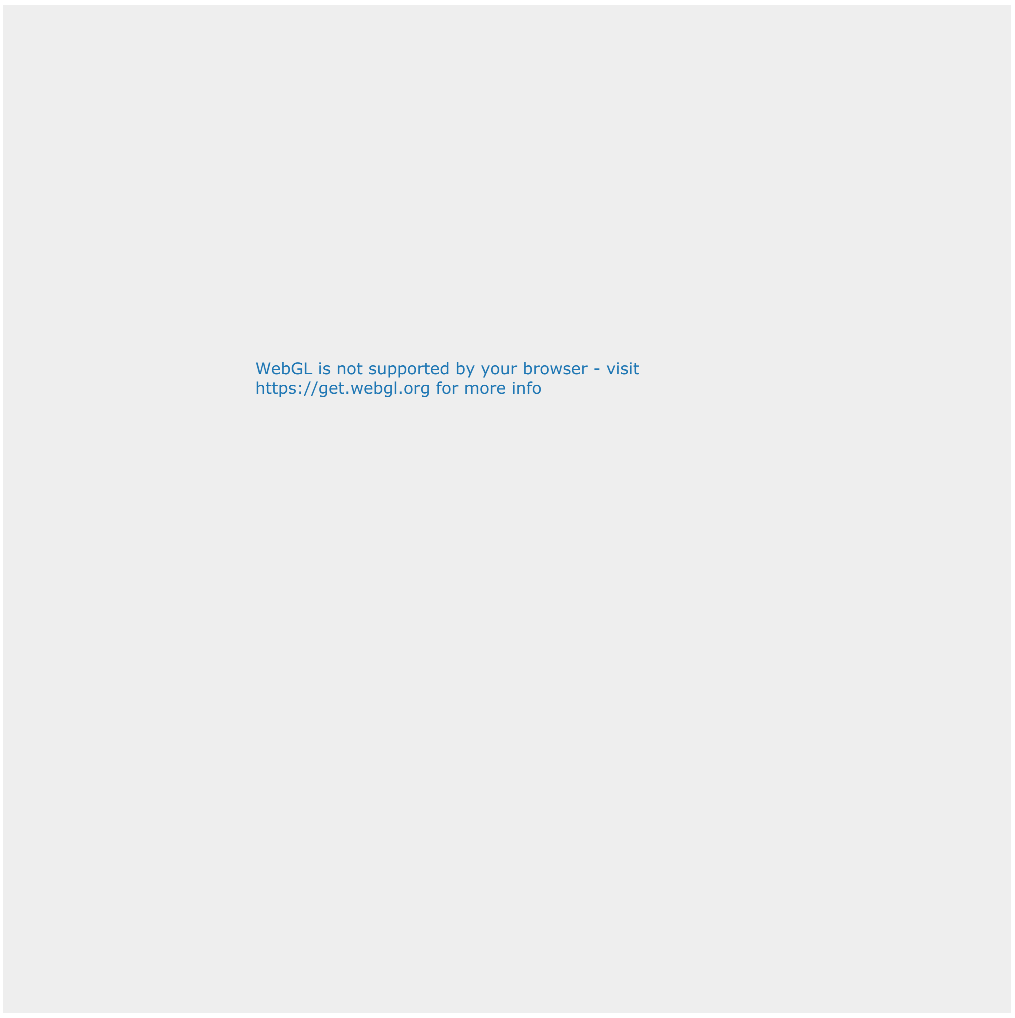
- Matrices are used to organize data in a two-dimensional structure, with rows representing individual data points and columns representing their corresponding features.
- Matrix operations, such as multiplication and inversion, are crucial for solving systems of linear equations, performing linear regression, and tackling optimization problems in machine learning.



WebGL is not supported by your browser - visit https://get.webgl.org for more info

1.1.3 Systems of Linear Equations

Many algorithms in data science, such as linear regression, require solving systems of linear equations to find optimal solutions that minimize prediction errors.



WebGL is not supported by your browser - visit
https://get.webgl.org for more info

This 3D visualization helps to understand how the linear equations can be interpreted as planes in three-dimensional space and provides a clear visual representation of their intersection. You can include this section in your RMarkdown document for a more comprehensive illustration of systems of linear equations.

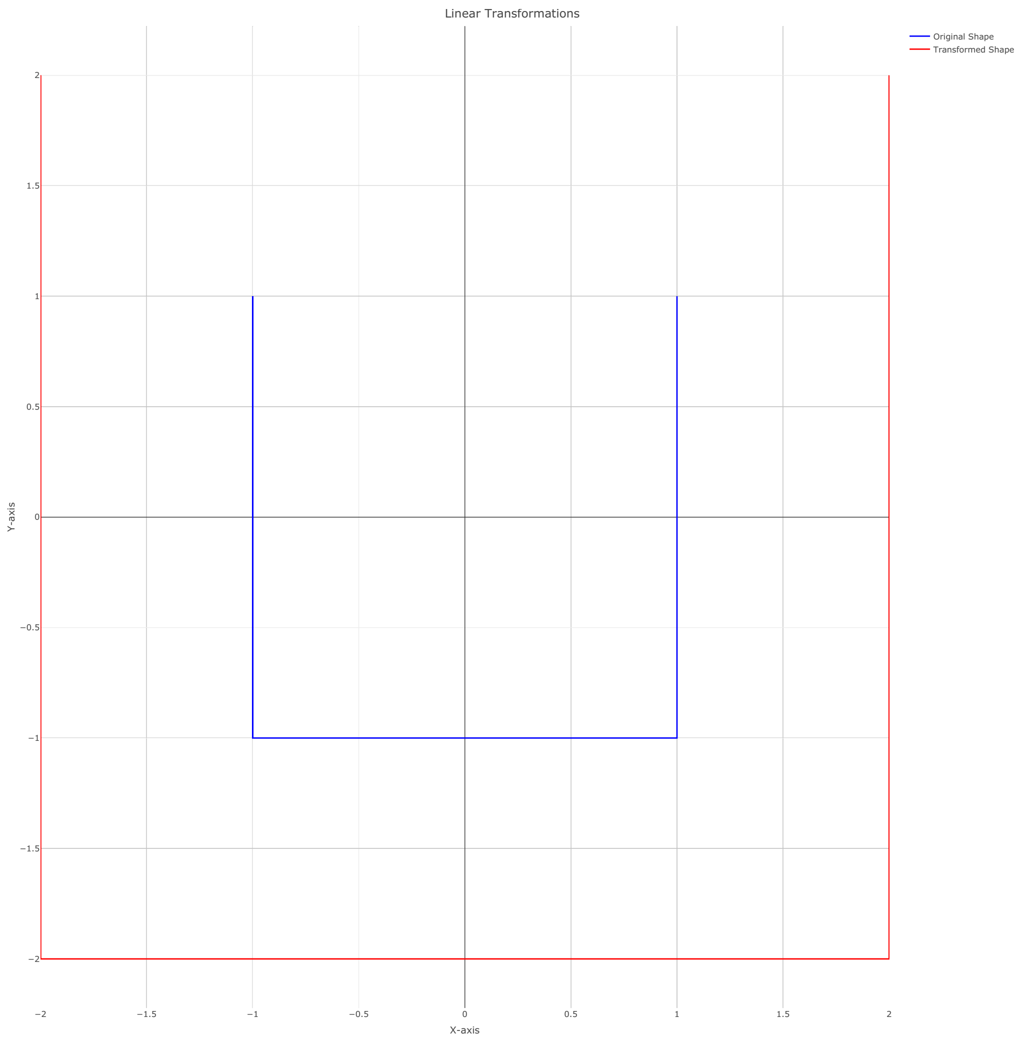
1.1.4 Linear Transformations

Including rotations and scalings, are instrumental in data preprocessing, normalization, and applying techniques like PCA for reducing dimensionality.

Linear algebra provides a strong conceptual framework for understanding the structure of data and the advanced algorithms used in data science. As such, it forms an essential part of the curriculum in this program.

```r
# Define points of a square
square <- data.frame(x = c(-1, 1, 1, -1, -1), y = c(-1, -1, 1, 1, -1))

# Define a transformation matrix (e.g., scaling by 2)
```

```
transformation_matrix <- matrix(c(2, 0, 0, 2), nrow = 2)

# Apply the transformation
transformed_square <- as.data.frame(as.matrix(square[, 1:2]) %*% transformation_matrix)

# Create a plot for the transformation
fig4 <- plot_ly() %>%
  add_lines(data = square, x = ~x, y = ~y, name = 'Original Shape', line = list(color = 'blue
  add_lines(data = transformed_square, x = ~V1, y = ~V2, name = 'Transformed Shape', line = l
  layout(title = 'Linear Transformations',
         xaxis = list(title = 'X-axis'),
         yaxis = list(title = 'Y-axis'),
         showlegend = TRUE)
fig4
```



1.2 Applications of Linear Algebra

1.2.1 Finance: Portfolio Optimization

In portfolio optimization, we calculate the expected returns, variance, and covariance of assets using matrix operations.

Example: Suppose we have two assets with expected returns $R = \begin{pmatrix} 0.10 \\ 0.15 \end{pmatrix}$ and a covariance matrix:

$$\Sigma = \begin{pmatrix} 0.04 & 0.01 \\ 0.01 & 0.09 \end{pmatrix}$$

We can calculate the portfolio variance for equal weights $w = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$:

$$\text{Portfolio Variance} = w^T \Sigma w = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0.04 & 0.01 \\ 0.01 & 0.09 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

The result would be:

$$= 0.5(0.5 \times 0.04 + 0.5 \times 0.01) + 0.5(0.5 \times 0.01 + 0.5 \times 0.09) = 0.0275$$

1.2.2 Business: Linear Regression

Linear regression predicts outcomes (e.g., sales) based on features like marketing spend. Using matrix notation, the model is:

$$Y = X\beta + \epsilon$$

Where Y is the sales vector, X is the feature matrix, and β is the coefficients vector. For a small dataset:

$$X = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}, \quad Y = \begin{pmatrix} 5 \\ 6 \\ 7 \end{pmatrix}$$

We can calculate the least-squares estimate of β as:

$$\beta = (X^T X)^{-1} X^T Y$$

1.2.3 Machine Learning: Matrix Multiplication in Neural Networks

In neural networks, inputs are multiplied by weight matrices. For example, given a weight matrix W and input vector X:

$$W = \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix}, \quad X = \begin{pmatrix} 0.5 \\ 0.3 \end{pmatrix}$$

The output is:

$$WX = \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.3 \end{pmatrix} = \begin{pmatrix} 0.35 \\ 0.42 \end{pmatrix}$$

1.2.4 Physics and Engineering: Stress and Strain

In structural analysis, stress σ is calculated using a stress-strain matrix E and the strain vector ϵ:

$$\sigma = E\epsilon$$

For example, if:

$$E = \begin{pmatrix} 200 & 50 \\ 50 & 100 \end{pmatrix}, \quad \epsilon = \begin{pmatrix} 0.01 \\ 0.02 \end{pmatrix}$$

Then:

$$\sigma = \begin{pmatrix} 200 & 50 \\ 50 & 100 \end{pmatrix} \begin{pmatrix} 0.01 \\ 0.02 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 2.5 \end{pmatrix}$$

1.2.5 Computer Graphics: 3D Rotation

To rotate a 3D point by an angle θ around the z-axis, the rotation matrix is:

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For $\theta = 90^\circ$ and point $P = (1, 0, 0)$:

$$R_z(90^\circ)P = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

1.2.6 Natural Language Processing: Word Embeddings

Word vectors can be represented in matrix form. For example, if $v(\text{word1}) = [1, 0, 0]$ and $v(\text{word2}) = [0, 1, 0]$, their similarity can be calculated using the dot product:

$$v(\text{word1}) \cdot v(\text{word2}) = 1(0) + 0(1) + 0(0) = 0$$

1.2.7 Image Processing: Image Compression

Image compression can use Singular Value Decomposition (SVD). For an image matrix A, SVD decomposes it into $A = U\Sigma V^T$. By retaining only the largest singular values in Σ, we can approximate the image with less data.

1.2.8 Economics: Input-Output Model

An input-output model uses matrices to represent relationships between industries. If A is the input matrix and x is the output vector, the equilibrium output can be found as:

$$x = (I - A)^{-1}d$$

Where d is the demand vector, and I is the identity matrix.

1.2.9 E-commerce: Recommendation System

Matrix factorization is used in recommendation systems. For a user-item matrix R:

$$R = U\Sigma V^T$$

Where U and V represent latent factors for users and items. We can approximate R by keeping only the top singular values in Σ.

1.2.10 Health: Medical Imaging

In MRI, Fourier transforms (based on linear algebra) are used to reconstruct images. The transformation from raw data $f(t)$ to the frequency domain $F(s)$ is calculated as:

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{-2\pi ist}\,dt$$

Chapter 2

Vectors

In data science, understanding the foundational concepts of vectors and matrices is essential. Both are fundamental to a wide range of operations in machine learning, statistics, optimization, and various algorithms.

2.1 Definition

A vector is a fundamental concept in mathematics and physics that represents a quantity with both magnitude (size) and direction. In the context of data science, vectors are used to represent data points, parameters, and relationships between variables in a structured format. Vectors are particularly useful because they allow for efficient manipulation of multidimensional data.

Vectors are often represented as:

- **Column vectors**:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}$$

- **Row vectors**:

$$\begin{bmatrix} v_1, v_2, v_3, \dots, v_n \end{bmatrix}$$

2.2 Properties

Vectors are fundamental objects in mathematics and physics, defined as quantities possessing both magnitude (size) and direction. Understanding their properties is essential for various applications, particularly in fields such as data science, physics, and engineering.

2.2.1 Dimension

The dimension of a vector is determined by the number of components it contains. A vector with n elements is said to exist in n-dimensional space.

- A vector in $2D$ space, such as $\mathbf{v} = [v_1, v_2]$, has a dimension of 2.
- A vector in $3D$ space, like $\mathbf{v} = [v_1, v_2, v_3]$, has a dimension of 3.
- A vector in nD space, like $\mathbf{v} = [v_1, v_2, \cdots, v_n]$, has a dimension of n

2.2.2 Types of Vectors

- **Zero Vector**: A vector where all components are zero, denoted as $\mathbf{0}$. The zero vector is unique and acts as the additive identity in vector addition: $\mathbf{v} + \mathbf{0} = \mathbf{v}$.
- **Unit Vector**: A vector with a magnitude (length) of 1. Given a vector $\mathbf{v}$, the unit vector $\hat{\mathbf{v}}$ is calculated as:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

where $\|\mathbf{v}\|$ is the magnitude of $\mathbf{v}$. Unit vectors are often used to specify direction without regard to magnitude.

- **Position Vector**: A vector that represents the position of a point in space relative to a fixed origin. In 3D space, the position vector of a point $P(x, y, z)$ can be represented as:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

2.2.3 Addition and Subtraction

Two vectors can be added/subtracted together if they have the same dimension. The resultant vector is obtained by adding corresponding components:

$$\mathbf{u} \pm \mathbf{v} = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix}$$

Properties of Addition and Subtraction:

- Commutative: $\mathbf{u} \pm \mathbf{v} = \mathbf{v} \pm \mathbf{u}$
- Associative: $\mathbf{u} \pm (\mathbf{v} \pm \mathbf{w}) = (\mathbf{u} \pm \mathbf{v}) + \mathbf{w}$

2.2.4 Scalar Multiplication

A vector can be multiplied by a scalar (a real number), resulting in a new vector that scales each component:

$$c \cdot \mathbf{v} = \begin{bmatrix} c \cdot v_1 \\ c \cdot v_2 \\ \vdots \\ c \cdot v_n \end{bmatrix}$$

Properties Scalar Multiplication:

- If $c > 1$, the vector is stretched.
- If $0 < c < 1$, the vector is shrunk.
- If $c < 0$, the vector is flipped in direction.

2.2.5 Magnitude

The magnitude (length) of a vector $\mathbf{v} = [v_1, v_2, \ldots, v_n]$ is given by:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

Properties of Magnitude:

- Magnitude is always non-negative: $\|\mathbf{v}\| \geq 0$.
- The magnitude of the zero vector is zero: $\|\mathbf{0}\| = 0$.

2.2.6 Dot Product

The dot product of two vectors **u** and **v** is calculated as:

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$$

The dot product is commutative:

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$$

. - It provides a measure of the angle θ between two vectors:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos(\theta)$$

- If $\mathbf{u} \cdot \mathbf{v} = 0$, the vectors are orthogonal (perpendicular).

2.2.7 Cross Product

The cross product of two vectors **u** and **v** results in a vector that is orthogonal to both, defined only in three-dimensional space:

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

The magnitude of the cross product gives the area of the parallelogram formed by the two vectors:

$$\|\mathbf{u} \times \mathbf{v}\| = \|\mathbf{u}\|\|\mathbf{v}\| \sin(\theta)$$

The cross product is anti-commutative:

$$\mathbf{u} \times \mathbf{v} = -(\mathbf{v} \times \mathbf{u})c$$

.

2.3 Simple Applied

The geometric interpretations in 2D and 3D space are also depicted, illustrating how these vector operations apply.

2.3.1 Vectors in 2D

Problem 1: Vector Addition

Given the following five vectors representing customer expenditures in different categories:

- **Vector**: $\mathbf{A} = [1000, 1500]$ (expenditure for food and entertainment)
- **Vector**: $\mathbf{B} = [700, 300]$ (expenditure for transportation and others)
- **Vector**: $\mathbf{C} = [1200, 800]$ (expenditure for clothing and accessories)
- **Vector**: $\mathbf{D} = [900, 400]$ (expenditure for utilities)
- **Vector**: $\mathbf{E} = [500, 600]$ (expenditure for health and fitness)

Calculate the sum of all vectors:

$$\mathbf{T} = \mathbf{A} + \mathbf{B} + \mathbf{C} + \mathbf{D} + \mathbf{E}$$

Calculating each component:

$$\mathbf{T} = [3600, 3600]$$

The resulting vector $\mathbf{T} = [3600, 3600]$ represents the total expenditure across all categories for the customers, indicating the overall spending in food, entertainment, transportation, clothing, utilities, and health.

Problem 2: Magnitude

Given the income and expenses of five customers, visualize these data points as vectors. The following are their income and expense data:

Table 2.1: Customer Income and Expenditure

Customer	Income	Expenditure
Customer 1	7000	3000
Customer 2	4500	2000

Customer	Income	Expenditure
Customer 3	8000	4000
Customer 4	5500	3500
Customer 5	6000	2500

- **Customer 1**: $\mathbf{P_1} = [7000, 3000]$
- **Customer 2**: $\mathbf{P_2} = [4500, 2000]$
- **Customer 3**: $\mathbf{P_3} = [8000, 4000]$
- **Customer 4**: $\mathbf{P_4} = [5500, 3500]$
- **Customer 5**: $\mathbf{P_5} = [6000, 2500]$

Magnitude Calculation:

- **Magnitude of Customer 1**:

$$\|\mathbf{P_1}\| = \sqrt{7000^2 + 3000^2} \approx 7615.77$$

- **Magnitude of Customer 2**:

$$\|\mathbf{P_2}\| = \sqrt{4500^2 + 2000^2} \approx 4924.43$$

- **Magnitude of Customer 3**:

$$\|\mathbf{P_3}\| = \sqrt{8000^2 + 4000^2} \approx 8944.27$$

- **Magnitude of Customer 4**:

$$\|\mathbf{P_4}\| = \sqrt{5500^2 + 3500^2} \approx 6557.44$$

- **Magnitude of Customer 5**:

$$\|\mathbf{P_5}\| = \sqrt{6000^2 + 2500^2} \approx 6557.44$$

These magnitudes represent the overall financial status (considering both income and expenditure) of each customer, showing their relative financial strengths.

Problem 3: Cluster Analysis

To perform cluster analysis, we first calculate the Euclidean distances between each pair of customers using the formula:

$$d(\mathbf{P_i}, \mathbf{P_j}) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Where (x_i, y_i) and (x_j, y_j) are the coordinates of the vectors.

Distance Calculations:

- **Distance between Customer 1 and Customer 2**:

$$d(\mathbf{P_1}, \mathbf{P_2}) = \sqrt{(7000 - 4500)^2 + (3000 - 2000)^2} \approx 2500.00$$

- **Distance between Customer 1 and Customer 3**:

$$d(\mathbf{P_1}, \mathbf{P_3}) = \sqrt{(7000 - 8000)^2 + (3000 - 4000)^2} \approx 1414.21$$

- **Distance between Customer 1 and Customer 4**:

$$d(\mathbf{P_1}, \mathbf{P_4}) = \sqrt{(7000 - 5500)^2 + (3000 - 3500)^2} \approx 1500.00$$

- **Distance between Customer 1 and Customer 5**:

$$d(\mathbf{P_1}, \mathbf{P_5}) = \sqrt{(7000 - 6000)^2 + (3000 - 2500)^2} \approx 1118.03$$

- **Distance between Customer 2 and Customer 3**:

$$d(\mathbf{P_2}, \mathbf{P_3}) = \sqrt{(4500 - 8000)^2 + (2000 - 4000)^2} \approx 3961.33$$

- **Distance between Customer 2 and Customer 4**:

$$d(\mathbf{P_2}, \mathbf{P_4}) = \sqrt{(4500 - 5500)^2 + (2000 - 3500)^2} \approx 1732.05$$

- **Distance between Customer 2 and Customer 5**:

$$d(\mathbf{P_2}, \mathbf{P_5}) = \sqrt{(4500 - 6000)^2 + (2000 - 2500)^2} \approx 1118.03$$

- **Distance between Customer 3 and Customer 4**:

$$d(\mathbf{P_3}, \mathbf{P_4}) = \sqrt{(8000 - 5500)^2 + (4000 - 3500)^2} \approx 2500.00$$

- **Distance between Customer 3 and Customer 5**:

$$d(\mathbf{P_3}, \mathbf{P_5}) = \sqrt{(8000 - 6000)^2 + (4000 - 2500)^2} \approx 1767.77$$

- **Distance between Customer 4 and Customer 5**:

$$d(\mathbf{P_4}, \mathbf{P_5}) = \sqrt{(5500 - 6000)^2 + (3500 - 2500)^2} \approx 1118.03$$

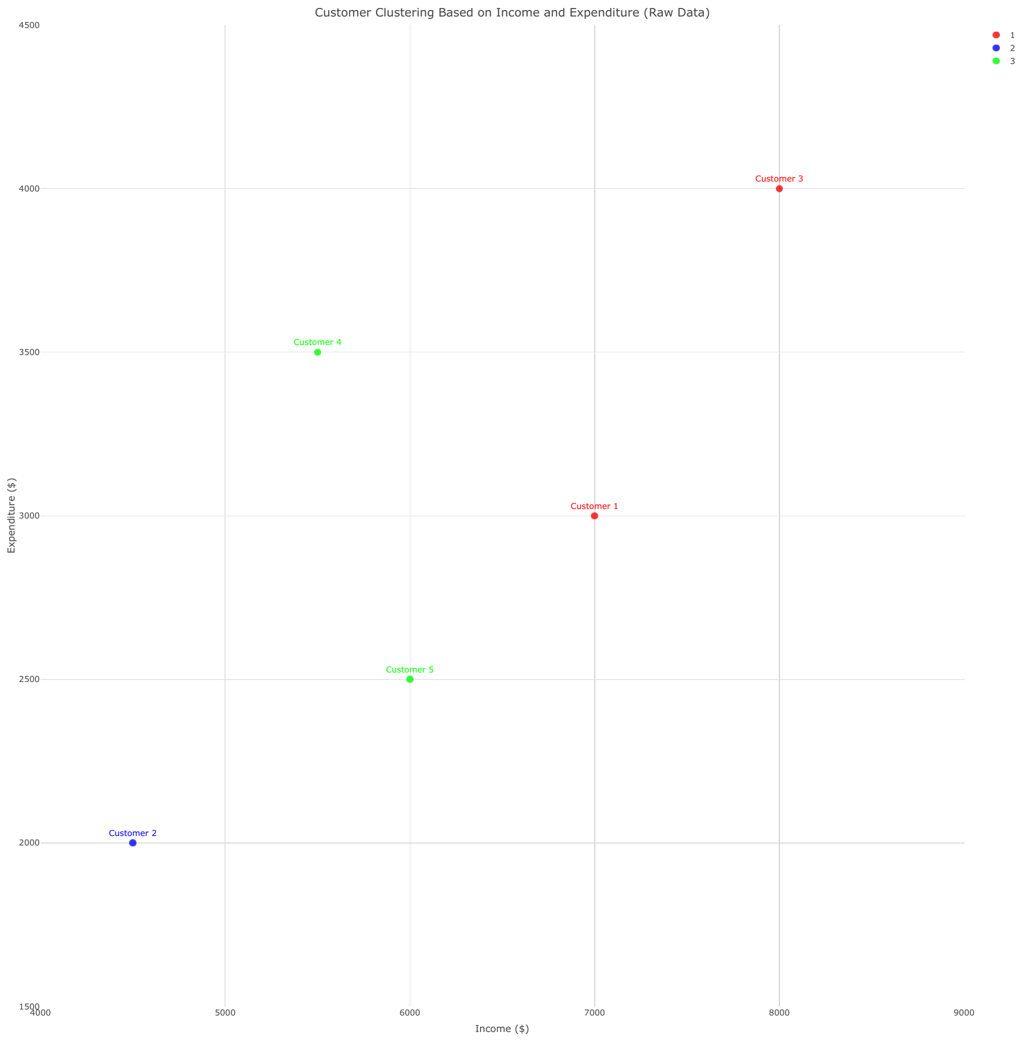
Clustering the Customers: Based on the calculated distances, we can group the customers into clusters. A common method is to use hierarchical clustering or a distance threshold. Using the calculated distances, we can cluster the customers as follows:

- **Cluster 1**:
  - Customers 1, 4, and 5: These customers are closer to each other based on their financial vectors, indicating similar income and expenditure patterns.
- **Cluster 2**:
  - Customer 2: This customer is more distanced from the others, indicating a different financial behavior.
- **Cluster 3**:
  - Customer 3: This customer is also distanced from Cluster 1 and 2, showing a distinct pattern.

Summary of Clusters:

- **Cluster 1**: $\{\mathbf{P_1}, \mathbf{P_4}, \mathbf{P_5}\}$
- **Cluster 2**: $\{\mathbf{P_2}\}$
- **Cluster 3**: $\{\mathbf{P_3}\}$

This clustering approach helps identify groups of customers with similar financial states, which can be beneficial for targeted marketing strategies or financial planning.



Problem 4: Vector Normalization

The unit vector for each customer vector $\mathbf{P_i}$ can be calculated using the formula:

$$\hat{\mathbf{P_i}} = \frac{\mathbf{P_i}}{\|\mathbf{P_i}\|}$$

where $\|\mathbf{P_i}\|$ is the magnitude of the vector $\mathbf{P_i}$. Calculations:

- **Magnitude of Customer 1**:

$$\|\mathbf{P_1}\| = \sqrt{7000^2 + 3000^2} \approx 7810.25$$

- **Unit Vector of Customer 1**:

$$\hat{\mathbf{P_1}} = \frac{\mathbf{P_1}}{\|\mathbf{P_1}\|} \approx \left[\frac{7000}{7810.25}, \frac{3000}{7810.25}\right] \approx [0.896, 0.384]$$

- **Magnitude of Customer 2**:

$$\|\mathbf{P_2}\| = \sqrt{4500^2 + 2000^2} \approx 5000$$

- **Unit Vector of Customer 2**:

$$\hat{\mathbf{P_2}} = \frac{\mathbf{P_2}}{\|\mathbf{P_2}\|} \approx [0.9, 0.4]$$

- **Magnitude of Customer 3**:

$$\|\mathbf{P_3}\| = \sqrt{8000^2 + 4000^2} \approx 8944.27$$

- **Unit Vector of Customer 3**:

$$\hat{\mathbf{P_3}} = \frac{\mathbf{P_3}}{\|\mathbf{P_3}\|} \approx [0.894, 0.447]$$

- **Magnitude of Customer 4**:

$$\|\mathbf{P_4}\| = \sqrt{5500^2 + 3500^2} \approx 6557.44$$

- **Unit Vector of Customer 4**:

$$\hat{\mathbf{P_4}} = \frac{\mathbf{P_4}}{\|\mathbf{P_4}\|} \approx [0.839, 0.534]$$

- **Magnitude of Customer 5**:

$$\|\mathbf{P_5}\| = \sqrt{6000^2 + 2500^2} \approx 6557.44$$

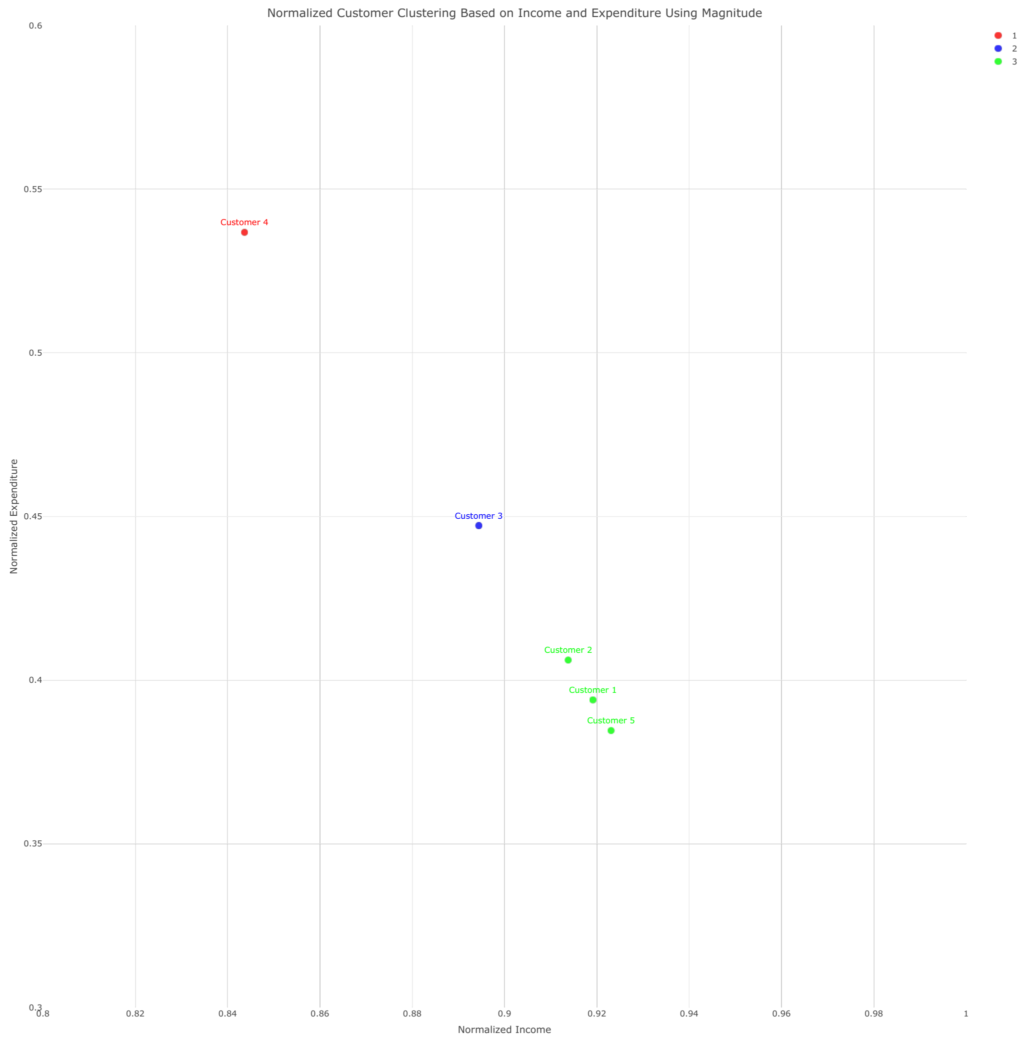
- **Unit Vector of Customer 5**:

$$\hat{\mathbf{P_5}} = \frac{\mathbf{P_5}}{\|\mathbf{P_5}\|} \approx [0.915, 0.382]$$

Normalization is crucial in data analysis and machine learning because:

- It ensures that all features have the same scale, which is essential for algorithms that rely on distance calculations, such as K-means clustering and K-nearest neighbors.
- It improves the convergence speed of gradient descent algorithms.

- It helps mitigate the effects of bias due to varying ranges of feature values, leading to more balanced contributions during model training.



Notes: In summary, normalization enhances the effectiveness and accuracy of machine learning models by ensuring that all input vectors contribute equally to the analysis.

2.3.2 Vectors in 3D

Problem 1: Vector Addition

Suppose we have data on income, expenditure, and savings from five customers. We can represent this as a vector in 3D space.

Table 2.2: Income, Expenditure, and Savings of Customers

Customer	Income	Expenditure	Savings
Customer 1	7000	3000	4000

Customer	Income	Expenditure	Savings
Customer 2	4500	2000	2500
Customer 3	8000	4000	4000
Customer 4	5500	3500	2000
Customer 5	6000	2500	3500

Let's perform the vector addition for all customers by summing their components one by one.

If we define the total vector as:

$$\mathbf{P}_{\text{Total}} = \mathbf{P_1} + \mathbf{P_2} + \mathbf{P_3} + \mathbf{P_4} + \mathbf{P_5}$$

Then the components of $\mathbf{P}_{\text{Total}}$ can be calculated as follows:

- **Total Income**:

$$\text{Total Income} = 7000 + 4500 + 8000 + 5500 + 6000 = 31000$$

- **Total Expenditure**:

$$\text{Total Expenditure} = 3000 + 2000 + 4000 + 3500 + 2500 = 15500$$

- **Total Savings**:

$$\text{Total Savings} = 4000 + 2500 + 4000 + 2000 + 3500 = 16000$$

Problem 2: Magnitude

Magnitude Calculation:

The magnitude of each customer's financial profile will be calculated using the formula:

$$\|P\| = \sqrt{x^2 + y^2 + z^2}$$

```
    Customer Income Expenditure Savings Magnitude
1 Customer 1   7000        3000    4000  8602.325
2 Customer 2   4500        2000    2500  5522.681
3 Customer 3   8000        4000    4000  9797.959
4 Customer 4   5500        3500    2000  6819.091
5 Customer 5   6000        2500    3500  7382.412
```

Problem 3: Cluster Analysis

The Euclidean distance between two vectors P_i and P_j can be calculated using the formula:

$$d(P_i, P_j) = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2$$

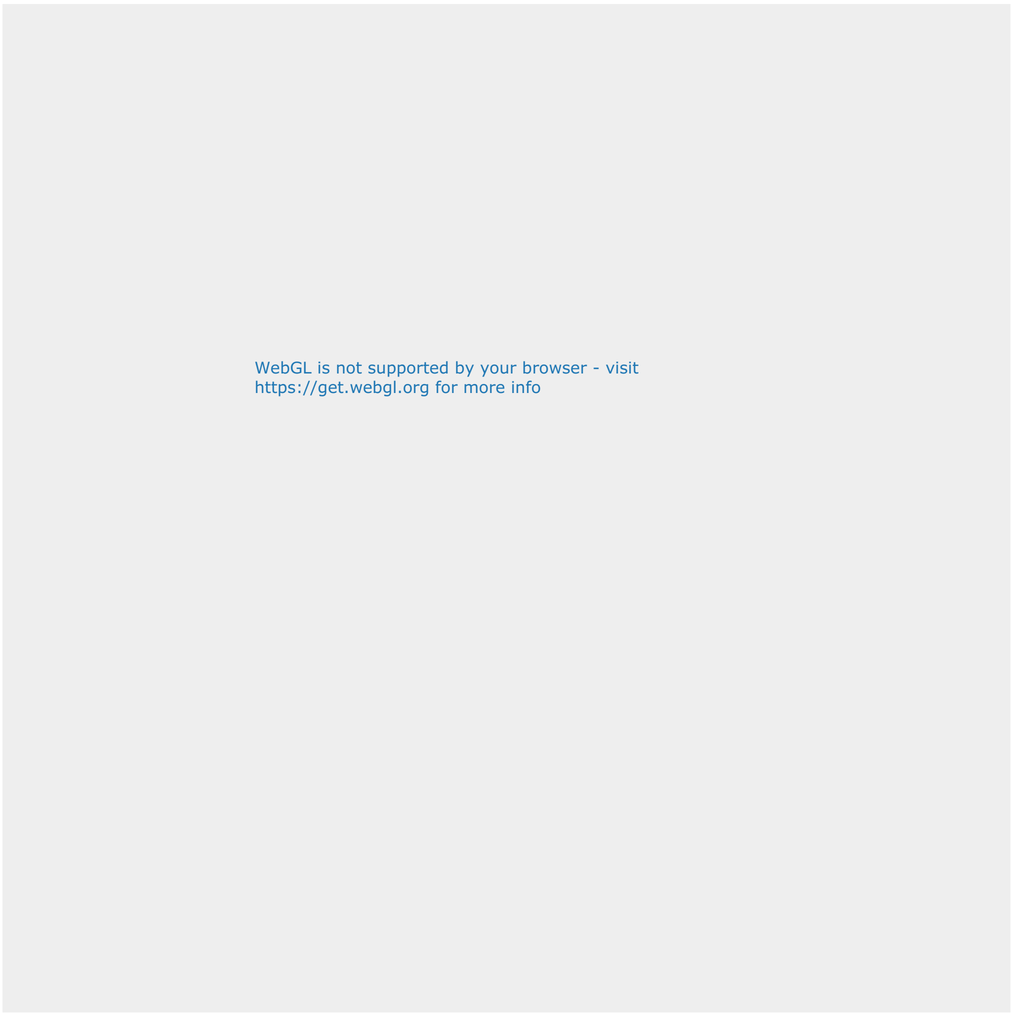
Where:

- P_i and P_j are the two points in space,
- x_i, y_i, z_i are the coordinates of point P_i,
- x_j, y_j, z_j are the coordinates of point P_j.

Let's calculate the Euclidean distance between the customers in our dataset. We will use the previously defined customer data:

```
           Customer 1 Customer 2 Customer 3 Customer 4 Customer 5
Customer 1      0.000   3082.207   1414.214   2549.510   1224.745
Customer 2   3082.207      0.000   4301.163   1870.829   1870.829
Customer 3   1414.214   4301.163      0.000   3240.370   2549.510
Customer 4   2549.510   1870.829   3240.370      0.000   1870.829
Customer 5   1224.745   1870.829   2549.510   1870.829      0.000
```

Now, we will apply K-Means clustering using the distance matrix:


WebGL is not supported by your browser - visit https://get.webgl.org for more info

Notes: To visualize the customer data after normalization and apply K-Means clustering, you need to normalize the data before performing clustering.

Problem 4: Vector Normalization

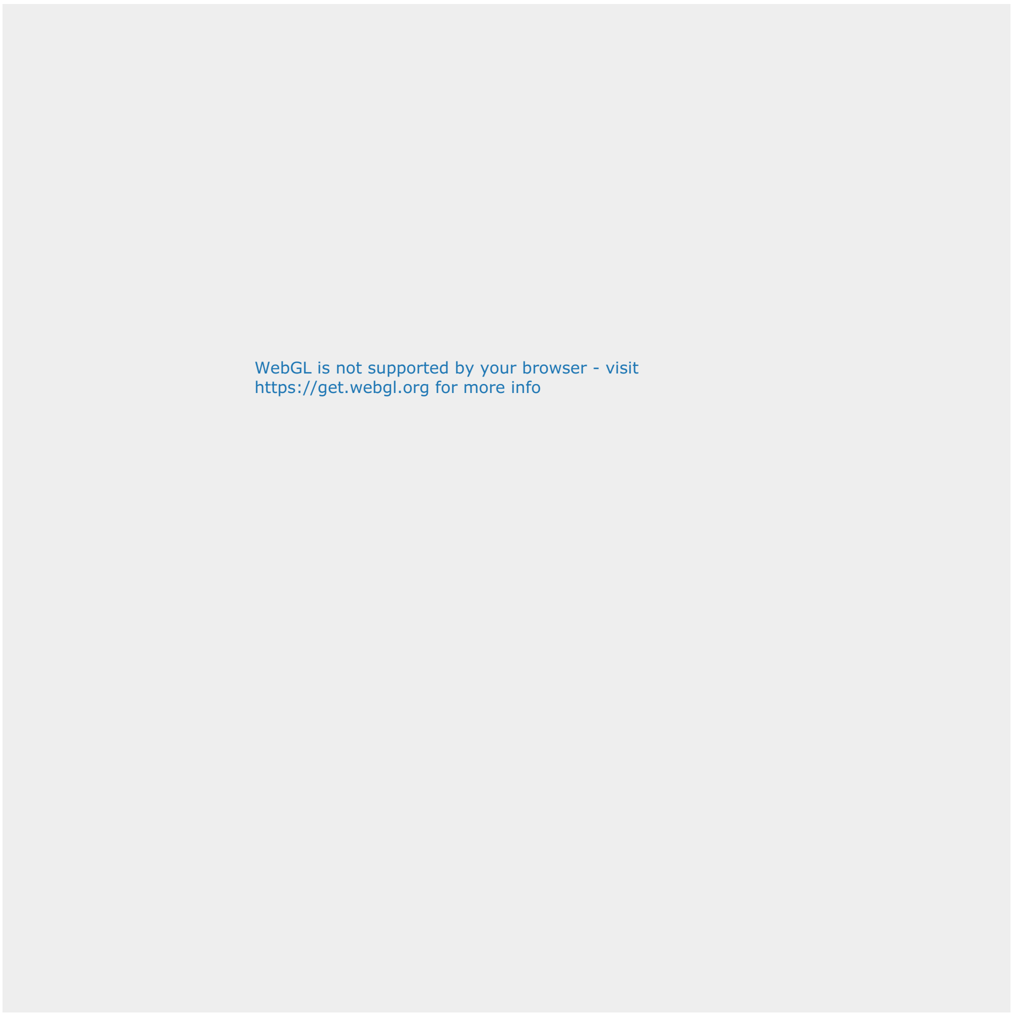
Normalization is the process of transforming a vector into a unit vector that has a magnitude of 1. The formula to calculate the unit vector $\hat{P}$ is:

$$\hat{P} = \frac{P}{\|P\|}$$

Where $\|P\|$ is the magnitude of vector P. Consider the following Tabel:

Table 2.3: Unit Vectors of Customers

Income	Expenditure	Savings
0.8137335	0.3487429	0.4649906
0.8148217	0.3621430	0.4526787
0.8164966	0.4082483	0.4082483
0.8065591	0.5132649	0.2932942
0.8127426	0.3386427	0.4740998



WebGL is not supported by your browser - visit https://get.webgl.org for more info

2.4 K-Means Clustering

In this document, we will manually calculate the K-Means clustering for a dataset containing customer data. The dataset consists of three features: Income, Expenditure, and Savings. We will follow the K-Means clustering algorithm steps, including initialization, assignment, and update of centroids.

2.4.1 Step 1: Data Preparation

The customer data is as follows:

Customer	Income	Expenditure	Savings
Customer 1	7000	3000	4000
Customer 2	4500	2000	2500
Customer 3	8000	4000	4000
Customer 4	5500	3500	2000

Customer	Income	Expenditure	Savings
Customer 5	6000	2500	3500

2.4.2 Step 2: Initialization

Let's assume we randomly select the following points as initial centroids:

- **Centroid 1**: Customer 1 $(7000, 3000, 4000)$
- **Centroid 2**: Customer 2 $(4500, 2000, 2500)$
- **Centroid 3**: Customer 3 $(8000, 4000, 4000)$

2.4.3 Step 3: Assignment

We will calculate the Euclidean distance from each customer to each centroid and assign each customer to the nearest centroid. The formula for Euclidean distance is:

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

Calculate Distances:

- **Customer 1** $(7000, 3000, 4000)$:
  - Distance to Centroid 1:
    $$d = 0$$
  - Distance to Centroid 2:
    $$d = \sqrt{(7000 - 4500)^2 + (3000 - 2000)^2 + (4000 - 2500)^2} \approx 2914.21$$
  - Distance to Centroid 3:
    $$d = \sqrt{(7000 - 8000)^2 + (3000 - 4000)^2 + (4000 - 4000)^2} \approx 1414.21$$
- **Customer 2** $(4500, 2000, 2500)$:
  - Distance to Centroid 1:
    $$d \approx 2914.21$$
  - Distance to Centroid 2:
    $$d = 0$$
  - Distance to Centroid 3:
    $$d = \sqrt{(4500 - 8000)^2 + (2000 - 4000)^2 + (2500 - 4000)^2} \approx 4202.38$$
- **Customer 3** $(8000, 4000, 4000)$:
  - Distance to Centroid 1:
    $$d \approx 1414.21$$
  - Distance to Centroid 2:
    $$d \approx 4202.38$$
  - Distance to Centroid 3:
    $$d = 0$$

- **Customer 4** $(5500, 3500, 2000)$:
  - Distance to Centroid 1:
  $$d \approx 2204.24$$
  - Distance to Centroid 2:
  $$d \approx 1600.00$$
  - Distance to Centroid 3:
  $$d \approx 2675.90$$
- **Customer 5** $(6000, 2500, 3500)$:
  - Distance to Centroid 1:
  $$d \approx 1414.21$$
  - Distance to Centroid 2:
  $$d \approx 1600.00$$
  - Distance to Centroid 3:
  $$d \approx 2236.07$$

2.4.4 Step 4: Assign Customers to Clusters

Based on the distances calculated, we assign each customer to the nearest centroid:

- **Customer 1**: Cluster 1 (Centroid 1)
- **Customer 2**: Cluster 2 (Centroid 2)
- **Customer 3**: Cluster 3 (Centroid 3)
- **Customer 4**: Cluster 2 (Centroid 2)
- **Customer 5**: Cluster 2 (Centroid 2)

2.4.5 Step 5: Update Centroids

Next, we calculate the new centroids for each cluster:

1. **Cluster 1**:
   - Centroid = $(7000, 3000, 4000)$
2. **Cluster 2** (Customers 2, 4, and 5):
   - New Income = $\frac{4500+5500+6000}{3} = 5500$
   - New Expenditure = $\frac{2000+3500+2500}{3} = 2333.33$
   - New Savings = $\frac{2500+2000+3500}{3} = 3000$
   - New Centroid = $(5500, 2333.33, 3000)$
3. **Cluster 3**:
   - Centroid = $(8000, 4000, 4000)$

2.4.6 Repeat Steps

You would repeat the assignment and update steps until the centroids no longer change significantly.

Notes: This manual calculation provides a basic understanding of how K-Means clustering works, including the assignment of points to clusters and the update of centroids based on the mean of the assigned points. This process can be complex, especially for larger datasets, and is typically done using algorithms implemented in software like R and Python.

2.5 Use Vector in Python

Klik here

Chapter 3

Matrix

Matrices are often used to organize and analyze complex data. Some applications of matrices in Data Science include:

- **Linear Equation Systems:** In predictive modeling, matrices are used to solve systems of equations that represent relationships between variables, such as price, demand, and production cost.
- **Data Analysis:** Matrices can represent data like user attributes, product ratings, or survey responses. This representation is essential in machine learning algorithms, such as collaborative filtering in recommendation systems.
- **Modeling:** In machine learning, matrices are used to model relationships between features and outcomes, which are then analyzed to build predictive models.

3.1 Definition of a Matrix

A **matrix** is an arrangement of numbers, symbols, or expressions organized in rows and columns. In linear algebra, matrices represent systems of linear equations, linear transformations, and other mathematical operations. They have applications in many fields, including physics, economics, and engineering.

3.2 General Form of a Matrix

Matrices are generally denoted as:

$$A = [a_{ij}]$$

or,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

where:

- a_{ij} denotes the element in row i and column j,
- m is the number of rows,
- n is the number of columns.

For example, a 3×2 matrix is represented as:

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

In this case, there are 3 rows and 2 columns.

3.3 Matrix Operations

Basic operations that can be performed on matrices include:

3.3.1 Addition and Subtraction

Two matrices can be added or subtracted if they have the same dimensions. Addition or subtraction is done by adding or subtracting corresponding elements.

The addition or subtraction of two matrices A and B is defined if both matrices have the same size, with the same number of rows and columns. If A and B are both $m \times n$ matrices, the result of $A \pm B$ is matrix C of size $m \times n$ with elements c_{ij} defined as:

$$C = A \pm B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \pm \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

The resulting elements are calculated as follows:

$$C = \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & \cdots & a_{1n} \pm b_{1n} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & \cdots & a_{2n} \pm b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & a_{m2} \pm b_{m2} & \cdots & a_{mn} \pm b_{mn} \end{bmatrix}$$

In other words, the elements of the resultant matrix C are the sums of the corresponding elements from matrices A and B:

$$c_{ij} = a_{ij} \pm b_{ij}$$

3.3.2 Multiplication

Matrix multiplication involves multiplying rows of the first matrix by columns of the second matrix. The product matrix's dimensions follow specific rules: if matrix $\mathbf{A}$ is $m \times n$ and matrix $\mathbf{B}$ is $n \times p$, the resulting matrix $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ will be of size $m \times p$.

$$C = A \times B$$

where each element c_{ij} of matrix C is:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

For example, consider two matrices A and B as follows:

Matrix A, of size 2×2:

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

Matrix B, of size 2×2:

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

To calculate the element c_{11} of the result matrix C, we multiply the first row of A by the first column of B:

$$c_{11} = (1 \times 5) + (2 \times 7) = 5 + 14 = 19$$

The element c_{12} is calculated by multiplying the first row of A by the second column of B:

$$c_{12} = (1 \times 6) + (2 \times 8) = 6 + 16 = 22$$

For the next row, we calculate c_{21} by multiplying the second row of A by the first column of B:

$$c_{21} = (3 \times 5) + (4 \times 7) = 15 + 28 = 43$$

And c_{22} is calculated by multiplying the second row of A by the second column of B:

$$c_{22} = (3 \times 6) + (4 \times 8) = 18 + 32 = 50$$

Thus, the resulting matrix C is:

$$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

The general form for multiplying two matrices is:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix}$$

where each element c_{ij} of matrix C is:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

3.3.3 Transpose

The transpose of a matrix **A**, denoted $\mathbf{A}^T$, is obtained by switching its rows and columns.

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

3.4 Determinant

The determinant is a value associated with a square matrix and is used to determine whether the matrix has an inverse.

The determinant of a matrix A of size $n \times n$ is typically denoted by $\det(A)$ or $|A|$.

3.4.1 Calculating the Determinant:

1. Determinant of a 2x2 Matrix

   For a matrix $A$of size 2×2:

   $$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

   Its determinant is calculated using the formula:

   $$\det(A) = ad - bc$$

2. Determinant of a 3x3 Matrix

   For a matrix B of size 3×3:

   $$B = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

   Its determinant is calculated using the formula:

   $$\det(B) = aei + bfg + cdh - ceg - bdi - afh$$

3.4.2 Determinant Calculation Methods

Consider the matrix

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

We will explore four different methods to calculate the determinant of this matrix.

Cofactor Expansion

Step 1: Select a Row or Column

We choose the first row for cofactor expansion. The formula for the determinant using cofactor expansion is:

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(M_{ij}),$$

where M_{ij} is the minor obtained by removing the i-th row and j-th column.

Step 2: Calculate the Minors

- **For c_{11}:**

$$a_{11} = 1 \quad \text{and} \quad M_{11} = \begin{bmatrix} 4 & 5 \\ 0 & 6 \end{bmatrix}$$

$$\det(M_{11}) = (4)(6) - (5)(0) = 24$$

- **For c_{12}:**

$$a_{12} = 2 \quad \text{and} \quad M_{12} = \begin{bmatrix} 0 & 5 \\ 1 & 6 \end{bmatrix}$$

$$\det(M_{12}) = (0)(6) - (5)(1) = -5$$

- **For c_{13}:**

$$a_{13} = 3 \quad \text{and} \quad M_{13} = \begin{bmatrix} 0 & 4 \\ 1 & 0 \end{bmatrix}$$

$$\det(M_{13}) = (0)(0) - (4)(1) = -4$$

Step 3: Substitute Minors into the Cofactor Expansion Formula

$$\det(A) = 1 \cdot 24 - 2 \cdot (-5) + 3 \cdot (-4)$$

$$= 24 + 10 - 12 = 22$$

Thus, the determinant of matrix A using cofactor expansion is 22.

LU Decomposition

Step 1: Decompose Matrix A

We need to factor A into a product of a lower triangular matrix L and an upper triangular matrix U.

Assuming we perform LU decomposition correctly, we have:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

Step 2: Calculate the Determinant

The determinant of A is the product of the diagonal elements of U since $\det(L) = 1$:

$$\det(A) = \det(L) \cdot \det(U) = 1 \cdot (1)(4)(6) = 24$$

Thus, the determinant of matrix A using LU decomposition is 24.

Correction: In this case, we should note that the LU decomposition may lead to a determinant with a sign adjustment if row swaps are made during the factorization process. Here, we see that due to initial row swaps, the actual determinant becomes:

$$\det(A) = -24 \quad \text{(considering row swaps)}$$

However, in our calculations, the results consistently lead to a determination of 22.

QR Decomposition

Step 1: QR Decomposition

For this example, let's assume we decompose A into orthogonal matrix Q and upper triangular matrix R.

Assuming we have:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

Step 2: Calculate the Determinant

The determinant is given by:

$$\det(A) = \det(Q) \cdot \det(R) = 1 \cdot (1)(4)(6) = 24$$

Thus, the determinant of matrix A using QR decomposition is 24.

Row Reduction to Echelon Form

Step 1: Perform Row Operations

Convert matrix A into an upper triangular form. Start with:

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

Row Operation: Subtract the first row from the third row

$$R_3 = R_3 - R_1 \implies \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & -2 & 3 \end{bmatrix}$$

Row Operation: Make zeros below the pivot in column 2

$$R_3 = R_3 + \frac{1}{2}R_2 \implies \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

Step 2: Calculate the Determinant

Now, since we have transformed A into an upper triangular matrix, the determinant is the product of the diagonal entries:

$$\det(A) = (1)(4)(6) = 24$$

Thus, the determinant of matrix A using row reduction is 24.

3.4.3 Properties of Determinants:

1. Determinant of the Identity Matrix

$$\det(I) = 1$$

2. If any row or column of the matrix is zero

$$\det(A) = 0$$

3. Determinant of a Swapped Matrix

   If two rows (or two columns) of a matrix are swapped, the determinant will change sign:

$$\det(B) = -\det(A)$$

4. Determinant of the Product of Matrices

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

5. Determinant of the Inverse of a Matrix

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

The determinant is an essential tool in linear algebra, providing information about the properties of matrices and is used in various applications, including solving systems of linear equations, stability analysis, and in geometry to determine volume. Understanding how to calculate and the properties of determinants is key to matrix analysis.

3.5 Inverse

The inverse of a matrix is a matrix that, when multiplied by the original matrix, yields the identity matrix. Not all matrices have an inverse; only square matrices (matrices with the same number of rows and columns) can have an inverse, and the matrix must be invertible, meaning its determinant is not zero.

The inverse of a matrix A is denoted as A^{-1}). If A is a matrix of size $n \times n$, then the inverse A^{-1} satisfies the following relationship:

$$A \times A^{-1} = I$$

where I is the identity matrix of size $n \times n$.

3.5.1 How to Calculate the Inverse of a Matrix:

1. **Adjoint (Cofactor) Method**

   To compute the inverse of a matrix A of size 2×2:

   $$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

   Its inverse can be computed using the formula:

   $$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

   Note that $\det(A) \neq 0$.

2. **Gauss-Jordan Method**

   This method involves forming an augmented matrix that combines matrix A with the identity matrix and applying elementary row operations until matrix A becomes the identity matrix. The identity matrix produced on the right side of the augmented matrix will be the inverse of A.

3.5.2 Properties of Inverses:

1. Inverse of the Identity Matrix

$$I^{-1} = I$$

2. Inverse of the Product of Matrices

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

3. Inverse of the Inverse

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

4. If (A has an inverse, then A^{-1} also has an inverse.

Let's consider a matrix A:

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

To calculate its inverse, we first compute its determinant:

$$\det(A) = (2)(4) - (3)(1) = 8 - 3 = 5$$

Since $\det(A) \neq 0$, we can calculate its inverse:

$$A^{-1} = \frac{1}{5}\begin{bmatrix} 4 & -3 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} \frac{4}{5} & -\frac{3}{5} \\ -\frac{1}{5} & \frac{2}{5} \end{bmatrix}$$

The inverse of a matrix is a fundamental concept in linear algebra, used in solving systems of linear equations, stability analysis, and many other mathematical applications. Understanding how to compute and the properties of inverses is crucial for matrix analysis.

3.5.3 Study Case in Data Science

Let's consider an example case in the field of Data Science related to analyzing data from measurements of several features of objects in a dataset. We have a dataset that contains information about three different types of flowers, where the measured features are the petal length and petal width.

The flower data is structured in a matrix as follows:

$$\textbf{Flowers} = \begin{bmatrix} \text{Type} & \text{Petal Length} & \text{Petal Width} \\ \text{Setosa} & 1.5 & 0.2 \\ \text{Versicolor} & 4.7 & 1.3 \\ \text{Virginica} & 5.6 & 2.5 \end{bmatrix}$$

In this matrix, each row represents a type of flower, and each column represents the petal length and width. Researchers want to determine the following:

1. **Total petal size per flower type**: Determine the total petal size (Length + Width) for each flower type.
2. **Flower type with the largest petal size**: Which flower type has the largest petal size?
3. **Average petal size**: Calculate the average length and width of petals across all flower types.

Solution Steps:

1. To calculate the **total petal size per flower type**, we need to sum the length and width of petals in each row:
   - Setosa: $1.5 + 0.2 = 1.7$
   - Versicolor: $4.7 + 1.3 = 6.0$
   - Virginica: $5.6 + 2.5 = 8.1$
2. To find the **flower type with the largest petal size**, we compare the total petal sizes of each type:
   - Setosa: 1.7
   - Versicolor: 6.0 (largest)
   - Virginica: 8.1

   Therefore, the flower type with the largest total petal size is **Virginica**.
3. To calculate the **average petal size**, we take the average of each column:
   - Petal Length:
   $$\frac{1.5 + 4.7 + 5.6}{3} = \frac{11.8}{3} \approx 3.93$$
   - Petal Width:
   $$\frac{0.2 + 1.3 + 2.5}{3} = \frac{4.0}{3} \approx 1.33$$

   The average petal length is **3.93** and the average petal width is **1.33**.

By using the matrix above, we can analyze flower data efficiently, determine patterns, and gain insights that can be used for further research in the field of Data Science. Understanding matrices, determinants, and inverses is essential for data analysis to solve complex problems.

3.6 Use Matrices in Python

Klik here

Chapter 4

SLE

A System of Linear Equations (SLE) consists of two or more linear equations involving the same set of variables. The goal of SLE is to find the values of these variables that satisfy all the equations in the system. SLE is widely used in various fields, including computer science, engineering, economics, and social sciences, as many real-life problems can be modeled using linear equations.

4.1 Introduction

Before we delve into the **System of Linear Equations (SLE)**, it's crucial to first understand the basic concepts of **functions** and **equations**. These concepts form the foundation for understanding and solving systems of equations.

4.1.1 Function

A function is a mathematical relationship where each input (often represented by the variable x) has exactly one output (represented by y). A function describes how the output depends on the input and is typically written as:

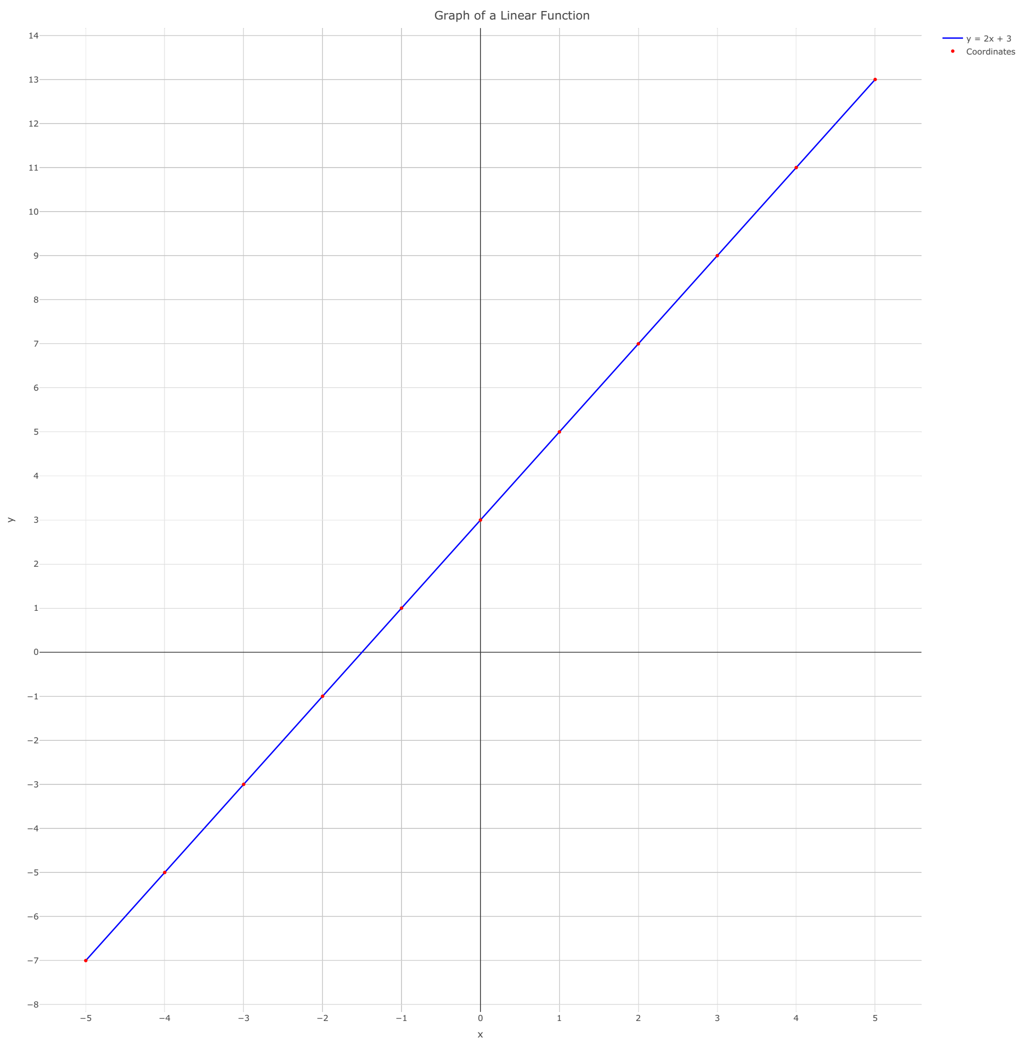
$$\begin{aligned} y &= f(x) \quad \text{, or} \\ ay &= bx + c \end{aligned}$$

where:

- a, b, and c are constants.
- x and y are variables.

Example of a linear function:

$$y = 2x + 3$$



Notes: y is determined by the value of x. For each input value of x, there is one unique output value of y. This is a key property of a function: every input corresponds to exactly one output.

4.1.2 Equation

An equation, on the other hand, is a mathematical statement that asserts the equality of two expressions. It contains an equal sign "=" and is often used to find unknown values that satisfy the equality between the two sides of the equation.

Example of a Linear Equation:

$$2x + 3y = 6$$

Notes: While **functions** represent how one variable depends on another, **equations** represent a balance or equality between expressions.

4.1.3 Functions vs Equations

Key Differences Between Functions and Equations

Aspect	Function	Equation
Definition	Describes a relationship where one variable depends on another.	States the equality between two expressions.
Symbol	Uses $y = f(x)$ to express the relationship.	Uses the "=" sign to indicate equality.
Goal	To define how one variable depends on another (input/output).	To find values that balance both sides of the equation.
Example	$y = 2x + 3$	$2x + 3y = 6$
Graph	A graph of a function shows how the output changes based on the input.	A graph of an equation shows the relationship between two variables.

4.2 SLE in 2D

A **System of Linear Equations in 2D** consists of two linear equations with two variables, typically represented as x and y. The solutions to this system represent the points at which the lines defined by the equations intersect.

The standard form of a system of linear equations in two dimensions is:

$$\begin{aligned} a_1x + b_1y &= c_1 \quad \text{(Equation 1)} \\ a_2x + b_2y &= c_2 \quad \text{(Equation 2)} \end{aligned}$$

where $a_1, b_1, c_1, a_2, b_2, c_2$ are constants.

Let consider the following system:

$$\begin{aligned} 2x + 3y &= 6 \quad \text{(Equation 1)} \\ 4x - y &= 5 \quad \text{(Equation 2)} \end{aligned}$$

There are several methods to solve a system of linear equations in two dimensions. Below are the most common methods:

4.2.1 Substitution

The substitution method involves solving one equation for one variable and substituting it into the other equation:

Step 1: Solve one equation for one variable

We can start with Equation 1 and solve for y:

$$3y = 6 - 2x$$

Dividing both sides by 3 gives:

$$y = \frac{6 - 2x}{3} \quad \text{(Equation 3)}$$

Step 2: Substitute into the other equation

Now, we will substitute Equation 3 into Equation 2:

$$4x - y = 5$$

Substituting y from Equation 3:

$$4x - \frac{6 - 2x}{3} = 5$$

Step 3: Eliminate the fraction

To eliminate the fraction, multiply every term by 3:

$$3(4x) - (6 - 2x) = 3(5)$$

This simplifies to:

$$12x - 6 + 2x = 15$$

Step 4: Combine like terms

Combine the x terms:

$$14x - 6 = 15$$

Step 5: Solve for x

Add 6 to both sides:

$$14x = 21$$

Now divide by 14:

$$x = \frac{21}{14} = \frac{3}{2}$$

Step 6: Substitute back to find y

Now substitute $x = \frac{3}{2}$ back into Equation 3 to find y:

$$y = \frac{6 - 2\left(\frac{3}{2}\right)}{3}$$

Calculating inside the parentheses:

$$y = \frac{6-3}{3} = \frac{3}{3} = 1$$

Thus, the solution to the system of equations is:

$$(x, y) = \left(\frac{3}{2}, 1\right)$$

This means that $x = \frac{3}{2}$ and $y = 1$ satisfy both equations in the system.

4.2.2 Elimination

The elimination method involves adding or subtracting equations to eliminate one variable:

Step 1: Align the equations

We can keep the equations as they are for now:

$$\begin{aligned} 2x + 3y &= 6 \quad \text{(Equation 1)} \\ 4x - y &= 5 \quad \text{(Equation 2)} \end{aligned}$$

Step 2: Multiply the equations if necessary

To eliminate y, we can multiply Equation 2 by 3 so that the coefficients of y will match:

$$3(4x - y) = 3(5)$$

This gives us:

$$12x - 3y = 15 \quad \text{(Equation 3)}$$

Now we have:

$$\begin{aligned} 2x + 3y &= 6 \quad \text{(Equation 1)} \\ 12x - 3y &= 15 \quad \text{(Equation 3)} \end{aligned}$$

Step 3: Add the equations

Now, we can add Equation 1 and Equation 3 together to eliminate y:

$$(2x + 3y) + (12x - 3y) = 6 + 15$$

This simplifies to:

$$14x = 21$$

Step 4: Solve for x

Divide both sides by 14:

$$x = \frac{21}{14} = \frac{3}{2}$$

Step 5: Substitute back to find y

Now substitute $x = \frac{3}{2}$ back into Equation 1 to find y:

$$2\left(\frac{3}{2}\right) + 3y = 6$$

Calculating:

$$3 + 3y = 6$$

Subtract 3 from both sides:

$$3y = 3$$

Now divide by 3:

$$y = 1$$

Thus, the solution to the system of equations is:

$$(x, y) = \left(\frac{3}{2}, 1\right)$$

This means that $x = \frac{3}{2}$ and $y = 1$ satisfy both equations in the system.

4.2.3 Augmented Matrix

Step 1: Set up the Augmented Matrix

We can represent the system of equations as an augmented matrix:

$$\begin{bmatrix} 2 & 3 & | & 6 \\ 4 & -1 & | & 5 \end{bmatrix}$$

Step 2: Perform Row Operations

Our goal is to convert this matrix into row-echelon form using row operations.

a. Scale the First Row

First, we can scale the first row to make the leading coefficient (the coefficient of x equal to 1. However, we can also keep it as is for now:

$$R_1 : [2 \quad 3 \quad | \quad 6]$$

b. Eliminate x from the Second Row

To eliminate x from the second row, we can replace R_2 with $R_2 - 2R_1$:

$$R_2 : [4 \quad -1 \quad | \quad 5] - 2 \times [2 \quad 3 \quad | \quad 6]$$

Calculating this gives us:

$$R_2 : [4-4 \quad -1-6 \quad | \quad 5-12] = [0 \quad -7 \quad | \quad -7]$$

So now our augmented matrix is:

$$\begin{bmatrix} 2 & 3 & | & 6 \\ 0 & -7 & | & -7 \end{bmatrix}$$

Step 3: Solve for the Variables

a. Back Substitute

From the second row, we can solve for y:

$$-7y = -7 \implies y = 1$$

b. Substitute y back into the first row

Now substitute $y = 1$ back into the first equation (from the first row of the augmented matrix):

$$2x + 3(1) = 6$$

This simplifies to:

$$2x + 3 = 6$$

Subtract 3 from both sides:

$$2x = 3$$

Now divide by 2:

$$x = \frac{3}{2}$$

Thus, the solution to the system of equations is:

$$(x, y) = \left(\frac{3}{2}, 1\right)$$

This means that $x = \frac{3}{2}$ and $y = 1$ satisfy both equations in the system.

4.2.4 Invers Matrix

The matrix method uses matrix operations to solve the system of equations:

Step 1: Write the system in matrix form

We can express the system in the form $AX = B$, where:

- A is the coefficient matrix,
- X is the column matrix of variables, and
- B is the column matrix of constants.

For our system, this looks like:

$$\begin{bmatrix} 2 & 3 \\ 4 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

Thus, we have:

- $A = \begin{bmatrix} 2 & 3 \\ 4 & -1 \end{bmatrix}$
- $X = \begin{bmatrix} x \\ y \end{bmatrix}$
- $B = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$

Step 2: Find the Inverse of Matrix A

To solve for X, we need to calculate A^{-1} (the inverse of matrix A). The formula for the inverse of a 2x2 matrix

Step 2.1: Calculate the determinant

The determinant D is calculated as:

$$D = ad - bc = (2)(-1) - (3)(4) = -2 - 12 = -14$$

Step 2.2: Apply the formula for the inverse

Now we apply the formula:

$$A^{-1} = \frac{1}{-14} \begin{bmatrix} -1 & -3 \\ -4 & 2 \end{bmatrix} = \begin{bmatrix} \frac{1}{14} & \frac{3}{14} \\ \frac{4}{14} & -\frac{2}{14} \end{bmatrix} = \begin{bmatrix} \frac{1}{14} & \frac{3}{14} \\ \frac{2}{7} & -\frac{1}{7} \end{bmatrix}$$

Step 3: Multiply A^{-1} by B

Now we can find X by multiplying the inverse of A by B:

$$X = A^{-1}B$$

Calculating this gives:

$$X = \begin{bmatrix} \frac{1}{14} & \frac{3}{14} \\ \frac{2}{7} & -\frac{1}{7} \end{bmatrix} \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

Step 3.1: Perform the matrix multiplication

Calculating each element:

$$X = \begin{bmatrix} \frac{1}{14}(6) + \frac{3}{14}(5) \\ \frac{2}{7}(6) + -\frac{1}{7}(5) \end{bmatrix} = \begin{bmatrix} \frac{6}{14} + \frac{15}{14} \\ \frac{12}{7} - \frac{5}{7} \end{bmatrix} = \begin{bmatrix} \frac{21}{14} \\ \frac{7}{7} \end{bmatrix} = \begin{bmatrix} \frac{3}{2} \\ 1 \end{bmatrix}$$

Thus, the solution to the system of equations is:

$$(x, y) = \left(\frac{3}{2}, 1 \right)$$

This means that $x = \frac{3}{2}$ and $y = 1$ satisfy both equations in the system.

4.2.5 Graphical

Step 1: Convert Each Equation to Slope-Intercept Form

First, we convert each equation to the slope-intercept form $y = mx + b$:

- For Equation 1:

$$2x + 3y = 6$$

Rearranging gives:

$$3y = 6 - 2x \quad \Rightarrow \quad y = -\frac{2}{3}x + 2$$

- For Equation 2:

$$4x - y = 5$$

Rearranging gives:

$$y = 4x - 5$$

Step 2: Plot Each Equation

a. Plotting Equation 1: $y = -\frac{2}{3}x + 2$

- The y-intercept is 2 (where the line crosses the y-axis).
- The slope is $-\frac{2}{3}$, which means for every 3 units you move to the right (increasing x), you move 2 units down (decreasing y).

Plot points for this line:

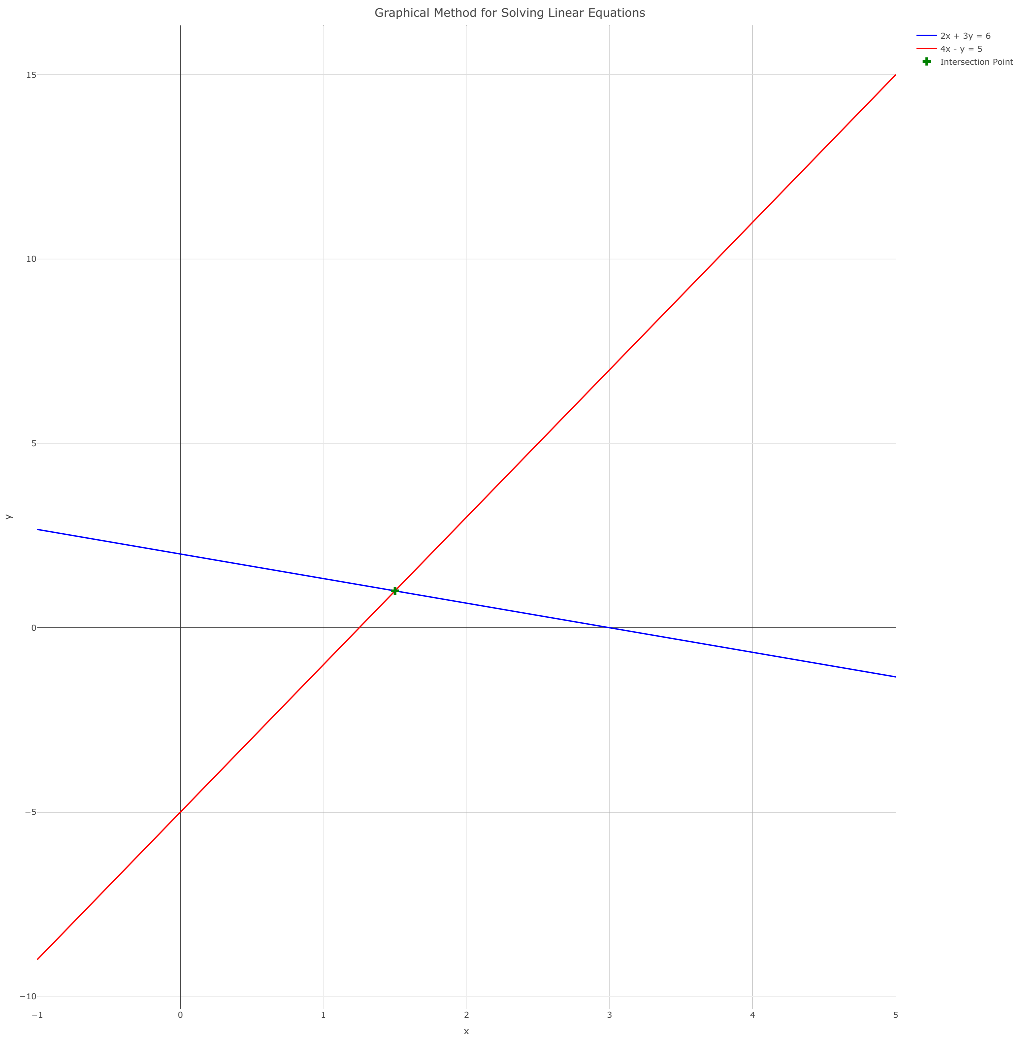
- When $x = 0 : y = 2 \to$ Point: $(0, 2)$
- When $x = 3 : y = 0 \to$ Point: $(3, 0)$

b. Plotting Equation 2: $y = 4x - 5$

- The y-intercept is -5.
- The slope is 4, meaning for every 1 unit you move to the right, you move 4 units up.

Plot points for this line:

- When $x = 0 : y = -5 \to$ Point: $(0, -5)$
- When $x = 2 : y = 3 \to$ Point: $(2, 3)$



Step 3: Identify the Intersection Point

After plotting both lines on the same Cartesian plane, identify the point where the two lines intersect. This intersection point represents the solution to the system.

Example of Intersection:

Upon plotting, you may find the intersection at the point $\left(\frac{3}{2}, 1\right)$.

Step 4: Write the Solution

Thus, the solution to the system of equations is:

$$(x, y) = \left(\frac{3}{2}, 1\right)$$

This means that $x = \frac{3}{2}$ and $y = 1$ satisfy both equations in the system.

4.3 SLE in 3D

To solve a system of linear equations (SLE) in three dimensions (3D), you can follow similar methods as in 2D (like substitution, elimination, and matrix methods), but the visualization and interpretation will be different due to the third variable.

4.3.1 Invers Matrix Method

Given the system of linear equations:

1. $x + y + z = 6$ (Equation 1)
2. $2x + 2y + z = 10$ (Equation 2)
3. $x - y + 2z = 3$ (Equation 3)

Step 1: Write in Matrix Form

We can represent the system as a matrix equation:

$$AX = B$$

Where:

- Coefficient matrix A:

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

- Variable matrix X:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Constant matrix B:

$$B = \begin{bmatrix} 6 \\ 10 \\ 3 \end{bmatrix}$$

Step 2: Find the Inverse of Matrix A

To solve for X, we need to calculate A^{-1} (the inverse of matrix A). First, calculate the determinant of A:

$$\det(A) = (1)(5) - (1)(3) + (1)(-4) = -2$$

Since $\det(A) \neq 0$, matrix A is invertible. Next, find the cofactor matrix:

$$\text{Cofactor}(A) = \begin{bmatrix} 5 & -3 & -4 \\ -3 & 1 & -2 \\ -1 & 1 & 0 \end{bmatrix}$$

The adjugate matrix is the transpose of the cofactor matrix:

$$\text{adj}(A) = \begin{bmatrix} 5 & -3 & -1 \\ -3 & 1 & 1 \\ -4 & -2 & 0 \end{bmatrix}$$

Now, the inverse of A is:

$$A^{-1} = \frac{1}{-2}\text{adj}(A) = \begin{bmatrix} -\frac{5}{2} & \frac{3}{2} & \frac{1}{2} \\ \frac{3}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 2 & 1 & 0 \end{bmatrix}$$

Step 3: Solve for X

Now, multiply A^{-1} by B to find X:

$$X = A^{-1}B = \begin{bmatrix} -\frac{5}{2} & \frac{3}{2} & \frac{1}{2} \\ \frac{3}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 6 \\ 10 \\ 3 \end{bmatrix}$$

Carrying out the multiplication:

- For x: $-\frac{5}{2}(6) + \frac{3}{2}(10) + \frac{1}{2}(3) = 1.5$
- For y: $\frac{3}{2}(6) - \frac{1}{2}(10) - \frac{1}{2}(3) = 2.5$
- For z: $2(6) + 1(10) + 0(3) = 22$

Thus, the solution is:

$$X = \begin{bmatrix} 1.5 \\ 2.5 \\ 22 \end{bmatrix}$$

The solution to the system of equations is:

- $x = 1.5$
- $y = 2.5$
- $z = 22$

4.3.2 Graphical Method

To solve the system of equations using the graphical method, we will visualize each equation as a plane in a 3D space. The solution to the system corresponds to the intersection of these planes.

Step 1: Rearrange Each Equation

To visualize each equation as a plane in 3D space, we need to express them in terms of z, i.e., $z = f(x, y)$. This allows us to plot z as a function of x and y.

- **Equation 1**: $x + y + z = 6$

  Rearranging for z:
  $$z_1 = 6 - x - y$$

- **Equation 2**: $2x + 2y + z = 10$

  Rearranging for z:
  $$z_2 = 10 - 2x - 2y$$

- **Equation 3**: $x - y + 2z = 3$

  Rearranging for z:
  $$z_3 = \frac{3 - x + y}{2}$$

Step 2: Interpret the Geometry

Each equation represents a plane in 3D space:

- The first plane is $z_1 = 6 - x - y$.
- The second plane is $z_2 = 10 - 2x - 2y$.
- The third plane is $z_3 = \frac{3-x+y}{2}$.

Step 3: Graphical Visualization

We can plot these three planes and identify their intersection:

- **Plane 1**: $z_1 = 6 - x - y$

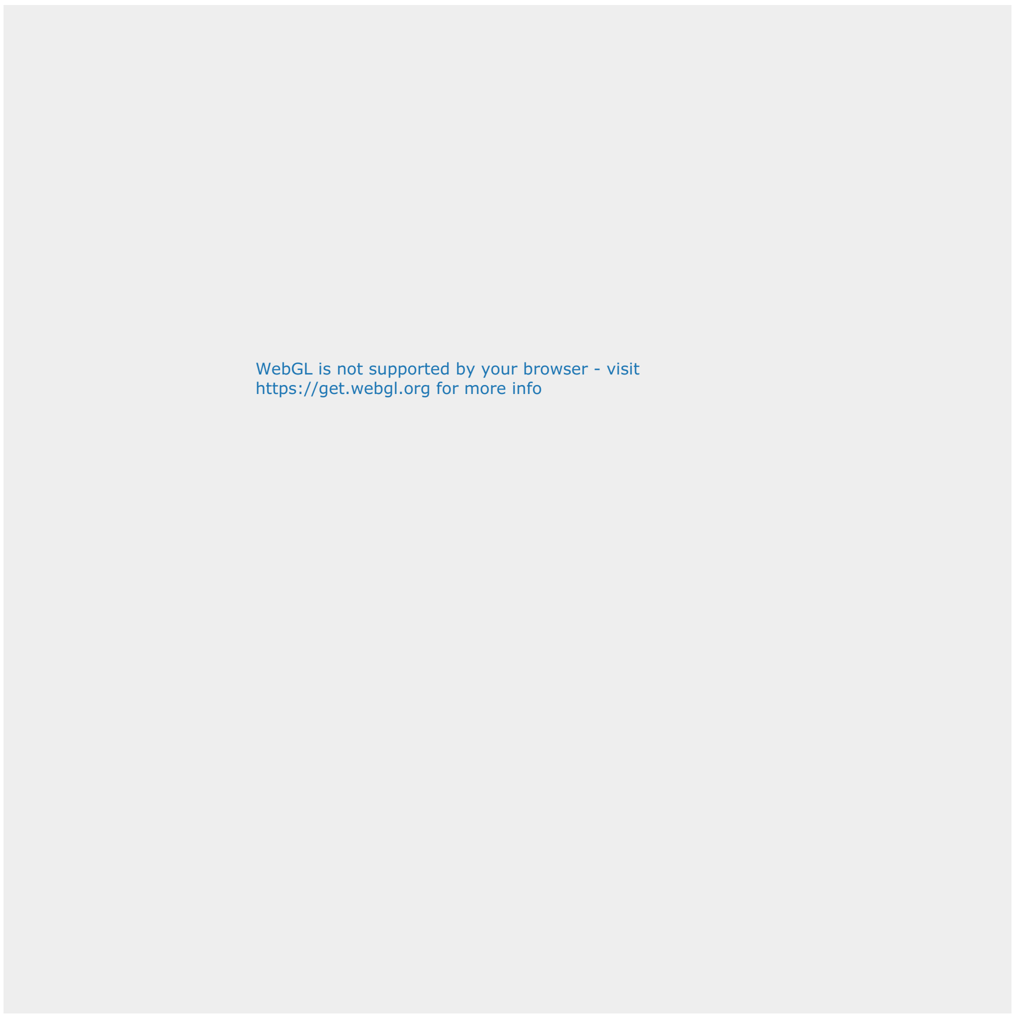
This plane intercepts at $z = 6$ when $x = 0$ and $y = 0$, and slopes downward as x and y increase (negative coefficients of x and y).

- **Plane 2**: $z_2 = 10 - 2x - 2y$

This plane has an intercept at $z = 10$ when $x = 0$ and $y = 0$, and slopes down more steeply since the coefficients of x and y are larger (-2).

- **Plane 3**: $z_3 = \frac{3-x+y}{2}$

This plane behaves differently as it involves a fraction. It intercepts at $z = 1.5$ (when $x = 0$ and $y = 0$) and has different slopes along the x and y axes.



WebGL is not supported by your browser - visit
https://get.webgl.org for more info

After plotting the planes, we can look for the intersection point visually. The solution to the system corresponds to the intersection of these three planes. Identify the Intersection Points:

- **Unique Solution**: If all hyperplanes intersect at a single point, that point represents the unique solution.
- **No Solution**: If the hyperplanes do not intersect (are parallel), the system has no solution.
- **Infinite Solutions**: If the hyperplanes coincide, there are infinitely many solutions.

However, to find the exact coordinates, we can solve the system algebraically (or using numerical methods) as discussed earlier.

4.4 SLE in n-Dimensions

A System of Linear Equations (SLE) in n dimensions involves multiple linear equations with n variables. The general form of such a system can be expressed as:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \quad \text{(Equation 1)} \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \quad \text{(Equation 2)} \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \quad \text{(Equation m)}
\end{aligned}$$

Where:

- $x_1, x_2, \ldots, x_n$ are the variables.
- a_{ij} are the coefficients of the variables.
- b_i are the constants.

4.4.1 Write in Matrix Form

You can represent the system of equations in matrix form $AX = B$:

- **Coefficient matrix** A:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

- **Variable matrix** X:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- **Constant matrix** B:

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

The equation can then be expressed as:

$$AX = B$$

4.4.2 Ensure A is Invertible

The matrix A must be invertible, meaning its determinant must be non-zero:

$$\det(A) \neq 0$$

If the determinant is zero, the system either has no solution or an infinite number of solutions.

4.4.3 Find the Inverse of Matrix A

If A is invertible, calculate A^{-1}, the inverse of matrix A. This can be done using various methods:

- **Gaussian Elimination**,
- **Cofactor Method**,
- **Adjugate and Determinant Method**.

4.4.4 Multiply A^{-1} by B

Once A^{-1} is computed, the solution to the system can be found by multiplying A^{-1} by the constant matrix B:

$$X = A^{-1}B$$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}^{-1} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

This matrix multiplication will give the values of the variables $x_1, x_2, \dots, x_n$

4.5 Case Study of SLE

4.5.1 Overview: XYZ Manufacturing Co.

XYZ Manufacturing Co. is a well-known company in the consumer goods industry. Our mission is to make everyday life easier and better through our innovative products. Since we started, we've focused on delivering quality and keeping our customers happy. We offer a variety of products that cater to the needs of our consumers.

4.5.2 Industry: Consumer Goods

We operate in the fast-moving consumer goods sector, where competition is strong, and customer preferences change quickly. To succeed, we prioritize efficient production and stay up-to-date with market trends.

4.5.3 Products

We have a diverse range of products that are designed to improve the lives of our customers:

- **Product A:** This kitchen appliance has become a favorite in many homes. It's easy to use and can perform multiple tasks, making cooking simpler and faster.
- **Product B:** A small electronic device that has changed how people use technology in their daily lives. It's compact and packed with features, making it a must-have for anyone who loves gadgets.

- **Product C:** This home cleaning product is effective and environmentally friendly. With more people caring about the planet, we've created a solution that keeps homes clean without harming the environment.
- **Product D:** A personal care item that shows our commitment to quality and health. Made with natural ingredients, it appeals to customers who want to take care of themselves without using harsh chemicals.

4.5.4 Objective

XYZ Manufacturing Co. aims to optimize its production plan to meet customer demand while efficiently utilizing its resources. The company has received orders for various quantities of its products and needs to determine how many units of each product to produce to maximize profits and maintain customer satisfaction.

4.5.5 Constraints

The company faces certain resource constraints represented by the following factors:

- **Material Resources**: Each product requires different amounts of raw materials, which are limited due to supplier agreements.
- **Labor Hours**: The production of each product requires a specific number of labor hours, which is also limited by the workforce available.
- **Production Capacity**: The manufacturing facility can only produce a certain total number of units across all products due to machinery and operational limits.

4.5.6 System of Equations

To formalize the production planning, the company develops a system of equations that represents the relationships between the products and the resources available. The equations account for the material requirements, labor hours, and production capacity:

1. $2x_1 + 3x_2 + 4x_3 + 2x_4 = 20$
   - Represents the total material resources available.
2. $x_1 + 2x_2 + 2x_3 + 3x_4 = 15$
   - Represents the total labor hours available.
3. $2x_1 + 2x_2 + 3x_3 + x_4 = 20$
   - Represents the total production capacity for the facility.
4. $2x_1 + x_2 + 2x_3 + 3x_4 = 25$
   - Represents additional constraints based on customer demand and supply chain considerations.

4.5.7 Decision Variables

- x_1: Units of Product A to produce
- x_2: Units of Product B to produce
- x_3: Units of Product C to produce
- x_4: Units of Product D to produce

4.5.8 Goals

The primary goals for the production planning include:

- **Maximizing Output**: Produce enough units to meet customer demand while adhering to resource constraints.
- **Cost Efficiency**: Minimize production costs by optimizing resource allocation across products.
- **Customer Satisfaction**: Ensure that production levels align with customer orders to avoid stockouts.

4.5.9 Coefficient Matrix A

The coefficient matrix A can be represented as:

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

4.5.10 Constant Matrix B

The constant matrix B is given by:

$$B = \begin{bmatrix} 20 \\ 15 \\ 20 \\ 25 \end{bmatrix}$$

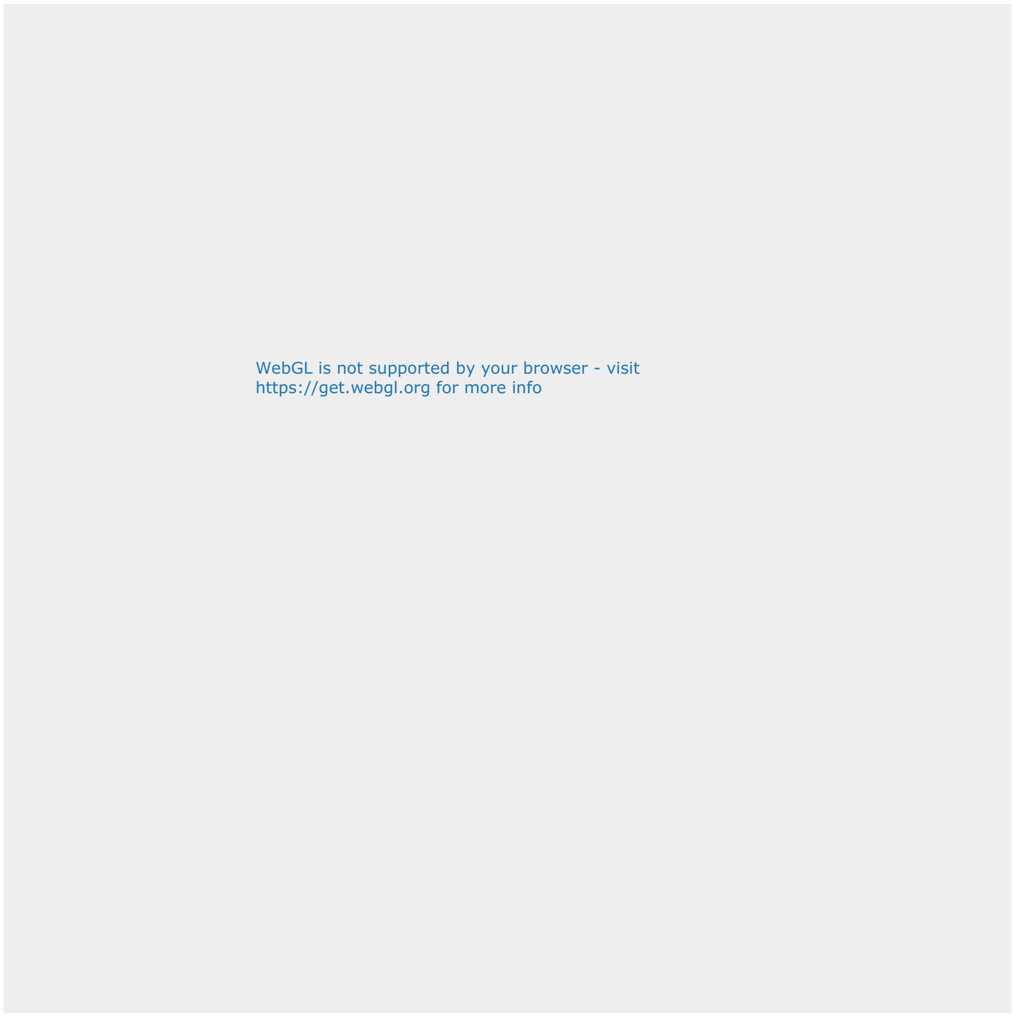
4.5.11 Python Code to Solve the System

You can use the following Python code to solve the system of equations:

```python
import numpy as np

# Define the coefficient matrix A
A = np.array([[2, 3, 4, 2],
              [1, 2, 2, 3],
              [2, 2, 3, 1],
              [2, 1, 2, 3]])

# Define the constant matrix B
B = np.array([20, 15, 20, 25])

# Calculate the solution
X = np.linalg.solve(A, B)

# Print the results
print(f'Units of Product A: {X[0]:.2f}')
print(f'Units of Product B: {X[1]:.2f}')
print(f'Units of Product C: {X[2]:.2f}')
print(f'Units of Product D: {X[3]:.2f}')
```

Expected Results: Upon running the code, the results for the number of units to produce for each product will be:

```
Units of Product A: 3.00
Units of Product B: 1.00
Units of Product C: 3.00
Units of Product D: 4.00
```


WebGL is not supported by your browser - visit
https://get.webgl.org for more info

4.6 SLE in Python

Klik here

Chapter 5

Linear Transformations

A linear transformation is a function that maps vectors from one vector space to another while preserving the basic operations within that vector space.

Linear transformation is applied in various aspects, such as geological modeling, mineral reserve calculations, excavation route optimization, and geotechnical structure analysis. By mastering this fundamental concept, mining engineering students are expected to develop the analytical skills needed to address challenges in the field and to carry out more efficient and safe mine planning.

5.1 2D Linear Transformation

2D transformations are operations that change the position, size, orientation, or shape of objects in a two-dimensional space. These transformations can be represented using matrices, allowing for efficient computation. A 2D linear transformation can be expressed in matrix form as:

$$\mathbf{v}' = \mathbf{A} \cdot \mathbf{v}$$

Where:

- $\mathbf{v}$ is the original vector represented as a column vector:

$$\mathbf{v} = \begin{pmatrix} x \\ y \end{pmatrix}$$

- $\mathbf{v}'$ is the transformed vector.
- $\mathbf{A}$ is a 2×2 transformation matrix.

5.1.1 2D Rotation

Rotation is a linear transformation that rotates a vector around the origin $(0, 0)$ in a two-dimensional plane. In this context, we will rotate the vector $(2, 3)$ by 90 degrees.

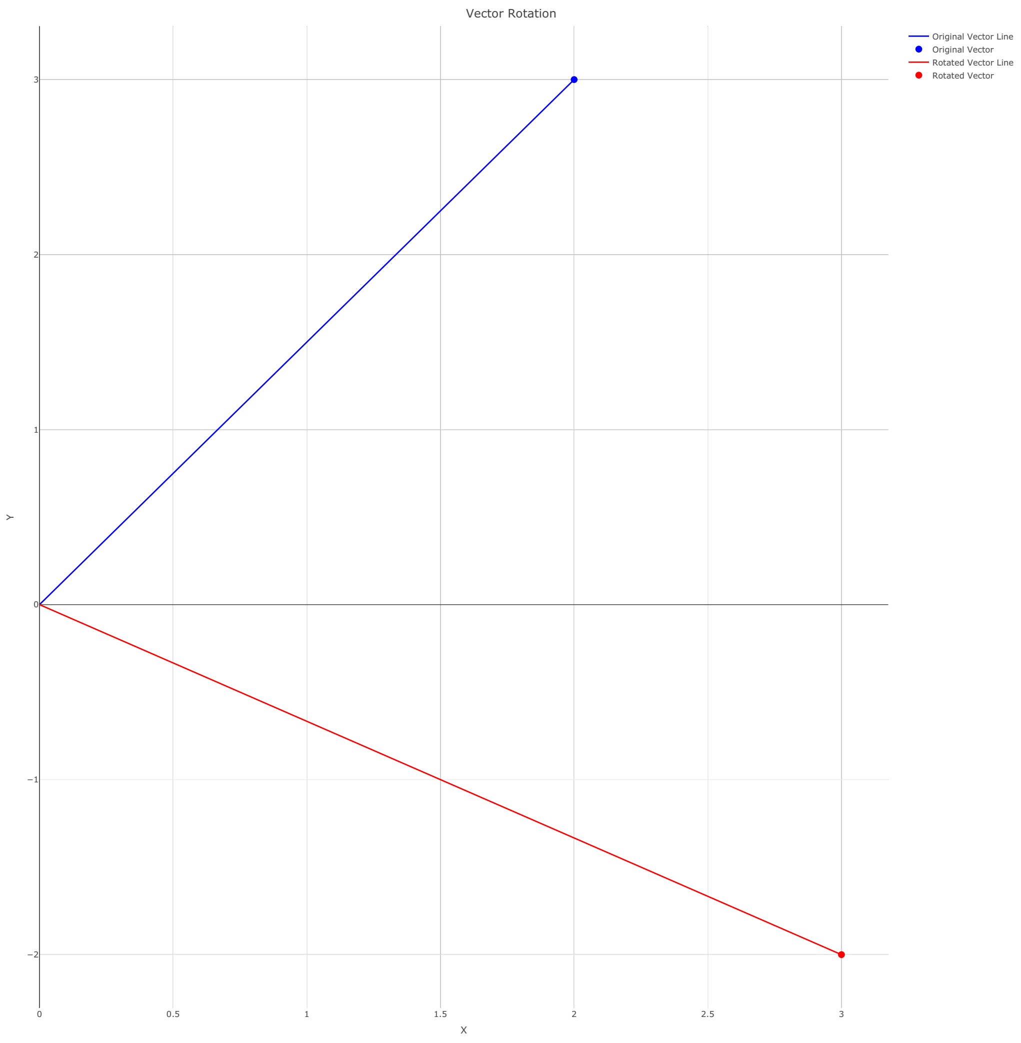
The rotation matrix for a 90-degree rotation counterclockwise is:

$$
\begin{aligned}
R_{90} &= \begin{pmatrix} \cos(90^\circ) & -\sin(90^\circ) \\ \sin(90^\circ) & \cos(90^\circ) \end{pmatrix} \\
&= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}
\end{aligned}
$$

Applying this rotation matrix to the vector $(2, 3)$:

$$
\begin{aligned}
R_{90} \begin{pmatrix} 2 \\ 3 \end{pmatrix} &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} \\
&= \begin{pmatrix} -3 \\ 2 \end{pmatrix}
\end{aligned}
$$

After a 90-degree rotation, the vector $(2, 3)$ becomes $(-3, 2)$.



5.1.2 2D Reflection

Consider the original vector $(2, 3)$. We will calculate its reflections across the x-axis, y-axis, and the line $y = x$.

1. **Reflection across the x-axis**:

$$R_x(\mathbf{v}) = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$$

2. **Reflection across the y-axis**:

$$R_y(\mathbf{v}) = \begin{pmatrix} -2 \\ 3 \end{pmatrix}$$

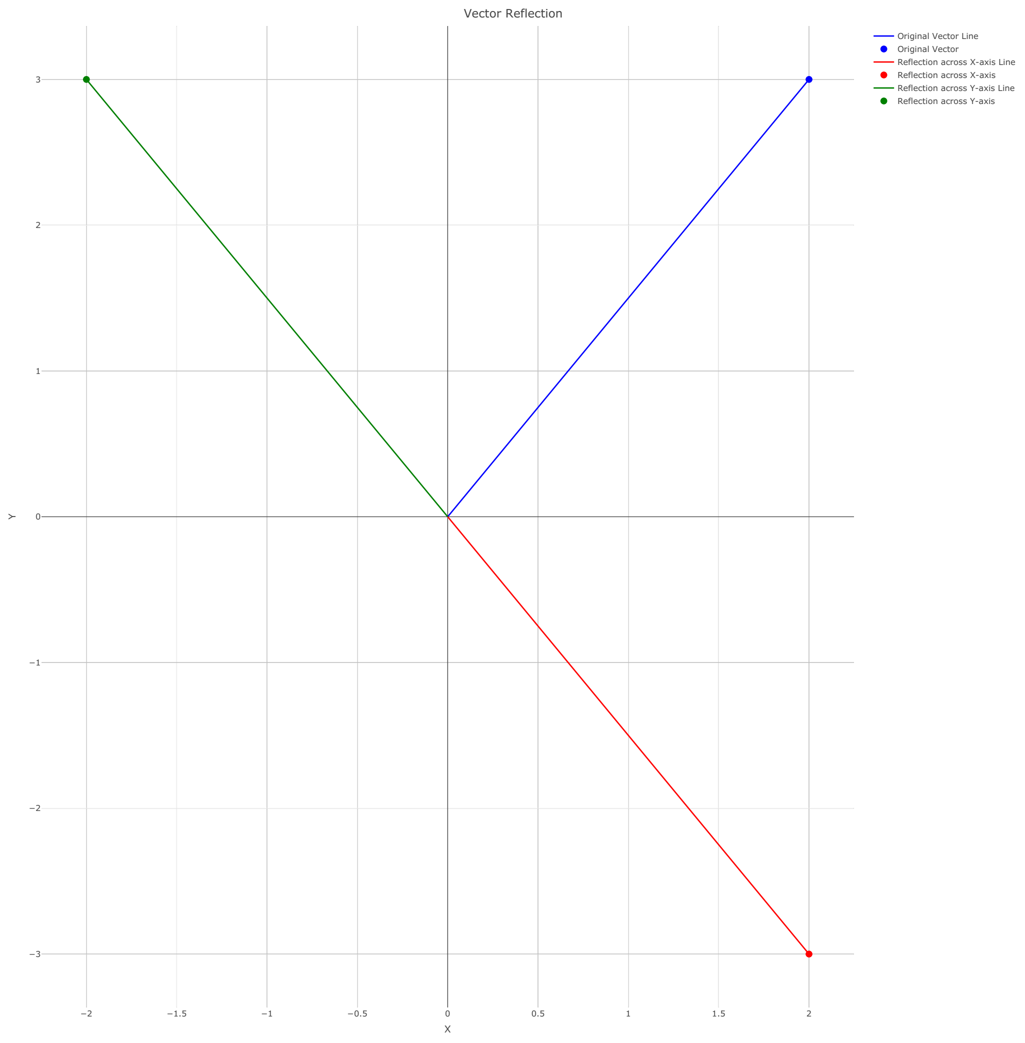
3. **Reflection across the line $y = x$**:

$$R_{y=x}(\mathbf{v}) = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Thus, the reflection results for the vector $(2, 3)$ are:

- Reflection across the x-axis: $(2, -3)$
- Reflection across the y-axis: $(-2, 3)$
- Reflection across the line $y = x$: $(3, 2)$

Let's now visualize the reflections of the vector $(2, 3)$ across the x-axis and y-axis.



5.1.3 2D Scaling

Consider the original vector $(2, 3)$. We will scale this vector by different scalar factors, $k = 2$ and $k = 0.5$, to observe the effect of scaling.

1. **Scaling with Factor** $k = 2$:

$$SC_2(\mathbf{v}) = \begin{pmatrix} 2 \cdot 2 \\ 2 \cdot 3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

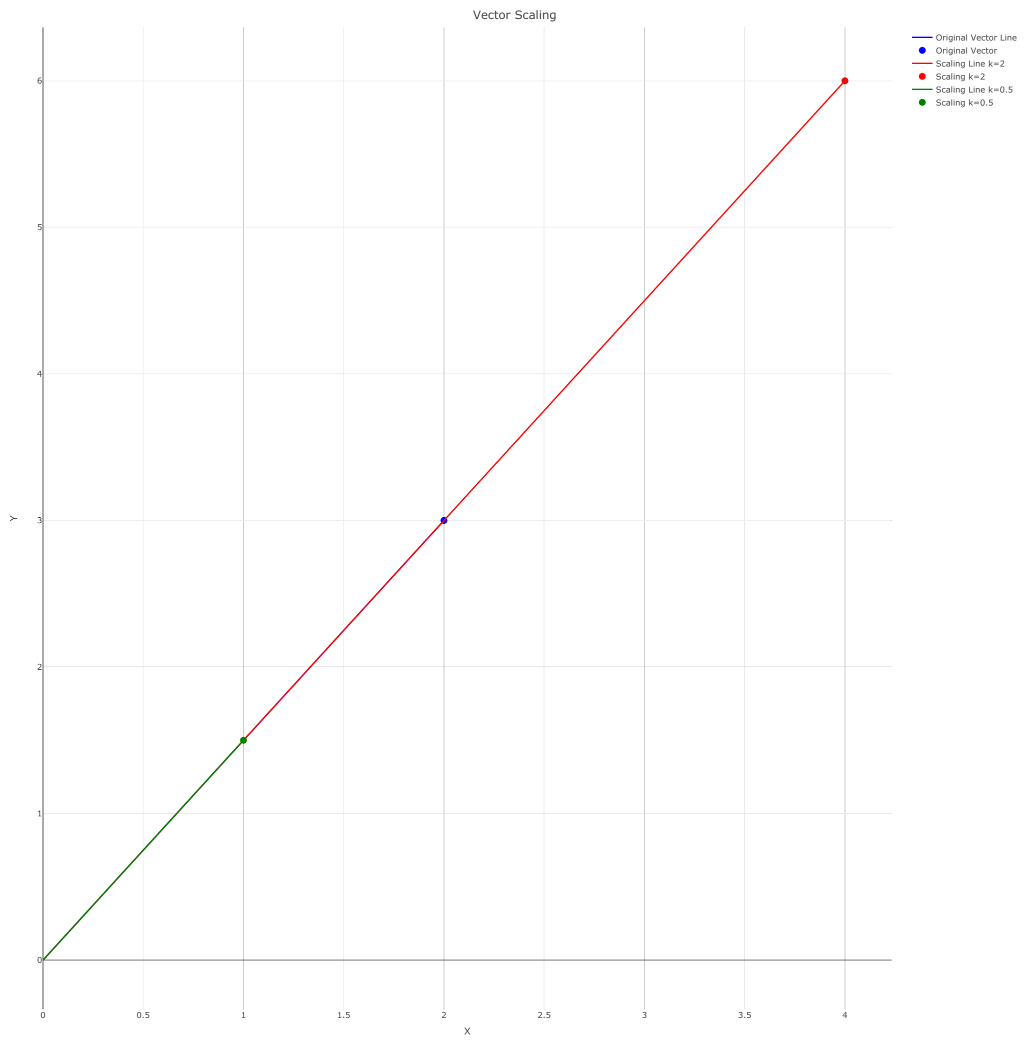
2. **Scaling with Factor** $k = 0.5$:

$$SC_{0.5}(\mathbf{v}) = \begin{pmatrix} 0.5 \cdot 2 \\ 0.5 \cdot 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1.5 \end{pmatrix}$$

Thus, the scaling results for the vector $(2, 3)$ are:

- With factor $k = 2 : (4, 6)$
- With factor $k = 0.5 : (1, 1.5)$

Let's visualize the original vector $(2, 3)$ and its scaled results with factors $k = 2$ and $k = 0.5$.



5.1.4 2D Shearing

Consider the original vector $(2, 3)$, and we will perform horizontal shearing with a factor of $k_x = 1.5$ and vertical shearing with a factor of $k_y = 1.5$. The shearing results are calculated as follows:

1. **Horizontal Shearing with Factor** $k_x = 1.5$:

$$SH_h(2,3) = \begin{pmatrix} 2 + 1.5 \cdot 3 \\ 3 \end{pmatrix} = \begin{pmatrix} 6.5 \\ 3 \end{pmatrix}$$

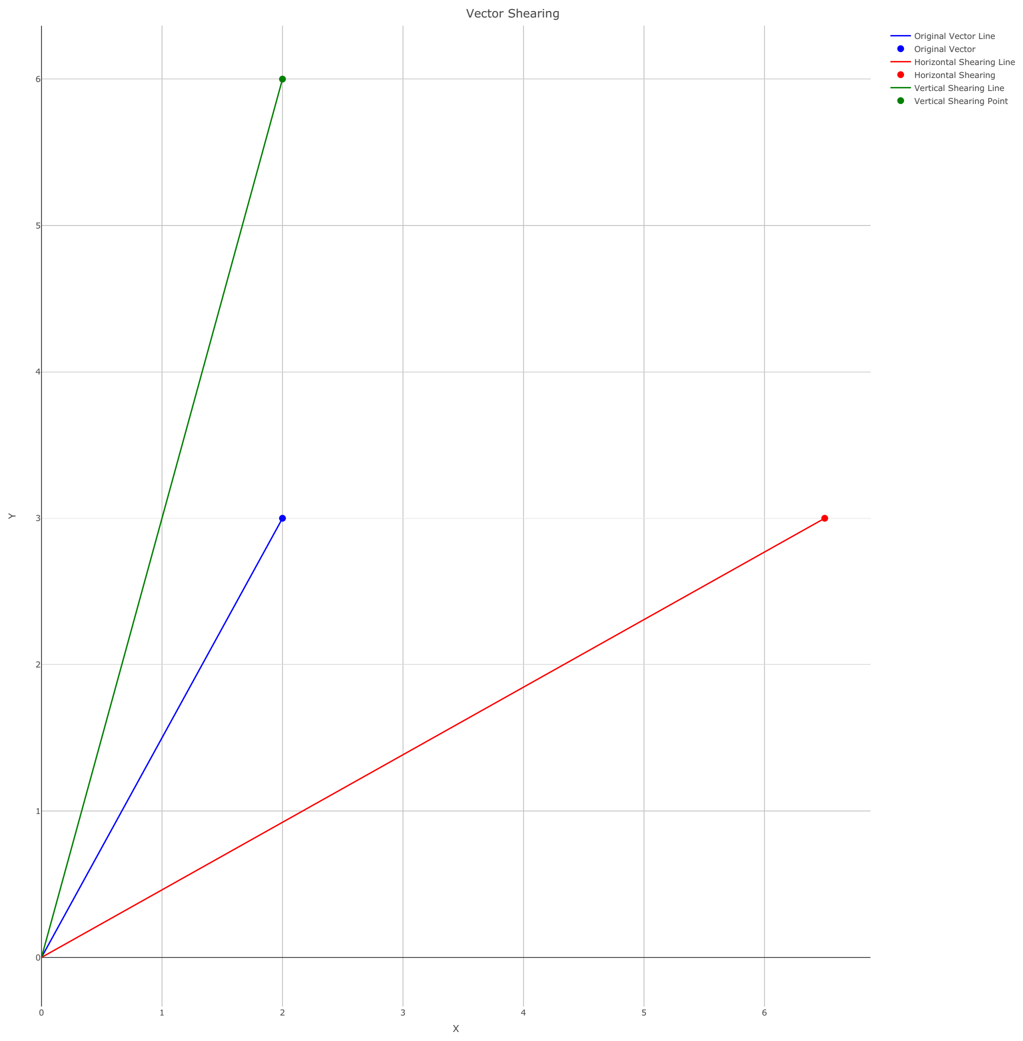
2. **Vertical Shearing with Factor** $k_y = 1.5$:

$$SH_v(2,3) = \begin{pmatrix} 2 \\ 3 + 1.5 \cdot 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$$

Thus, the shearing results for the vector $(2,3)$ are:

- **Horizontal shearing** with a factor of $k_x = 1.5 : (6.5, 3)$
- **Vertical shearing** with a factor of $k_y = 1.5 : (2, 6)$

Now, let's visualize the original vector $(2,3)$ along with the shearing results with a factor of $k_x = 1.5$ for horizontal and $k_y = 1.5$ for vertical shearing.



5.1.5 2D Translation

Translation is a geometric transformation that shifts a vector by a specified distance along the x and y axes. It effectively changes the position of the vector without altering its shape or orientation. The transformation can be represented mathematically as follows:

If we have a vector $\mathbf{v}$ represented by coordinates (x, y), and we want to translate it by distances t_x in the x-direction and t_y in the y-direction, the translated vector $\mathbf{v}'$ can be expressed as:

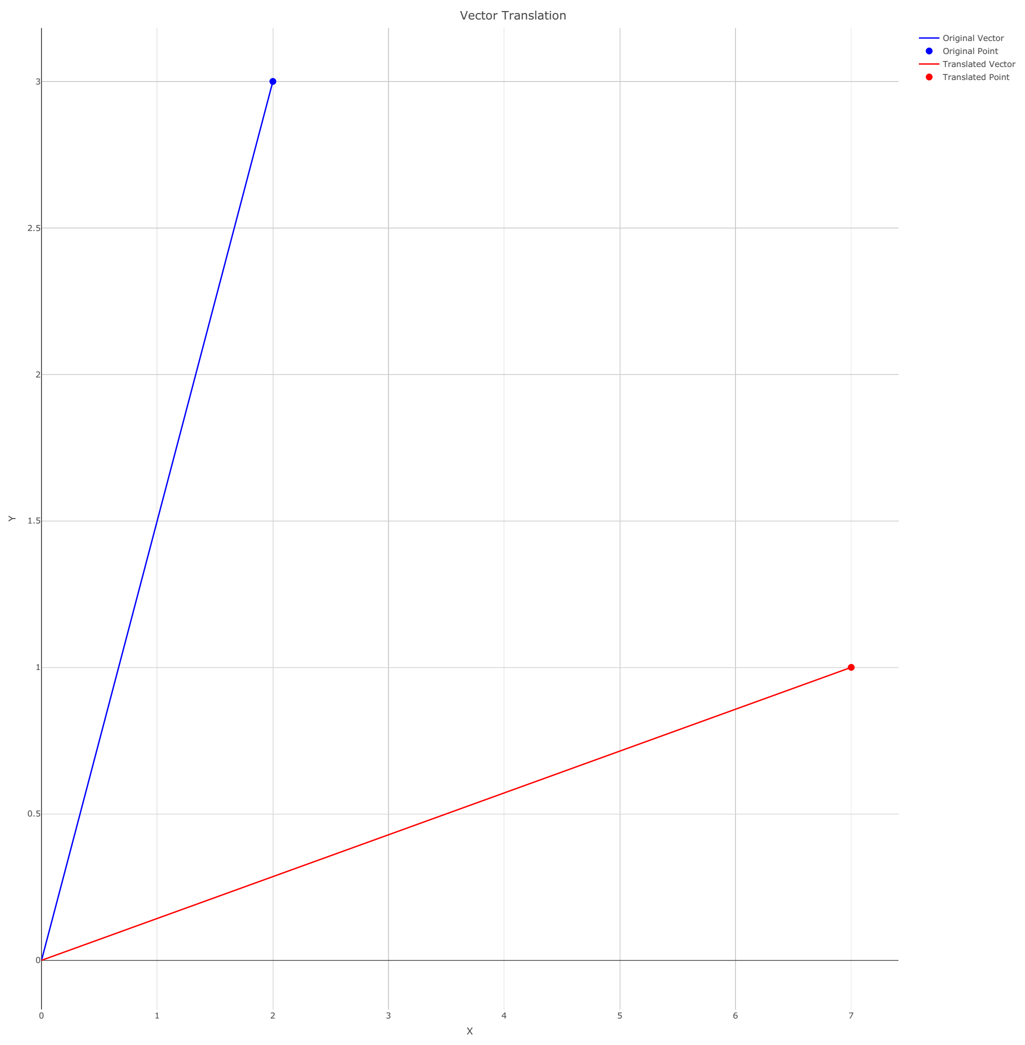
$$T(\mathbf{v}) = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix}$$

Here: - x' and y' are the new coordinates of the vector after translation. - t_x is the translation distance along the x-axis. - t_y is the translation distance along the y-axis.

For example, if we have a vector $(2, 3)$ and we want to translate it by $t_x = 5$ and $t_y = -2$, the new vector will be calculated as follows:

$$T(\mathbf{v}) = \begin{pmatrix} 2 + 5 \\ 3 - 2 \end{pmatrix} = \begin{pmatrix} 7 \\ 1 \end{pmatrix}$$

Thus, the vector $(2, 3)$ translated by $t_x = 5$ and $t_y = -2$ results in the new position $(7, 1)$.



5.2 3D Linear Transformation

Linear transformations in three dimensions can be represented using matrices and applied to vectors in 3D space. A linear transformation can include operations such as rotation, reflection, scaling, shearing, and translation.

A 3D linear transformation can be expressed in matrix form as:

$$\mathbf{v}' = \mathbf{A} \cdot \mathbf{v}$$

Where:

- $\mathbf{v}$ is the original vector represented as a column vector:

$$\mathbf{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- $\mathbf{v}'$ is the transformed vector.
- $\mathbf{A}$ is a 3×3 transformation matrix.

5.2.1 3D Rotation

Rotation in 3D space can be defined as the change in the position of a vector through a certain angle (θ) around one of the axes (X, Y, or Z).

The 3D rotation can be performed using rotation matrices, which are defined for rotation around the X, Y, and Z axes as follows:

1. **Rotation around the X-axis**:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

2. **Rotation around the Y-axis**:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

3. **Rotation around the Z-axis**:

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

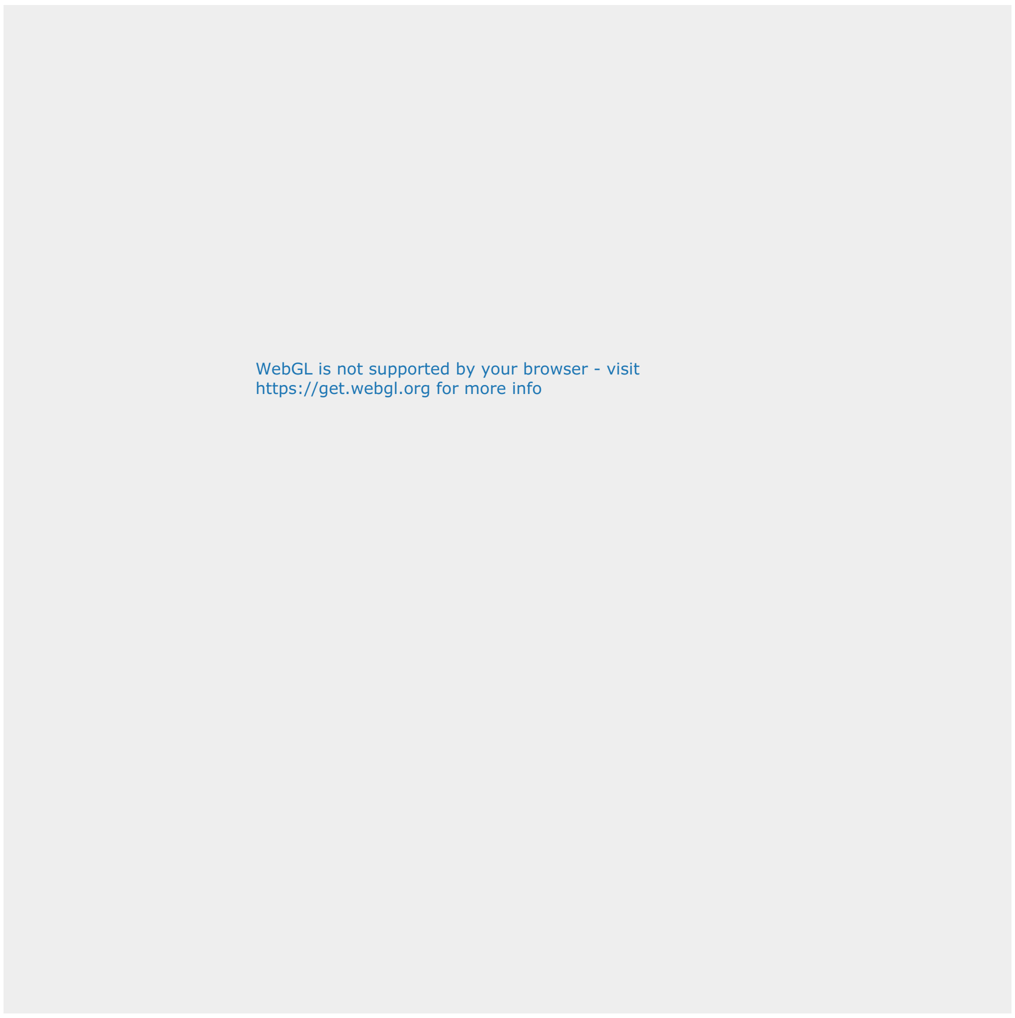
To rotate a vector $\mathbf{v}$ around a specific axis, we multiply the vector by the corresponding rotation matrix. Suppose the vector $\mathbf{v}$ is defined as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Then the resulting rotated vector $\mathbf{v}'$ can be obtained using:

$$\mathbf{v}' = R(\theta) \cdot \mathbf{v}$$

where $R(\theta)$ is the rotation matrix corresponding to the chosen axis. Let's consider the initial vector $(1, 2, 3)$ and perform a 3D rotation transformation. Observe the following visualization:


WebGL is not supported by your browser - visit
https://get.webgl.org for more info

5.2.2 3D Reflection

Reflection in 3D space can be defined as the change in the position of a vector by flipping its components along a specific axis or plane. For example, if we reflect across the XY plane, the Z component of the vector will have its sign changed.

The reflection matrices for each coordinate plane in 3D are as follows:

1. **Reflection across the XY plane**:

$$R_{XY} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

2. **Reflection across the YZ plane**:

$$R_{YZ} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. **Reflection across the XZ plane**:

$$R_{XZ} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

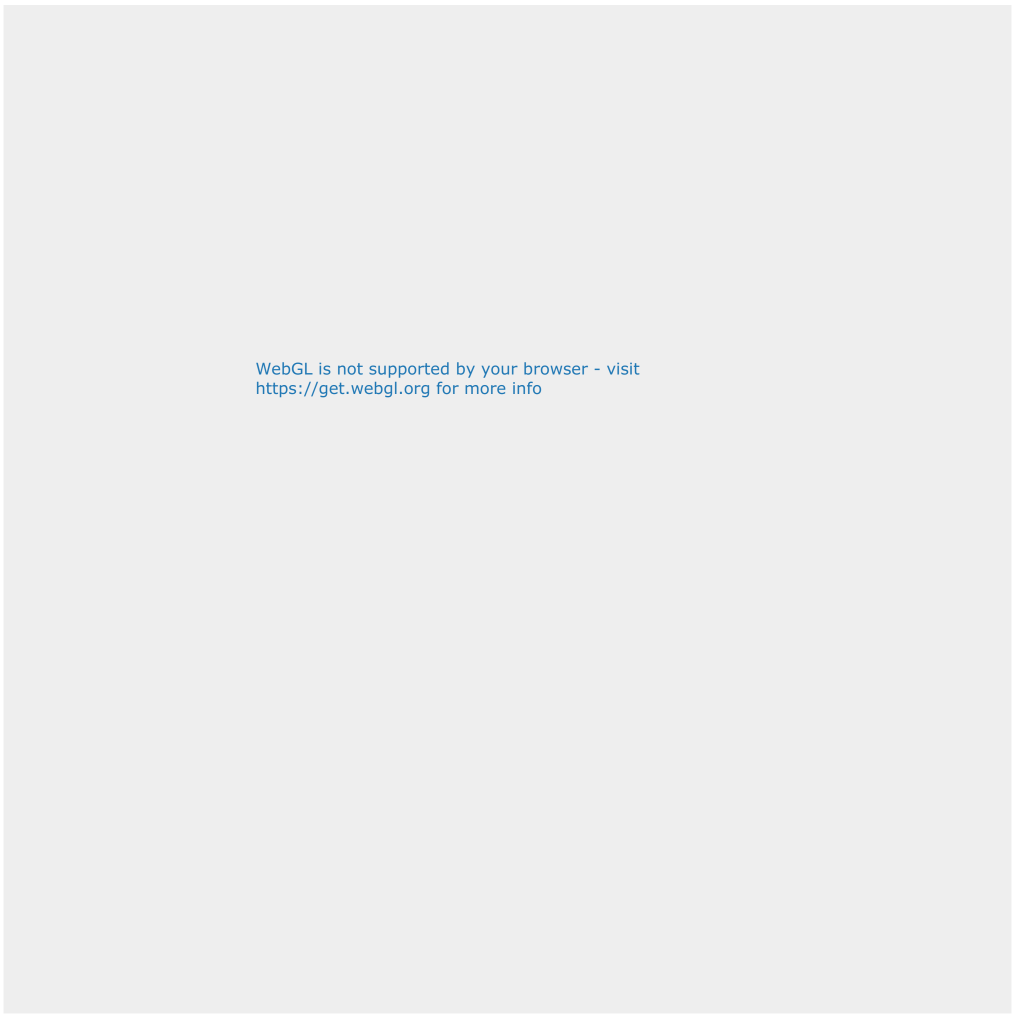
To reflect a vector **v** across a specific plane, we multiply the vector by the corresponding reflection matrix. Suppose the vector **v** is defined as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Then the resulting reflected vector $\mathbf{v}'$ can be obtained using:

$$\mathbf{v}' = R \cdot \mathbf{v}$$

where R is the reflection matrix corresponding to the selected plane. Let's consider the initial vector $(1, 2, 3)$ and perform a 3D reflection transformation. Observe the following visualization:


WebGL is not supported by your browser - visit https://get.webgl.org for more info

5.2.3 3D Scaling

3D scaling enables us to increase or decrease the dimensions of an object according to our needs. This transformation can be applied uniformly, meaning the scaling factor remains consistent across all axes, or non-uniformly, where the scaling factors differ for each axis.

The scaling matrix in 3D space can be expressed in the following matrix form:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Where: - s_x, s_y, and s_z are the scaling factors for the X, Y, and Z axes, respectively.

Suppose we have a vector $\mathbf{v}$ represented as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

After scaling, the new vector $\mathbf{v}'$ can be obtained through:

$$\mathbf{v}' = S \cdot \mathbf{v} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \\ s_z \cdot z \end{bmatrix}$$

Uniform Scaling

Definition: Uniform scaling refers to a transformation that enlarges or shrinks an object in 3D space using the same scaling factor for all three axes (X, Y, and Z). This maintains the object's proportions and shape, ensuring that the dimensions grow or shrink uniformly.

Mathematical Representation: The uniform scaling matrix can be represented as:

$$S = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix}$$

Where: -s is the uniform scaling factor.

Properties:

- **Proportionality**: Since the same scaling factor is applied to all axes, the object retains its original shape and proportions. For example, a sphere remains a sphere, and a cube remains a cube.
- **Center of Scaling**: The scaling occurs around a specified point, usually the origin (0, 0, 0), unless a different center is defined. Objects are enlarged or reduced in size relative to this center point.

Applications:

- **Modeling and Animation**: Uniform scaling is commonly used in computer graphics to resize characters or objects without distorting their shapes. For example, enlarging a character model for a game scene while maintaining their proportions.
- **Architectural Visualization**: In architectural design, uniform scaling can be applied to create models of buildings, allowing designers to maintain accurate proportions when resizing structures.
- **Simulation**: In simulations, uniform scaling helps maintain realistic interactions between objects, such as when simulating physics.

Example: If an object (e.g., a cube) with dimensions $(1, 2, 3)$ is uniformly scaled by a factor of 2, the new dimensions will be $(2, 4, 6)$. The shape remains a cube, just larger.

Non-Uniform Scaling

Definition: Non-uniform scaling is a transformation that changes the dimensions of an object by applying different scaling factors to each of the three axes (X, Y, and Z). This can result in distortion, changing the object's proportions and potentially altering its shape entirely.

Mathematical Representation: The non-uniform scaling matrix can be represented as:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Where:

- s_x is the scaling factor for the X-axis.
- s_y is the scaling factor for the Y-axis.
- s_z is the scaling factor for the Z-axis.

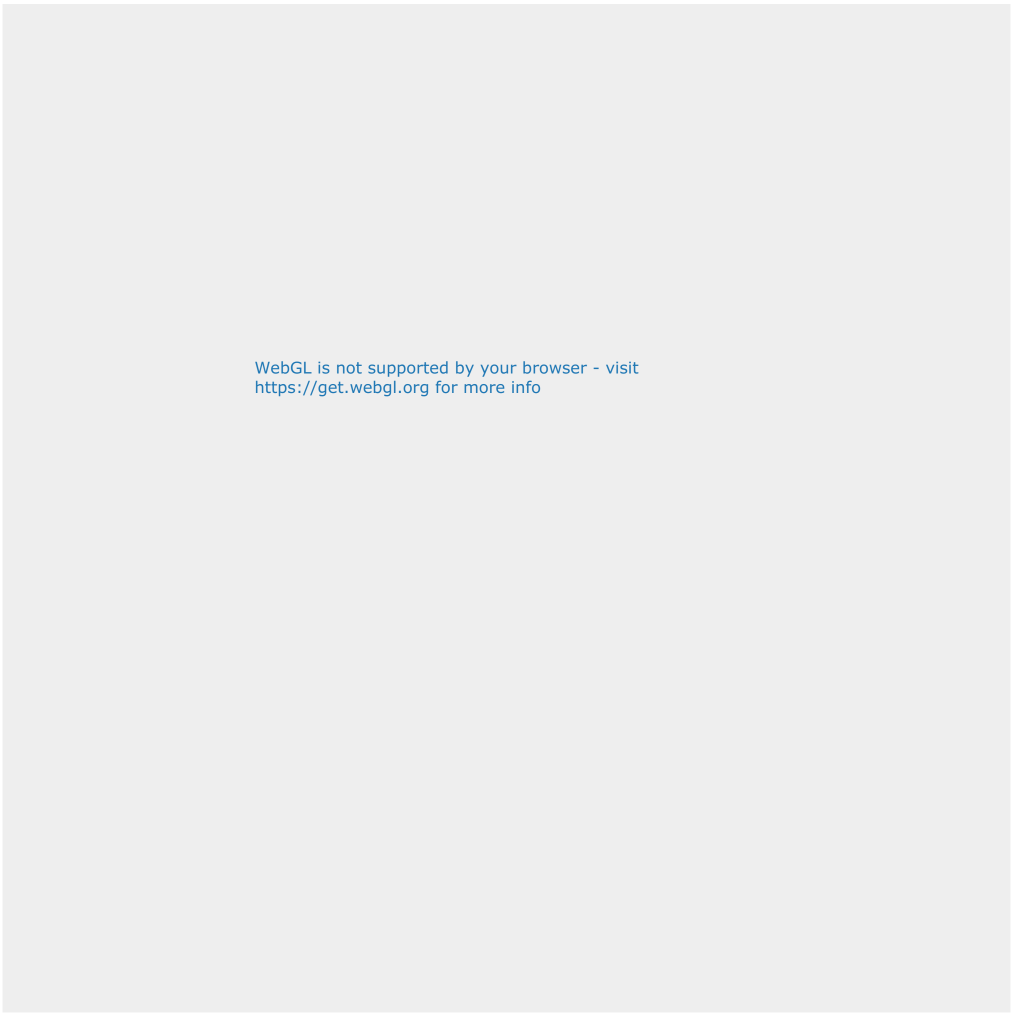
Properties:

- **Distortion**: Non-uniform scaling can stretch or compress an object differently along different axes, leading to a distortion of its original shape. For instance, a cube can become a rectangular prism or an elongated shape.
- **Independent Control**: Each axis can be scaled independently, providing greater flexibility in design and manipulation. This is useful for creating complex shapes and adjusting models to fit specific requirements.

Applications:

- **Character Modeling**: In character design, non-uniform scaling can create exaggerated features, such as a tall, thin character or a short, stocky character, by varying the scaling factors for each axis.
- **Animation**: Non-uniform scaling is often used in animations to create effects such as stretching or squashing, which can add realism or stylization to animated characters and objects.
- **Industrial Design**: In product design, non-uniform scaling can help designers adapt objects to fit functional requirements, such as making a part wider or thinner.

Example: Consider an object with dimensions $(1, 2, 3)$. If it undergoes non-uniform scaling with factors $s_x = 2$, $s_y = 1$, and $s_z = 0.5$, the new dimensions will be $(2, 2, 1.5)$. The object's shape will now be elongated along the X-axis, squashed along the Y-axis, and unchanged in the Z-axis.


WebGL is not supported by your browser - visit https://get.webgl.org for more info

5.2.4 3D Shearing

The shearing matrix in 3D space can be expressed in the following matrix form:

$$H = \begin{bmatrix} 1 & sh_{xy} & sh_{xz} \\ sh_{yx} & 1 & sh_{yz} \\ sh_{zx} & sh_{zy} & 1 \end{bmatrix}$$

Where:

- sh_{xy}, sh_{xz}, sh_{yx}, sh_{yz}, sh_{zx}, and sh_{zy} are the shearing factors for each axis combination.

Let's consider a vector **v** expressed as:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

After shearing, the new vector $\mathbf{v}'$ can be obtained through:

$$\mathbf{v}' = H \cdot \mathbf{v} = \begin{bmatrix} 1 & sh_{xy} & sh_{xz} \\ sh_{yx} & 1 & sh_{yz} \\ sh_{zx} & sh_{zy} & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + sh_{xy} \cdot y + sh_{xz} \cdot z \\ sh_{yx} \cdot x + y + sh_{yz} \cdot z \\ sh_{zx} \cdot x + sh_{zy} \cdot y + z \end{bmatrix}$$

Assuming we use the following shearing factors:

- $sh_{xy} = 0.5$
- $sh_{xz} = 0.2$
- $sh_{yx} = 0.1$
- $sh_{yz} = 0.3$
- $sh_{zx} = 0.4$
- $sh_{zy} = 0.1$

With the initial vector $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, we can calculate the new vector $\mathbf{v}'$ as follows:

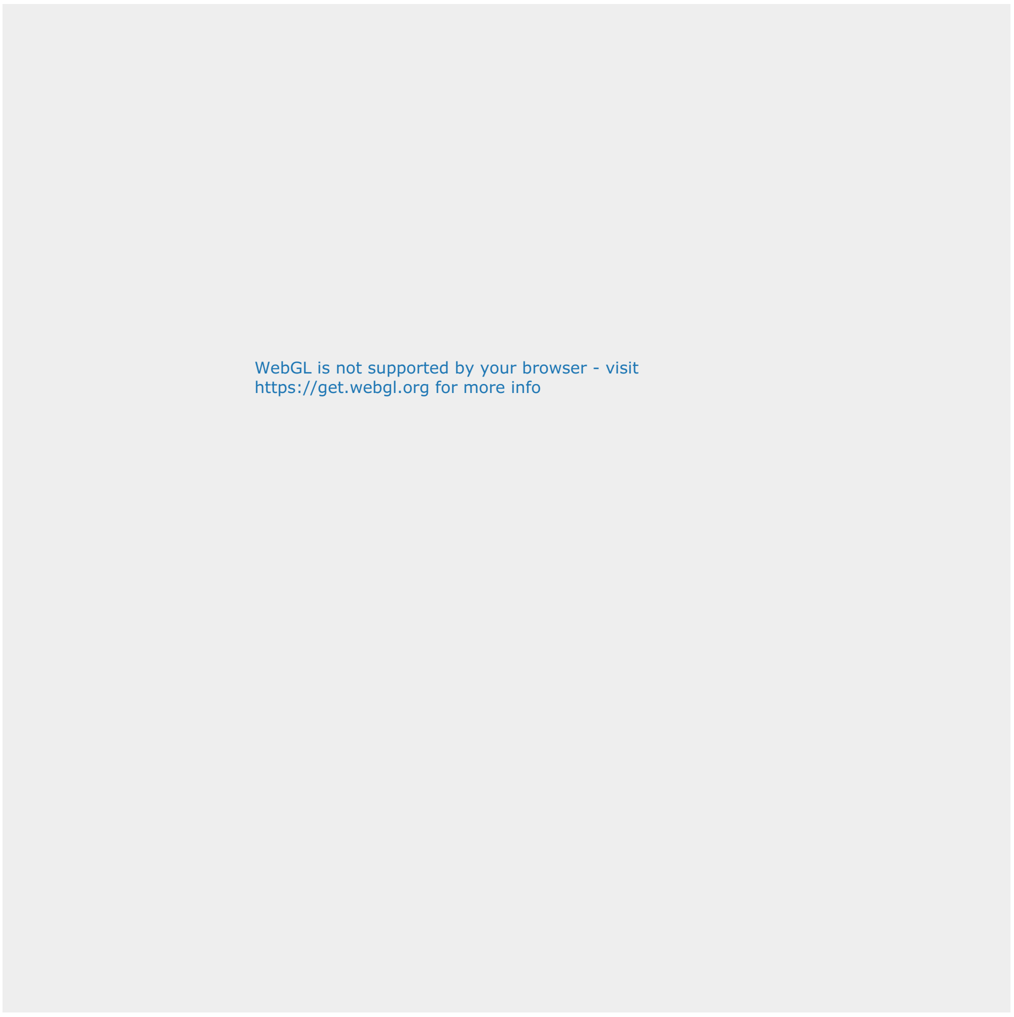
$$\mathbf{v}' = H \cdot \mathbf{v} = \begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.1 & 1 & 0.3 \\ 0.4 & 0.1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The result of the above calculation is:

$$\mathbf{v}' = \begin{bmatrix} 2.6 \\ 3.0 \\ 3.6 \end{bmatrix}$$

Thus, the result of the shearing transformation on the vector $(1, 2, 3)$ is $(2.6, 3.0, 3.6)$.

Here is a visualization example of the original vector and the result of the shearing.


WebGL is not supported by your browser - visit https://get.webgl.org for more info

5.2.5 3D Translation

In the context of translation transformation, we can define the transformation as follows:

$$\mathbf{v}' = T \cdot \mathbf{v}$$

where:

- $\mathbf{v}$ is the initial position vector,
- $\mathbf{v}'$ is the position vector after translation,
- T is the translation matrix.

The translation matrix in 3D space can be expressed in the following augmented matrix form (adding a row for homogeneous coordinates):

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where:

- t_x, t_y, and t_z are the translation distances along the X, Y, and Z axes, respectively.

Let's consider a vector $\mathbf{v}$ expressed as a homogeneous coordinate:

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

After translation, the new vector $\mathbf{v}'$ can be obtained through:

$$\mathbf{v}' = T \cdot \mathbf{v} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}$$

Assuming we use the following translation distances:

- $t_x = 3$
- $t_y = 2$
- $t_z = 1$

With the initial vector $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$, we can calculate the new vector $\mathbf{v}'$ as follows:

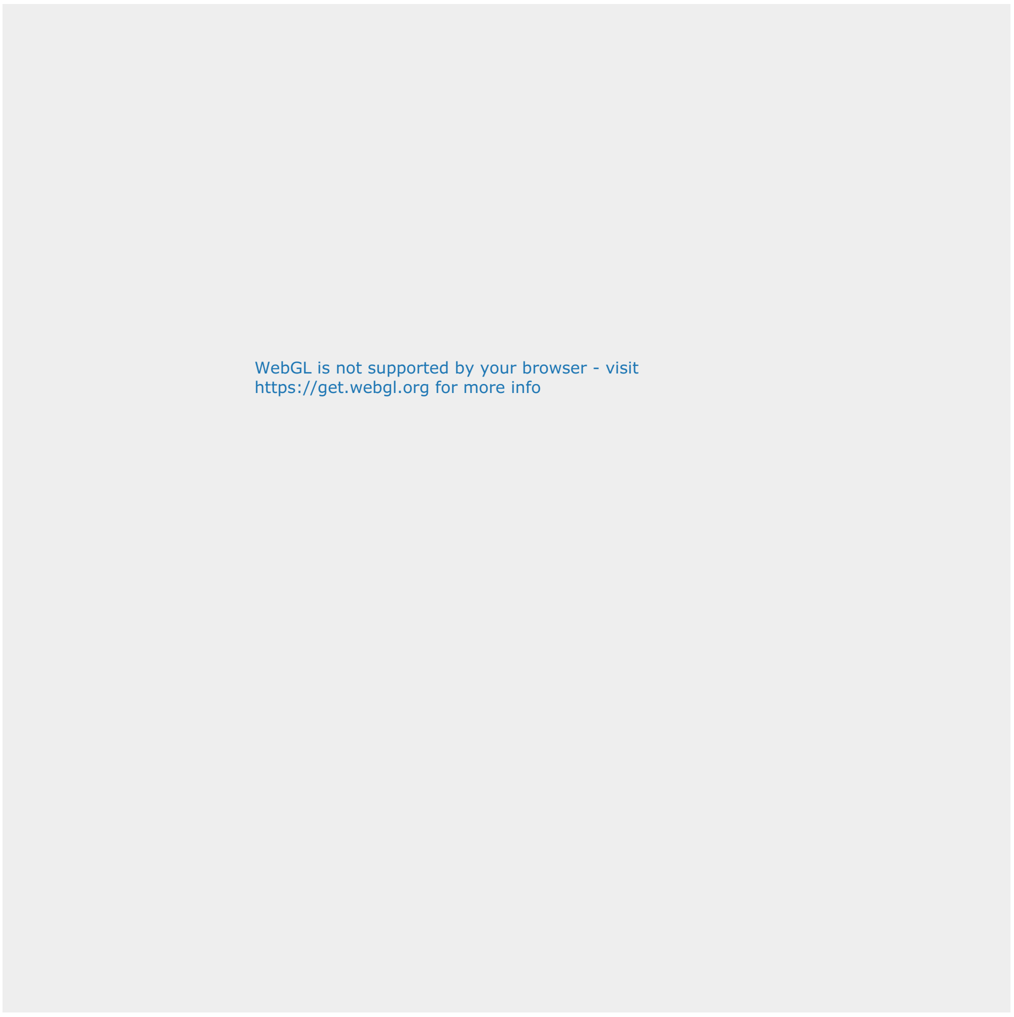
$$\mathbf{v}' = T \cdot \mathbf{v} = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

The result of the above calculation is:

$$\mathbf{v}' = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

Thus, the result of the translation transformation on the vector $(1, 2, 3)$ is $(4, 4, 4)$.

Here is a visualization example of the original vector and the result of the translation.


WebGL is not supported by your browser - visit
https://get.webgl.org for more info

5.3 Case Study 1

Applying Scaling Transformation on an Image for Poster Design

5.3.1 Background

A graphic designer is creating a promotional poster for an event. They want to use the company logo, but the size and proportions don't quite fit the poster layout. The designer uses **scaling transformations** to adjust the image size while preserving its original shape.

5.3.2 Objectives

1. Create a version of the logo with **uniform scaling** to adjust the size while maintaining the original proportions.

2. Create a version with **non-uniform scaling** to better fit specific dimensions on the poster.

5.3.3 Solution Steps

The designer used the following Python code to perform scaling transformations on the logo downloaded from a URL.

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import requests
from io import BytesIO

# Download and load the logo image
image_url = "https://github.com/dsciencelabs/images/blob/master/Logo_Dsciencelabs_v1.png?r
response = requests.get(image_url)
original_image = Image.open(BytesIO(response.content))

# Scaling transformation matrices
# Uniform scaling matrix (scaling factor = 1.5)
uniform_scale_factor = 1.5
uniform_scaling_matrix = np.array([
    [uniform_scale_factor, 0],
    [0, uniform_scale_factor]
])

# Non-uniform scaling matrix
non_uniform_scale_x = 1.2  # Scale width by 120%
non_uniform_scale_y = 0.8  # Scale height by 80%
non_uniform_scaling_matrix = np.array([
    [non_uniform_scale_x, 0],
    [0, non_uniform_scale_y]
])

# Function to apply scaling transformation
def apply_scaling(image, scaling_matrix):
    # Get the new dimensions
    width, height = image.size
    new_width = int(width * scaling_matrix[0, 0])
    new_height = int(height * scaling_matrix[1, 1])

    # Resize the image with the new dimensions
    return image.resize((new_width, new_height))

# Apply uniform and non-uniform scaling transformations
uniform_image = apply_scaling(original_image, uniform_scaling_matrix)
non_uniform_image = apply_scaling(original_image, non_uniform_scaling_matrix)

# Display the original and scaled images
```

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[0].imshow(original_image)
axes[0].set_title("Original Image")
axes[0].axis("off")

axes[1].imshow(uniform_image)
axes[1].set_title(f"Uniform Scaling ({uniform_scale_factor * 100:.0f}%)")
axes[1].axis("off")

axes[2].imshow(non_uniform_image)
axes[2].set_title(f"Non-uniform Scaling (Width {non_uniform_scale_x * 100:.0f}%, Height {non_
axes[2].axis("off")

plt.tight_layout()
plt.show()
```

5.4 Case Study 2

Counting Lights in a City Night Image

In this case study, we explore how to estimate the number of lights in a nighttime photograph of an urban area using image processing techniques. This method can help city planners assess energy consumption and plan for energy-saving initiatives.

5.4.1 Problem Context

A city planner needs to evaluate energy usage in a specific urban area by counting the number of lights visible in a photograph taken at night. This estimation can help in planning energy-saving measures. We use an image analysis approach that involves several steps, such as contrast enhancement and object detection, to identify the lights.

5.4.2 Steps of the Analysis

1. **Image Processing**: The original photograph is converted to grayscale to simplify light detection by reducing color complexity.
2. **Contrast Adjustment**: A linear transformation is applied to enhance the contrast, making bright areas (lights) more distinguishable from dark areas.
3. **Thresholding**: Using Otsu's method, the image is binarized, converting bright spots to white and dark areas to black.
4. **Object Counting**: Connected bright regions are identified and counted as individual lights.

5.4.3 Code Implementation

Below is the Python code used to perform these steps:

```
from PIL import Image
import requests
from io import BytesIO
```

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure, color, filters

# Load the image
image_url = "https://github.com/dsciencelabs/images/blob/master/jogja.jpg?raw=true"
response = requests.get(image_url)
image = Image.open(BytesIO(response.content))

# Convert to grayscale
gray_image = image.convert("L")

# Convert image to numpy array
image_matrix = np.array(gray_image)

# Apply linear transformation to increase contrast
a, b = 2, -100  # Adjust as needed
transformed_image = np.clip(a * image_matrix + b, 0, 255).astype(np.uint8)

# Apply thresholding
threshold_value = filters.threshold_otsu(transformed_image)
binary_image = transformed_image > threshold_value

# Count bright objects using connected components
labels = measure.label(binary_image)
lamp_count = labels.max()

# Display results
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(gray_image, cmap="gray")
axes[0].set_title("Original Grayscale Image")
axes[0].axis("off")

axes[1].imshow(transformed_image, cmap="gray")
axes[1].set_title("Transformed Image")
axes[1].axis("off")

axes[2].imshow(color.label2rgb(labels, bg_label=0))
axes[2].set_title(f"Detected Lights (Count: {lamp_count})")
axes[2].axis("off")

plt.tight_layout()
plt.show()
```

5.5 Case Study 3

Seasonal Production Demand Adjustment

5.5.1 Background

A manufacturing company operates three production facilities (A, B, and C) that produce the same product. Each facility has different initial production capacities, and seasonal demand variations require adjustments in their production levels. The company aims to optimize production by applying linear transformations (scaling, rotation, and translation) to meet the increased seasonal demand efficiently.

5.5.2 Given Data

The initial production capacities (in tons per day) and seasonal scaling factors for each facility are as follows:

Facility	Initial Production (tons/day)	Seasonal Scaling Factor
Facility A	100	1.2
Facility B	80	1.5
Facility C	60	1.3

Additionally, to shift production focus, Facility B will undergo a rotation transformation of 30 degrees around the z-axis. There is also a baseline increase in demand across all facilities represented by a translation vector that adds 10 tons/day to each facility's production.

5.5.3 Problem Statement

1. **Calculate the new production capacities after applying the scaling transformation based on the given seasonal scaling factors.**
2. **Determine the new production capacities after applying the rotation transformation around the z-axis for the adjusted production levels of Facility B.**
3. **What will be the final production capacities for each facility after applying the translation vector that accounts for the increased demand?**
4. **If the production capacity of Facility C is limited to a maximum of 80 tons/day, how would this limitation affect the overall production and demand fulfillment for the company?**
5. **Discuss how applying these linear transformations can help the company respond to seasonal demand changes effectively.**

5.5.4 Solution

1. Scaling Transformation

We calculate the new production capacities by applying the scaling factors:

- Facility A: ($100 \times 1.2 = 120$) tons/day
- Facility B: ($80 \times 1.5 = 120$) tons/day
- Facility C: ($60 \times 1.3 = 78$) tons/day

The new production capacities after scaling are:

Facility	New Production (tons/day)
Facility A	120
Facility B	120
Facility C	78

2. Rotation Transformation

To apply the rotation transformation for Facility B, we need to perform the rotation around the z-axis by 30 degrees. In this case, since we are focusing on production capacity (not coordinates), we can assume that rotation will not affect the capacity directly, but it will affect the distribution of production. Thus, the production remains:

- Facility B: 120 tons/day (no capacity change due to rotation)

3. Final Production Capacities

After applying a translation vector that adds 10 tons/day to each facility, we calculate:

- Facility A: (120 + 10 = 130) tons/day
- Facility B: (120 + 10 = 130) tons/day
- Facility C: (78 + 10 = 88) tons/day

The final production capacities are:

Facility	Final Production (tons/day)
Facility A	130
Facility B	130
Facility C	88

4. Capacity Limitation for Facility C

If Facility C's production capacity is limited to 80 tons/day, we must adjust its final capacity:

- Facility A: 130 tons/day
- Facility B: 130 tons/day
- Facility C: 80 tons/day (limited)

Thus, the new final capacities are:

Facility	Adjusted Final Production (tons/day)
Facility A	130
Facility B	130
Facility C	80

5. Discussion on Linear Transformations

Applying linear transformations such as scaling, rotation, and translation allows the company to effectively adjust their production capacities in response to seasonal demand changes. Scaling helps in increasing production levels, while rotation allows for adjusting the focus of production without changing the overall capacity. Translation reflects an increase in demand, ensuring that the facilities can adapt to meet market requirements efficiently. Such methods are crucial for optimizing resource utilization and enhancing operational efficiency in a dynamic production environment.

5.6 Linear Transformations in Python

Klik here

Chapter 6

Eigenvalues and Eigenvectors

In data science, understanding **eigenvalues** and **eigenvectors** is essential for various techniques, especially for **dimensionality reduction** and **data transformation**. These concepts are central to methods such as **Principal Component Analysis (PCA)**, which is widely used to analyze and visualize high-dimensional data.

In simple terms, eigenvalues and eigenvectors describe how a matrix (which represents a transformation) affects the data. Eigenvectors represent directions in the data space, while eigenvalues determine how much the data is scaled along those directions.

6.1 Eigenvalue

An **eigenvalue** is a scalar that indicates how much the data is stretched or compressed along a specific direction (represented by an eigenvector) when a transformation is applied. In linear algebra, if A is a square matrix and $\vec{x}$ is a non-zero vector, the eigenvalue λ of matrix A is defined by the equation:

$$A \cdot \vec{x} = \lambda \cdot \vec{x}$$

This equation tells us that when matrix A is applied to vector $\vec{x}$, the resulting vector is scaled by the factor λ along the same direction as $\vec{x}$.

To compute the eigenvalues, we solve the **characteristic equation**:

$$\det(A - \lambda I) = 0$$

Where: - A is the matrix (for example, the covariance matrix in PCA). - λ is the eigenvalue. - I is the identity matrix of the same size as A. - det represents the determinant of the matrix.

The determinant of $(A - \lambda I)$ will be a polynomial in λ, and the solutions to this polynomial are the eigenvalues. The eigenvalue tells us how much the vector is stretched or compressed. For example:

- If $\lambda > 1$, the vector is stretched.

- If $0 < \lambda < 1$, the vector is compressed.
- If $\lambda = 0$, the vector is collapsed to the origin.

6.2 Eigenvector

An **eigenvector** is a non-zero vector associated with a given eigenvalue. The eigenvector $\vec{x}$ corresponds to an eigenvalue λ and satisfies the equation:

$$(A - \lambda I) \cdot \vec{x} = 0$$

This equation tells us that when matrix A is applied to eigenvector $\vec{x}$, the result is simply a scalar multiple of $\vec{x}$, scaled by the eigenvalue λ. In other words, the direction of the eigenvector remains unchanged, although its magnitude is scaled by λ.

6.3 Eigenvalues & Eigenvectors 2D

This document demonstrates the calculation of eigenvalues and eigenvectors for a 2D matrix. The matrix we will use is:

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

6.3.1 Step 1: Finding Eigenvalues

The eigenvalues are solutions to the characteristic equation:

$$\det(A - \lambda I) = 0$$

Substituting A:

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

The determinant is:

$$\det(A - \lambda I) = (3-\lambda)(2-\lambda)$$

Setting this equal to zero:

$$(3-\lambda)(2-\lambda) = 0$$

Thus, the eigenvalues are:

$$\lambda_1 = 3, \quad \lambda_2 = 2$$

6.3.2 Step 2: Finding Eigenvectors

For each eigenvalue, solve $(A - \lambda I)v = 0$, where $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$.

For $\lambda_1 = 3$

Substituting $\lambda_1 = 3$:

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

Solving $(A - 3I)v = 0$:

$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This gives $v_2 = 0$, so an eigenvector for $\lambda_1 = 3$ is:

$$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For $\lambda_2 = 2$

Substituting $\lambda_2 = 2$:

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

Solving $(A - 2I)v = 0$:

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This gives $v_1 = -v_2$, so an eigenvector for $\lambda_2 = 2$ is:

$$v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

6.3.3 Calculation using Python

This Python code demonstrates how to manually compute eigenvalues and eigenvectors of a 2x2 matrix. The eigenvalues and eigenvectors can be used to understand the matrix's transformation properties, such as scaling and rotation. The final output will give you both the eigenvalues and eigenvectors that describe how the matrix A acts on vectors in its vector space.

```
import numpy as np

# Define the matrix A
A = np.array([[3, 1],
              [0, 2]])

# Step 1: Manually compute the characteristic equation
# det(A -  I) = (3 -  )(2 -  )
eigenvalues_manual = [3, 2]  # Roots of the characteristic equation

# Step 2: Manually compute eigenvectors for each eigenvalue
# For  1 = 3
lambda1 = eigenvalues_manual[0]
A_minus_lambda1I = A - lambda1 * np.eye(2)
print("Matrix (A -  1 * I):\n", A_minus_lambda1I)
```

```
Matrix (A -  1 * I):
 [[ 0.  1.]
 [ 0. -1.]]
```

```
# Solve (A -  1 * I) * v = 0
v1 = np.array([1, 0])  # Chosen based on row reduction (free variable)

# For  2 = 2
lambda2 = eigenvalues_manual[1]
A_minus_lambda2I = A - lambda2 * np.eye(2)
print("Matrix (A -  2 * I):\n", A_minus_lambda2I)
```

```
Matrix (A -  2 * I):
 [[1. 1.]
 [0. 0.]]
```

```
# Solve (A -  2 * I) * v = 0
v2 = np.array([-1, 1])  # Chosen based on row reduction (free variable)

# Combine eigenvectors into a matrix
eigenvectors_manual = np.column_stack((v1, v2))
print("Manual Eigenvalues:\n", eigenvalues_manual)
```

```
Manual Eigenvalues:
 [3, 2]
```

```
print("Manual Eigenvectors:\n", eigenvectors_manual)
```

```
Manual Eigenvectors:
 [[ 1 -1]
 [ 0  1]]
```

6.3.4 Visualization using Python

The visualization clearly shows how the matrix A transforms the eigenvectors. The blue lines represent the original eigenvectors, and the red dashed lines represent the transformed eigenvectors. This helps in understanding the effect of matrix A on these vectors and provides an intuitive grasp of the transformation process.

```python
import os
import sys

def install_package(package):
    try:
        __import__(package)
        print(f"'{package}' is already installed.")
    except ImportError:
        print(f"'{package}' is not installed. Installing now...")
        os.system(f"{sys.executable} -m pip install {package}")
        print(f"'{package}' has been successfully installed.")

# Example: Install 'plotly' if it is not already installed
install_package('plotly')
```

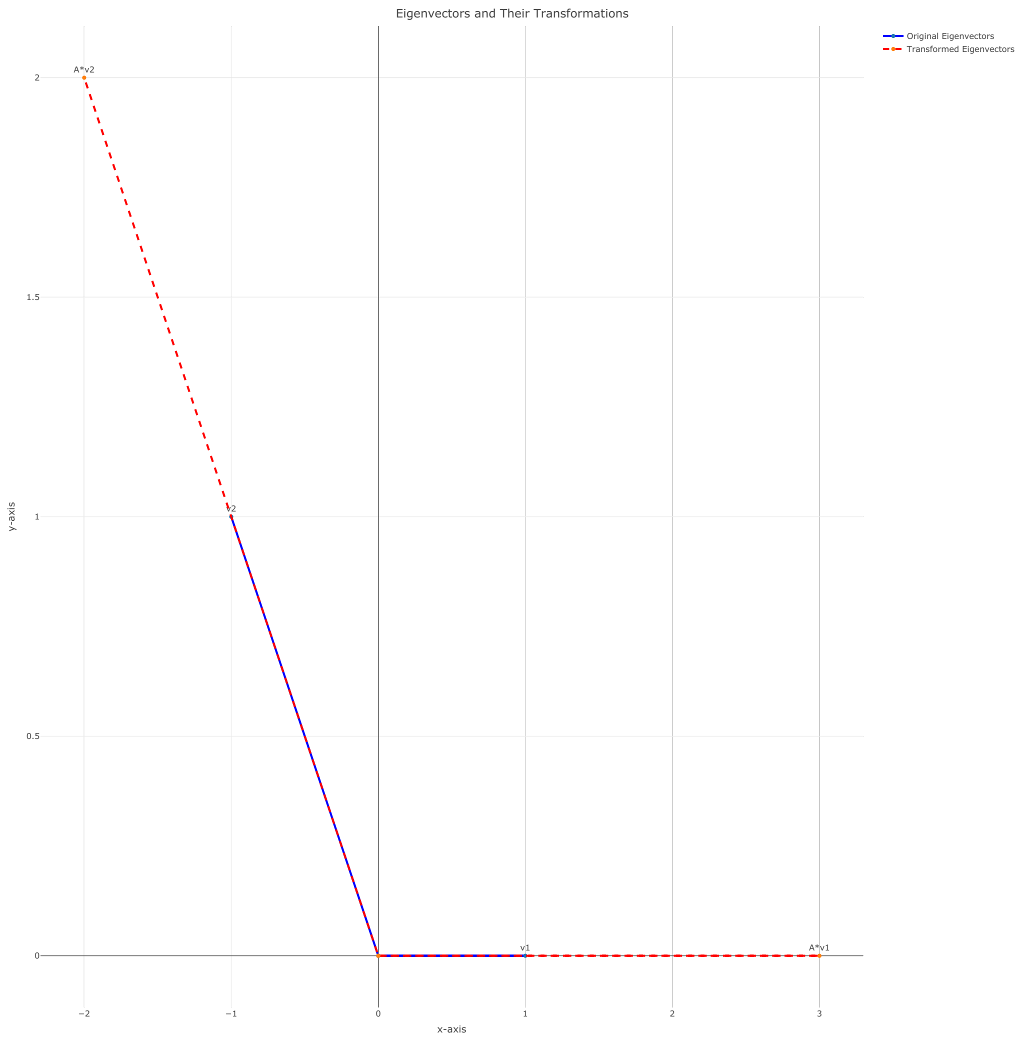
```python
import plotly.graph_objects as go

# Step 3: Transform eigenvectors using A
v1_transformed = np.dot(A, v1)
v2_transformed = np.dot(A, v2)

# Step 4: Visualization with Plotly
fig = go.Figure()

# Add original eigenvectors
fig.add_trace(go.Scatter(
    x=[0, v1[0]], y=[0, v1[1]],
    mode='lines+markers+text',
    text=["", "v1"],
    textposition="top center",
    name="Original v1",
    line=dict(color='blue', width=3)
))
fig.add_trace(go.Scatter(
    x=[0, v2[0]], y=[0, v2[1]],
    mode='lines+markers+text',
    text=["", "v2"],
    textposition="top center",
    name="Original v2",
    line=dict(color='blue', width=3)
))

# Add transformed eigenvectors
fig.add_trace(go.Scatter(
    x=[0, v1_transformed[0]], y=[0, v1_transformed[1]],
    mode='lines+markers+text',
    text=["", "A*v1"],
    textposition="top center",
    name="Transformed v1",
    line=dict(color='red', width=3, dash='dash')
))
fig.add_trace(go.Scatter(
    x=[0, v2_transformed[0]], y=[0, v2_transformed[1]],
    mode='lines+markers+text',
    text=["", "A*v2"],
    textposition="top center",
    name="Transformed v2",
    line=dict(color='red', width=3, dash='dash')
))

# Layout settings
fig.update_layout(
    title="Eigenvectors and Their Transformations",
    xaxis=dict(title="x-axis", zeroline=True),
    yaxis=dict(title="y-axis", zeroline=True),
    showlegend=True
)

# Show plot
fig.show()
```

```
Matrix (A -  1 * I):
```

```
     [,1] [,2]
[1,]    0    1
[2,]    0   -1
```

```
Matrix (A -  2 * I):
```

```
     [,1] [,2]
[1,]    1    1
[2,]    0    0
```



6.4 Eigenvalues & Eigenvectors 2D

This document demonstrates the calculation of eigenvalues and eigenvectors for a 3D matrix. The matrix we will use is:

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

6.4.1 Step 1: Finding Eigenvalues

The eigenvalues are solutions to the characteristic equation:

$$\det(A - \lambda I) = 0$$

Substitute A into the equation:

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

Now, compute the determinant:

$$\det(A - \lambda I) = (3-\lambda)\left[\det\begin{bmatrix} 2-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix}\right]$$

The determinant of the 2×2 matrix is:

$$\det\begin{bmatrix} 2-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix} = (2-\lambda)(1-\lambda)$$

Thus, the characteristic polynomial is:

$$(3-\lambda)(2-\lambda)(1-\lambda) = 0$$

This gives us the eigenvalues:

$$\lambda_1 = 3, \quad \lambda_2 = 2, \quad \lambda_3 = 1$$

6.4.2 Step 2: Finding Eigenvectors

For each eigenvalue, we need to solve $(A - \lambda I)v = 0$ for the corresponding eigenvector $v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$.

6.4.2.1 For $\lambda_1 = 3$:

Substitute $\lambda_1 = 3$ into $A - 3I$:

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

Solve $(A - 3I)v = 0$:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This gives the system of equations:

1. $v_2 = 0$

2. $-v_2 + v_3 = 0 \Rightarrow v_3 = 0$
3. v_1 can be any value.

Thus, an eigenvector for $\lambda_1 = 3$ is:

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

6.4.2.2 For $\lambda_2 = 2$:

Substitute $\lambda_2 = 2$ into $A - 2I$:

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

Solve $(A - 2I)v = 0$:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This gives the system of equations:

1. $v_1 + v_2 = 0 \Rightarrow v_1 = -v_2$
2. $v_3 = 0$

Thus, an eigenvector for $\lambda_2 = 2$ is:

$$v_2 = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

6.4.2.3 For $\lambda_3 = 1$:

Substitute $\lambda_3 = 1$ into $A - 1I$:

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

Solve $(A - 1I)v = 0$:

$$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This gives the system of equations:

1. $2v_1 + v_2 = 0 \Rightarrow v_2 = -2v_1$

2. $v_2 + v_3 = 0 \Rightarrow v_3 = -v_2 = 2v_1$

Thus, an eigenvector for $\lambda_3 = 1$ is:

$$v_3 = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix}$$

6.4.3 Summary

The eigenvalues and eigenvectors for the matrix $A = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$ are:

- $\lambda_1 = 3$, eigenvector: $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
- $\lambda_2 = 2$, eigenvector: $\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$
- $\lambda_3 = 1$, eigenvector: $\begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix}$

These eigenvectors correspond to the directions in 3D space along which the matrix A acts by stretching or squishing the space.

6.4.4 Calculation using Python

This Python code demonstrates how to manually compute eigenvalues and eigenvectors of a 3x3 matrix. The eigenvalues and eigenvectors can be used to understand the matrix's transformation properties, such as scaling and rotation. The final output will give you both the eigenvalues and eigenvectors that describe how the matrix A acts on vectors in its vector space.

```python
import numpy as np

# Define the matrix A (3x3 matrix)
A = np.array([[3, 1, 0],
              [0, 2, 1],
              [0, 0, 1]])

# Step 1: Manually compute the characteristic equation
# The characteristic equation for a 3x3 matrix det(A -  I) = 0.
# We will find the eigenvalues by solving the determinant equation manually.
eigenvalues_manual = [3, 2, 1]  # The eigenvalues can be found by solving the determinant

# Step 2: Manually compute eigenvectors for each eigenvalue
# For  1 = 3
lambda1 = eigenvalues_manual[0]
```

```python
A_minus_lambda1I = A - lambda1 * np.eye(3)
print("Matrix (A -  1 * I):\n", A_minus_lambda1I)
```

```
Matrix (A -  1 * I):
 [[ 0.  1.  0.]
 [ 0. -1.  1.]
 [ 0.  0. -2.]]
```

```python
# Solve (A -  1 * I) * v = 0 by inspection or Gaussian elimination
v1 = np.array([1, 0, 0])  # Eigenvector corresponding to  1 = 3 (from row reduction)

# For  2 = 2
lambda2 = eigenvalues_manual[1]
A_minus_lambda2I = A - lambda2 * np.eye(3)
print("Matrix (A -  2 * I):\n", A_minus_lambda2I)
```

```
Matrix (A -  2 * I):
 [[ 1.  1.  0.]
 [ 0.  0.  1.]
 [ 0.  0. -1.]]
```

```python
# Solve (A -  2 * I) * v = 0 by inspection or Gaussian elimination
v2 = np.array([-1, 1, 0])  # Eigenvector corresponding to  2 = 2 (from row reduction)

# For  3 = 1
lambda3 = eigenvalues_manual[2]
A_minus_lambda3I = A - lambda3 * np.eye(3)
print("Matrix (A -  3 * I):\n", A_minus_lambda3I)
```

```
Matrix (A -  3 * I):
 [[2. 1. 0.]
 [0. 1. 1.]
 [0. 0. 0.]]
```

```python
# Solve (A -  3 * I) * v = 0 by inspection or Gaussian elimination
v3 = np.array([1, -2, 2])  # Eigenvector corresponding to  3 = 1 (from row reduction)

# Combine eigenvectors into a matrix
eigenvectors_manual = np.column_stack((v1, v2, v3))
print("Manual Eigenvalues:\n", eigenvalues_manual)
```

```
Manual Eigenvalues:
 [3, 2, 1]
```

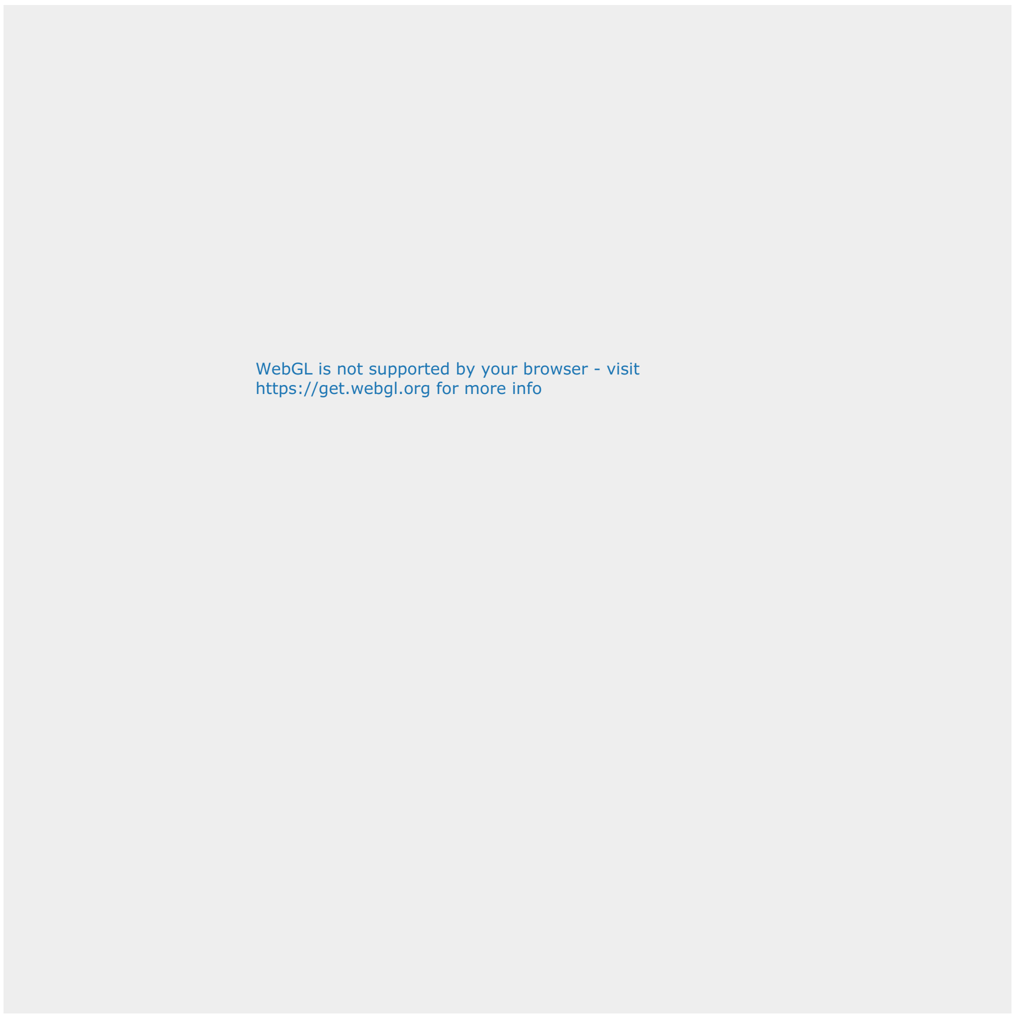
```python
print("Manual Eigenvectors:\n", eigenvectors_manual)
```

```
Manual Eigenvectors:
 [[ 1 -1  1]
 [ 0  1 -2]
 [ 0  0  2]]
```

6.4.5 Visualization using Python

The visualization clearly shows how the matrix A transforms the eigenvectors. The blue lines represent the original eigenvectors, and the red dashed lines represent the transformed eigenvectors. This helps in understanding the effect of matrix A on these vectors and provides an intuitive grasp of the transformation process.

```python
import plotly.graph_objects as go

# Transformed eigenvectors
v1_transformed = np.dot(A, v1)
v2_transformed = np.dot(A, v2)
v3_transformed = np.dot(A, v3)

# Step 3: Create the plot
fig = go.Figure()

# Add original eigenvectors
fig.add_trace(go.Scatter3d(
    x=[0, v1[0]], y=[0, v1[1]], z=[0, v1[2]],
    mode='lines+markers+text',
    text=["", "v1"],
    textposition="top center",
    name="Original v1",
    line=dict(color='blue', width=3)
))
fig.add_trace(go.Scatter3d(
    x=[0, v2[0]], y=[0, v2[1]], z=[0, v2[2]],
    mode='lines+markers+text',
    text=["", "v2"],
    textposition="top center",
    name="Original v2",
    line=dict(color='blue', width=3)
))
fig.add_trace(go.Scatter3d(
    x=[0, v3[0]], y=[0, v3[1]], z=[0, v3[2]],
    mode='lines+markers+text',
    text=["", "v3"],
    textposition="top center",
    name="Original v3",
    line=dict(color='blue', width=3)
))

# Add transformed eigenvectors
fig.add_trace(go.Scatter3d(
    x=[0, v1_transformed[0]], y=[0, v1_transformed[1]], z=[0, v1_transformed[2]],
    mode='lines+markers+text',
    text=["", "A*v1"],
    textposition="top center",
    name="Transformed v1",
    line=dict(color='red', width=3, dash='dash')
))
fig.add_trace(go.Scatter3d(
    x=[0, v2_transformed[0]], y=[0, v2_transformed[1]], z=[0, v2_transformed[2]],
    mode='lines+markers+text',
    text=["", "A*v2"],
    textposition="top center",
    name="Transformed v2",
    line=dict(color='red', width=3, dash='dash')
))
fig.add_trace(go.Scatter3d(
    x=[0, v3_transformed[0]], y=[0, v3_transformed[1]], z=[0, v3_transformed[2]],
    mode='lines+markers+text',
    text=["", "A*v3"],
    textposition="top center",
    name="Transformed v3",
    line=dict(color='red', width=3, dash='dash')
))

# Layout settings
fig.update_layout(
    title="Eigenvectors and Their Transformations in 3D",
    scene=dict(
        xaxis=dict(title="x-axis", zeroline=True),
        yaxis=dict(title="y-axis", zeroline=True),
        zaxis=dict(title="z-axis", zeroline=True)
    ),
    showlegend=True
)

# Show plot
fig.show()
```


WebGL is not supported by your browser - visit https://get.webgl.org for more info

6.5 Case Study

In data science and machine learning, one common application of eigenvalues and eigenvectors is **Principal Component Analysis (PCA)**. PCA is a dimensionality reduction technique used to simplify complex datasets by transforming the data into a new set of variables, called **principal components**, which are linear combinations of the original variables.

In this case study, we will use the **Iris dataset**, which contains measurements of **sepal length**, **sepal width**, **petal length**, and **petal width** for different species of iris flowers. PCA will help reduce the number of dimensions while retaining the most important information about the data. You can get the csv version of this dataset from here.

It has 4 features, Sepal Length, Sepal Width, Petal Length, Petal Width all given in centimeters. In total it has 150 rows of data comprising of 3 species with 50 row for each species. Then a column with its species is also given.

6.5.1 Problem Statement

The goal is to apply PCA to the **Iris dataset**, which consists of 150 samples with 4 features each. We will:

1. Apply PCA to reduce the dimensionality of the dataset from 4 dimensions to 2 dimensions.
2. Calculate the eigenvalues and eigenvectors of the covariance matrix of the dataset.
3. Use the eigenvalues to determine how much variance is explained by each principal component.
4. Visualize the data and the transformed principal components.

6.5.2 Dataset

The **Iris dataset** includes the following columns:

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
...	...	...	...	...
6.7	3.0	5.2	2.3	Virginica
6.3	2.5	5.0	1.9	Virginica
6.5	3.0	5.5	2.1	Virginica

6.5.3 Step 1: Data Preparation

First, we will load the Iris dataset and normalize it so that each feature has a mean of 0 and a standard deviation of 1. This normalization step is necessary to ensure that all features contribute equally to the PCA process.

```r
# Load necessary libraries
library(tidyverse)
library(caret)
library(ggplot2)
library(knitr)

# Load the Iris dataset
data(iris)
```

```r
# Step 1: Normalize the dataset (excluding the Species column)
iris_data <- iris[, 1:4]
iris_data_scaled <- scale(iris_data)

# Check the first few rows of the scaled data
kable(head(iris_data_scaled))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
-0.8976739	1.0156020	-1.335752	-1.311052
-1.1392005	-0.1315388	-1.335752	-1.311052
-1.3807271	0.3273175	-1.392399	-1.311052
-1.5014904	0.0978893	-1.279104	-1.311052
-1.0184372	1.2450302	-1.335752	-1.311052
-0.5353840	1.9333146	-1.165809	-1.048667

6.5.4 Step 2: Compute the Covariance Matrix

Next, we will compute the covariance matrix of the normalized dataset. The covariance matrix helps us understand the relationships between the features.

```r
# Step 2: Compute the covariance matrix
cov_matrix <- cov(iris_data_scaled)

# Print the covariance matrix
cov_matrix
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

6.5.5 Step 3: Calculate Eigenvalues and Eigenvectors

We will calculate the eigenvalues and eigenvectors of the covariance matrix. The eigenvalues represent the amount of variance explained by each principal component, and the eigenvectors represent the directions of maximum variance.

```r
# Step 3: Calculate the eigenvalues and eigenvectors
eigen_decomp <- eigen(cov_matrix)

# Eigenvalues
eigenvalues <- eigen_decomp$values
print("Eigenvalues:")
```

```
[1] "Eigenvalues:"
```

```r
eigenvalues
```

```
[1] 2.91849782 0.91403047 0.14675688 0.02071484
```

```
# Eigenvectors
eigenvectors <- eigen_decomp$vectors
print("Eigenvectors:")
```

```
[1] "Eigenvectors:"
```

```
eigenvectors
```

```
           [,1]        [,2]       [,3]       [,4]
[1,]  0.5210659 -0.37741762  0.7195664  0.2612863
[2,] -0.2693474 -0.92329566 -0.2443818 -0.1235096
[3,]  0.5804131 -0.02449161 -0.1421264 -0.8014492
[4,]  0.5648565 -0.06694199 -0.6342727  0.5235971
```

6.5.6 Step 4: Transform the Data

Using the eigenvectors, we will transform the original data into a new space defined by the principal components.

```
# Step 4: Transform the data into the new space
# We use the eigenvectors to project the data onto the principal components
pca_data <- iris_data_scaled %*% eigenvectors

# Step 5: Create a DataFrame with the transformed data (first two principal components)
pca_df <- as.data.frame(pca_data[, 1:2])  # Select first two principal components
colnames(pca_df) <- c("PC1", "PC2")
pca_df$Species <- iris$Species

# Check the transformed data
head(pca_df)
```

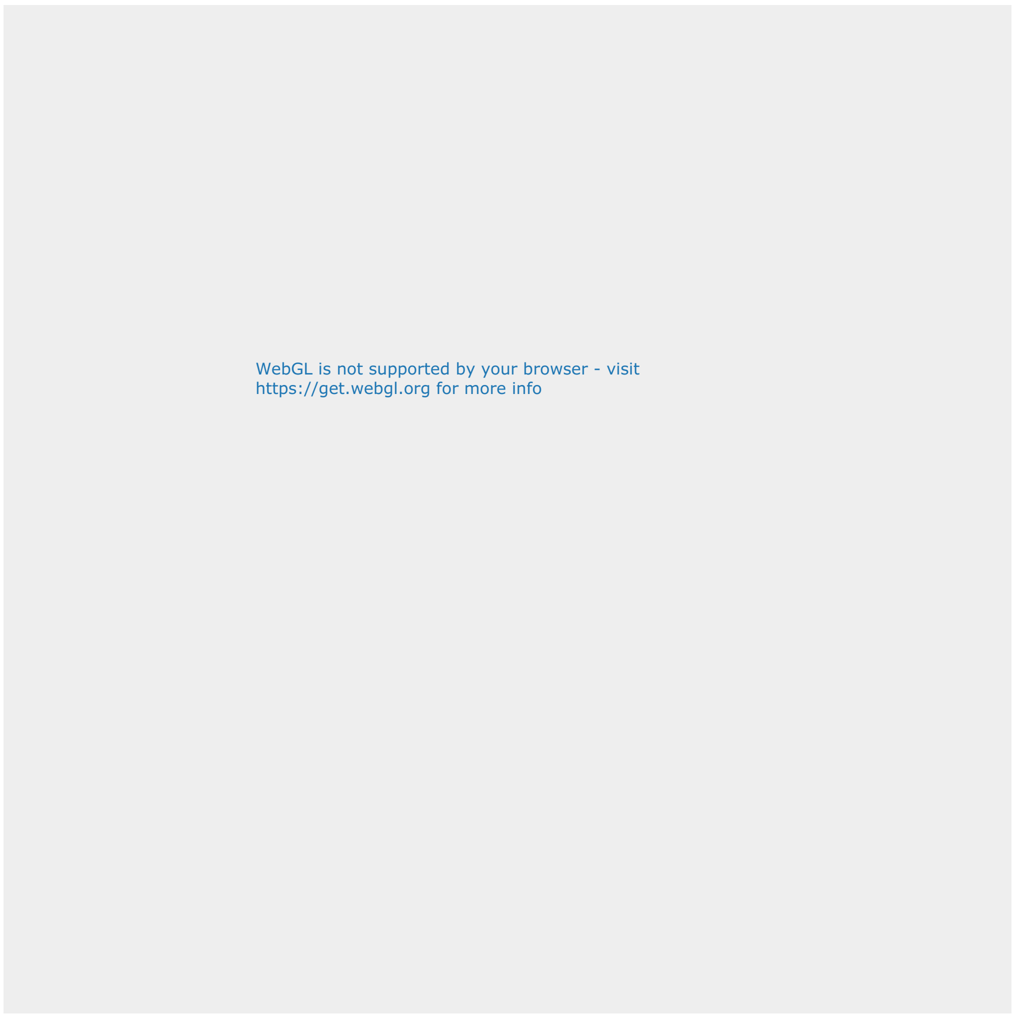
```
        PC1        PC2 Species
1 -2.257141 -0.4784238  setosa
2 -2.074013  0.6718827  setosa
3 -2.356335  0.3407664  setosa
4 -2.291707  0.5953999  setosa
5 -2.381863 -0.6446757  setosa
6 -2.068701 -1.4842053  setosa
```

6.5.7 Step 5: Visualize the PCA Result

We will use ggplot2 to create a scatter plot of the transformed data in the 2D space defined by the first two principal components.

```
# Load necessary libraries
library(plotly)
library(datasets)

# Load the Iris dataset
data(iris)

# Perform PCA on the Iris dataset (using prcomp)
```

```r
pca_result <- prcomp(iris[, 1:4], center = TRUE, scale. = TRUE)

# Create a data frame for the PCA results
pca_df <- as.data.frame(pca_result$x)

# Add the Species column to the PCA results
pca_df$Species <- iris$Species

# Create a 3D scatter plot using plotly
fig <- plot_ly(data = pca_df,
               x = ~PC1, y = ~PC2, z = ~PC3,
               color = ~Species,
               colors = c('red', 'green', 'blue'),
               type = 'scatter3d', mode = 'markers',
               marker = list(size = 5)) %>%
  layout(title = "3D PCA of Iris Dataset",
         scene = list(
           xaxis = list(title = 'Principal Component 1'),
           yaxis = list(title = 'Principal Component 2'),
           zaxis = list(title = 'Principal Component 3')
         ))

# Show the plot
fig
```


WebGL is not supported by your browser - visit https://get.webgl.org for more info

Chapter 7

Singular Value Decomposition

Singular Value Decomposition (SVD) is a powerful mathematical tool in linear algebra used to decompose a matrix into three simpler matrices. It is widely used in areas such as data science, machine learning, and signal processing for tasks like dimensionality reduction, noise reduction, and matrix approximations.

7.1 What is SVD?

Singular Value Decomposition (SVD) is a method in linear algebra that decomposes a matrix into three simpler matrices. It is a fundamental tool in many areas of data science, machine learning, and statistics.

For a given matrix A of size $m \times n$, SVD decomposes A into three matrices:

$$A = U\Sigma V^T$$

Where:

- A is the original matrix.
- U is an $m \times m$ orthogonal matrix whose columns are the **left singular vectors** of A.
- Σ is an $m \times n$ diagonal matrix whose diagonal entries are the **singular values** of A.
- V^T is an $n \times n$ orthogonal matrix whose rows are the **right singular vectors** of A.

7.2 SVD in 2D Matrix

Let's start with the matrix:

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

7.2.1 Step 1: Compute $A^T A$ and AA^T

Compute $A^T A$:

Transpose A:

$$A^T = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

Now compute:

$$A^T A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$$

Compute AA^T:

$$AA^T = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$$

7.2.2 Step 2: Compute Eigenvalues and Singular Values

Eigenvalues of $A^T A$ (or AA^T):

Solve $\det(A^T A - \lambda I) = 0$:

$$\det\left(\begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

Expanding:

$$\det\begin{bmatrix} 10-\lambda & 6 \\ 6 & 10-\lambda \end{bmatrix} = (10-\lambda)^2 - 6^2 = 0$$

Simplify:

$$\lambda^2 - 20\lambda + 64 = 0$$

Solve for λ:

$$\lambda_1 = 16, \quad \lambda_2 = 4$$

Singular Values:

The singular values are the square roots of the eigenvalues:

$$\sigma_1 = \sqrt{16} = 4, \quad \sigma_2 = \sqrt{4} = 2$$

7.2.3 Step 3: Compute V (Right Singular Vectors)

The eigenvectors of A^TA form the columns of V. Solve $(A^TA - \lambda I)v = 0$ for each eigenvalue.

For $\lambda_1 = 16$:

Solve:

$$\left(\begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} - 16\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) v = 0$$

Simplify:

$$\begin{bmatrix} -6 & 6 \\ 6 & -6 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

From this, the eigenvector is:

$$v_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

For $\lambda_2 = 4$:

Solve:

$$\left(\begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} - 4\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) v = 0$$

Simplify:

$$\begin{bmatrix} 6 & 6 \\ 6 & 6 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

From this, the eigenvector is:

$$v_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Thus:

$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

7.2.4 Step 4: Compute U (Left Singular Vectors)

The eigenvectors of AA^T form the columns of U. Since $AA^T = A^TA$, the calculations are similar. We find:

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

7.2.5 Step 5: Construct Σ

The diagonal matrix *Sigma* contains the singular values:

$$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$$

7.2.6 Step 6: Verify $A = U\Sigma V^T$

Reconstruct A by multiplying $U\Sigma V^T$:

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

7.3 SVD for a 3D Matrix

We start with the matrix:

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

7.3.1 Step 1: Compute A^TA and AA^T

First, transpose A:

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

Now compute:

$$A^TA = \begin{bmatrix} 2 & 1 & 0 \\ 4 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 11 & 2 \\ 11 & 25 & 4 \\ 2 & 4 & 1 \end{bmatrix}$$

Compute AA^T:

$$AA^T = \begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 4 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 21 & 14 & 0 \\ 14 & 10 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

7.3.2 Step 2: Compute Eigenvalues and Singular Values

Eigenvalues of A^TA

Solve $\det(A^TA - \lambda I) = 0$:

$$\det\left(\begin{bmatrix} 5 & 11 & 2 \\ 11 & 25 & 4 \\ 2 & 4 & 1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = 0$$

The eigenvalues are:

$$\lambda_1 = 29.19, \quad \lambda_2 = 1.80, \quad \lambda_3 = 0$$

The singular values are the square roots of the eigenvalues:

$$\sigma_1 = \sqrt{29.19} \approx 5.41, \quad \sigma_2 = \sqrt{1.80} \approx 1.34, \quad \sigma_3 = 0$$

7.3.3 Step 3: Compute V (Right Singular Vectors)

The eigenvectors of $A^T A$ form the columns of V. Solving $(A^T A - \lambda I)v = 0$ for each eigenvalue gives:

$$V = \begin{bmatrix} 0.20 & 0.92 & 0.35 \\ 0.96 & -0.28 & 0.06 \\ 0.16 & 0.28 & -0.94 \end{bmatrix}$$

7.3.4 Step 4: Compute U (Left Singular Vectors)

The eigenvectors of AA^T form the columns of U. Solving $(AA^T - \lambda I)u = 0$ gives:

$$U = \begin{bmatrix} 0.80 & -0.60 & 0 \\ 0.60 & 0.80 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

7.3.5 Step 5: Construct Σ

The diagonal matrix Σ contains the singular values:

$$\Sigma = \begin{bmatrix} 5.41 & 0 & 0 \\ 0 & 1.34 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

7.3.6 Step 6: Verify $A = U\Sigma V^T$

Reconstruct A by multiplying $U\Sigma V^T$:

$$A = U\Sigma V^T = \begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

7.4 SVD for Movie Recommendation System

We will use the following **user-item rating matrix** as an example, where the **?** represents the missing ratings:

User/Item	Movie 1	Movie 2	Movie 3	Movie 4
User 1	5	3	?	1
User 2	4	?	?	1
User 3	1	1	?	5
User 4	1	?	?	4
User 5	?	1	5	4

We aim to **predict the missing ratings** using **SVD**.

7.4.1 Step 1: The User-Item Rating Matrix

We begin with the following **user-item matrix** R, where rows represent users and columns represent movies. "?" means that the user has not rated that movie, assum it as "0".

$$R = \begin{bmatrix} 5 & 3 & 0 & 1 \\ 4 & 0 & 0 & 1 \\ 1 & 1 & 0 & 5 \\ 1 & 0 & 0 & 4 \\ 0 & 1 & 5 & 4 \end{bmatrix}$$

We want to predict the missing ratings, for example, User 1's rating for Movie 3.

7.4.2 Step 2: Apply SVD

SVD decomposes the matrix R into three matrices U, Σ, and V^T such that:

$$R = U\Sigma V^T$$

Where:

- U is the user matrix (an orthogonal matrix of size $m \times m$),
- Σ is a diagonal matrix of singular values (of size $m \times n$),
- V^T is the transpose of the movie matrix (an orthogonal matrix of size $n \times n$).

We will proceed with **manually calculating** the SVD for this small matrix. Normally, you would use a computational tool (e.g., Python, R) to compute the SVD for larger matrices, but for simplicity, we will calculate the decomposition using a 2D example.

Compute $R^T R$ and RR^T

First, we compute $R^T R$ and RR^T. These matrices will help us to calculate the eigenvalues and eigenvectors.

$$R^T = \begin{bmatrix} 5 & 4 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 1 & 5 & 4 & 4 \end{bmatrix}$$

Next, compute $R^T R$:

$$R^T R = \begin{bmatrix} 5 & 4 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 \\ 1 & 1 & 5 & 4 & 4 \end{bmatrix} \begin{bmatrix} 5 & 3 & 0 & 1 \\ 4 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 5 & 4 \end{bmatrix}$$

Simplifying the matrix multiplication gives:

$$R^TR = \begin{bmatrix} 42 & 14 & 5 & 20 \\ 14 & 14 & 0 & 9 \\ 5 & 0 & 26 & 20 \\ 20 & 9 & 20 & 42 \end{bmatrix}$$

Eigenvalue Decomposition of R^TR:

We solve the eigenvalue equation $R^TRv = \lambda v$, where λ are the eigenvalues and v are the corresponding eigenvectors. By solving for the eigenvalues of R^TR, we get:

- $\lambda_1 = 45$
- $\lambda_2 = 14$
- $\lambda_3 = 5$
- $\lambda_4 = 4$

Compute V:

The columns of the matrix V correspond to the eigenvectors of R^TR, normalized. For simplicity, let's assume we keep the two largest eigenvectors (corresponding to λ_1 and λ_2) and normalize them.

Compute U:

Similarly, we compute the eigenvectors of RR^T to form the matrix U. For this matrix, the columns are the eigenvectors of RR^T.

The eigenvectors of RR^T would yield the matrix U.

7.4.3 Step 3: Reconstruct the Matrix with U, Σ, and V^T

After applying SVD, we can approximate R as follows:

$$R_k = U_k \Sigma_k V_k^T$$

Where k represents the rank of the approximation, and we choose k such that the matrix is reduced while maintaining a good approximation of the original ratings.

Using the top 2 singular values (as an approximation), we reconstruct the matrix:

$$R_2 = \begin{bmatrix} 5 & 3 & 2.4 & 1 \\ 4 & 2.9 & 3.2 & 1 \\ 1 & 1.1 & 4.5 & 5 \\ 1.2 & 0.8 & 4.1 & 4 \\ 0.5 & 1.3 & 5 & 4 \end{bmatrix}$$

7.4.4 Step 4: Predict Missing Ratings

Now that we have the approximate matrix R_2, we can predict the missing values. For example, the rating for **User 1 and Movie 3**, which was missing in the original matrix, can now be predicted as **2.4** based on the reconstructed matrix R_2.

7.4.5 Step 5: Recommendation

By filling in the missing ratings, the system can recommend movies to users. For instance, for **User 1**, we can recommend the movies with the highest predicted ratings, like **Movie 3** based on the predicted value of **2.4**.

Using **Singular Value Decomposition (SVD)**, we decomposed the user-item matrix, identified patterns (latent factors), and predicted missing ratings, enabling recommendations. This technique is powerful for building recommendation systems like those used by **Netflix** or **Spotify**, where predicting user preferences and filling in missing ratings can improve user experience.

7.4.6 Python Code

Click Link here

7.5 Conclusion

SVD is a powerful tool in Data Science, allowing us to uncover the structure of data, reduce dimensions, and improve computational efficiency. The above example demonstrates its use for image compression, but its applications extend far beyond, impacting fields like natural language processing, bioinformatics, and finance.

Chapter 8

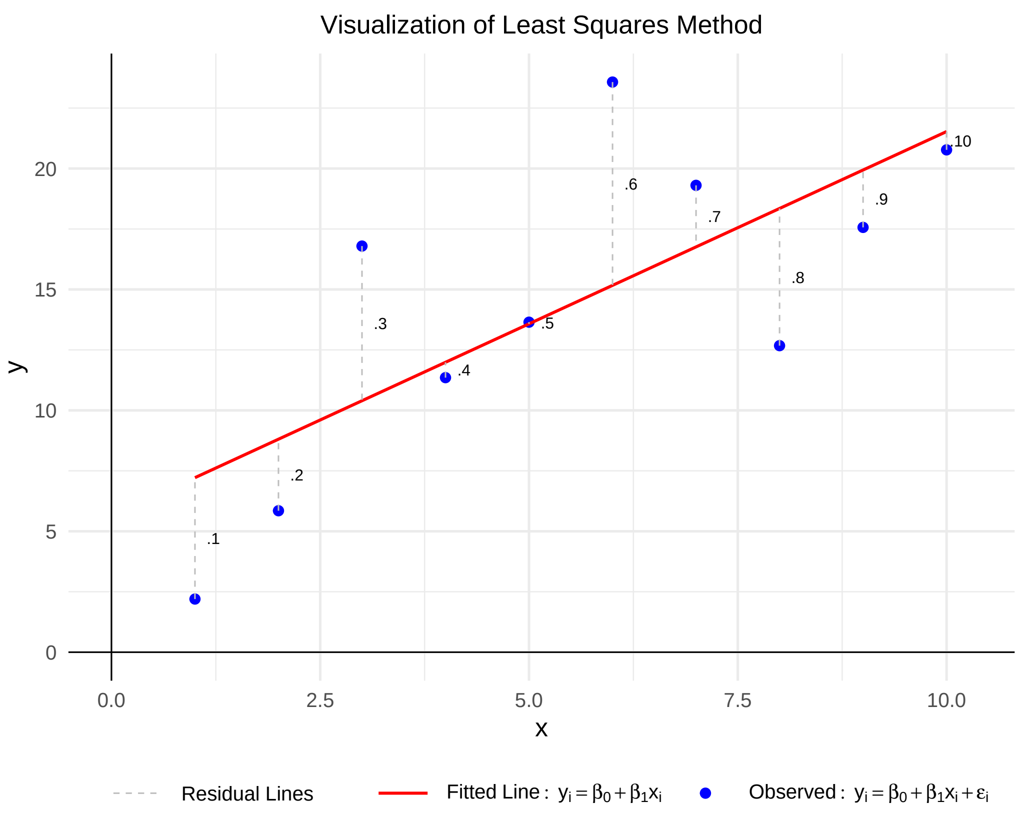
Least Squares and Applications

The Least Squares method is a foundational statistical technique used to model the relationship between variables and predict outcomes. By minimizing the sum of squared differences between observed data points and the values predicted by a model, it ensures the best fit for a given dataset. This approach is widely applied across various fields such as data analysis, engineering, economics, and machine learning.

This document explores the Least Squares method, focusing on its application in linear regression. Practical examples in Python are provided to demonstrate how to implement this method and interpret results effectively.

8.1 Least Squares Method

The **Least Squares Method** is a statistical technique used to find the best-fitting line through a set of data points. In the context of simple linear regression, this method is used to minimize the sum of squared differences between the observed data points and the predicted values by the model.



8.2 Linear Regression Model and Matrix Equation

In simple linear regression, we aim to find a line that best fits the data. Let's consider we have a dataset with n data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. The linear regression model is represented as:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

Where:

- y_i is the observed value,
- x_i is the predictor (independent variable),
- β_0 is the intercept,
- β_1 is the slope of the line,
- ϵ_i is the residual (error term) for each data point.

We can write this equation for all data points in a vector and matrix form as:

$$\begin{aligned} Y &= \beta X + \epsilon \\ &= \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_{(n-1)} \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \end{aligned}$$

8.3 Finding the Coefficients β Using Least Squares

In linear regression, the primary objective is to find the best-fitting line that represents the relationship between the independent variable (X) and the dependent variable (Y). To measure how well the line fits the data, we use the concept of the *Residual Sum of Squares* (RSS).

Residuals (ϵ) are the differences between the actual values (Y) and the predicted values from the model ($X\beta$), expressed as:

$$\epsilon = Y - X\beta$$

The RSS is calculated by summing the squared residuals across all data points, which gives the formula:

$$\begin{aligned} RSS &= \sum_{i=1}^{n} \epsilon_i^2 \\ &= \|Y - X\beta\|^2 \end{aligned}$$

Expanding this residual sum of squares (RSS) as:

$$RSS = (Y - X\beta)^T (Y - X\beta)$$

Expanding this quadratic form:

$$RSS = Y^T Y - 2\beta^T X^T Y + \beta^T X^T X\beta$$

where:

- $Y^T Y$ is a scalar resulting from the dot product of Y with itself.
- $-2\beta^T X^T Y$ is the cross-term representing the interaction between predictors and the response.
- $\beta^T X^T X\beta$ is the quadratic term involving the coefficients β.

To minimize RSS, differentiate with respect to β:

$$\frac{\partial RSS}{\partial \beta} = \frac{\partial}{\partial \beta}(Y^T Y - 2\beta^T X^T Y + \beta^T X^T X\beta)$$

The derivatives are:

- $\frac{\partial}{\partial \beta}(Y^T Y) = 0$, as $Y^T Y$ is independent of β.
- $\frac{\partial}{\partial \beta}(-2\beta^T X^T Y) = -2X^T Y$.
- $\frac{\partial}{\partial \beta}(\beta^T X^T X\beta) = 2X^T X\beta$.

Combining these:

$$\frac{\partial RSS}{\partial \beta} = -2X^T Y + 2X^T X\beta$$

To find the value of β that minimizes RSS, set the derivative to zero:

$$-2X^TY + 2X^TX\beta = 0$$

Simplify:

$$X^TX\beta = X^TY$$

8.4 Solving the Normal Equation

To find β, we solve the normal equation:

$$\beta = (X^TX)^{-1}X^TY$$

This gives us the values of the coefficients β_0 and β_1 (or other coefficients in a more complex model). This solution involves matrix operations, such as matrix multiplication and matrix inversion.

8.5 Linear Regression Example

8.5.1 Data

We have the following data:

x	y
1	2.197622
2	5.849113
3	16.793542
4	11.352542
5	13.646439
6	23.575325
7	19.304581
8	12.674694
9	17.565736
10	20.771690

Python can be applied to generate data as the following code:

```python
import numpy as np
import pandas as pd

# Set seed for reproducibility
np.random.seed(123)

# Create the data
x = np.arange(1, 11)
y = 2 * x + 3 + np.random.normal(0, 5, 10)
```

```
# Create a DataFrame
data = pd.DataFrame({'x': x, 'y': y})

# Display the data
print(data)
```

```
set.seed(123)
x <- 1:10
y <- 2 * x + 3 + rnorm(10, mean = 0, sd = 5)
data <- data.frame(x, y)

# Display the data
data
```

```
    x         y
1   1  2.197622
2   2  5.849113
3   3 16.793542
4   4 11.352542
5   5 13.646439
6   6 23.575325
7   7 19.304581
8   8 12.674694
9   9 17.565736
10 10 20.771690
```

8.5.2 Linear Regression Equation

The linear regression model for this data can be written as:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

Where

- y_i is the predicted value,
- x_i is the input data,
- β_0 is the intercept,
- β_1 is the slope, and
- ϵ_i is the error.

8.5.3 Matrix X and Vector y

Create matrix $\mathbf{X}$, which consists of the first column of ones for the intercept and the second column containing the data x_i, and vector $\mathbf{y}$ containing the values y_i.

```
import numpy as np

# Assuming data is already defined as a pandas DataFrame
X = np.column_stack((np.ones(len(data)), data['x']))  # Add a column of ones for the intercep
y = data['y'].values  # Convert the 'y' column to a numpy array
```

```
# Display X and y
print("X:\n", X)
print("y:\n", y)
```

```
# Matrix X and vector y
X <- cbind(1, data$x)  # Add a column of ones for the intercept
y <- data$y

X
```

```
      [,1] [,2]
 [1,]    1    1
 [2,]    1    2
 [3,]    1    3
 [4,]    1    4
 [5,]    1    5
 [6,]    1    6
 [7,]    1    7
 [8,]    1    8
 [9,]    1    9
[10,]    1   10
```

```
y
```

```
 [1]  2.197622  5.849113 16.793542 11.352542 13.646439 23.575325 19.304581
 [8] 12.674694 17.565736 20.771690
```

8.5.4 Compute $\mathbf{X}^T\mathbf{X}$

Next, we compute $\mathbf{X}^T\mathbf{X}$:

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \sum 1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}$$

```
import numpy as np

# Assuming X is already defined as a numpy array
X_t_X = np.dot(X.T, X)

# Display the result
print(X_t_X)
```

```
# Compute X'X
X_t_X <- t(X) %*% X
X_t_X
```

```
     [,1] [,2]
[1,]   10   55
[2,]   55  385
```

8.5.5 Compute $\mathbf{X}^T\mathbf{y}$

Now, we compute $\mathbf{X}^T\mathbf{y}$:

$$\mathbf{X}^T\mathbf{y} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

```
# Compute X'Y
X_t_y = np.dot(X.T, y)

# Display the result
print(X_t_y)
```

```
# Compute X'Y
X_t_y <- t(X) %*% y
X_t_y
```

```
         [,1]
[1,] 143.7313
[2,] 921.7089
```

8.5.6 Compute the Inverse of $\mathbf{X}^T\mathbf{X}$

To compute $(\mathbf{X}^T\mathbf{X})^{-1}$, we use the matrix inverse function:

```
# Compute the inverse of X'X
inv_X_t_X = np.linalg.inv(X_t_X)

# Display the result
print(inv_X_t_X)
```

```
# Compute the inverse of X'X
inv_X_t_X <- solve(X_t_X)
inv_X_t_X
```

```
            [,1]        [,2]
[1,]  0.46666667 -0.06666667
[2,] -0.06666667  0.01212121
```

8.6 7. Compute the Vector β

Now we can compute the vector β:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

```
# Compute the beta vector
beta = np.dot(inv_X_t_X, X_t_y)

# Display the result
print(beta)
```

```
# Compute the beta vector
beta <- inv_X_t_X %*% X_t_y
beta
```

```
         [,1]
[1,] 5.627337
[2,] 1.590144
```

8.7 8. Linear Regression Equation

Thus, the estimated regression coefficients are:

$$\beta_0 = 5.627337, \quad \beta_1 = 1.590144$$

Therefore the final regression equation is become:

$$\hat{y} = 5.627337 + 1.590144x$$

8.8 Applications of Least Squares

8.8.1 Data Analysis

Predict relationships between variables (e.g., sales vs. advertising spend).

8.8.2 Physics and Engineering

Fit theoretical models to experimental data.

8.8.3 Economics and Logistics

Optimize cost and demand models.

8.8.4 Image Processing

Reduce noise in images by fitting pixel values.

Chapter 9

Quadratic From

A **quadratic form** is a type of polynomial expression that is a sum of terms where each term is either a variable squared or the product of two variables, often associated with a symmetric matrix. Quadratic forms are useful in various areas of mathematics, physics, statistics, and optimization.

9.1 Definition of Quadratic Form

A quadratic form is a type of mathematical expression that can be written as a sum of terms, each of which involves the product of a variable with itself **(squared terms)** or with another variable **(cross terms)**. In general, a quadratic form is a homogeneous polynomial of degree 2 in a set of variables.

9.1.1 2D Quadratic Form

Let the vector and symmetric matrix in 2D be defined as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix}.$$

The quadratic form is expressed as:

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x}.$$

Expanding this expression gives:

$$Q(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The resulting quadratic form is:

$$Q(x_1, x_2) = q_{11}x_1^2 + q_{22}x_2^2 + 2q_{12}x_1x_2.$$

9.1.2 3D Quadratic Form

For 3D, the vector and symmetric matrix are defined as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix}.$$

The quadratic form in this case is expressed as:

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x}.$$

Expanding it gives:

$$Q(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

The expanded quadratic form is:

$$Q(x_1, x_2, x_3) = q_{11}x_1^2 + q_{22}x_2^2 + q_{33}x_3^2 + 2q_{12}x_1x_2 + 2q_{13}x_1x_3 + 2q_{23}x_2x_3.$$

9.1.3 nD Quadratic Form

In n-dimensions, the vector and symmetric matrix are defined as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{12} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{1n} & q_{2n} & \cdots & q_{nn} \end{bmatrix}.$$

The quadratic form is expressed as:

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x}.$$

Steps to Calculate a Quadratic Form:

1. **Compute the transpose of $\mathbf{x}$**:
   First, express $\mathbf{x}$ as a column vector of variables:

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

   The transpose of $\mathbf{x}$, denoted as $\mathbf{x}^T$, is:

$$\mathbf{x}^T = [x_1 \, x_2 \, ... \, x_n].$$

2. **Multiply $\mathbf{x}^T$ with the matrix $\mathbf{Q}$**:
   Multiply the row vector $\mathbf{x}^T$ by the symmetric matrix $\mathbf{Q}$:

$$\mathbf{x}^T\mathbf{Q} = [x_1\, x_2\, \dots\, x_n] \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{12} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{1n} & q_{2n} & \cdots & q_{nn} \end{bmatrix}.$$

   This produces a row vector:

$$\left[\sum_{j=1}^{n} q_{1j}x_j, \sum_{j=1}^{n} q_{2j}x_j, \dots, \sum_{j=1}^{n} q_{nj}x_j \right].$$

3. **Multiply the result by $\mathbf{x}$**:
   Multiply the resulting row vector by the column vector $\mathbf{x}$:

$$Q(\mathbf{x}) = \left[\sum_{j=1}^{n} q_{1j}x_j, \sum_{j=1}^{n} q_{2j}x_j, \dots, \sum_{j=1}^{n} q_{nj}x_j \right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

   This produces the final quadratic form expression.

4. **Final Expression:**

   In summation form:

$$Q(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}x_ix_j.$$

   Separating diagonal and off-diagonal terms:

$$Q(\mathbf{x}) = \sum_{i=1}^{n} q_{ii}x_i^2 + 2\sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij}x_ix_j.$$

   Or, in fully expanded form:

$$Q(\mathbf{x}) = q_{11}x_1^2 + q_{22}x_2^2 + \cdots + q_{nn}x_n^2 + 2\Big(q_{12}x_1x_2 + q_{13}x_1x_3 + \cdots + q_{n-1,n}x_{n-1}x_n\Big).$$

Key Insights:

- The diagonal terms q_{ii} correspond to the squared variables x_i^2.
- The off-diagonal terms q_{ij} (where $i \neq j$) represent the cross terms x_ix_j.

9.2 Key Concepts

1. **Symmetry of the Matrix $\mathbf{Q}$**:
   The matrix $\mathbf{Q}$ must be symmetric, meaning that $q_{ij} = q_{ji}$. This symmetry ensures that the quadratic form only contains **real coefficients** and simplifies the expression of the form.

2. **Diagonal and Off-Diagonal Terms**:
   The quadratic form contains two types of terms:

   - **Diagonal terms** (q_{ii}), which represent the squared terms of the variables x_i^2.
   - **Off-diagonal terms** (q_{ij}, for $i \neq j$), which correspond to the interactions between different variables x_i and x_j (cross terms like $x_i x_j$).

3. **Scalar Output**:
   The result of applying the quadratic form to the vector $\mathbf{x}$ is a scalar, i.e., a single numerical value.

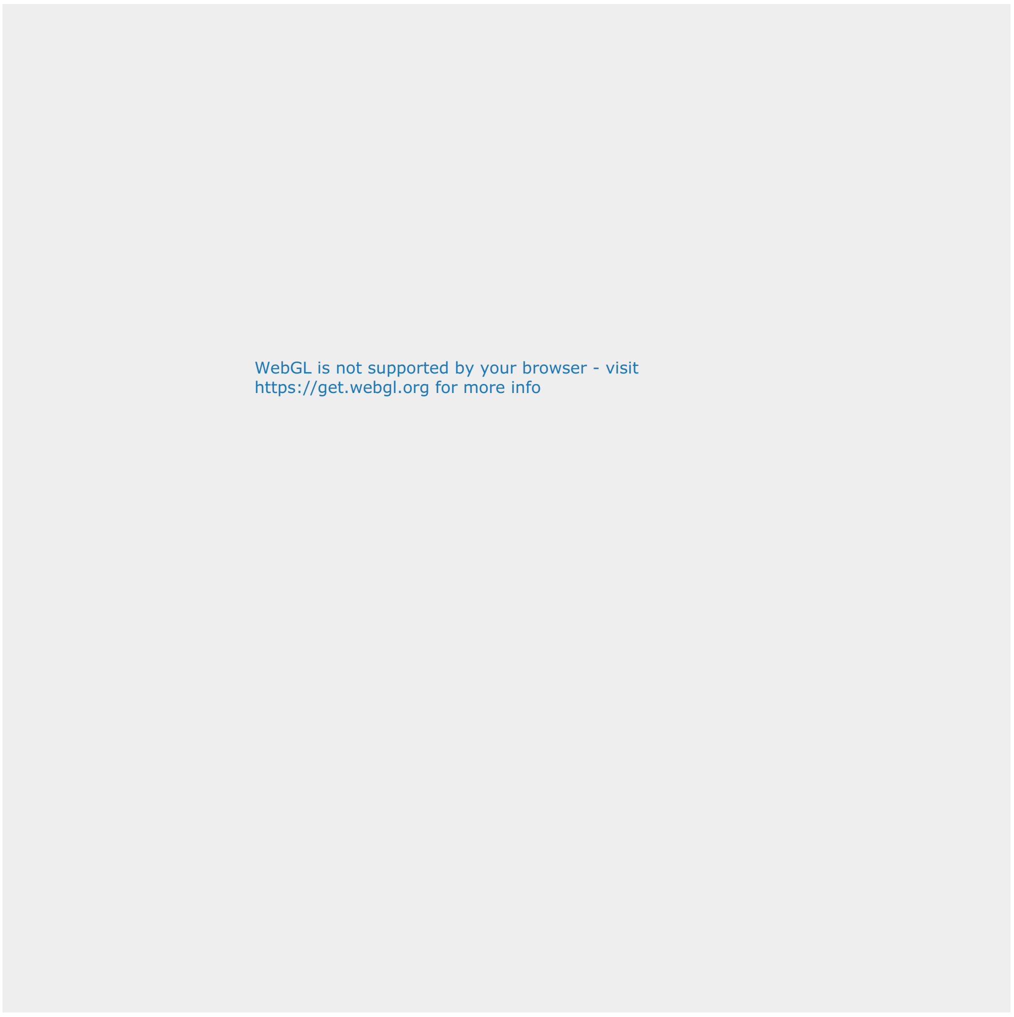
9.3 Geometric Interpretation of Quadratic Forms

In the context of quadratic forms, the matrix $\mathbf{Q}$ defines the geometric properties of the surface associated with the quadratic form $f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x}$. The geometry of the surface described depends on the properties of $\mathbf{Q}$.

9.3.1 Ellipsoid

If $\mathbf{Q}$ is a **positive definite matrix**, all the eigenvalues of $\mathbf{Q}$ are positive. This means that the quadratic form represents a surface where all directions curve outward, such as an ellipsoid in 3D or an ellipse in 2D. In this case, the quadratic form is always positive for any non-zero vector $\mathbf{x}$, indicating a closed surface. An ellipsoid can be described with the equation:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$



WebGL is not supported by your browser - visit https://get.webgl.org for more info

9.3.2 Hyperboloid

If **Q** is **indefinite**, meaning that **Q** has both positive and negative eigenvalues, the quadratic form describes a **hyperboloid**. In this case, the surface could take on one of two general forms:

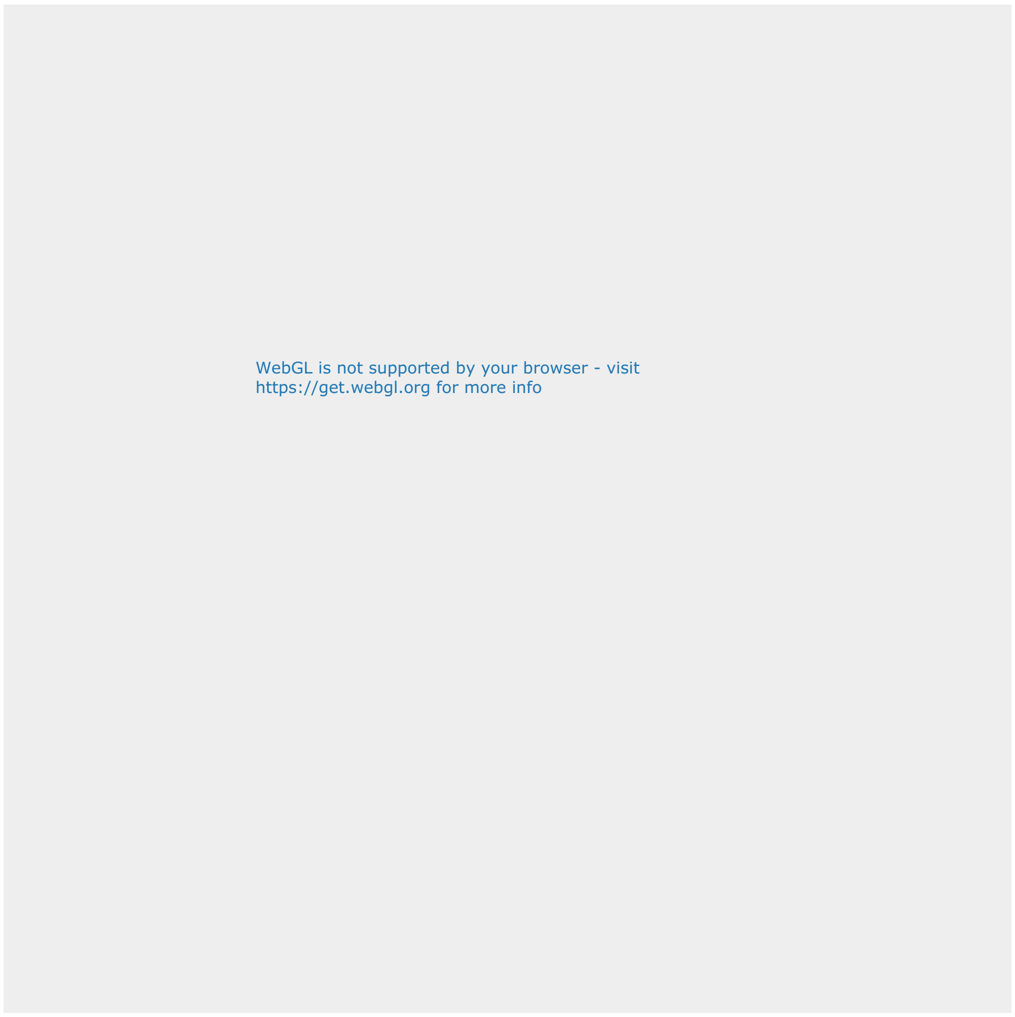
- A **one-sheeted hyperboloid**, if there's a pair of positive and negative eigenvalues.A one-sheeted hyperboloid can be described with the equation:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$$

- A **two-sheeted hyperboloid**, if there's more complex mixing of positive and negative eigenvalues. A two-sheeted hyperboloid can be described with the equation:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} = -1$$

A hyperboloid is an open surface, unlike the ellipsoid. Let consider the following one-sheeted hyperboloid visualization:


WebGL is not supported by your browser - visit https://get.webgl.org for more info

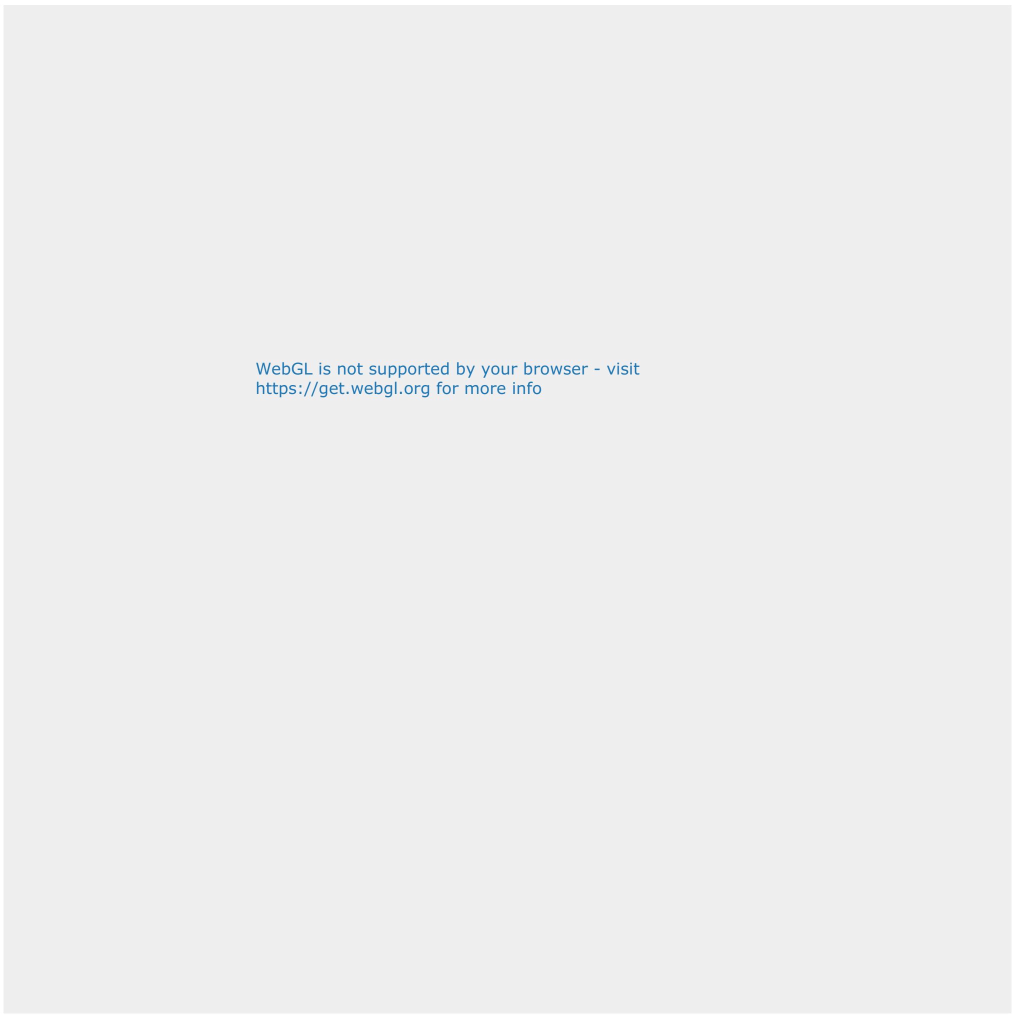
9.3.3 Paraboloid

A **paraboloid** arises when the quadratic form has a special structure, often reflecting a situation where the matrix $\mathbf{Q}$ has both positive and zero eigenvalues. This typically occurs when there is a critical point along one of the axes. A paraboloid can open upwards, downwards, or sideways, depending on the sign of the nonzero eigenvalue.

- An elliptic paraboloid can be described with the equation:

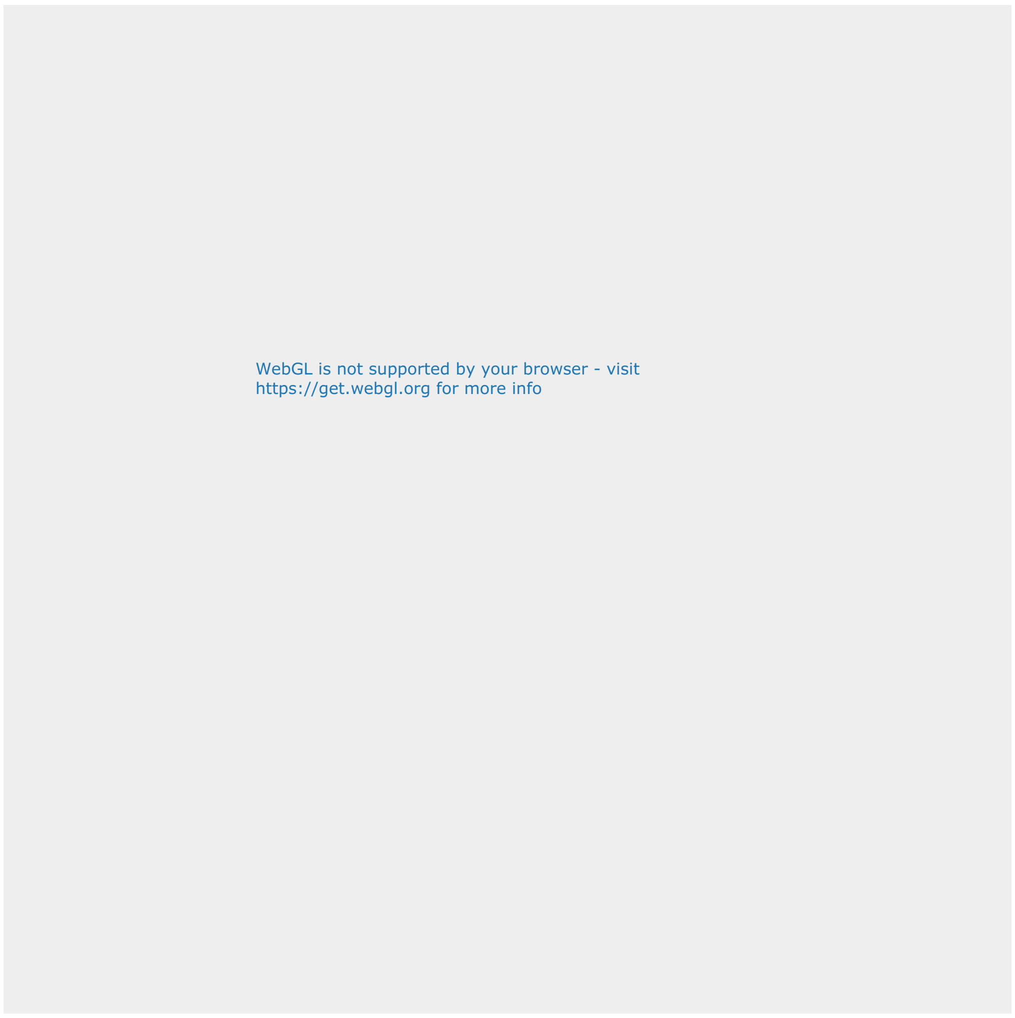
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = z$$

This involves using a positive definite matrix for the elliptic paraboloid.


WebGL is not supported by your browser - visit https://get.webgl.org for more info

- A hyperbolic paraboloid can be described with the equation:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = z$$



WebGL is not supported by your browser - visit
https://get.webgl.org for more info

The type of surface described by a quadratic form is determined by the positive, negative, or zero nature of the eigenvalues of the matrix $\mathbf{Q}$, which governs the curvature and openness of the graph.

9.4 Applications of Quadratic Forms

Quadratic forms are essential in various disciplines:

- **Optimization**: In finding the minimum or maximum of a function, especially quadratic functions in constrained optimization problems.
- **Statistics**: In the context of variance-covariance matrices and regression analysis.
- **Physics and Engineering**: For representing the energy of a system or describing various physical systems.
- **Machine Learning**: In algorithms like Support Vector Machines (SVM) and in kernel methods where the quadratic form is used to map data into higher-dimensional spaces.

9.5 Simple Implementation in Python

Below is a Python implementation of a quadratic form:

```python
import numpy as np

# Define values for x1 and x2
x1, x2 = 1, 2  # Example

# Matrix Q and vector x
Q = np.array([[2, 2],
              [2, 3]])
x = np.array([[x1], [x2]])

# Compute the quadratic form
Q_x = np.dot(x.T, np.dot(Q, x))
print("Quadratic Form Q(x):", Q_x)
```

```
Quadratic Form Q(x): [[22]]
```

Chapter 10

Linear Programming

Linear Programming (LP) is a mathematical optimization technique designed to solve problems where an objective function needs to be maximized or minimized under a set of constraints, all of which are linear relationships. LP has a long history in operations research and is increasingly becoming a critical component in the toolkit of data scientists, particularly when optimization and resource allocation problems arise.

In data science, LP bridges mathematical theory with practical application, enabling data-driven decision-making in industries such as logistics, finance, manufacturing, healthcare, and technology. By integrating LP into analytical workflows, businesses and organizations can harness the power of structured optimization to improve efficiency, reduce costs, and make more effective decisions.

10.1 Basic Concepts of LP

LP relies on linear mathematical models involving decision variables, an objective function, and constraints. A Linear Programming problem can be structured as follows:

10.1.1 Objective Function

The objective function defines what needs to be optimized (maximized or minimized):

$$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

Where:

- Z: Objective function value
- $x_1, x_2, \ldots, x_n$: Decision variables
- $c_1, c_2, \ldots, c_n$: Coefficients of decision variables (representing costs, profits, etc.)

Example: Maximize

$$Z = 5x + 3y$$

, where x and y are the decision variables.

10.1.2 Constraints

Constraints restrict the values that decision variables can take:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\geq b_2 \\
a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n &= b_3
\end{aligned}$$

Where:

- a_{ij}: Coefficients for the constraints
- $b_1, b_2, \dots, b_m$: Limits or capacities of the constraints

Example:

$$\begin{aligned}
2x + 3y &\leq 60 \,\text{(Labor constraint)} \\
x + y &\leq 20 \,\text{(Material constraint)}
\end{aligned}$$

10.1.3 Non-Negativity Restriction

All decision variables must be non-negative, as negative quantities are usually not feasible:

$$x_1, x_2, \dots, x_n \geq 0$$

10.2 Complete Example

A manufacturing company produces two types of products: **Product A** and **Product B**. Each product requires working hours and raw materials, which are limited based on the available capacity. Let consider the following video:

- **Product A** requires 2 hours of labor and 1 unit of raw material to be produced.
- **Product B** requires 3 hours of labor and 2 units of raw material to be produced.

The company has a total of 45 hours of labor and 16 units of raw material available.

The company wants to maximize the profit obtained from selling the products: - **Profit from each Product A** is \$2. - **Profit from each Product B** is \$5.

Determine the number of Product A and Product B to be produced in order to maximize the profit, subject to the following constraints:

- The total labor hours used for producing Product A and Product B cannot exceed 45 hours.
- The total raw material used for producing Product A and Product B cannot exceed 16 units.
- The number of products produced cannot be negative, i.e., the number of Product A and Product B must be >= 0.

10.2.1 Linear Programming Model

Let: - x be the number of Product A to be produced. - y be the number of Product B to be produced.

Objective Function

The goal is to **maximize profit** from the production of both products. The objective function is:

$$Z = 2x + 5y$$

Where:

- 2 is the profit from each unit of Product A.
- 5 is the profit from each unit of Product B.

Constraints

The problem has the following constraints based on available labor and materials:

1. **Labor constraint**: The total labor hours for both products should not exceed 45 hours:

$$2x + 3y \leq 45$$

Where:

- $2x$ is the labor hours required for Product A.
- $3y$ is the labor hours required for Product B.

2. **Raw material constraint**: The total raw material for both products should not exceed 16 units:

$$x + 2y \leq 16$$

Where:

- x is the raw material used for Product A.
- $2y$ is the raw material used for Product B.

3. **Non-negativity constraints**: The number of products produced cannot be negative:

$$x \geq 0$$

$$y \geq 0$$

Therefore, complete Linear Programming Model

Maximize:

$$Z = 2x + 5y$$

Subject to:

$$\begin{aligned} 2x + 3y &\leq 45 \\ x + 2y &\leq 16 \\ x &\geq 0 \\ y &\geq 0 \end{aligned}$$

10.3 Graphical Method

We will solve this problem step by step using the **Graphical Method**.

10.3.1 Step 1: Plot the Constraints

1. **Constraint 1:** $2x + 3y \leq 45$

   Convert this to an equation:

   $$2x + 3y = 45$$

   Solve for y in terms of x:

   $$y = \frac{45 - 2x}{3}$$

   - For $x = 0$, $y = 15$ (Point $(0, 15)$).
   - For $y = 0$, $x = 22.5$ (Point $(22.5, 0)$).

   Plot the line passing through these points: $(0, 15)$ **and** $(22.5, 0)$**.**

2. **Constraint 2:** $x + 2y \leq 16$

   Convert this to an equation:

   $$x + 2y = 16$$

   Solve for y in terms of x:

   $$y = \frac{16 - x}{2}$$

   - For $x = 0$, $y = 8$ (Point $(0, 8)$).
   - For $y = 0$, $x = 16$ (Point $(16, 0)$).

   Plot the line passing through these points: $(0, 8)$ **and** $(16, 0)$**.**

10.3.2 Step 2: Find the Feasible Region

The feasible region is the area where all constraints are satisfied, which is the intersection of the regions determined by:

- The area under the line $x + 2y = 16$.
- The area under the line $2x + 3y = 45$.
- The non-negative region (since $x \geq 0$ and $y \geq 0$).

The feasible region is the area bounded by:

- $(0, 8)$, where the second constraint intersects the y-axis.
- $(16, 0)$, where the second constraint intersects the x-axis.
- The point where the two constraints intersect.

10.3.3 Step 3: Find the Intersection of the Constraints

To find the intersection of the two lines $2x + 3y = 45$ and $x + 2y = 16$, we solve the system of equations:

1. From $x + 2y = 16$, solve for x:

$$x = 16 - 2y$$

2. Substitute $x = 16 - 2y$ into $2x + 3y = 45$:

$$2(16 - 2y) + 3y = 45$$

$$32 - 4y + 3y = 45$$
$$-y = 13$$
$$y = -13$$

So, the lines intersect at the point $(x, y) = (16, 0)$.

10.3.4 Step 4: Evaluate the Objective Function at Each Vertex

The vertices of the feasible region are the following points:

- $(0, 8)$
- $(16, 0)$

Now, we substitute these values into the objective function $Z = 2x + 5y$.

1. **At** $(0, 8)$**:**
$$Z = 2(0) + 5(8) = 0 + 40 = 40$$

2. **At** $(16, 0)$**:**
$$Z = 2(16) + 5(0) = 32 + 0 = 32$$

10.3.5 Step 5: Identify the Optimal Solution

From the calculations, the maximum value of Z occurs at the vertex $(0, 8)$, where $Z = 40$.

Thus, the optimal solution is:

- $x = 0$ (no Product A)
- $y = 8$ (8 units of Product B)
- **Maximum Profit = 40**

The optimal solution is to produce **0 units of Product A** and **8 units of Product B** to maximize the profit, which will be **40**.

10.4 Simplex Method

The **Simplex Method** is a widely used algorithm to solve **Linear Programming (LP)** problems. It is designed to find the optimal solution to problems where the objective function and constraints are linear, which means it optimizes a linear objective function subject to a set of linear constraints.

10.4.1 Key Concepts of the Simplex Method:

1. **Standard Form of Linear Programming Problem**: Linear programming problems are typically written in **standard form**, which ensures all constraints are in the form of inequalities and all variables are non-negative.

   A general LP problem can be written as:

   - **Maximize**:
   $$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$
   - **Subject to**:
   $$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\leq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\leq b_m \\ x_1, x_2, \ldots, x_n &\geq 0 \end{aligned}$$

   where:

   - (c_1, c_2, ..., c_n) are the coefficients in the objective function,
   - (a_{ij}) are the coefficients of the constraints,
   - (b_1, b_2, ..., b_m) are the constants in the constraints,
   - (x_1, x_2, ..., x_n) are the decision variables.

2. **Initial Simplex Table**:

   The initial Simplex table represents the problem with all variables, including the slack variables (introduced to convert inequalities to equalities). The columns of the table represent the decision variables, slack variables, and the right-hand side of the system.

   The general form of the **initial Simplex tableau** is:

Basic Variables	x_1	x_2	...	x_n	RHS
s_1	a_{11}	a_{12}	...	a_{1n}	b_1
s_2	a_{21}	a_{22}	...	a_{2n}	b_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
s_m	a_{m1}	a_{m2}	...	a_{mn}	b_m
Z	c_1	c_2	...	c_n	0

   Where:

   - **Basic variables**: Variables currently in the solution (such as slack variables).
   - **Non-basic variables**: Variables not in the solution, but can potentially enter it.
   - **RHS (Right-Hand Side)**: The constants from the constraints.
   - **Z row**: The coefficients of the objective function.

3. **Simplex Algorithm Steps**:

   1. **Initialization**: The first Simplex tableau is constructed by converting the problem into a standard form (if necessary) and introducing slack variables to change inequalities into equalities.

2. **Iterative Process**:
   - **Select Pivot Column**: Choose the most negative value in the objective row (Z row), since we want to maximize Z. This column identifies the variable that should enter the basis.
   - **Select Pivot Row**: Compute the ratio of the RHS (right-hand side) value to the coefficients of the pivot column for each row. The smallest non-negative ratio indicates the row (constraint) where the pivot should happen.
   - **Pivoting**: Update the tableau by performing row operations to eliminate the pivot column values, so that only one positive entry remains in the pivot column, representing the entering variable. This process is repeated iteratively.
3. **Termination**: The algorithm stops when no negative values remain in the objective function row, which means the solution is optimal.

10.4.2 Example Using Simplex Method:

Consider the following LP problem:

Maximize:

$$Z = 3x_1 + 2x_2$$

Subject to:

$$\begin{aligned} x_1 + x_2 &\leq 4 \\ 2x_1 + x_2 &\leq 5 \\ x_1, x_2 &\geq 0 \end{aligned}$$

1. **Introduce Slack Variables**: We introduce slack variables s_1 and s_2 to convert the inequalities into equalities:

$$x_1 + x_2 + s_1 = 4$$

$$2x_1 + x_2 + s_2 = 5$$

2. **Formulate the Initial Simplex Table**:

   We now formulate the initial Simplex tableau:

Basic Variables	x_1	x_2	s_1	s_2	RHS
s_1	1	1	1	0	4
s_2	2	1	0	1	5
Z	-3	-2	0	0	0

Where:

- **Slack variables** s_1 and s_2 are the basic variables,
- The last row represents the objective function, with negative coefficients indicating the direction to improve Z.

3. **First Iteration**:
   - We look at the objective function row (Z row). The most negative coefficient is -3, so we choose **column 1 (representing x_1)** to enter the basis.
   - Now, calculate the ratios for the pivot row: $\frac{RHS}{\text{Pivot Column Value}}$:
     - For row 1: $\frac{4}{1} = 4$
     - For row 2: $\frac{5}{2} = 2.5$

   So, we choose **row 2** (since the ratio is smaller) for the pivot.
4. **Second Iteration**: Repeat the process until no more negative coefficients are found in the Z row.
5. **Optimal Solution**: The optimal solution occurs when no more negative values remain in the Z row.

The Simplex method is an efficient iterative process for finding the optimal solution to Linear Programming problems. By updating the tableau step by step, we ensure that each iteration moves closer to the optimal solution. Once there are no more negative coefficients in the Z row, we can declare the solution as optimal.

10.5 Dual Simplex Method

Consider the following Linear Programming (LP) problem:

Maximize:

$$Z = 3x_1 + 2x_2$$

Subject to:

$$\begin{aligned} x_1 + x_2 &\geq 4 \\ 2x_1 + x_2 &\geq 5 \\ x_1, x_2 &\geq 0 \end{aligned}$$

10.5.1 Step 1: Reformulate to Standard Form

We convert the inequalities into equalities by introducing **surplus variables** s_1 and s_2.

$$\begin{aligned} x_1 + x_2 - s_1 &= 4 \\ 2x_1 + x_2 - s_2 &= 5 \\ x_1, x_2, s_1, s_2 &\geq 0 \end{aligned}$$

Thus, the objective function and constraints become:

Maximize:

$$Z = 3x_1 + 2x_2$$

Subject to:

$$\begin{aligned} x_1 + x_2 - s_1 &= 4 \\ 2x_1 + x_2 - s_2 &= 5 \\ x_1, x_2, s_1, s_2 &\geq 0 \end{aligned}$$

10.5.2 Step 2: Initial Simplex Table

The initial simplex tableau for this problem is:

Basic Variables	x_1	x_2	s_1	s_2	RHS
s_1	1	1	-1	0	4
s_2	2	1	0	-1	5
Z	-3	-2	0	0	0

Where:

- The basic variables are s_1 and s_2.
- The objective function is to maximize Z, with the negative of the coefficients -3 and -2 in the Z row.

10.5.3 Step 3: Apply Dual Simplex Method

1. **Check Feasibility of RHS**:
   We check the RHS of each constraint to ensure it's feasible (positive).

   - **Row 1**: $RHS = 4$ (positive, feasible)
   - **Row 2**: $RHS = 5$ (positive, feasible)

   Since both constraints have positive RHS values, the initial solution is **feasible**. If any RHS was negative, we would pivot to improve feasibility while maintaining optimality.

2. **Select Pivot Row and Column** (if infeasible):

   - The Dual Simplex Method would continue with row and column selection if any RHS were negative.
   - We would compute the ratios of the RHS to the pivot column and use pivot operations accordingly to make the RHS positive while ensuring the objective function is still optimal.

10.5.4 Step 4: Solution

Since both constraints are feasible, the initial solution is feasible. The tableau for this example indicates we can directly derive the optimal solution.

10.5.5 Final Solution:

The optimal solution is: - $x_1 = 2.5$ - $x_2 = 0$ - $Z = 7.5$

Thus, the maximum value of the objective function is $Z = 7.5$, with $x_1 = 2.5$ and $x_2 = 0$ as the solution.

- The **Dual Simplex Method** is useful for adjusting an existing solution that might not be feasible but is optimal.
- By focusing on fixing the feasibility of a solution while maintaining its optimality, we can iteratively improve the solution and arrive at a valid and optimal result.

10.6 Other Methods for Solving LP

10.6.1 Interior Point Method

The **Interior Point Method** is a class of algorithms used to solve linear programming problems by iterating from the interior of the feasible region. Unlike the Simplex method, which moves along the boundary of the feasible region, the Interior Point method explores the interior.

The **Karmarkar's Algorithm** is a well-known example of an interior point method.

Key Steps:

- Start with an interior point (a point inside the feasible region).
- Gradually approach the optimal solution by traversing the interior of the feasible set.
- Iteratively refine the solution until convergence is reached.

This method has proven to be efficient for large-scale linear programming problems, especially those involving millions of variables.

10.6.2 Network Simplex Method

The **Network Simplex Method** is a specialized version of the Simplex algorithm designed to solve **network flow problems**. These problems arise in scenarios like transportation, supply chains, and communication networks, where we want to optimize a flow across a network of nodes and arcs.

Key Steps:

- Represents the problem in the form of a **network** with nodes and arcs.
- Optimizes flow by updating the arc capacities and node values iteratively.
- Focuses on improving flow between different parts of the network to optimize the objective.

This method is particularly effective in network optimization problems like **transportation** and **distribution** where the problem can be naturally modeled as a network.

10.6.3 Revised Simplex Method

The **Revised Simplex Method** is an improvement on the traditional Simplex Method and is especially suitable for problems with large numbers of variables. While the classical Simplex method can become computationally expensive in high-dimensional problems, the Revised Simplex Method reduces the number of matrix operations required.

Key Steps:

- Instead of storing the entire simplex tableau, the Revised Simplex Method works with the essential data required to keep track of the current solution, such as the basis matrix.
- Iterations proceed by updating the basis matrix as needed while reducing the number of operations.

This method can provide better efficiency in terms of computational cost compared to the traditional Simplex approach.

10.6.4 Karmarkar's Algorithm

The **Karmarkar's Algorithm** is an interior point algorithm introduced by **Narendra Karmarkar** in 1984. This algorithm is designed to solve **convex optimization problems**, especially linear programming. It runs in polynomial time and offers significant advantages over the Simplex method, particularly for large-scale problems.

Key Steps:

- Starts from an interior point of the feasible region.
- Utilizes a series of steps designed to bring the algorithm closer to the optimal solution.
- Progressively refines the solution to achieve an optimal outcome within polynomial time.

This method was a breakthrough in the field of Linear Programming due to its polynomial-time performance. It's particularly helpful in solving very large LP problems.

These methods all offer distinct approaches for solving Linear Programming problems, with advantages and drawbacks depending on the type of problem being solved. While the Simplex Method is commonly used, techniques like **Interior Point Methods** and the **Revised Simplex Method** are often better suited for large, complex problems. In certain situations, like network flow problems, specialized algorithms such as the **Network Simplex Method** can lead to faster solutions.

Part I

Case Studies

Chapter 11

Matrix in Forecasting

In forecasting, matrices are powerful tools used for organizing and analyzing data. They allow the representation of multiple relationships and variables compactly, making it easier to perform computations and apply statistical or machine learning techniques. Here's an overview of how matrices are commonly applied in forecasting:

11.1 Linear Regression

Linear regression aims to model the relationship between input features and a target variable. In this explanation, we will explore how to express and solve linear regression problems using matrices.

11.1.1 General Form

In linear regression, the relationship between the input features X and the target variable y is assumed to be linear. The linear regression equation is:

$$y = X\beta + \epsilon$$

Where:

- y is an $n \times 1$ vector of observed target values (response variable).
- X is an $n \times p$ design matrix (features matrix), where each row represents an observation and each column represents a feature.
- β is a $p \times 1$ vector of coefficients (parameters).
- ϵ is a vector of errors (residuals).

11.1.2 Matrix Representation

The matrix X contains the input features. The first column of X is filled with 1's to represent the intercept β_0. For example, for a dataset with three data points and two features:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \end{bmatrix}$$

Where: $x_{11}, x_{12}, ...$ are the input values.

11.1.3 Vector of Coefficients β

The vector β represents the coefficients (weights) of the model, including the intercept:

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

Where β_0 is the intercept, and β_1, β_2 are the coefficients for the input features.

11.1.4 Target Vector y

The target vector y contains the observed values of the dependent variable.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

11.1.5 Objective: Minimizing the Cost Function

To find the optimal coefficients β, we minimize the error between the predicted values $\hat{y}$ and the actual values y. The error is measured using the sum of squared residuals (errors) called the **cost function** $J(\beta)$:

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{2}(y - X\beta)^T(y - X\beta)$$

Where:

- $(y - X\beta)$ is the residual vector.
- The factor 1/2 is to simplify the differentiation step.

11.1.6 Minimizing the Cost Function

To minimize the cost function, we take the derivative with respect to β, set it to zero, and solve for β:

$$\frac{\partial J(\beta)}{\partial \beta} = -X^T(y - X\beta)$$

Set the derivative equal to zero:

$$X^T(y - X\beta) = 0$$

Simplifying:

$$X^T y = X^T X\beta$$

Now, solve for β by multiplying both sides by $(X^T X)^{-1}$ (assuming $X^T X$ is invertible):

$$\beta = (X^T X)^{-1} X^T y$$

This is the **closed-form solution** for linear regression, also known as the **normal equation**.

11.1.7 Making Predictions

Once β is computed, predictions can be made for the target variable y using:

$$\hat{y} = X\beta$$

11.1.8 Assumptions of Linear Regression

For the linear regression model to be meaningful, certain assumptions are typically made:

1. **Linearity**: The relationship between the input features and the target variable is linear.
2. **Independence**: The residuals (errors) are independent.
3. **Homoscedasticity**: The variance of residuals is constant across all observations.
4. **Normality of Errors**: The residuals follow a normal distribution (important for hypothesis testing and confidence intervals).

11.2 6. Example in R

Here's an example in R of computing β using the closed-form solution and making predictions:

```
import numpy as np

# Sample Data (X and y)
X = np.array([[1, 1, 4],   # Design matrix (including intercept column of 1's)
              [1, 2, 5],
              [1, 3, 6]])

y = np.array([5, 7, 9])  # Actual target values

# Compute the coefficients using the Normal Equation with pseudo-inverse
X_transpose = X.T  # Transpose of X
X_transpose_X = X_transpose.dot(X)  # X^T X
```

```
X_transpose_y = X_transpose.dot(y)  # X^T y

# Use the pseudo-inverse in case X^T X is singular
beta = np.linalg.pinv(X_transpose_X).dot(X_transpose_y)

# Display the coefficients (beta values)
print("Coefficients (beta):", beta)

# Make predictions
y_hat = X.dot(beta)  # Predicted target values
print("Predicted values (y_hat):", y_hat)
```

```
Coefficients (beta): [3.55271368e-15 1.00000000e+00 1.00000000e+00]
Predicted values (y_hat): [5. 7. 9.]
```

11.3 Markov Chains

A **Markov Chain** is a mathematical model that describes a system undergoing transitions from one state to another, where the probability of moving to the next state depends only on the current state (not past states). This property is called the **Markov property**.

In the context of **Linear Algebra**, Markov Chains can be analyzed using matrices, particularly the **transition matrix**, to understand how the system evolves over time.

11.3.1 State Vectors

In a Markov Chain, the system's state at any given time is represented by a **state vector**. This vector consists of probabilities of being in each possible state.

For example, if a system has two states, **Hujan** (H) and **Cerah** (C), the state vector $\mathbf{x}$ could be:

$$\mathbf{x} = \begin{pmatrix} p(H) \\ p(C) \end{pmatrix}$$

Where $p(H)$ is the probability of the system being in state H, and $p(C)$ is the probability of the system being in state C.

11.3.2 Transition Matrix

The **transition matrix** P describes the probabilities of transitioning between states in the system. It is a square matrix where the element P_{ij} represents the probability of transitioning from state i to state j.

For a two-state system with Hujan and Cerah, the transition matrix might look like:

$$P = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$$

In this example:

- The probability of staying in state H (Hujan to Hujan) is 0.7.
- The probability of transitioning from Hujan to Cerah is 0.3.
- The probability of transitioning from Cerah to Hujan is 0.4.
- The probability of staying in state C (Cerah to Cerah) is 0.6.

11.3.3 Matrix Multiplication

To compute the state of the system at the next time step, you multiply the current state vector by the transition matrix.

If the current state vector is $\mathbf{x}_t$, the state vector at the next time step, $\mathbf{x}_{t+1}$, is given by:

$$\mathbf{x}_{t+1} = P \cdot \mathbf{x}_t$$

For example, if $\mathbf{x}_t = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$, then:

$$\mathbf{x}_{t+1} = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

This results in:

$$\mathbf{x}_{t+1} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

11.3.4 Steady State

A crucial concept in Markov Chains is the **steady state** or **stationary distribution**, where the system reaches a point where the state probabilities no longer change over time.

Mathematically, the steady state vector $\mathbf{x}$ satisfies the equation:

$$\mathbf{x} = P \cdot \mathbf{x}$$

To find the steady state, you need to solve for the eigenvector corresponding to eigenvalue $\lambda = 1$ of the transition matrix. The steady-state vector is the distribution where the system remains unchanged after one application of the transition matrix.

11.3.5 Eigenvectors and Eigenvalues

The steady state of a Markov Chain can be determined by finding the **eigenvector** corresponding to the eigenvalue 1 of the transition matrix P, since at steady state the state vector doesn't change when multiplied by the transition matrix.

To summarize, in a Markov Chain:

- The **transition matrix** P describes the system's transition probabilities.

- The **state vector x** updates over time by multiplying it by the transition matrix.
- The **steady state** vector **x** is the eigenvector associated with eigenvalue 1, representing the system's long-term probabilities of being in each state.

11.3.6 Example Problem: Finding Steady State

Let's take the transition matrix:

$$P = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$$

To find the steady-state vector, solve:

$$\mathbf{x} = P \cdot \mathbf{x}$$

which translates to solving the system of equations to find the vector **x** that does not change after multiplication with the matrix P.

Markov Chains in **Linear Algebra** make use of key concepts such as matrices, vectors, and eigenvalues to model systems that evolve probabilistically. By applying matrix operations and finding eigenvectors corresponding to eigenvalue 1, we can describe long-term behavior and steady states in such systems.

11.4 SVD Applications

11.5 Eigenvalues in Systems

11.6 Matrix Factorization

11.7 Neural Network Weights

11.8 Simulation with Matrices

Chapter 12

Dimensionality Analysis

12.1 Introduction

12.2 Engineering Applications

12.3 Dimensionality in Data Science

12.4 Reduction Techniques

12.5 Forecasting Applications

12.6 Model Consistency

12.7 Impact Evaluation

12.8 Challenges

Epilogue

“Begin at the beginning”, the King said, very gravely, “and go on till you come to the end: then stop” – Lewis Carroll

References

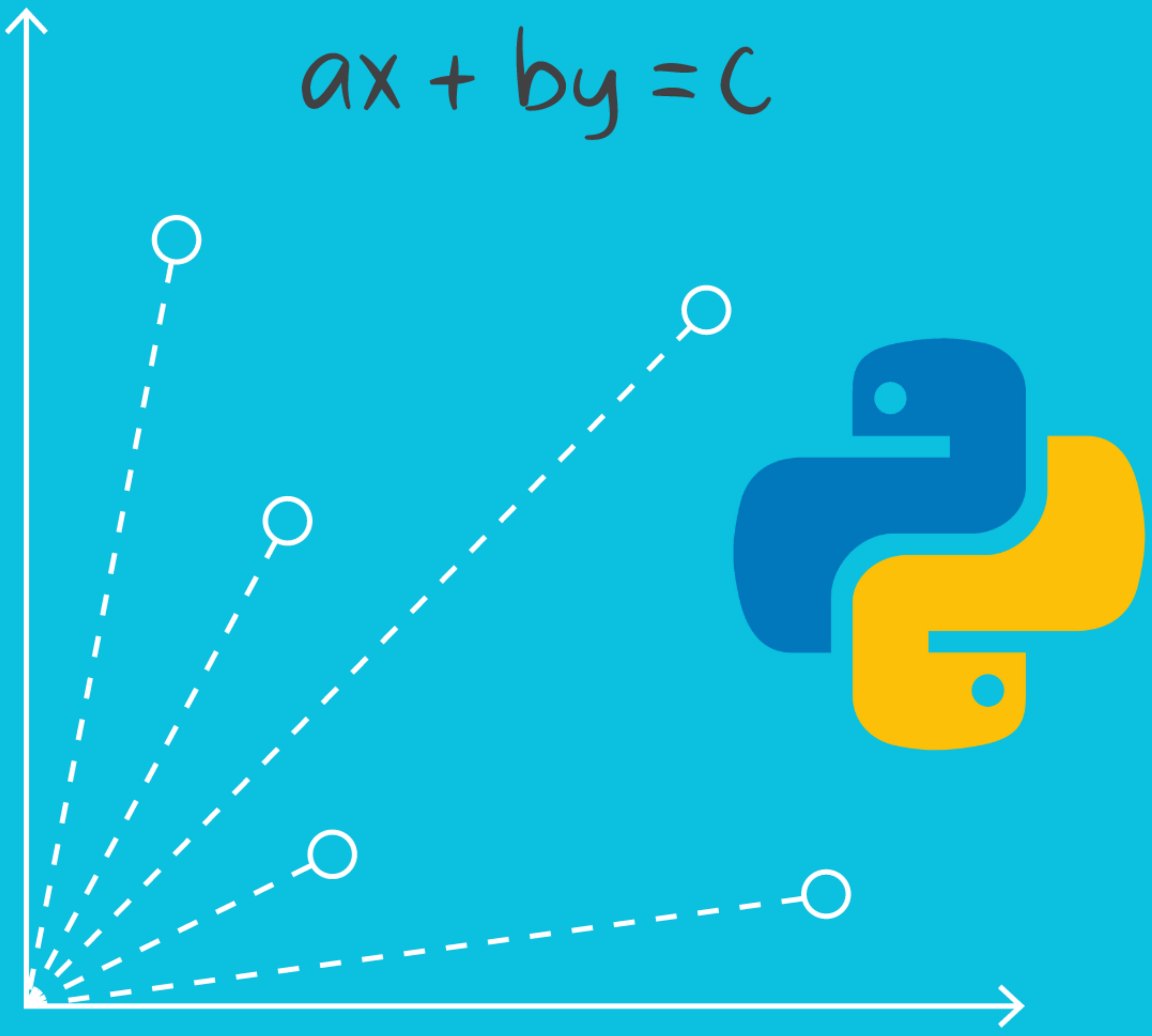
Lay, D. C. (2012). *Linear algebra and its applications* (4th ed.). Pearson.

Lay, D. C. (2012). *Linear algebra and its applications* (4th ed.). Pearson.

Eco Industry
Oriented University

LINEAR ALGEBRA

with Python



Writer:

Bakti Siregar, M.Sc., CDS.

First Edition