

**VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS**

**Grade A++ Accredited Institution by NAAC**

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

**SCHOOL OF ENGINEERING & TECHNOLOGY**

**B.Tech. Programme:** CSE - B

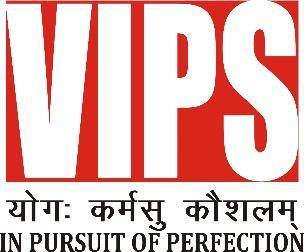
**Course Title:** Statistics, Statistical Modeling & Data Analytics

**Course Code: DA-304P**

**Submitted By Name:** ISHAAN JAIN

**Enrolment No:** 06117702722

**Class and Section:** CSE-B

****

**VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS**

**Grade A++ Accredited Institution by NAAC**

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

**VISION OF INSTITUTE**

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

**MISSION OF INSTITUTE**

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building  
capabilities to enable them to make significant contributions to the world.

**INDEX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.no** | **EXPERIMENT** | **Date** | **Marks** | | | **Remark** | **Updated Marks** | **Faculty Signature** |
| **Laboratory Assessment** | **Class Participation** | **Viva** |
| **1.** | Exercises to implement the basic matrix operations in Scilab. |  |  |  |  |  |  |  |
| **2.** | Exercises to find the Eigenvalues and Eigenvectors in Scilab. |  |  |  |  |  |  |  |
| **3.** | Exercises to solve equations by Gauss elimination, Gauss Jordan Method and Gauss Seidel in Scilab. |  |  |  |  |  |  |  |
| **4.** | Exercises to implement the associative, commutative and distributive property in a matrix in Scilab. |  |  |  |  |  |  |  |
| **5.** | Exercises to find the reduced row echelon form of a matrix in Scilab. |  |  |  |  |  |  |  |
| **6.** | Exercises to plot the functions and to find its first and second derivatives in Scilab. |  |  |  |  |  |  |  |

**INDEX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.no** | **EXPERIMENT** | **Date** | **Marks** | | | **Remark** | **Updated Marks** | **Faculty Signature** |
| **Laboratory Assessment** | **Class Participation** | **Viva** |
| **7.** | Exercises to present the data as a frequency table in SPSS. |  |  |  |  |  |  |  |
| **8.** | Exercises to find the outliers in a dataset in SPSS. |  |  |  |  |  |  |  |
| **9.** | Exercises to find the most-risky project out of two mutually exclusive projects in SPSS. |  |  |  |  |  |  |  |
| **10.** | Exercises to draw a scatter diagram, residual plots, outliers leverage and influential data points in R. |  |  |  |  |  |  |  |
| **11.** | Exercises to calculate correlation using R. |  |  |  |  |  |  |  |
| **12.** | Exercises to implement Time series Analysis using R. |  |  |  |  |  |  |  |
| **13.** | Exercises to perform Linear Regression using R and ggplot2. |  |  |  |  |  |  |  |

**EXPERIMENT NO - 01**

**Aim of the program: Implement the basic matrix operations in Scilab**

**Theory:**

**Introduction to Matrices**

A **matrix** is a rectangular array of numbers arranged in rows and columns. It is a fundamental mathematical structure used in various applications, including engineering, physics, computer science, and data science.

A matrix with **m** rows and **n** columns is called an **m × n** matrix and is generally represented as:

A= [a11 a12 … a1n

a21 a22 … a2n

⋮ ⋮ ⋱ ⋮

am1 am2 … amn]

**Basic Matrix Operations**

1. **Matrix Addition**
   * If two matrices AAA and BBB have the same dimensions, their sum is given by:

C=A+B

* + Each element of the resulting matrix is the sum of corresponding elements:

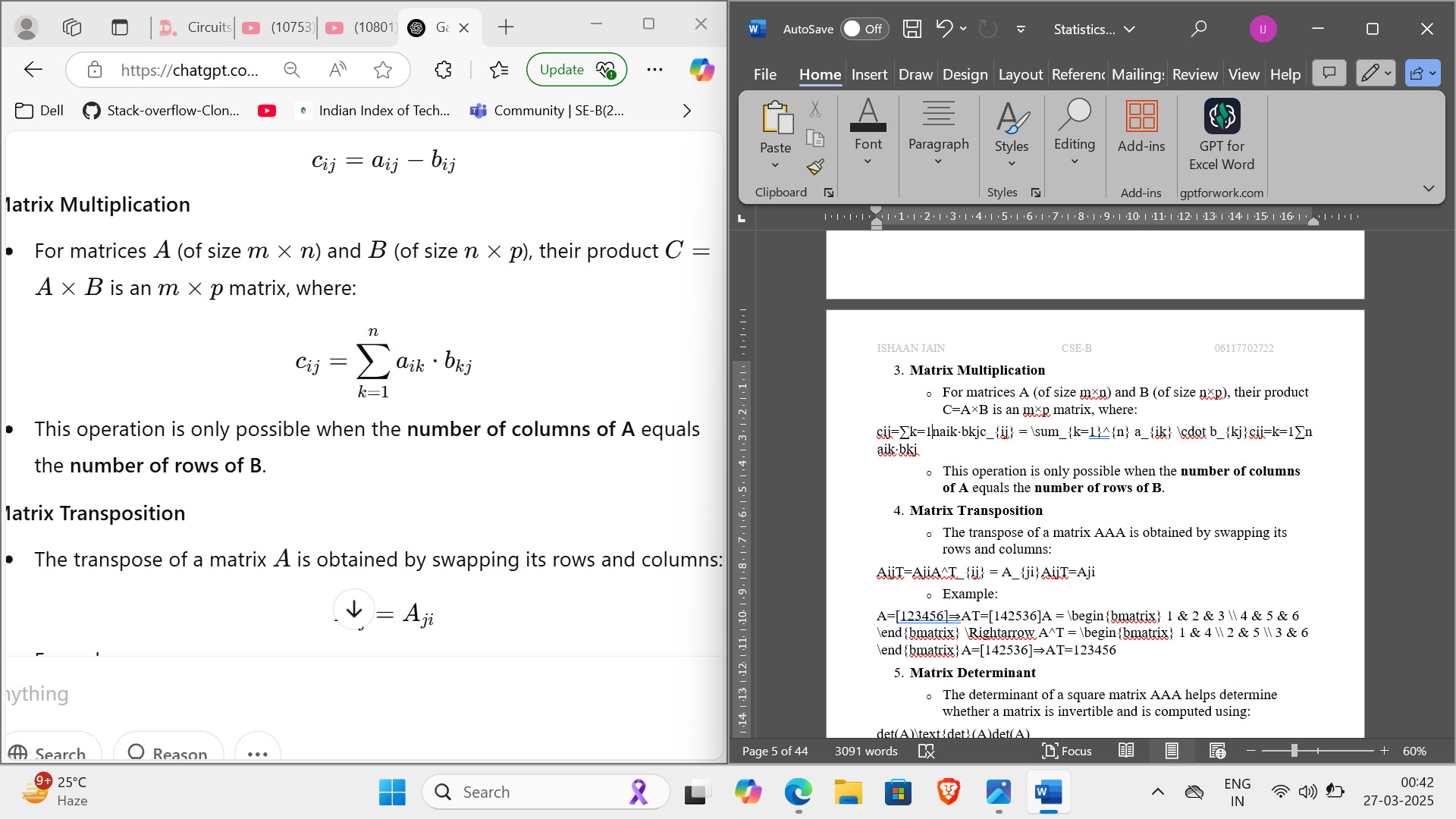
cij=aij+bij

1. **Matrix Subtraction**
   * The difference of two matrices of the same size is:

C=A−B

Each element is computed as:

cij=aij−bij

1. **Matrix Multiplication**
   * For matrices A (of size m×n) and B (of size n×p), their product C=A×B is an m×p matrix, where:
   * This operation is only possible when the **number of columns of A** equals the **number of rows of B**.
2. **Matrix Transposition**
   * The transpose of a matrix A is obtained by swapping its rows and columns:

ATij=Aji ​

* + Example:

A=[123] ⇒AT=[ 14 ]

[456] [ 25 ]

[ 36 ] ​​

1. **Matrix Determinant**
   * The determinant of a square matrix A helps determine whether a matrix is invertible and is computed using:

det(A)

* + Example for a 2×2 matrix:

A= [ab ⇒det(A)=(a×d)−(b×c)

cd]

1. **Matrix Inversion**
   * The inverse of a square matrix A (if it exists) is denoted by A^−1 and satisfies:

A×A^−1=I

* + The inverse is computed using:

A−1=(1/det(A))⋅adj(A)

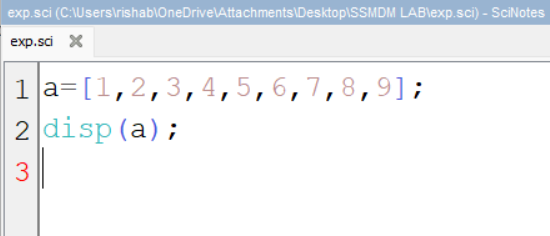
* + A matrix must have a nonzero determinant to be invertible.

**Applications of Matrix Operations**

* **Computer graphics:** Used for transformations (rotation, scaling, translation).
* **Engineering & Physics:** Solving systems of equations, representing networks.
* **Machine Learning & AI:** Neural networks, data representation.
* **Cryptography:** Encryption and decryption algorithms.
* **Economics & Statistics:** Modelling relationships between variables.

**Source Code:**

1. **Matrix creation**

****

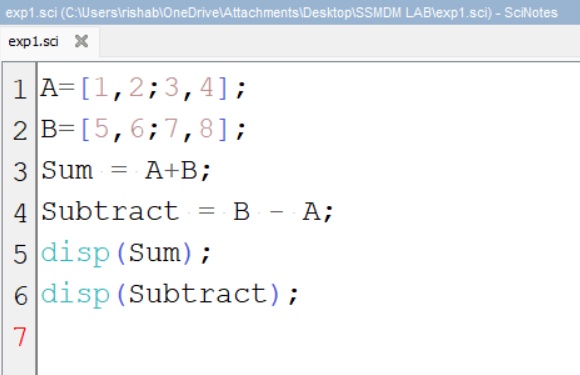
**Output:**

**A= 1 2 3**

**4 5 6**

**7 8 9**

1. **Matrices addition and Subtraction**

****

**Output:**

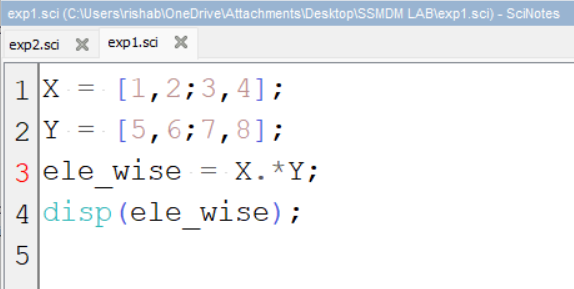
**Sum = 6 8**

**10 12**

**Difference= -4 -4**

**-4 -4**

1. **Element-wise dot product of Matrices**

****

**Output:**

**“Element-wise Multiplication”: 5 12**

**21 32**

**Learning Outcome:**

1. Understand basic matrix operations and their properties.
2. Perform addition, subtraction, multiplication, transposition, determinant, and inversion in Scilab.
3. Apply matrix concepts to solve real-world problems.
4. Develop Scilab programs to automate matrix computations.

**EXPERIMENT NO - 02**

**Aim of the program: Exercises to implement the associations, commutative and distributive property in a matrix in Scilab**

**Theory:**

**1. Associative Property**

**The associative property states that the grouping of matrices does not affect the result in both addition and multiplication.**

* **Matrix Addition:**

**(A+B)+C=A+(B+C)**

* **Matrix Multiplication:**

**(A×B)×C=A×(B×C**

**In matrix operations, parentheses can be rearranged without changing the outcome.**

**2. Commutative Property**

**The commutative property states that the order of operations does not affect the result only in addition, but not always in multiplication.**

* **Matrix Addition (Commutative):**

**A+B=B+A**

**Matrix Multiplication (Not Commutative in General):**

**A×B≠B×A**

**Matrix multiplication is NOT always commutative unless the matrices are diagonal, identity, or special cases.**

**3. Distributive Property**

**The distributive property states that matrix multiplication distributes over addition.**

* **Left Distributive Law:**

**A×(B+C)=A×B+A×C**

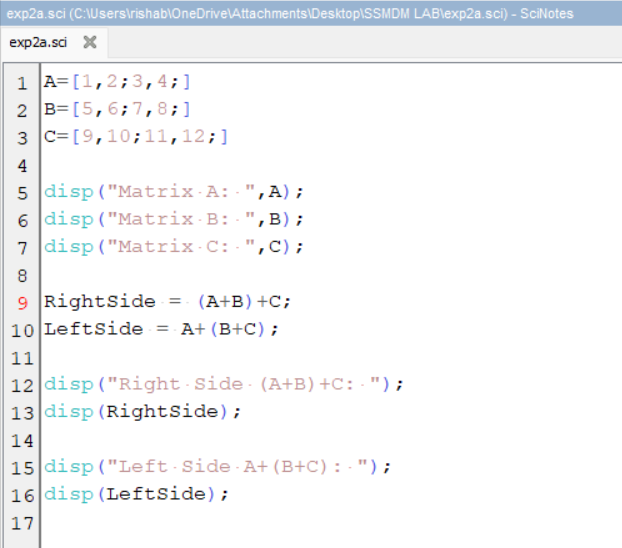
* **Right Distributive Law:**

**(A+B)×C=A×C+B×C**

**This means that multiplying a matrix by a sum of matrices gives the same result as multiplying each matrix separately and then adding the results.**

**Source Code:s**

1. **Associative property**

****

**Output:**

**Matrix A = 1 2**

**3 4**

**Matrix B = 5 6**

**7 8**

**Matrix C = 9 0**

**1 2**

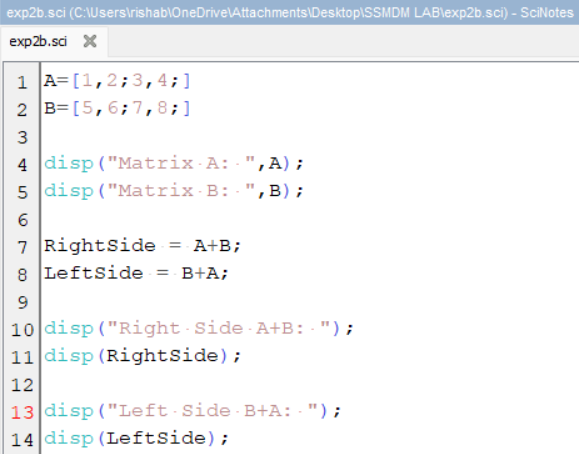
**Left Side(A+B)+C = 15 8**

**11 14**

**Rights SideA+(B+C) = 15 8**

**11 14**

1. **Commutative Property**

****

**Output:**

**Matrix A = 1 2**

**3 4**

**Matrix B = 5 6**

**7 8**

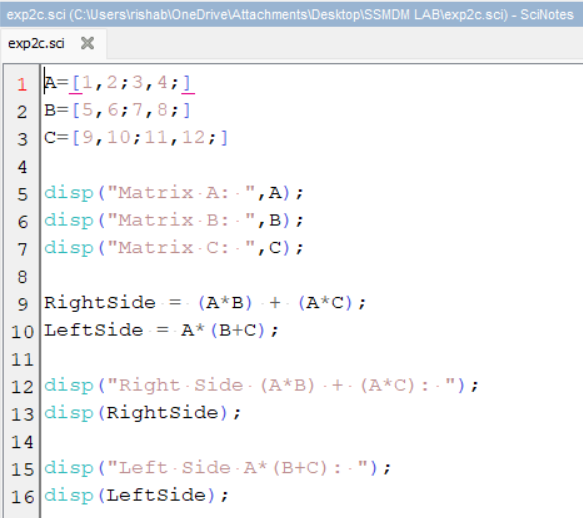
**Left Side(A+B) = 6 8**

**10 12**

**Rights Side(B+A) = 6 8**

**10 12**

1. **Distributive Property**

****

**Output:**

**Matrix A = 1 2**

**3 4**

**Matrix B = 5 6**

**7 8**

**Matrix C = 9 0**

**1 2**

**Left SideA\*(B+C)= 30 26**

**74 58**

**Rights Side(A\*B) + (A\*C) = 30 26**

**74 58**

**Learning Outcome:**

1. Understand the associative, commutative, and distributive properties in matrix algebra.  
2. Implement these properties using Scilab.  
3. Learn that matrix multiplication is not always commutative, unlike scalar multiplication.

**EXPERIMENT NO - 03**

**PROGRAM 01**

**Aim of the program: Write a program to solve equations by Gauss Elimination.**

**2x +3y+1z=9**

**4x +1y-2z=2**

**2x-3y+1z=-3**

**Theory:**

The Gauss Elimination Method is a systematic approach used to solve a system of linear equations by converting it into an upper triangular form and then performing back-substitution to find the values of unknowns.

**This method consists of the following steps:**

1. **Formation of Augmented Matrix:**
   * The system of linear equations is written in matrix form as AX = B, where A is the coefficient matrix, X is the unknown variable matrix, and B is the constant matrix.
   * The augmented matrix [A | B] is constructed by appending the constant matrix B to A.
2. **Forward Elimination:**
   * The augmented matrix is transformed into an upper triangular matrix by applying row operations such that all elements below the main diagonal become zero.
   * This is achieved by dividing the pivot row by the leading coefficient (diagonal element) and using it to eliminate the corresponding elements in subsequent rows.
3. **Backward Substitution:**
   * Once the system is in upper triangular form, the values of the unknowns are found starting from the last equation and substituting the known values into previous equations.
   * This step ensures that all variables are solved sequentially from bottom to top.

**The Gauss Elimination Method is widely used due to its efficiency in solving large systems of equations and is fundamental in numerical linear algebra.**

**Source Code:**

A = [2,3,1; 4,1,-2; 2,-3,1];

printf("Coefficients of A:\n");

disp(A);

B = [9; 2; -3];

printf("Constant Matrix:\n");

disp(B);

C = [A, B];

printf("Augmented Matrix:\n");

disp(C);

n = 3;

for i = 1:n

C(i, :) = C(i, :) / C(i, i);

printf("C-Matrix:\n");

disp(C);

for j = 1:n-1

if i + j<n+1

C(j+I,:) = C(j+i, :) - C(j+i, i) \* C(i, :);

end

end

end

printf("C-Matrix after elimination:\n");

disp(C);

printf("Solution:\n");

z = C(3, 4);

y = C(2, 4) - C(2, 3) \* z;

x = C(1, 4) - C(1, 3) \* z - C(1, 2) \* y;

printf("x = ");

disp(x);

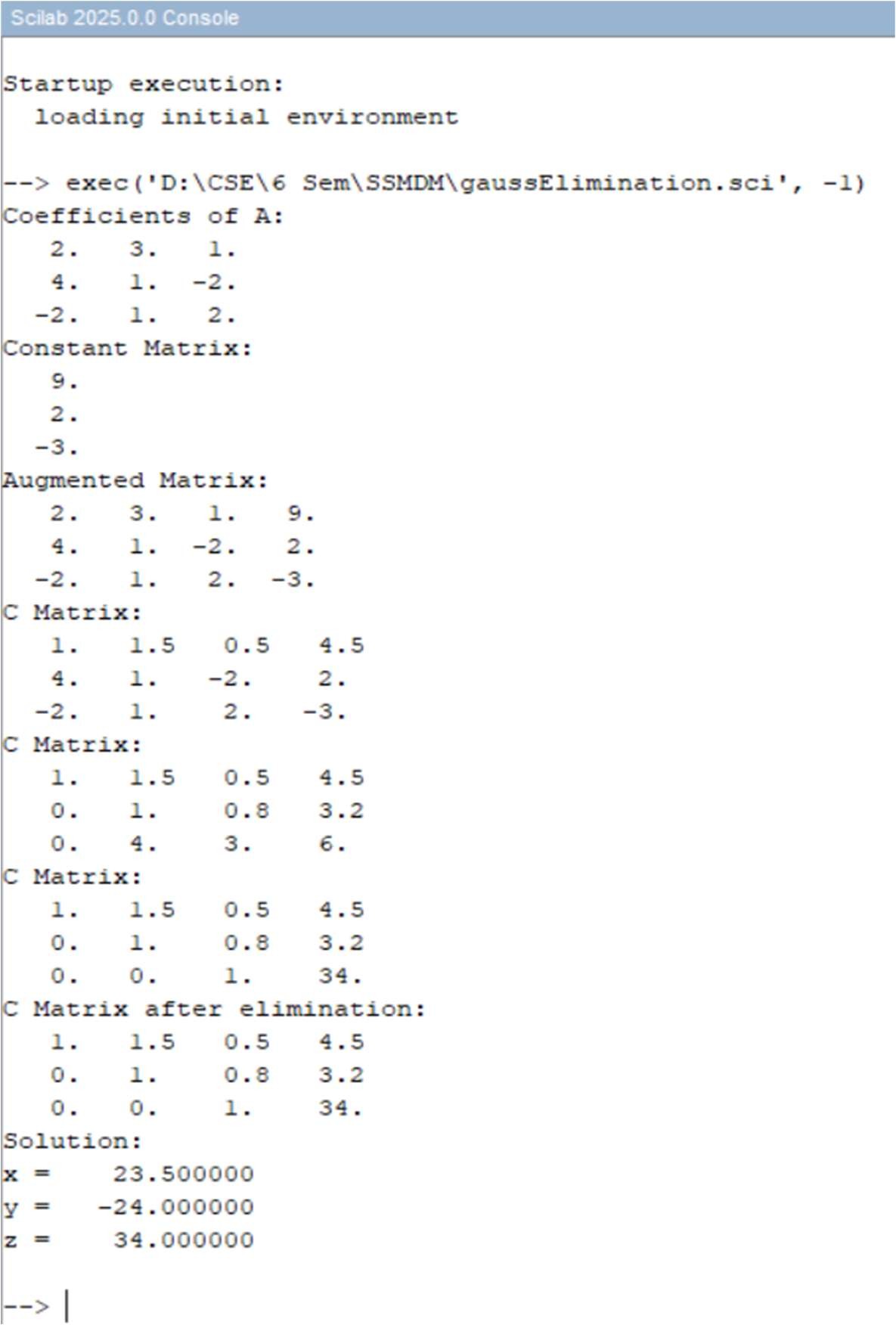
printf("y = ");

disp(y);

printf("z = ");

disp(z);

**Output:**

****

**Learning Outcome:**

1. Understand the mathematical formulation of the Gauss Elimination Method.
2. Learn how to convert a system of equations into an augmented matrix.
3. Apply row operations to achieve an upper triangular matrix.
4. Implement forward elimination and backward substitution to solve linear equations.
5. Develop skills in coding numerical methods using programming languages.
6. Analyze the importance of Gauss Elimination in real-world applications such as engineering and scientific computing.

**PROGRAM 02**

**Aim of the program: Write a program to solve equations by Gauss Jordan Method.**

**1x +2y+1z=6**

**2x +3y+3z=14**

**1y+2z=8**

**Theory:**

The Gauss-Jordan Method is an extension of the Gauss Elimination Method used to solve a system of linear equations. It transforms the given system into reduced row echelon form (RREF), where the coefficient matrix becomes an identity matrix.

This method consists of the following steps:

1. **Formation of Augmented Matrix:**
   * The system of equations is represented in the matrix form AX = B, where A is the coefficient matrix, X is the unknown variable matrix, and B is the constant matrix.
   * The augmented matrix [A | B] is constructed by appending B to A.
2. **Pivoting and Row Operations:**
   * The pivot element (diagonal element) is made 1 by dividing the entire row by the pivot value.
   * The other elements in the pivot column are eliminated by using row operations so that all elements above and below the pivot become 0.
3. **Reduction to Identity Matrix:**
   * The process is repeated for all rows until the coefficient matrix is converted into an identity matrix.
   * The final matrix will have the solution values directly in the last column.

Unlike the Gauss Elimination method, the Gauss-Jordan Method eliminates the need for backward substitution, making it more efficient for solving equations and for computing matrix inverses.

**Source Code:**

A = [1,2,1; 2,3,3; 0,1,2];

printf("Coefficients of A:\n");

disp(A);

B = [6;14;8];

printf("Constant Matrix:\n");

disp(B);

C = [A,B];

printf("Augmented Matrix:\n");

disp(C);

n = 3;

for i = 1:n

C(i,:) = C(i,:) / C(i,i);

printf("After making pivot element 1 (Row: %d):\n", i);

disp(C);

for j = 1:n

if i ~= j then C(j,:) = C(j,:) - C(j,i) \* C(i,:);

end

end

printf("Matrix After eliminating column %d:\n", i);

disp(C);

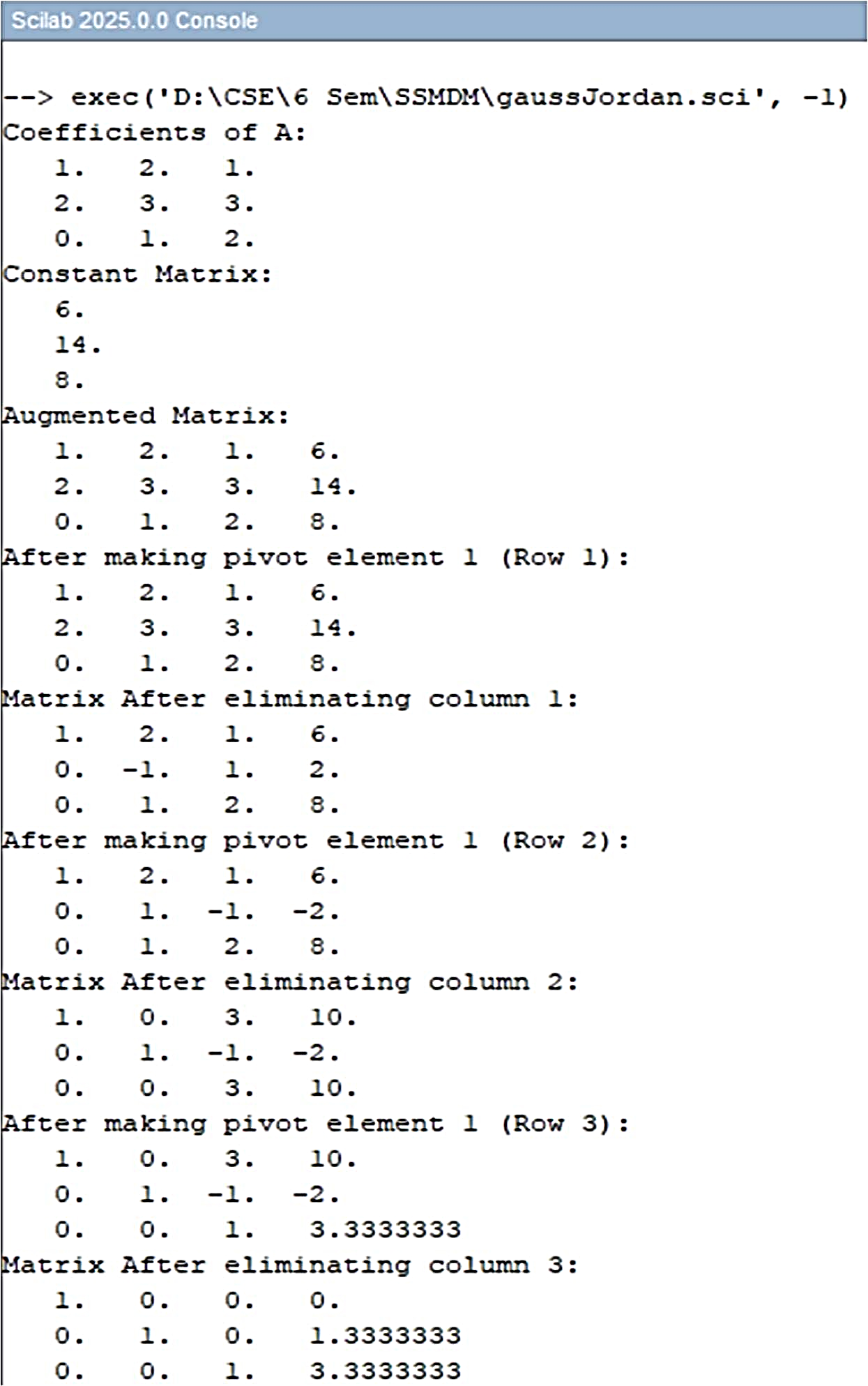
end

x = C(:, $);

printf("Solution:\n");

disp(x);

**Output:**



**Learning Outcome:**

1. Understand the concept of **row operations** in solving linear equations.
2. Learn the **Gauss-Jordan Method** and how it transforms a matrix into **row echelon form**.
3. Implement **pivoting** and **column elimination** to convert a coefficient matrix into an identity matrix.
4. Gain hands-on experience in writing a **numerical computation program**.
5. Analyze the advantages of the **Gauss-Jordan Method** over **Gauss Elimination**, especially in finding matrix inverses.
6. Apply the method to real-world engineering and scientific problems.

**PROGRAM 03**

**Aim of the program: Write a program to solve equations by Gauss Seidel Method in Scilab.**

**4x​−x​+0z=2​**

**−x​+4y​−z=6**

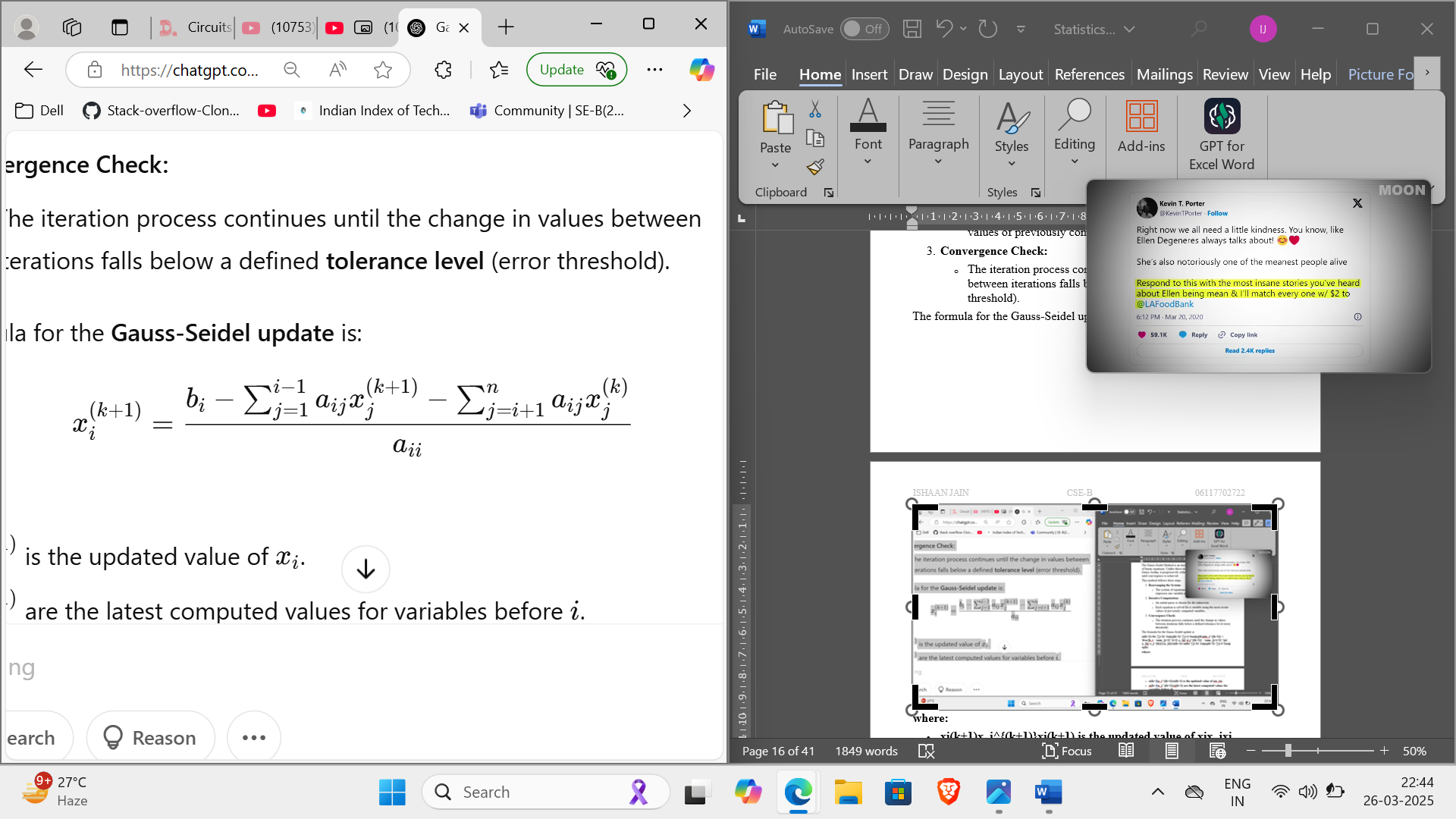
**​0x​−y​+4z​​=2**

**Theory:**

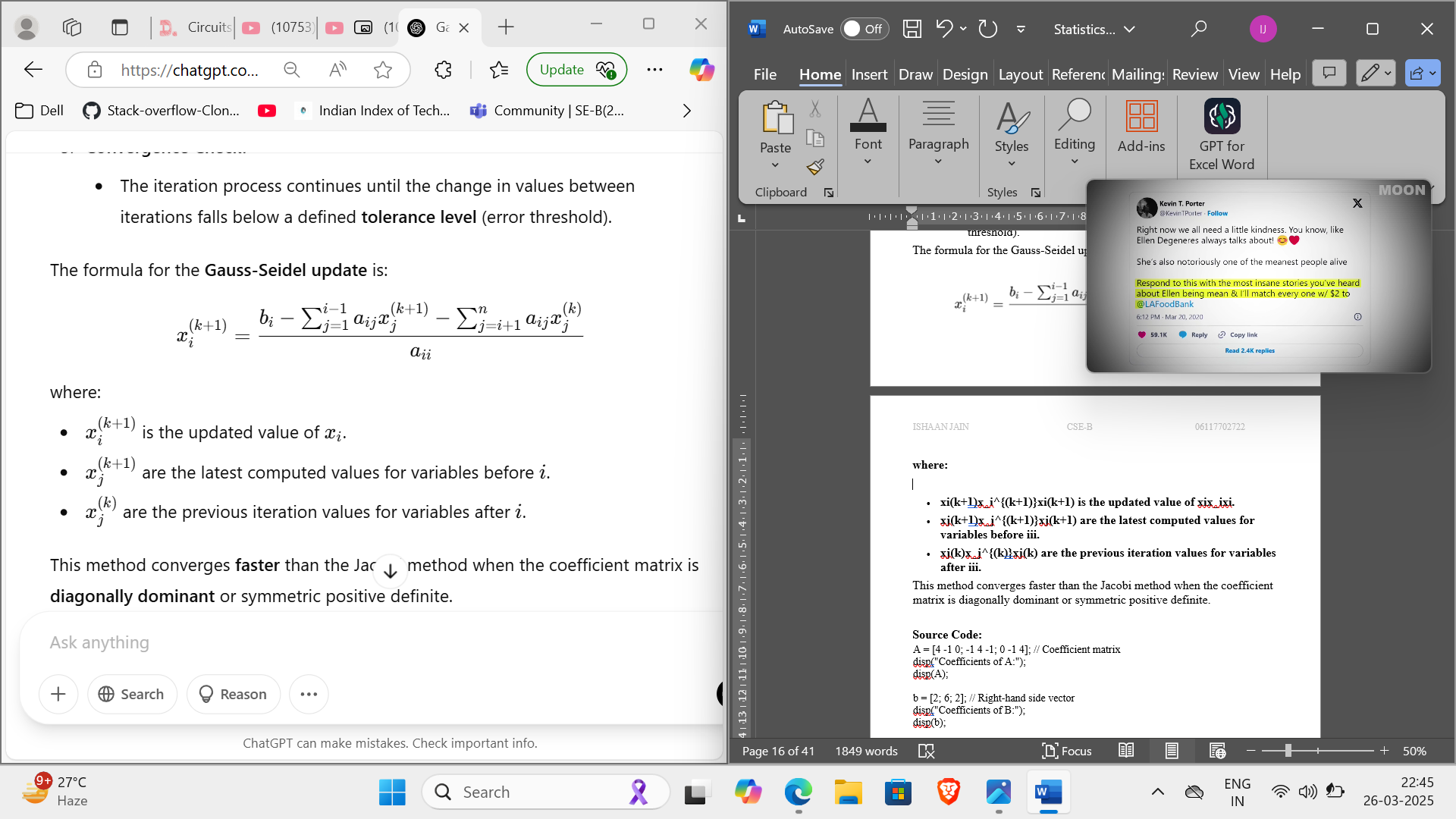
The Gauss-Seidel Method is an iterative technique used to solve a system of linear equations. Unlike direct methods like Gauss Elimination and Gauss-Jordan, it progressively refines the solution using an initial guess until convergence is achieved.

This method follows these steps:

1. **Rearranging the System:**
   * The system of equations is rewritten such that each equation expresses one variable in terms of the others.
2. **Iterative Computation:**
   * An initial guess is chosen for the unknowns.
   * Each equation is solved for a variable using the most recent values of previously computed variables.
3. **Convergence Check:**
   * The iteration process continues until the change in values between iterations falls below a defined tolerance level (error threshold).

The formula for the Gauss-Seidel update is:

**where:**



This method converges faster than the Jacobi method when the coefficient matrix is diagonally dominant or symmetric positive definite.

**Source Code:**

A = [4 -1 0; -1 4 -1; 0 -1 4]; // Coefficient matrix

disp("Coefficients of A:");

disp(A);

b = [2; 6; 2]; // Right-hand side vector

disp("Coefficients of B:");

disp(b);

x = [0; 0; 0]; // Initial guess

tol = 1e-6; // Tolerance

max\_iter = 100; // Maximum iterations

n = size(A, 1); // Number of equations

for k = 1:max\_iter

x\_old = x;

for i = 1:n

sum1 = A(i, 1:i-1) \* x(1:i-1); // Using updated values

sum2 = A(i, i+1:n) \* x\_old(i+1:n); // Using old values

x(i) = (b(i) - sum1 - sum2) / A(i, i);

end

if norm(x - x\_old, "inf") < tol // Convergence check

break;

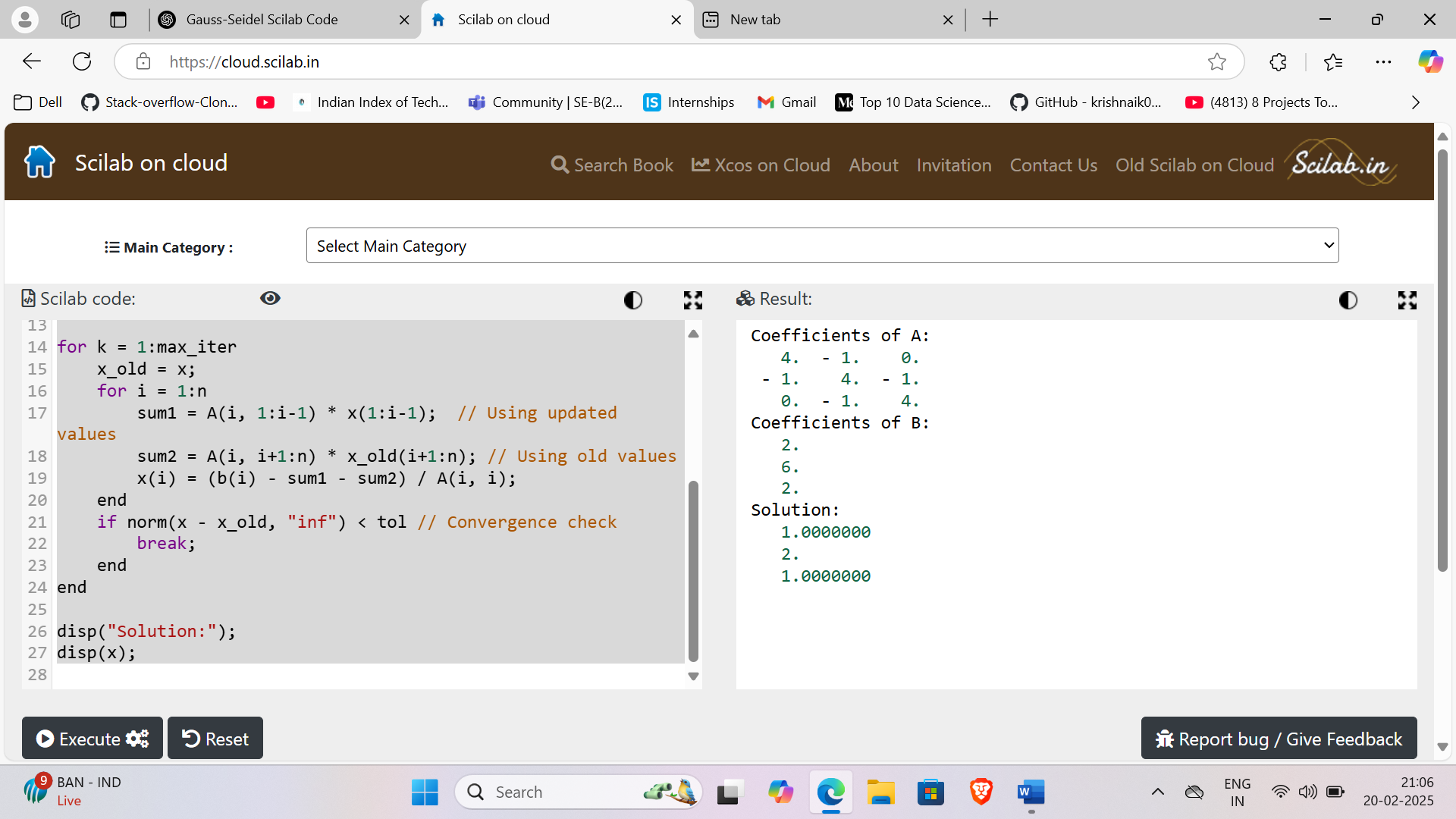
end

end

disp("Solution:");

disp(x);

**Output:**



**Learning Outcome:**

1. Understand the **Gauss-Seidel iterative method** for solving linear systems.
2. Learn how to implement **iterative numerical methods** in **Scilab**.
3. Compare the **efficiency and convergence** of iterative methods vs. direct methods like Gauss-Jordan.
4. Understand the importance of **diagonal dominance** for the convergence of the Gauss-Seidel method.
5. Develop problem-solving skills by analyzing the **rate of convergence** and adjusting tolerance levels.
6. Apply iterative methods to **real-world engineering problems**, such as solving large systems in structural analysis and electrical circuit simulations.

**EXPERIMENT NO - 04**

**Aim of the program: Write a program to find the Eigenvalues and Eigenvectors in Scilab**

**]**

4 -2

1 -1

**[**

**A =**

**Theory:**

Eigenvalues and eigenvectors are fundamental concepts in linear algebra and are widely used in engineering, physics, and data science applications such as PCA (Principal Component Analysis), vibration analysis, and stability analysis.

**Eigenvalues and Eigenvectors**

For a given square matrix A, an eigenvalue λ and an eigenvector X satisfy the equation:

AX=λ

where:

* A is an n×n matrix,
* X is a nonzero vector (eigenvector),
* λ is a scalar (eigenvalue).

**Finding Eigenvalues and Eigenvectors**

1. Eigenvalues (λ) are found by solving the characteristic equation:

det(A−λI)=0

where I is the identity matrix.

1. Eigenvectors (X) are obtained by solving:

(A−λI)X=0

for each eigenvalue λ.

**Interpretation**

* Eigenvalues represent scaling factors by which an eigenvector is stretched or compressed during a transformation by matrix A.
* Eigenvectors indicate directions in which the transformation occurs.

Eigenvalues and eigenvectors have applications in:

* Machine Learning (PCA for dimensionality reduction)
* Structural Engineering (vibrations and modal analysis)
* Quantum Mechanics (Schrödinger's equation solutions)
* Markov Chains (steady-state probabilities)

**Source Code:**

// Define matrix A

A = [4 -2; 1 -1];

[eigen\_vectors, eigen\_values] = spec(A);

printf("Eigenvalues:\n");

disp(diag(eigen\_values));

for i = 1:size(eigen\_vectors, 2)

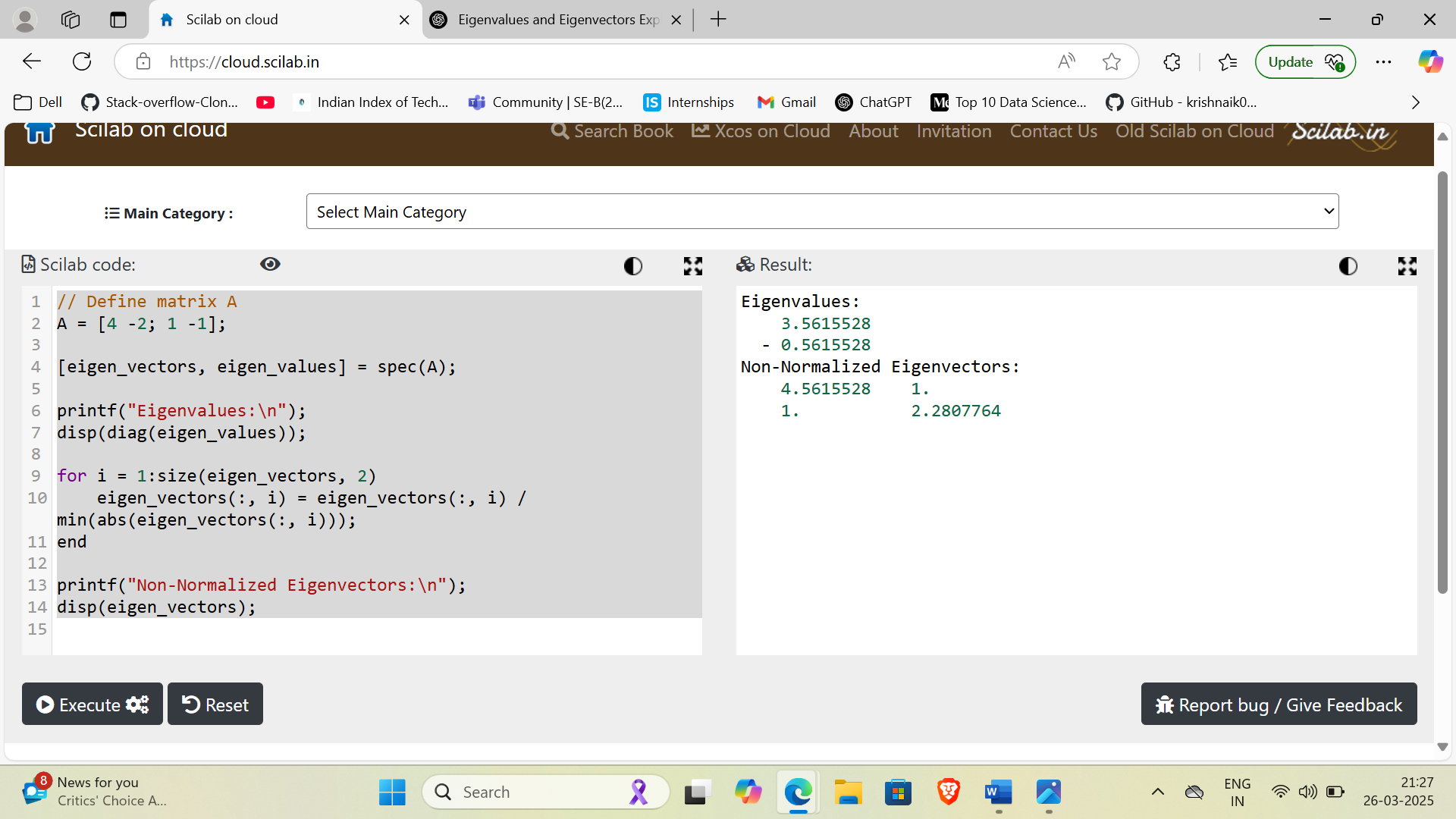
eigen\_vectors(:, i) = eigen\_vectors(:, i) / min(abs(eigen\_vectors(:, i)));

end

printf("Non-Normalized Eigenvectors:\n");

disp(eigen\_vectors);

**Output:**



**Learning Outcome:**

1. Understand the concepts of eigenvalues and eigenvectors and their significance in linear algebra.
2. Learn how to compute eigenvalues and eigenvectors in Scilab using built-in functions.
3. Gain insights into diagonalization and how matrices behave under transformations.
4. Analyze how eigenvalues help in stability and control systems analysis.
5. Develop coding skills in Scilab for matrix operations, useful in engineering and machine learning.

**EXPERIMENT NO - 05**

**Aim of the program: Write a program to find reduced row echelon form of a matrix in Scilab**

]

[

2 3 1

4 1 -2

**A =**

**Theory:**

**Row Echelon Form (REF)**

A matrix is said to be in **Row Echelon Form (REF)** if:

1. All nonzero rows are above any rows of all zeros.
2. Each leading (leftmost nonzero) entry of a row is to the right of the leading entry of the previous row.
3. The leading entry in each nonzero row is 1 (this is also known as a **pivot**).

**Reduced Row Echelon Form (RREF)**

A matrix is in **Reduced Row Echelon Form (RREF)** if it satisfies all the REF conditions and:

1. Each pivot is the only nonzero entry in its column.
2. Each leading 1 is **the only nonzero element** in its column.

**Purpose of RREF**

* **Solving linear equations** using the **Gaussian elimination** or **Gauss-Jordan elimination** method.
* **Finding rank** of a matrix.
* **Determining linear dependence** of a system.
* **Checking for inverse matrices** (if a square matrix reduces to the identity matrix in RREF, it is invertible).

The process of converting a matrix to **RREF** involves:

1. Making each pivot element **1** (by dividing the row by the pivot).
2. Eliminating other entries in the pivot column to make them **zero** using row operations.
3. Repeating the process for each row until the entire matrix is in RREF.

**Source Code:**

A=[2,3,1;4,1,-2;-2,1,2];

[n,m]=size(A);

if m~=n then

disp("Error: the matrix is not square");

else

disp("Original Matrix:");

disp(A);

for i = 1:n

A(i, :) = A(i, :) / A(i, i);

for j = 1:n

if i ~= j then

fact = A(j, i);

A(j, :) = A(j, :) - fact \* A(i, :);

end

end

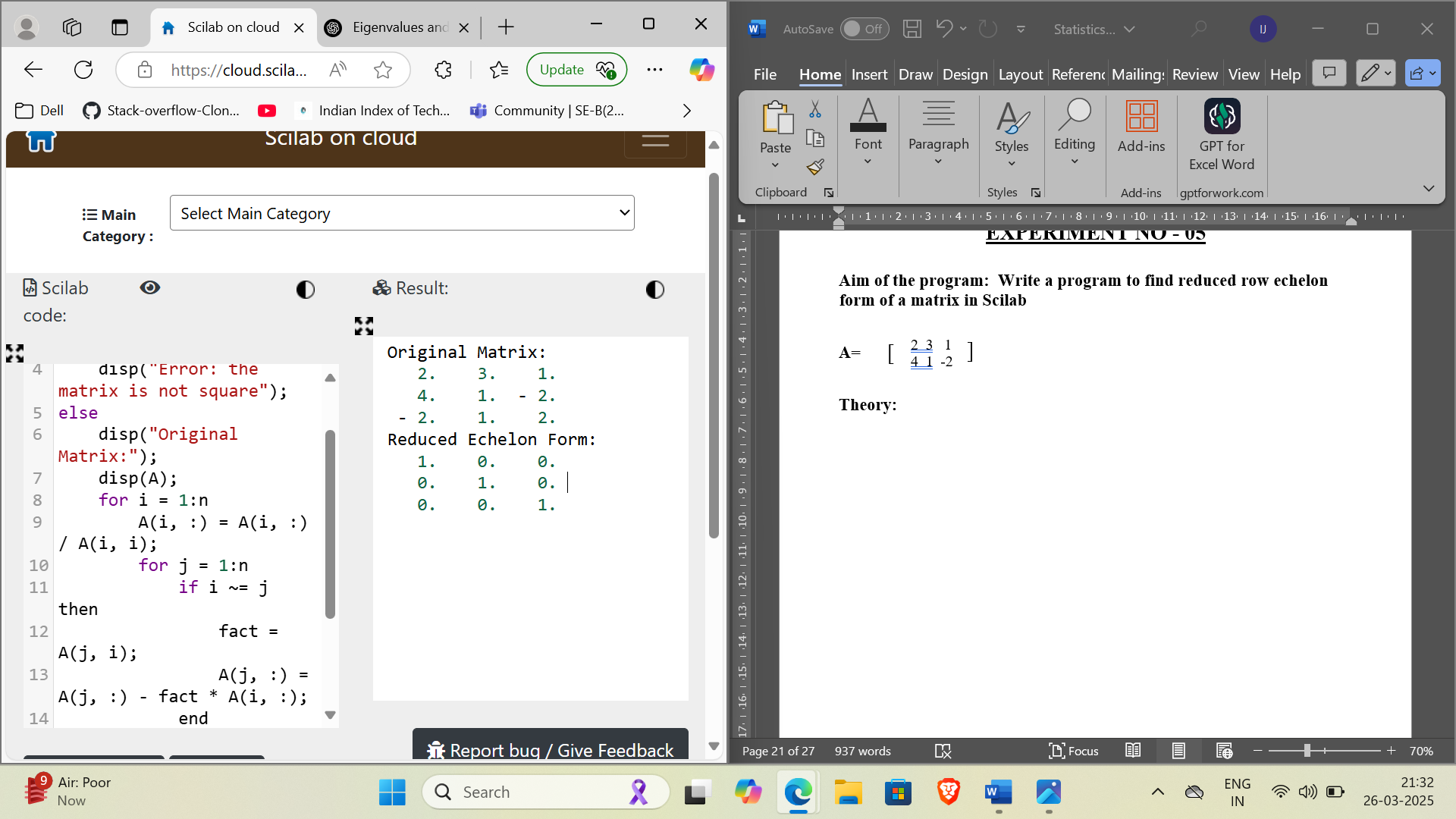
end

end

disp("Reduced Echelon Form:");

disp(A);

**Output:**



**Learning Outcome:**

1. Understand Gaussian elimination and Gauss-Jordan elimination methods.
2. Learn how to convert a matrix into Reduced Row Echelon Form (RREF).
3. Gain knowledge of row operations in matrix algebra.
4. Apply RREF to solve systems of linear equations.
5. Implement matrix manipulation techniques using Scilab.
6. Understand the significance of rank, independence, and consistency of a system of equations.

**EXPERIMENT NO - 06**

**Aim of the program: Write a program to plot the functions and to find its first and second derivatives in Scilab**

**y = x^3 - 5x.^2 + 2x + 10;**

**Theory:**

**Introduction**

A function describes a relationship between an input and output. **Differentiation** helps analyze how a function changes by determining its rate of change. In this experiment, we plot a given function and compute its **first and second derivatives** in Scilab.

**Concept of Differentiation**

1. **First Derivative (f'(x))**
   * Measures the rate of change of a function.
   * Represents the **slope of the tangent line** at any point.
   * Helps determine **increasing and decreasing** behavior of the function.
   * Formula:

f'(x) = dy/dx

* + For y=x^3−5x^2+2x+10, the first derivative is:

f′(x)=3x^2−10x+2

1. **Second Derivative (f''(x))**
   * Measures the rate of change of the first derivative.
   * Indicates whether the function is **concave up or concave down**.
   * Helps in finding points of **inflection**.
   * Formula:

f′′(x)=d^2y/dx^2 ​

* + For y=x^3−5x^2+2x+10 second derivative is:

f′′(x)=6x−10

**Numerical Differentiation in Scilab**

In Scilab, numerical differentiation is approximated using small increments h:

* **First Derivative Approximation** (Forward Difference Method):

f′(x)≈f(x+h)−f(x)

h

* **Second Derivative Approximation** (Central Difference Method):

f′′(x)≈f(x+h)−2f(x)+f(x−h)

h^2

where h is a small step size.

**Graphical Representation**

* The **original function f(x)** is plotted in **blue**.
* The **first derivative f′(x)** is plotted in **red**, showing the slope.
* The **second derivative f′′(x)** is plotted in **green**, indicating concavity.

**Source Code:**

function y = f(x)

y = x.^3 - 5\*x.^2 + 2\*x + 10;

endfunction

disp("Original Function: f(x) = x^3 - 5x^2 + 2x + 10");

f1\_x = "3\*x^2 - 10\*x + 2"; // First derivative

f2\_x = "6\*x - 10"; // Second derivative

disp("First Derivative: f1(x) = " + f1\_x);

disp("Second Derivative: f2(x) = " + f2\_x);

x = linspace(-2, 5, 100);

y = f(x);

h = 0.01;

df = (f(x + h) - f(x)) / h;

d2f = (f(x + h) - 2\*f(x) + f(x - h)) / (h^2);

figure();

plot(x, y, 'b', "LineWidth", 2);

plot(x, df, 'r', "LineWidth", 2);

plot(x, d2f, 'g', "LineWidth", 2);

xlabel("x-axis");

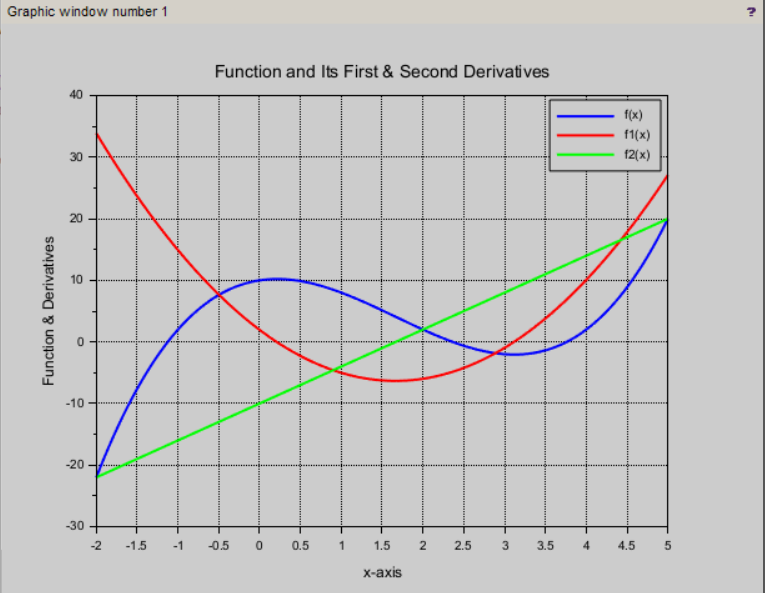
ylabel("Function & Derivatives");

title("Function and Its First & Second Derivatives");

legend("f(x)", "f1(x)", "f2(x)", "Location", "best");

xgrid();

**Output:**

****

**Learning Outcome:**

1. Understand the concept of differentiation and its significance in calculus.
2. Compute first and second derivatives of a function both analytically and numerically.
3. Use Scilab functions and plotting tools for mathematical analysis.
4. Interpret derivative graphs to understand the function’s behavior.
5. Learn about numerical differentiation methods for estimating derivatives.

**EXPERIMENT NO - 07**

**Aim of the program: Write a program to present the data as a frequency table in R**

**Theory:**

A frequency table is a statistical tool used to display the number of occurrences (frequency) of each unique value in a dataset. It helps summarize categorical or discrete data in a compact and understandable form. In this experiment, we use R, a powerful language for statistical computing, to create a frequency table from a given set of binary data representing gender:

* 0 indicates a **boy**
* 1 indicates a **girl**

The table() function in R is used to count how many times each value appears in the dataset. This function returns a table of frequencies that can be easily printed or used for further analysis. Frequency tables are essential in data analysis for understanding distribution patterns, drawing visualizations like bar plots or pie charts, and making informed decisions based on data.

Top of Form

Bottom of Form

**Source Code:**

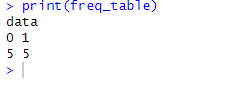
data = c(0,1,0,1,0,1,0,1,0,1)

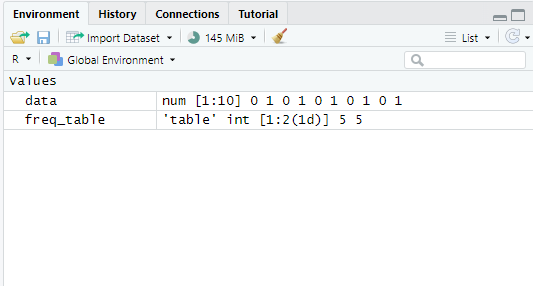
# 0 represent boy, 1 represents girl. This is data of 10 students.

freq\_table = table(data)

print(freq\_table)

**Output:**





**Learning Outcome:**

* Understood the concept and use of frequency tables in statistical analysis.
* Learned how to represent categorical data in tabular form using R programming.
* Gained hands-on experience using the table() function in R to compute frequency distributions.
* Improved ability to interpret raw data and summarize it effectively.

**EXPERIMENT NO - 08**

**Aim of the program: Write a program to find the outliers in a dataset in R**

**Theory:**

Outliers are data points that significantly differ from the rest of the dataset. They may arise due to variability in measurement, data entry errors, or unique experimental conditions. Identifying outliers is crucial because they can skew the results of data analysis and lead to misleading interpretations.

In R, outliers are often detected using **boxplots**, which graphically depict the distribution of data based on five summary statistics: minimum, first quartile (Q1), median, third quartile (Q3), and maximum.  
Outliers are typically defined as values that fall below:  
**Q1 - 1.5 × IQR** or above **Q3 + 1.5 × IQR**,  
where

**IQR (Interquartile Range)** = Q3 - Q1.

The boxplot() function in R visually shows outliers as individual points, and the boxplot.stats() function returns these values explicitly under the $out component.

This experiment helps in understanding how to detect and interpret outliers using built-in R functions.

**Source Code:**

# Sample data

data <- c(100, 120, 140, 150, 100, 1000, 200, 220, 240, 800)

boxplot(data, main = "Boxplot of Sample Data", col = "lightblue")

boxplot\_stats <- boxplot.stats(data)

outliers <- boxplot\_stats$out # outlier values

# Display the outliers

cat("\nOutliers:\n")

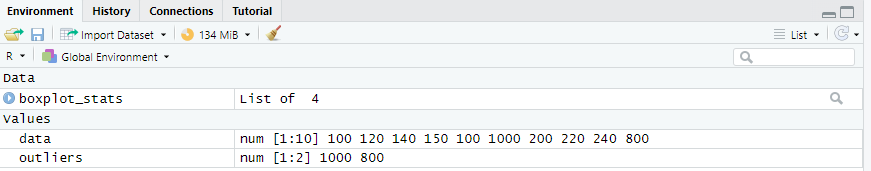
print(outliers)

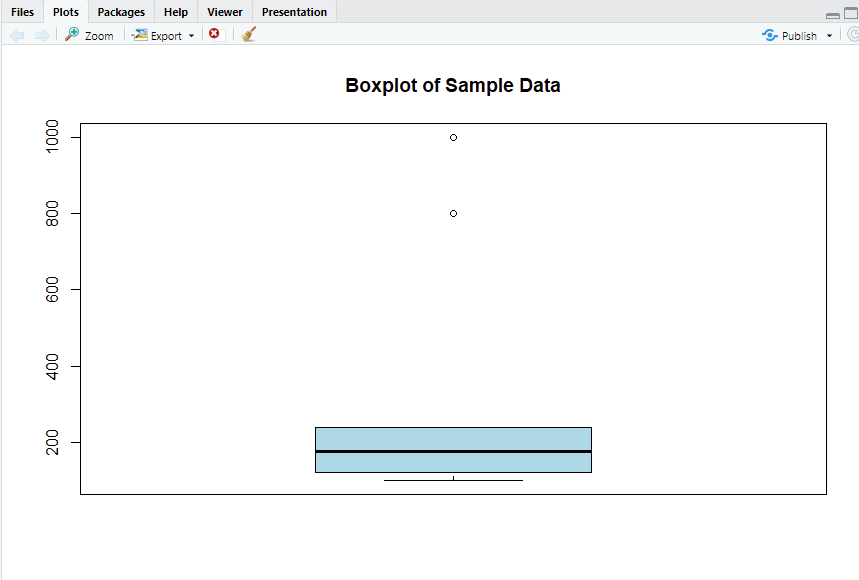
cat("\nBoxplot Summary Statistics:\n")

print(boxplot\_stats)

**Output:**







**Learning Outcome:**

1. Gained a clear understanding of what outliers are and why they matter in data analysis.
2. Learned to use the boxplot() and boxplot.stats() functions in R for outlier detection.
3. Interpreted boxplot visuals and summary statistics effectively.
4. Understood how to extract and display outliers programmatically using R.

**EXPERIMENT NO - 09**

**Aim of the program: Write a program to find the riskiest project out of two mutually exclusive projects in R.**

**Theory:**

In capital budgeting, when two projects are **mutually exclusive**, only one can be selected. To make an informed decision, especially when both projects have different levels of return and risk, it is important to assess **risk** using statistical measures.

Two common metrics used are:

* **Standard Deviation (SD):** Measures the amount of variability or spread in a set of data. A higher SD indicates more risk.
* **Coefficient of Variation (CV):** A standardized measure of dispersion calculated as the ratio of standard deviation to the mean. It allows for comparison of risk across datasets with different scales.

The project with the **higher CV** is considered **riskier**, as it has more variability relative to its expected return. In this program, R is used to calculate the standard deviation and CV for both projects and compare them to determine which is riskier.

**Source Code:**

# Cash flows or returns of two projects

project\_A <- c(100, 120, 130, 150, 160) # cash flows for Project A

project\_B <- c(90, 110, 140, 180, 220) # cash flows for Project B

# Compute standard deviation

sd\_A <- sd(project\_A)

sd\_B <- sd(project\_B)

# Compute coefficient of variation (CV = sd/mean)

cv\_A <- sd\_A / mean(project\_A)

cv\_B <- sd\_B / mean(project\_B)

# Determine the riskier project

if (cv\_A > cv\_B) {

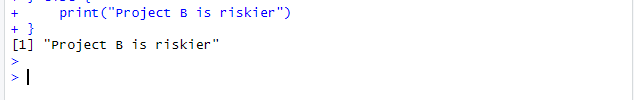
print("Project A is riskier")

} else {

print("Project B is riskier")

}

**Output:**





**Learning Outcome:**

1. Understood the concept of mutually exclusive projects in project selection.
2. Learned to calculate standard deviation and coefficient of variation in R.
3. Gained the ability to compare risks between projects using statistical analysis.
4. Developed decision-making skills for evaluating project risks using R programming.

**EXPERIMENT NO - 10**

**Aim of the program: Write a program to draw a scatter diagram, residual plots, outliers leverage and influential data points in R.**

**Theory:**

**Scatter plots**, **residual plots**, **leverage**, and **influential data points** are essential tools in regression analysis used to understand model performance, data patterns, and identify problematic observations.

* **Scatter Plot:** A graphical representation of the relationship between two variables. It helps visualize the trend, spread, and any visible correlation.
* **Linear Model (lm) in R:** Fits a straight line to the data (simple linear regression), estimating the relationship between the independent (x) and dependent (y) variables.
* **Residual Plot:** Shows the difference between observed and predicted values. Ideally, residuals should be randomly scattered around zero, indicating a good model fit.
* **Outliers:** Data points that lie far from the trend. Identified using boxplots of residuals.
* **Leverage:** Measures the influence of each x-value on its fitted y-value. Points with unusually high leverage can distort the regression line.
* **Cook’s Distance:** Measures the influence of each data point on the regression coefficients. Points with large Cook’s distance are considered **influential**.

These diagnostic tools help refine models, identify data issues, and improve predictions.

**Source Code:**

# Sample dataset

x <- c(1, 2, 3, ..., 25)

y <- c(2, 4, 6, 8, ..., 50)

model <- lm(y ~ x) # Fit a linear model

plot(x, y, main = "Scatter Plot", xlab = "x", ylab = "y", pch = 16, col = "blue")

abline(model, col = "red") # Add regression line

residuals <- resid(model)

plot(x, residuals, main = "Residual Plot", xlab = "x", ylab = "Residuals", pch = 16, col = "blue")

abline(h = 0, col = "red") # Reference line at 0

boxplot\_result <- boxplot(residuals, main = "Boxplot of Residuals", col = "lightblue", ylab = "Residuals")

outliers <- boxplot\_result$out # Extract outliers

cat("Outliers:\n")

if (length(outliers) > 0) {

cat("Residuals:", outliers, "\n\n")

} else {

cat("None\n\n")

}

leverage <- hatvalues(model)

high\_leverage <- which(leverage > (2 \* mean(leverage)))

cat("High Leverage Points:\n")

# If found, print their indices and corresponding x, y values

cooks\_distances <- cooks.distance(model)

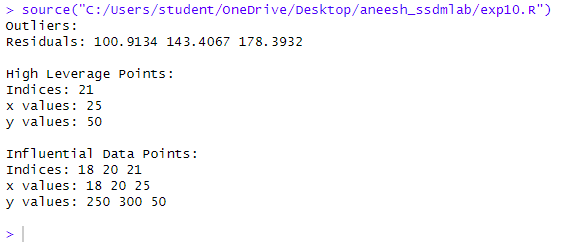
influential\_points <- which(cooks\_distances > (4 / length(x)))

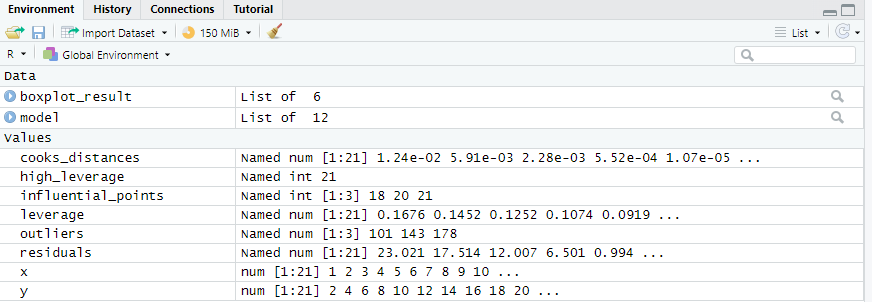
cat("Influential Data Points:\n")

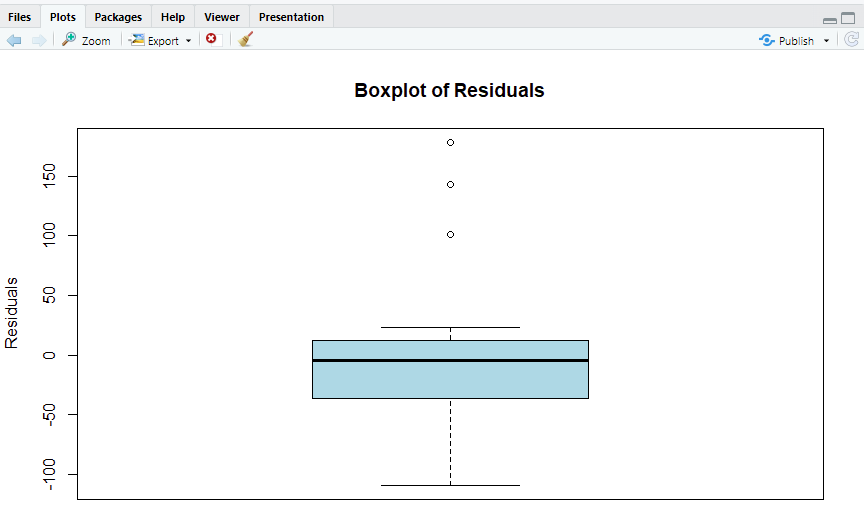
# If found, prints indices and x/y values of influential points

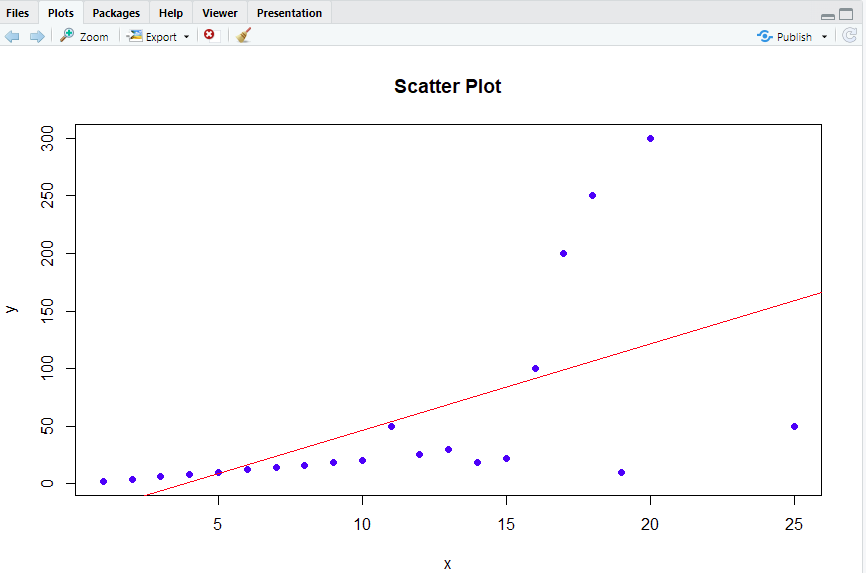
\

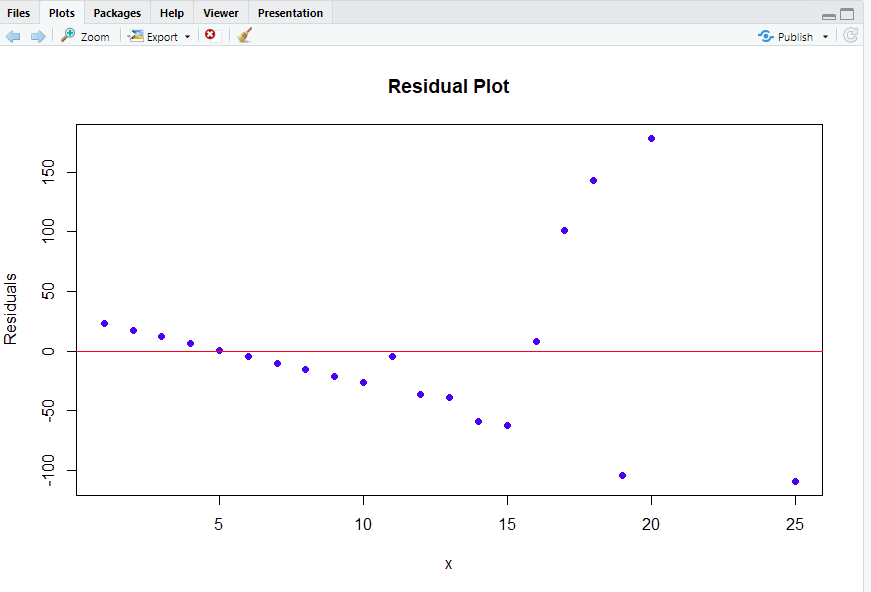
**Output:**











**Learning Outcome:**

1. Learned how to visually analyze data using scatter plots and regression lines in R.
2. Understood the significance of residuals, and how residual plots help evaluate model accuracy.
3. Gained the ability to detect outliers, high leverage points, and influential observations.
4. Developed skills in using R functions like lm(), resid(), hatvalues(), and cooks.distance() for regression diagnostics.
5. Improved interpretation and evaluation of linear regression models through graphical and statistical analysis.

**EXPERIMENT NO - 11**

**Aim of the program: Write a program to calculate correlation using R.**

**Theory:**

**Correlation** is a statistical measure that expresses the extent to which two variables are linearly related. It indicates the strength and direction of a relationship between variables. The most commonly used type is the **Pearson correlation coefficient**, which ranges from:

* **+1**: Perfect positive linear relationship
* **0**: No linear relationship
* **−1**: Perfect negative linear relationship

In R, the cor() function is used to calculate the correlation between two numeric vectors. The Pearson method is the default and assumes both variables are normally distributed and measured on interval/ratio scales.

Correlation helps in identifying patterns in data and is often the first step before applying regression models or other predictive analytics.

**Source Code:**

x <- c(1, 2, 3, 4, 5)

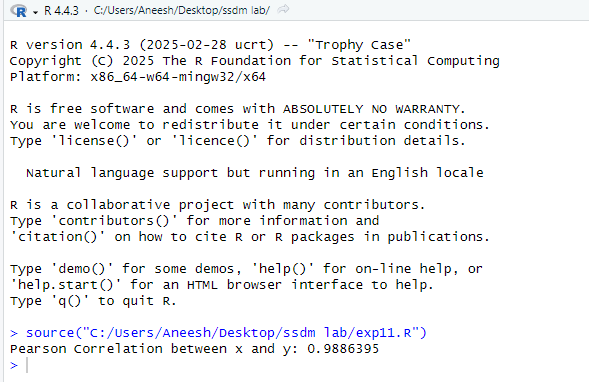
y <- c(1.5, 3.1, 5.7, 8.55, 9.4)

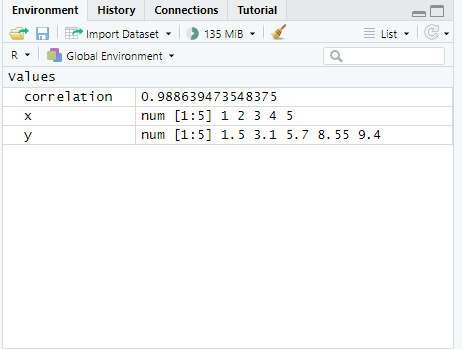
# Pearson correlation

correlation <- cor(x, y)

cat("Pearson Correlation between x and y:", correlation)

**Output:**





**Learning Outcome:**

1. Understood the concept and importance of **correlation** in statistical analysis.
2. Learned how to calculate **Pearson correlation** using the cor() function in R.
3. Interpreted the **strength and direction** of the relationship between two variables.
4. Developed foundational skills for analyzing relationships in datasets, aiding in further machine learning or data analysis tasks.

**EXPERIMENT NO - 12**

**Aim of the program: Write a program to implement Time Series using R.**

**Theory:**

**Time Series** is a sequence of data points collected or recorded at regular time intervals. Time series analysis involves understanding patterns such as **trend**, **seasonality**, and **cyclic behavior** in data that is ordered in time.

In R, the ts() function is used to create time series objects. Key components include:

* **Start:** Specifies the starting time of the data.
* **Frequency:** Indicates the number of observations per unit time (e.g., 12 for monthly data).
* **Matrix Format:** Enables comparison of multiple series (e.g., rainfall in two years side-by-side).

**Time Series Plot** provides a visual overview of how the data changes over time and helps identify patterns, anomalies, or shifts.

This experiment uses rainfall data over 12 months for two years to construct and visualize a multivariate time series.

**Source Code:**

rainfall1 <- c(799, 1174.8, 865.1, 1334.6, 635.4, 918.5, 685.5, 998.6, 784.2, 985, 882.8, 1071)

rainfall2 <- c(655, 1306.9, 1323.4, 1172.2, 562.2, 824, 822.4, 1265.5, 799.6, 1105.6, 1106.7, 1337.8)

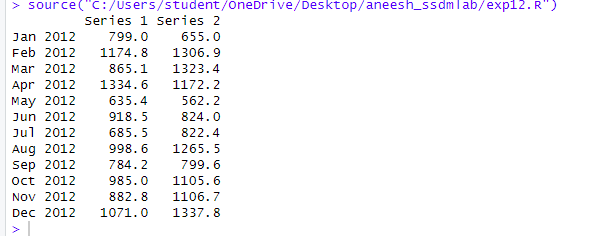
combined.rainfall <- matrix(c(rainfall1, rainfall2), nrow = 12)

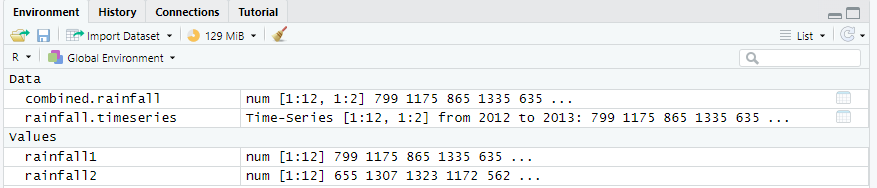
rainfall.timeseries <- ts(combined.rainfall, start = c(2012, 1), frequency = 12)

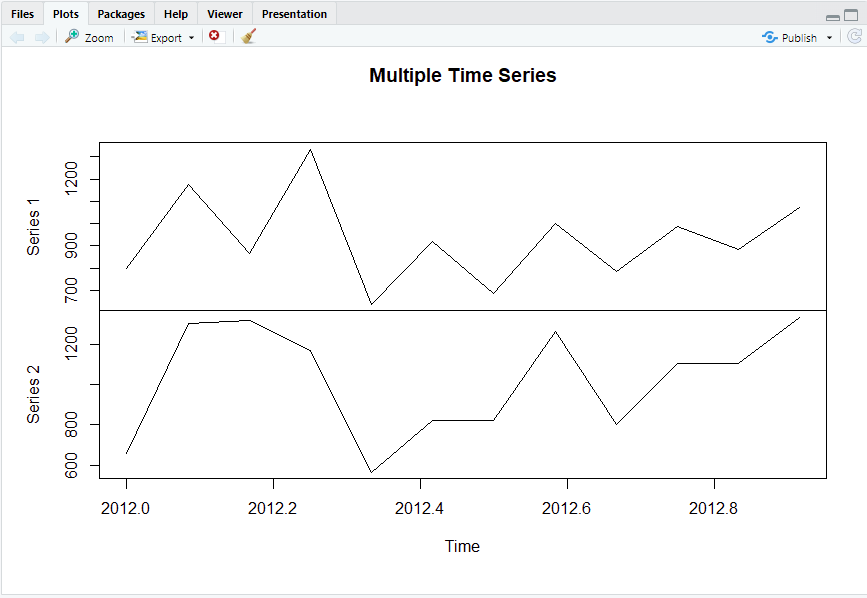
print(rainfall.timeseries)

plot(rainfall.timeseries, main = "Multiple Time Series")

**Output:**







**Learning Outcome:**

1. Gained understanding of time series data and its structure.
2. Learned how to represent and manipulate time-based data using R’s ts() function.
3. Visualized multiple time series on a single plot for comparative analysis.
4. Developed skills necessary for further analysis such as forecasting and trend detection.

**EXPERIMENT NO - 13**

**Aim of the program: Write a program to perform and visualize simple linear regression analysis using R and ggplot2.**

**Theory:**

Simple Linear Regression is a statistical technique used to model the relationship between a dependent variable (y) and a single independent variable (x). The model is based on fitting a straight line (y = mx + c) through the data points that best describes the relationship.

In R, the lm() function is used to perform linear regression. The output provides useful statistics like the slope, intercept, R-squared value (which tells how well the data fits the model), and p-values for significance testing.

The ggplot2 package is widely used in R for creating aesthetic and customizable plots. It allows the addition of regression lines using geom\_smooth() and enhances the visualization with titles, axis labels, and point styles.

**Source Code:**

# Install and load ggplot2 package

install.packages("ggplot2")

library(ggplot2) # Load the ggplot2 package

# Sample dataset

x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

y <- c(2, 4, 6, 7, 10, 12, 14, 15, 20, 22)

data <- data.frame(x, y) # Combine x and y into a dataframe

# Fit linear model

model <- lm(y ~ x, data)

print(summary(model))

plot <- ggplot(data, aes(x = x, y = y)) +

geom\_point(color = "blue", size = 3) + # Scatter points

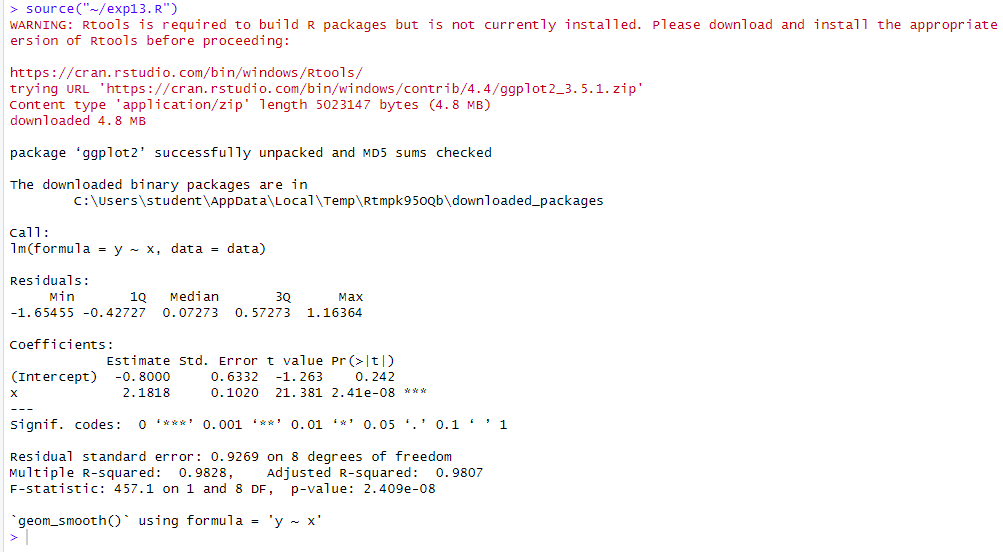
geom\_smooth(method = "lm", se = FALSE, color = "red") + # Regression line

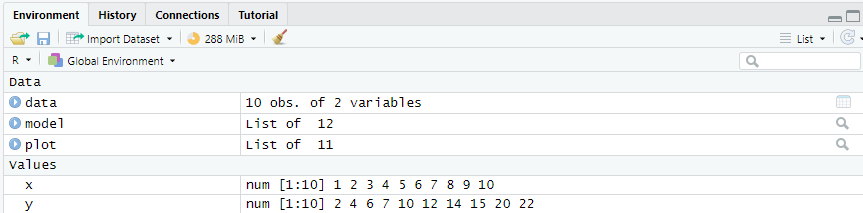
ggtitle("Linear Regression") +

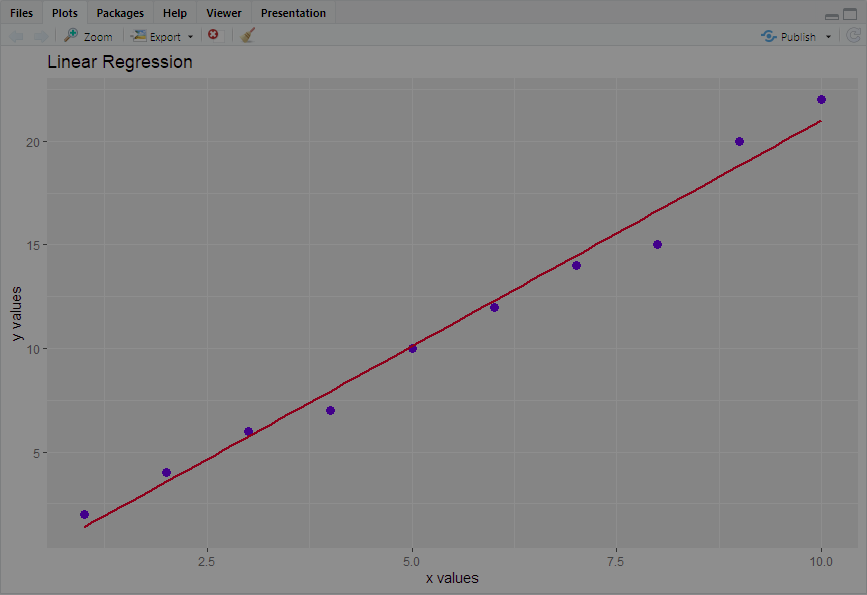
xlab("x values") + # X-axis label

ylab("y values") # Y-axis label

print(plot)

**Output:**





**Learning Outcomes:**

1. Understood the concept of simple linear regression and its role in predictive analytics.
2. Learned to use the lm() function to build and summarize a regression model.
3. Gained skills in data visualization using ggplot2, including adding scatter plots and regression lines.
4. Interpreted regression statistics such as R-squared, slope, and p-values to evaluate model accuracy and significance.