Statistics, Statistical Modelling & Data Analytics LAB

(DA-304P)

Faculty Name : Mr. Akshay Mool

Name	:	Amit Singhal
Enrollment No.	:	11614802722
Semester	:	6
Group	:	AIML-II-B

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

उद्यमेन हि सिध्यन्ति
कार्याणि न मनोरथैः

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

VISION

"To attain global excellence through **education, innovation, research,** and **work ethics** with the commitment to **serve humanity."**

MISSION

M1. To promote diversification by adopting advancement in science, technology, management, and allied discipline through continuous learning

M2. To foster **moral values** in students and equip them for developing sustainable solutions to serve both national and global needs in society and industry.

M3. To **digitize educational resources and process** for enhanced teaching and effective learning.

M4. To cultivate an **environment** supporting **incubation, product development, technology transfer, capacity building and entrepreneurship**.

M5. To encourage **faculty-student networking with alumni, industry, institutions,** and other **stakeholders** for collective engagement.

Department of Computer Science and Engineering

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

VISION

"To attain global excellence through education, innovation, research, and work ethics in the field of Computer Science and engineering with the commitment to serve humanity."

MISSION

M1. To lead in the advancement of computer science and engineering through internationally recognized research and education.

M2. To prepare students for full and ethical participation in a diverse society and encourage lifelong learning.

M3. To foster development of problem solving and communication skills as an integral component of the profession.

M4. To impart knowledge, skills and cultivate an environment supporting incubation, product development, technology transfer, capacity building and entrepreneurship in the field of computer science and engineering.

M5. To encourage faculty, student's networking with alumni, industry, institutions, and other stakeholders for collective engagement.

LAB INDEX

S.No	Experiment	R1 Is able to identify and define the objective of the given problem? 2 Marks	R2 Is proposed design /procedure /algorithm solves the problem? 2 Marks	R3 Has the understanding of the tool/programming language to implement the proposed solution? 2 Marks	R4 Are the result(s) verified using sufficient test data to support the conclusions? 2 Marks	R5 Individuality of submission? 2 Marks	Total Marks (10)	Remarks	Faculty Signature

LAB EXERCISE - 1

AIM :: Implement the Basic Matrix Operations in Scilab.

Theory ::

In Scilab, matrices are a fundamental data structure, and there are several basic operations that can be performed on them. Below are the key matrix operations in Scilab:

1. **Matrix Creation**:
   - You can create matrices using square brackets `[]`. Elements within the matrix are separated by spaces (for rows) or semicolons (for columns).
   - Example: `A = [1, 2, 3; 4, 5, 6]`

2. **Matrix Addition and Subtraction**:
   - Matrices of the same dimension can be added or subtracted element-wise.
   - Example: `C = A + B` or `D = A - B`

3. **Matrix Multiplication**:
   - Matrix multiplication in Scilab is done using the `*` operator.
   - For two matrices `A` (of size m x n) and `B` (of size n x p), the resulting matrix `C = A * B` will be of size m x p.
   - Example: `C = A * B`

4. **Element-wise Operations**:
   - Element-wise operations are performed using `.*`, `./`, `.^` for multiplication, division, and exponentiation respectively.
   - Example: `C = A .* B` multiplies corresponding elements of `A` and `B`.

5. **Transpose of a Matrix**:
   - The transpose of a matrix is obtained using the `'` operator.
   - Example: `B = A'`

6. **Matrix Inversion**:
   - To find the inverse of a square matrix, use the `inv()` function.
   - Example: `A_inv = inv(A)`

7. **Determinant of a Matrix**:
   - The determinant of a matrix is calculated using the `det()` function.
   - Example: `d = det(A)`

These basic operations provide the foundation for more advanced matrix manipulations and are essential in scientific computing and linear algebra tasks in Scilab.

Code ::

```
disp("Amit Singhal - 11614802722");

// Create Identity Matrix first
I_matrix = eye(3, 3);
disp("Identity Matrix of size 3x3:");
disp(I_matrix);

A = [1, 2, 3; 0, 1, 4; 5, 6, 0];
B = [2, 1, 1; 1, 3, 2; 1, 2, 4];

disp("Matrix A:");
disp(A);

size_A = size(A);
disp("Size of Matrix A:");
disp(size_A);

disp("Matrix B:");
disp(B);

size_B = size(B);
disp("Size of Matrix B:");
disp(size_B);

// Matrix Addition
C = A + B;
disp("Matrix A + B:");
disp(C);

// Matrix Subtraction
D = A - B;
disp("Matrix A - B:");
disp(D);

// Matrix Multiplication
E = A * B;
disp("Matrix A * B:");
disp(E);

// Matrix Transpose
F = A';
disp("Transpose of Matrix A:");
disp(F);

// Element-wise multiplication with itself
H = A .* A;
disp("Element-wise Multiplication of A and A:");
disp(H);
```

```
// Element-wise squaring
I = A .^ 2;
disp("Element-wise Squared Matrix A:");
disp(I);

// Matrix Determinant
det_A = det(A);
disp("Determinant of Matrix A:");
disp(det_A);

// Matrix Inversion with error handling
try
    inv_A = inv(A); // Inverse of A
    disp("Inverse of Matrix A:");
    disp(inv_A);

    // Verify inverse by multiplying A * A^(-1)
    verify = A * inv_A;
    disp("Verification A * A^(-1) (should be identity matrix):");
    disp(verify);
catch
    disp("Matrix A is singular and cannot be inverted.");
end
```

Output ::

```
-->exec('/home/singhal-amit/Downloads/Sem 6/LabWork/Stats/prg - 1.sci', -1)

  "Amit Singhal - 11614802722"

  "Identity Matrix of size 3x3:"

   1.   0.   0.
   0.   1.   0.
   0.   0.   1.

  "Matrix A:"

   1.   2.   3.
   0.   1.   4.
   5.   6.   0.

  "Size of Matrix A:"

   3.   3.

  "Matrix B:"

   2.   1.   1.
   1.   3.   2.
   1.   2.   4.

  "Size of Matrix B:"

   3.   3.
```

```
"Matrix A + B:"

 3.   3.   4.
 1.   4.   6.
 6.   8.   4.

"Matrix A - B:"

-1.   1.   2.
-1.  -2.   2.
 4.   4.  -4.

"Matrix A * B:"

 7.    13.   17.
 5.    11.   18.
 16.   23.   17.

"Transpose of Matrix A:"

 1.   0.   5.
 2.   1.   6.
 3.   4.   0.

"Element-wise Multiplication of A and A:"

 1.    4.    9.
 0.    1.    16.
 25.   36.   0.

"Element-wise Squared Matrix A:"

 1.    4.    9.
 0.    1.    16.
 25.   36.   0.

"Determinant of Matrix A:"

 1.0000000

"Inverse of Matrix A:"

-24.   18.   5.
 20.  -15.  -4.
-5.    4.    1.

"Verification A * A^(-1) (should be identity matrix):"

 1.           0.           8.882D-16
 0.           1.           0.
-1.421D-14   2.842D-14   1.
```

LAB EXERCISE - 2

AIM :: Find the Eigen Values and Eigen Vectors in Scilab.

Theory ::

Eigenvalues and Eigenvectors in Scilab

Eigenvalues and eigenvectors are fundamental concepts in linear algebra, widely used in fields like physics, engineering, machine learning, and data science. Given a square matrix A, an eigenvalue λ (lambda) and its corresponding eigenvector v satisfy the equation:

$$A \cdot v = \lambda \cdot v$$

Where:

- A is a square matrix (e.g., 2x2, 3x3),
- λ (lambda) is the eigenvalue,
- v is the eigenvector.

1. Using `spec` function in Scilab

Scilab provides the `spec` function to compute the eigenvalues and eigenvectors of a matrix.

Syntax:

```
[values, vectors] = spec(A)
```

Where:

- `A` is the matrix for which eigenvalues and eigenvectors are to be found.
- `values` is a column vector containing the eigenvalues.
- `vectors` is a matrix where each column is an eigenvector corresponding to the eigenvalue in `values`.

Example:

```
A = [4, 1; 2, 3];
[values, vectors] = spec(A);
disp(values);    // Eigenvalues
disp(vectors);   // Eigenvectors
```

In this example, `spec` computes the eigenvalues and eigenvectors of matrix AA.

2. Without using `spec` function

To find the eigenvalues and eigenvectors manually, we need to solve the characteristic equation:

$$|A - \lambda I| = 0$$

Where:

- I is the identity matrix,

- λ are the eigenvalues.

Once the eigenvalues are found, the eigenvectors can be calculated by solving the equation

$$(A - \lambda I) \cdot v = 0$$

Steps:

1. Compute the characteristic polynomial and solve for eigenvalues λ\lambda.
2. For each eigenvalue, substitute λ\lambda into $(A-\lambda I)$ and solve $(A-\lambda I) \cdot v = 0$ for the eigenvectors.

Example:

```
A = [4, 1; 2, 3];
det_A = det(A);

// Find eigenvalues manually
lambda1 = 5;
lambda2 = 2;

// Solve for eigenvectors corresponding to each lambda
v1 = null(A - lambda1*eye(2));
v2 = null(A - lambda2*eye(2));

disp(v1);
disp(v2);
```

In this approach:

- We manually find the eigenvalues, then solve for the eigenvectors by using the null space of $(A-\lambda I)$.
- This method is more tedious but gives deeper insight into the computation process.

1. Using `spec` function in Scilab

Code ::

```
disp("Amit Singhal - 11614802722");

A = [-5, 2; -9, 6];

disp("The input matrix A is:");
disp(A);

// Use the spec function to compute eigenvalues and eigenvectors
[eigenvectors, eigenvalues] = spec(A);

// Display eigenvalues
disp("Eigenvalues:");
for i = 1:size(eigenvalues, 1)
    disp("λ" + string(i) + " = " + string(eigenvalues(i, i)));
end

// Display eigenvectors
disp("Eigenvectors:");
for i = 1:size(eigenvectors, 2)
    disp("Eigenvector corresponding to λ" + string(i) + " = " + string(eigenvalues(i, i)) + ":");
```

```
    disp(eigenvectors(:, i));
end

// Verification: Check A*v = λ*v for each eigenvalue and eigenvector
disp("Verification (A*v = λ*v):");
for i = 1:size(eigenvalues, 1)
    lambda_val = eigenvalues(i, i);
    eigenvector = eigenvectors(:, i);
    left_side = A * eigenvector;
    right_side = lambda_val * eigenvector;
    disp("For λ" + string(i) + " = " + string(lambda_val) + ":");
    disp("A*v = ");
    disp(left_side);
    disp("λ*v = ");
    disp(right_side);
end
```

Output ::

```
-->exec('/home/singhal-amit/Downloads/prg - 2.sci', -1)

  "Amit Singhal - 11614802722"

  "The input matrix A is:"

  -5.   2.
  -9.   6.

  "Eigenvalues:"

  "λ1 = -3"

  "λ2 = 4"

  "Eigenvectors:"

  "Eigenvector corresponding to λ1 = -3:"

  -0.7071068
  -0.7071068

  "Eigenvector corresponding to λ2 = 4:"

  -0.2169305
  -0.9761871
```

```
"Verification (A*v = λ*v):"

"For λ1 = -3:"

"A*v = "

 2.1213203
 2.1213203

"λ*v = "

 2.1213203
 2.1213203

"For λ2 = 4:"

"A*v = "

-0.8677218
-3.9047482

"λ*v = "

-0.8677218
-3.9047482
```

2. Without using spec function

Code ::

```
disp("Amit Singhal - 11614802722");

A = [-5, 2; -9, 6];

disp("The input matrix A is:");
disp(A);

// Solve the characteristic equation det(A - λI) = 0
lambda = poly(0, 'lambda');
I = eye(A);
char_matrix = A - lambda * I;
characteristic_eq = det(char_matrix)
```

```
// Solve the characteristic equation for λ - roots = Eigen Values
eigenvalues = roots(characteristic_eq);

disp("Eigenvalues:");
for i = 1:length(eigenvalues)
    disp("λ" + string(i) + " = " + string(eigenvalues(i)));
end

// Solve for Eigen Vectors corresponding to each Eigen Value
disp("Eigenvectors:");
for i = 1:length(eigenvalues)
    lambda_val = eigenvalues(i);
    eigenvector_matrix = A - lambda_val * I;
    disp("Eigenvector corresponding to λ" + string(i) + " = " + string(lambda_val) + ":");
    eigenvector = kernel(eigenvector_matrix);
    disp(eigenvector);
end

// Verification: Check A*v = λ*v for each Eigen Value & Eigen Vector
disp("Verification (A*v = λ*v):");
for i = 1:length(eigenvalues)
    lambda_val = eigenvalues(i);
    eigenvector = kernel(A - lambda_val * I);
    left_side = A * eigenvector;
    right_side = lambda_val * eigenvector;
    disp("For λ" + string(i) + " = " + string(lambda_val) + ":");
    disp("A*v = ");
    disp(left_side);
    disp("λ*v = ");
    disp(right_side);
end
```

Output ::

```
-->exec('/home/singhal-amit/Downloads/prg - 3.sci', -1)

  "Amit Singhal - 11614802722"

  "The input matrix A is:"

  -5.   2.
  -9.   6.

  "Eigenvalues:"

  "λ1 = 4"

  "λ2 = -3"

  "Eigenvectors:"

  "Eigenvector corresponding to λ1 = 4:"

   0.2169305
   0.9761871

  "Eigenvector corresponding to λ2 = -3:"

   0.7071068
   0.7071068

  "Verification (A*v = λ*v):"

  "For λ1 = 4:"

  "A*v = "

   0.8677218
   3.9047482

  "λ*v = "

   0.8677218
   3.9047482

  "For λ2 = -3:"

  "A*v = "

  -2.1213203
  -2.1213203

  "λ*v = "

  -2.1213203
  -2.1213203
```

LAB EXERCISE - 3

AIM :: Solve equations by Gauss Elimination, Gauss Jordan Method and Gauss Seidel in Scilab.

Theory ::

Gauss Elimination Method

The **Gauss Elimination Method** is a systematic technique for solving a system of linear equations. It transforms a given system into an upper triangular form using elementary row operations, making it easier to solve using back-substitution.

This method is widely used due to its efficiency and simplicity, especially in numerical computations.

Mathematical Foundation

A system of linear equations can be represented in matrix form as:

$$AX = B$$

where:

- A is an $n \times n$ coefficient matrix.
- X is an $n \times 1$ column matrix of unknown variables.
- B is an $n \times 1$ column matrix of constants.

The goal of Gauss Elimination is to transform the augmented matrix $[A|B]$ into an upper triangular form so that the system can be solved by back-substitution.

Steps of Gauss Elimination Method

Step 1: Form the Augmented Matrix

Construct the augmented matrix $[A|B]$ by appending the column matrix B to the coefficient matrix A.

Step 2: Convert to Upper Triangular Form

Perform **forward elimination** by applying **row operations** to transform the matrix into an upper triangular form.

- Select a pivot element (typically, the first nonzero entry in a column).
- Use the pivot row to eliminate all elements below the pivot by subtracting appropriate multiples of the pivot row.
- Repeat the process for each column until the matrix becomes upper triangular.

Step 3: Back Substitution

After obtaining the upper triangular matrix, solve for the unknowns starting from the last row:

- Solve for the last variable using the last equation.
- Substitute this value into the previous equation and solve for the second-last variable.
- Continue the process until all variables are determined.

Example

Solve the following system of equations using Gauss Elimination:

$$2x+y-z = 8$$

$$x-y+2z = -11$$

$$-2x+y+2z = -3$$

Step 1: Form the Augmented Matrix

$$\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}$$

Step 2: Convert to Upper Triangular Form

Eliminate the first column below the pivot (2 in row 1):

- Multiply row 1 by 3/2 and add it to row 2.
- Multiply row 1 by 2/2 and add it to row 3.

New matrix:

$$\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5 & -1 \\ 0 & 2 & 1 & 5 \end{array}$$

Eliminate the second column below the pivot (-0.5 in row 2):

- Multiply row 2 by −4 and add to row 3.

New matrix:

$$\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5 & -1 \\ 0 & 0 & 3 & 9 \end{array}$$

Step 3: Back Substitution

- From the last equation: $3z=9 \Rightarrow z=3$

- Substitute z=3 into the second equation:

$$-0.5y+0.5(3)=1 \Rightarrow y=-1$$

- Substitute y=−1, z=3 into the first equation:

$$2x+(-1)-3=8 \Rightarrow x=6$$

Final Solution

$x = 6$

$y = -1$

$z = 3$

Code ::

```
disp("Amit Singhal - 11614802722");
printf("\n");

A = input("Enter the matrix A: ");
b = input("Enter the vector b: ");
Aug = [A, b];

disp("Augmented Matrix:");
disp(Aug);
printf("\n");

n = size(A, 1);

// Forward elimination (Gauss Elimination)
for k = 1:n-1
    if Aug(k, k) == 0 then
        disp("Pivot element is zero, row swapping required!");
        break;
    end

    for i = k+1:n
        factor = Aug(i, k) / Aug(k, k);
        Aug(i, k:n+1) = Aug(i, k:n+1) - factor * Aug(k, k:n+1);
    end
end

printf("\n");
disp("Upper Triangular Matrix:");
disp(Aug);
printf("\n");

// Back substitution
x = zeros(n, 1);

for i = n:-1:1
    sum_val = 0;

    if i < n then
        sum_val = sum(Aug(i, i+1:n) .* x(i+1:n));
    end

    x(i) = (Aug(i, n+1) - sum_val) / Aug(i, i);
end
```

```
printf("\n");
disp("Solution:");
for i = 1:n
    printf("x%d = %.6f\n", i, x(i));
end
printf("\n");
```

Output ::

```
  "Amit Singhal - 11614802722"

Enter the matrix A: [2 3 -1; 4 1 2; 3 2 3]


Enter the vector b: [5; 6; 7]


  "Augmented Matrix:"
   2.   3.  -1.   5.
   4.   1.   2.   6.
   3.   2.   3.   7.

  "Upper Triangular Matrix:"
   2.   3.  -1.    5.
   0.  -5.   4.   -4.
   0.   0.   2.5   1.5

  "Solution:"
   0.8800000
   1.28
   0.6
```

Theory ::

Gauss-Jordan Method

The **Gauss-Jordan Method** is an extension of the **Gauss Elimination Method** that reduces a system of linear equations to **reduced row echelon form (RREF)** instead of just an upper triangular form. This method eliminates variables both above and below the pivot, resulting in a diagonal matrix where each equation directly provides the value of one variable. It is particularly useful for solving linear systems, finding inverses of matrices, and determining rank.

Mathematical Foundation

A system of linear equations can be written in matrix form as:

$$\mathbf{AX = B}$$

where:

- **A** is an $n \times n$ coefficient matrix.
- **X** is an $n \times 1$ column matrix of unknown variables.
- **B** is an $n \times 1$ column matrix of constants.

The goal of **Gauss-Jordan Elimination** is to transform the augmented matrix **[A | B]** into **reduced row echelon form**, making it easier to solve for the unknowns directly.

Steps of Gauss-Jordan Method

Step 1: Form the Augmented Matrix

Construct the augmented matrix **[A | B]** by appending the column matrix **B** to the coefficient matrix **A**.

Step 2: Convert to Row Echelon Form

Perform row operations to convert the matrix into an upper triangular form, similar to Gauss Elimination:

- Select a **pivot element** (the first nonzero entry in a column).
- Use the pivot row to **eliminate all elements below the pivot** by subtracting appropriate multiples of the pivot row.
- Repeat this process for each column.

Step 3: Convert to Reduced Row Echelon Form (RREF)

- Normalize each pivot row by making the pivot element **1** (by dividing the row by the pivot element).
- Use the pivot row to eliminate all elements **above the pivot**, making the matrix diagonal.

Step 4: Extract the Solution

Once the matrix is in reduced row echelon form, the solutions for the unknown variables can be directly read from the matrix.

Example

Solve the following system of equations using **Gauss-Jordan Elimination**:

$$2x+y-z = 8$$

$$x-y+2z = -11$$

$$-2x+y+2z = -3$$

Step 1: Form the Augmented Matrix

$$\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 1 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}$$

Step 2: Convert to Row Echelon Form

- Swap row 1 and row 2 to get a leading 1 in the first column:

$$\begin{array}{ccc|c} 1 & -1 & 2 & -11 \\ 2 & 1 & -1 & 8 \\ -2 & 1 & 2 & -3 \end{array}$$

- Eliminate the first column below the pivot using row operations:

$$\begin{array}{ccc|c} 1 & -1 & 2 & -11 \\ 0 & 3 & -5 & 30 \\ 0 & -1 & 6 & -25 \end{array}$$

- Eliminate the second column below the pivot:

$$\begin{array}{ccc|c} 1 & -1 & 2 & -11 \end{array}$$

$$\begin{matrix} 0 & 1 & -5/3 & | & 10 \\ 0 & 0 & 11/3 & | & -15 \end{matrix}$$

Step 3: Convert to Reduced Row Echelon Form

- Normalize the last row by making the pivot 1:

$$\begin{matrix} 1 & -1 & 2 & | & -11 \\ 0 & 1 & -5/3 & | & 10 \\ 0 & 0 & 1 & | & -15/11 \end{matrix}$$

- Eliminate elements above the last pivot:

$$\begin{matrix} 1 & -1 & 0 & | & -43/11 \\ 0 & 1 & 0 & | & 85/11 \\ 0 & 0 & 1 & | & -15/11 \end{matrix}$$

Step 4: Extract the Solution

From the final matrix:

$$x = -43/11$$

$$y = 85/11$$

$$z = -15/11$$

Thus, the final solution is: $x=-3.91$, $y=7.73$, $z=-1.36$

Code ::

```
disp("Amit Singhal - 11614802722");
printf("\n");

A = input("Enter the matrix A: ");
printf("\n");
b = input("Enter the vector b: ");
printf("\n");

Aug = [A, b];  // Augmented matrix

disp("Augmented Matrix:");
disp(Aug);
printf("\n");

n = size(A, 1);

for k = 1:n
    // Pivoting: Make the diagonal element 1
    if Aug(k, k) == 0 then
        disp("Division by zero detected.");
        break;
    end
    Aug(k, :) = Aug(k, :) / Aug(k, k);  // Scale the k-th row

    // Make the elements below and above the pivot 0
    for i = 1:n
```

```
        if i ~= k then
            factor = Aug(i, k);
            Aug(i, :) = Aug(i, :) - factor * Aug(k, :);
        end
    end
end

printf("\n");
disp("Reduced Row Echelon Form (RREF) matrix:");
disp(Aug);
printf("\n");

x = Aug(:, $);  // Extract the solution from the last column

disp("Solution:");
disp(x);
printf("\n");
```

Output ::

```
  "Amit Singhal - 11614802722"

Enter the matrix A: [2 1 -1; -3 -1 2; -2 1 2]


Enter the vector b: [8; -11; -3]


  "Augmented Matrix:"
   2.    1.   -1.    8.
  -3.   -1.    2.   -11.
  -2.    1.    2.   -3.


  "Reduced Row Echelon Form (RREF) matrix:"
   1.    0.    0.    2.
   0.    1.    0.    3.
   0.    0.    1.   -1.

  "Solution:"
   2.
   3.
  -1.
```

Theory ::

Gauss-Seidel Method

The **Gauss-Seidel Method** is an iterative technique for solving a system of linear equations. Unlike the **Gauss Elimination Method**, which directly transforms the system into an upper triangular form, the **Gauss-Seidel Method** refines an initial approximation of the solution iteratively until convergence is achieved. It is particularly useful for large systems where direct methods become computationally expensive.

Mathematical Foundation

A system of linear equations can be expressed in matrix form as:

AX = B

where:

- **A** is an **n** × **n** coefficient matrix.
- **X** is an **n** × **1** column matrix of unknown variables.
- **B** is an **n** × **1** column matrix of constants.

The Gauss-Seidel Method improves an initial guess iteratively by solving for each variable sequentially and updating its value immediately for use in subsequent calculations.

Steps of the Gauss-Seidel Method

Step 1: Rearranging the Equations

Ensure that the system of equations is **diagonally dominant**, meaning that for each equation, the absolute value of the coefficient of the variable being solved for is greater than the sum of the absolute values of the other coefficients in the row. If not, row swapping may be required to achieve diagonal dominance.

Step 2: Iterative Computation

Use the following iterative formula for each variable:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

Where:

- $x_i^{(k+1)}$ is the updated value of the **i-th** variable in the **(k+1)-th** iteration.
- $x_j^{(k+1)}$ are the most recent updates for variables already computed in the iteration.
- $x_j^{(k)}$ are the previous iteration values for variables not yet updated.
- a_{ij} represents the elements of matrix **A**.
- b_i represents elements of matrix **B**.

This process is repeated until the solution converges within a specified tolerance.

Example

Solve the following system of equations using the **Gauss-Seidel Method**:

$$10x + 2y + z = 6 \; x + 10y - 2z = 4 \; -2x + 3y + 10z = 1$$

Step 1: Initial Guess

Let the initial approximation be: $x^{(0)} = 0, y^{(0)} = 0, z^{(0)} = 0.$

Step 2: Iterative Computation

Using the iterative formula:

1st iteration: $x^{(1)} = \frac{1}{10}(6 - 2(0) - 0) = 0.6$ $y^{(1)} = \frac{1}{10}(4 - 1(0.6) + 2(0)) = 0.34$ $z^{(1)} = \frac{1}{10}(1 + 2(0.6) - 3(0.34)) = 0.052$

2nd iteration: $x^{(2)} = \frac{1}{10}(6 - 2(0.34) - (0.052)) = 0.5628$ $y^{(2)} = \frac{1}{10}(4 - 1(0.5628) + 2(0.052)) = 0.3520$ $z^{(2)} = \frac{1}{10}(1 + 2(0.5628) - 3(0.3520)) = 0.02588$

This process is repeated until the solution converges to a specified tolerance.

Final Solution

After sufficient iterations, the approximate solution is obtained as:

x ≈ 0.55

y ≈ 0.35

z ≈ 0.03

Code ::

```
disp("Amit Singhal - 11614802722");
printf("\n");

disp("Solving system using Gauss-Seidel Method:");
printf("\n");

disp("Enter the coefficient matrix A:");
A = input("A = ");
printf("\n");

disp("Enter the constant matrix b:");
b = input("b = ");

n = size(A, 1);
x = zeros(n, 1);  // Initial guess (x, y) = (0, 0)
tolerance = 1e-6;
max_iterations = 100;

printf("Iteration\t x\t\t y");
printf("\n");

for iter = 1:max_iterations
    x_old = x;  // Store previous values for convergence check

    // Update x1
    x(1) = (1/2) * (8 - x(2));

    // Update x2 (y)
    x(2) = (1/2) * (1 - x(1));

    // Display current iteration values
    printf("%d\t\t %.6f\t %.6f\n", iter, x(1), x(2));

    // Check for convergence
    if norm(x - x_old, "inf") < tolerance
```

```
        printf("\nConvergence reached in %d iterations.\n", iter);
        break;
    end
end

printf("\nFinal Solution:\n");
printf("x = %.6f ≅ %.0f\n", x(1), round(x(1)));
printf("y = %.6f ≅ %.0f\n", x(2), round(x(2)));
printf("\n");
```

Output ::

```
  "Amit Singhal - 11614802722"

  "Solving system using Gauss-Seidel Method:"

  "Enter the coefficient matrix A:"
A = [2 1; 1 2]


  "Enter the constant matrix b:"
b = [8; 1]

Iteration        x              y
1                4.000000       -1.500000
2                4.750000       -1.875000
3                4.937500       -1.968750
4                4.984375       -1.992188
5                4.996094       -1.998047
6                4.999023       -1.999512
7                4.999756       -1.999878
8                4.999939       -1.999969
9                4.999985       -1.999992
10               4.999996       -1.999998
11               4.999999       -2.000000
12               5.000000       -2.000000

Convergence reached in 12 iterations.

Final Solution:
x = 5.000000 ≅ 5
y = -2.000000 ≅ -2
```

LAB EXERCISE - 4

AIM :: Exercises to implement the associative, commutative and distributive property in a matrix in Scilab and R.

Scilab Code ::

```
clc;
clear;
disp("Amit Singhal - 11614802722");
disp("Matrix Properties: Associative, Commutative, and Distributive");
n = input("Enter the size of square matrices (n x n): ");
disp("Enter elements of Matrix A:");
A = zeros(n, n);
for i = 1:n
  for j = 1:n
    A(i, j) = input("A(" + string(i) + "," + string(j) + ") = ");
  end
end
disp("Enter elements of Matrix B:");
B = zeros(n, n);
for i = 1:n
  for j = 1:n
    B(i, j) = input("B(" + string(i) + "," + string(j) + ") = ");
  end
end
disp("Enter elements of Matrix C:");
C = zeros(n, n);
for i = 1:n
  for j = 1:n
    C(i, j) = input("C(" + string(i) + "," + string(j) + ") = ");
  end
end
LHS_assoc = (A + B) + C;
RHS_assoc = A + (B + C);
disp("Associative Property Check: (A + B) + C == A + (B + C)");
if isequal(LHS_assoc, RHS_assoc)
  disp("True");
else
  disp("False");
end
```

```
disp("Commutative Property Check: A + B == B + A");
if isequal(A + B, B + A)
   disp("True");
else
   disp("False");
end
LHS_dist = A * (B + C);
RHS_dist = (A * B) + (A * C);
disp("Distributive Property Check: A(B + C) == AB + AC");
if isequal(LHS_dist, RHS_dist)
   disp("True");
else
   disp("False");
end
```

Output ::

```
  "Amit Singhal - 11614802722"
  "Matrix Properties: Associative, Commutative, and Distributive"
Enter the size of square matrices (n x n): 2

  "Enter elements of Matrix A:"
A(1,1) = 1

A(1,2) = 2

A(2,1) = 3

A(2,2) = 4

  "Enter elements of Matrix B:"
B(1,1) = 5

B(1,2) = 6

B(2,1) = 7

B(2,2) = 8

  "Enter elements of Matrix C:"
C(1,1) = 9

C(1,2) = 10

C(2,1) = 11

C(2,2) = 12

  "Associative Property Check: (A + B) + C == A + (B + C)"
  "True"
  "Commutative Property Check: A + B == B + A"
  "True"
  "Distributive Property Check: A(B + C) == AB + AC"
  "True"
```

R Code ::

```r
# Create matrices
A <- matrix(c(1, 2, 3, 4), nrow = 2, byrow = TRUE)
B <- matrix(c(5, 6, 7, 8), nrow = 2, byrow = TRUE)
C <- matrix(c(9, 10, 11, 12), nrow = 2, byrow = TRUE)

# Test Commutative Property: A*B = B*A
cat("Commutative Property:\n")
AB <- A %*% B
BA <- B %*% A
cat("\nA * B:\n"); print(AB)
cat("\nB * A:\n"); print(BA)
if (identical(AB, BA)) {
  cat("\nMatrix multiplication is commutative.\n\n")
} else {
  cat("\nMatrix multiplication is not commutative.\n\n")
}

# Test Associative Property: (A*B)*C = A*(B*C)
cat("Associative Property:\n")
left_associative <- (A %*% B) %*% C
right_associative <- A %*% (B %*% C)
cat("\n(A * B) * C:\n"); print(left_associative)
cat("\nA * (B * C):\n"); print(right_associative)
if (identical(left_associative, right_associative)) {
  cat("\nMatrix multiplication is associative.\n\n")
} else {
  cat("\nMatrix multiplication is not associative.\n\n")
}

# Test Distributive Property: A*(B+C) = A*B + A*C
cat("Distributive Property (A * (B + C) = A * B + A * C):\n")
left_distributive1 <- A %*% (B + C)
right_distributive1 <- (A %*% B) + (A %*% C)
cat("\nA * (B + C):\n"); print(left_distributive1)
cat("\nA * B + A * C:\n"); print(right_distributive1)
if (identical(left_distributive1, right_distributive1)) {
  cat("\nDistributive property (A * (B + C) = A * B + A * C) holds.\n\n")
} else {
```

```
  cat("\nDistributive property does not hold for A * (B + C) = A * B + A * C.\n\n")
}

# Test Distributive Property: (A+B)*C = A*C + B*C
cat("Distributive Property ((A + B) * C = A * C + B * C):\n")
left_distributive2 <- (A + B) %*% C
right_distributive2 <- (A %*% C) + (B %*% C)
cat("\n(A + B) * C:\n"); print(left_distributive2)
cat("\nA * C + B * C:\n"); print(right_distributive2)
if (identical(left_distributive2, right_distributive2)) {
  cat("\nDistributive property ((A + B) * C = A * C + B * C) holds.\n")
} else {
  cat("\nDistributive property does not hold for (A + B) * C = A * C + B * C.\n")
}
```

Output ::

```
[AmitSinghal@nixos: ~]$ Rscript exp4.r
Amit Singhal - 11614802722

Commutative Property:

A * B:
     [,1] [,2]
[1,]   19   22
[2,]   43   50

B * A:
     [,1] [,2]
[1,]   11   16
[2,]   23   34

Matrix multiplication is not commutative.

Associative Property:

(A * B) * C:
     [,1] [,2]
[1,]  439  494
[2,]  983 1106

A * (B * C):
     [,1] [,2]
[1,]  439  494
[2,]  983 1106

Matrix multiplication is associative.
```

```
Distributive Property (A * (B + C) = A * B + A * C):

A * (B + C):
     [,1] [,2]
[1,]   33   38
[2,]   75   86

A * B + A * C:
     [,1] [,2]
[1,]   33   38
[2,]   75   86

Distributive property (A * (B + C) = A * B + A * C) holds.

Distributive Property ((A + B) * C = A * C + B * C):

(A + B) * C:
     [,1] [,2]
[1,]   81   88
[2,]  179  194

A * C + B * C:
     [,1] [,2]
[1,]   81   88
[2,]  179  194

Distributive property ((A + B) * C = A * C + B * C) holds.
```

LAB EXERCISE - 5

AIM :: Exercises to find the reduced row echelon form of a matrix in Scilab.

Scilab Code ::

```
clc;
clear;
disp("Amit Singhal - 11614802722")

function R = rref(A)
  [m, n] = size(A);
  R = A;
  lead = 1;
  for r = 1:m
    if lead > n then
      return;
    end
    i = r;
    while R(i, lead) == 0
      i = i + 1;
      if i > m then
        i = r;
        lead = lead + 1;
        if lead > n then
          return;
        end
      end
    end
    temp = R(i, :);
    R(i, :) = R(r, :);
    R(r, :) = temp;
    R(r, :) = R(r, :) / R(r, lead);
    for i = 1:m
      if i <> r then
        R(i, :) = R(i, :) - R(i, lead) * R(r, :);
      end
    end
    lead = lead + 1;
  end
endfunction

printf("\n");
n = input("Enter number of rows: ");
```

```
m = input("Enter number of columns: ");
A = zeros(n, m);

disp("Enter elements of the matrix row-wise:");
for i = 1:n
  for j = 1:m
    A(i, j) = input("A(" + string(i) + "," + string(j) + ") = ");
  end
end

disp("Original Matrix A:");
disp(A);
R = rref(A);

printf("\n");
disp("Reduced Row Echelon Form (RREF) of A:");
disp(R);
```

Output ::

```
  "Amit Singhal - 11614802722"

Enter number of rows: 3

Enter number of columns: 3

  "Enter elements of the matrix row-wise:"
A(1,1) = 2

A(1,2) = 5

A(1,3) = 4

A(2,1) = 1

A(2,2) = 3

A(2,3) = 6

A(3,1) = 5

A(3,2) = 7

A(3,3) = 8

  "Original Matrix A:"
   2.   5.   4.
   1.   3.   6.
   5.   7.   8.

  "Reduced Row Echelon Form (RREF) of A:"
   1.   0.   0.
   0.   1.   0.
   0.   0.   1.
```

LAB EXERCISE - 6

AIM :: Exercises to plot the functions and to find its first and second derivatives in Scilab.

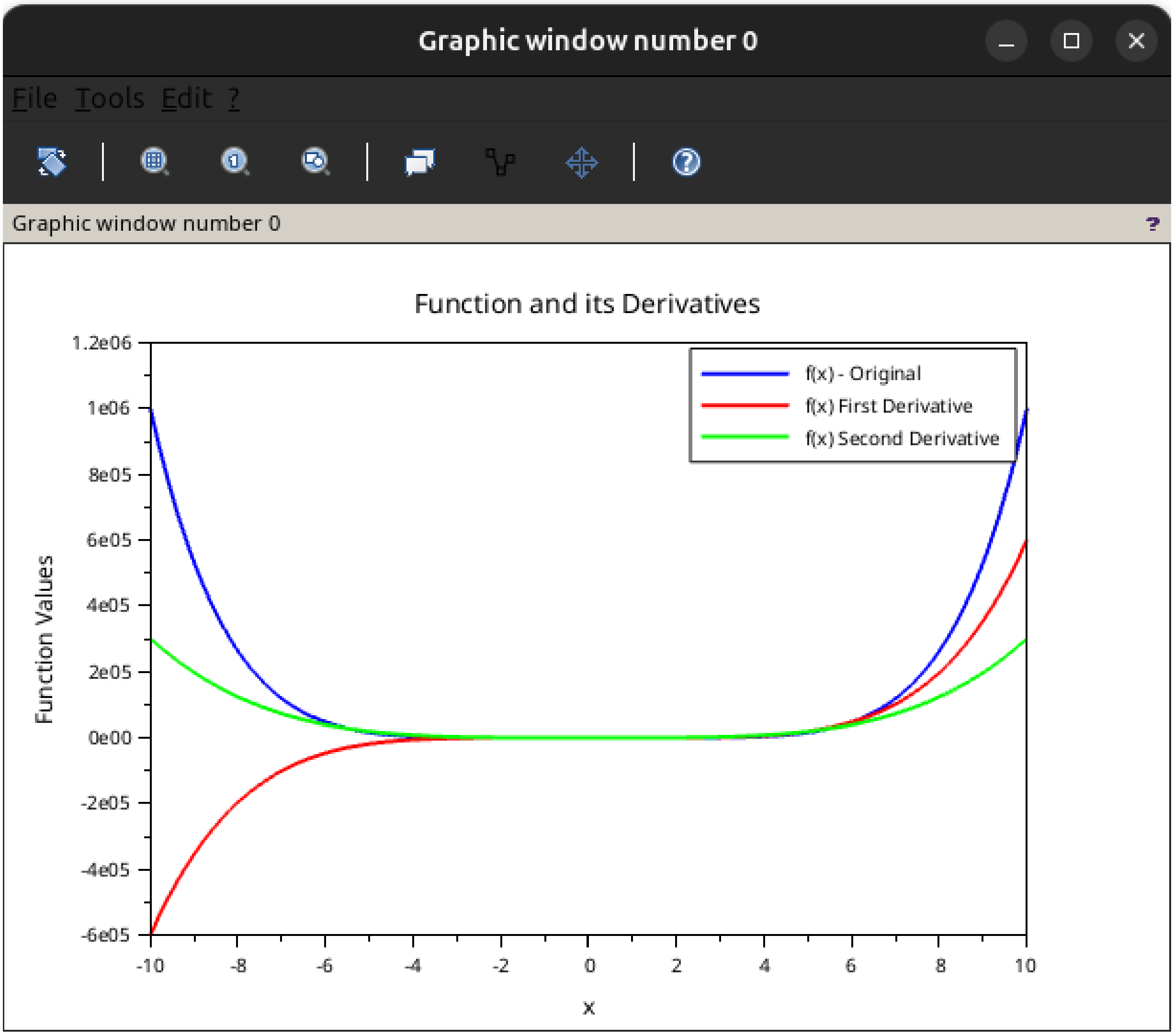
Scilab Code ::

```
clc;
clear;
disp("Amit Singhal - 11614802722")

disp("Enter the function in terms of x (e.g., x^2 + 3*x + 5): ");
func_str = input("Function f(x) = ", "s");

x = poly(0, "x");

func = evstr(func_str);

first_derivative = derivat(func);
second_derivative = derivat(first_derivative);

disp("First Derivative f(x): " + string(first_derivative));
disp("Second Derivative f(x): " + string(second_derivative));

x_vals = linspace(-10, 10, 100);

// Ensure function and derivatives are properly evaluated
y_vals = horner(func, x_vals');
y_first_derivative = horner(first_derivative, x_vals');
y_second_derivative = horner(second_derivative, x_vals');

// Create the plot
clf;
plot(x_vals, y_vals, "b", "LineWidth", 2);

// Disable auto-clear so that new plots add to the current graph
a = gca();
a.auto_clear = "off";

plot(x_vals, y_first_derivative, "r", "LineWidth", 2);
```

```
plot(x_vals, y_second_derivative, "g", "LineWidth", 2);

// Add labels and title
xlabel("x");
ylabel("Function Values");
title("Function and its Derivatives");

// Create legend for identification
legend(["f(x) - Original", "f(x) First Derivative", "f(x) Second Derivative"]);

// Show grid
grid on;
```

Output ::

```
  "Amit Singhal - 11614802722"
  "Enter the function in terms of x (e.g., x^2 + 3*x + 5): "
Function f(x) = x^6 + 4*x + 4

  "First Derivative f(x): 4 +6x^5"
  "Second Derivative f(x): 30x^4"
```



LAB EXERCISE - 7

AIM :: Exercises to present the data as a frequency table in SPSS.

Scilab Code ::

```
data = [23, 45, 23, 45, 23, 56, 56, 23, 45, 56, 23, 45];
[frequencies, edges] = hist(data, unique(data));
disp("Frequency Table using hist():");
disp(frequencies);
unique_values = unique(data);
disp("Unique values:");
disp(unique_values);
frequency_table = zeros(1, length(unique_values));
for i = 1:length(unique_values)
frequency_table(i) = sum(data == unique_values(i));
value
end
disp("Frequency Table using unique() and count():");
disp(frequency_table);
```

Output ::

```
grep: warning: stray \ before -
Scilab 6.1.1 (Jul 15 2021, 14:04:46)

  "Unique values:"

   23.   45.   56.

  "Frequency Table using unique() and count():"

   5.   4.   3.
```

R Code ::

```
data <- c(23, 45, 23, 45, 23, 56, 56, 23, 45, 56, 23, 45)frequency_table <- table(data)
cat("Frequency Table using table():\n")
print(frequency_table)
cat("\nSummary using summary():\n")
print(summary(data))
```

Output ::

```
Frequency Table using table():
data
23 45 56
 5  4  3

Summary using summary():
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  23.00   23.00   45.00   38.58   47.75   56.00
```

LAB EXERCISE - 8

AIM :: Exercises to find the outliers in a dataset in SPSS.

Scilab Code ::

```
data = [100, 102, 104, 120, 130, 130, 150, 200, 250, 300];
boxplot(data);
mean_data = mean(data);
std_data = stdev(data);
z_scores = (data - mean_data) / std_data;
outliers_z = find(abs(z_scores) > 3);
disp("Outliers based on Z-scores:");
disp(data(outliers_z));
Q1 = median(data(1:round(end/2)));
Q3 = median(data(round(end/2)+1:end));
IQR = Q3 - Q1;
lower_bound = Q1 - 1.5 * IQR;
upper_bound = Q3 + 1.5 * IQR;
outliers_iqr = find(data < lower_bound | data > upper_bound);disp("Outliers based on IQR:");
disp(data(outliers_iqr));
```

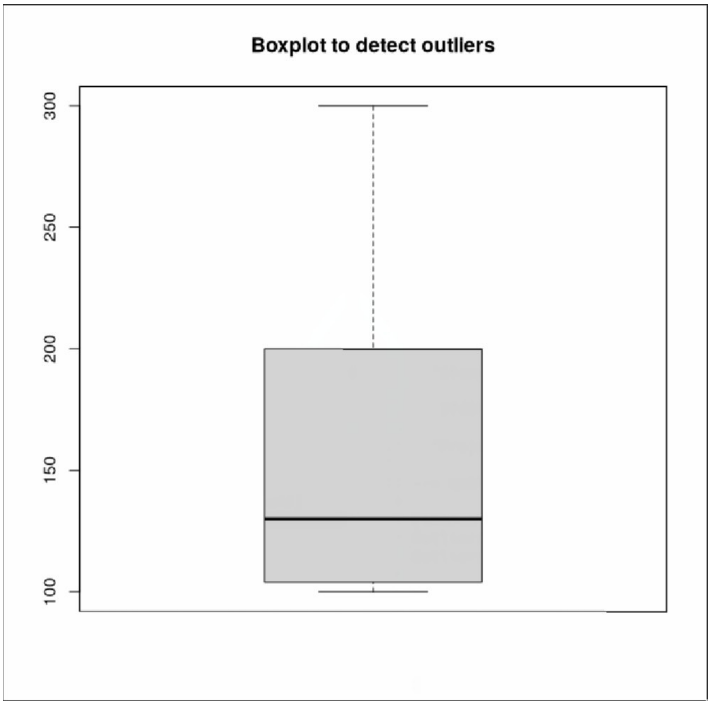
Output ::

```
Startup execution:
  loading initial environment

  "Outliers based on Z-scores:"

    []

  "Outliers based on IQR:"

   100.   102.   104.   120.   130.   130.   150.   200.   250.   300.

--> |
```

R Code ::

```
data <- c(23, 45, 23, 45, 23, 56, 56, 23, 45, 56, 23, 45)frequency_table <- table(data)
cat("Frequency Table using table():\n")
print(frequency_table)
cat("\nSummary using summary():\n")
print(summary(data))data <- c(100, 102, 104, 120, 130, 130, 150, 200, 250, 300)
boxplot(data, main="Boxplot to detect outliers") # Outliers are shown as dots outside
the whiskers
z_scores <- (data - mean(data)) / sd(data)
outliers_z <- data[abs(z_scores) > 3]
cat("Outliers based on Z-scores:", outliers_z, "\n")
Q1 <- quantile(data, 0.25)
Q3 <- quantile(data, 0.75)
IQR <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR
outliers_iqr <- data[data < lower_bound | data > upper_bound]
cat("Outliers based on IQR:", outliers_iqr, "\n")
```

Output ::




```
Z-scores: -0.8489644 -0.8199895 -0.7910146 -0.5592154 -0.414341 -0.
414341 -0.124592 0.5997803 1.324153 2.048525
Outliers based on Z-scores:
Q1: 108
Q3: 187.5
IQR: 79.5
Lower Bound: -11.25
Upper Bound: 306.75
Outliers based on IQR:
```

LAB EXERCISE - 9

AIM :: Exercises to find the most risky project out of two mutually exclusive projects in SPSS

Scilab Code ::

```
cash_flow_A = [100000, 150000, 200000];
cash_flow_B = [90000, 120000, 180000];
prob_A = [0.3, 0.4, 0.3];
prob_B = [0.4, 0.3, 0.3];
EV_A = sum(cash_flow_A .* prob_A);
EV_B = sum(cash_flow_B .* prob_B);
disp("Expected Value of Project A:");
disp(EV_A);
disp("Expected Value of Project B:");
disp(EV_B);
variance_A = sum((cash_flow_A - EV_A).^2 .* prob_A);
SD_A = sqrt(variance_A);
disp("Standard Deviation of Project A:");
disp(SD_A);
variance_B = sum((cash_flow_B - EV_B).^2 .* prob_B);
SD_B = sqrt(variance_B);disp("Standard Deviation of Project B:");
disp(SD_B);
if SD_A > SD_B then
disp("Project A is riskier");
else
disp("Project B is riskier");
end
```

Output ::

```
grep: warning: stray \ before -
Scilab 6.1.1 (Jul 15 2021, 14:04:46)

  "Expected Value of Project A:"

   150000.

  "Expected Value of Project B:"

   126000.

  "Standard Deviation of Project A:"

   38729.833

  "Standard Deviation of Project B:"

   37469.988

  "Project A is riskier"
```

R Code ::

```
cash_flow_A <- c(100000, 150000, 200000)
cash_flow_B <- c(90000, 120000, 180000)
prob_A <- c(0.3, 0.4, 0.3)
prob_B <- c(0.4, 0.3, 0.3)
EV_A <- sum(cash_flow_A * prob_A)
EV_B <- sum(cash_flow_B * prob_B)
cat("Expected Value of Project A:", EV_A, "\n")
cat("Expected Value of Project B:", EV_B, "\n")
variance_A <- sum((cash_flow_A - EV_A)^2 * prob_A)
SD_A <- sqrt(variance_A)
cat("Standard Deviation of Project A:", SD_A, "\n")
variance_B <- sum((cash_flow_B - EV_B)^2 * prob_B)
SD_B <- sqrt(variance_B)
cat("Standard Deviation of Project B:", SD_B, "\n")if (SD_A > SD_B) {
cat("Project A is riskier\n")
} else {
cat("Project B is riskier\n")
}
```

Output ::

```
Expected Value of Project A: 150000
Expected Value of Project B: 126000
Standard Deviation of Project A: 38729.83
Standard Deviation of Project B: 37469.99
Project A is riskier
```

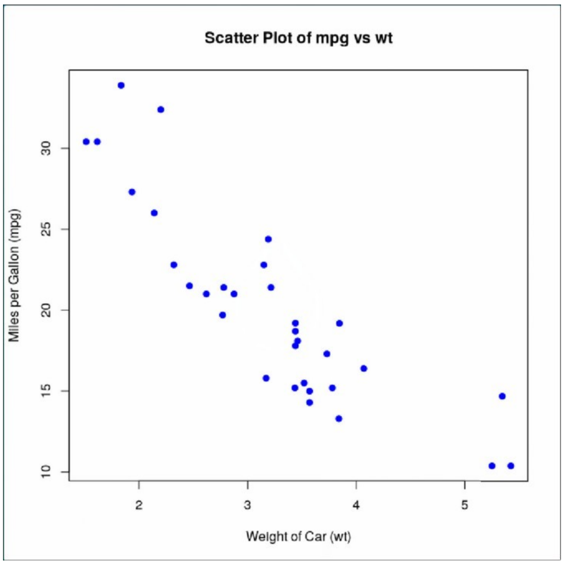
LAB EXERCISE - 10

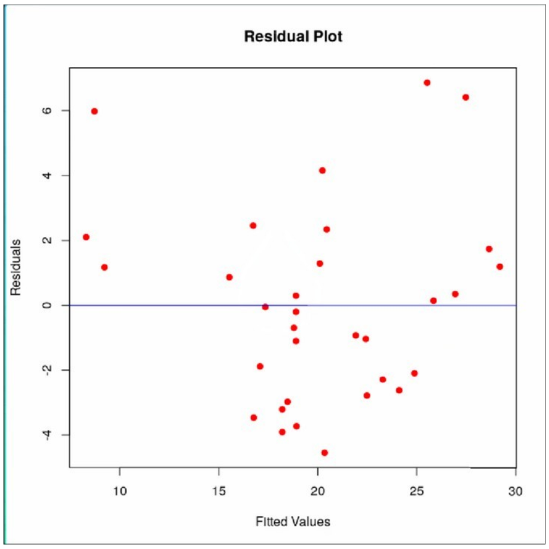
AIM :: Exercises to draw a scatter diagram, residual plots, outliers leverage and influential data points in R

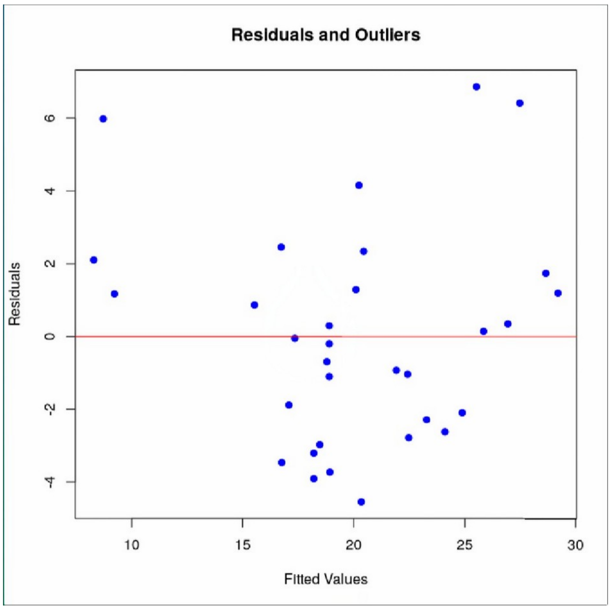
Code ::

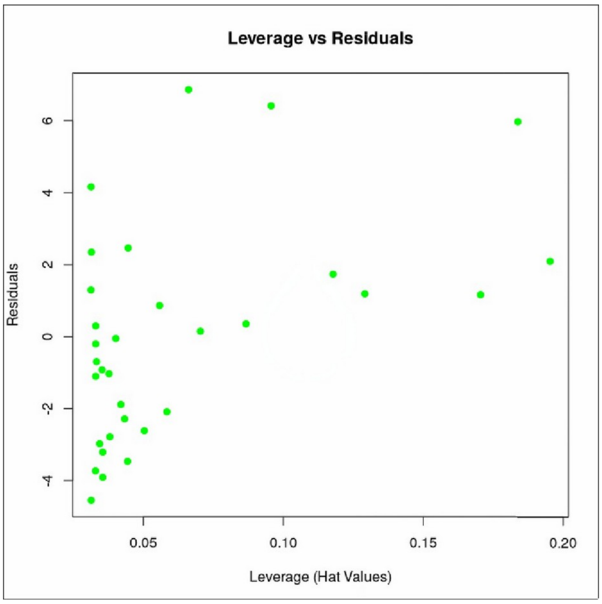
```
library(car)

data(mtcars)

plot(mtcars$wt, mtcars$mpg, main="Scatter Plot of mpg vs wt", xlab="Weight of Car
    (wt)", ylab="Miles per Gallon (mpg)", pch=19, col="blue")

lm_model <- lm(mpg ~ wt, data = mtcars)

residuals <- residuals(lm_model)

plot(lm_model$fitted.values, residuals, main="Residual Plot", xlab="Fitted Values",
     ylab="Residuals", pch=19, col="red")

abline(h = 0, col="blue") # Add a horizontal line at zero

outliers <- outlierTest(lm_model)
cat("Outliers:\n")
print(outliers)

plot(lm_model$fitted.values, residuals(lm_model), main="Residuals and Outliers",
     xlab="Fitted Values", ylab="Residuals", pch=19, col="blue")

abline(h = 0, col="red")

leverage <- hatvalues(lm_model)

cooks_d <- cooks.distance(lm_model)

plot(leverage, residuals(lm_model), main="Leverage vs Residuals", xlab="Leverage
    (Hat Values)", ylab="Residuals", pch=19, col="green")

plot(cooks_d, main="Cook's Distance", ylab="Cook's Distance", pch=19, col="purple")

abline(h = 4/nrow(mtcars), col="red") # Threshold for influence (4/n)

influencePlot(lm_model, main="Influence Plot for mpg vs wt")

high_leverage <- which(leverage > 2 * mean(leverage))

plot(mtcars$wt, mtcars$mpg, main="Scatter Plot with High Leverage Points",
     xlab="Weight (wt)", ylab="Miles per Gallon (mpg)", pch=19, col="blue")


points(mtcars$wt[high_leverage], mtcars$mpg[high_leverage], col="red", pch=19)
```

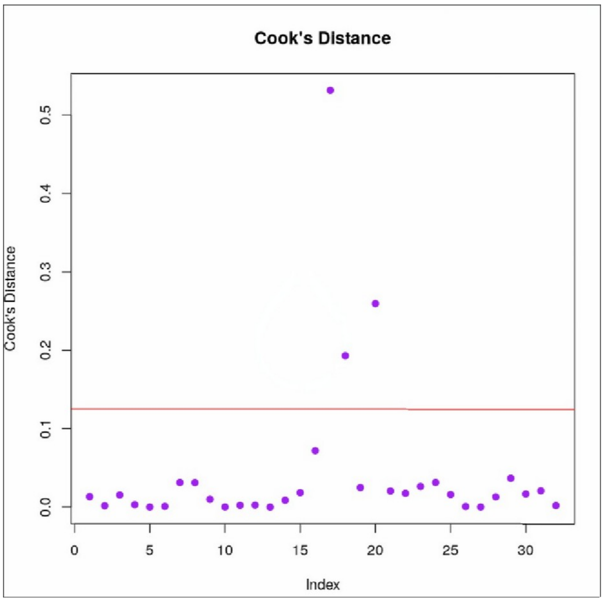
Output ::

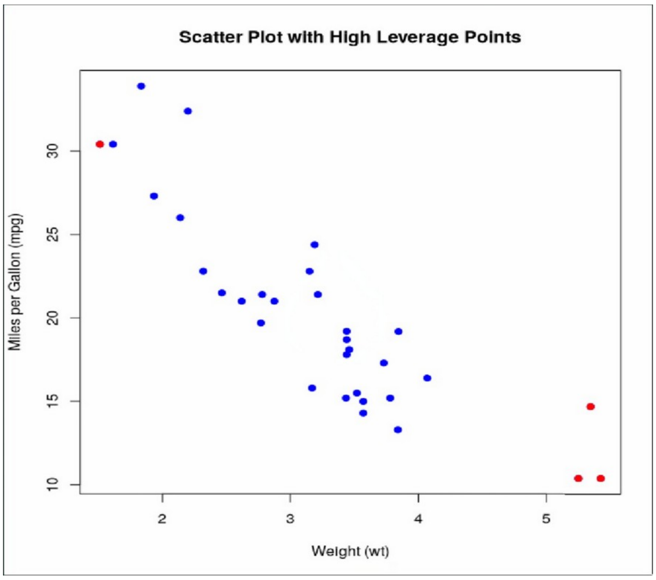

Scatter Plot of mpg vs wt
Miles per Gallon (mpg)
Weight of Car (wt)


Residual Plot
Residuals
Fitted Values


Residuals and Outliers
Residuals
Fitted Values


Leverage vs Residuals
Residuals
Leverage (Hat Values)


Cook's Distance
Cook's Distance
Index


Scatter Plot with High Leverage Points
Miles per Gallon (mpg)
Weight (wt)

LAB EXERCISE - 11

AIM :: Exercises to calculate correlation using R

Code ::

```
clc; clear;

// Correlation manually

// Data
X = [1, 2, 3, 4, 5];
Y = [2, 4, 5, 4, 6];

// means of X and Y
mean_X = mean(X);
mean_Y = mean(Y);

//Calculate the numerator and denominator

// Covariance part
numerator = sum((X - mean_X).* (Y - mean_Y));

// Product of standard deviations
denominator = sqrt(sum((X - mean_X).^2) sum((Y – mean_Y).^2));

// correlation coefficient
r_manual = numerator / denominator;

// Display the result
mprintf("Manually Calculated Correlation Coefficient (r) = %.2f\n", r_manual);

//Interpretation

if r_manual > 0 then
mprintf("=> Positive Correlation\n");

elseif r_manual <0 then
mprintf("=> Negative Correlation\n");

else
mprintf("=> No Correlation\n");

end
```

Output ::

```
Scilab 2025.0.0 Console

Manually Calculated Correlation Coefficient (r) = 0.85
=> Positive Correlation

-->
```

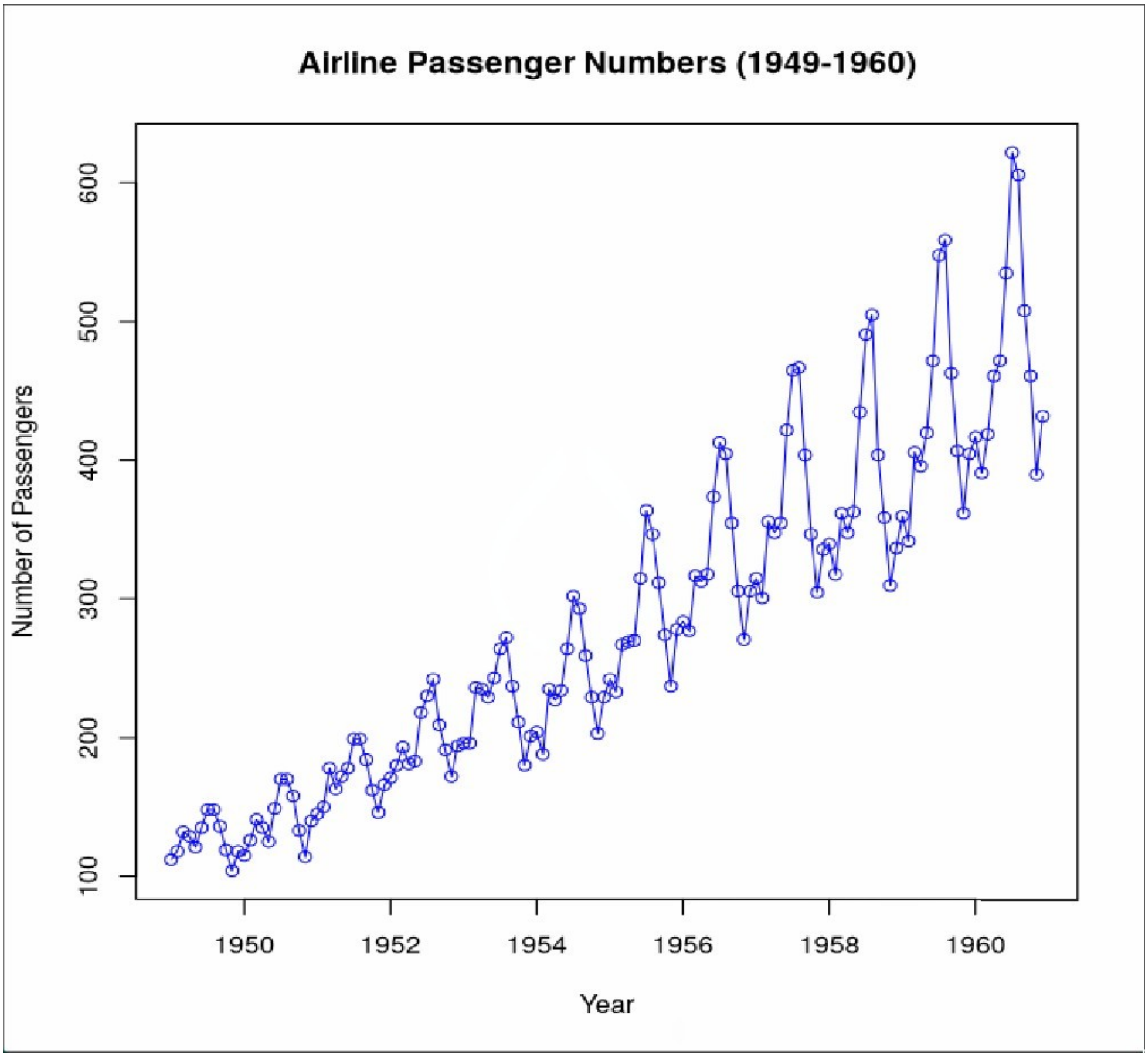
LAB EXERCISE - 12

AIM :: Exercises to implement Time Series using R

Code ::

```
data("AirPassengers")
plot(AirPassengers, main="Airline Passenger Numbers (1949-1960)",
      xlab="Year", ylab="Number of Passengers", col="blue", type="o")
```

Output ::



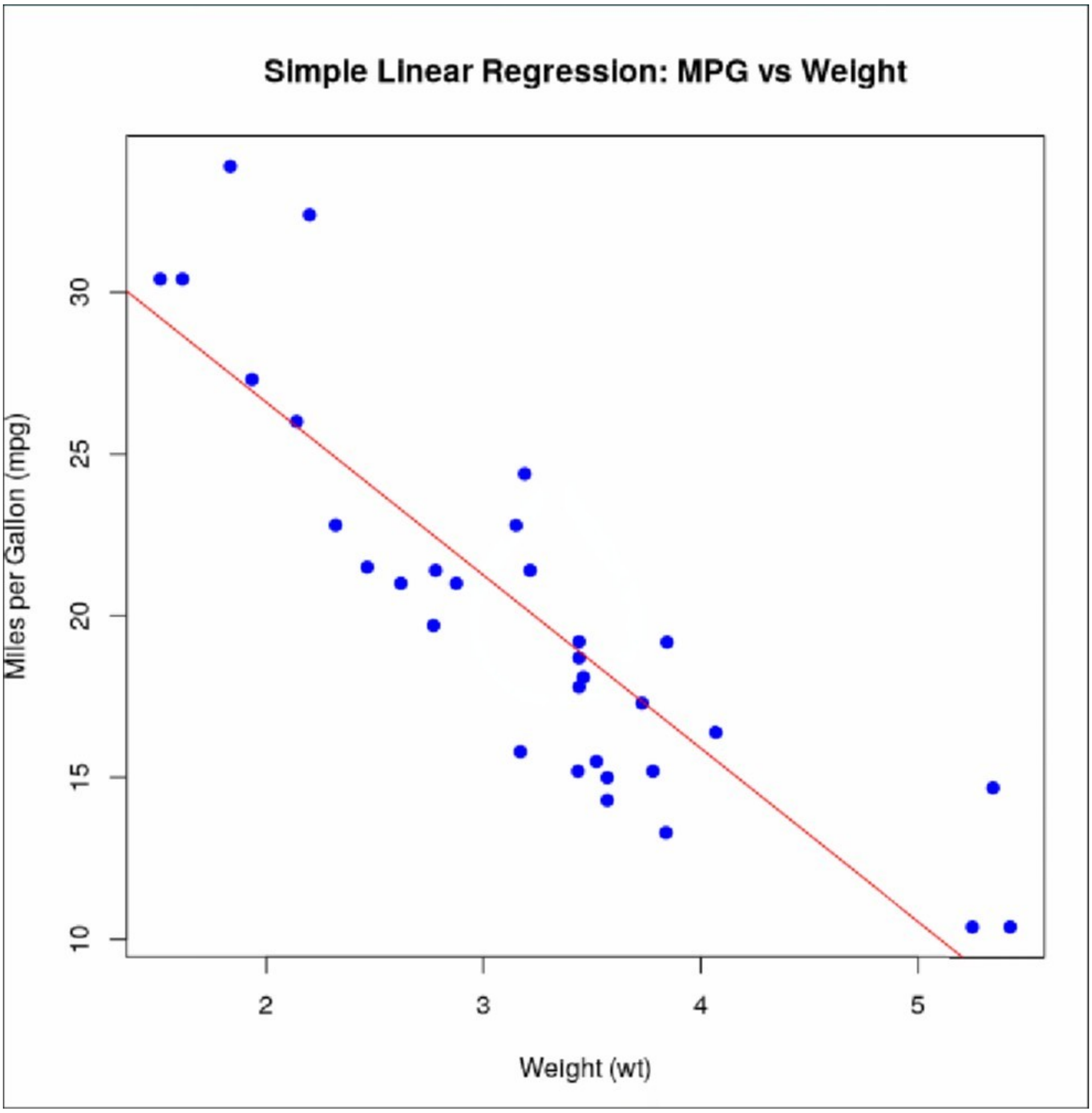
LAB EXERCISE - 13

AIM :: Exercises to implement Linear Regression using R

Code ::

```
data(mtcars)

lm_model <- lm(mpg ~ wt, data = mtcars)

summary(lm_model)

plot(mtcars$wt, mtcars$mpg, main = "Simple Linear Regression: MPG vs Weight",
     xlab = "Weight (wt)", ylab = "Miles per Gallon (mpg)", pch = 19, col = "blue")

abline(lm_model, col = "red")
```

Output ::



LAB EXERCISE - 14

AIM :: Exercises to implement concepts of probability and distributions R

Code ::

```r
normal_samples <- rnorm(1000, mean = 0, sd = 1)

pdf_1 <- dnorm(1, mean = 0, sd = 1)
cat("PDF at x = 1:", pdf_1, "\n")

hist(normal_samples, main="Histogram of Normal Distribution", xlab="Values",
col="lightgreen", breaks=20)

cdf_1 <- pnorm(1, mean = 0, sd = 1)
cat("CDF at x = 1:", cdf_1, "\n")

quantiles <- qnorm(c(0.25, 0.50, 0.75), mean = 0, sd = 1)
cat("Quantiles at 0.25, 0.50, 0.75:", quantiles, "\n")
```

Output ::

```
PDF at x = 1: 0.2419707
CDF at x = 1: 0.8413447
Quantiles at 0.25, 0.50, 0.75: -0.6744898 0 0.6744898
```