PSG INSTITUTE OF TECHNOLOGY AND APPLIED RESEARCH

INTRODUCTION TO MATLAB

prepared by

Mr M Senthil Vel, Assistant Professor, Senior Grade, Mechanical

Mr S Ravikrishna, Assistant Professor, Selection Grade, EEE

DEPARTMENTS OF MECHANICAL AND EEE

Table of Contents

Introduction to MATLAB Environment

1. Welcome to MATLAB

MATLAB (Matrix Laboratory) is a high-level language and interactive environment for numerical computation, visualization, and programming.

You can use MATLAB to solve mathematical problems, analyze data, and visualize results interactively.

2. Exploring the MATLAB Interface

Run the following commands to explore the Command Window and Workspace.

Type each command in the Command Window

```
% Basic operations
2 + 2            % Addition
```

```
ans = 4
```

```
5 - 3            % Subtraction
```

```
ans = 2
```

```
3 * 4            % Multiplication
```

```
ans = 12
```

```
12 / 4           % Division
```

```
ans = 3
```

```
sqrt(16)         % Square root
```

```
ans = 4
```

Question for Reflection: Can you calculate the result of (5 + 3) * (8 / 2)? Try it in MATLAB!

3. Working with Variables

Assign values to variables and perform operations.

```

% Assigning values
a = 10;          % Assign 10 to variable 'a'
b = 20;          % Assign 20 to variable 'b'

% Perform operations with variables
sum = a + b      % Addition
```

```
sum = 30
```

```
difference = b - a  % Subtraction
```

```
difference = 10
```

```matlab
product = a * b  % Multiplication
```

```
product = 200
```

4. Rules for Variable Naming in MATLAB

When naming variables in MATLAB, you need to follow specific rules and best practices to avoid errors and ensure code readability.

Must start with a letter (A-Z or a-z).

- □ `velocity`
- □ `1speed` (Invalid: starts with a number)

Can be followed by letters, digits, or underscores.

- □ `temp1`, `mass_kg`
- □ `mass-kg` (Invalid: contains `-` which is not allowed)

Cannot use spaces.

- □ `my_variable`
- □ `my variable` (Invalid: spaces are not allowed)

Cannot use MATLAB reserved keywords (e.g., `if`, `for`, `while`, `end`, `function`).

- □ `loop_counter`
- □ `for` (Invalid: `for` is a reserved keyword)

Case-sensitive.

- `Pressure`, `pressure`, and `PRESSURE` are treated as different variables.

Best Practices

Use meaningful names instead of single letters.

- □ `temperature_Celsius`
- □ `t` (Avoid using non-descriptive names)

Use underscores (_) to improve readability instead of camel case.

- □ `total_force`
- □ `totalForce` (Allowed but not preferred in MATLAB)

Avoid very long names.

- □ `velocity_x`

- □ `velocity_of_the_particle_in_x_direction` (Too long)

Use consistent naming across your code.

- **Avoid starting with `i` or `j`** if not used as imaginary numbers.
- □ `index` instead of `i` (since `i` is commonly used for imaginary numbers in MATLAB)

MATLAB allows variable names up to 63 characters, but keep them concise.

Introduction to Plotting in MATLAB

Objective: Understand the basics of plotting in MATLAB to visualize mathematical functions and datasets effectively.

1. Why Plotting Matters?

Visualization is a powerful way to:

- Understand relationships between variables.
- Solve problems interactively.
- Communicate complex data effectively.

In this session, we will explore plotting as a tool to visualize solutions of linear systems.

2. Creating Your First Plot

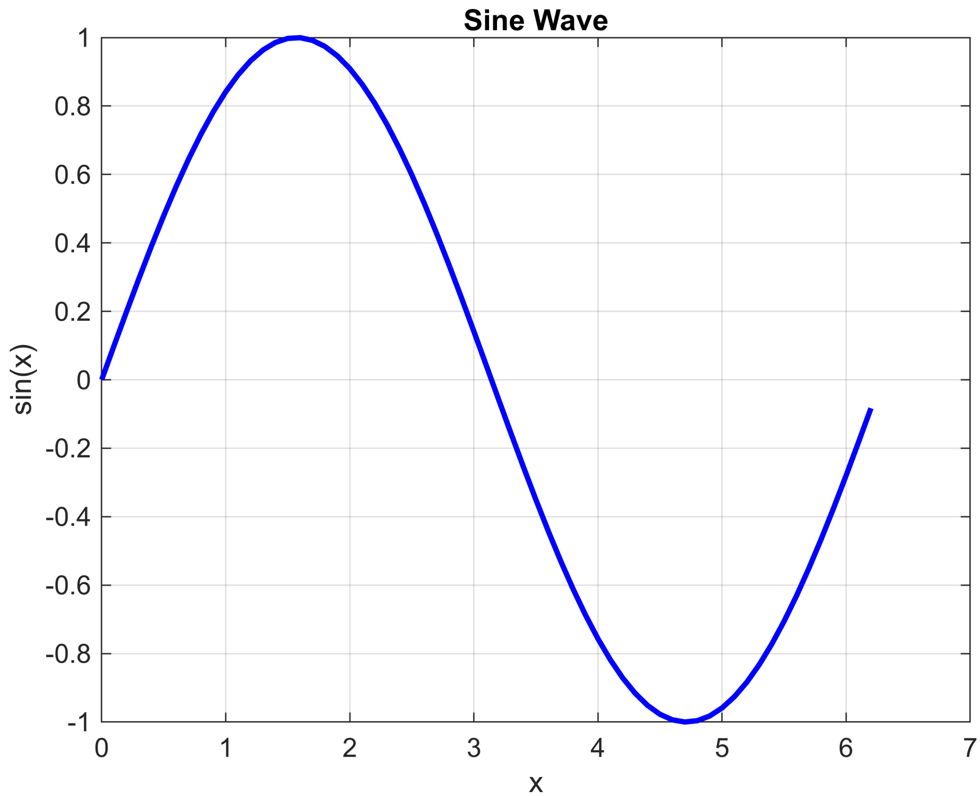
Let's start with a simple sine wave plot.

```
% Define x values
x = 0:0.1:2*pi;          % Create a range of values from 0 to 2*pi

% Define y values
y = sin(x);              % Compute sine of each x value

% Create the plot
figure;
plot(x, y, 'b', 'LineWidth', 2); % Blue line with a thickness of 2
title('Sine Wave');              % Add a title
xlabel('x');                     % Label the x-axis
ylabel('sin(x)');                % Label the y-axis
grid on;                         % Add a grid

xlim([0.00 7.00])
ylim([-1.00 1.00])
```



3. Adding Multiple Plots

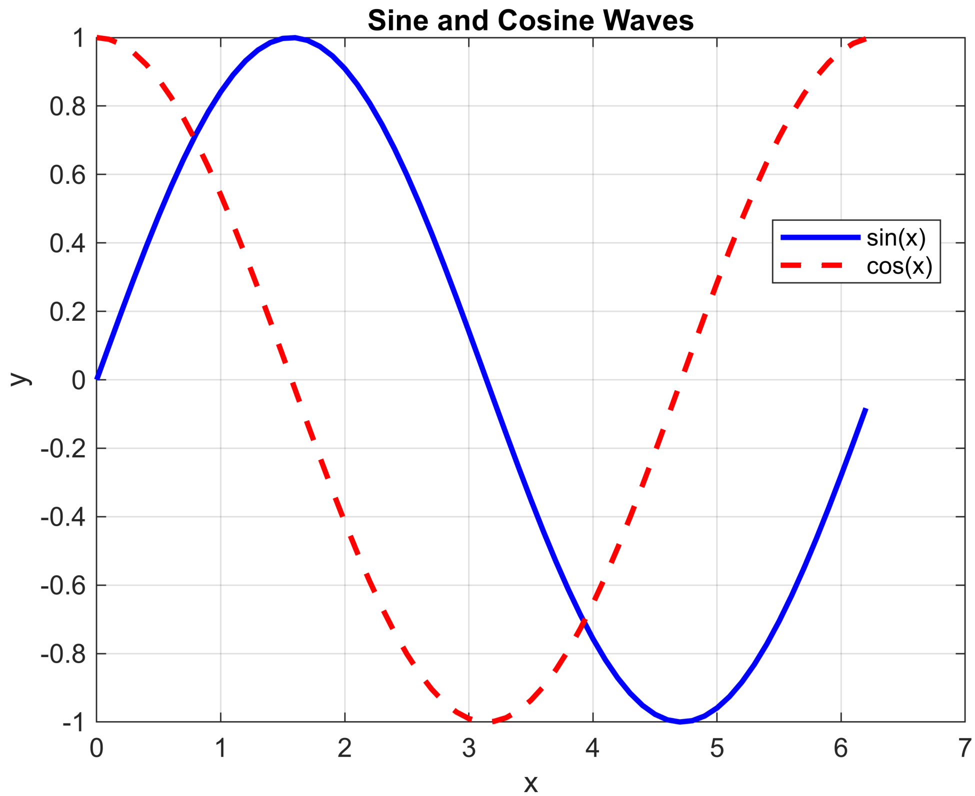
You can plot multiple lines on the same figure using the `hold on` command.

```
% Define x values
x = 0:0.1:2*pi;

% Define y values for sine and cosine
y1 = sin(x);
y2 = cos(x);

% Plot both sine and cosine waves
figure;
plot(x, y1, 'b', 'LineWidth', 2); hold on;
plot(x, y2, 'r--', 'LineWidth', 2); % Dashed red line
hold off;

% Add title, labels, and legend
title('Sine and Cosine Waves');
xlabel('x');
ylabel('y');
legend('sin(x)', 'cos(x)', 'Location', 'Best');
grid on;
```

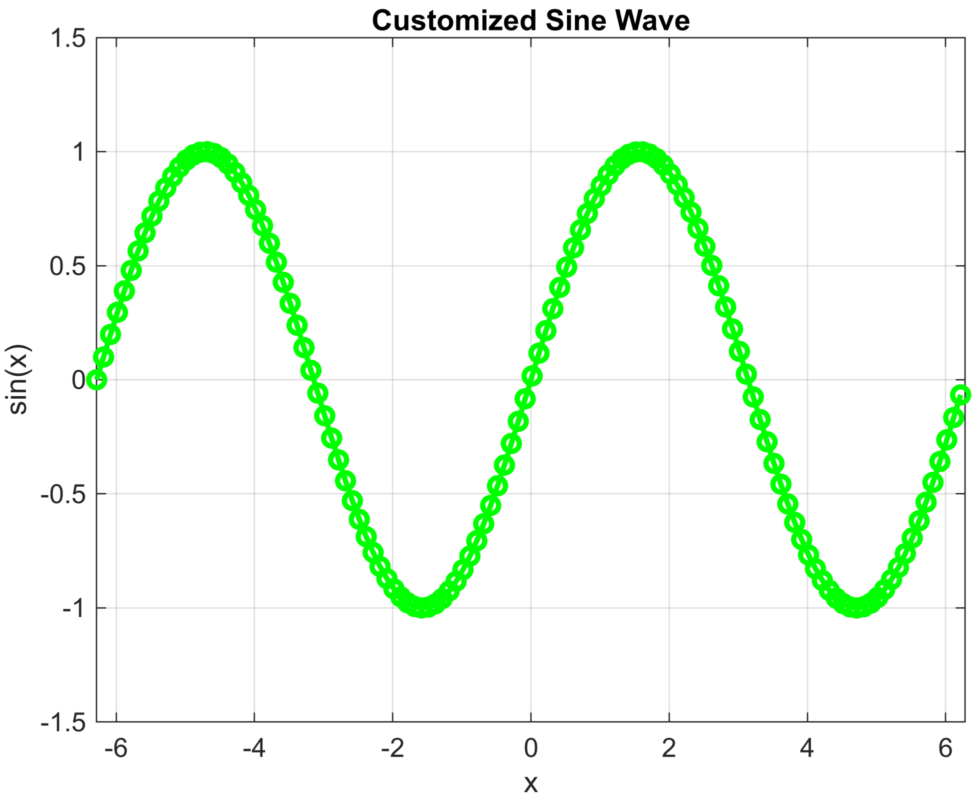


4. Customizing Plots

MATLAB allows customization of plots for better clarity.

```
% Define x values and a function
x = -2*pi:0.1:2*pi;
y = sin(x);

% Create a customized plot
figure;
plot(x, y, 'g-o', 'LineWidth', 2, 'MarkerSize', 6); % Green line with circular
markers
title('Customized Sine Wave');                   % Title
xlabel('x');                                     % X-axis label
ylabel('sin(x)');                                % Y-axis label
grid on;                                         % Add grid
xlim([-2*pi, 2*pi]);                             % Set x-axis limits
ylim([-1.5, 1.5]);                               % Set y-axis limits
```



5. Using Subplots

You can display multiple plots in the same figure using subplots.

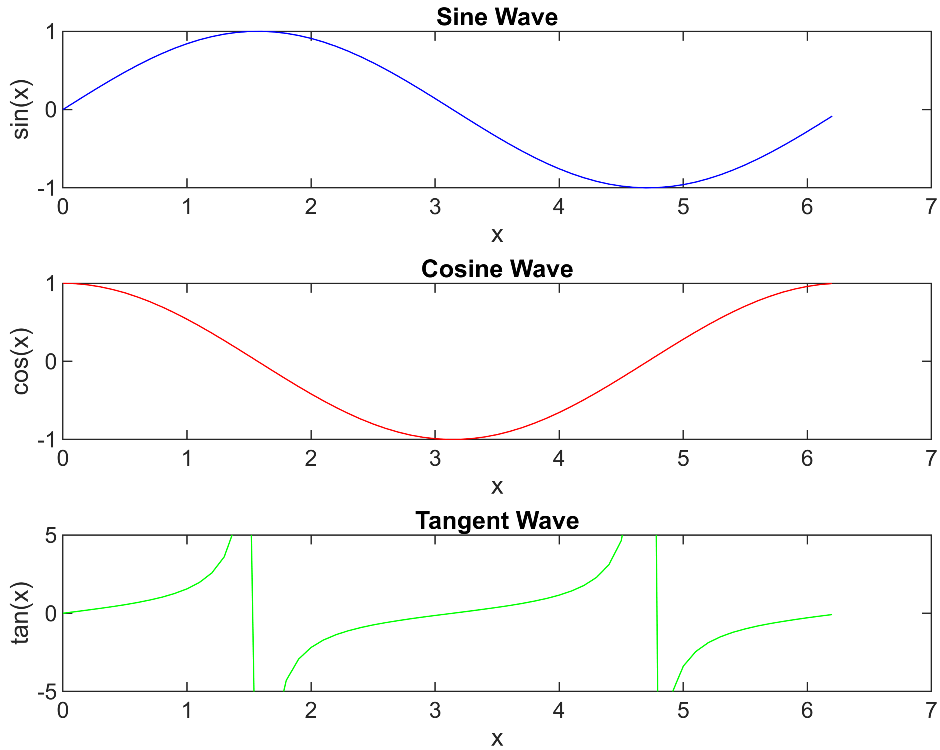
```
% Define x values
x = 0:0.1:2*pi;

% Compute y values for sine, cosine, and tangent
y1 = sin(x);
y2 = cos(x);
y3 = tan(x);

% Create subplots
figure;
subplot(3, 1, 1);  % First subplot
plot(x, y1, 'b');
title('Sine Wave');
xlabel('x');
ylabel('sin(x)');

subplot(3, 1, 2);  % Second subplot
plot(x, y2, 'r');
title('Cosine Wave');
xlabel('x');
ylabel('cos(x)');
```

```
subplot(3, 1, 3);  % Third subplot
plot(x, y3, 'g');
title('Tangent Wave');
xlabel('x');
ylabel('tan(x)');
ylim([-5, 5]); % Limit y-axis to avoid large values
```



5. Basic Help and Documentation

Use the help command to learn about any MATLAB function. For example:

```
help plot      % Learn about the 'plot' function
```

```
plot   Linear plot.
   plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
   then the vector is plotted versus the rows or columns of the matrix,
   whichever line up.  If X is a scalar and Y is a vector, disconnected
   line objects are created and plotted as discrete points vertically at
   X.

   plot(Y) plots the columns of Y versus their index.
   If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
   In all other uses of plot, the imaginary part is ignored.

   Various line types, plot symbols and colors may be obtained with
   plot(X,Y,S) where S is a character string made from one element
```

from any or all the following 3 columns:

```
       b     blue          .     point              -     solid
       g     green         o     circle             :     dotted
       r     red           x     x-mark             -.    dashdot
       c     cyan          +     plus               --    dashed
       m     magenta       *     star             (none)  no line
       y     yellow        s     square
       k     black         d     diamond
       w     white         v     triangle (down)
                           ^     triangle (up)
                           <     triangle (left)
                           >     triangle (right)
                           p     pentagram
                           h     hexagram
```

For example, **plot**(X,Y,'c+:') plots a cyan dotted line with a plus at each data point; **plot**(X,Y,'bd') plots blue diamond at each data point but does not draw any line.

plot(TBL,XVAR,YVAR) plots the variables xvar and yvar from the table tbl. To plot one data set, specify one variable for xvar and one variable for yvar. To plot multiple data sets, specify multiple variables for xvar, yvar, or both. If both arguments specify multiple variables, they must specify the same number of variables

plot(TBL,YVAR) plots the specified variable from the table against the row indices in the table. If the table is a timetable, the specified variable is plotted against the row times from the timetable.

plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by the (X,Y,S) triples, where the X's and Y's are vectors or matrices and the S's are strings.

For example, **plot**(X,Y,'y-',X,Y,'go') plots the data twice, with a solid yellow line interpolating green circles at the data points.

The **plot** command, if no color is specified, makes automatic use of the colors specified by the axes ColorOrder property. By default, **plot** cycles through the colors in the ColorOrder property. For monochrome systems, **plot** cycles over the axes LineStyleOrder property.

Note that RGB colors in the ColorOrder property may differ from similarly-named colors in the (X,Y,S) triples. For example, the second axes ColorOrder property is medium green with RGB [0 .5 0], while **plot**(X,Y,'g') plots a green line with RGB [0 1 0].

If you do not specify a marker type, **plot** uses no marker.
If you do not specify a line style, **plot** uses a solid line.

plot(AX,...) plots into the axes with handle AX.

plot returns a column vector of handles to lineseries objects, one handle per plotted line.

The X,Y pairs, or X,Y,S triples, can be followed by parameter/value pairs to specify additional properties of the lines. For example, **plot**(X,Y,'LineWidth',2,'Color',[.6 0 0]) will create a plot with a dark red line width of 2 points.

Example
```
   x = -pi:pi/10:pi;
   y = tan(sin(x)) - sin(tan(x));
   plot(x,y,'--rs','LineWidth',2,...
```

```
                    'MarkerEdgeColor','k',...
                    'MarkerFaceColor','g',...
                    'MarkerSize',10)

    See also title, xlabel, ylabel, xlim, ylim, legend, hold, gca, yyaxis,
    plot3, semilogx, semilogy, loglog, tiledlayout, hold, legend, scatter

    Documentation for plot
    Other uses of plot
```

6. Saving Your Work

Save your work as a .m file or a Live Script .mlx file.

To save your current code, click on File > Save As, or use the shortcut Ctrl+S.

Matrix Fundamentals in MATLAB

Objective: Learn how to create, manipulate, and perform operations on matrices, solve systems of equations, and compute eigenvalues and eigenvectors with real-world engineering applications.

1. Introduction to Matrices

- A **matrix** is a two-dimensional array of numbers, fundamental in many engineering and mathematical computations.
- MATLAB is designed to handle matrices efficiently, making it an ideal tool for such operations.

2. Creating Matrices

Matrices can be created in various ways:

```
% Row vector (1x4)
row_vec = [1, 2, 3, 4]
```

```
row_vec = 1×4
     1     2     3     4
```

```
% Column vector (4x1)
col_vec = [1; 2; 3; 4]
```

```
col_vec = 4×1
     1
     2
     3
     4
```

```
% 3x3 matrix
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

```
A = 3×3
```

```
     1     2     3
     4     5     6
     7     8     9
```

```
% Identity matrix (3x3)
I = eye(3)
```

```
I = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
% Zero matrix (2x3)
Z = zeros(2, 3)
```

```
Z = 2×3
     0     0     0
     0     0     0
```

```
% Random matrix (3x2)
R = rand(3, 2)
```

```
R = 3×2
    0.6948    0.0344
    0.3171    0.4387
    0.9502    0.3816
```

```
% Diagonal matrix
D = diag([10, 20, 30])
```

```
D = 3×3
    10     0     0
     0    20     0
     0     0    30
```

3. Accessing and Modifying Matrix Elements

You can access specific elements, rows, columns, or submatrices.

```
% Example matrix
B = [10, 20, 30; 40, 50, 60; 70, 80, 90]
```

```
B = 3×3
    10    20    30
    40    50    60
    70    80    90
```

```
% Access a specific element
element = B(2, 3)          % Element in the 2nd row, 3rd column (60)
```

```
element = 60
```

```
% Access a row
row1 = B(1, :)              % Entire first row
```

```
row1 = 1×3
    10    20    30
```

```
% Access a column
col2 = B(:, 2)              % Entire second column
```

```
col2 = 3×1
    20
    50
    80
```

```
% Modify an element
B(3, 1) = 99                % Change the element in the 3rd row, 1st column to 99
```

```
B = 3×3
    10    20    30
    40    50    60
    99    80    90
```

3.1 Logical Indexing in MATLAB

Logical indexing allows accessing or modifying elements of a matrix based on conditions instead of specifying explicit row and column indices.

Examples of Logical Indexing

3.1.1 Extracting Elements Using Logical Conditions

```
% Define a matrix B
B = [10, 20, 30; 40, 50, 60; 70, 80, 90];

% Extract all elements greater than 50
greater_than_50 = B(B > 50);
```

This extracts elements in `B` where the condition `B > 50` is true. The output will be `[60, 70, 80, 90]`.

3.1.2 Modifying Elements Using Logical Indexing

```
% Set all values greater than 50 to 100
B(B > 50) = 100;
```

This modifies only the elements that satisfy the condition. After execution, `B` becomes:

3.1.3 Finding Row and Column Indices Using `find()`

`% Get row and column indices where B is greater than 50`

```
[row, col] = find(B > 50);
```

This returns the row and column positions of elements greater than 50.

3.1.4 Extracting Elements with Multiple Conditions

```
% Find elements between 30 and 80
filtered_values = B(B >= 30 & B <= 80);
```

This extracts elements in the range `[30, 80]` from `B`.

4. Basic Matrix Operations

MATLAB allows you to perform various matrix operations:

```
% Example matrices
C = [1, 2; 3, 4]
```

```
C = 2×2
     1     2
     3     4
```

```
D = [5, 6; 7, 8]
```

```
D = 2×2
     5     6
     7     8
```

```
% Matrix addition
sum_matrix = C + D
```

```
sum_matrix = 2×2
     6     8
    10    12
```

```
% Matrix subtraction
diff_matrix = C - D
```

```
diff_matrix = 2×2
    -4    -4
    -4    -4
```

```
% Matrix multiplication
prod_matrix = C * D        % Standard matrix multiplication
```

```
prod_matrix = 2×2
    19    22
    43    50
```

```
% Element-wise multiplication
elem_mult = C .* D         % Multiply element by element
```

```
elem_mult = 2×2
     5    12
    21    32
```

```
% Transpose of a matrix
```

```
transpose_C = C'
```

```
transpose_C = 2×2
     1     3
     2     4
```

5. Determinants and Inverses

The determinant and inverse of a matrix are essential in various computations.

```
% Example matrix
A = [4, 2; 3, 1];

% Compute determinant
det_A = det(A)
```

```
det_A = -2
```

```
% Compute inverse
inv_A = inv(A)
```

```
inv_A = 2×2
   -0.5000    1.0000
    1.5000   -2.0000
```

```
% Verify: A * inv(A) should be the identity matrix
identity_check = A * inv_A
```

```
identity_check = 2×2
     1     0
     0     1
```

6. Vector operations

6.1 Salar Multiplication with a Vector

Multiply a vector by a scalar to scale its elements.

```
v = [2; 4; 6; 8];  % Column vector
scalar = 3;

result = scalar * v;  % Multiply each element by 3

disp('Original Vector:'); disp(v);
```

```
Original Vector:
     2
     4
     6
     8
```

```
disp('After Scalar Multiplication:'); disp(result);
```

```
After Scalar Multiplication:
     6
    12
    18
    24
```

6.2 Matrix-Vector Multiplication (Ax = b)

Multiply a matrix with a vector to get a transformed vector.

```
A = [2 3; 4 1; 5 -2];  % 3×2 Matrix
x = [1; 2];  % 2×1 Column Vector

b = A * x;  % Matrix-vector multiplication

disp('Matrix A:'); disp(A);
```

```
Matrix A:
     2     3
     4     1
     5    -2
```

```
disp('Vector x:'); disp(x);
```

```
Vector x:
     1
     2
```

```
disp('Result (Ax):'); disp(b);
```

```
Result (Ax):
     8
     6
     1
```

6.3 Selecting and Operating on a Complete Row Using a Vector

Extract a specific row from a matrix as a vector and perform operations.

```
A = [1 2 3; 4 5 6; 7 8 9];  % 3×3 Matrix
rowIndex = 2;  % Select the second row

rowVector = A(rowIndex, :);  % Extract the entire row
newRow = rowVector * 2;  % Multiply the row by 2

disp('Original Matrix A:'); disp(A);
```

```
Original Matrix A:
     1     2     3
     4     5     6
     7     8     9
```

```
disp(['Extracted Row ', num2str(rowIndex), ':']); disp(rowVector);
```

```
Extracted Row 2:
     4     5     6
```

```
disp('After Multiplication:'); disp(newRow);
```

```
After Multiplication:
     8    10    12
```

6.4 Swapping Two Rows Using Vector Indexing

Swap two rows in a matrix using vector operations.

```
A = [1 2 3; 4 5 6; 7 8 9];  % 3×3 Matrix
row1 = 1; row2 = 3;  % Rows to swap

A([row1, row2], :) = A([row2, row1], :);  % Swap rows

disp('Matrix After Row Swap:'); disp(A);
```

```
Matrix After Row Swap:
     7     8     9
     4     5     6
     1     2     3
```

6.5 Adding a Multiple of One Row to Another (Row Operation)

Perform row operations by adding a **scaled version of one row to another** (useful in Gaussian elimination).

```
A = [1 2 3; 4 5 6; 7 8 9];  % 3×3 Matrix
multiplier = -2;  % Scale factor
rowTarget = 3;  % Row to modify
rowSource = 1;  % Row to scale and add

A(rowTarget, :) = A(rowTarget, :) + multiplier * A(rowSource, :);  % Row operation

disp('Matrix After Row Operation:'); disp(A);
```

```
Matrix After Row Operation:
     1     2     3
     4     5     6
     5     4     3
```

7. Conditional and Looping Statements

7.1. Access and Display Elements of a Matrix

This exercise shows **how to loop through a matrix** using nested `for` loops and display each element.

```
% Define a 3x3 matrix
A = [10 20 30; 40 50 60; 70 80 90];

disp('Matrix elements and their indices:');
```

```
Matrix elements and their indices:
```

```

% Loop through rows
for i = 1:3
    % Loop through columns
```

```
    for j = 1:3
        fprintf('Element at A(%d, %d) = %d\n', i, j, A(i, j));
    end
end
```

```
Element at A(1, 1) = 10
Element at A(1, 2) = 20
Element at A(1, 3) = 30
Element at A(2, 1) = 40
Element at A(2, 2) = 50
Element at A(2, 3) = 60
Element at A(3, 1) = 70
Element at A(3, 2) = 80
Element at A(3, 3) = 90
```

7.2. Filling a Matrix Using Nested Loops

This example **constructs a matrix dynamically** by assigning values inside a loop.

```
% Create an empty 3x3 matrix
B = zeros(3,3);

% Fill the matrix with values where each element is i * j
for i = 1:3
    for j = 1:3
        B(i,j) = i * j; % Example: Multiply row index by column index
    end
end

disp('Generated matrix:');
```

```
Generated matrix:
```

```
disp(B);
```

```
     1     2     3
     2     4     6
     3     6     9
```

7.3. Extracting a Specific Row and Column

This shows how to **extract individual rows and columns** using loops.

Extracting a Column

```
% Define a matrix
C = [1 2 3; 4 5 6; 7 8 9];

disp('Extracted column 2:');
```

```
Extracted column 2:
```

```
for i = 1:3
    fprintf('%d\n', C(i,2)); % Printing each element in column 2
end
```

```
2
5
8
```

Extracting a Row

```
disp('Extracted row 1:');
```

```
Extracted row 1:
```

```
for j = 1:3
    fprintf('%d ', C(1,j)); % Printing each element in row 1
end
```

```
1 2 3
```

```
fprintf('\n');
```

7.4. Creating an Identity Matrix Using Nested Loops

This helps **build an identity matrix dynamically**, which is useful in numerical methods.

```
% Create a 4x4 identity matrix
I = zeros(4,4);

for i = 1:4
    for j = 1:4
        if i == j
            I(i,j) = 1;
        end
    end
end
disp('Generated Identity Matrix:');
```

```
Generated Identity Matrix:
```

```
disp(I);
```

```
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

7.5. Swapping Two Rows in a Matrix

Row swapping is a **critical operation** in Gaussian elimination.

```
% Define a matrix
D = [1 2 3; 4 5 6; 7 8 9];

disp('Original Matrix:');
```

```
Original Matrix:
```

```
disp(D);
```

```
     1     2     3
     4     5     6
     7     8     9
```

```

% Swap row 1 and row 3
for j = 1:3
    temp = D(1,j);
    D(1,j) = D(3,j);
    D(3,j) = temp;
end

disp('Matrix after swapping row 1 and row 3:');
```

```
Matrix after swapping row 1 and row 3:
```

```
disp(D);
```

```
     7     8     9
     4     5     6
     1     2     3
```

7.6. Scaling a Row by a Factor

This prepares them for **row transformations** used in Gaussian elimination.

```
% Define a matrix
E = [1 2 3; 4 5 6; 7 8 9];

disp('Original Matrix:');
```

```
Original Matrix:
```

```
disp(E);
```

```
     1     2     3
     4     5     6
     7     8     9
```

```

% Multiply row 2 by 3
for j = 1:3
    E(2, j) = 3 * E(2, j);
end

disp('Matrix after multiplying row 2 by 3:');
```

```
Matrix after multiplying row 2 by 3:
```

```
disp(E);
```

```
    1     2     3
   12    15    18
    7     8     9
```

7.7 Summing Elements of Each Row in a Matrix

Write a MATLAB program that computes the **sum of each row** in a given matrix.

Example Input:

```
A = [1 2 3; 4 5 6; 7 8 9];
```

Expected Output:

Row 1 sum = 6

Row 2 sum = 15

Row 3 sum = 24

Solution Using Loops:

```
A = [1 2 3; 4 5 6; 7 8 9];
[m, n] = size(A); % Get matrix size

for i = 1:m  % Loop through each row
    row_sum = 0;
    for j = 1:n  % Loop through each column
        row_sum = row_sum + A(i, j);
    end
    fprintf('Row %d sum = %d\n', i, row_sum);
end
```

```
Row 1 sum = 6
Row 2 sum = 15
Row 3 sum = 24
```

7.8 Transform a Matrix to Upper Triangular Form

Write a MATLAB program that **converts a matrix into upper triangular form**.

Example Input:

```
A = [2 3 4; 5 6 7; 8 9 10];
```

Expected Output:

Upper Triangular Matrix:

2 3 4

0 6 7

0 0 10

Solution Using Loops:

```
A = [2 3 4; 5 6 7; 8 9 10];
n = size(A,1); % Matrix size

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

disp('Upper Triangular Matrix:');
```

```
Upper Triangular Matrix:
```

```
disp(A);
```

```
    2.0000    3.0000    4.0000
         0   -1.5000   -3.0000
         0         0         0
```

7.9 While loop for Matrix Indexing

This MATLAB program demonstrates **how to use a `while` loop for matrix indexing** by iterating through **each element of a vector**.

```
A = [10, 20, 30, 40, 50];  % Define a row vector
i = 1;  % Start index

while i <= length(A)  % Loop through all elements
    disp(A(i));  % Display the current element
    i = i + 1;  % Move to the next index
end
```

```
    10

    20

    30

    40

    50
```

```

```