PRACTICAL FILE

MODELING AND SIMULATION LAB

(CS 603)

BE CSE 6TH SEM

(GROUP-4)

University Institute of Engineering and Technology (UIET), Panjab University, Chandigarh, India- 160014

Under the guidance of

Priyanka Mam

Department of Computer Science and Engineering

Submitted By

Ojas Arora

Roll No: UE223073

Practical 1

Aim

I. Introduction to MATLAB

II. Working with:

1. Branching statements

2. Loops

3. Datatypes

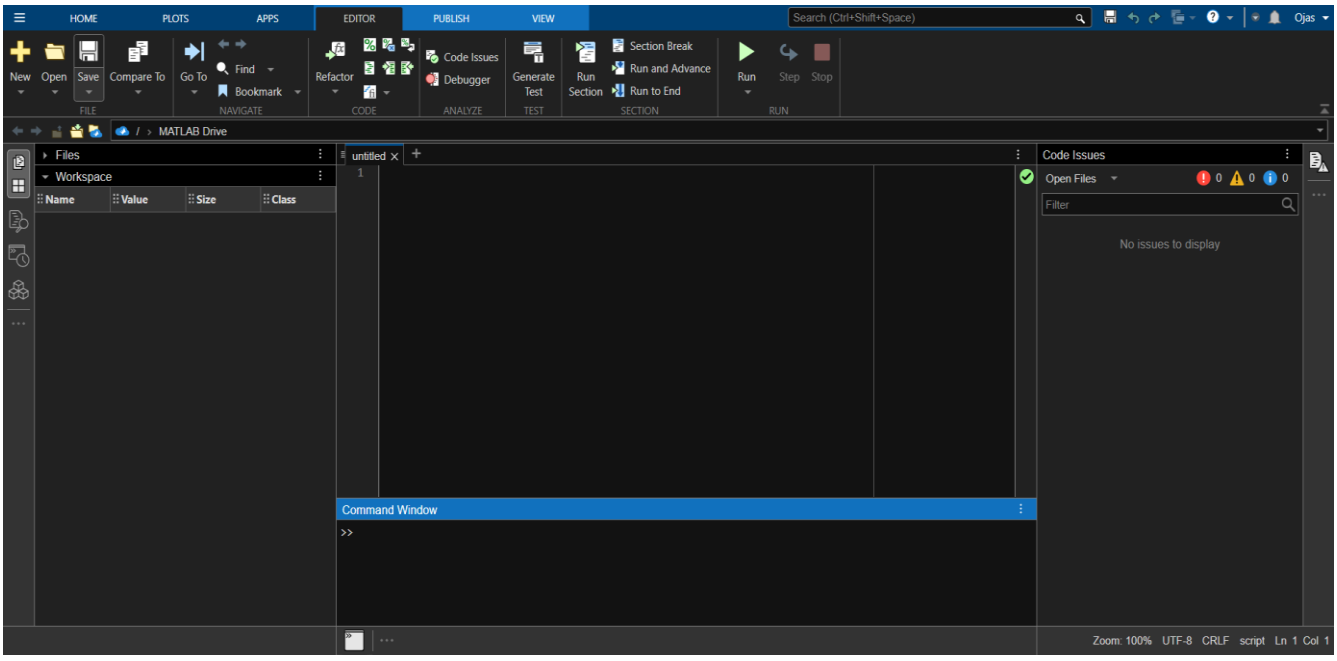
4. Functions

5. Plots

6. Array and Cell Array

7. Inputs and Outputs

I. Introduction to MATLAB



MATLAB stands for Matrix Laboratory. It is a high-performance language that is used for technical computing. It allows matrix manipulations, plotting of functions, implementation of algorithms and creation of user interfaces.

Getting Started with MATLAB

It is both a programming language as well as a programming environment. It allows the computation of statements in the command window itself.

- **Command Window:** In this window one must type and immediately execute the statements, as it requires quick prototyping. These statements cannot be saved. Thus, this is can be used for small, easily executable programs.
- **Editor (Script):** In this window one can execute larger programs with multiple statements, and complex functions These can be saved and are done with the file extension '.m '.
- **Workspace:** In this window the values of the variables that are created in the course of the program (in the editor) are displayed.

II. Working with:

1. Branching statements

In MATLAB, branching statements are used to execute different code based on conditions. They allow for decision-making in the program. The primary branching statements in MATLAB are:

A. if: Executes a block of code if a condition is true.

```
x = 10;
if x > 5
    disp('x is greater than 5');
end
```

```
>> if_example
x is greater than 5
```

B. if-else: Allows you to execute one block of code if a condition is true and another block of code if the condition is false.

```
x = 10;
if x > 0
    disp('Positive number');
else
    disp('Negative or zero');
end
```

```
Command Window
>> ifelse
Positive number
```

C. if-else-if: used to check multiple conditions in sequence and execute different code blocks depending on which condition is true.

```
x = -5;
if x > 0
    disp('Positive number');
elseif x == 0
    disp('Zero');
else
    disp('Negative number');
end
```

```
>> ifelseif
Negative number
```

D. nested-if: statement placed inside another if statement. It allows for more complex decision-making by evaluating multiple conditions in a hierarchical manner.

```
x = 5;
y = 3;
if x > 0
    if y > 0
        disp('Both x and y are positive');
    end
end
```

```
Command Window
>> nestedif
Both x and y are positive
```

E. switch: used to evaluate a variable or expression against multiple possible values. It provides a cleaner alternative to using multiple if-else statements when you need to compare a variable with several different values.

```
choice = 2;
switch choice
    case 1
        disp('You selected option 1');
    case 2
        disp('You selected option 2');
    case 3
        disp('You selected option 3');
    otherwise
        disp('Invalid choice');
end
```

```
Command Window
>> switch_example
You selected option 2
```

2. Loops

In MATLAB, loops are used to repeatedly execute a block of code. The different types of Loops in MATLAB are:

A. For Loop: Used to iterate a specific number of times over a range or array.

```
disp('For Loop Example');
for i = 1:5
    fprintf('Iteration %d\n', i);
end
```

```
Command Window
For Loop Example
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

B. While Loop: Repeats the code block as long as the specified condition is true.

```
disp('While Loop Example');
n = 1;
while n <= 5
    fprintf('Iteration %d\n', n);
    n = n + 1;
end
```

```
Command Window
While Loop Example
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

C. Do-While Loop: Executes the code block at least once and then checks the condition.

```
disp('Do-While Example');
n = 1;
while true
    fprintf('Iteration %d\n', n);
    n = n + 1;
    if n > 5
        break;
    end
end
```

```
Command Window
>> dowhile
Do-While Example
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

D. Nested-For Loop: Loops inside another loop, used for multidimensional operations.

```
disp('Nested For Loop Example');
for i = 1:3
    for j = 1:2
        fprintf('i = %d, j = %d\n', i, j);
    end
end
```

```
Command Window
>> nestedfor
Nested For Loop Example
i = 1, j = 1
i = 1, j = 2
i = 2, j = 1
i = 2, j = 2
i = 3, j = 1
i = 3, j = 2
```

E. Break Statement: Exits the loop immediately, regardless of the loop condition.

```
disp('Break Statement Example');
for i = 1:10
    if i == 5
        disp('Breaking the loop');
        break;
    end
    fprintf('Iteration %d\n', i);
end
```

```
Command Window
>> break_example
Break Statement Example
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Breaking the loop
```

F. Continue Statement: Skips the current iteration and moves to the next one.

```
disp('Continue Statement Example');
for i = 1:5
    if i == 3
        disp('Skipping iteration 3');
        continue;
    end
    fprintf('Iteration %d\n', i);
end
```

```
Command Window
>> continue_example
Continue Statement Example
Iteration 1
Iteration 2
Skipping iteration 3
Iteration 4
Iteration 5
```

3. Datatypes

In MATLAB, data types refer to the kind of data that can be stored and manipulated. Common data types include:

A. Int: refers to integer data types that store whole numbers. These are used when you want to work with numerical values without any decimal points.

```
a = int32(5);
disp(['Variable "a" is of type: ', class(a)])
```

```
Command Window
>> int_datatype
Variable "a" is of type: int32
```

B. String: a sequence of characters used to represent text.

```
b = 'Hello, MATLAB!';
disp(['Variable "b" is of type: ', class(b)])
```

```
>> string_example
Variable "b" is of type: char
```

C. Float: In MATLAB, a float (short for floating-point number) refers to a data type used to represent real numbers (i.e., numbers that may have decimal points).

```
c = 3.14133359;
disp(['Variable "c" is of type: ', class(c)])
```

```
Command Window
>> float_example
Variable "c" is of type: float
```

D. Double: In MATLAB, double is the default data type used to store floating-point numbers. It provides double-precision (64-bit) representation.

```
d = 3.14159;
disp(['Variable "d" is of type: ', class(d)])
```

```
Command Window
>> double_example
Variable "d" is of type: double
```

E. Char: In MATLAB, char (short for character) is a data type used to store text as a character array. Each character in the array is stored as an individual element.

```
e = 'A';
disp(['Variable "e" is of type: ', class(e)])
```

```
Command Window
>> char_example
Variable "e" is of type: char
```

4. Functions

In MATLAB, functions are blocks of code designed to perform specific tasks and can be reused throughout a program. Functions take input arguments, process them, and return output values.

Types of Functions in MATLAB:

A. User-defined function: a custom function created by the user to perform specific tasks that are not built into MATLAB. These functions help in organizing code, avoiding redundancy, and improving program readability.

```
function result = addNumbers(a, b)
    result = a + b;
end

result = addNumbers(5, 3);
disp(['The sum of 5 and 3 is: ', num2str(result)])
```

```
Command Window
>> Addition_Function
The sum of 5 and 3 is: 8
```

B. Condition Function: a user-defined or built-in function that uses conditional statements to make decisions based on certain criteria. It typically evaluates conditions using if, else, and elseif constructs.

```
function result = isPositive(num)
    if num > 0
        result = true;
    else
        result = false;
    end
end

num = -5;
if isPositive(num)
    disp([num2str(num), ' is positive.'])
else
    disp([num2str(num), ' is not positive.'])
end
```

```
Command Window
>> Condition_Function
-5 is not positive.
```

C. Function with Loops: contains repetitive operations using for or while loops. Loops inside functions allow repeated execution of tasks until a condition is met or for a fixed number of iterations.

```
function sumResult = sumNumbers(n)
    sumResult = 0;
    for i = 1:n
        sumResult = sumResult + i;
    end
end

result = sumNumbers(5);
disp(['The sum of numbers from 1 to 5 is: ', num2str(result)])
```

```
Command Window
>> FunctionwithLoops
The sum of numbers from 1 to 5 is: 15
```

D. Recursive Function: function that calls itself in order to solve smaller instances of the same problem.

```
function result = factorialRecursive(n)
    if n == 0 || n == 1
        result = 1;
    else
        result = n * factorialRecursive(n - 1);
    end
end

result = factorialRecursive(5);
disp(['The factorial of 5 is: ', num2str(result)])
```

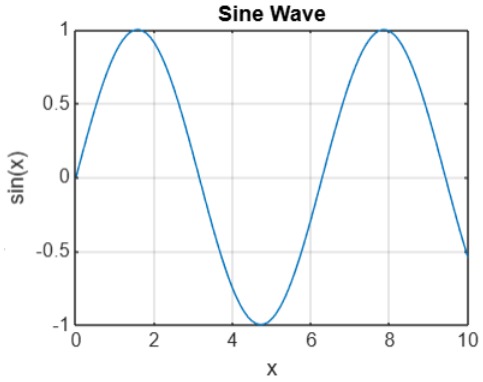
```
Command Window
>> Recursive_Function
The factorial of 5 is: 120
```

5. Plots

In MATLAB, plotting is a fundamental feature used for visualizing data. You can create various types of plots such as line plots, scatter plots, bar plots, and more. Here are some common plotting functions and their usage:

A. Line Plot: A line plot is a type of graph that represents data points as dots connected by straight lines. It is commonly used to show trends or continuous data over a variable.

```
x = 0:0.1:10;
y = sin(x);
plot(x, y);
title('Sine Wave');
xlabel('x');
ylabel('sin(x)');
grid on;
```

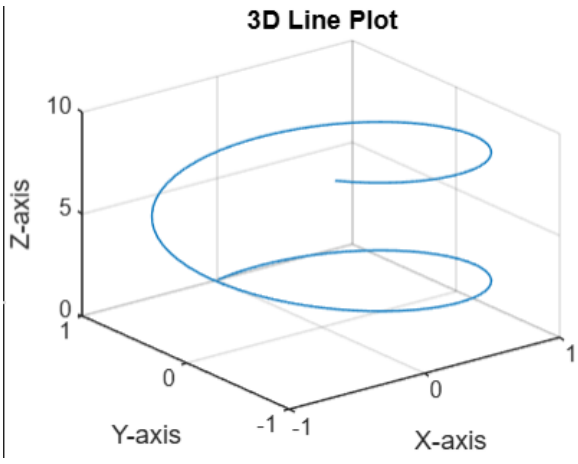


B. Scatter Plot: A scatter plot is a type of graph that displays individual data points on a two-dimensional plane. It is useful to visualize the relationship between two variables.

```
x = rand(1, 100);
y = rand(1, 100);

scatter(x, y);
title('Random Scatter Plot');
xlabel('x');
ylabel('y');
```

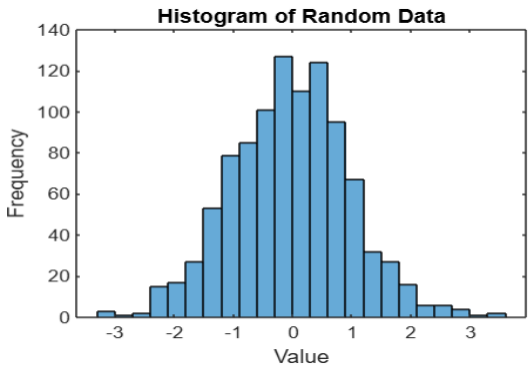
C. 3D Line Plot: A 3D line plot is used to represent data in three dimensions. It plots data points in 3D space and connects them with lines.

```
t = 0:0.1:10;
x = sin(t);
y = cos(t);
z = t;

plot3(x, y, z);
title('3D Line Plot');
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
grid on;
```



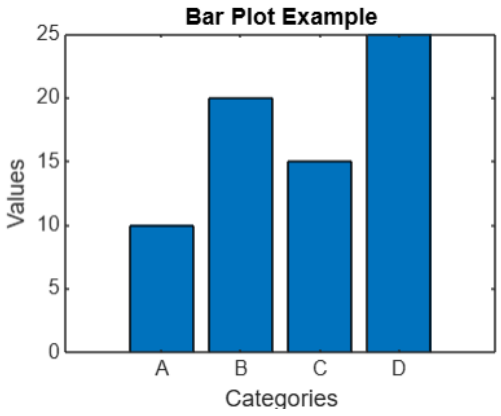
D. Histogram: A histogram is a graphical representation of the distribution of a dataset. It shows the frequency of data points within certain ranges or bins.

```
data = randn(1, 1000);
histogram(data);
title('Histogram of Random Data');
xlabel('Value');
ylabel('Frequency');
```



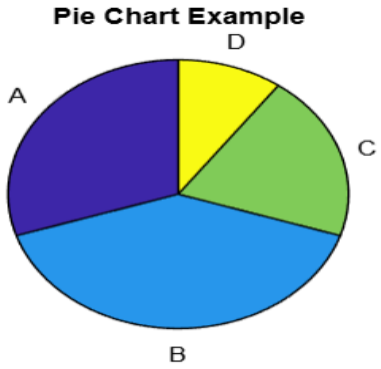
E. Bar Plot: A bar plot represents data with rectangular bars where the length of each bar is proportional to the value of the data. It is commonly used for categorical data.

```
categories = {'A', 'B', 'C', 'D'};
values = [10, 20, 15, 25];

bar(values);
set(gca, 'xticklabel', categories);
title('Bar Plot Example');
xlabel('Categories');
ylabel('Values');
```



F. Pie Chart: A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportions. It is often used to represent parts of a whole.

```
data = [30, 40, 20, 10];
labels = {'A', 'B', 'C', 'D'};

pie(data, labels);
title('Pie Chart Example');
```



6. Array and Cell Array

In MATLAB, arrays and cell arrays are two commonly used data structures, but they are used for different purposes and have different characteristics.

A. Array: An array in MATLAB is a collection of elements of the same data type stored in a single variable. Arrays can be 1D (vectors), 2D (matrices), or even higher-dimensional. Types of Arrays are:

a) Numeric Array: A numeric array in MATLAB is an array that contains numbers, either integers or floating-point values. Numeric arrays are the most commonly used type for storing and manipulating numerical data.

- **1D Numeric Array (Vector):** A one-dimensional array with a sequence of numbers.
- **2D Numeric Array (Matrix):** A two-dimensional array (matrix) of numbers arranged in rows and columns.

b) Logical Array: A logical array contains logical values: true (1) or false (0). Logical arrays are typically used for storing conditions or performing logical operations such as comparisons or filtering.

- **1D Logical Array:** A one-dimensional logical array storing true or false values.

c) Character Array: A character array in MATLAB is used to store characters (individual letters, symbols, or words). It is essentially a sequence of characters and is also used for strings in earlier versions of MATLAB.

- **1D Character Array (String):** A single string represented by an array of characters.

```
numericArray1D = [1, 2, 3, 4, 5];
numericArray2D = [1, 2, 3; 4, 5, 6];

logicalArray = [true, false, true, false];

charArray = 'Hello MATLAB';

disp('Numeric Array (1D):');
disp(numericArray1D);

disp('Numeric Array (2D):');
disp(numericArray2D);

disp('Logical Array (1D):');
disp(logicalArray);

disp('Character Array (1D):');
disp(charArray);
```

```
>> Array
Numeric Array (1D):
     1     2     3     4     5

Numeric Array (2D):
     1     2     3
     4     5     6

Logical Array (1D):
   1   0   1   0

Character Array (1D):
Hello MATLAB
```

B. Cell Array

A cell array is a data structure in MATLAB that allows you to store data of different types and sizes in a single array. Each element in a cell array can be of any data type (numeric, string, array, another cell array, etc.), making it extremely flexible.

- **1D Cell Array:** A one-dimensional cell array where each element can store different data types, such as numbers, strings, or arrays.
- **2D Cell Array:** A two-dimensional cell array, where each cell can contain different types of data.

```
cellArray1D = {1, 'MATLAB', [1, 2, 3], rand(3)};
cellArray2D = {1, 'Hello'; rand(3), [4, 5, 6]};

disp('1D Cell Array:');
disp(cellArray1D);

disp('2D Cell Array:');
disp(cellArray2D);
```

```
>> CellArray
1D Cell Array:
    {[1]}    {'MATLAB'}    {[1 2 3]}    {3x3 double}

2D Cell Array:
    {[       1]}    {'Hello'}
    {3x3 double}    {[4 5 6]}
```

7. Inputs and Outputs

In MATLAB, handling inputs and outputs is an essential part of creating functions and scripts. The concept of inputs and outputs allows functions to accept data and return results.

A. Inputs in MATLAB: Inputs in MATLAB are variables or data passed into functions for processing. These inputs allow a function to operate on different data each time it is called.

B. Outputs in MATLAB: Outputs in MATLAB are the results returned by a function. These outputs are the processed data that is produced from the function's input arguments.

```
function result = squareNumber(x)
    result = x^2;
end

inputNumber = input('Enter a number: ');
outputResult = squareNumber(inputNumber);

disp(['The square of ', num2str(inputNumber), ' is ', num2str(outputResult)]);
```

```
>> Input
Enter a number:
10
The square of 10 is 100
```

```
function [squareVal, cubeVal] = squareAndCube(x)
    squareVal = x^2;
    cubeVal = x^3;
end

inputValue = input('Enter a number: ');
[squareOutput, cubeOutput] = squareAndCube(inputValue);

disp(['The square of ', num2str(inputValue), ' is ', num2str(squareOutput)]);
disp(['The cube of ', num2str(inputValue), ' is ', num2str(cubeOutput)]);
```

```
Enter a number:
22
The square of 22 is 484
The cube of 22 is 10648
```