LAB 2: Introduction to MATLAB (Part II)

1. OBJECTIVE:

Following are the objectives of this lab session:

- To become familiar with basic interface of MATLAB software
- To learn about the built-in functions for arrays
- To understand and apply conditional statements
- To get an understanding of loops and types of loops

2. INTRODUCTION:

MATLAB is a powerful language for technical computing. It can be used for math computation, modeling and simulations, data analysis and processing, visualization and graphics, and algorithm development. MATLAB has also optional toolboxes that are collection of specialized programs designed to solve specific types of problems for example toolboxes for signal processing, symbolic calculations and control systems. Most important key feature of MATLAB is that it has functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, Net and Microsoft Excel.

When you start MATLAB, the desktop appears in its default layout.

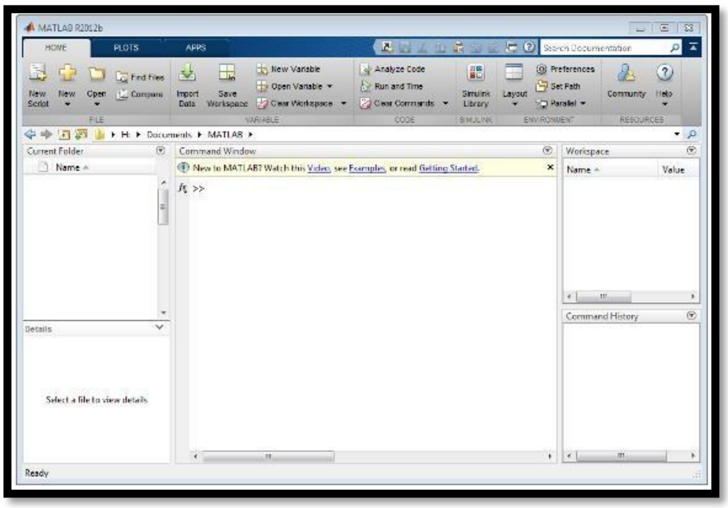

Figure 2: MATLAB Desktop Layout

2.1. Desktop Basics:

The desktop includes these panels:

- **Current Folder:** Access your files.
- **Command Window:** Enter commands at the command line, indicated by the prompt (>>).
- **Workspace:** Explore data that you create or import from files.
- **Command History:** View or rerun commands that you entered at the command line.

3. BASIC SYNTAX:

Some commonly used operators in MATLAB are:

Table 1: Basic Operators in MATLAB

Operators	Purpose
+	Plus; Addition operator
-	Minus: Subtraction Operator
*	Scalar and Matrix Multiplication Operator
^	Scalar and Matrix Exponential Operator
/	Right Division Operator
\	Left Division Operator
:	Colon: Generates Regularly Spaced Elements and Represents an Entire Row or Column
()	Parentheses: Encloses Function Arguments and Array Indices; Overrides Precedence
[ ]	Brackets: Encloses Array Elements
.	Decimal points
…	Ellipsis: line Continuation Operator
,	Comma: Separates Columns
;	Semicolon: Separate Columns
%	Percentage: Designates a comment and specifies formatting
=	Assignment Operator

4. ARRAYS:

An array is a list of numbers arranged in rows and/or columns. A one-dimensional array is a row or a column of numbers and a two-dimensional array has a set of numbers arranged in rows and columns. An array operation is performed element-by-element. In a row vector the elements are entered with a space or a comma between the elements inside the square brackets.

- Classes of Arrays:

  i. Numeric (an ordered collection of numbers)

  ii. Character (array that contains strings)

  iii. Logical (elements of arrays are 'true' & 'false')

  iv. Cell

  v. Structure

A numerical array is an ordered collection of numbers. We use square "[]" brackets to define the variable that contains the collection of data.

- X= [0, 2, 4, 7] or x= [0 2 4 7]
- Y=[0:1:10]

A matrix is a two-dimensional array which has numbers in rows and columns. A matrix is entered row-wise with consecutive elements of a row separated by a space or a comma, and the rows separated by semicolons. The entire matrix is enclosed within square brackets. The elements of the matrix may be real numbers or complex numbers. For example to enter the matrix:

$$Z = [\begin{matrix} 1 & 3 & -4 \\ 0 & -2 & 8 \end{matrix}];$$

5. Matrices Operation in MATLAB

Some more built-in functions for arrays are listed below:

Table 2: Built-in array functions in MATLAB

Function	Description	Example

mean (A)	If A is a vector, returns the mean value of the elements	>> A = [3 7 2 16];
		>> mean(A)
		ans = 7

C = max (A)	If A is a vector, C is the largest element in A. If A is a matrix, C is a row vector containing the largest element of each column of A	>> A = [3 7 2 16 9 5 18 13 0 4];
		>> C = max(A)
		C = 18

[d, n] = max (A)	If A is a vector, d is the largest element in A, n is the position of the element (the first if several have the max value)	>> A = [3 7 2 16 9 5 18 13 0 4];
		>> [d, n] = max (A)
		d = 18 n = 7

C = min (A)	The same as **max (A)** but for the smallest element	>> A = [3 7 2 16 9 5 18 13 0 4];
		>> C = min(A)
		C = 0

[d, n] = min (A)	Same as **[d, n] = max (A)**, but for the smallest element	>> A = [3 7 2 16 9 5 18 13 0 4];
		>> [d, n] = min (A)
		d = 0
		n = 9

sum (A)	If A is a vector, returns the sum of the elements of the vector	>> A = [3 7 2 16];
		>> sum (A)
		ans = 28

sort (A)	If A is a vector, arranges the elements of the vector in ascending order	>> A = [3 7 2 16];
		>> sort (A)
		ans = 2 3 7 16

median (A)	If A is a vector, returns the median value of the elements of the vector	>> A = [3 7 2 16]; >> median(A)

std (A)	If A is a vector, returns the standard deviation of the elements of the vector	>> A = [3 7 2 16];
		>> std (A)
		ans = 6.3770

det (A)	Returns the determinant of a square matrix A	>> A = [3 7; 2 6];
		>> det (A)
		ans = 4

dot (a, b)	Calculates the scalar (dot) product of two vectors a and b. the vector can each be row or column vectors	>> a = [5 6 7];
		>> b = [4 3 2];
		>> dot (a, b)
		ans = 52

cross (a, b)	Calculates the cross product of two vectors a and b (a x b). The two vectors must have 3 elements	>> a = [5 6 7];
		>> b = [4 3 2];
		>> cross (a, b)
		ans = –9 18 –9

inv (A)	Returns the inverse of a square matrix A	>> A = [1 2 3; 4 6 8; –1 2 3]; >> inv (A) ans = –0.5000 0.0000 –0.5000 –5.0000 1.5000 1.0000 3.5000 –1.0000 –0.5000

TASK 1:

Write MATLAB code that will take one array as an input and sort that array in ascending order.

Hint: You will start comparing index 1 with index 2 and index 2 with index 3 and so on until the array is sorted.

```
clc
a=input("Enter an array (Use []) >> ");

for i= 1:length(a)-1
    for j= 1:length(a)-1
        if a(j)>a(j+1)
            temp=a(j);
            a(j)=a(j+1);
            a(j+1)=temp;
        end
    end
end

fprintf("\n\nThe array after applying the sorting algorithm is: \n");
a
```

Output:

```
Enter an array (Use []) >> [9 8 7 6 5 4 3 2 1]

The array after applying the sorting algorithm is:

a =

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

TASK 2:

Write MATLAB code, that will take a 2-D matrix as input and will check if it is sparse or not.

- The sparse matrix is the one having at least 50% entries as 0.

```
clc
a=input("Enter an array (Use []) >> ");
zeroNum=0;
num=0;
for i = 1:length(a)
    if a(i)==0
        zeroNum=zeroNum+1;
    end
    num=num+1;
end

if zeroNum>=round(num/2)
    fprintf('It is a sparse matrix.\n')
else
    fprintf('It is not a sparse matrix.\n')
end
```

Output 1:

```
Enter an array (Use []) >> [1 2 3 4 0 0 0 0]

It is a sparse matrix.

>>
```

Output 2:

```
Enter an array (Use []) >> [1 2 3 4 0 0 0]

It is not a sparse matrix.

>>
```

TASK 3:

Write MATLAB code that will take a positive integer as input and will generate an array at the output with the following sequence: If the number is even, halve it; if the number is odd, multiply it by 3 and add 1. Repeat this sequence until the value is 1.

```
clc
a=input("Enter a positive integer >> ");
b=[];
b(1)=a;
i=2;
while a~=1
    if rem(a,2)
        a=3*a+1;
        b(i)=a;
        i=i+1;
    else
        a=a/2;
        b(i)=a;
        i=i+1;

    end
end

b
```

Output:

```
Enter a positive integer >> 33

b =

  Columns 1 through 15

   33  100  50   25   76   38   19   58   29   88   44   22   11   34   17

  Columns 16 through 27

   52   26   13   40   20   10    5   16    8    4    2    1
```

TASK 4:

An interesting number problem is known as The Necklace Problem. In this problem, we begin with twosingle-digit numbers. The next number in the sequence is generated by adding the previous two numberstogether and saving only the unit digit (rightmost). The process of generating a new number is continueduntil the necklace is closed i.e. the starting two numbers are obtained.

```
clc
a1=input("Enter first positive
integer >> "); a2=input("Enter
second positive integer >> ");
b=[a1 a2];
i=3;
temp=a1; a1=a2;
a2=rem((a2+temp),10); b(i)=a2;
while (a2~=b(2) || a1~=b(1)) i=i+1;
temp=a1; a1=a2;
a2=rem((a2+temp),10); b(i)=a2;
end

fprintf("\n\nThe solution to this necklace problem is:\n") b
```

Output:

```
Enter first positive integer >> 2 Enter second positive integer
>> 4


The solution to this necklace problem is:

b =

Columns 1 through 15

2  4     6     0     6     6     2     8     0     8     8     6     4
   0     4

Columns 16 through 22

4  8     2     0     2     2     4
```