Noakhali Science and Technology University
Department of Information and Communication Engineering

# DIGITAL IMAGE PROCESSING

# LAB MANUAL 1

Introduction to MATLAB

(MATLAB Basics)

Prepared By: Md. Sabbir Ejaz
Lecturer, Dept. of ICE

Noakhali Science and Technology University

Department of Information and Communication Engineering

# DIGITAL IMAGE FUNDAMENTALS

## Lab Objectives:

This objective of this lab is to understand

1. What is MATLAB?
2. All MATLAB windows in detail. [Command window, Graphics window, Editor window, Workspace, Variable panel, Command history, Current directory]
3. Basic MATLAB command with example.
4. Working with variables, vectors, and Matrix
5. Working with library functions.
6. Getting started with MATLAB plotting tools.

## Software:

MATLAB (any version higher than 2014a)

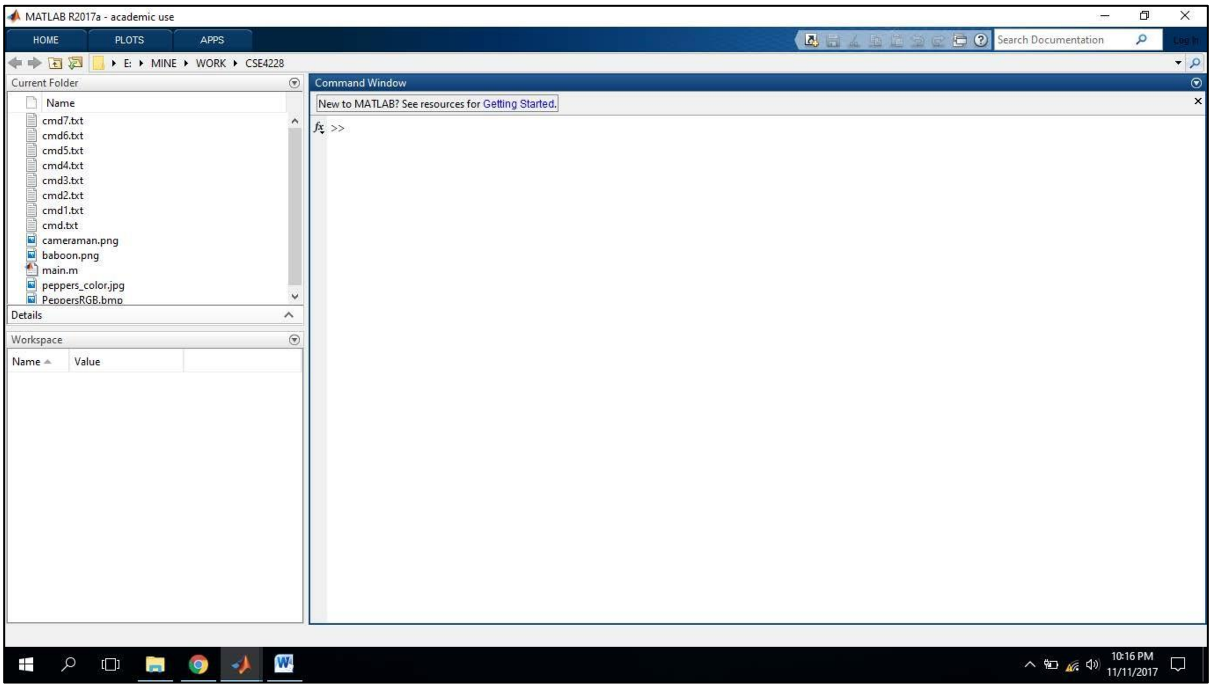



Figure 1

The desktop includes these panels (Figure 1):

- 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.

## Variables and the Workspace:

Let's start with variable.

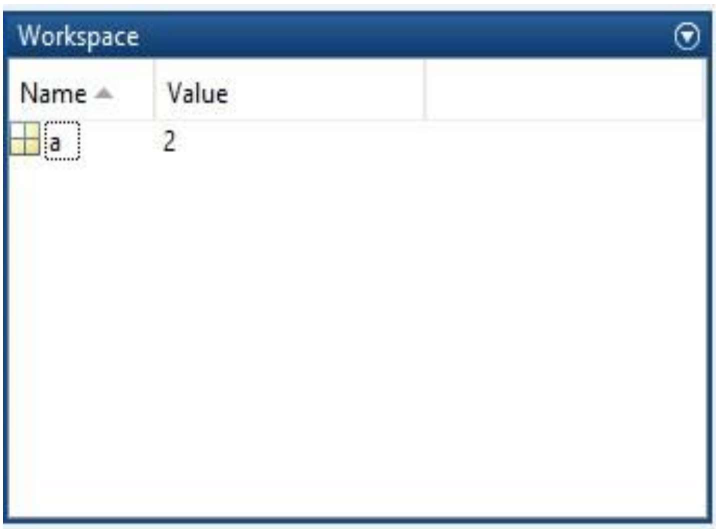
```
a = 2
```

After entering, the immediate next line will display the variable with the assigned value.

```
a=
     2
```

And you can see that, the variable is appeared in the workplace like Figure 2(a). If you double click on that variable icon, a panel named 'Variables' will popup like Figure 2(b).

MATLAB stores everything as matrix. You can note the indication of `1 x 1 double` in the variable panel. It mean `a` is a 1 by 1 matrix (eventually a matrix with only one element), which is `double` type. By default any numerical data is double type. There are other data types as well such as `uint8` (unsigned integer 8 bit), `char` (character), `logical` (boolean).




(a)


(b) Figure 2

Another example, the following is a command.

```
b = 3
```

And the following is the immediate prompt.

```
b=
     3
```

Now, Let’s work more with variables and arithmetic operations. Let’s play with some simple arithmetic operations. Assume that we have already have `a` and `b` in our workspace.

```
a + b

ans=
     5
```

Observe that, `ans` is a default variable that stores the result of `a + b`, and it is stored in workspace. If you simply use `ans` as a command, the value inside `ans` will be prompted.

```
ans

ans=
     5
```

But, we can store the result in a different variable as well.

```
c = a + b
c =
     5

c = b^3 % power of 3
c =
    27
```

There are some special variables.

```
pi
ans =
    3.1416
c = a*pi
c =
    6.2832
```

You can observe that, % is used for commenting in MATLAB.

## Some Library Functions:

Some useful library functions are shown below-

```
c = mod(10,3) % remainder
c =
     1

c = exp(2) % exponential
c =
     7.3891

c = ceil(c) % ceiling (there is floor() and round() as well)
d =
     8

c = log(10) %natural logarithm
c =
     2.3026
```

There is a huge collection of functions dedicated for elementary math. You can find them here [1].

## Using / Not Using Semicolon:

Semicolons (`;`) can be used after each command. If used, the output prompt will be suppressed. For example, if you use the following command without a semicolon at the end, the output will be prompted.

```
d = cos(pi)
d =
    -1
```

But, if you use the following command with a semicolon at the end, the output will not be prompted.

```
d = cos(pi);
```

## Working with Vector and Matrix:

Defining a row vector –

```
v = [1, 2, 3, 4]
v =
     1     2     3     4
```

You can also omit the commas (`,`) in between the elements. This will give you the same vector.

```
v = [1 2 3 4]
v =
    1     2     3     4
```

Now, Defining a column vector –

```
v = [1; 2; 3; 4]
v =
    1
    2
    3
    4
```

Having the idea of row and column vector, now we can easily define a matrix –

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

In MATLAB, the matrix element indices start from the top left corner. As we traverse right, we go through each column. And as we traverse down, we go through each row.

```
             1st column         4th column
                  |                  |
                  v                  v
     1st row -> 1      2      3      4
                5      6      7      8
     3rd row -> 9      1      2      3
```

## Accessing Elements of a Matrix:

Now let's see how to access elements from a matrix. To access we use the following format of the command: `Matrix (which_row, which_column)`. For example, Let's assume that we have matrix M in our workspace.

```
m = M(1,1) % accessing 1st row, 1st column from M
m =
     1

m = M(2,3) % accessing 2nd row, 3rd column from M
m =
     7
```

You can use `end` to access the last element in a row or column.

```
M(1, end) % 1st row, last column
ans =
      4
```

## Assigning Elements of a Matrix:

```
M(2,2) = 99
M =
     1     2     3     4
     5    99     7     8
     9     1     2     3
```

## Some Library Functions to Generate Matrix:

`eye` (`m`) : Generating an identity matrix, For example, for m = 5

```
I = eye(5) % identity matrix of 5 x 5
I =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

`ones` (`m`) : Generating a matrix that contains only 1 (one). For example, m = 5

```
I = ones(5)
I =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

You can also generate matrix rather than square shape. You can use `ones (m,n)` to define rows and columns. For example,

```
I = ones(3,5) % 3 rows and 4 columns
I =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Similarly,

```
I = zeros(3,5)
I =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Let's apply matrix operations of two matrices `A` and `B`.

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

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

C = A + B
C =
    10    10    10
    10    10    10
    10    10    10

C = A * B
C =
    30    24    18
    84    69    54
   138   114    90

C = A^2
C =

    30    36    42
    66    81    96
   102   126   150
```

We can apply element wise operations. To perform that, we need to put a dot(.) in front of the operator. For example, the following command will do element wise multiplication, that means 1st element of `A` will be multiplied with the 1st element of `B`, 2nd of A will be multiplied with 2nd element of B, and so on.

```
C = A .* B
C =
      9    16    21
     24    25    24
     21    16     9
```

## Some Other Library Functions for Matrix/ Vector:

- Transpose of vector & matrix: `transpose(data) or data'`
- Sorting a vector & matrix: `sort(data)`
- Sum of all the elements of a vector & matrix : `sum(data)`
- Mean, median: `mean(data), median(data)`

## Using Colon Operator:

Let's take the matrix A in previous examples.

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

Say, we want to access all the columns at 1st row, that is `1, 2` and `3`

```
c1 = A(1,:)
c1 =
     1     2     3
```

Moreover,

```
c1 = A(2,:) % 2nd row, all columns
c1 =
     4     5     6
```

Similarly, to get all rows at 3rd columns -

```
r1 = A(:,3)
r1 =
     3
     6
     9
```

The colon operator is useful to generate a range of values. Say, we want to create a vector named `data` with values from 1 to 10.

```
data = 1:10
data =

     1     2     3     4     5     6     7     8     9    10

data = 1:2:10 % increment by 2 steps
data =

     1     3     5     7     9
```

## Control & Loop Statements:

***if*** expression evaluates an expression, and executes a group of statements when the expression is true.

```
if expression
    statements
else
    statements
end
```

```
A=5;
if A > 0
    disp('Positive')
else
    disp('Negative')
end
```

With loop control statements, you can repeatedly execute a block of code. ***for*** statements loop a specific number of times, and keep track of each iteration with an incrementing index variable. For example, display five values:

```
for index = values
   statements
end
```

```
for n = 1:1:5
    disp(n)
end
```

## Plotting:

Assume that, we want to plot five 2-D points. The `(x,y)` coordinates are `(-5,-2)`, `(6,4)`, `(8,-3)`, `(9,5)` and `(-1,1)`. Now, let's store this coordinate data in two vectors X and Y, where X contains all x-coordinates and Y contains all the y-coordinates.

```
X = [-5 6 8 9 -1];

Y = [-2 4 -3 5 1];
```

To plot them, we can simply call the `plot()` function.

```
plot(X,Y,'*')
```

A window will pop up (Figure 3) showing the axis and the plotted points. Points will be plotted with `(*)` as we pass this as the third parameter. These are called markers. You can use `'.'`, `'O'`, `'o'`, `'X'`, `'x'` etc. as markers.

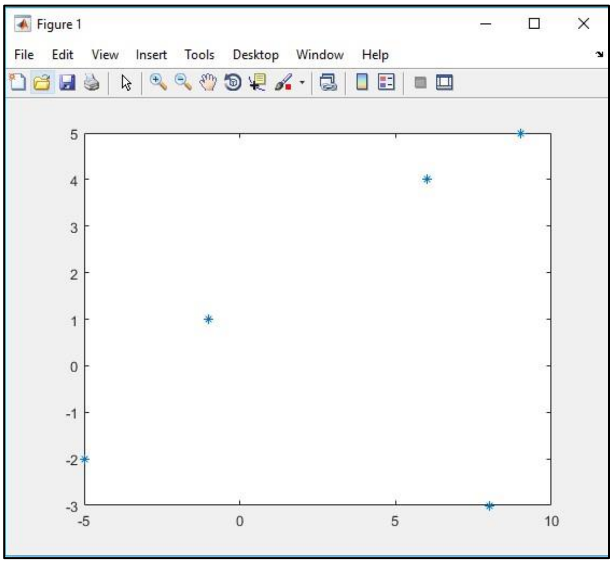



Figure 3

Let's plot `y = x`$^3$ equation. We define 50 points for x. Then obtain the y from `y = x`$^3$ equation (Figure 4).

```
x = 1:50; y = x.^3;
plot(x,y,'.')
```

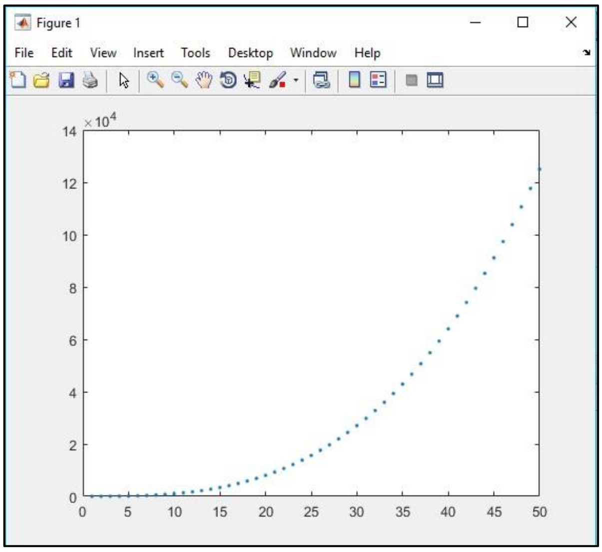



Figure 4

We can connect the points by passing different parameter.

```
plot(x,y,'.-')
```

A hyphen after the dot means to connect the dots with a continuous line. You can also change the colors of the markers (Figure 5).

```
plot(x,y,'.-r') % r stands for red.
```

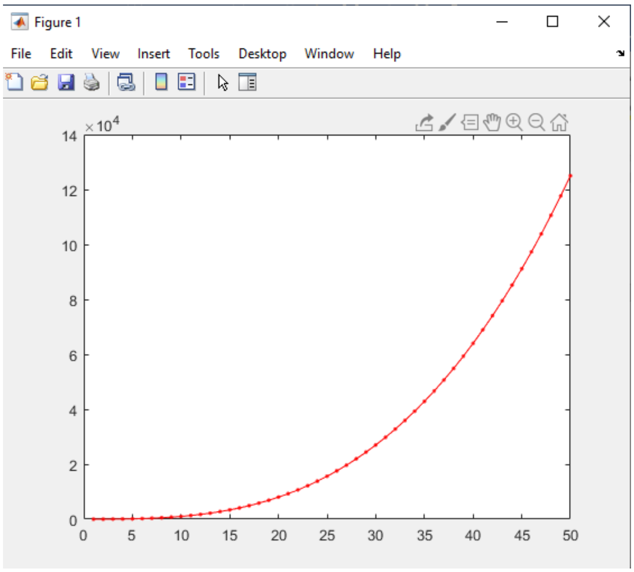



Figure 5

Also, you can have multiple figures for multiple plotting. Say, we want the previous two equations to be plotted in two different figure windows. In that case, enter `figure` command to open a new figure (Figure 6).

```
x = 1:50; y = x.^3;
% new window
figure; plot(x,y,'.-b'); y = (x+10).^3;
% another new window
figure; plot(x,y,'.-r');
```

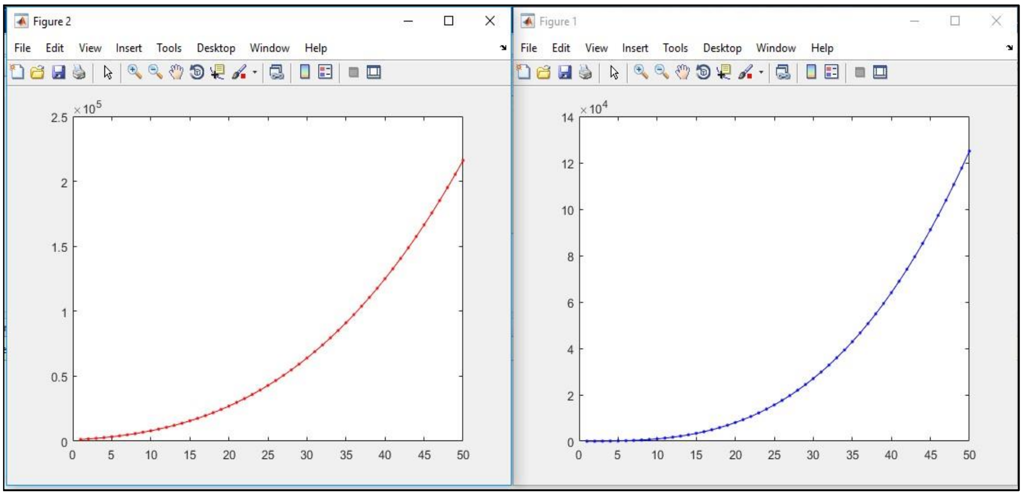



Figure 6

More about MATLAB plotting is available here [5].

## TASK 4

Q1. Square every element of the matrix A using one single line of command.

Q2. Square each elements of both A and B. Add the squared elements of A with the squared elements B and store them in the matrix C.

Q3. Multiply Two Matrices Using Multi-dimensional Arrays.

## References:


[1] https://www.mathworks.com/help/matlab/learn_matlab/desktop.html

[2] https://www.mathworks.com/help/matlab/elementary-math.html

[3] https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html

[4] https://www.mathworks.com/help/matlab/functionlist.html?s_cid=doc_ftr

[5] https://www.mathworks.com/help/matlab/2-and-3d-plots.html

[6] https://aust.edu/lab_manuals/CSE/CSE4228-Lab%20Manual.pdf