American International University- Bangladesh

Department of Computer Engineering

COE 3201: Data Communication Laboratory

Title: **Introduction to MATLAB**

Abstract:

This experiment is designed to-

1.To understand the use of MATLAB for solving communication engineering problems.
2.To develop understanding of MATLAB environment, commands and syntax.

Introduction:

Matlab is a high-performance language for technical computing. It integrates computation, programming and visualization in a user-friendly environment where problems and solutions are expressed in an easy-to-understand mathematical notation.

Matlab is an interactive system whose basic data element is an array that does not require dimensioning. This allows the user to solve many technical computing problems, especially those with matrix and vector operations, in less time than it would take to write a program in a scalar noninteractive language such as C or Fortran.

Matlab features a family of application-specific solutions which are called toolboxes. It is very important to most users of Matlab, that toolboxes allow to learn and apply specialized technology. These toolboxes are comprehensive collections of Matlab functions, so-called M files, that extend the Matlab environment to solve particular classes of problems.

Matlab is a matrix-based programming tool. Although matrices often need not to be dimensioned explicitly, the user has always to look carefully for matrix dimensions. If it is not defined otherwise, the standard matrix exhibits two dimensions' n × m. Column vectors and row vectors are represented consistently by n × 1 and 1 × n matrices, respectively.

Matlab operations can be classified into the following types of operations:

- arithmetic and logical operations,
- mathematical functions,
- graphical functions, and
- input/output operations.

Expressions

Like most other programming languages, Matlab provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are

- Variables
- Numbers
- Operators
- Functions

Variables

Matlab does not require any type declarations or dimension statements. When a new variable name is introduced, it automatically creates the variable and allocates the appropriate amount of memory. If the variable already exists, Matlab changes its contents and, if necessary, allocates new storage. For example

```
>> books = 10
```

creates a 1-by-1 matrix named books and stores the value 10 in its single element. In the expression above, >> constitutes the Matlab prompt, where the commands can be entered. Variable names consist of a string, which start with a letter, followed by any number of letters, digits, or underscores. Matlab is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

Numbers

Matlab uses the conventional decimal notation. A decimal point and a leading plus or minus sign is optional. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are:

```
7       -55     0.0041      9.657838    6.10220e-10   7.03352e21    2i      -2.71828j
        2e3i    2.5+1.7j.
```

Operators

Expressions use familiar arithmetic operators and precedence rules. Some examples are:

+ Addition
- Subtraction
* Multiplication
/ Division
' Complex conjugate transpose
( ) Brackets to specify the evaluation order.

Functions

Matlab provides a large number of standard elementary mathematical functions, including sin, sqrt, exp and abs. Taking the square root or logarithm of a negative number does not lead to an error; the appropriate complex result is produced automatically. Matlab also provides a lot of advanced mathematical functions, including Bessel and Gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

```
>> help elfun
```

Some of the functions, like sqrt and sin are built-in. They are a fixed part of the Matlab core so they are very efficient. The drawback is that the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in so called M-files. You can see the code and even modify it if you want.

Getting Started:

a) Go to the start button, then programs, MATLAB and then start MATLAB. It is preferred that you have MATLAB2016a. You can then start MATLAB by double clicking on its icon on Desktop, if there is any.

b) The Prompt:

>>

The operator shows above is the prompt in MATLAB. MATLAB is interactive language like C, Java etc. We can write the commands over here.

c) In MATLAB we can see our previous commands and instructions by pressing the up key. Press the key once to see the previous entry, twice to see the entry before that and so on. We can also edit the text by using forward and back-word keys.

Entering Matrices and Addressing the Elements

The elements of a matrix must be entered one-by-one in a list where the elements of a row be separated with commas or blank spaces and the rows are divided by semicolons.

The whole list must be surrounded with square brackets, e.g.

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

After pressing "**Enter**" Matlab displays the numbers entered in the command line

A = 1 2 3
8 6 4
3 6 9

Addressing an element of a matrix is also very easy. The n-th element of the m-th column in matrix A from above is A(n,m). So typing

```
>> A(1,3) + A(2,1) + A(3,2)
```

will compute the answer
ans = 17

The k-th to l-th elements of the m-th to n-th columns can be addressed by A(k:l,m:n), e.g.

```
>> A(2:3,1:2)
```

ans = 8 6
3 6

Further examples:

```
>> A(1,1:2)
```

addresses the first two elements of the first row.

ans = 1 2

```
>> A(2,:)
```

addresses all elements of the second column.

ans = 8
6
4

Generating Matrices

There are different ways to generate matrices. Assigning elements explicitly was presented in the paragraph above. To create a row vector with 101 equidistant values starting at 0 and ending by π, this method would be very tedious. So two other possibilities are shown below:

```
>> x = linspace(0,pi,101)
```

Or

```
>> x = (0:0.01:1)*pi
```

In the first case, the Matlab function linspace is used to create x. The function's arguments are described by:

linspace(first value, last value, number of values) with the default number of values = 100.

In the second case, the colon notation (0:0.01:1) creates an array that starts at 0, increments by 0.01 and ends at 1. Afterwards each element in this array is multiplied by π to create the desired values in x. Both of these array creation forms are common in Matlab. While the colon notation form allows to specify the increment between data elements directly, but not the number of data elements, the Matlab function linspace allows to specify the number of data elements directly, but not the increment value between these data elements. The colon notation is very often used in Matlab, therefore a closer look should be taken on it. (first value:increment:last value) creates an array starting at first value, ending at last value with an increment which can be negative as well, e.g.

```
>> v = (10:-2:0)
```

v = 10 8 6 4 2 0

If the increment is 1, then its usage is optional:

```
>> w = (5:10)
```

w = 5 6 7 8 9 10

Matlab also provides four functions that generate basic matrices: zeros, ones, rand and randn. Some examples:

```
>> B = zeros(3,4)
```

B = 0 0 0 0
0 0 0 0
0 0 0 0

```
>> C = ones(2,5)*6
```

C = 6 6 6 6 6
6 6 6 6 6

```
>> D = rand(1,5)
```

generates uniformly distributed random elements

D = 0.5028 0.7095 0.4289 0.3046 0.1897

```
>> E = randn(3,3)
```

generates normally -also called Gaussian- distributed random elements

E = -0.4326 0.2877 1.1892
-1.6656 1.1465 -0.0376
0.1253 1.1909 0.3273

Deleting rows and columns

To delete rows or columns of a matrix, just use a pair of square brackets, e.g.

```
>> A(2,:) = [ ]
```

deletes the second row of A.

A = 1 2 3
3 6 9

It is not possible to delete a single element of a matrix, because afterwards it would not still be a matrix.(Exception: vectors, since here deleting an element is the same as deleting a row/column.)

Array Orientation

The orientation of an array can be changed with the Matlab transpose operator':

```
>> a = 0:3
```

a = 0 1 2 3

```
>> b = a'
```

b = 0
1
2
3

Scalar-Array Mathematics

Addition, subtraction, multiplication and division by a scalar apply the operation to all elements of the array:

```
>> c = [1 2 3 4;5 6 7 8;9 10 11 12]
```

```
c = 1  2   3   4
    5  6   7   8
    9 10 11 12
```

```
>> 2*c-1
```

multiplies each element in c by two and subtracts one from each element of the result.

```
ans = 1   3  5   7
      9 11 13 15
     17 19 21 23
```

Array-Array Mathematics

When two arrays have the same dimensions, which means that they have the same number of rows and columns, addition, subtraction, multiplication and division apply on an element-by-element basis in Matlab.

```
>> d = [1 2 3; 4 5 6]
```

```
d = 1 2 3
    4 5 6
```

```
>> e = [2 2 2; 3 3 3]
```

```
e = 2 2 2
    3 3 3
```

```
>> f = d+e
```

adds d to e on an element-by-element basis

```
f = 3 4 5
    7 8 9
```

```
>> g = 2*d-e
```

multiplies d by two and subtracts e from the result

```
g = 0 2 4
    5 7 9
```

Element-by-element multiplication and division work similarly, but the notation is slightly different:

```
>> h = d.*e
```

h = 2 4 6
12 15 18

The element-by-element multiplication uses the dot multiplication symbol .*, the elementby element array division uses either ./ or .\

```
>> d./e
```

ans = 0.500 1.000 1.500
1.333 1.666 2.000

```
>> e.\d
```

ans = 0.500 1.000 1.500
1.333 1.666 2.000

In both cases, the elements of the array in front of the slash is divided by the elements of the array behind the slash. To compute a matrix multiplication only the asterisk * must be used, e.g.

```
>> C = A * B
```

Therefore, the number of columns of A must equal the number of rows of B.

```
>> A = [1 2 3; 4 5 6]
```

A = 1 2 3
4 5 6

```
>> B = [1 2; 3 4; 5 6]
```

B = 1 2
3 4
5 6

```
>> C = A * B
```

C = 22 28
49 64

Creating a Plot

The plot function has different forms, depending on the input arguments. If y is a vector, plot(y) produces a piecewise linear graph of the elements of y versus the index of the elements of y. If two vectors are specified as arguments, plot(x,y) produces a graph of y versus x. For example to plot the value of the sine function from zero to 2π, use

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
```

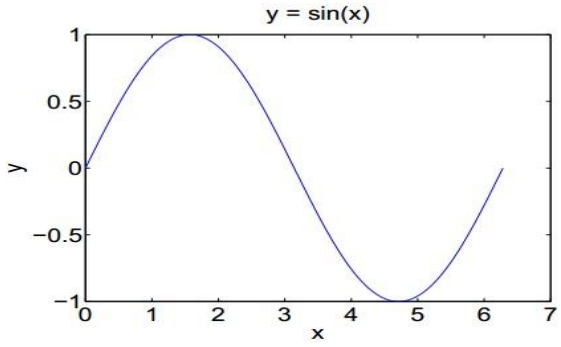

Figure 1.1: Sine plot

The xlabel, ylabel and zlabel functions are useful to add x-, y- and z-axis labels. The function is only necessary for three-dimensional plots. The title function adds a title to a graph at the top of the figure and the text function inserts a text in a figure. The following commands create the final appearance of figure 1.1 .

```
>> xlabel('x');
>> ylabel('y');
>> title('y = sin(x)')
```

Multiple x-y pairs create multiple graphs with a single call to plot. Matlab automatically cycles through a predefined (but user settable) list of colors to distinguish between different graphs. For example, these statements plot three related functions of x1, each curve in a separate distinguishing color:

```
>> x1 = 0:pi/100:2*pi;
>> y1 = sin(x1);
>> y2 = sin(x1 - 0.25);
>> y3 = sin(x2 - 0.5);
>> plot(x1,y1,x1,y2,x1,y3)
```

The number of points of the individual graphs may be even different. It is possible to specify the color, the line style and the markers, such as plus signs or circles, with:
plot(x,y,'color style marker')

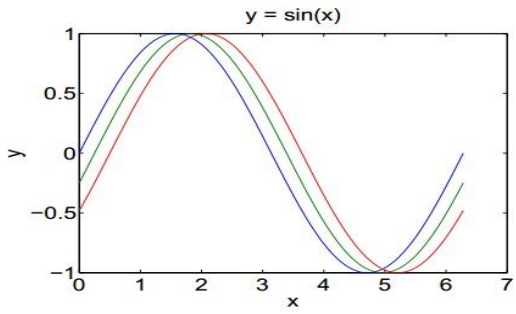

Figure 1.2: Multiple graphs with a single call to plot

A color style marker is a 1-, 2-, or 3-character string. It may consist of a color type, a line style type, and a marker type:

Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w' and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.

Line style strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dotted and 'none' for no line.

The most common marker types include '+', 'o', '*' and 'x'.

For example, the statement plot(x1,y1,'b:*') plots a blue dotted line and places asterisk sign markers at each data point. If only a marker type is specified but not a line style, Matlab draws only the marker.

The plot function automatically opens a figure window to plot the graphic. If there is already an existing figure window, this windows will be used for the new plot. The command figure can be used to keep an existing figure window and open a new one, which will be used for the next plot. To make an existing window the current window, type figure(n) where n is the number in the title bar of the window to be selected. The next graphic will be plotted in this selected window.

To add further plots to an existing graph, the hold command is useful. The hold on command keeps the content of the figure and plots can be added. So the above example could be done with three single plot commands and the hold on command. hold off ends the hold on status of a figure window. hold can be used to toggle between on and off.

Controlling Axes

Usually, Matlab finds the maxima and minima of the data to be plotted by itself and uses them to create an appropriate plot box and axes labeling. The axis function overwrites this default by setting custom axis limits,

```
>> axis([xmin xmax ymin ymax]).
```

The following example illustrates the use of the functions presented above.

```
>> t = -pi:pi/100:pi;
>> s = cos(t);
>> plot(t,s)
>> axis([-pi pi -1 1])
>> xlabel('-\pi \leq t \leq \pi')
>> ylabel('cos(t)')
>> title('The cosine function')
>> text(-2, -0.5,'This is a note at position (-2, -0.5)')
\leq is used to generate the less-equal sign.
```

To take a closer look at an interesting part of a plot, the zoom command can be used. Afterwards it is possible to zoom by marking this part with the mouse. The grid command is used to turn a grid on and off.

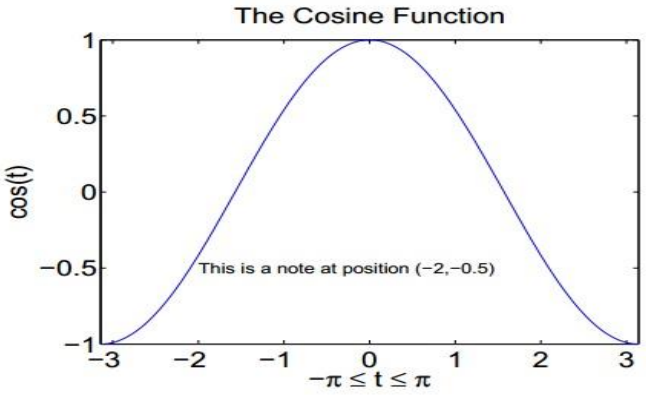

Figure 1.3: Example for controlling the axes

Pre-Lab task:

1) Install MATLAB2016a Software in your personal Laptop/Desktop.
2) Go through the User Guide of MATLAB Software.

Software:

MATLAB2016a

Performance Task for Lab Report: (your ID = AB-CDEFG-H)

**Generate two CDEF hertz sinusoids with different amplitudes and phases.
$x_1(t) = A_1 \cos(2\pi(\text{CDEF})t + j_1)$ $x_{2(}t) = A_2 \cos(2\pi(\text{CDEF})t + j_2)$

(a) Select the value of the amplitudes as follows: let A_1 = AB and A_2 = GH. For the phases, use j_1 = DG (in degrees), and take j_2 = GE (in degrees). *When doing computations in* Matlab, *make sure to convert degrees to radians.*

(b) Make a plot of both signals over a range of *t* that will exhibit approximately 3 cycles. Make sure the plot starts at a negative time so that it will include *t* = 0, *and make sure that you have at least 20 samples per period of the wave.*

(c) Verify that the phase of the two signals $x_1(t)$ and $x_2(t)$ is correct at *t* = 0, and also verify that each one has the correct maximum amplitude.

(d) Use subplot(3,1,1) and subplot(3,1,2) to make a three-panel subplot that puts both of these plots on the same window. See help subplot.

(e) Create a third sinusoid as the sum: $x_3(t) = x_1(t) + x_2(t)$. In Matlab this amounts to summing the vectors that hold the samples of each sinusoid. Make a plot of x_3 (*t*) over the same range of time as used in the previous two plots. Include this as the third panel in the window by using subplot (3,1,3).

(f) Measure the magnitude and phase of x_3 (*t*) directly from the plot. In your lab report, explain how the magnitude and phase were measured by making annotations on each of the plots.

References:

1. MATLAB user guide.
2. Prof. Dr.-Ing. Andreas Czylwik, "MATLAB for Communications"