CENG331 Data Communications
Fall 2013 – Laboratory Manual I

MATLAB (**mat**rix **lab**oratory) is a numerical computing environment. During this semester, we will use MATLAB for laboratory exercises and assignments.

MATLAB uses an interpreted language; which does not get compiled to create a binary executable but rather gets executed line by line. For example, if we type the following expression into MATLAB command window:

```
> 5 + 3
```

MATLAB will produce the following result as we hit Enter key:

```
ans = 8
```

Here, we asked MATLAB to calculate the result of the given mathematical operation but did not specify a variable name for storing the result. MATLAB calculated this expression and since no variable name was given, it automatically created a new variable named **ans** and assigned the resulting value.

If we had wanted to store the results in variables, we could do the following:

```
> result = 5 + 3
result = 8
```

Try the following and write down the results you see on screen:

```
> 5 * 30
```

```
> var = 5 * 5
```

```
> result = ans / var
```

```
> result / 0
```

In addition to basic arithmetic operations, we can also use MATLAB's built-in functions. For example;

```
> power(5,3)
ans = 125
> sqrt(81)
ans = 9
> log(2)
ans = 0.69315
```

Arrays and Matrices

Creating arrays on MATLAB is easy:

```
> arr = [1 2 3 4 5]
arr =
   1 2 3 4 5

> arr = [1, 2, 3, 4, 5]
arr =
   1 2 3 4 5
```

As you have seen, adding commas between array elements has no effect and both statements has created the same result. Here we have created an array of size 1x5 (1 row, 5 columns). We can also call this array a "row vector". To create an array with more than 1 rows (or a "column vector"), we can use semicolons:

```
> arr2 = [5; 10; 15; 20]
arr2 =

    5
   10
   15
   20
```

Or, we can ask MATLAB to calculate the transpose of a row vector by using an apostrophe:

```
> arr3 = [5 10 15 20]'
arr3 =

    5
   10
   15
   20
```

One feature of MATLAB is using colon notation to generate an array of sequential elements. Examine the following examples:

```
> 1:5
ans =

   1   2   3   4   5

> -2:1:4
ans =

   -2.1000   -1.1000   -0.1000    0.9000    1.9000    2.9000    3.9000
```

As you see, colon notation requires two values: beginning and ending points. As a result, we obtain an array of values that increment by 1. If required, we can specify a third value for increment amount:

```
> 1:0.5:3
ans =

    1.0000    1.5000    2.0000    2.5000    3.0000
```

Creating a matrix is also an easy job with MATLAB:

```
> m1 = [1 2 3 4 5; 2 4 6 8 10; 1 5 25 125 625; 3 1 4 1 5; 2 3 5 8 13]
m1 =

     1     2     3     4     5
     2     4     6     8    10
     1     5    25   125   625
     3     1     4     1     5
     2     3     5     8    13
```

To access an element of this given matrix, we should specify its coordinates.

```
> m1(2,5)
ans =  10
```

Here, we have obtained the element at the 2nd row and 5th column of the matrix m1. Do note that array indices start from 1 in MATLAB, not 0.

We can modify the value of a specific matrix element by assigning a new value to it, such as:

```
> m1(2,5) = 15
m1 =

     1     2     3     4     5
     2     4     6     8    15
     1     5    25   125   625
     3     1     4     1     5
     2     3     5     8    13
```

Now think that you have a 2-dimensional array in your C / C ++ / Java program. To get a sub-matrix, you would have to write two for loops; one for traversing y axis and one for traversing x axis. Of course, such an approach is also possible with MATLAB but we can simply prefer using colon notation:

```
m1(2:3,3:5)
ans =

     6     8    15
    25   125   625
```

Here, we have asked MATLAB to fetch the rows from 2 to 3, and columns from 3 to 5. The intersection of these boundaries are returned as the answer.

Plotting Graphs

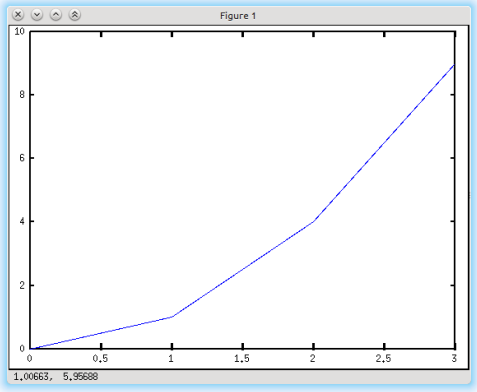
A graph is a 2-dimensional representation of two related datasets (x and y axes). For example lets create the following variables:

```
> x = 0:3;
> y = power(x,2);
```

To display the graph of these data, **plot** function should be used:

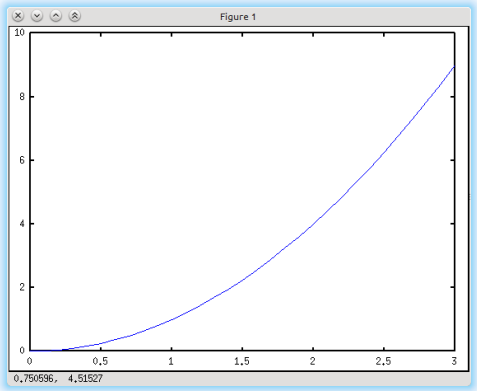
```
> plot(x,y)
```

As you hit Enter key, a new window will open which displays the graph:



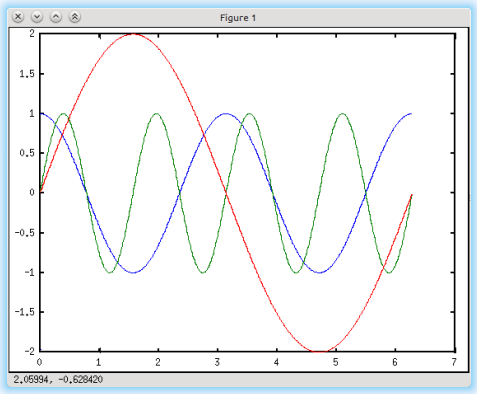
Do note that; the more data points you use, the more precise results you will get. Here we have 4 points (0, 1, 2 and 3), thus we observe a somewhat broken line. Lets try the same with more points:

```
> x = 0:0.1:3;
> y = power(x,2);
> plot(x,y)
```



To display more than one plot in one graph, you should keep listing the variables as plot parameters:

```
x = 0:0.01:2*pi;
a = cos(2*x);
b = sin(4*x);
c = 2*sin(x);
plot(x,a,x,b,x,c)
```



You can alter how each plot is represented (use “help plot”):

```
plot(x,a,'LineWidth',3,x,b,x,c,'r*')
```