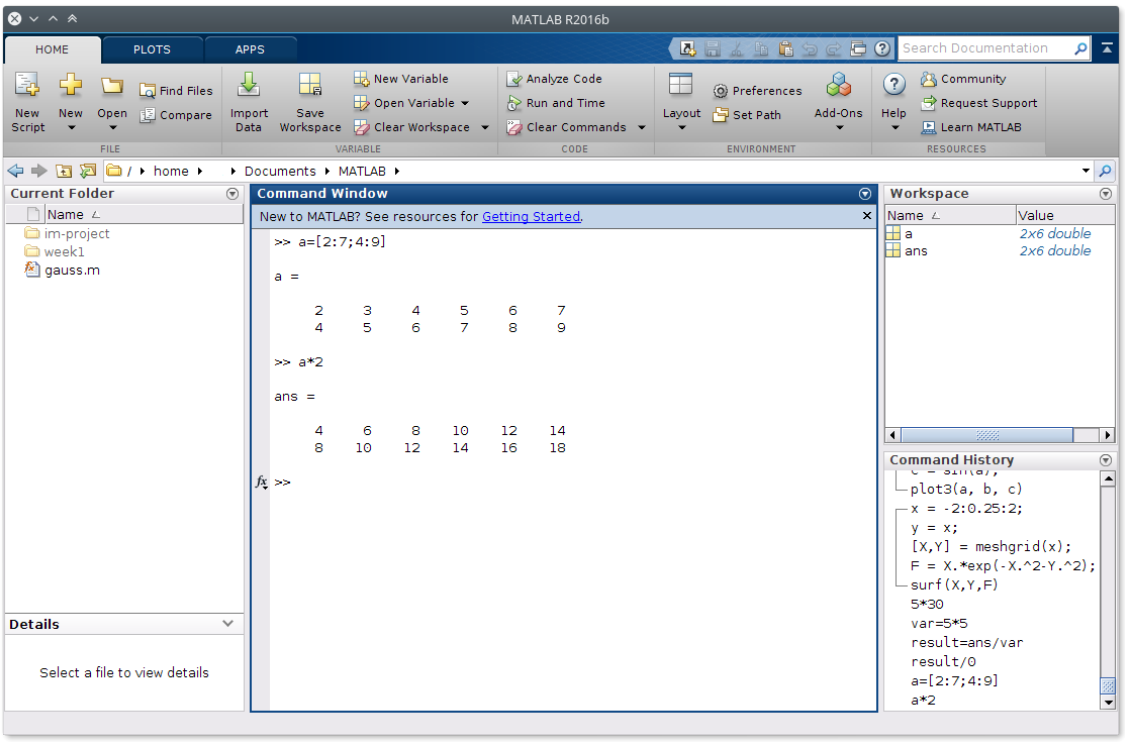
CENG290 Data Communications
Labwork 1

MATLAB (**mat**rix **lab**oratory) is a numerical computing environment.



1 Prelimineries

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 to this new variable. If we had wanted to store the result in a variable, we could do the following:

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

In MATLAB, variable types such as int, char are not necessary to define while declaring variables. MATLAB by default creates all variables with double precision. Also, lines in MATLAB do not have to end with semicolon character. When a line with no terminating semicolon is executed, the calculated output is printed on command window. If a line is terminated with semicolon, its output is suppressed.

1.1 Exercise

Try the following commands one by one on your own now and inspect the output as you execute each command:

```
>> 5 * 30
>> var = 5 * 5;
>> result = ans / var
>> result / 0
```

2 Built-in Functions

MATLAB provides a wide range of built-in functions which greatly helps writing MATLAB programs easily. Some of them are,

```
>> power(5, 3)        >> sqrt(81)        >> log(2)        >> abs(-5)
ans = 125             ans = 9            ans = 0.6931     ans = 5
```

3 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 see, 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 to identify rows:

```
>> 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.

Or, we can ask MATLAB to return the complete second row without defining beginning and ending indices of columns. This is useful if you don't know the exact size of your matrix.

```
>> m1(2,:)
ans = 2     4     6     8    10
```

3.1 Exercise

Given the 5x5 matrix above, write a command which returns the following submatrix:

```
1   4   1
3   5   8
```

4 Matrix Operations

Two matrices can be concatenated via:

```
>> a = [1 2; 5 6]
a =
     1     2
     5     6

>> b = [3 4; 7 8]
b =
     3     4
     7     8

>> c = [a b]
c =
     1     2     3     4
     5     6     7     8
```

With MATLAB, adding and subtracting elements of two different matrices can be achieved via:

```
>> a = [2 6; 4 8];
>> b = [3 1; 7 5];
>> a + b
ans =
     5     7
    11    13

>> a - b
ans =
    -1     5
    -3     3
```

Multiplication, however, takes two different forms: matrix multiplication as a whole, or element-wise multiplication. The first one can be achieved via:

```
>> a * b
ans =
    48    32
    68    44
```

By which formula did ans variable get calculated?

The other multiplication, element-wise, is achieved via:

```
>> a .* b
ans =
     6     6
    28    40
```

As you can see, `*` and `.*` are two different multiplication operators in MATLAB.

4.1 Generating Arrays and Matrices from Functions

- **zeros(M, N):** MxN matrix of zeros.

  ```
  >> x = zeros(1, 5)
  x = 0     0     0     0     0
  ```

- **ones(M, N):** MxN matrix of ones.

  ```
  >> x = ones(1, 3)
  x = 1     1     1
  ```

- **rand(M, N):** MxN matrix of random doubles on (0, 1).

  ```
  >> x = rand(1, 4)
  x = 0.5469    0.9575    0.9649    0.1576
  ```

5 Plotting Graphs

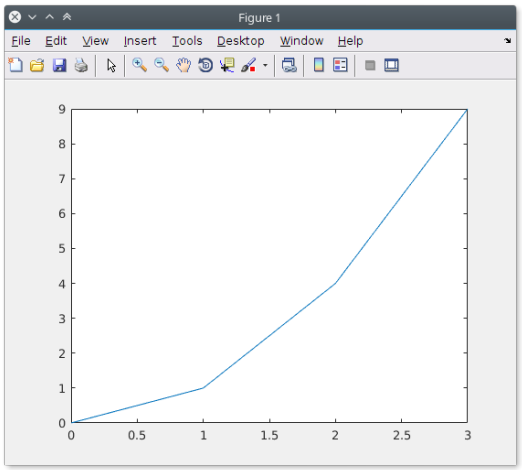
A graph is a 2-dimensional representation of two related datasets (x and y axes). For example lets create the following variables:

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

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

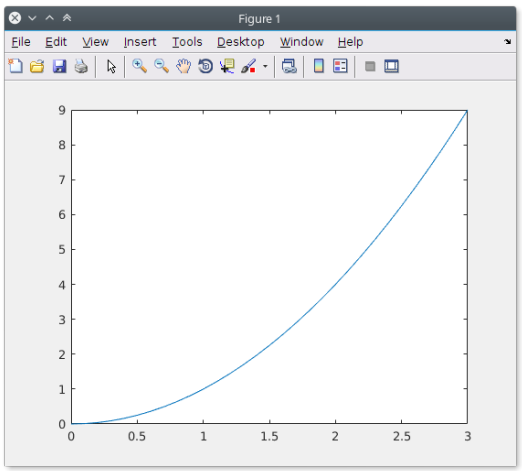
```
>> plot(x, y)
```

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



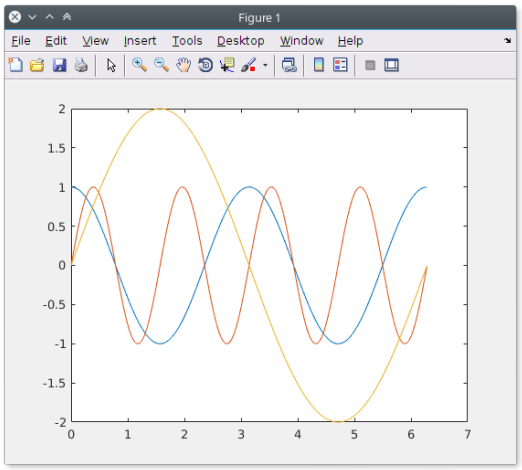
Do note that; the more data points you use, the more precise results you will get. Here we have 4 points (0, 1, 2 and 3), thus we observe a somewhat broken line. Lets try the same with more points by decreasing the increment from 1 to 0.1:

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



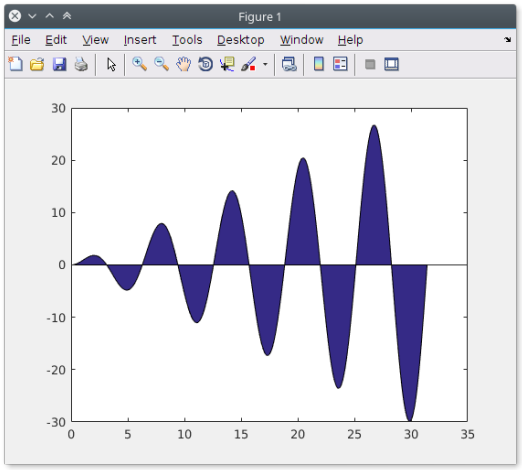
To display more than one plot in one graph, you should keep listing the variable pairs as plot parameters:

```
>> 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)
```



If you'd like to plot a filled graph, you may use **area()** function:

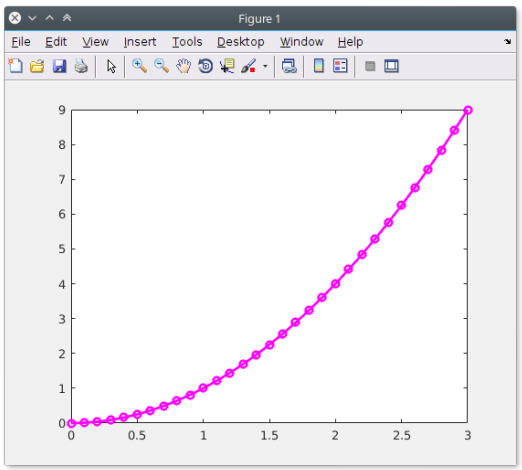
```
>> a = 0:0.01:10*pi;
>> b = sin(a).*a;
>> area(a, b)
```



6 Getting Help

MATLAB provides an extensive help system which can be accessed via **help** command, followed by function name. For example, typing "`help plot`" displays information about plot command, which displays various line formatting options. For example to get a 2 points thick magenta line with circles per data point, we should execute:

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



7 Scripting in MATLAB

When writing large programs in MATLAB, a change in your program may result in executing the same commands from beginning to end over and over again. To overcome this problem, you may create new MATLAB script files (ending with .m extension) and store your MATLAB scripts in these files to modify and run them at any time later.