**CCIT4076 Introduction to Engineering**

***Spring 2022***

HKU SPACE Community College

| | | | | | |
|---|---|---|---|---|---|
| Class: | | Name: | | Date: | |

# Octave's Starter Pack

## *What is Octave?*

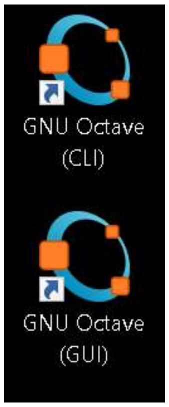
Ocatve is a software tool commonly used by engineers for making numerical computations, for analyzing data. One of the key advantages of Ocatve is that it is interpreted, which means that it can show you the results of your computations right away, just like a regular calculator.

However, Ocatve is considerably more powerful, as we can create programs, which are stored sequences of calculations, very easily. Intermediate results of calculations can either be shown right away, as when Ocatve works like a calculator, or hidden to avoid clutter. These programs, also called Ocatve scripts, are stored in files with the extension *.m, e.g. test.m.

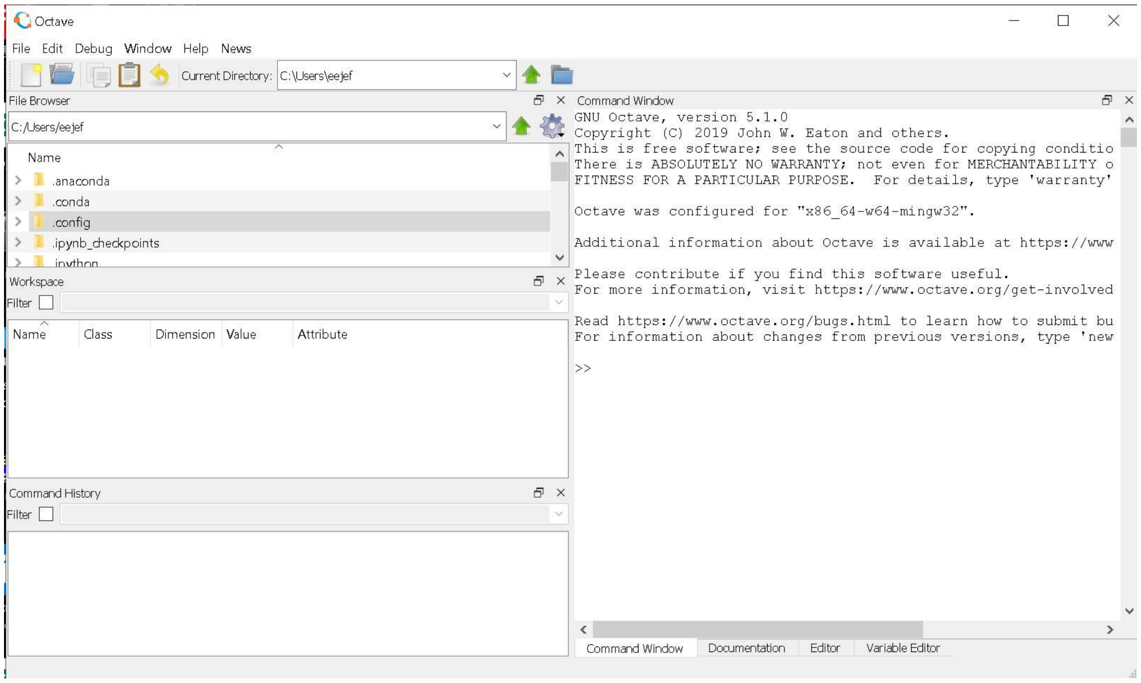
Much of the work in this class will be in modifying existing programs to investigate different tradeoffs in engineering design, i.e. what happens when you change some part of a system design.

## *Starting and Exiting Ocatve*

1. Start Ocatve by accessing it through the Windows "Start" menu, or by clicking on the Ocatve icon on the desktop, which looks something like this:



2. You should see a screen that looks something like this


Octave
File Edit Debug Window Help News
Current Directory: C:\Users\eejef
File Browser
C:/Users/eejef
Name
.anaconda
.conda
.config
.ipynb_checkpoints
.ipython
Workspace
Filter
Name Class Dimension Value Attribute
Command History
Filter
Command Window
Command Window Documentation Editor Variable Editor

```
GNU Octave, version 5.1.0
Copyright (C) 2019 John W. Eaton and others.
This is free software; see the source code for copying conditio
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY o
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'

Octave was configured for "x86_64-w64-mingw32".

Additional information about Octave is available at https://www

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved

Read https://www.octave.org/bugs.html to learn how to submit bu
For information about changes from previous versions, type 'new

>>
```

3. You will be working mostly with the "Command Window" by typing commands at the command prompt, which looks like this

```
>>
```

4. Exit Octave by typing "exit" at the command prompt and hitting return:

```
>> exit
```

You can also exit Octave by selecting "Exit" from the "File" menu at the top left of the screen or simply closing the window by clicking the "x" at the top right side of the screen.

### *Performing calculations in Octave*

Octave can be used just like your normal calculator.  You simply type the expression you wish to find the value of at the command prompt, and Octave evaluates it.  For example, if you type:

```
>> 2+2
```

Octave responds with

```
ans = 4
```

Indicating that the answer is 4.  Try several other calculations using the basic math operators +, -, * and / and see whether Octave returns the right answer.  For example,

```
>> 18 - 9
>> 2 * 3
>> 12 / 4
```

## *Variables*

Just like in algebra, a variable is a name or symbol that can assume a numerical value. You can assign a variable a value simply by performing an assignment operation using the "=" sign. For example, the following command sets the variable "a" equal to 2:

```
>> a = 2
```

When you type this, Octave will confirm the assignment by responding with

```
a = 2
```

Always seeing these confirmations can get a bit confusing, so you can suppress them by putting a semicolon after the value. Try the following command to see that Octave does not give you the confirmation text, but simply responds with another command prompt, allowing you to immediately enter another command.

```
>> b = 2;
>>
```

Now that you have defined values for "a" and "b", you can use them, just as if they were regular numbers. For example, try the following:

```
>> a + b + 6
```

and verify that Octave returns the value that you expect.

Make sure that you define your variables before you use them. Octave will warn you if you do not do this.

In fact, you have already seen an example of variables above. The variable "ans" is used by Octave to indicate the value of the previous expression. For example,

```
>> 2 + 2

ans = 4
>> ans + 2

ans = 6
```

Octave comes with many “built-in” variables. These are variables with special names that are already assigned with their commonly associated values. The ones most valuable for this course will be “pi”, which represents π, and “i” and “j”, which both represent $\sqrt{-1}$.

However, you need to be careful, as Octave will not prevent you from assigning different values to these variables. For example,

```
>> pi

pi = 3.1416

>> pi = 5
pi = 5
```

If you quit Octave, all of the values of the variables you have defined are lost. However, “built-in” variables always have their pre-assigned values.

## *Functions*

Octave has many built in functions that can be used, including the commonly used trigonometric functions, powers and exponentials. These can be applied to both numbers and values. For example, the cosine and sine functions of π/6 can be evaluated by typing the following functions at the command prompt.

```
>> sin(pi/6)
>> cos(pi/6)
```

Verify that $\cos\frac{\pi}{6} = \frac{\sqrt{3}}{2}$ by comparing the last result with

```
>> sqrt(3)/2
```

You can also raise functions to powers using the “^” operator. For example, $5^2$ can be computed by typing

```
>> 5^2
```

Thus, another way to compute the square root is to raise to the 0.5 power. For example, the following two expressions have the same value:

```
>> sqrt(3)/2
>> 3^0.5/2
```

Note that Octave follows common conventions of precedence when evaluating functions. The order in which functions are evaluated can be changed by using parenthesis. For example, compare

```
>> 2*3+5
>> 2*(3+5)
```

You can take a look at the Octave build-in function from the following link:
http://octave.sourceforge.net/octave/overview.html

***Exercise 1***

***Compute the value of the function y = $x^2e^{-2x}$ sin(3x) when x = 5.***

### *Vectors and Matrices in Octave*

One of the primary functions of Octave is to manipulate vectors and matrices. Variables can be used to represent vectors and matrices. In specifying vectors or matrices, square brackets are used to define the start and end. Spaces are used to separate elements on the same row, and semicolons to separate different rows. For example:

```
>> x = [1 2 3]

x =

    1    2    3

>> x = [1; 2; 3]

x =

    1
    2
    3
```

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

A =

    1    2
    3    4
```

The notation A' operator indicates the transpose of A.

```
>> A'

ans =

    1    3
    2    4
```

Standard matrix operations have their common meanings in Octave. For example, the matrix equation

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}3\\ 2\end{bmatrix}=\begin{bmatrix}7\\ 17\end{bmatrix}$$

can be evaluated in Octave as follows (assuming the previous definition of A is still active).

```
>> b = [3; 2];
>> A*b

ans =

    7
   17
```

One good thing about Octave is that the vector size can be increase even the variable is generated. For example

>> v = [ 2 4 6 8 ]

>> v(5) = 10;

v = [2 4 6 8 10] – 1x5 vector

We can also combine two or more vector easily. Example:

>> a= [12 14];

>> b=[v a];

then b = [2 4 6 8 10 12 14]

Similar to vector, matrices (2-D) can be constructed using similar way. Type

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =  1    2    3
     4    5    6    3x3 matrices
     7    8    9
```

We can combine the matrix easily

```
>> r = [10 11 12]            >> r = [10 ; 11; 12]
>> A1 = [A ; r]              >> A2 = [A r]
1    2    3                  1    2    3    10
4    5    6                  4    5    6    11
7    8    9                  7    8    9    12
10   11   12
```

We get obtain a portion of matrix element. e.g. Type

```
>> A3=A1( 1:3 , :     )
     A3 =     1    2    3
              4    5    6
              7    8    9
```

We can extract individual element in the matrices by

```
>> A1(4,2)
     = 11 (Take the element in 4th row and 2nd column)
```

***Exercise 2***

(a) Construct a matrix A where

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 10 & 9 & 8 & 7 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

(b) Construct matrix B from A where B takes only the 3 & 4 columns of rows 2 &3

One common usage of vectors is to represent a set of evenly space numbers. There are two commonly used ways to do this. The first way is to use the ":" operator. The function below creates a horizontal vector of integers from 1 to 3.

```
>> x = [1:3]

x =

     1     2     3
```

We can change the spacing between adjacent integers by providing an increment between the start and end values. For example,

```
>> x = [1:0.5:3]

x =

    1.0000    1.5000    2.0000    2.5000    3.0000
```

The general format of this operator to create a vector of numbers from “a” to “b” with a desired spacing “s” is “[a:s:b]”. When using the colon operator, the square brackets are optional.

The second way is to use the “linspace” function. For example, the function below creates a vector of 5 numbers evenly spaced between 1 and 2.

```
>> x = linspace(1,2,5)

x =

    1.0000    1.2500    1.5000    1.7500    2.0000
```

The general format of this operator to create a vector of “n” numbers from “a” to “b” is “linspace(a,b,n).”

When given a vector or matrix as input, a function that is usually applied to a single scalar argument creates a vector or matrix of values corresponding to the function applied to each element of the matrix. For example,

```
>> sqrt(x)

ans =

    1.0000    1.1180    1.2247    1.3229    1.4142
```

For multiplication of two vectors or matrices of the same size element by element, we can use the “.*” operator, which results in a vector or matrix of the same size where each element is the product of corresponding elements in the. For example,

```
>> y = x.*x
```

```
y =

    1.0000    1.5625    2.2500    3.0625    4.0000

>> x.*y

ans =

    1.0000    1.9531    3.3750    5.3594    8.0000
```

Similar commands are the element-wise division, './" and the element-wise power, e.g.

```
>> x./x

ans =

     1     1     1     1     1

>> x.^2

ans =

    1.0000    1.5625    2.2500    3.0625    4.0000
```

***Exercise 3***

(a) Generate a vector x such that

$x(n) = 15\, e^{-n} \cos(0.125\, \pi\, n)$ $n=0,1,2,3,4 \ldots .20$

(b) Generate a 5x5 matrix M those element (You can use one line to complete the task)

$M(x,y)=x^2y$ x=1 to 5

## *Loops*

In many cases, we wish to repeat the same operation a number of times. One way to do this is by using "for" loops. For example, suppose that we wish to add together the numbers from 1 to 5, we could do it in the following way:

```
>> y = 0;
>> for c=1:5, y = y+1; end
>> y

y =

     5
```

In the first command, we initialize the value of y to zero. In the second command, we execute the commands in the loop (i.e. between the "`for c=1:5`" command and the `end` statement) once for every element of the vector `1:5`. In this example, we add 1 to `y` each time. Since there are five elements, we repeat the loop five times, so the final value of y is 5. The comma and semicolons allow us to put multiple commands on one line.

Each time the loop is executed, the value of "`c`" takes on the next value in the list. In other words, for the code above, the first time the loop is executed, the value of "c" equals 1, the second time it equals 2 and so on. Thus, to add the numbers from one to five we can execute the following code:

```
>> y = 0;
>> for c=1:5,
y = y+c;
end
>> y

y =

     15
```

Note that in this case, we put the commands on separate lines, but Octave does not start executing the loop until it finds the `end` command. This is indicated by the lack of the command prompt ">>".

There are also "while" loops which execute as long as a logical condition is true. For example, the following code does the same calculation as above:

```
>> y = 0;
>> c = 1;
>> while (c < 6), y = y+c; c = c+1; end
>> y

y =

    15
```

The expression `(a < b)` returns the value "1" if `a` is less than `b` and 0 otherwise. The while loop executes whatever is between it and the "end" statement, as long as the expression after the word `while` is nonzero. Thus, the following commands will just cause Octave to run in an endless loop:

```
>> while 1; end
>> while 5; end
>> while -1, end
```

If you type one of these, you can just type "CTRL-c" to stop the loop.

"For" and "while" loops can often be replaced by vector or matrix operations. These typically take fewer lines and run much faster. For example, the operation above can be replaced by the following two commands:

```
>> c=1:5;
>> y = sum(c)

y =

   15
```

The "sum" function simply sums all the values in the vector.

*IF statement*

The general form of IF statement

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

***Exercise 4***

(a) Using FOR loops, generate a 6x6 matrix Y those elements (x,y = 1 to 6)

$Y(x,y) = 2x+y^2$

(b) Using While Loops compute the value of

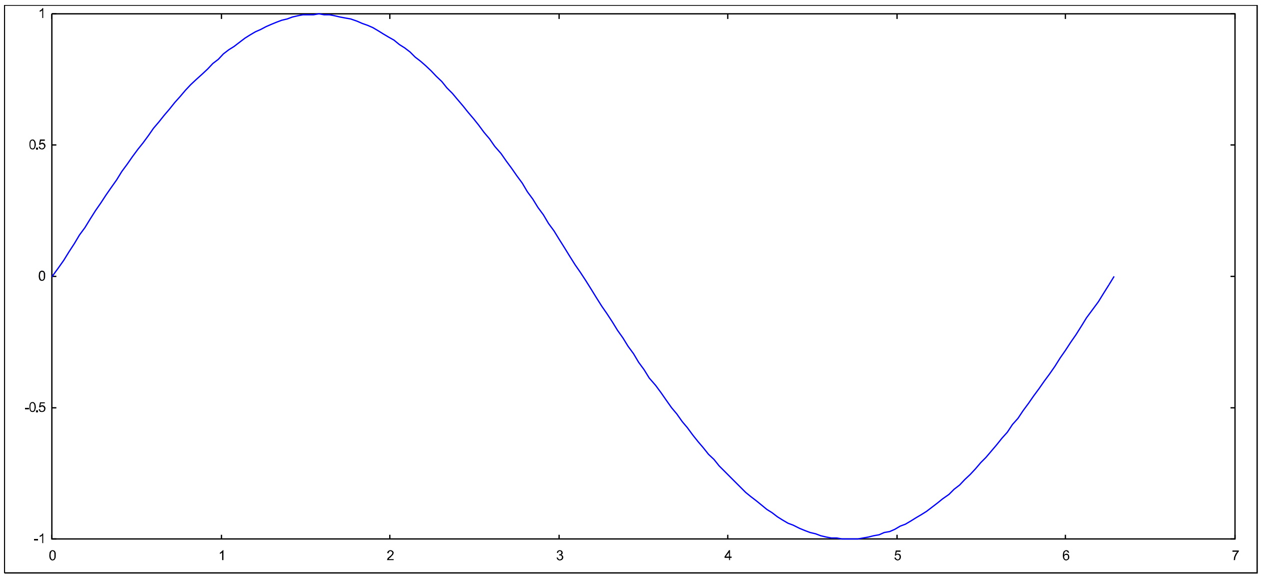
$$\sum_{i=0,\,j=0}^{i=20,\,j=5} i+2j$$

## *Creating plots in Octave*

Another big advantage of Octave is the ability to easily create plots and graphs. The basic way this is done is using the "plot" command, which plots each element of one vector versus another one. For example, to plot one period of a sine wave, we would execute the following commands:

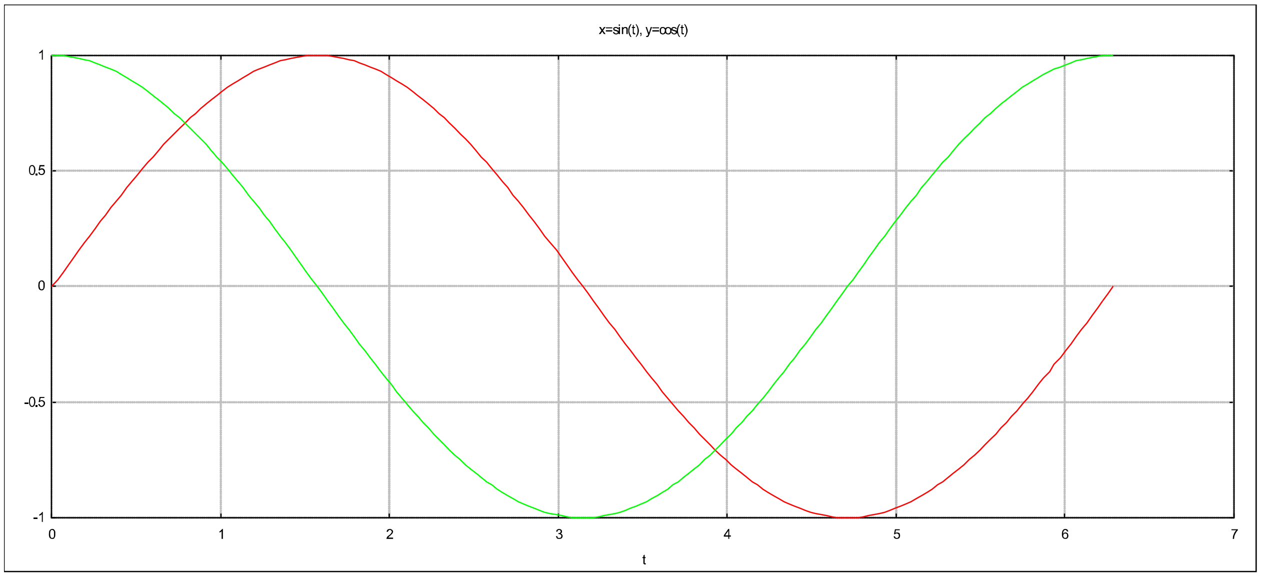
```
>> x = linspace(0,2*pi,200);
>> y = sin(x);
>> plot(x,y);
```

In the first command, we create x, a vector of 200 numbers running from 0 to 2π. In the second command, the sine function is applied to every element of x so that y is also a vector of 200 numbers. In the last command, we generate a plot of every value of y versus its corresponding value of x. When you execute these commands, a new window should appear that looks like this:



We can have more option to choose

```
>> t=0 : pi/100 : 2*pi;
>> x=sin(t);
>> y=cos(t);
>> plot(t, x, 'r-', t, y, 'g—'), grid on
>> title('x=sin(t), y=cos(t)');
>> xlabel('t');
```



We can also plot several signal separately in the figure.

Type

```
>> subplot (m,n,k)
```

it will generate a graph which can have mxn plot. Your current plot will be display on the kth position.

Example

```
>> subplot (2,3,k)
```

| | |
|---|---|
| k=1 | k=2 |
| k=3 | k=4 |
| k=5 | K=6 |

***Exercise 5***

(a) Plot y(t) and z(t) on the same graph, for 5>=t>=1 (with 1000 data points), such that

$$y(t) = 3e^{-0.5t}\cos(\pi t^2)$$

$$z(t) = 2 - y(t)$$

(b) Plot 4 signals on the same graph (set the step size to 0.1)

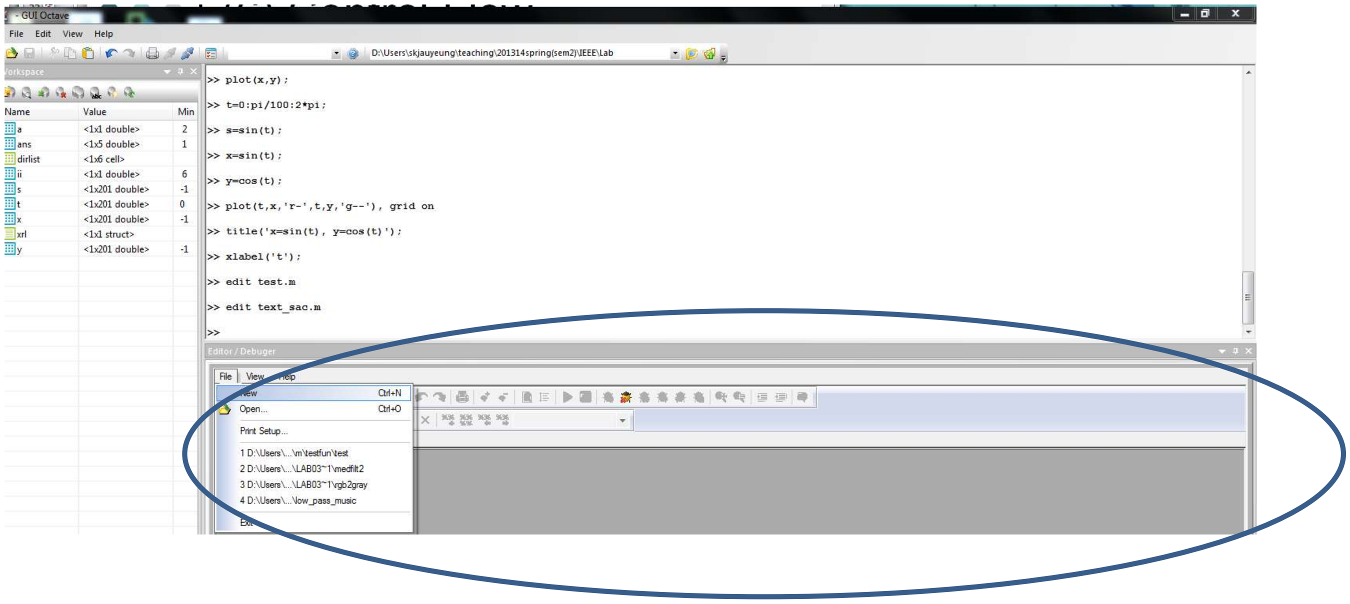
|  |  |  |
|---|---|---|
| (a) | $y(t1)=1+t1^{3;}$ | $-5<t1<5$ |
| (b) | $x(t2)=\cos(2\pi t2)$ | $0<t2<10$ |
| (c) | $z(t3)= (2+1/t3)^2$ | $2<t3<30$ |
| (d) | $w(t4)= 5*t4-8$ | $-100<t4<100$ |

|  |  |
|---|---|
| W | X |
| Y | Z |

## ***Scripts (*.m files)***

Scripts are ways to specify sequences of commands that you wish to do many times, but do not want to type over and over again. Octave scripts are simply text files with a *.m extension. For example

File -> New in the Editor/Debugger

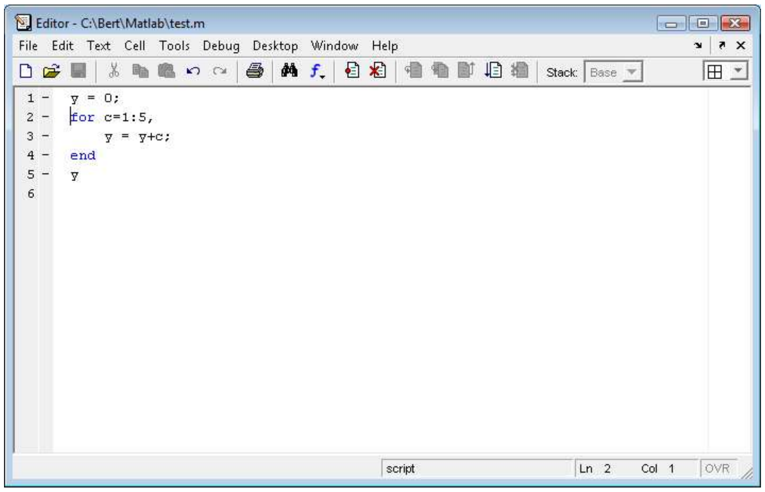

- GUI Octave
File Edit View Help
Workspace
Editor / Debugger
File View Help
New Ctrl+N
Open... Ctrl+O
Print Setup...
1 D:\Users\...\m\testfun\test
2 D:\Users\...\LAB03~1\medfilt2
3 D:\Users\...\LAB03~1\rgb2gray
4 D:\Users\...\low_pass_music
Exit

D:\Users\skjauyeung\teaching\201314spring(sem2)\IEEE\Lab

| Name | Value | Min |
|---|---|---|
| a | <1x1 double> | 2 |
| ans | <1x5 double> | 1 |
| dirlist | <1x6 cell> |  |
| ii | <1x1 double> | 6 |
| s | <1x201 double> | -1 |
| t | <1x201 double> | 0 |
| x | <1x201 double> | -1 |
| xrl | <1x1 struct> |  |
| y | <1x201 double> | -1 |

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

>> t=0:pi/100:2*pi;

>> s=sin(t);

>> x=sin(t);

>> y=cos(t);

>> plot(t,x,'r-',t,y,'g--'), grid on

>> title('x=sin(t), y=cos(t)');

>> xlabel('t');

>> edit test.m

>> edit text_sac.m

>>
```

You can type any set of commands into this window and save it (by selecting “Save” from the “File” menu). For example, type in the following set of commands:


Editor - C:\Bert\Matlab\test.m
File Edit Text Cell Tools Debug Desktop Window Help
Stack: Base
script Ln 2 Col 1 OVR

```
y = 0;
for c=1:5,
    y = y+c;
end
y
```

and save the file as test.m.  Now in the command window, when you type the "test", you should see the following:

```
>> test
y = 15
```

Let's try another one

```
% comment text - test_2.m script
Clear             % remove all the variable from the work space
x=2*pi*[0:0.1:1];
x=sin(x);
stem(y)           % plot Y with the data sequence Y as stems from the x axis
```

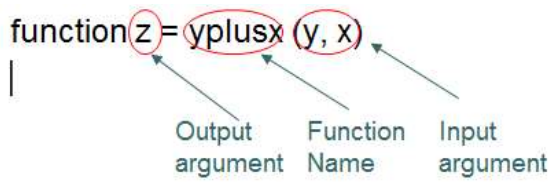
Try to see the result by running test_2.m

***Function***

- Function M-file can accept input and return output arguments.  Functions operate on variables within their own workspace which is separated from the workspace you access on the Octave command prompt.

- Example
  -- open a new M-file which similar to Scirpts
  Type
  function z = yplusx (y, x)
  Z=y+x;

--save the file as yplusx.m


```
function z = yplusx (y, x)
```

In the command window

type

```
>> B=yplusx(7,8)

B=15

>> C=yplusx(B,12)

C=27
```

***Exercise 6***

Given that

$y(t) = 3e^{-0.5}t\cos(\pi t^2)$ 5>t>1 (step = 0.01)

Write a function to plot y(t-t0) (call new_plot). The output should show the plot of y(t-t0) together with the original y(t), then use the function to plot y(t-t0) for t0=0.5. The function will have the output argument on the average different for these two signals.