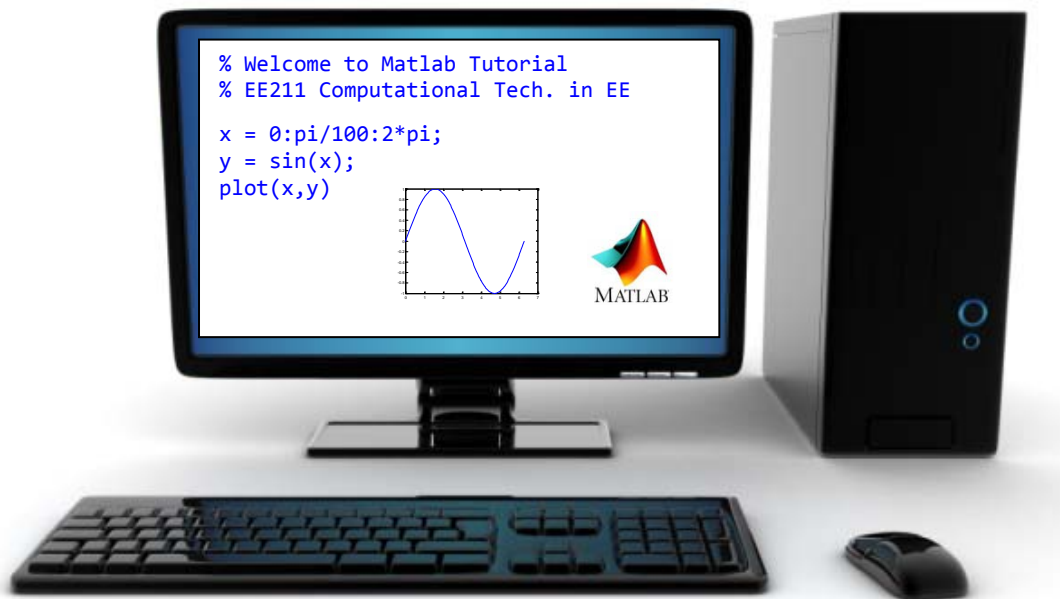


King Saud University



College of Engineering, Electrical Engineering Department

MATLAB TUTORIAL: EE 211 Computational Techniques in Electrical Engineering



Student Name:

ID Number :

Section :

MATLAB TUTORIAL
for “EE 211 Computational Techniques in EE”

Course Description EE211 (2, 0, 2):

Linear and Nonlinear Systems of Equations; Concept of Error; Least Square Methods; Function Expansions; Taylor Series; Numerical Integration; Initial Value and Boundary Value Problems for ODE and PDE; Introduction to Finite Difference and Finite Element Methods; Introduction to Optimization Techniques.

Textbook: K. Atkinson and W. Han, “Elementary Numerical Analysis, John Wiley & Sons, L.E.
Pre-requisite: GE 211. Co-requisite: MATH 244. [EE KSU new plan, 2008]

Grading Policy: -

(Attendances, quizzes, final exam)

Contents:

Labworks	Material	Week #
Lab 01	MATLAB: Introduction	
Lab 02	MATLAB: Vector and Matrix	
Lab 03	MATLAB: Plotting	
Lab 04	MATLAB: Flow control	
Lab 05	MATLAB: Function, saving data, ...etc	
Lab 06	Taylor Approximation	
Lab 07	Root Finding by Bisection Method	
Lab 08	Interpolation I: Polyfit	
Lab 09	Interpolation II: Newton's formula	
Lab 10	Numerical Integration I: Trapezoidal Rule	
Lab 11	Numerical Integration II: Simpson Rule	

Note: *This manual is still a draft to make easy the instructor and the student during the tutorial class. Further improvements are still needed to enhance this manual. I just compiled this manual from the previous available materials: Lab01-05 from slide “Introduction to MATLAB” by Umut Orguner, lab06-11 by Dr Mubashir Alam. (Sutrisno Ibrahim: suibrahim@ksu.edu.sa)*

Lab (1): Introduction to MATLAB

1. About MATLAB:

- MATLAB: MATrix LABoratory
- A software environment for interactive numerical computations. Examples:
 - Matrix computations and linear algebra
 - Solving nonlinear equations
 - Numerical solution of differential equations
 - Mathematical optimization
 - Statistics and data analysis
 - Signal processing ... Etc

2. MATLAB environment:

Important part:

1. Command Window

- type commands / code

2. Current Directory

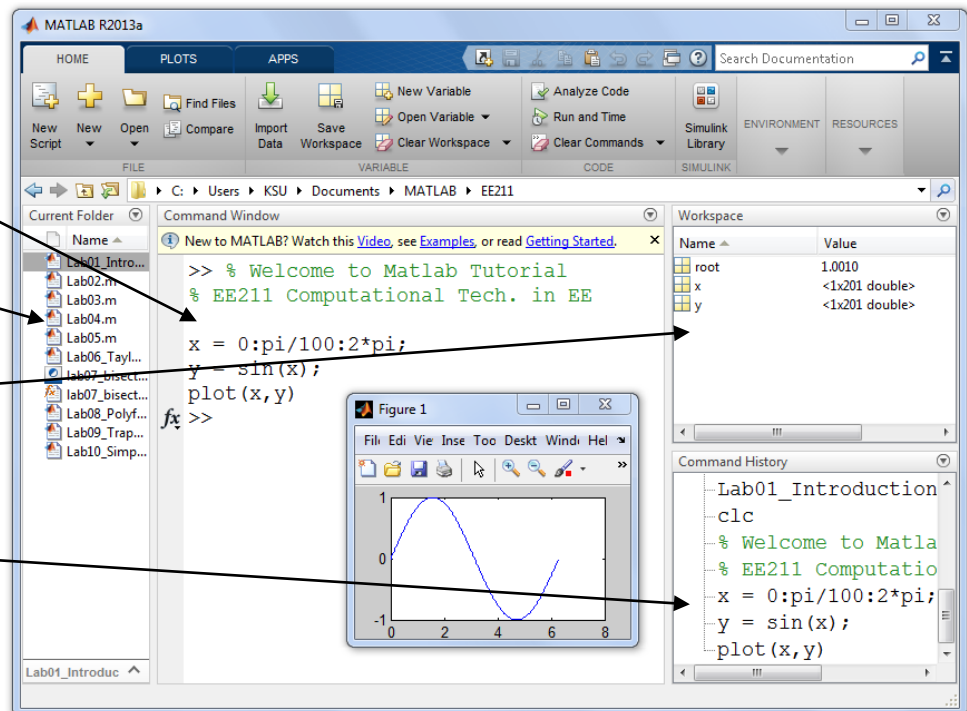
- View folders and m-files

3. Workspace

- View program variables
- Double click on a variable to see it in the Array Editor

4. Command History

- View past commands
- Save a whole session using diary



3. How to write a code/program in MATLAB?

- First, open MATLAB by choosing **Start > (Program Files) > Matlab R2013a** (version may different).
- There are several ways to write a code: using Command Window, M-File, GUI, etc.
- We will start using *Command Window*, write the following code (or copy-paste) in the *Command Window* and then pres “**Enter**”.

```
% Welcome to Matlab Tutorial
% EE211 Computational Tech. in EE
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

4. MATLAB Variable Names:

- Variable names ARE case sensitive
- Variable names must start with a letter followed by letters, digits, and underscores.

MATLAB Special Variables:

- *Ans* Default variable name for results
- *pi* Value of π
- *eps* Smallest incremental number
- *inf* Infinity
- *NaN* Not a number e.g. 0/0
- *i* square root of -1

5. MATLAB Math & Assignment Operators:

- Power : \wedge or $\cdot\wedge$ a^b or $a.\wedge b$
 - Multiplication : $*$ or $\cdot*$ $a*b$ or $a.*b$
 - Division : $/$ or $\cdot/$ a/b or $a./b$
or \backslash or $\cdot\backslash$ $b\backslash a$ or $b.\backslash a$
- NOTE: $56/8 = 8\backslash 56$
- Addition : $+$ $a + b$
 - Subtraction : $-$ $a - b$
 - Assignment : $=$ $a = b$ (assign b to a)

Interactive Calculations:

- Matlab is interactive, no need to declare variables


```
>> a=5
>> b=a/2
>> a=2+3*4/2
>> a=5e-3;
b=1;
c=a+b
```
- Most elementary functions and constants are already defined


```
>> cos(pi)
>> abs(1+i)
>> sin(pi)
```
- Last call gives answer 1.2246e-016 !?

6. Important command for further information:

>>help functionname (example: >>help plot)
>>lookfor keyword

Lab (2): Vector and Matrix

In this lab we will learn about vectors and matrices and some operations on them. MATLAB works a lot with matrix to represent or handle the data. Here several important points:

1. What are vector and matrix?

- Vectors (arrays) are defined as
- `>> v = [1, 2, 4, 5]`
- `>> w = [1; 2; 4; 5]`

$$v = [1 \ 2 \ 4 \ 5]$$

$$w = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}$$

- Matrices (2D arrays) defined similarly
- `>> A = [1,2,3;4,-5,6;5,-6,7]`

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

2. Generating Vectors/Matrices from functions:

- | | |
|---|---|
| ■ <code>zeros(M,N)</code> <u>MxN</u> matrix of <u>zeros</u> | <code>x = zeros(1,5)</code> |
| <code>>> zeros(5,1);</code> | <code>x =</code>
<code>0 0 0 0 0</code> |
| <hr/> | |
| ■ <code>ones(M,N)</code> <u>MxN</u> matrix of <u>ones</u> | <code>x = ones(1,3)</code> |
| | <code>x =</code>
<code>1 1 1</code> |
| <hr/> | |
| ■ <code>rand(M,N)</code> <u>MxN</u> matrix of uniformly distributed random numbers on (0,1) | <code>x = rand(1,3)</code> |
| <code>A=rand(5); B=rand(1,5); C=rand(5,1)</code> | <code>x =</code>
<code>0.9501 0.2311 0.6068</code> |
| <code>a=randn(5); b=randn(1,5); c=randn(5,1)</code> | |

You may also generate matrices with any size using the same function, such as:

`>>zeros(3,4)` ---> to generate a zero matrix with size 3 by 4.

3. Matrix operators:

- All common operators are overloaded
- `>> v + 2`
- Common operators are available
- `>> B = A'`
- `>> A*B`
- `>> A+B`
- Note:
Matlab is case-sensitive
- **A and a are two different variables**
- Transponate conjugates complex entries; avoided by
`>> B=A'`

$$v = [1 \ 2 \ 4 \ 5]$$

$$w = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

4. Indexing Matrices:

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

- Index complete row or column using the colon operator

■ `>> A(1,:)`

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

- Can also add limit index range

■ `>> A(1:2,:)`

■ `>> A([1 2],:)`

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

- General notation for colon operator

■ `>> v=1:5`

■ `>> w=1:2:5`

■

$$v = [1 \ 2 \ 3 \ 4 \ 5]$$

$$w = [1 \ 3 \ 5]$$

5. Concatenation (Combination) of Matrices:

■ $x = [1 \ 2], y = [4 \ 5], z = [0 \ 0]$

$A = [x \ y]$

1 2 4 5

$B = [x ; y]$

1 2
4 5

$C = [x \ y ; z]$
 Error:
 ??? Error using ==> vertcat CAT arguments dimensions are not consistent.

6. Operators (arithmetic):

Operators: + addition, - subtraction, * multiplication, / division, ^ power

Given A and B:

<pre>>> A = [1 2 3;4 5 6;7 8 9]</pre> <p>A =</p> <table style="margin-left: 40px;"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td><td>6</td></tr> <tr><td>7</td><td>8</td><td>9</td></tr> </table>	1	2	3	4	5	6	7	8	9	<pre>>> B = [3 5 2; 5 2 8; 3 6 9]</pre> <p>B =</p> <table style="margin-left: 40px;"> <tr><td>3</td><td>5</td><td>2</td></tr> <tr><td>5</td><td>2</td><td>8</td></tr> <tr><td>3</td><td>6</td><td>9</td></tr> </table>	3	5	2	5	2	8	3	6	9
1	2	3																	
4	5	6																	
7	8	9																	
3	5	2																	
5	2	8																	
3	6	9																	

Addition	Subtraction	Product	Transpose																																				
<pre>>> X = A + B</pre> <p>X =</p> <table style="margin-left: 40px;"> <tr><td>4</td><td>7</td><td>5</td></tr> <tr><td>9</td><td>7</td><td>14</td></tr> <tr><td>10</td><td>14</td><td>18</td></tr> </table>	4	7	5	9	7	14	10	14	18	<pre>>> Y = A - B</pre> <p>Y =</p> <table style="margin-left: 40px;"> <tr><td>-2</td><td>-3</td><td>1</td></tr> <tr><td>-1</td><td>3</td><td>-2</td></tr> <tr><td>4</td><td>2</td><td>0</td></tr> </table>	-2	-3	1	-1	3	-2	4	2	0	<pre>>> Z = A * B</pre> <p>Z =</p> <table style="margin-left: 40px;"> <tr><td>22</td><td>27</td><td>45</td></tr> <tr><td>55</td><td>66</td><td>102</td></tr> <tr><td>88</td><td>105</td><td>159</td></tr> </table>	22	27	45	55	66	102	88	105	159	<pre>>> T = A'</pre> <p>T =</p> <table style="margin-left: 40px;"> <tr><td>1</td><td>4</td><td>7</td></tr> <tr><td>2</td><td>5</td><td>8</td></tr> <tr><td>3</td><td>6</td><td>9</td></tr> </table>	1	4	7	2	5	8	3	6	9
4	7	5																																					
9	7	14																																					
10	14	18																																					
-2	-3	1																																					
-1	3	-2																																					
4	2	0																																					
22	27	45																																					
55	66	102																																					
88	105	159																																					
1	4	7																																					
2	5	8																																					
3	6	9																																					

Operators (Element by Element):

- .* element-by-element multiplication
- ./ element-by-element division

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

```
x=A(1,:)
x=
     1     2     3
y=A(:,3)
y=
     3     4    -1
```

```
b=x.*y
b=
     3     8    -3
c=x./y
c=
    0.33    0.5    -3
d=x.^2
d=
     1     4     9
```

```
K=x^2
```

```
Error:
```

```
??? Error using ==> mpower Matrix must be square.
```

```
B=x*y
```

```
Error:
```

```
??? Error using ==> mtimes Inner matrix dimensions must agree.
```


Lab (3): PLOTTING

In this lab we will practice how to make several basic plotting in MATLAB:

1. Plot the function $\sin(x)$ between $0 \leq x \leq 4\pi$

- Create an x-array of 100 samples between 0 and 4π .

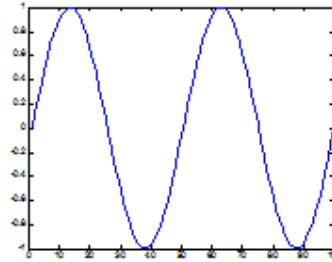
```
>>x=linspace(0,4*pi,100);
```

- Calculate $\sin(\cdot)$ of the x-array

```
>>y=sin(x);
```

- Plot the y-array

```
>>plot(y)
```



2. Plot the function $e^{-x}/3\sin(x)$ between $0 \leq x \leq 4\pi$

- Create an x-array of 100 samples between 0 and 4π .

```
>>x=linspace(0,4*pi,100);
```

- Calculate $\sin(\cdot)$

```
>>y=sin(x);
```

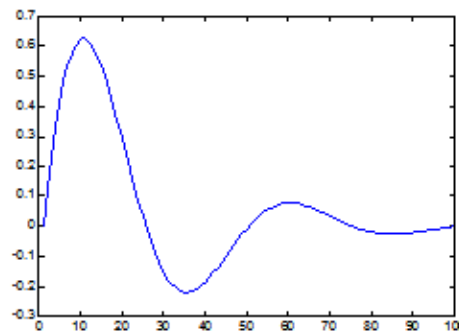
- Calculate $e^{-x/3}$

```
>>y1=exp(-x/3);
```

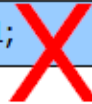
- Multiply arrays y and y1

```
>>y2=y.*y1;
```

```
>>plot(y2)
```



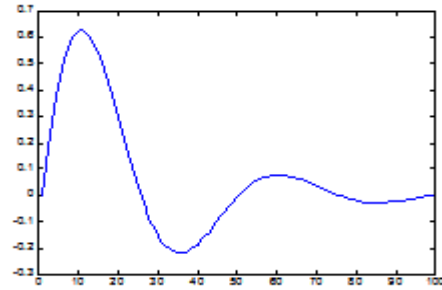
```
>>y2=y*y1;
```



3. Display facilities: plot, stem, title, xlabel, ylabel, etc

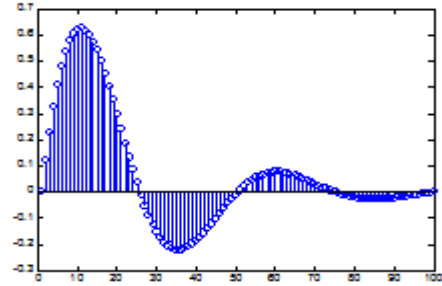
■ plot(.)

```
Example:
>>x=linspace(0,4*pi,100);
>>y=sin(x);
>>plot(y)
>>plot(x,y)
```



■ stem(.)

```
Example:
>>stem(y)
>>stem(x,y)
```



■ title(.)

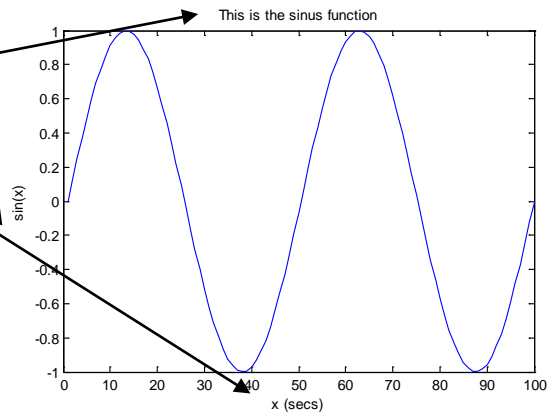
```
>>title('This is the sinus function')
```

■ xlabel(.)

```
>>xlabel('x (secs)')
```

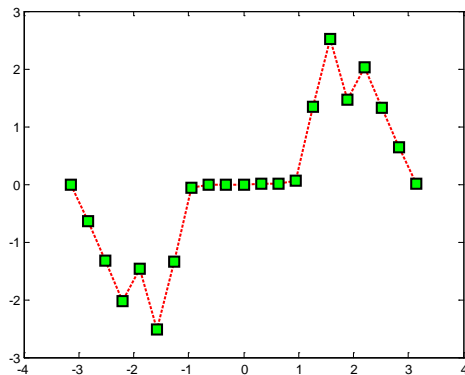
■ ylabel(.)

```
>>ylabel('sin(x)')
```



You may create line plot using specific line width, marker color, and marker size (for more detail see `>> help plot`). For example:

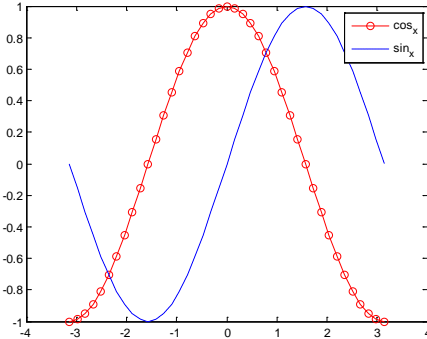
```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
      'MarkerEdgeColor','k',...
      'MarkerFaceColor','g',...
      'MarkerSize',10)
```



4. Another Plotting style/Display in MATLAB:

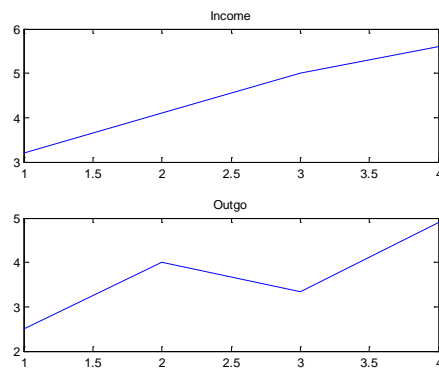
- Display several curves in a plot (add legend for each curve)

```
figure
x = -pi:pi/20:pi;
plot(x,cos(x), '-ro',x,sin(x), '-.b')
hlegl = legend('cos_x', 'sin_x');
```



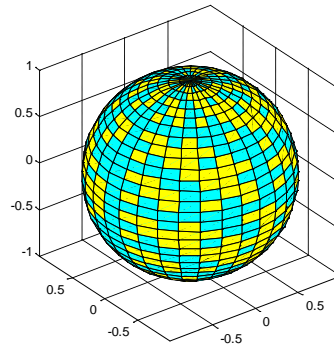
- Make several plots in one figure (using subplot)

```
income = [3.2,4.1,5.0,5.6];
outgo = [2.5,4.0,3.35,4.9];
subplot(2,1,1); plot(income)
title('Income')
subplot(2,1,2); plot(outgo)
title('Outgo')
```



- 3-D shaded surface plot

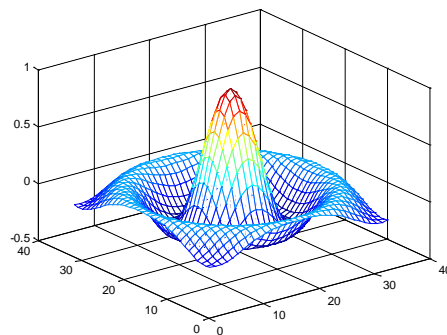
```
k = 5;
n = 2^k-1;
[x,y,z] = sphere(n);
c = hadamard(2^k);
surf(x,y,z,c);
colormap([1 1 0; 0 1 1])
axis equal
```



- MESH Plot

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
```

```
figure
mesh(Z)
```



Lab (4): Flow Control

In this lab we will learn how to do flow control in MATLAB. We will learn about if-condition, for-loop, while-loop and others.

1. For-loop

Execute statements specified number of times.

Syntax	Example
<pre>for index = values program statements : end</pre>	<pre>% Create a Hilbert matrix using nested for loops % Hilbert matrix is a square matrix with entries being the unit fractions k = 5; hilbert = zeros(k,k); % Pre-allocate matrix for m = 1:k for n = 1:k hilbert(m,n) = 1/(m+n -1); end end hilbert</pre>

2. While-loop

Repeatedly execute statements while condition is true.

Syntax	Example
<pre>while expression statements end</pre>	<pre>% Find the first integer n for which factorial(n) is a 5-digit number. % Remember, factorial (n)=1x2x3x....xn n = 1; nFactorial = 1; while nFactorial < 1e5 n = n + 1 nFactorial = nFactorial * n end</pre>

3. If/elseif/else statement

Execute statements if condition is true.

Syntax	Example
<pre>if expression statements elseif expression statements else statements end</pre>	<pre>A = rand(1,10) limit = .75; B = (A > limit) % B is a vector of logical values if any(B) fprintf('Indices of values > %4.2f: \n', limit); disp(find(B)) else disp('All values are below the limit.') end</pre>

4. Switch/case/otherwise statement

Switch among several cases based on expression.

Syntax	Example
<pre>switch switch_expression case case_expression statements case case_expression statements : otherwise statements end</pre>	<pre>% Conditionally display different text depending on a value entered at the command line: mynumber = input('Enter a number:'); switch mynumber case -1 disp('negative one'); case 0 disp('zero'); case 1 disp('positive one'); otherwise disp('other value'); end</pre>

5. Another flow control: Break condition, continue, pause

- Break: terminate execution of for or while loop.

Syntax	Example
<pre>break</pre>	<pre>% Continuously generate random number until the generated number more than 0.9 while (1) x=rand if (x>0.9), break; end end</pre>

- Continue: pass control to next iteration of for or while loop.

Syntax	Example
<pre>continue</pre>	<pre>% Count the number of lines of code in the file magic.m, skipping all blank lines and comments: fid = fopen('magic.m','r'); count = 0; while ~feof(fid) line = fgetl(fid); if isempty(line) strcmp(line, '%',1) ~ischar(line) continue end count = count + 1; end fprintf('%d lines\n',count); fclose(fid);</pre>

- Pause: halt execution temporarily.
example:
pause %wait until any key
pause(3) %wait 3 seconds

6. Logical operators

Logical operators are very useful for flow control, but they can be used also for other purposes.

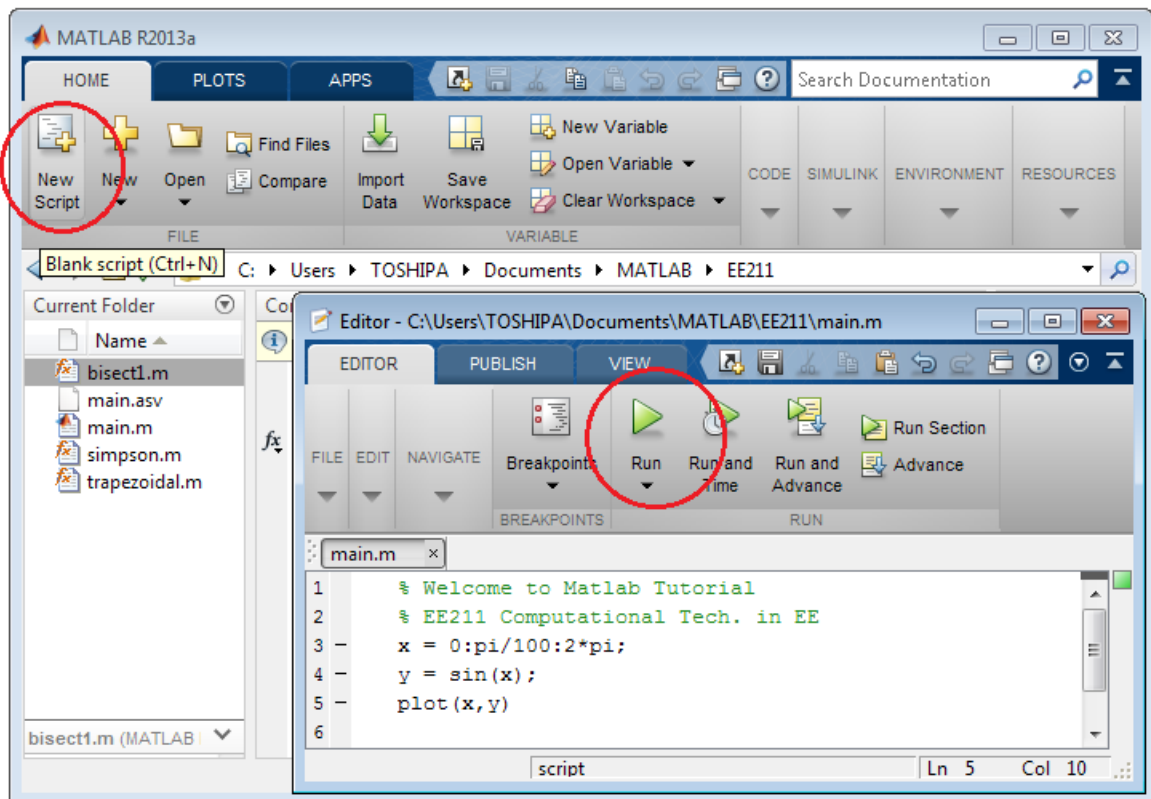
==	Equal to
~=	Not equal to
<	Strictly smaller
>	Strictly greater
<=	Smaller than or equal to
>=	Greater than equal to
&	And operator
	Or operator

Lab (5): M-File, Function, Data Handling, etc

In this lab we will learn about M-File, user defined functions, loading and saving data, etc:

1. Using M-File

- You may use M-file to write script, function or program.
- To create a new M-File, click "New Script" in the menu button. You may also use short cut "*Ctrl+N*" from the keyboard.
- Write the program that you want and then save it in ".m" extension such as "*main.m*". To execute your program, you just click "Run" button in the M-File editor.



2. User-defined functions

- MATLAB already has many pre-defined functions (such as `sin()`, `cos()`, `round()`, `floor()`, `rand()`, etc). However the users may also define their own function using MATLAB.
- Functions are m-files which can be executed by specifying some inputs and supply some desired outputs.
- The code telling the Matlab that an m-file is actually a function is


```
function out1=functionname(in1)
function out1=functionname(in1,in2,in3)
function [out1,out2]=functionname(in1,in2)
```

- You should write this command at the beginning of the m-file and you should save the m-file with a file name same as the function name.

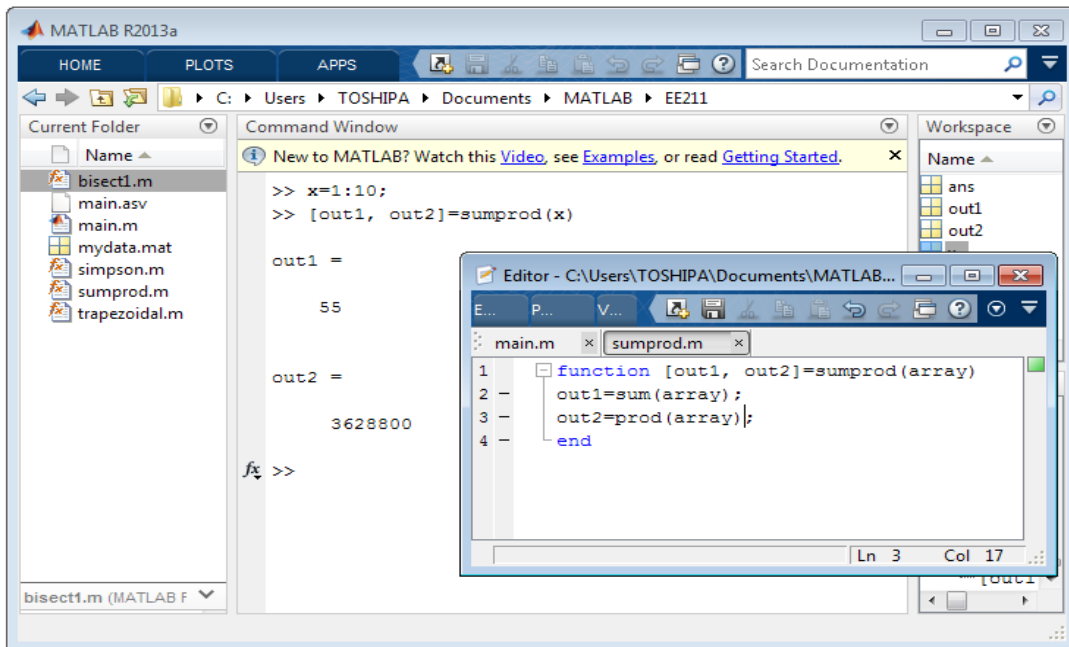
Example:

Make a function which takes an input array and returns the sum and product of its elements as outputs.

```
function [out1, out2]=sumprod(array)
out1=sum(array);
out2=prod(array);
end
```

The function sumprod(.) can be called from command window or an m-file as follow:

```
>> x=1:10;
>> [out1, out2]=sumprod(x)
```



3. Data handling

Sometime we need to load data to MATLAB or saving data to a file. You may used "load" function or other specific file reader/writer.

Example:

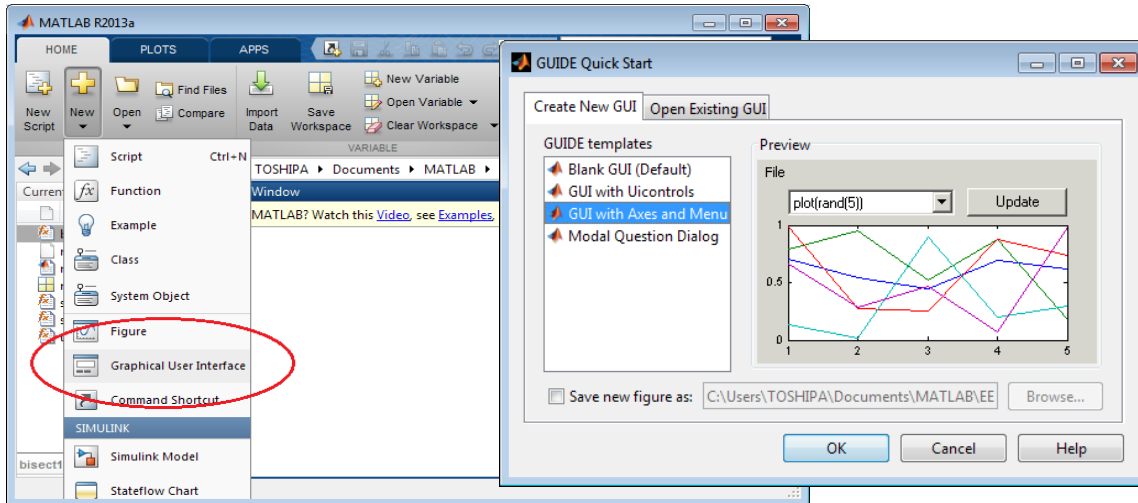
% Create an file from several 4-column matrices, and load the data back into a double array named mydata:

```
clc; clear all;
a = magic(4);
b = ones(2, 4) * -5.7;
c = [8 6 4 2];
save mydata.mat a b c;
clear a b c
```

```
load mydata.mat % Load all variable (a, b, c)
x=load('mydata.mat', 'a'); % Load only specific variable
```

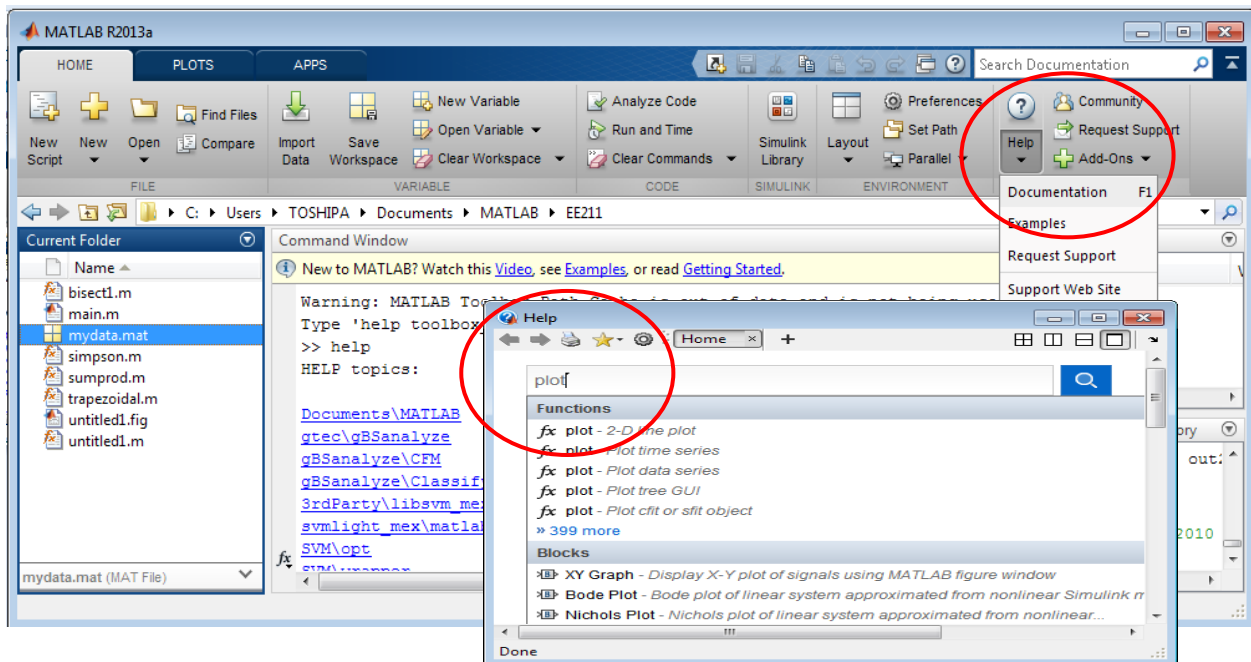

4. Graphical User Interface (GUI)

You may also create program using GUI. Click "New", then select "Graphical User Interface". You may choose create blank GUI or start with existing template (just select and then press "OK"). (There are a lot of material in the Internet to learn more about MATLAB GUI).



5. MATLAB HELP

MATLAB help is very useful to learn more about MATLAB and its functions. There are a lot of examples provided by MATLAB. To open MATLAB help just click "Help", then "Documentation". Then you may search function or example by writing it in the search bar, then press *Enter*.



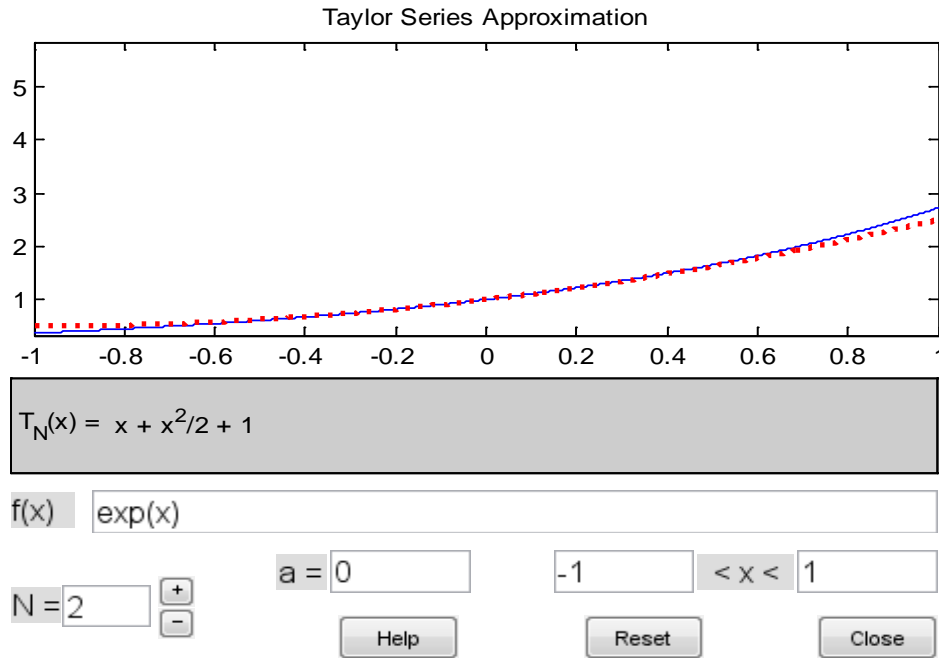
Lab (6): Taylor's Polynomial using MATLAB Tools

Topic#1: Use of Matlab GUI for Taylor's Polynomial: `taylor` tool

Matlab has a very interesting device that computes and plots a function and its Taylor polynomial. A pop-up window is obtained by typing

`>> taylor`

in Matlab command window. Try this. You should see the following window:



The window contains a region where the graph of f (in blue) is plotted with the graph of the Taylor polynomial $p_n(x)$ (in dashed red). You can

- type a new function inside the box next to $f(x) =$,
- choose the order N of the Taylor polynomial (either type it or use the arrows to increase or decrease it),
- choose the point a where the Taylor polynomial is centered,
- choose the region along the x -axis where the functions are plotted.

Topic #2: Computing the Taylor's polynomial for the function and its Error using Matlab

We will learn the use of Matlab function `taylor` () to calculate the Taylor's polynomial for a given function. Then use Matlab to plot the error between the given function and its Taylor's polynomial representation. This error is usually plotted as a function of x .

We know how to measure the error between the values of $f(x)$ and $p_n(x)$ at a point x . This error is

$$|f(x) - p_n(x)|.$$

On the other hand, $f(x)$ and $p_n(x)$ are defined over an interval $[a, b]$ and for some choices of x , this error may be large while for some other values of x this error may be small. What do we do when have to consider a whole range of values for x ? Take the largest one!

Definition The *error* between f and its n th order Taylor's polynomial over the interval $[a,b]$ is the maximum value of

$$|f(x) - p_n(x)| \text{ ----- (Eq\#1)}$$

when x belongs to $[a, b]$.

In practice, the **Eq\#1** as given above is a function of x and by plotting it we can estimate its largest value. Let's do this for $\cos(x)$ and its Taylor's polynomial of order 4. Type the following:

```
>> syms x
>> f=cos(x);
>> t4=taylor(f,'Order',5,'ExpansionPoint',0);
>> ezplot(abs(f-t4),[-2,2]);
```

What do these matlab commands represent?

The first line takes the variable 'x' as symbolic, and then generate a symbolic function 'f' using symbolic variable 'x'. (For symbolic representations see the Symbolic Toolbox in Matlab).

The expression **t4** is the 4th order (note for 4th order we use 5) Taylor's polynomial for f at $x=0$. For the above function **t4** should be:

$$t4 = x^4/24 - x^2/2 + 1$$

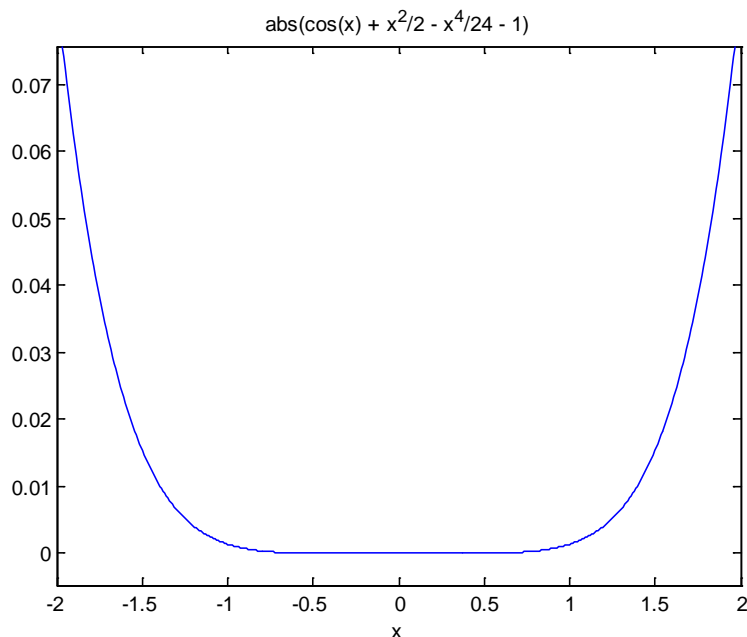
The **abs** command stands for absolute value. The **ezplot** command plotted the **Eq\#1** over the interval $[-2,2]$.

Unfortunately, we can't see the largest value of the graph because it blows up near the ends of the interval. Let's resize our screen with the **axis** command. Try

```
>> axis([-2,2,0,0.1]);
```

- You can also change the y-axis limits by directly editing the plot in the Figure window.

The final plot is shown below



Now we can see that the graph of the **Eq\#1** is always smaller than 0.09. Therefore we claim that the error between f and $t4$ over the interval $[-2, 2]$ is less than 0.09.

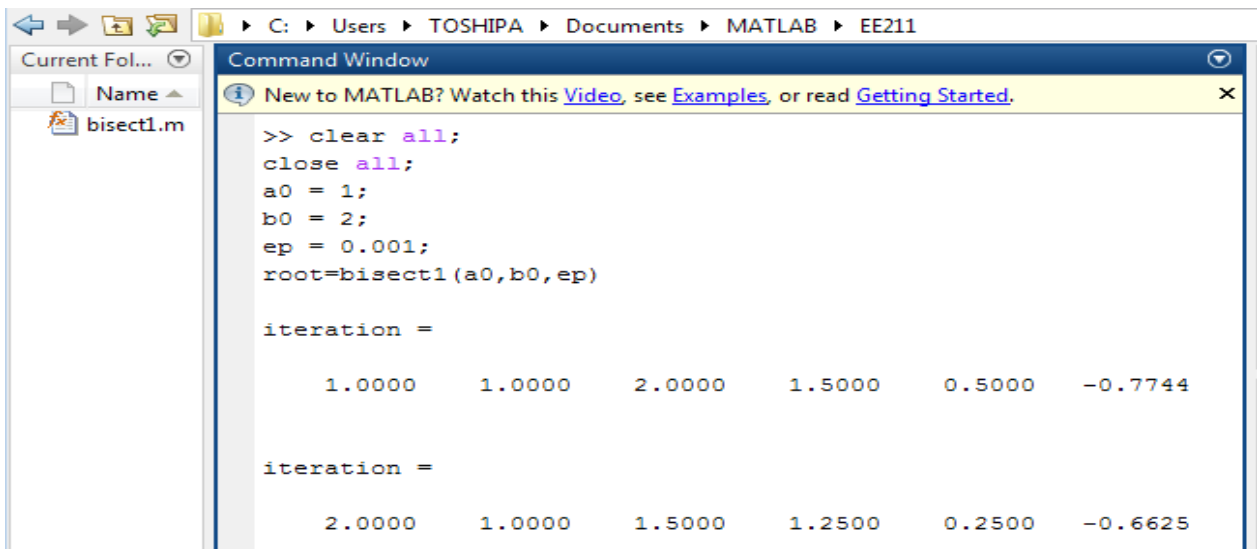
Lab (7): Root Finding by Bisection Method

Bisection Method:

- An implementation of the bisection method in Matlab is given in the textbook (page 75) and is made available to you with minor modification as Matlab function **bisect1** saved as file **bisect1.m**. You can download this file from course website (Or just copy-paste from the code bellow).
- The following script calls the function **bisect1** for the example 3.1.1 of the textbook (page73), with $f(x) = x^6 - x - 1$ and values of interval as $[a, b] = [1, 2]$ and the tolerance of 0.001.

```
clear all;
close all;
a0 = 1;
b0 = 2;
ep = 0.001;
root=bisect1(a0,b0,ep)
```

- Type/Download this script in Matlab and save it as a file.
- Run this script and compare your results with Table 3.1 (page 73) of the textbook.
- Verify your results upto $n = 5$ by doing the calculations using your calculator as done in class
- Change the error tolerance (ep) to 0.00001, does your answer change, does number of iteration changes?
- From Matlab help command find the use of “eps” function. Change the error tolerance to machine precision (ep = eps) does your answer change?
- Change the error tolerance to $1e-18$, does your answer change, if no, why?
- Carefully study the Matlab code bisect1.m and make sure you understand the algorithm and Matlab commands.
- Implement the steps in Lab7HW. You are not supposed to turn in this HW assignment. However, there will be Lab Quiz next week based on this HW assignment.



```
>> clear all;
close all;
a0 = 1;
b0 = 2;
ep = 0.001;
root=bisect1(a0,b0,ep)

iteration =

    1.0000    1.0000    2.0000    1.5000    0.5000   -0.7744

iteration =

    2.0000    1.0000    1.5000    1.2500    0.2500   -0.6625
```

```
% Lab 07: Root Finding by Bisection Method
% Save this code as "bisect1.m"
function root=bisect1(a0,b0,ep)
%
% function bisect1(a0,b0,ep)
%
% This is the bisection method for solving an equation f(x)=0.
%
% The function f is defined below by the user. The function f is
% to be continuous on the interval [a0,b0], and it is to be of
% opposite signs at a0 and b0. The quantity ep is the error
% tolerance.
% For the given function f(x), an example of a calling sequence
% might be the following:
%     root = bisect1(1,2,0.001)
%
%
% The following will print out for each iteration the values of
%     count, a, b, c, b-c, f(c)

format short
a = a0; b = b0;
fa = f(a);
fb = f(b);

c = (a+b)/2;
it_count = 0;
while b-c > ep
    it_count = it_count + 1;
    fc = f(c);
%     Internal print of bisection method. Tap the carriage
%     return key to continue the computation.

    iteration = [it_count a b c b-c fc]
    if sign(fb)*sign(fc) <= 0
        a = c;
        fa = fc;
    else
        b = c;
        fb = fc;
    end
    c = (a+b)/2;
    pause
end

root = c
error_bound = b-c
```

```
it_count

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function value = f(x)
    % value = tan(x) - x;
    value=exp(-x)-sin(x);
end

end
```

Homework Lab #07

Question #1

- For the exercise done in the lab, plot the function $f(x) = x^6 - x - 1$ over the interval $[a, b] = [1, 2]$ in Matlab, verify there is a change of sign at the endpoints for this function, what does this tell you?
- Change the interval $[a, b]$ to $[-1, 2]$ and again plot the function, what do you observe?

Question #2

Modify the Matlab code to find the root of the functions given in the problem 1 (page 77) of your textbook. Note in each case you must first plot the function to find the interval $[a, b]$ which contains only one root

Lab (8): Polynomial Interpolation using Polyfit

1. Use of Matlab command polyfit () to implement polynomial interpolation and use of polyval () to evaluate the polynomial.

To see what the function polyfit() does, type this at Matlab command window:

```
>> help polyfit
```

And it gives the following definition:

$P = \text{POLYFIT}(X,Y,N)$ finds the coefficients of a polynomial $P(X)$ of degree N that fits the data Y best in a least-squares sense. P is a row vector of length $N+1$ containing the polynomial coefficients in descending powers, $P(1)*X^N + P(2)*X^{(N-1)} + \dots + P(N)*X + P(N+1)$.

Let's implement the Example 4.1.1 on page: 119 of the textbook, which implement the linear interpolation for the data points, $x = [1, 4]$, $y = [1, 2]$. If we simplify the equation (4.2) then we get the following polynomial:

$$P_1(x) = 0.333x + 0.667 \text{ ----- equation(1)}$$

To determine the above polynomial representation uses the following Matlab code:

```
>> x = [1, 4], y = [1, 2]
>> p1 = polyfit(x,y,1)
p1 = 0.3333 0.6667
```

This give the coefficient for linear interpolation polynomial (use $n=1$) as given in equation (1), Suppose we want to evaluate this polynomial as given by equation (1) at $x=2$, i.e. $P_1(2) = 1.3333$. We can use the Matlab command polyval () to evaluate the polynomial at any point x . For the p_1 coefficient calculated above use the following code to evaluate the above polynomial at $x=2$.

```
>> y = polyval(p1,2)
y = 1.3333
```

Consider the Example 4.1.3 on page:121, which determine the quadratic polynomial for the data points $x = [0 1 2]$ and $y = [-1 -1 7]$. The simplified equation (4.9) is given below:

$$P_2(x) = 4x^2 - 4x - 1 \text{ ----- equation(2)}$$

To find the $P_2(x)$ representation (in terms of coefficient) use the following code:

```
>> x = [0 1 2], y = [-1 -1 7]
>> p2 = polyfit(x,y,2)
p2 = 4 -4 -1
```

2. Constructing the Lagrange Interpolating Polynomial using Matlab

1. In order to construct the Lagrange Coefficients for the Lagrange Polynomial in MATLAB, we can use the built-in function poly, which constructs a polynomial with given roots. Enter the following to construct a polynomial with roots 1 and 2 for example:

```
>> poly([1 2])
ans = 1 -3 2
```

Thus, this is the polynomial $(x-1)(x-2) = x^2 - 3x + 2$ which has roots 1 and 2.

2. Consider the Example 4.1.3, page 121 which construct Lagrange interpolating polynomial of degree two (quadratic) approximating the data points $[(0,-1),(1,-1),(2,7)]$

Consider the Lagrange basis function $L_0(x)$ given as

$$L_0(x) = \frac{(x-1)(x-2)}{(0-1)(0-2)}$$

We can see that we need the numerator to be the polynomial with roots $x_1 = 1$, and $x_2 = 2$, i.e. $(x-1)(x-2)$ The denominator is the constant $(x_0-x_1)(x_0-x_2) = (0-1)(0-2) = 2$.

Assuming that we want to store the Lagrange coefficient polynomials in the 3x3 array (matrix) L (with the 1st row being the coefficients for $L_0(x)$, the 2nd row being the coefficients for $L_1(x)$, and the third row being the coefficients for $L_2(x)$), we proceed as follows:

```
>> L(1,:)= poly([1 2])/((0 - 1)*(0 - 2))
L = 0.5000 -1.5000 1.0000
>> L(2,:)= poly([0 2])/((1 - 0)*(1 - 2))
L = 0.5000 -1.5000 1.0000
-1.0000 2.0000 0
```

```
>> L(3,:)= poly([0 1])/((2 - 0)*(2 - 1))
L =
0.5000 -1.5000 1.0000
-1.0000 2.0000 0
0.5000 -0.5000 0
```

The final Lagrange polynomial is: $P_2(x) = y_0 * L_0(x) + y_1 * L_1(x) + y_2 * L_2(x)$. We compute the $P_2(x)$ as:

```
>> P = (-1)*L(1,:) + (-1)*L(2,:) + (7)*L(3,:)
P = 4 -4 -1
```

This has the same coefficient as equation (2) above.

To evaluate the polynomial at 2 we use:

```
>> polyval(P,2)
ans =7
```


Lab (9): Interpolation using Divided Difference and Newton's Formula

1. As a starting example we will construct the divided difference table as given in lecture slides for the following data points $x=[1 \ 1.1 \ 1.2 \ 1.3 \ 1.4]$ and $y=[0.5403 \ 0.45360 \ 0.36236 \ 0.26750 \ 0.16997]$. The divided difference table for these data points is given below:

i	x_i	$y=f(x_i)$	$D^1f(x_i)$	$D^2f(x_i)$	$D^3f(x_i)$	$D^4f(x_i)$
0	1.0	0.54030	-0.8670	-0.2270	0.15333	0.0125
1	1.1	0.45360	-0.9124	-0.1810	0.15830	0
2	1.2	0.36236	-0.9486	-0.1335	0	0
3	1.3	0.26750	-0.9753	0	0	0
4	1.4	0.16997	0	0	0	0

2. In order to construct the Newton polynomial in MATLAB, we would want to first construct the divided difference table. We can do this by storing the values in the rows of a 5 x 5 matrix D. The **first column** of D, referenced in MATLAB as $D(:, 1)$, will store the function values at the interpolating points. The **second column** of D -- $D(:, 2)$ -- will store the **first** divided differences. The **third column** of D -- $D(:, 3)$ -- will store the **second** divided differences. The **fourth column** of D -- $D(:, 4)$ -- will store the **third** divided differences. The **fifth column** of D -- $D(:, 5)$ -- will store the **fourth** divided difference.

The entries in the matrix D will be:

$D(:,1)$	$D(:,2)$	$D(:,3)$	$D(:,4)$	$D(:,5)$
$D(1,1)=0.54030$	$D(1,2)=-0.8670$	$D(1,3)=-0.2270$	$D(1,4)=0.15333$	$D(1,5)=0.0125$
$D(2,1)=0.45360$	$D(2,2)=-0.9124$	$D(2,3)=-0.1810$	$D(2,4)=0.15830$	$D(2,5)=0$
$D(3,1)=0.36236$	$D(3,2)=-0.9486$	$D(3,3)=-0.1335$	$D(3,4)=0$	$D(3,5)=0$
$D(4,1)=0.26750$	$D(4,2)=-0.9753$	$D(4,3)=0$	$D(4,4)=0$	$D(4,5)=0$
$D(5,1)=0.16997$	$D(5,2)=0$	$D(5,3)=0$	$D(5,4)=0$	$D(5,5)=0$

3. Create a 5x5 matrix D initially with all zeros:

```
>> D = zeros(5,5);
```

4. Set up the vector X and Y with the x-coordinates of the interpolating values:

```
>> X=[1 1.1 1.2 1.3 1.4];
```

```
>> Y=[0.5403 0.45360 0.36236 0.26750 0.16997];
```

These entries will be stored as:

For X as:

$X(1)=1$	$X(2)=1.1$	$X(3)=1.2$	$X(4)=1.3$	$X(5)=1.4$
----------	------------	------------	------------	------------

If you run this on Matlab command window

```
>>X(3)
```

```
ans=1.2
```

```
>>X(1:3)
```

```
ans = 1 1.1 1.2
```

And for Y as:

Y(1)=0.5403	Y(2)=0.45360	Y(3)=0.36236	Y(4)=0.26750	Y(5)=0.16997
-------------	--------------	--------------	--------------	--------------

5. Now start computing the divide differences column by column for the matrix D The first column is just the values of the function at the interpolating points, stored in Y:

» D(:,1) = Y;

6. We next work on the **second column of D** -- starting in first row (D(1,2)) and working down to fourth row: >> D(1,2) = (D(2,1)-D(1,1))/(X(2)-X(1));

>> D(2,2) = (D(3,1)-D(2,1))/(X(3)-X(2));

>> D(3,2) = (D(4,1)-D(3,1))/(X(4)-X(3));

>> D(4,2) = (D(5,1)-D(4,1))/(X(5)-X(4));

7. Fill the remaining column by using the following commands:

>> D(1,3) = (D(2,2)-D(1,2))/(X(3)-X(1));

>> D(2,3) = (D(3,2)-D(2,2))/(X(4)-X(2));

>> D(3,3) = (D(4,2)-D(3,2))/(X(5)-X(3));

>> D(1,4) = (D(2,3)-D(1,3))/(X(4)-X(1));

>> D(2,4) = (D(3,3)-D(2,3))/(X(5)-X(2));

>> D(1,5) = (D(2,4)-D(1,4))/(X(5)-X(1));

The final matrix D will have the following form:

>>D

D =

0.5403 -0.8670 -0.2270 0.1533 0.0125

0.4536 -0.9124 -0.1810 0.1583 0

0.3624 -0.9486 -0.1335 0 0

0.2675 -0.9753 0 0 0

0.1700 0 0 0 0

8. We can now construct the Newton Polynomials of degrees 1 through 4 recursively as follows:

>> P1 = [0 D(1,1)] + D(1,2)*poly(X(1))

P1 =

-0.8670 1.4073

And also you can go to higher polynomials like this.

Lab (10): Numerical Integration: Implementing Trapezoidal Rule in Matlab

- The trapezoidal rule as discussed in class is given as

$$I(f) = \int_a^b f(x)dx \approx T_n(f) = \frac{h}{2}[f(x_0) + f(x_n)] + h[f(x_1) + f(x_2) + \dots + f(x_{n-1})]$$

Here $a = x_0$ and $b = x_n$ and $h = \frac{b-a}{n}$, $x_j = a + jh$ $j = 0, 1, 2, \dots, n$

- The following file trapezoidal.m implements the trapezoidal rule in Matlab

```
function [integral] = trapezoidal(index_f,a,b,n)
% function to calculate the integral using trapezoidal rule
% input parameters:
% index_f: parameter for the integrand
% a: lower limit
% b: upper limit
% n: Number of intervals must be positive integer greater than or equal
% to 1
%
% output parameter: integral
% sum the endpoints
sumend =( f(a,index_f) + f(b,index_f))/2; % f is a function which contains
% multiple integrands
h = (b-a)/n; % size of interval
sum = 0;
if ( n > 1)
for j = 1:1:n-1
xj = a + j*h;
sum = sum + f(xj,index_f);
end
end
integral = h*(sumend + sum);
function f_value = f(x,index)
% this function defines the integrand
switch index
case 1
f_value = exp(-x.^2);
case 2
f_value = 1./(1+x.^2);
end
```

- Read the **m** file carefully and make sure you understand all the Matlab commands
- Run this function by writing a script file (e.g. main.m). A sample script file is as follows:

```
clear all
a = 0;
b = 1;
n = 1;
index_f = 1
integral = trapezoidal(index_f,a,b,n)
```

- Run this script file to calculate the integral of $I = \int_0^1 e^{-x^2} dx \approx 0.746824132812427$, as given on page 193 of your textbook and fill in the following table

Number of interval (n)	Step size (h)	Tn(f)	Error
1			
2			
4			
8			
16			
32			
64			
128			

- Modify your script file to generate the table 5.1, page 194 of your textbook. A sample script file is as follows.

```
clear all
a = 0;
b = 1;
n = 1;
index_f = 1
n1 = [ 2 4 8 16 32 64 128];
for q = 1:length(n1)
n = n1(q);
integral(q) = trapezoidal(index_f,a,b,n);
end
trueval = 0.746824132812427;
err = trueval - integral
y = [n1; err];
fprintf(1, '%6.2f %12.8f\n', y);
```

- Plot the error versus n, i.e., `plot(n1,err,'-o')`
- Repeat the above exercises for the integrals given on page 193 of your textbok. Note you will have to change the `trapezoidal.m` file to add the new integrand.

Lab (11): Numerical Integration#2: Implementing Simpson Rule in Matlab

- The Simpson rule as discussed in class is given as

$$I(f) = \int_a^b f(x)dx \approx S_n(f)$$

$$= \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

Here $a = x_0$ and $b = x_n$ and $h = \frac{b-a}{n}$, $x_j = a + jh$ $j = 0, 1, 2, \dots, n$

- The following file `simpson.m` implements the Simpson's rule in Matlab

```
function [integral]=simpson(index_f,a,b,n)
%
% function [integral]=simpson(a,b,n0,index_f)
%
% function to calculate the integral using Simpson rule
% input parameters:
% index_f: parameter for the integrand
% a: lower limit
% b: upper limit
% n: Number of intervals must be positive even integer greater than or %
%equal to 2
%
% output parameter:integral
% Initialize for Simpson integration.
sumend = f(a,index_f) + f(b,index_f);
sumodd = 0;
sumeven = 0;
% For case of n=2
if(n == 2)
h = (b-a)/n;
k=1;
sumodd1 = f(a+k*h,index_f);
integral= h*(sumend + 4*sumodd1)/3; %%% this will calculate the integral
end
%% For case of n > 2.
if(n > 2)
h = (b-a)/n;
for i=2:2:n-2
sumeven = sumeven + f(a+i*h,index_f);
end
for k=1:2:n-1
sumodd = sumodd + f(a+k*h,index_f);
end
integral= h*(sumend + 4*sumodd + 2*sumeven)/3; %%% this will caluclate the
integral
end
%
% This defines the integrand.
function f_value = f(x,index)
```

```
switch index
case 1
f_value = exp(-x.^2);
case 2
f_value = 1 ./ (1+x.^2);
case 3
f_value = 1 ./ (2+sin(x));
case 4
f_value = exp(cos(x));
end
```

- Read the **m** file carefully and make sure you understand all the Matlab commands
- Run this function by writing a script file (e.g. main.m). A sample script file is as follows:

```
clear all
a = 0;
b = 1;
n = 2;
index_f = 1
integral = simpson(index_f,a,b,n)
```

- Run this script file to calculate the integral of $I = \int_0^1 e^{-x^2} dx \approx 0.746824132812427$, as given on page 198 of your textbook and fill in the following table

Number of interval (n)	Step size (h)	Sn(f)	Error
2			
4			
8			
16			
32			
64			
128			

- Modify your script file to generate the table 5.2, page 198 of your textbook. A sample script file is as follows.

```
clear all
a = 0;
b = 1;
n = 1;
index_f = 1
n1 = [ 2 4 8 16 32 64 128];
for q = 1:length(n1)
n = n1(q);
integral(q) = simpson(index_f,a,b,n);
end
trueval = 0.746824132812427;
err = trueval - integral
y = [n1; err];
fprintf(1, '%6.2f %12.8f\n', y);
```

- Plot the error versus n , i.e., `plot(n1,err,'-o')`
- Repeat the above exercises for the integrals given on page 198 of your textbook. Note you will have to change the `simpson.m` file to add the new integrand.