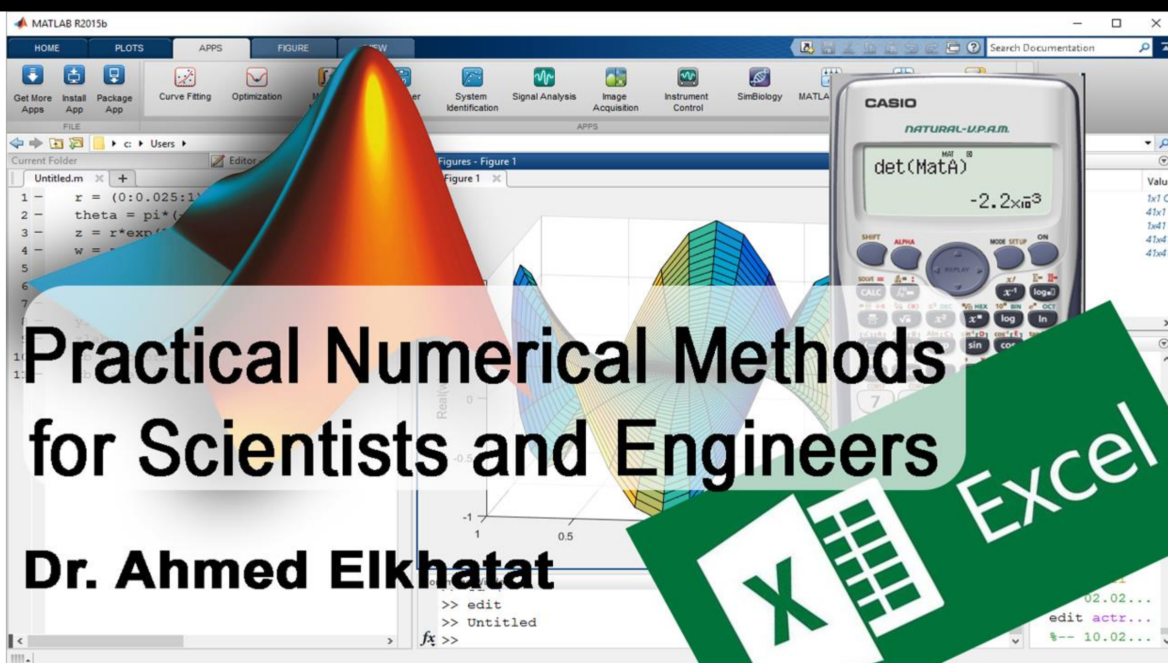
2022

Practical Numerical Methods for Scientists and Engineers



Dr. Ahmed Elkhatat

Qatar University

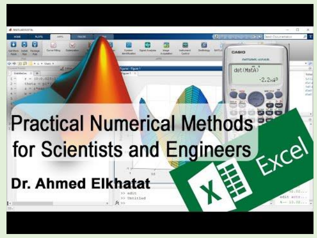
Table of Contents

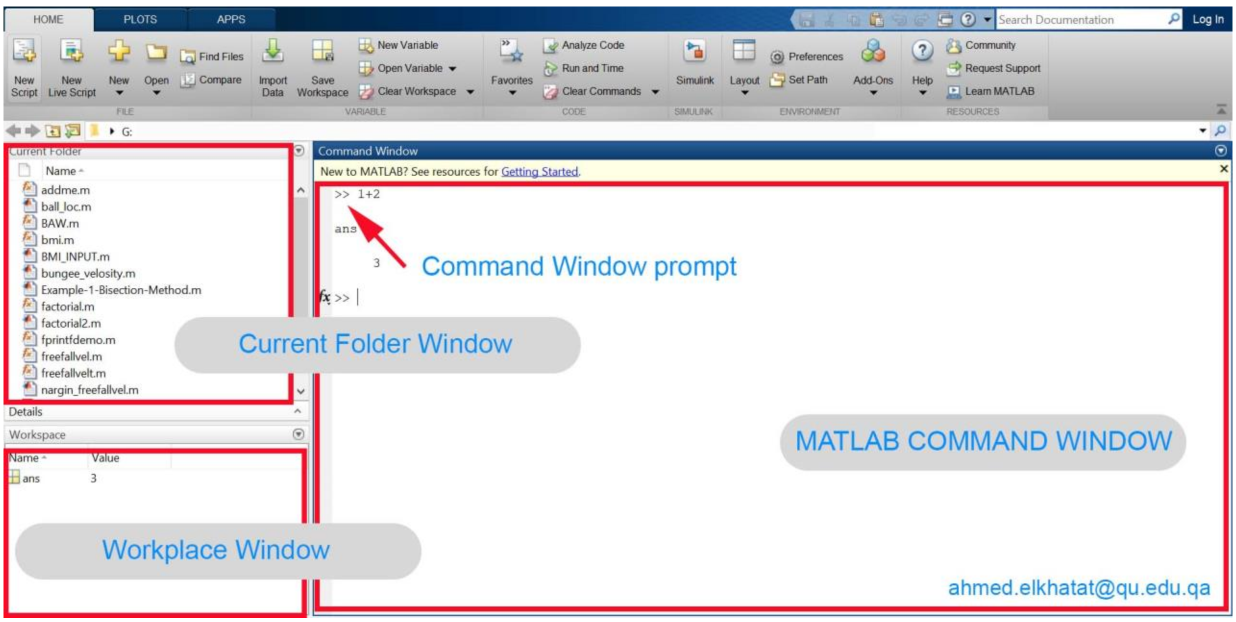
Module (1): Introduction to MATLAB

MATLAB: [Matrix] [Laboratory]

1. MATLAB User Interface

Watch the online Video





A) Interface Windows

i. Command Window

The Command Window is one of the main tools you use to enter data, run MATLAB functions and other M-files, and display results.

The Command Window prompt, >>, is where you enter statements. You can enter a MATLAB function with arguments or assign values to variables.

For example, 1+2,

ans = 3.

ii. Workspace Window

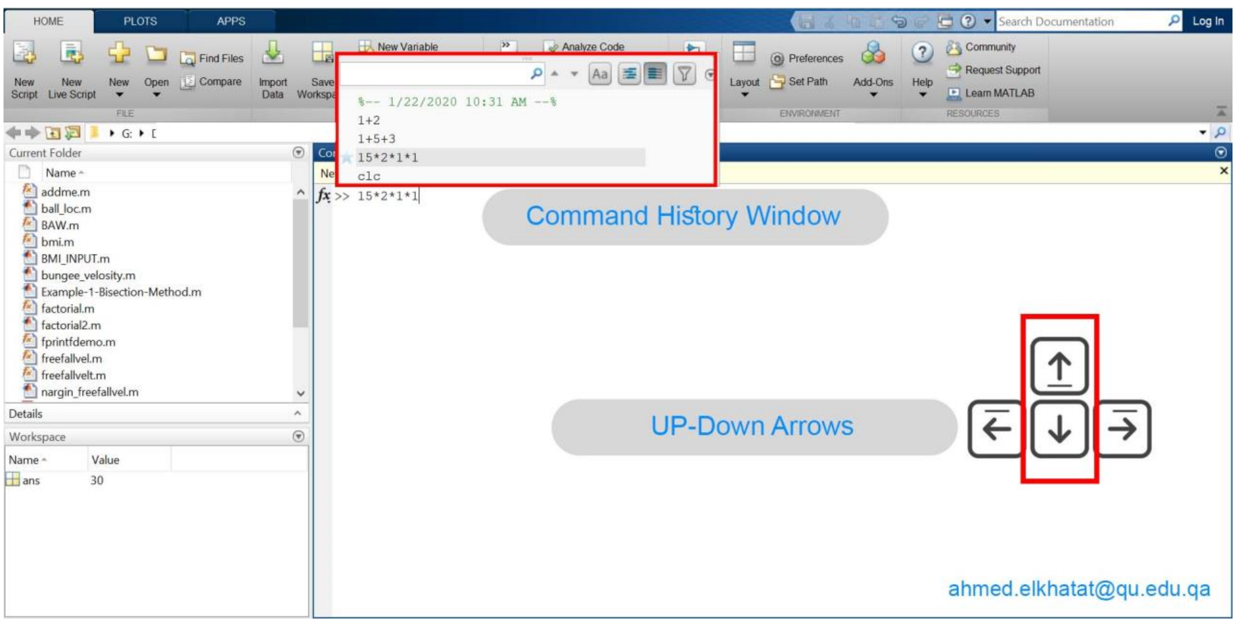
Contains any variables generated as the result of working in the Command window. In this case, the Workspace window contains a variable named **ans** that holds a value of 3.

iii. Current folder Window

Contain saved MATLAB files.

iv. Command History window

Using Up-Down Arrows at your keyboard, you can show this widow that displays the series of formulas or commands that you type, along with the date and time you typed them. You can replay a formula or command in this window. Just select the formula or command that you want to use from the list to replay it.



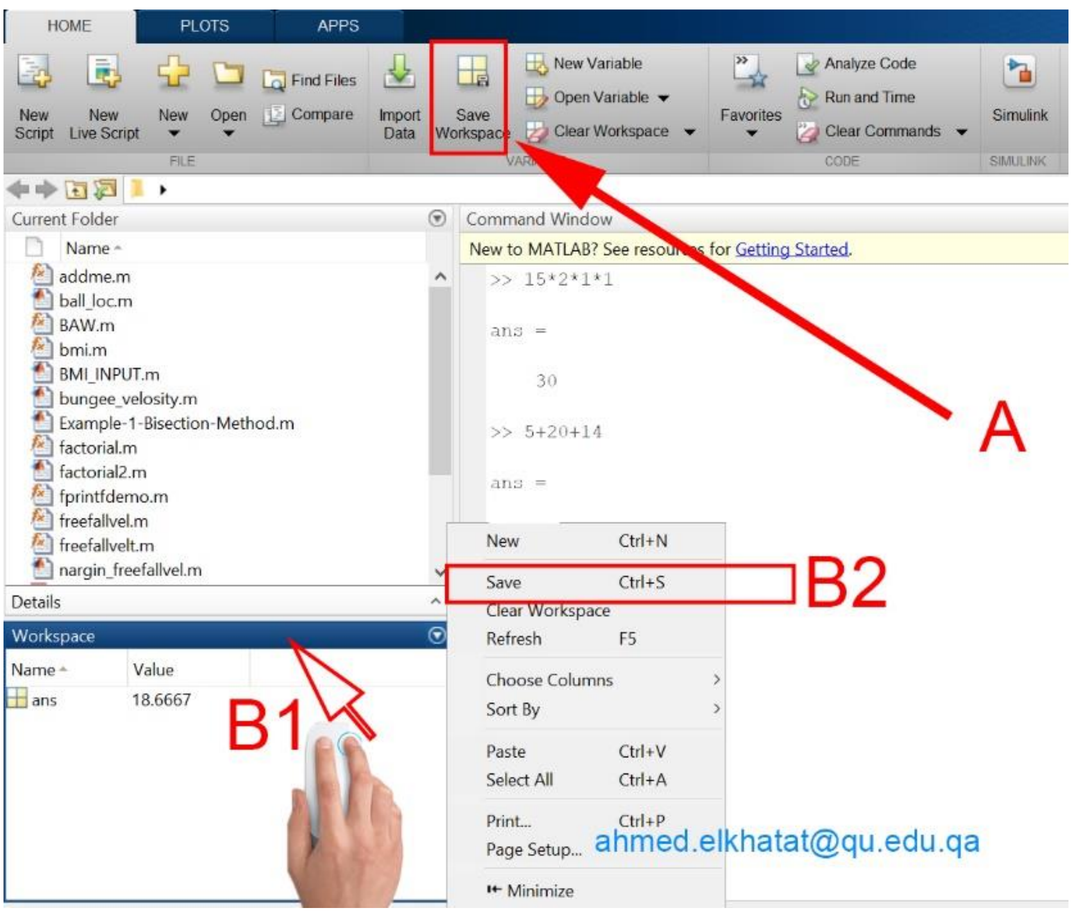
B) Saving the workspace results in MATLAB

The workspace is not maintained across sessions of MATLAB®. When you quit MATLAB, the workspace clears. However, you can save any or all the variables in the current workspace to a MAT-file (.mat).

You can then reuse the workspace variables later during the current MATLAB session or another session by loading the saved MAT-file.

There are several ways to save workspace variables interactively:

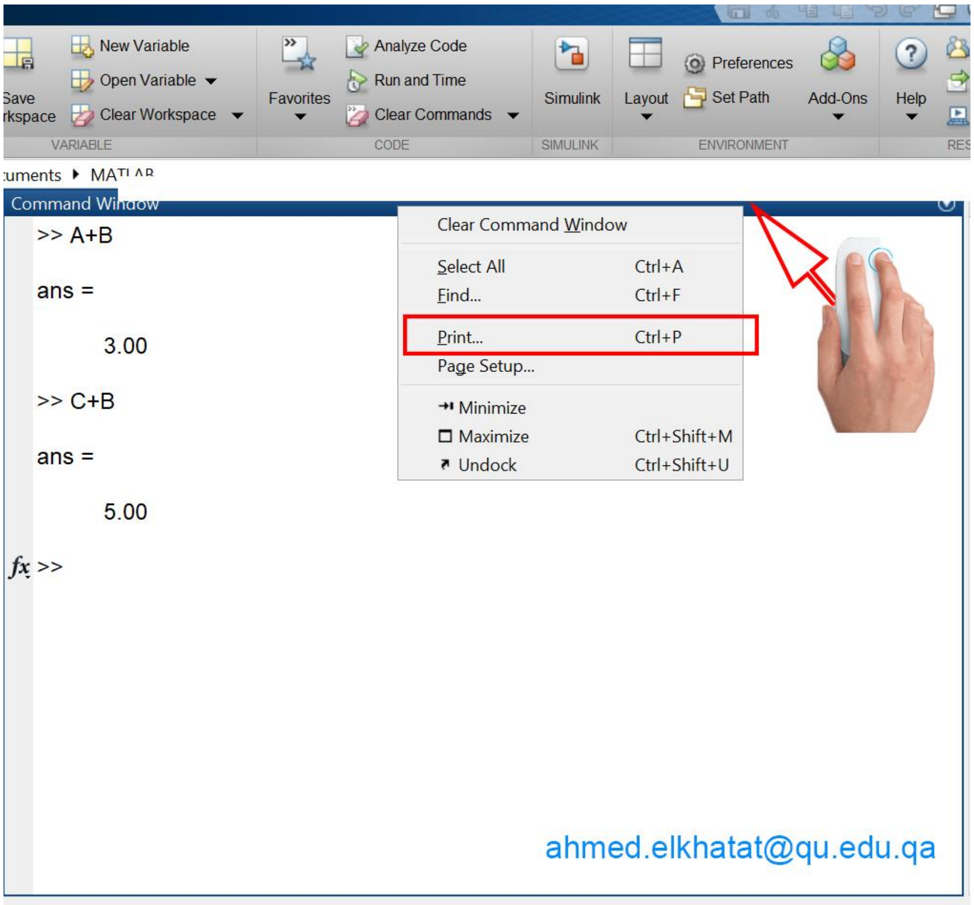
1) **To save all workspace** variables to a MAT-file, on the Home tab, in the Variable section, click Save Workspace.
2) **To save a subset of your workspace variables** to a MAT-file, select the variables in the Workspace browser, right-click, and then select Save As. You also can drag the selected variables from the Workspace browser to the Current Folder browser.
3) **To save variables** to a MATLAB script, click the Save Workspace button or select the Save As option, and in the Save As window, set the Save as type option to MATLAB Script. Variables that cannot be saved to a script are saved to a MAT-file with the same name as that of the script.



C) Saving the Command window in MATLAB

To save the contents of your Command Window to PDF. **Right-click** on the Command Window bar, and then **Print**.

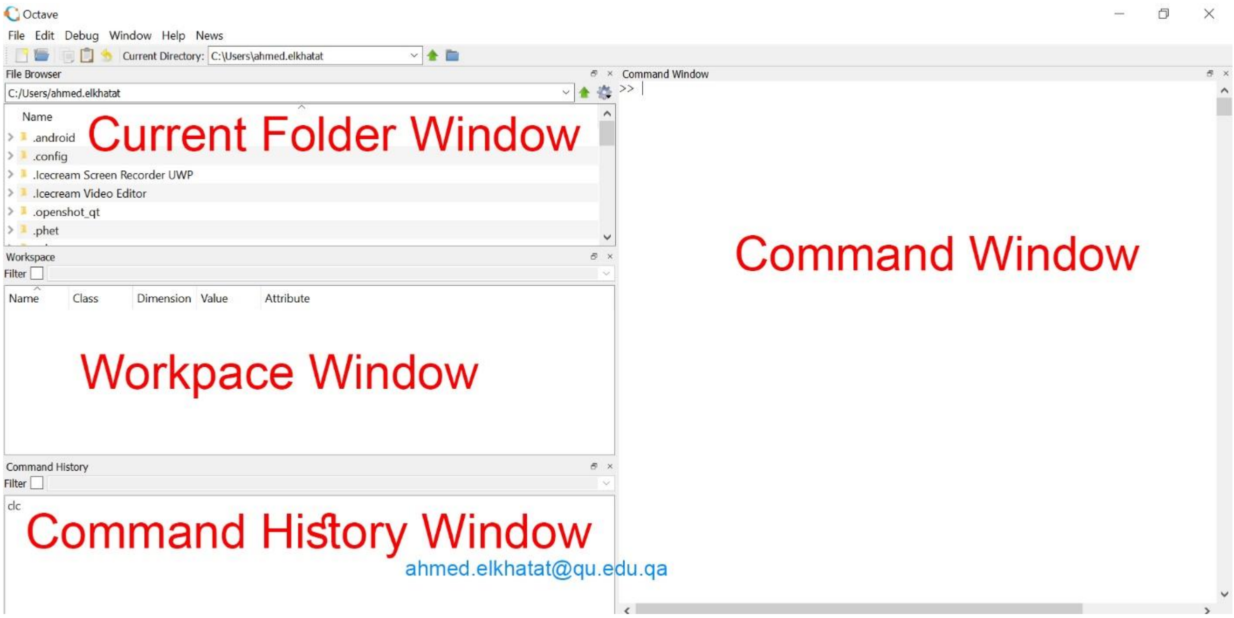
Alternatively, **copy and paste** the commands into an editable file.



D) OCTAVE User Interface

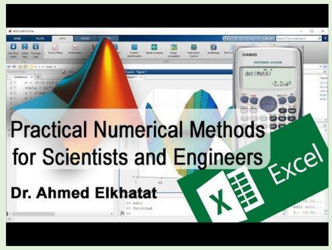
If you don't have access to MATLAB for practicing, you can download OCTAVE. It is a powerful open-source (free) and has almost the same platform as MATLAB.

To download it, please follow this link https://www.gnu.org/software/octave/

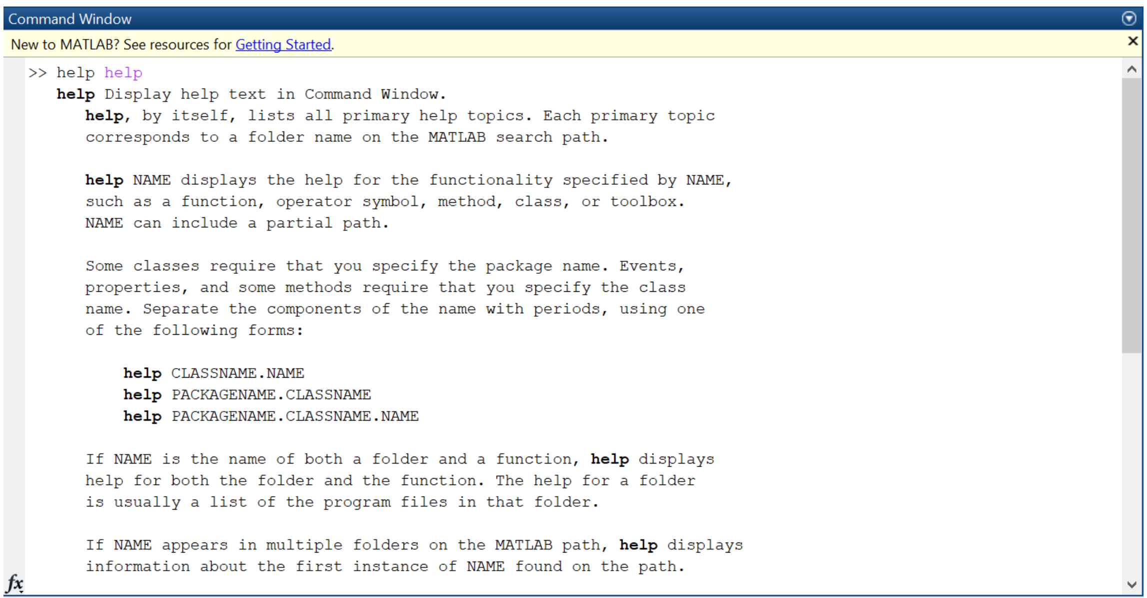


2. Some Useful Commands in MATLAB

Watch the online Video



<u>help:</u> Display help text in Command Window.



```
Command Window
New to MATLAB? See resources for Getting Started.
>> help help
 help Display help text in Command Window.
    help, by itself, lists all primary help topics. Each primary topic
    corresponds to a folder name on the MATLAB search path.

    help NAME displays the help for the functionality specified by NAME,
    such as a function, operator symbol, method, class, or toolbox.
    NAME can include a partial path.

    Some classes require that you specify the package name. Events,
    properties, and some methods require that you specify the class
    name. Separate the components of the name with periods, using one
    of the following forms:

        help CLASSNAME.NAME
        help PACKAGENAME.CLASSNAME
        help PACKAGENAME.CLASSNAME.NAME

    If NAME is the name of both a folder and a function, help displays
    help for both the folder and the function. The help for a folder
    is usually a list of the program files in that folder.

    If NAME appears in multiple folders on the MATLAB path, help displays
    information about the first instance of NAME found on the path.
fx
```

<u>doc</u>: Opens the Help browser if it is not already running and otherwise brings the Help browser to the top.

<u>clear</u>: Removes all variables from the workspace.

<u>clear All</u>: Removes all variables, globals, functions and MEX links.

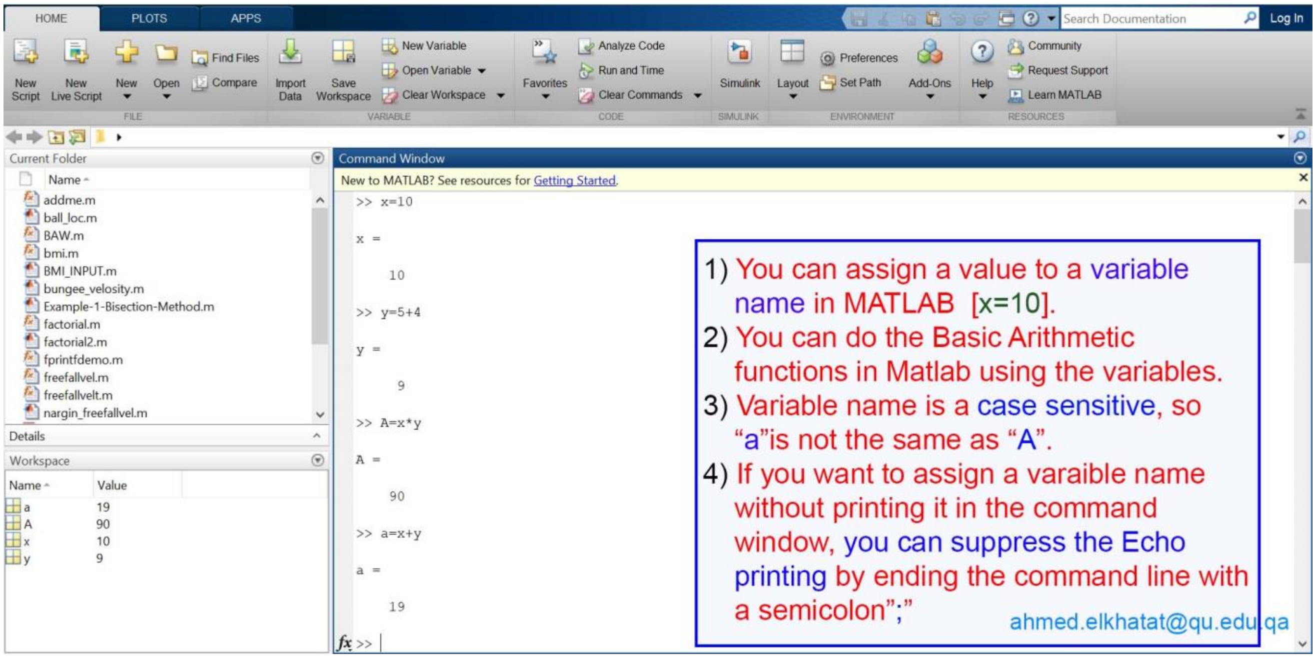
<u>home</u>: Moves the cursor to the upper left corner of the window. It also scrolls the visible text in the window up out of view; you can use the scroll bar to see what was previously on the screen.

<u>clc</u>: Clear command window.

<u>who</u>: Lists the variables in the current workspace..

<u>whos</u>: is a long-form of WHO. It lists all the variables in the current workspace, together with their size, bytes, class, etc..

A) Basic Arithmetic functions in MATLAB



B) List of Commonly Used Operators

Operator	Purpose
+	Plus; addition operator.
-	Minus; subtraction operator.
*	Scalar and matrix multiplication operator.
.*	Array multiplication operator.
^	Scalar and matrix exponentiation operator.
.^	Array exponentiation operator.
\	Left-division operator.
/	Right-division operator.

Operator	Purpose
.\	Array left-division operator.
./	Array right-division operator.
:	Colon; generates regularly spaced elements and represents an entire row or column.
( )	Parentheses; encloses function arguments and array indices; overrides precedence.
[ ]	Brackets; enclosures array elements.
.	Decimal point.
...	Ellipsis; line-continuation operator
,	Comma; separates statements and elements in a row
;	Semicolon; separates columns and suppresses display.
%	Percent sign; designates a comment and specifies formatting.
_	Quote sign and transpose operator.
._	Nonconjugated transpose operator.
=	Assignment operator.

C) Practice

```
>> x=10;

>> y=2;

>> z1=x+y

        z1 =12

>> z2=x-y

        z2 =8

>> z3=x*y

        z3 =20

>> z4=x/y

        z4 =5

>> z5=x\y

        z5 =0.2000

>> z6=y-x

        z6 =-8

>> A1=abs(z6)

        A1 = 8
```

```
>> x=4.454
        x =4.4540
>> floor(x)
        ans =4
>> ceil(x)
        ans =5
>> round(x)
        ans =4
>> round(x,1)
        ans = 4.5000
>> round(x,2)
        ans =4.4500
>> y=315.214
        y = 315.2140
>> round(y,-1)
        ans = 320
>> round(y,-2)
        ans = 300
>>sqrt (x)
        ans = 2.1105
```

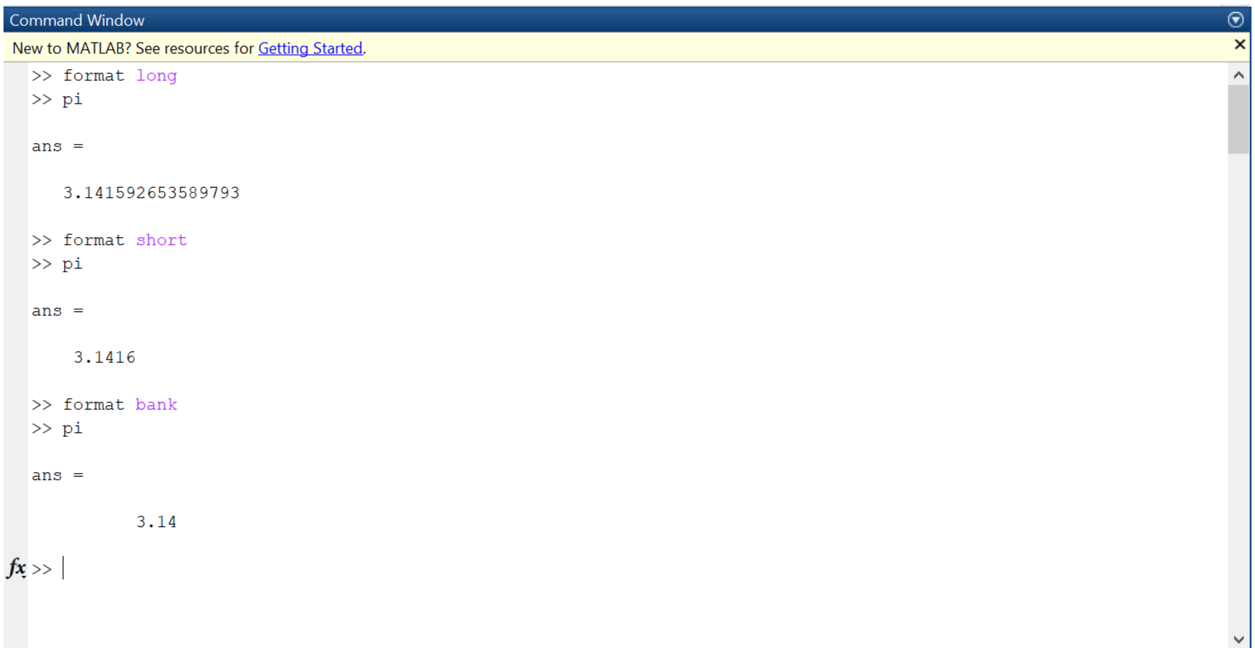
D) Formatting in MATLAB

Format command can be used to set the output format to the default appropriate for the class of the variable.

format long: Scaled fixed point format with 15 digits.

format short: Scaled fixed point format with 5 digits.

format bank: Fixed format for dollars and cents.



E) Symbolic and Numeric in MATLAB

In MATLAB, you can construct symbolic numbers using the command (**sym**), and you can return them to scalar using the command (**double**).

```
Command Window
New to MATLAB? See resources for Getting Started.
>> a=12/15

a =

   0.800000000000000

>> a=sym(12/15)

a =

4/5

>> double(a)

ans =

   0.800000000000000

fx >>
```

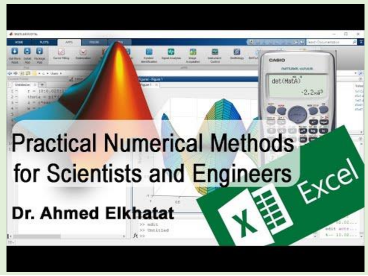
You can also construct symbolic variables and objects using command **syms x**. For more than one variable, leave a space between them.

You can also use the function pretty to present the answer in a pretty way.

```
Command Window
New to MATLAB? See resources for Getting Started.
>> syms A B C D
>> C=2*A^3+5*B^2+4

C =

2*A^3 + 5*B^2 + 4

>> D=3*A^2+4*B^3+3

D =

3*A^2 + 4*B^3 + 3

>> C/D

ans =

(2*A^3 + 5*B^2 + 4)/(3*A^2 + 4*B^3 + 3)

>> pretty (ans)
   3     2
2 A  + 5 B  + 4
---------------
   2     3
3 A  + 4 B  + 3

fx >>
```

3. Vectors and Matrices in MATLAB

Watch the online Video



A) Creating Vectors in MATLAB

Vectors can be created **horizontally** using the following commands and functions

```
>> A=[1 2 3 4 5]          or     A=[1,2,3,4,5]

     A =1    2    3    4    5


>> A=[1:5]

     A =1    2    3    4    5


>> D=[1:2:5]

     D =1    3    5


>> A=linspace(1,5,5)

     A =1    2    3    4    5


>> linspace(1,10,3)

     ans =1.000000000000000   5.500000000000000  10.000000000000000


>> A=rand(1,8)             %to create 1 raw and 8 columns (0-1) randomly

     A =0.4173    0.0497    0.9027    0.9448    0.4909    0.4893    0.3377    0.9001


>> A=500*rand(1,7)
```

A =390.1260 194.8694 120.8456 201.9561   48.2273   65.9866 471.0253

>> A=round(500*rand(1,10))

A =117   177   411   8   22   84   325   366   324   225

Note: if you want to create random numbers between two values, the following formula can be used: A=round(low +(up-low)*rand(m,n))

[Example: 5 integers from 100-500]

>> A=round(100+(500-100)*rand(1,5))

A =117   477   411   325   266

Vectors can be created **vertically** using the following commands and functions

```
>> a=[1;2;3;4;5]
a =
     1
     2
     3
     4
     5
```

```
>> a= [1:2:10]'
a =
     1
     3
     5
     7
     9
```

MATLAB can reshape an array into a specific size. So, it can be used to make arrays only verticals or horizontal using

reshape(X,M,N)

- Where X is the array, M is the number of rows wanted, and N is the number of columns needed

Example

- Write a MATLAB syntax that convert (X) horizontal array to a vertical one, and do nothing for vertical arrays.
- Here, the column=1, and rows should be any numbers according to the array size, then the syntax should be: **reshape(X,[],1)**

```
>> X=[1:5]
    X =  1.00     2.00     3.00     4.00     5.00
>> Y=reshape(X,[],1)
Y =
1.00
2.00
3.00
4.00
5.00
```

Another functional syntax is Y=x(:)

Example

- Write a MATLAB syntax that converts (X) vertical array to a horizontal one and does nothing for horizontal arrays.
- Here, the rows=1 and columns should be any numbers according to the array size. Then the syntax should be: **reshape(X,1,[])**

```
>> X=[1:5]'

X =

1.00

2.00

3.00

4.00

5.00

>> Y=reshape(X,1,[])

    Y =  1.00        2.00        3.00        4.00        5.00
```

B) Creating Matrices in MATLAB

Matrices can be created using the following commands and functions

```
>> a=[1 2 3 4 5]; b=[2 3 1 4 5]; c=[1 0 3 2 4];

>> d=[a;b;c]

d =

    1    2    3    4    5

    2    3    1    4    5

    1    0    3    2    4


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

A =

    1    2    3

    4    5    6

    7    8    9
```

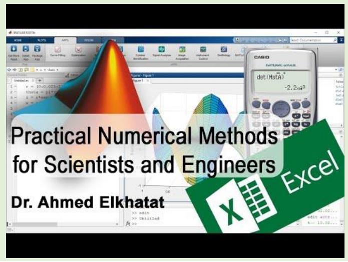
Special matrices can be also created using the following commands and functions

```
>> ones(3)

ans =

    1    1    1

    1    1    1

    1    1    1


>> ones(3,2)

ans =

    1    1

    1    1

    1    1
```

```
>> zeros(2)
        ans =
    0     0
    0     0

>> eye(3)
ans =
    1     0     0
    0     1     0
    0     0     1
```

C) Extracting, Replacing or Eliminating an element or more from Arrays and Matrices

Watch the online Video



In order to extract an element or array from vectors or matrices; the following commands and functions can be executed **A(Row number, Column Number)**

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

A =

    1    2    3

    4    5    6

    7    8    9


>> A(2,3)

ans =6


>> A(2)

ans =4


>> A(2,:)

ans = 4    5    6


>> A(:,1)

ans =

    1
```

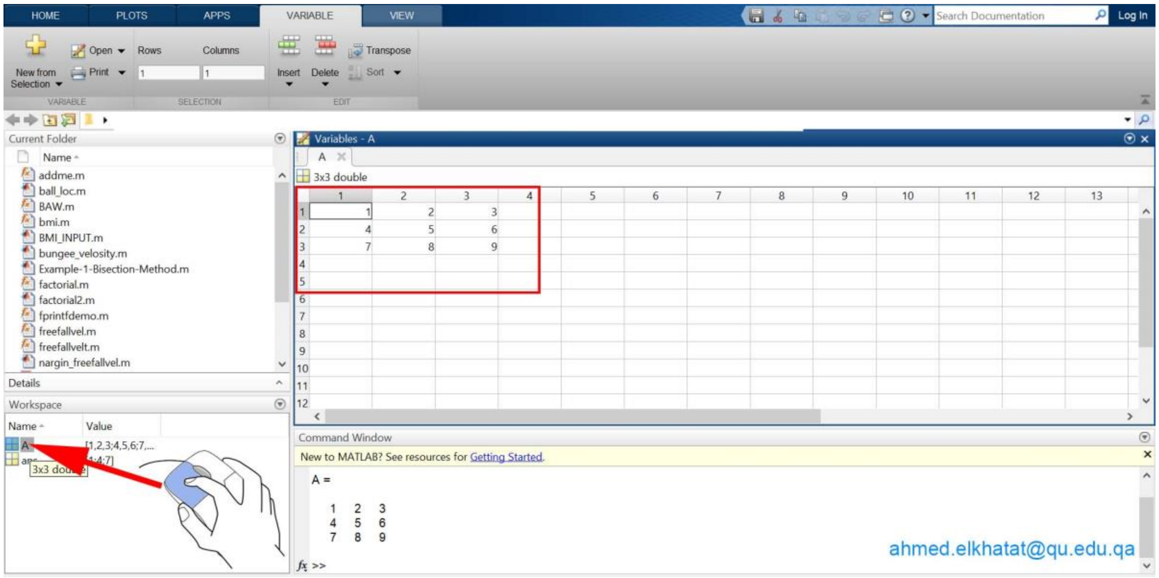
```
    4
    7
>> A(3,1:2)
ans = 7.00        8.00
```

In order to extract a small matrix from a larger one, for example to extract [2 3; 5 6]:

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

>> A(1:2,2:3)
ans =
        2.00        3.00
        5.00        6.00
```

D) Changing a vector in MATLAB

If you want to change a value of a vector in an array or a matrix, you can double click in the workplace; this will open a window contains all variables, so you can change the value you want.



In order to replace an element or array in a vectors or matrix

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

>> A(1:2,2:3)=0
A =
     1     0     0
     4     0     0
     7     8     9
```

In order to remove elements from an array, [] can be used as follows

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

A = 1.00    2.00    3.00    4.00    5.00

>> A(2)=[]

A = 1.00    3.00    4.00    5.00
```

E) Some useful commands in Matrices and Vectors

Find: To find the location of a vector

```
A =

     1     2     3
     4     2     5
     2     8     1

>> find(A==2)
ans =
     3
     4
     5

>> find(A==5)
ans =8
```

Sort: To sort the vectors of each column.

```
A =

     1     2     3
     4     2     5
     2     8     1

>> sort(A)              %ascending by default
ans =
     1     2     1
     2     2     3
     4     8     5
```

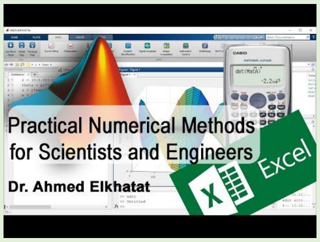
```
>> sort(A,'ascend')
ans =

     1     2     1
     2     2     3
     4     8     5

>> sort(A, 'descend')
ans =
     4     8     5
     2     2     3
     1     2     1
```

numel, length, size: To calculate the number of vectors, length of vectors, and size of a matrix.

```
A =
     1     2     3
     4     2     5
     2     8     1

>> numel (A)
ans =9

>> length (A)
ans =3
>> size(A)
ans = 3     3
```

4. Mathematical Operations in Arrays/Matrices in MATLAB

Watch the online Video



The absolute value of the elements

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

>> abs(A)
ans =
     1     2     3
     4     5     6
     7     8     9
```

Trigonometric functions of the matrix

```
>> cos(A)
ans =
        0.54       -0.42       -0.99
       -0.65        0.28        0.96
        0.75       -0.15       -0.91
```

```
>> sin(A)
ans =

    0.84    0.91    0.14
    0.76   -0.96   -0.28
    0.66    0.99   -0.41

>> log(A)
ans =
    0.00    0.69    1.10
    1.39    1.61    1.79
    1.95    2.08    2.20
```

The root of the matrix

```
A =
    4.00    9.00   16.00
    8.00   27.00   64.00

>> sqrt(A)
ans =
    2.00    3.00    4.00
    2.83    5.20    8.00

>> nthroot(A,2)
ans =
    2.00    3.00    4.00
    2.83    5.20    8.00
```

>> nthroot(A,3)

ans =

1.59	2.08	2.52
2.00	3.00	4.00

Multiplication of arrays and matrix

A =

4.00	9.00	16.00
8.00	27.00	64.00

>> 2*A

ans =

8.00	18.00	32.00
16.00	54.00	128.00

>> A/2

ans =

2.00	4.50	8.00
4.00	13.50	32.00

Note the following command

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

>> A*B

Error using *

Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in the second matrix. To perform elementwise multiplication, use '.*'.

The number of columns in the first matrix should match the number of rows in the second matrix.

Write the following command

```
>> A=[1 2 3]; B=[4; 5; 6];
>> A*B
ans =32.00
```

Why??, Because it is (1*4+2*5+3*6)=32

So, in order to perform it, you run the following commands

```
>> A*B'        %Transpose
ans =32.00
```

or

```
>> dot(A,B)
ans =32.00
```

If you want to perform a Cross Product, run the following commands

```
>> cross(A,B)
ans = -3.00       6.00       -3.00
```

If you want one by one multiplication then.

Write the following commands

```
>> A=[1 2 3]; B=[4; 5; 6];
>> A.*B
ans =
4.00       8.00       12.00
```

5.00	10.00	15.00
6.00	12.00	18.00

Write the following commands

>> A=[1 2 3]

A =

1.00 2.00 3.00

>> B= [2 4 6]

B =

2.00 4.00 6.00

>> A.*B

ans =

2.00 8.00 18.00

>> A.^B

ans =

1.00 16.00 729.00

LU: To find the Lower triangular and upper triangular matrices for a matrix.

```
Command Window
>> A=[15 22 67; 34 51 93; 22 51 78]

A =

      15.00       22.00       67.00
      34.00       51.00       93.00
      22.00       51.00       78.00

>> [L, U]=lu(A)

L =

       0.44       -0.03        1.00
       1.00           0           0
       0.65        1.00           0


U =

      34.00       51.00       93.00
          0       18.00       17.82
          0           0       26.47

fx >>   ahmed.elkhatat@qu.edu.qa
```

A) Some Useful Statistical Commands

A =

1 2 3

4 2 5

2 8 1

To find the min for each column.

>> min(A)

ans =1 2 1

To find the minimum value in a matrix

>> min (min(A))

>> min(A(:))

>> min(A, [], 'all') % Starting in R2018b

ans =1

To find the max for each column.

>> max(A)

ans = 4 8 5

To find the maximum value in a matrix

>> max (max(A))

>> max(A(:))

>> max(A, [], 'all') % Starting in R2018b

To find the mean in a matrix

>> mean(A)

>> mean(A(:))

To find the median in a matrix

```
>> median(A)
>> median(A(:))
```

To find the standard deviation in a matrix

```
>> std(A)
>> std(A(:))
```

To find the variance in a matrix

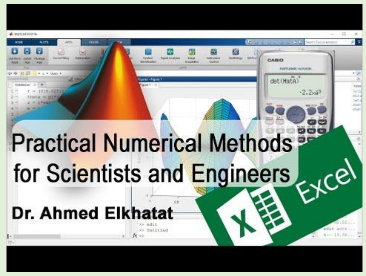
```
>> var(A)
>> var(A(:))
```

To find the correlation coefficient in a matrix

```
>> corrcoef (A)
>> corrcoef (A(:))
```

5. Input/output functions

Watch the online Video



Input function prompts the user for values directly from the command window. and its syntax is

n=input('promtstring')

Similarly, the output can be displayed as a value or a string using ***disp, fprintf, msgbox, error***

A) disp

disp(variable)

disp('string')

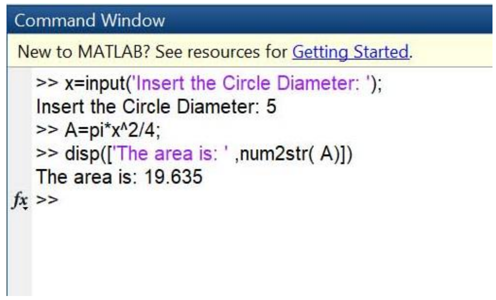
if you want to combine both string and variable or value in the same line, you can write the syntax as

disp(['string' , num2str(variable)])

Practice

Can you calculate the area of a circle, where the diameter is an input variable? Properly display the results.

```
>> x=input('Insert the Circle Diameter: ');
Insert the Circle Diameter: 5
>> A=pi*x^2/4;
>> disp(['The area is: ' ,num2str( A)])
The area is: 19.635
```



B) fprinf

Another valuable way to display several strings and values is by using the function **fprinf**

```
>> fprintf('The Area of the Circle is %f cm \n', A)

The Area of the Circle is 19.634954 cm

>> fprintf('The Area of the Circle is %0.2f cm \n', A)

The Area of the Circle is 19.63 cm

>> fprintf('The Area of the Circle is %0.0f cm \n', A)

The Area of the Circle is 20 cm
```

Notes:

%s	→	print a string
%c	→	print a single character
%d	→	print a whole number
%f	→	print a floating point number
%0.2f	→	print a number with two decimal
%0.1f	→	print a number with one decimal
%0.0f	→	print a number with no decimal (as Integer)
\n	→	print a new line (go to the next line to continue printing)

C) msgbox

msgbox is a display tool to display a message in a box
msgbox(‘Good Morning’)

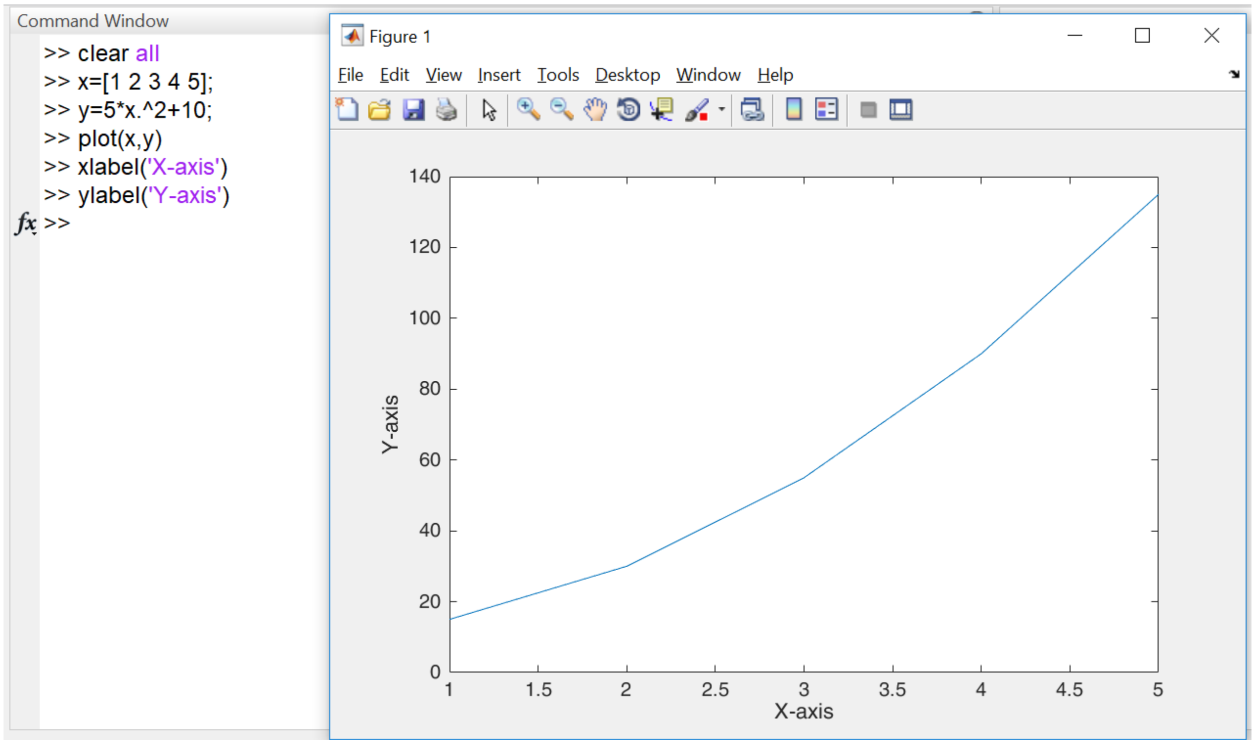
D) error

error can be used also to display a massage with a notification sound
error('Good morning')

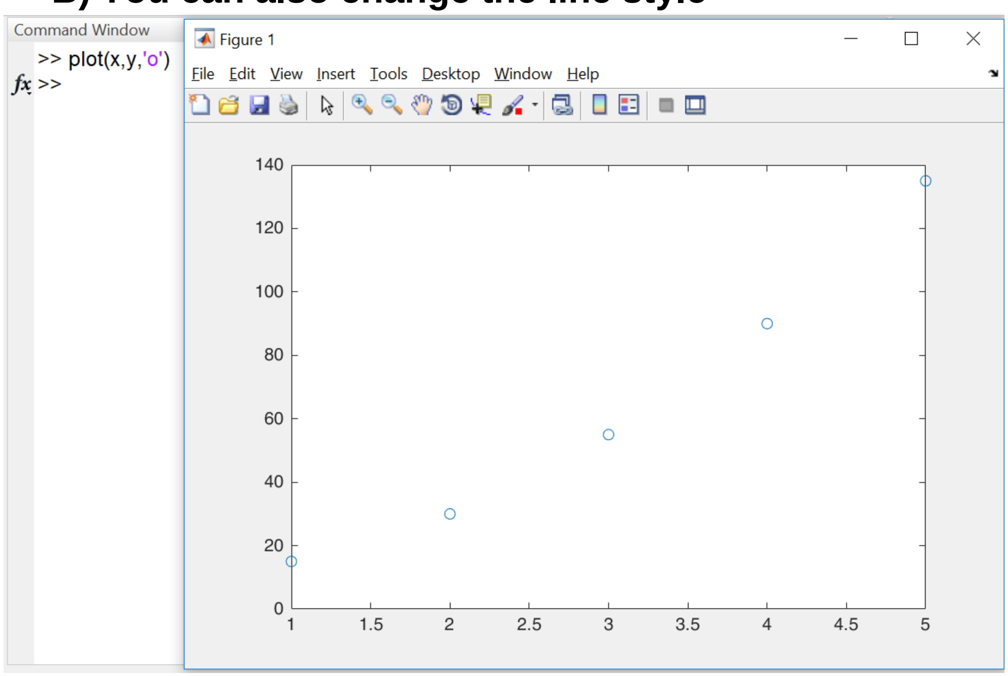
6. Plotting

A) Plot, xlabel, ylabel, title

You can plot X vs Y and put titles for the axis using these function



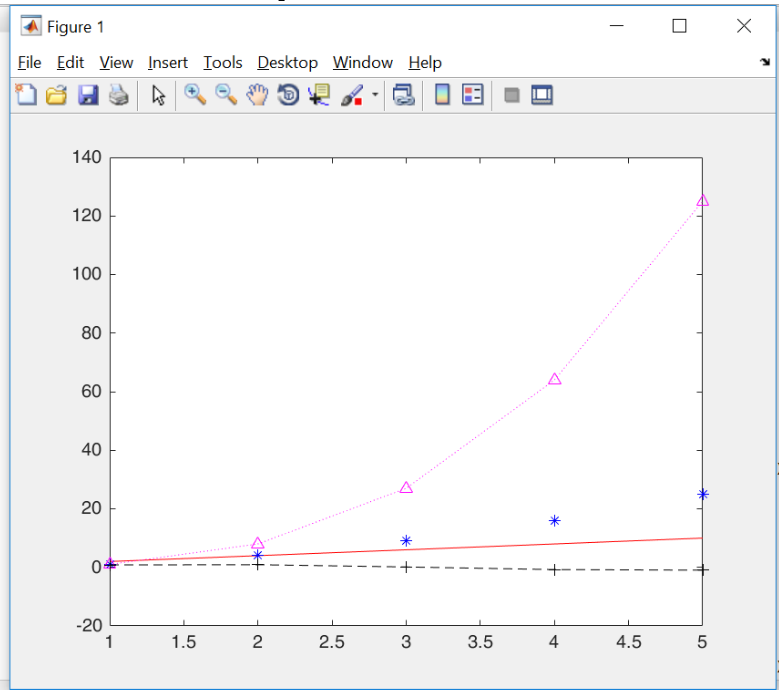
B) You can also change the line style



C) You can change the line color and style

Command Window

```
>> x=[1 2 3 4 5];
>> y=2*x;
>> Y1=x.^2;
>> Y2=x.^3;
>> Y3=sin(x);
>> plot(x,y,'r', x,Y1,'b*',x,Y2,'m:^',x,Y3,'k--+')
fx >>
```



Color	Description
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

Line Style	Description
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Marker	Description
o	Circle
+	Plus sign
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
p	Pentagram
h	Hexagram

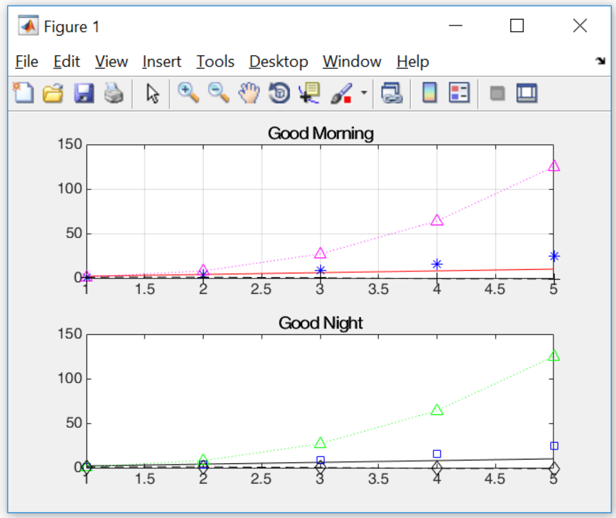
D) Subplot

If you want to have more than one plot in the same page, then the subplot can be used

subplot(number of rows, number of columns, location of figure)

Untitled2.m

```
x=[1 2 3 4 5];
y=2*x;
Y1=x.^2;
Y2=x.^3;
Y3=sin(x);
subplot(2,1,1);          %to assign the location of the plot
plot(x,y,'r', x,Y1,'b*',x,Y2,'m:^',x,Y3,'k--+')
grid on                  % to add grids
title('Good Morning')  % to add plot title
subplot(2,1,2);          %to assign the location of the plot
plot(x,y,'k-', x,Y1,'bs',x,Y2,'g:^',x,Y3,'k--d')
title('Good Night')
```



E) Scale limits

To assign limits for x and y axis, you can use the functions **ylim** and **xlim**

ylim([min max])

xlim([min max])

F) Grid

To add gird to your plot

grid on

To remove gird from your plot

grid off

G) Hold on

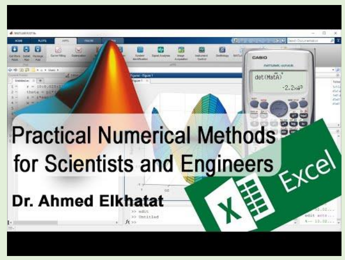
It used to keep the plots in the same chart

e.g plot (x,y) hold on plot (x2,y)

Module (2) Loops and Condition in MATLAB and Excel

1. Logical and Relational Operations in MATLAB

Watch the online Video



Suppose that you have two arrays A and B, and you want to test the relationship between them, So you can check this by using some **logical operators** such as (>, <,=, ~,&,|). If the condition is matched, it will (1) or give (0).

Let's test the following commands:

<	Smaller Than
>	Greater than
=	Equal
~=	Not equal
&	And
|	OR

> A=[1 2 3 4 5]

A =1.00 2.00 3.00 4.00 5.00

>> B=[2 1 5 4 -1]

B = 2.00 1.00 5.00 4.00 -1.00

>> A>B

ans = 1×5 logical array

0 1 0 0 1

>> A>=B

ans =1×5 logical array

0 1 0 1 1

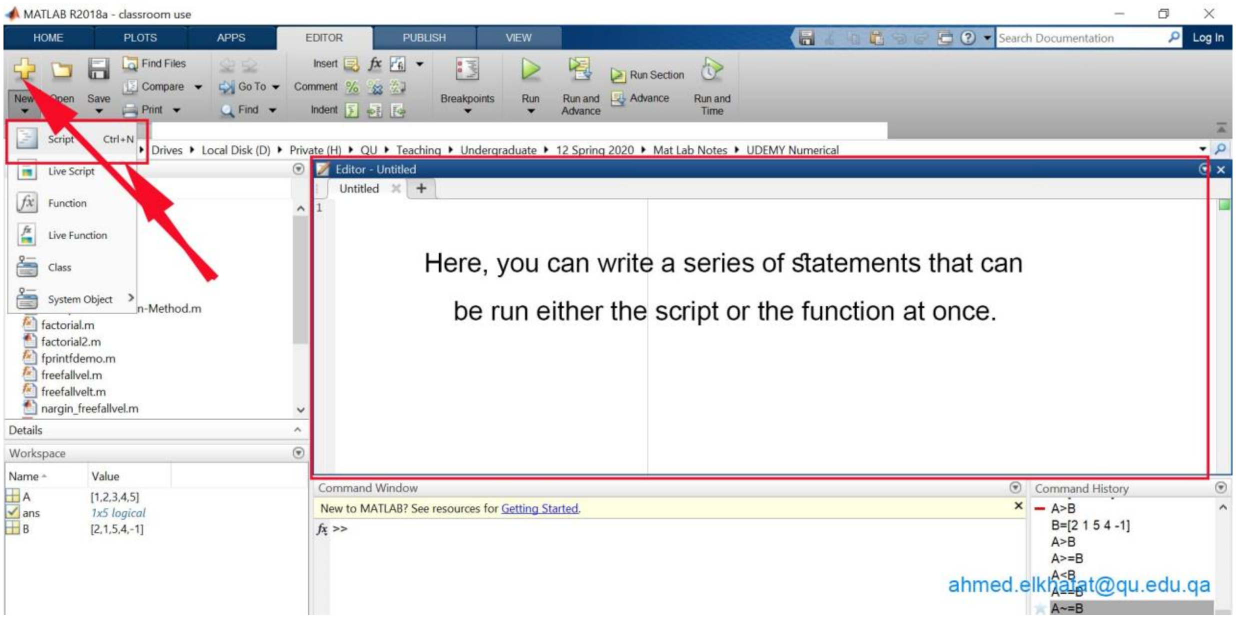
Note

if you inser (**A=B**) instead of (**A==B**), then vector of **A** array will be replaced by **B** array

```
>> A<B
ans = 1×5 logical array
1   0   1   0   0
>> A==B
ans =1×5 logical array
0   0   0   1   0
>> A~=B
ans =1×5 logical array
1   1   1   0   1
>> A==B|A~=B
ans =1×5 logical array
   1   1   1   1   1
>> find(A>B)                    %This is to find the variable location in the array
ans =
         2.00          5.00
>> B(find(B<A))                 %This is to extract value of the variable location in the array
ans =
         1.00          -1.00
```

2. Creating a script in MATLAB using M.Files

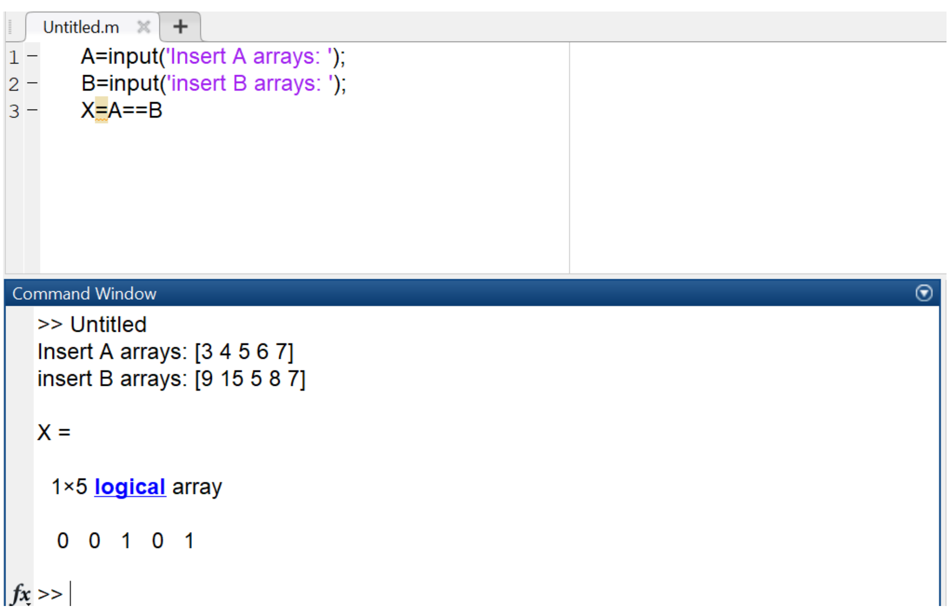
You can write a series of statements in M.file to run all at once by CTR+N or New Script.



M files can be a script file or function file.

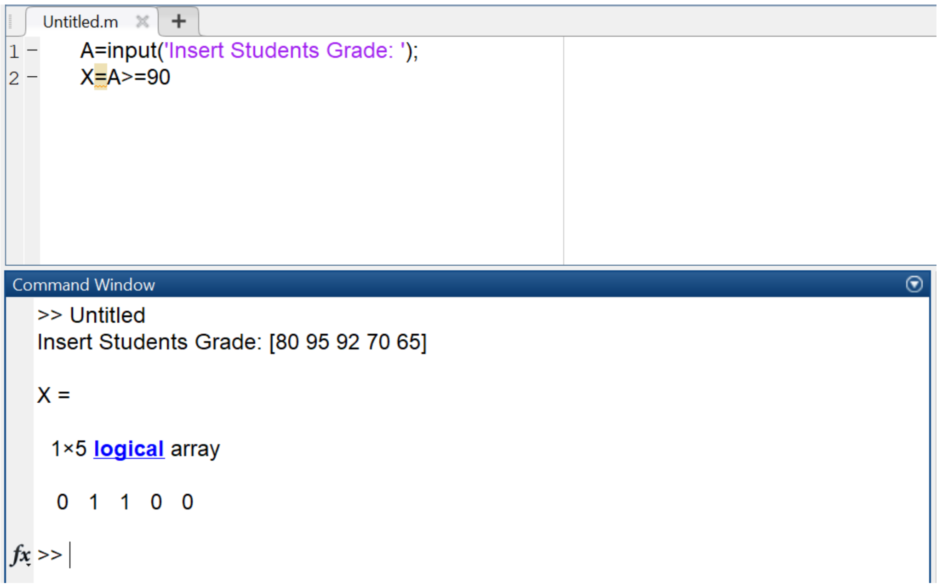
In a script file, a series of MATLAB commands can be executed at once either by typing **the saved file name** in the command window or by select **Run**.

You can perform the logical and relational operations in MATLAB using M.Files. Let us try it. In a new script file, write the following codes and save it as **Untitled**, then; in the command window, type **Untitled** or select **Run**



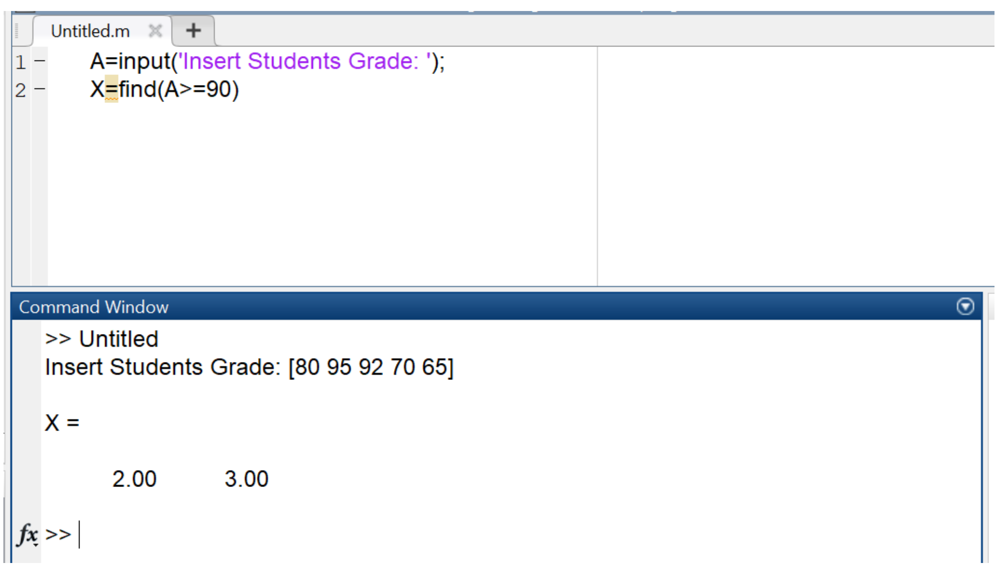
Example

Create a script Insert students grades and find who got A



A) Function (find)

It is used to locate the nonzero elements in a vector or a matrix.



Example

Create a script file where students' grades can be inserted in a vector.

Show the students' grade who got (A) (i.e. >=90) and their location in the vector.

```
Untitled.m
%ahmed.elkhatat@qu.edu.qa
A=input('Insert Students Grade: ');
X=find(A>=90);
Y=A(X);                       % to extract the values from (A)
X1=reshape(X,[],1);           %to have them in vertical vector
Y1=reshape(Y,[],1);           %to have them in vertical vector
Z=[X1, Y1];                   %to combine the vectors in a matrix
disp('     St.Sr     Grade')
disp('____________________')
disp(Z)
```

```
Command Window
>> Untitled
Insert Students Grade: [80 95 92 70 65]
     St.Sr     Grade
____________________
       2.00       95.00
       3.00       92.00
```

Example

Create a script file where students' ID and grades can be inserted in a vector.

Show the students' grade who got (A) (i.e. >=90) and their ID

```
X= input('Insert the student codes as array: ');
Y=input('Insert the student scores as array: ');
Z=find (Y>=90);
H=X(Z);
G=Y(Z);
J=[H',G'];
disp("                    ")
disp('Students who got A:')
disp('    ID        Score:')
disp('____________________')
disp(J)
```

```
Command Window
>> Untitled
Insert the student codes as array: [2314565, 34456525, 944567563, 34456752]
Insert the student scores as array: [90,89, 70,99]

Students who got A:
    ID             Score:
____________________
    2314565.00       90.00
   34456752.00       99.00

fx >>
```

Example

Create a script file to insert car plate numbers and speeds to catch those speeds>80.

```
%Car Speed by Ahmed Elkhatat
X= input('Insert the cars plate numbers as array: ');
Y=input('Insert the cars speed as array: ');
Z=find (Y>80);                  % to find cars of speed>80
H=X(Z);                         % to extract plate numbers of >80
G=Y(Z);                         % to extract plate numbers of >80
H1=reshape(H,[],1);             % to reshape the array into a column
G1=reshape(G,[],1);             % to reshape the array into a column
J=[H1 G1];                      % to create a matrix of the car speed and plate numbers
disp('                ')        % to create a space
disp('Cars exceed the speed limits:')
disp('    Plate      speed:')
disp('_______________________')
disp(J)
```

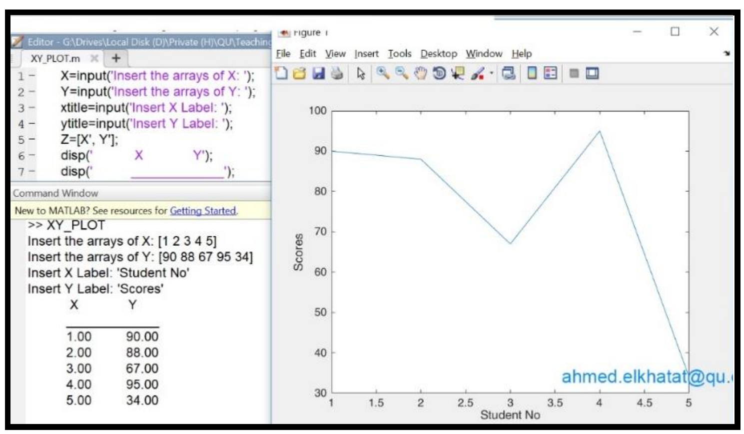
```
Command Window
>> Lab2_cars
Insert the cars plate numbers as array: [123 456 324 345 333 678]
Insert the cars speed as array: [80 98 50 110 60 99]

Cars exceed the speed limits:
   Plate              speed:
_______________________
      456.00      98.00
      345.00     110.00
      678.00      99.00

fx >>
```

Example

Create M file to list values of X and Y in a proper table and plot them

```
XY_PLOT.m
X=input('Insert the arrays of X: ');
Y=input('Insert the arrays of Y: ');
xtitle=input('Insert X Label: ');
ytitle=input('Insert Y Label: ');
Z=[X', Y'];
disp('         X           Y');
disp('       ______________________');
disp(Z);
plot(X,Y);
xlabel(xtitle);
ylabel(ytitle);
```

ahmed.



B) Function (tic&toc)

tic and **toc** functions are used to calculate the elapsed time between them

pause function is used to hold for a certain number of seconds

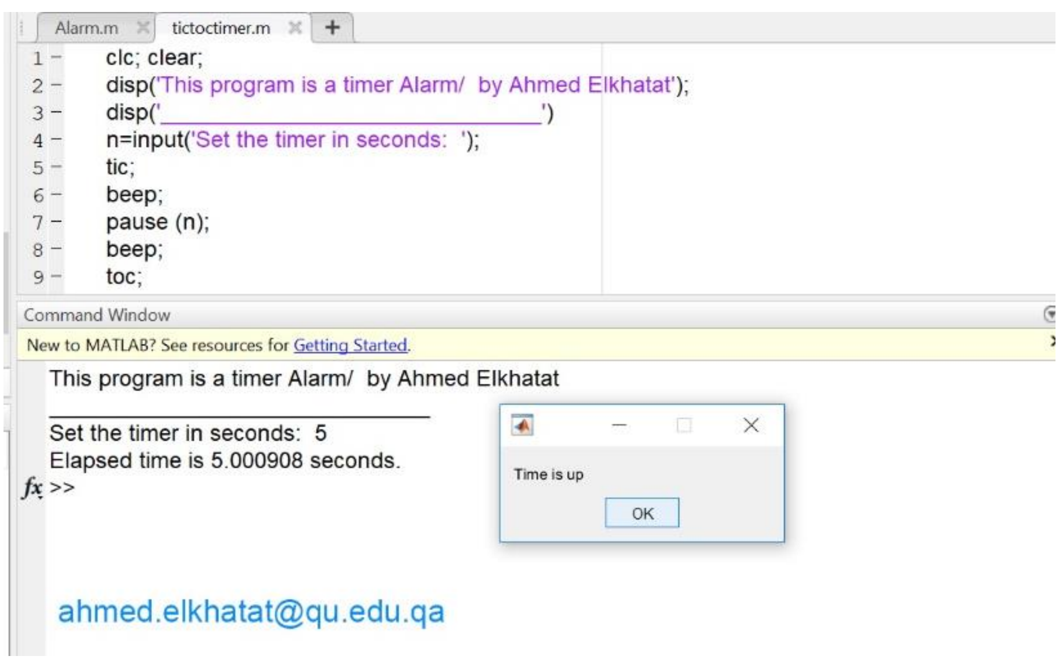
beep function is to produce a beep sound

msgbox function is to create a message box

Example

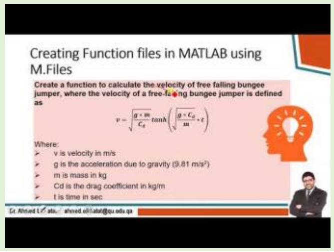
Create M file to Program a Timer Alarm in seconds

```
clc; clear;
disp('This program is a timer Alarm/  by Ahmed Elkhatat');
disp('_______________________________')
n=input('Set the timer in seconds:  ');
tic;
beep;
pause (n);
beep;
toc;
msgbox('Time is up')
```



3. Creating Function files in MATLAB using M.Files

Watch the online Video



Functions are files that take specific inputs and execute a sequence of steps and then return outputs at the end.

Function files has different forms, and their syntax can be presented as following:

function output= function_name (input 1, input 2,..)

function [out 1, out2,..] = function_name (input 1, input 2,..)

function [out 1, out2,..]= function_name

function function_name (input 1, input 2,..)

function function_name

Important Notes:

Note (1) :

You can use **%** to add comments to your script. This will be very helpful as you can recall these comments if you write the command **help (function name)**

Note (2):

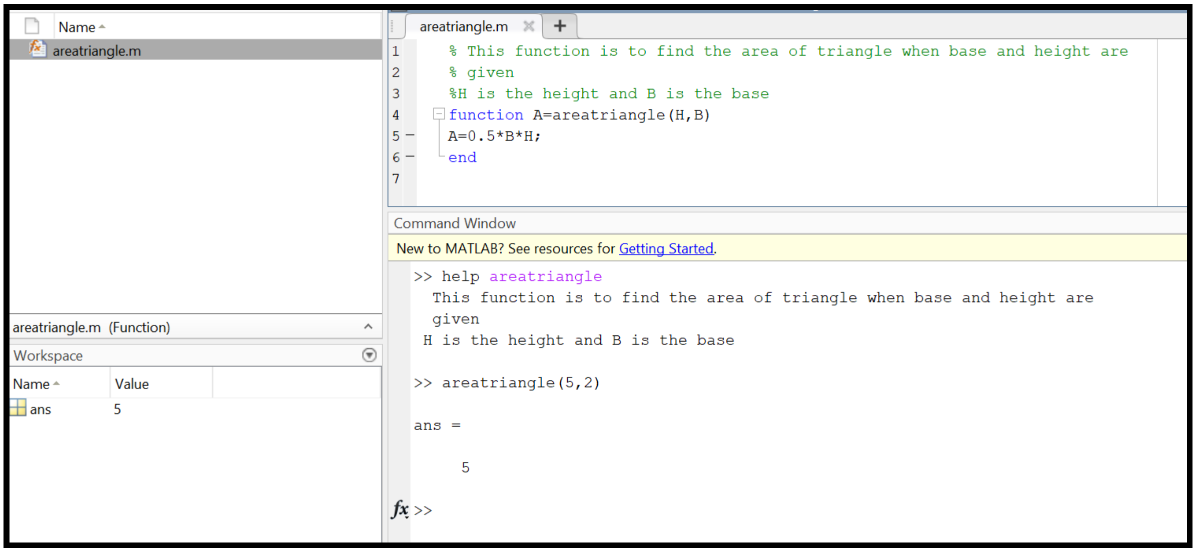
When You save the function, make sure that the **file name** is precisely the **function name** to execute the function correctly.

Note (3):

When You execute the function, make sure that the **function** is in the correct directory (Correct current folder window).

Example

Create a function to find the area of a triangle when its height and base are given.



Example

Create a function to calculate the velocity of a free-falling bungee jumper, where the velocity of a free-falling bungee jumper is defined as

$$v = \sqrt{\frac{g*m}{C_d}}\,tanh\left(\sqrt{\frac{g*C_d}{m}}*t\right)$$

Where:

- v is velocity in m/s
- g is the acceleration due to gravity (9.81 m/s^2)
- m is mass in kg
- Cd is the drag coefficient in kg/m
- t is time in sec

Editor - C:\Users\ahmed.elkhatat\Documents\MATLAB\bungeevel.m

bungeevel.m

```
%This function will be used to find the  velocity of a free falling bungee
%jumper, where the velocity of a free falling bungee jumper when time (t),
%mass(m) grag coefficient (Cd) are given.
function velocity=bungeevel(m,t,cd);
g=9.81;
velocity=sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);
end
```

Command Window

```
>> bungeevel(5,10,0.3)

ans =

       12.79

>> help bungeevel
 This function will be used to find the  velocity of a free falling bungee
 jumper, where the velocity of a free falling bungee jumper when time (t),
 mass(m) grag coefficient (Cd) are given.

fx >>
```

If you want to create a function without an output (i.e., to assign the output later as a display message, then you can write the function in this form:

function function_name (input 1, input 2,..)

bungeevel1.m

```
%This function will be used to find the  velocity of a free falling bungee
%jumper, where the velocity of a free falling bungee jumper when time (t),
%mass(m) grag coefficient (Cd) are given.
function bungeevel1(m,t,cd);
g=9.81;
v=sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);
fprintf('The velocity of the jumber at %0.0f s is %0.2f m/s \n', t, v)
end
```

Command Window

```
>> bungeevel1(5,10,0.3)
The velocity of the jumber at 10 s is 12.79 m/s
>>
```

Example

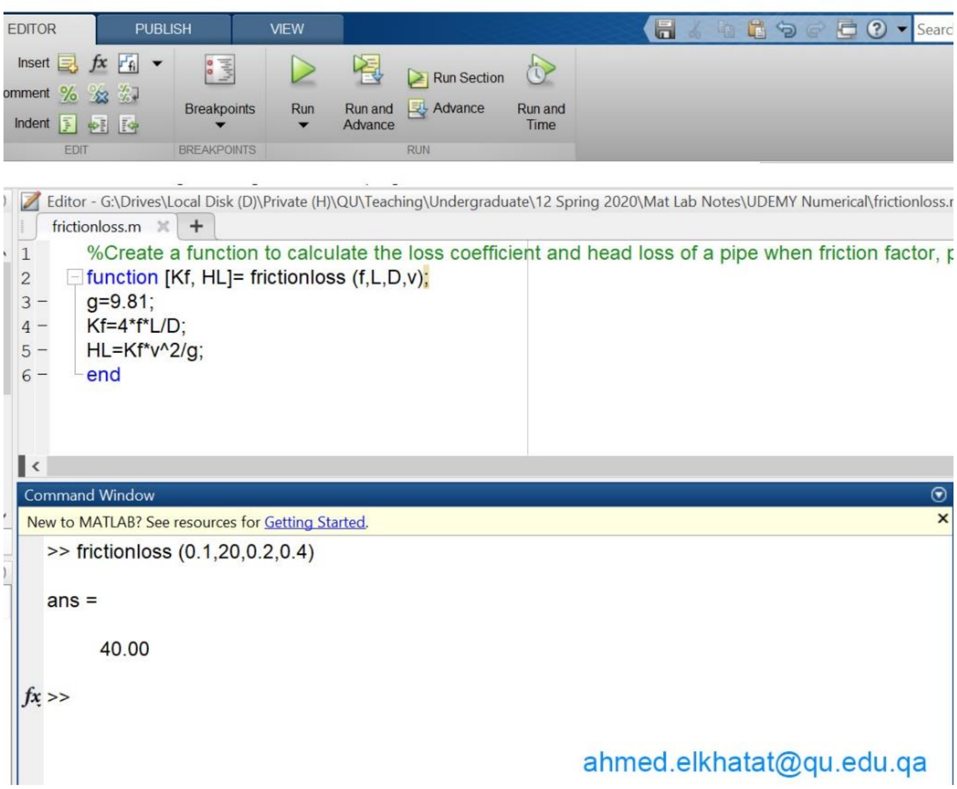
Create a function to calculate the loss coefficient and head loss of a pipe when friction factor, pipe length, pipe diameter, and fluid velocity are given

$$Kf = 4f\frac{L}{D}$$

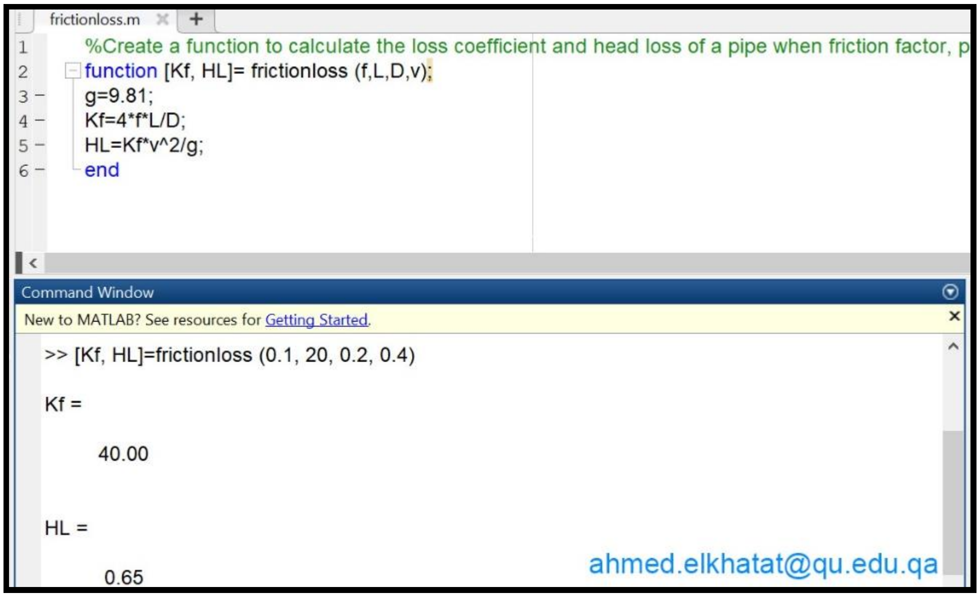
$$HL = Kf * \frac{v^2}{2g}$$

Where:

- v is velocity in m/s
- g is the acceleration due to gravity (9.81 m/s^2)
- f the friction factor (fanning)
- L is the pipe length (m)
- D is the pipe diameter (m)
- Kf is the loss coefficient
- HL is the head loss (m)

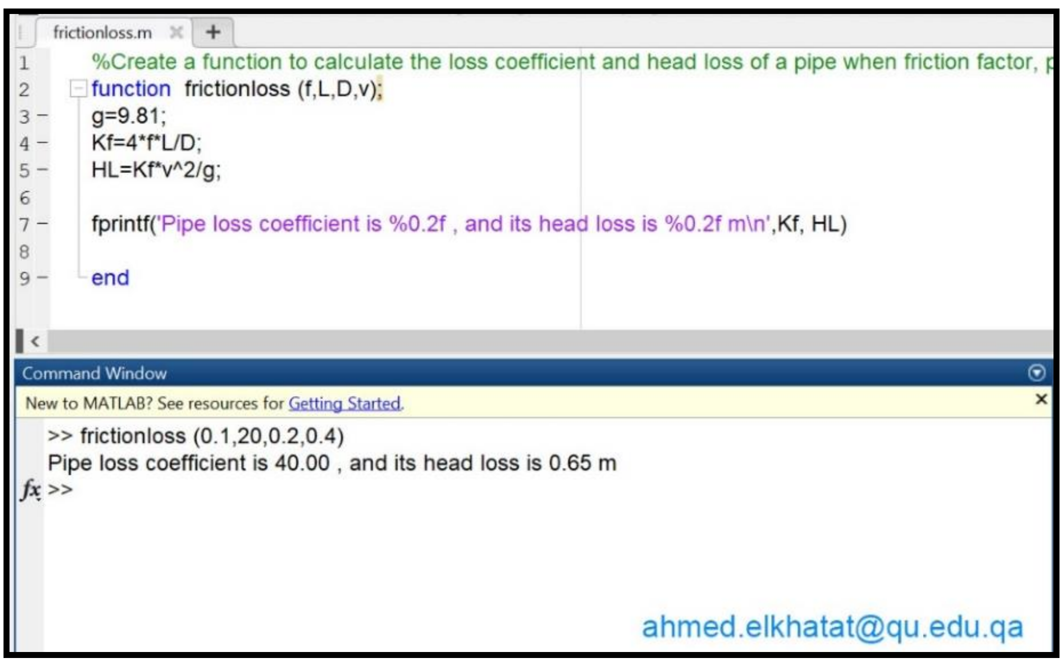


Note that you got only one value, so to get the other values, the function should be written in the command window as:



Another way to create the function file using

function function_name (input 1, input 2,..)



A) nargin (Number of function input arguments)

It returns the number of function input arguments given in the call to the currently executing function.

example: for the function (trianglearea); in order to find the number of input argument

>> nargin("trianglearea")

ans =

2.00

it is also helpful to set different scenarios for the inputs. For example, to let the function accept one or two input arguments.

Example

Create a function that finds the multiply of two numbers. If the user puts only one number, then find its squared value.

```
function x=multi(a,b)
switch nargin
    case 2
        x=a*b;
    case 1
        x=a^2;
end
end
```

```
Command Window
>> multi(2,5)

ans =

       10.00

>> multi(2)

ans =

        4.00

fx >>
```

Example

In the previous example to create a function to calculate the velocity of a free-falling bungee jumper, where the velocity of a free-falling bungee jumper is defined as

$$v = \sqrt{\frac{g * m}{C_d}} \tanh\left(\sqrt{\frac{g * C_d}{m}} * t\right)$$

Use ***nargin*** to allow either consider (g) as input argument or ignore it

bungeevel2.m

```
function v=bungeevel2(m,t,cd,g);
switch nargin
    case 4
v=sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);
    case 3
        g=9.81; v=sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);
end
```

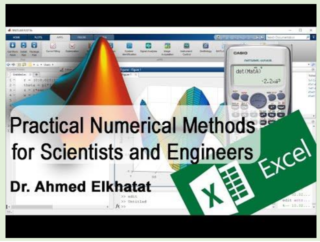
Command Window

New to MATLAB? See resources for Getting Started.

```
>> bungeevel2(5,10,0.3,9.81)

ans =

   12.7867

>> bungeevel2(5,10,0.3)

ans =

   12.7867
```

4. Loops and Conditions using (If expression) in MATLAB.

Watch the online Video



<u>If expression>> statements>>end</u>

evaluates an expression and executes a group of statements when the expression is true. An expression is true when it is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

Syntax

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

Example

Create M file to calculate the Body Mass Index (BMI) and inform the user what the index means using (if Condition) in a function file.

$$BMI = \frac{Weight\ (kg)}{Height^2\ (m)}$$

Note:

- BMI <18.5, then it is Underweight Body
- 18.5 < BMI < 24.9, them it is Normal Body
- 24.9 < BMI < 29.9 it is Overweight Body
- BMI > 29.9, then it is Obese Body

```
function BMIX1 (W,H);
%This function calculates the BMI of the Weight (Kg) an Height (m) are given
BMI=W/H^2;
if BMI<18.5;
   fprintf('Your BMI is %0.0f, and this indicates an Underweight Body \n', BMI)
end
if BMI>18.5 & BMI<24.9;
    fprintf('Your BMI is %0.0f, and this indicates a  Normal Body \n', BMI)
end
if BMI>24.9 & BMI<29.9;
    fprintf('Your BMI is %0.0f, and this indicates an Overweight Body \n', BMI)
end
if BMI >29.9;
   fprintf('Your BMI is %0.0f, and this indicates an Overweight Body \n', BMI)
end
end
```

Try it using (if/ ifelse) and nargin

```
function BMIX1 (W,H);
%This function calculates the BMI of the Weight (Kg) an Height (m) are given
if(nargin ~=2)
 disp('Error');
 return;
end
BMI=W/H^2;
if BMI<18.5;
    fprintf('Your BMI is %0.0f, and this indicates an Underweight Body \n', BMI)

elseif BMI<24.9;
     fprintf('Your BMI is %0.0f, and this indicates a  Normal Body \n', BMI)

elseif  BMI<29.9;
     fprintf('Your BMI is %0.0f, and this indicates an Overweight Body \n', BMI)
else
    fprintf('Your BMI is %0.0f, and this indicates an Overweight Body \n', BMI)
end
end
```

Example

Create M file to show a student's name and show his Graded letter using input function and (if Condition). (Inputs is name, %, and Course Code)

BMIXi.m

```
1  disp('This is to calculate a student total score and Grade letter in course');
2  disp('written by Ahmed Elkhatat');
3  disp('--------------------------------------------------------------------------------------');
4  disp(' ');
5  Name= input(' What is the student Names ? ');
6  Course= input(' What is the Course Code ? ');
7  Score= input(' Insert the total scores:   ');
8  if Score<60;
9     fprintf('The Final Grade letter of (%s) in (%s) is F \n', Name, Course);
10 elseif  Score<65
11     fprintf('The Final Grade letter of (%s) in (%s) is D \n', Name, Course);
12 elseif Score<70
13     fprintf('The Final Grade letter of (%s) in (%s) is D+ \n', Name, Course);
14 elseif Score<75
15     fprintf('The Final Grade letter of (%s) in (%s) is C  \n', Name, Course);
16 elseif Score<80
17     fprintf('The Final Grade letter of (%s) in (%s) is C+  \n', Name, Course);
18 elseif Score<85
19     fprintf('The Final Grade letter of (%s) in (%s) is B  \n', Name, Course);
20 elseif  Score<90
21     fprintf('The Final Grade letter of (%s) in (%s) is B+  \n', Name, Course);
22 else
23    fprintf('The Final Grade letter of (%s) in (%s) is A  \n', Name, Course);
24 end
25
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> BMIXi
This is to calculate a student total score and Grade letter in course
written by Ahmed Elkhatat
--------------------------------------------------------------------------------------

 What is the student Names ? Ahmed Elkhatat
Error using BMIXi (line 5)
Error: Invalid expression. Check for missing multiplication operator,
missing or unbalanced delimiters, or other syntax error. To construct
matrices, use brackets instead of parentheses.

 What is the student Names ? 'Ahmed Elkhatat'
 What is the Course Code ? 'GENG 300'
 Insert the total scores:   89
The Final Grade letter of (Ahmed Elkhatat) in (GENG 300) is B+
>>
```

if the variable is a string, You shoud insert your variable as 'x'

Example

Create a function to calculate the labor payment according to the following rules.

100 QR/hr for the first 35 hrs.

150 QR/hr for the next 25 hrs.

200 Qr/hr for further hrs.

pay.m

```
function QR=pay(h)
if h<=35; QR=h*100;
elseif h<=60; QR=(35*100)+((h-35)*150);
else QR=(35*100)+(25*150)+((h-60)*200);
end
end
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> pay(12)

ans =

        1200

>> pay(50)

ans =

        5750

>> pay(80)

ans =
```

Example

Create M file to calculate the Calculate a series of students Grade letters using (Students' Scores should be inserts as an array.

scores_array.m

```
x=input('Insert students scores as array: ');
for i=1:length(x)
if x(i)<60;
fprintf('Student %.0f got [F], and his score is %.2f \n', i, x(i));
elseif x(i)<70;
    fprintf('Student %.0f got [D], and his score is %.2f \n', i, x(i));
elseif x(i)<80;
    fprintf('Student %.0f got [C], and his score is %.2f \n', i, x(i));
elseif x(i)<90;
    fprintf('Student %.0f got [B], and his score is %.2f \n', i, x(i));
else
    fprintf('Student %.0f got [A], and his score is %.2f \n', i, x(i));
end
end
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> scores_array
Insert students scores as array: [77, 64, 90]
Student 1 got [C], and his score is 77.00
Student 2 got [D], and his score is 64.00
Student 3 got [A], and his score is 90.00
```

Example

Create if loop file where one can insert his blood pressure. A message telling him about his health status is as follows:

1) Low blood pressure (hypotension): If the top number (systolic) is lower than 90 mm Hg or the bottom number (diastolic) is lower than 60 mm Hg
2) High blood pressure (hypertension): If the top number (systolic) is higher than 120 mm Hg or the bottom number (diastolic) is higher than 80 mm Hg
3) Normal blood pressure: If the top number (systolic) is between (90-120) mm Hg or the bottom number (diastolic) is between (60-80) mm Hg

Blood.m

```
%Blood Pressure Ahmed Elkhatat
clc;
L=input('insert the value of the bottom number (diastolic) of the blood pressure: ');
H=input('insert the value of the top number (systolic) of the blood pressure: ');

if L<60|H<90;
disp('Low blood pressure (hypotension)')
elseif L<=80|H<=120;
disp('Normal blood pressure')
else;
disp('High blood pressure (hypertension)')
end
```

Command Window

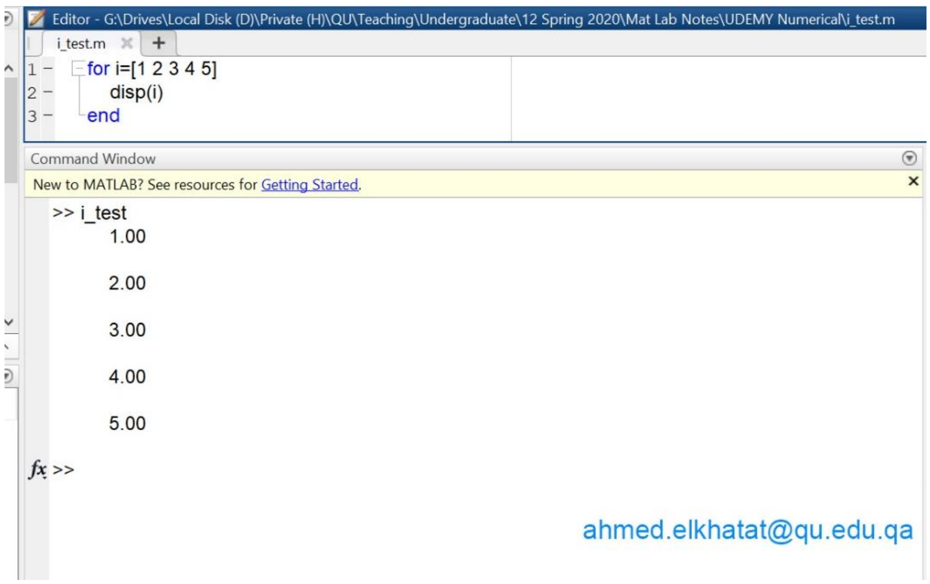
```
insert the value of the bottom number (diastolic) of the blood pressure: 80
insert the value of the top number (systolic) of the blood pressure: 120
Normal blood pressure
fx >>
```

5. Loops and Conditions using (For expression) in MATLAB.

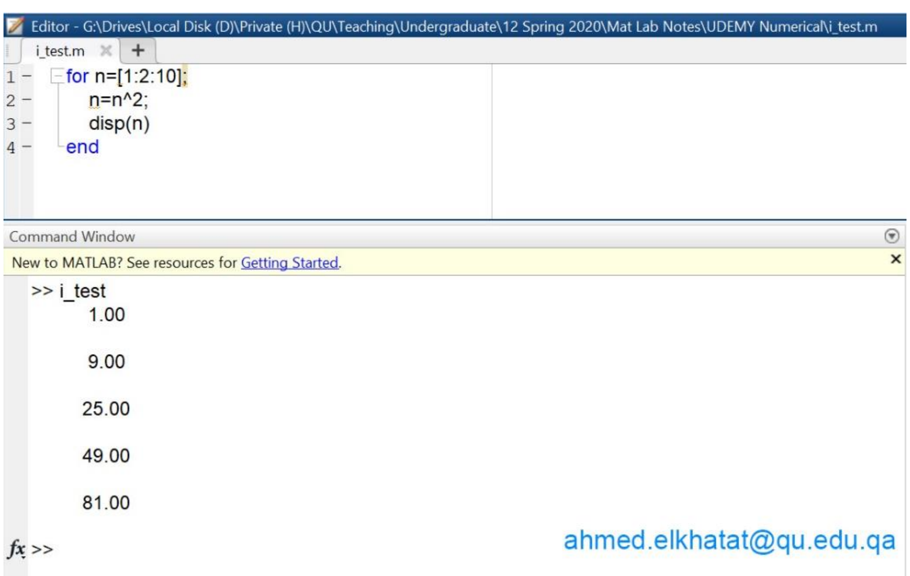
<u>For loop</u> is used to repeat a specified number of times

Syntax
```
for index = values
  statements
end
```

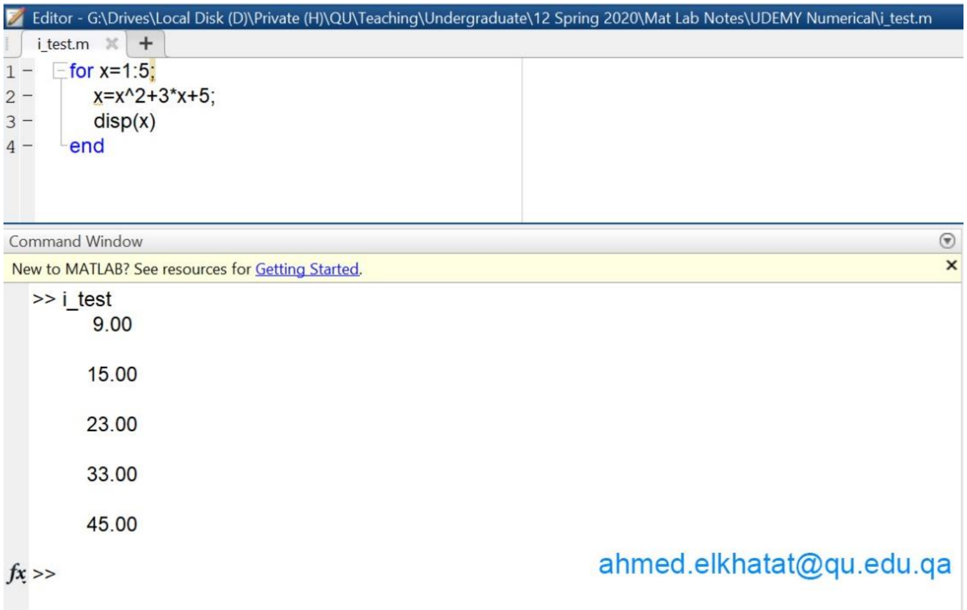
Example 1:



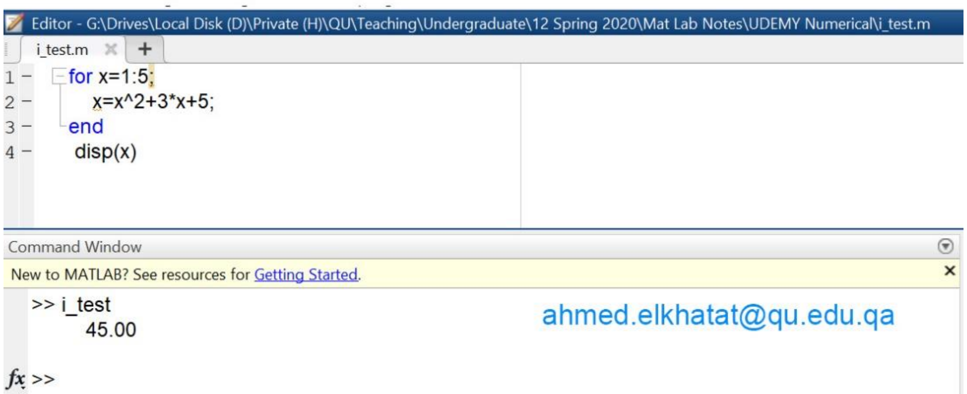
Example 2:



Example 3:



Note: if you want the MATLAB only to show the last value, then get disp(x) out of the loop as following



Example 4:

Use a For loop to calculate the factorial number.

Fact.m | Fact2.m | factorial.m

```
n=input('insert the number to calculate its factorial: ');
x=1;
for i=1:n;
    x=x*i;
end

fprintf('The factorial number of [%.0f] is [%0.f]\n',n,x)
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> Fact2
insert the number to calculate its factorial: 10
The factorial number of [10] is [3628800]
>> Fact2
insert the number to calculate its factorial: 5
The factorial number of [5] is [120]
fx >>
```

ahmed.elkhatat@qu.edu.qa

6. Loops and Conditions using (While expression) in MATLAB.

While loop is used to repeat when Condition is true.

Syntax

```
while expression
  statements
end
```

Example

Use while loop to calculate the factorial number.

```
Fact.m
n = input('Put the nubmer you want to calculate its factorial number: ');
f = n;
while n > 1
    n = n-1;
    f = f*n;
end
disp(['n! = ' num2str(f)])
```

```
Command Window
New to MATLAB? See resources for Getting Started.
>> Fact
Put the nubmer you want to calculate its factorial number: 10
n! = 3628800
>> Fact
Put the nubmer you want to calculate its factorial number: 5
n! = 120
>>
```

While True

while True means loop forever. The while statement runs the loop body while the expression evaluates to (boolean) "true".

```
while true
   % statements here
   % if ~WhileCondition, break; end
end
```

Example:To find a random pythagorean triple

```
clear all; clc;
i=1;
while true
    a  = randi(20);
    b = randi(20);
    c = randi(20);
   Table(i,:)=[i a b c];
    if a^2+b^2 == c^2 break; end
      i=i+1
end
Table
```

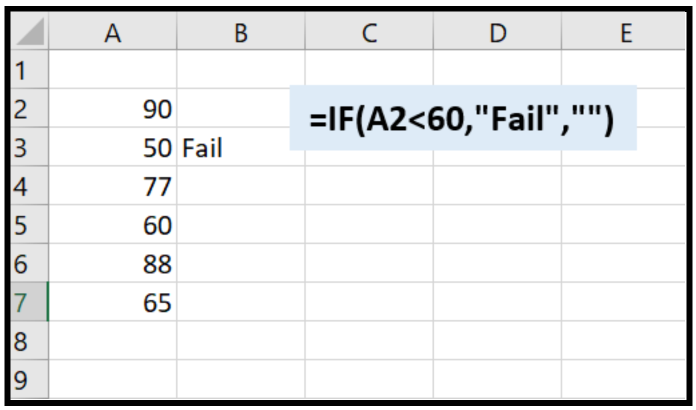
7. IF Condition using in Excel

The most straightforward formula of (IF) Condition in EXCEL is

IF(logical test, value if true, value if false)

Example

For the following given data, if the value is less than 60, then show Fail. If not, do nothing



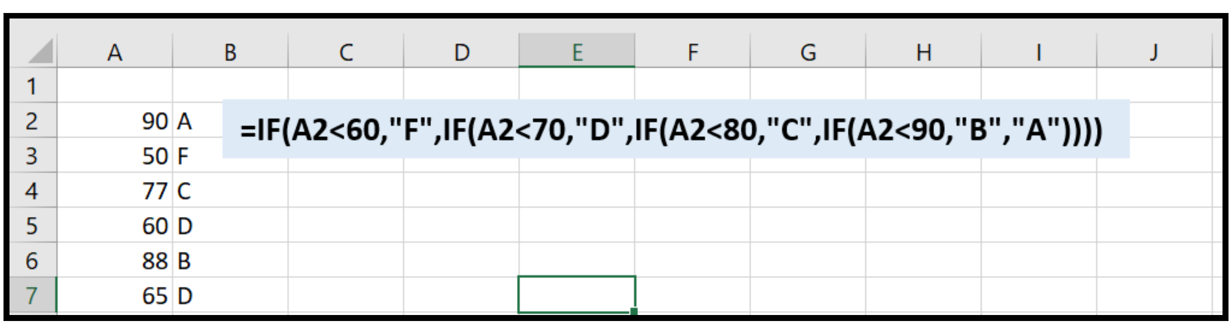
	A	B	C	D	E
1					
2	90				
3	50	Fail			
4	77				
5	60				
6	88				
7	65				
8					
9					

Note that we put Fail between two double quotes because it's text, not value. Also the false option, we put two empty double quotes ("") to do nothing. For not equal Condition, you can use <>.

Example

In the same example, if the value is less than 60, then show Fail. If it is less than 70, show D. If it is less than 80, then show C, Less than 90, then show B, else show A.

In this Condition, we replace the (false value) with the new conditional as follows:



	A	B	C	D	E	F	G	H	I	J
1										
2	90	A								
3	50	F								
4	77	C								
5	60	D								
6	88	B								
7	65	D								

For more complicated logical tests, (and) (or) can be used as following

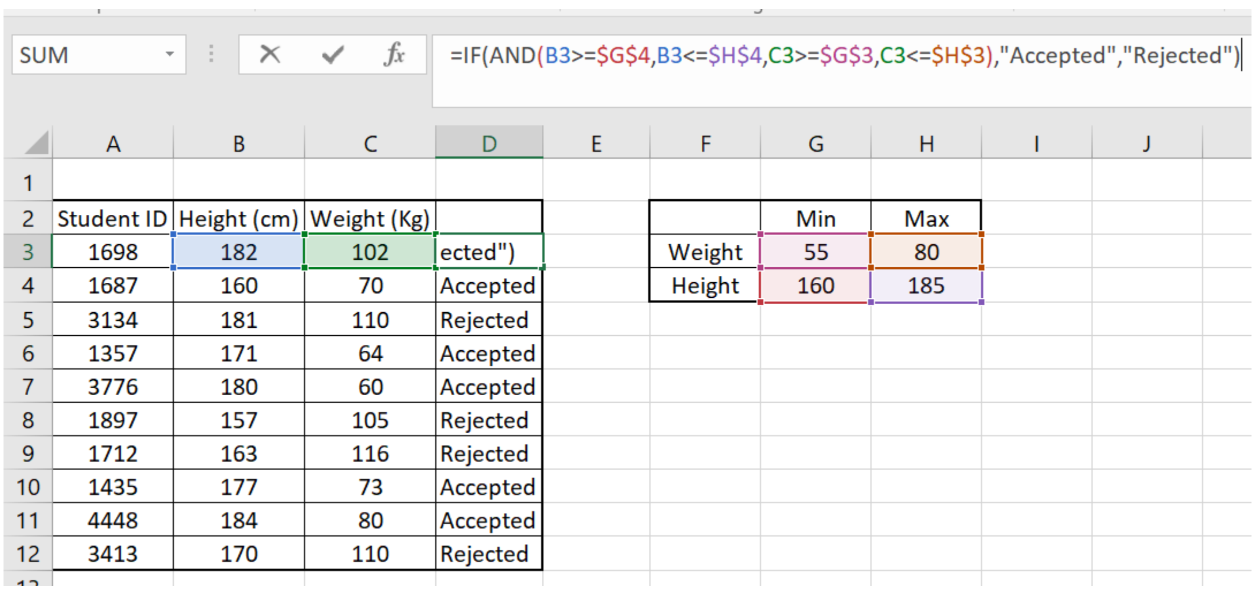
IF(and(logical test_1, logical test_2),value if true, value if false)

IF(or(logical test_1, logical test_2),value if true, value if false)

Example

A military college put height and weight standards for accepting new students as follows (Height 160-185 Kg) and (weight 55-80 Kg). Applicants out of these ranges should be rejected.

Student ID	Height (cm)	Weight (Kg)
1698	182	102
1687	160	70
3134	181	110
1357	171	64
3776	180	60
1897	157	105
1712	163	116
1435	177	73
4448	184	80
3413	170	110



SUM

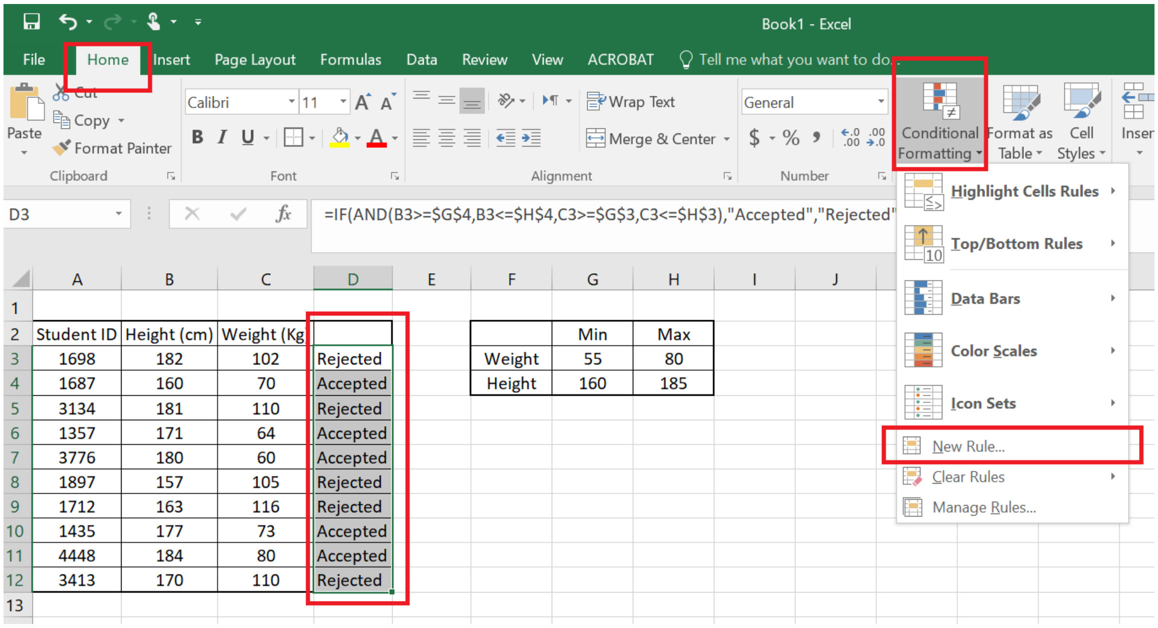
=IF(AND(B3>=$G$4,B3<=$H$4,C3>=$G$3,C3<=$H$3),"Accepted","Rejected")

	A	B	C	D
2	Student ID	Height (cm)	Weight (Kg)	
3	1698	182	102	ected")
4	1687	160	70	Accepted
5	3134	181	110	Rejected
6	1357	171	64	Accepted
7	3776	180	60	Accepted
8	1897	157	105	Rejected
9	1712	163	116	Rejected
10	1435	177	73	Accepted
11	4448	184	80	Accepted
12	3413	170	110	Rejected

F	G	H
	Min	Max
Weight	55	80
Height	160	185

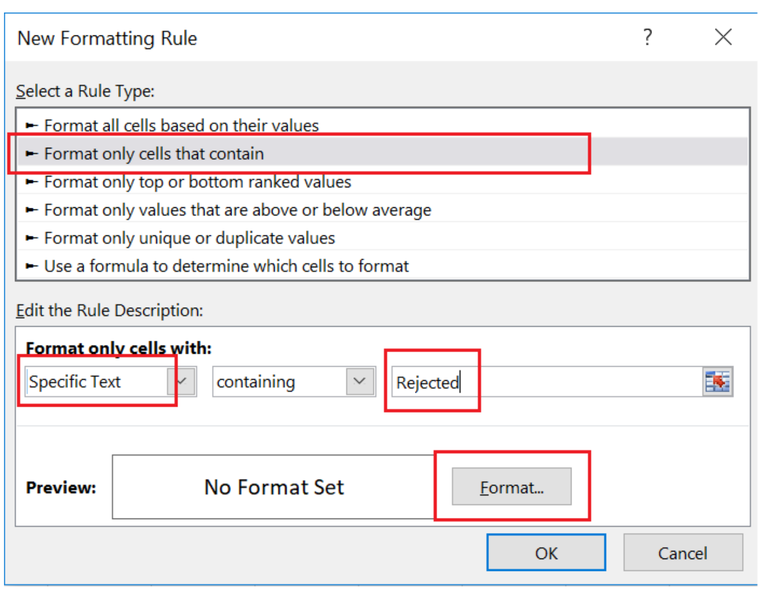
If you want to **color the Accepted cells** in green and the rejected cells in red, then

a) Select the Colum you want to apply the conditional formatting.
b) In the **Home tab**, select **Conditional Formatting**, from the drop list, select **New Rule**, as shown below

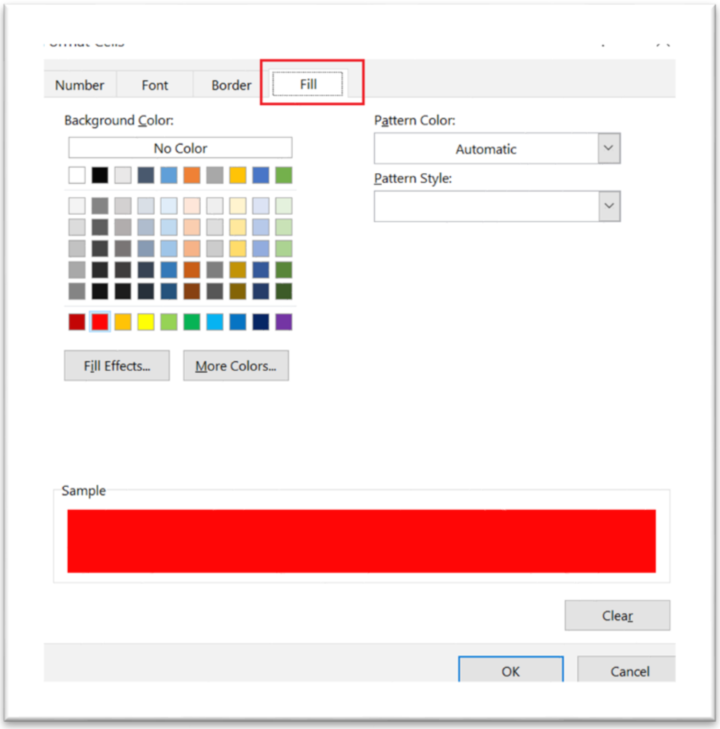


In the New Formatting Rule window

a) Rule Type: Select **Format only cells that contain**.
b) Edit the Rule Description: Format only cells with **Specific Text** and type the word or select the cell that contains the word (**Rejected**), then select **Format**.



c) In the Format Cells window Select **Fill**, Choose the color, then **OK**

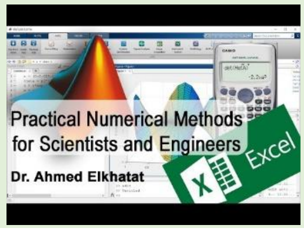


Student ID	Height (cm)	Weight (Kg)	
1698	182	102	Rejected
1687	160	70	Accepted
3134	181	110	Rejected
1357	171	64	Accepted
3776	180	60	Accepted
1897	157	105	Rejected
1712	163	116	Rejected
1435	177	73	Accepted
4448	184	80	Accepted
3413	170	110	Rejected

d) Repeat the same steps to color "Accepted in Green."

Module (3): Finding Roots of Single Equation using Excel and MATLAB

Watch the online Video



Graphical methods, Bracket methods, and Open methods are different methods to solve the roots of a single equation.

In Bracketing methods, two guesses for the root are required, which must bracket the root. These methods permanently reduce the width of the bracket, so they are said to be always convergent. In contrast, Open Methods need either one guess for the root or two guesses, but they don't necessarily bracket the root.

Bracket Methods	Open Methods
Graphical Method	Simple Fixed point
Bisection Method	Newton Raphson
False Position Method	Secant Method

1. Measures of Errors

There are four common measures of errors in numerical methods:

True Value=Approximation (Solution) + Error

1) True error:

$$X_{true} - X_{Numerical\,Solution}$$

2) Tolerance in function $f(x)$

$$|f(X)_{true} - f(X)_{Numerical\,Solution}|$$

$$|0 - f(X)_{Numerical\,Solution}|$$

3) **Tolerance in solution** (used for bracketing methods)

$$\frac{f(X_U) + f(X_L)}{2}$$

4) **Relative error** – used for iterative solutions

$$\left|\frac{X_{Numerical\ Solution(n)} - X_{Numerical\ Solution(n-1)}}{X_{Numerical\ Solution(n)}}\right|$$

2. Graphical Method

A simple method for estimating the root of $f(x) = 0$ by plotting the function and observing where it crosses the (x).
Both MatLab and Excel can be used to find it as follows:

MatLab

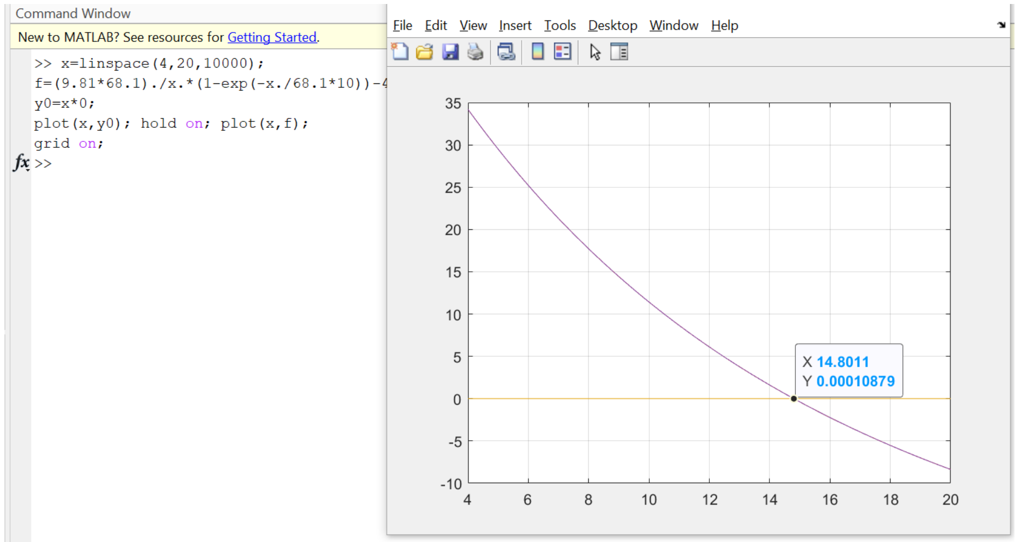
1) Using linspae create x points the potentially cover the function limits
x=linspace(-10,10,10)
2) Create the F(x) function
3) Create y points equal zeros
4) Plot the function

Example

Use the graphical approach to determine the drag coefficient (C) needed for a parachutist of mass m=68.1 kg to have a velocity of 40 m/s after free-falling for time t=10 s (4 and 20 are the initial guesses)

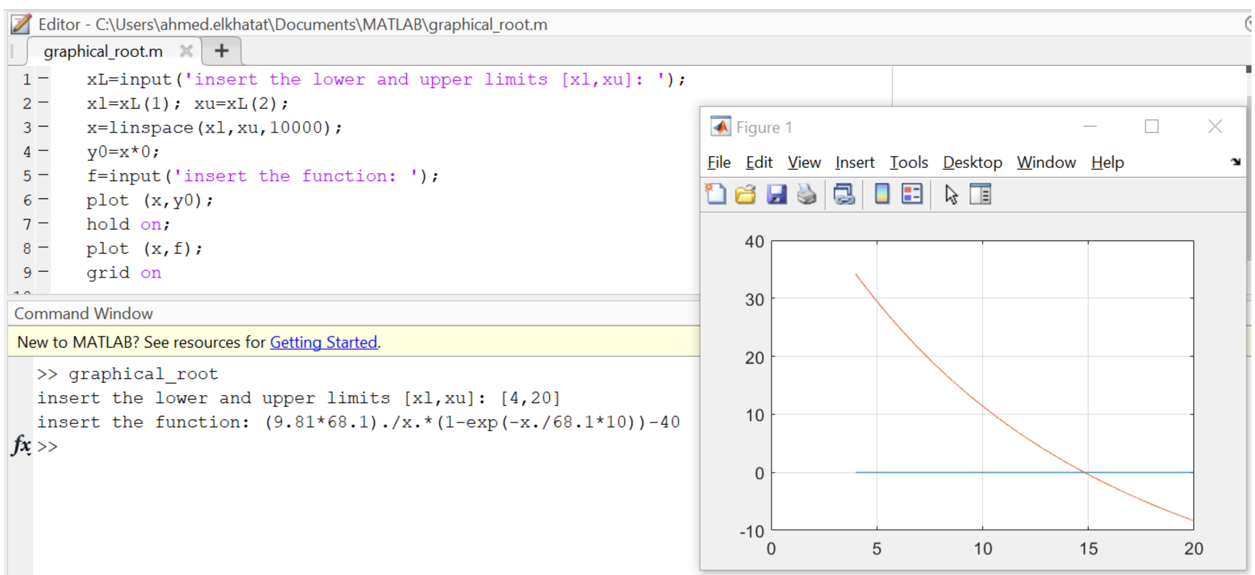
$$\boldsymbol{f(c) = \frac{g * m}{C} * \left(1 - e^{-\left(\frac{C}{m}\right)*t}\right) - v = 0}$$

```
x=linspace(4,20,10000);
f=(9.81*68.1)./x.*(1-exp(-x./68.1*10))-40;
y0=x*0;
plot(x,y0); hold on; plot(x,f);
grid on;
```



Example

Create m file (input file) allow to find roots of single equation graphically



Excel

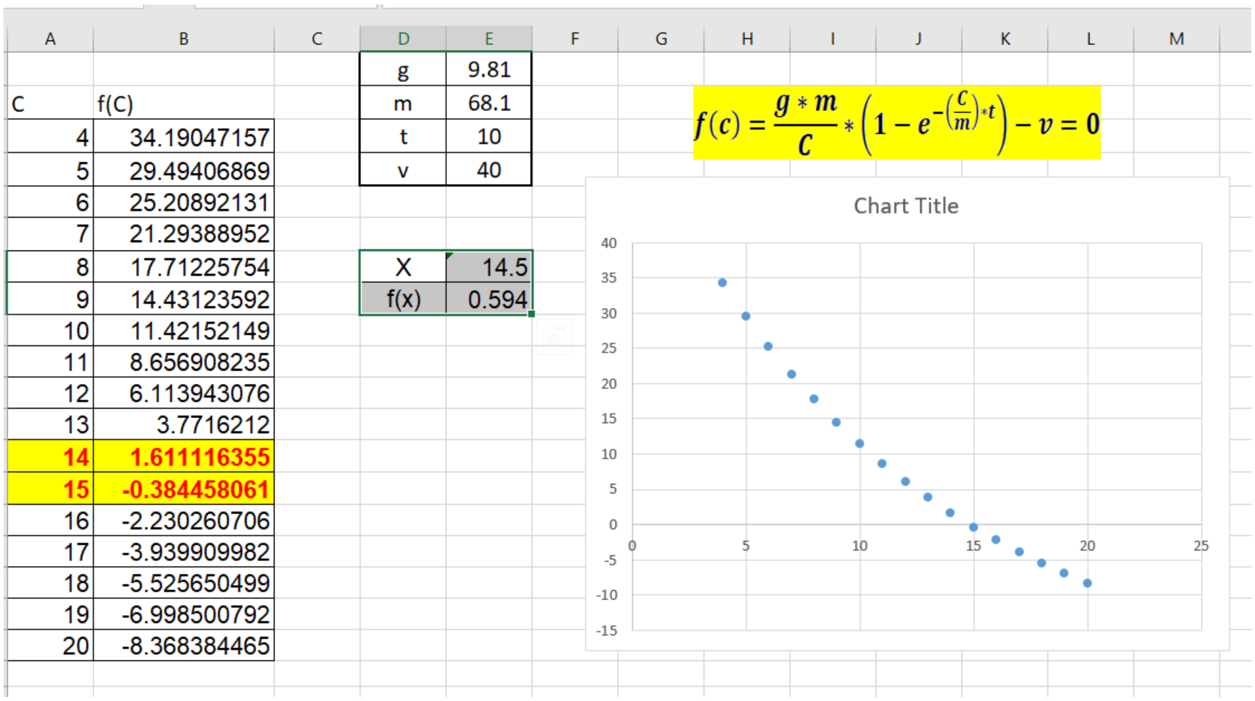
1) Begin with two guesses that bracket the real root.
2) Check when $f(x) = +, f(x) = -$.
3) Take the related (x) of the previous step as new brackets.
4) Repeat steps (3) and (4) until getting a satisfactory result.

Example 2:

Use the graphical approach to determine the drag coefficient (C) needed for a parachutist of mass m=68.1 kg to have a velocity of 40 m/s after free-falling for time t=10 s (4 and 20 are the initial guesses)

$$\boldsymbol{f(c) = \frac{g * m}{C} * \left(1 - e^{-\left(\frac{C}{m}\right)*t}\right) - v = 0}$$

First Trail:



C	f(C)
4	34.19047157
5	29.49406869
6	25.20892131
7	21.29388952
8	17.71225754
9	14.43123592
10	11.42152149
11	8.656908235
12	6.113943076
13	3.7716212
14	1.611116355
15	-0.384458061
16	-2.230260706
17	-3.939909982
18	-5.525650499
19	-6.998500792
20	-8.368384465

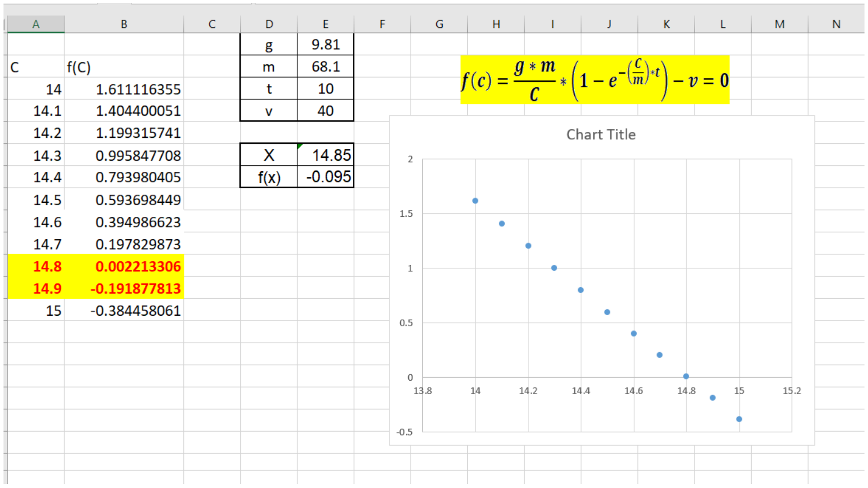
g	9.81
m	68.1
t	10
v	40

X	14.5
f(x)	0.594

$$f(c) = \frac{g * m}{C} * \left(1 - e^{-\left(\frac{C}{m}\right)*t}\right) - v = 0$$

$$\boldsymbol{Tolerance\ in\ function\ f(x) = |f(X)_{true} - f(X)_{Numerical\ Solution}| = 0.594}$$

Second Trail:



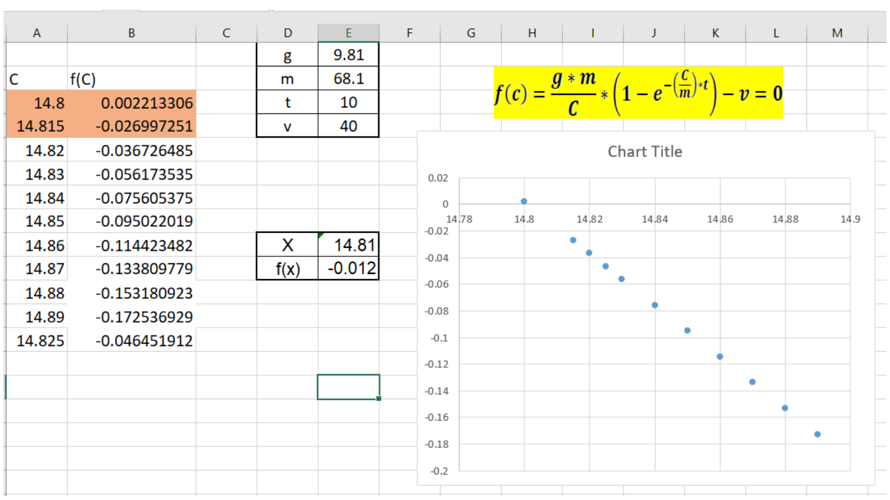
C	f(C)
14	1.611116355
14.1	1.404400051
14.2	1.199315741
14.3	0.995847708
14.4	0.793980405
14.5	0.593698449
14.6	0.394986623
14.7	0.197829873
14.8	0.002213306
14.9	-0.191877813
15	-0.384458061

g	9.81
m	68.1
t	10
v	40

X	14.85
f(x)	-0.095

$$Tolerance\ in\ function\ f(x) = |f(X)_{true} - f(X)_{Numerical\ Solution}| = 0.095$$

Third Trail:



C	f(C)
14.8	0.002213306
14.815	-0.026997251
14.82	-0.036726485
14.83	-0.056173535
14.84	-0.075605375
14.85	-0.095022019
14.86	-0.114423482
14.87	-0.133809779
14.88	-0.153180923
14.89	-0.172536929
14.825	-0.046451912

g	9.81
m	68.1
t	10
v	40

X	14.81
f(x)	-0.012

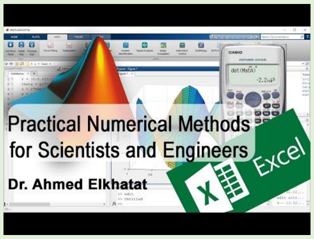
$$Tolerance\ in\ function\ f(x) = |f(X)_{true} - f(X)_{Numerical\ Solution}| = 0.012$$

Practice

Determine the real root of $f(x) = -0.5x^2 + 2.5x + 4.5$ graphically

3. Bisection Method

Watch the online Video



When $f(x)$ is real and continuous in the interval from x_L to x_u, $f(x_L)$ and $f(x_u)$ have opposite signs, i.e. $f(x_L) * f(x_u) < 0$, The real root is then between x_L and x_u, $x_r = \frac{x_L + x_u}{2}$, and thus the $f(x)$ for each new iteration becomes closer to zero.

Steps for the bisection method

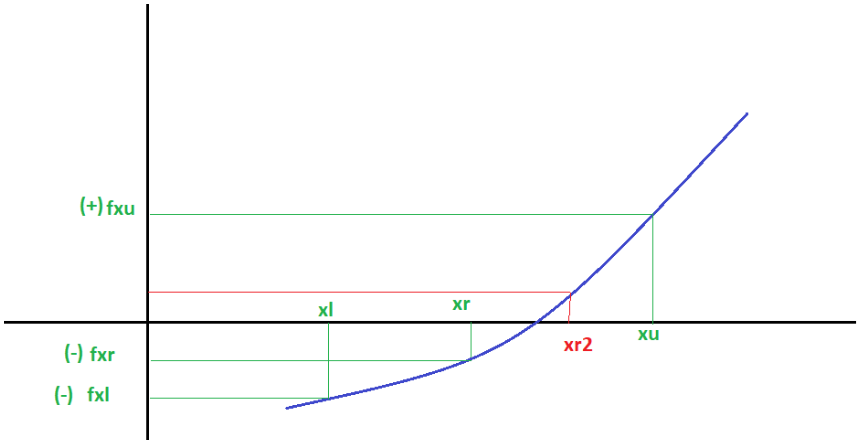
(1) Calculate $f(x)$for x_L , x_u and $x_r = \frac{x_L + x_u}{2}$

(2) Determine in which subinterval that root lies:

a. if $f(x_L) * f(x_r) < 0$, then
   i. $x_r = x_u$
   ii. Calculate $f(x)$for x_L , x_u and $x_r = \frac{x_L + x_u}{2}$

b. if
   i. $f(x_u) * f(x_r) < 0$, [also correct] $f(x_L) * f(x_r) > 0$ then $x_r = x_L$
   ii. Calculate $f(x)$for x_L , x_u and $x_r = \frac{x_L + x_u}{2}$

c. if $f(x_u) * f(x_r) = 0$, then
   i. $x_r = Root$
   ii. Terminate



A) Using MATLAB to find Roots of Single Equation using Bisection method

Write a MATLAB code to find the single equation roots using the Bisection Method.
Use the written codes to find the roots for the following equation

$$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$$

The intervals are 2.5 and 3.5; The Relative error is 0.01; The maximum iteration is 50

Hint: To add the equation as input, it should be written as
@(x)0.9*x^3-5.9*x^2+10.9*x-6

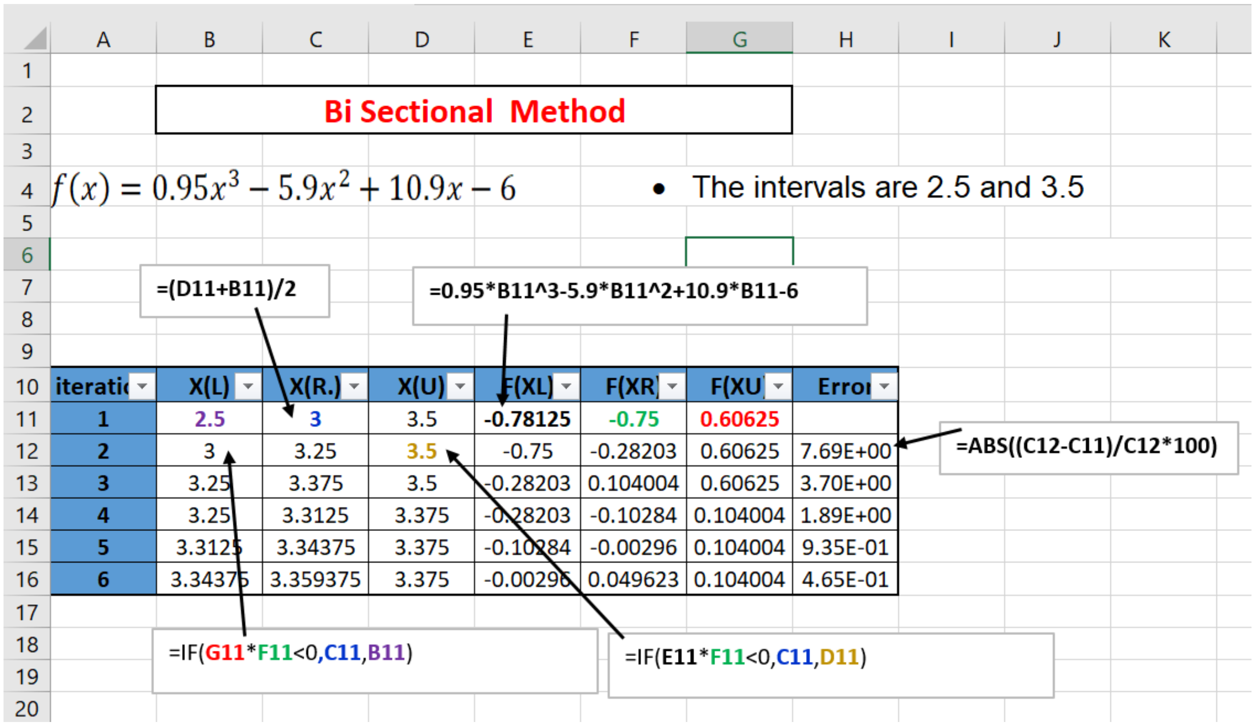
```
disp('These Codes are to find the roots of single equation using Bisection Method, written by Dr. Ahmed Ekhatat')
disp('____________________')
disp (' ')
xl=input('Insert xl: '); % to assign the lower guess value
xu=input('Insert xu: '); % to assign the upper guess value
tolerance=input('Insert the tolerance in function: '); % to assign the relative error
iter=input('Insert the maximum iteration: '); % to assign the maximum iteration
f=input('Insert the equation: '); % to assign the equation
done=0;  % Flag to indicate that the process is not completed
 i=1;
fl=f(xl);

while i<=iter && done==0
    xr=(xu+xl)/2;
    fxr=f(xr);
    if abs(fxr)< tolerance
        fprintf('The estimated root of equation is %f and the tolerance in function is %f in %0.0f iterations\n',xr, tolerance,i)

done=1; % Flag to indicate that the process is completed
    else
        i=i+1;
        if fl*fxr>0
            xl=xr;
            fl=fxr;
        else
            xu=xr;
        end
    end
end
if i>iter
    disp('Method failed')
end
```

#Another way#

```
disp('Bisection method  By Ahmed Elkhatat');
xl=input('Insert the lower guess value: ');
xu=input('Insert the upper guess value: ');
f=input('Insert the equation: ');
fl=f(xl);
fu=f(xu);

% check the signs
if fl*fu>0
error('Initial guess should have different signs: ');
end;
% Iteration
maxerr=input('Insert the maximum relative error: ');
maxit=input('Insert the maximum iteration: ');
iter=0;
xold=0;
while true;
xnew=(xu+xl)/2;
fnew=f(xnew);

%to check the correct sign value
if fnew*fl>0;
 xl=xnew; fl=fnew ;
else xu=xnew; fu=fnew; end
err=(xnew-xold)/xnew*100;
iter=iter+1;
xold=xnew;
if abs(err)<=maxerr | iter>=maxit; break; end
end
disp('-------------------')
disp(' ')
disp(['Iter: ' num2str(iter)])
disp(['x =' num2str(xnew)])
disp(['Error: ' num2str(err)])
```

B) Using Excel to find Roots of Single Equation using Bisection method

Similarly, if the condition can be used in MatLab to check the sign of the f(x) as follows



Bi Sectional Method

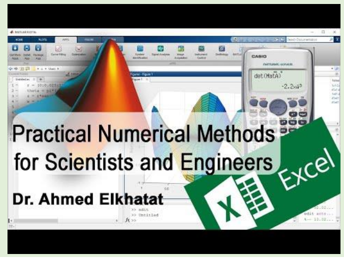
$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$

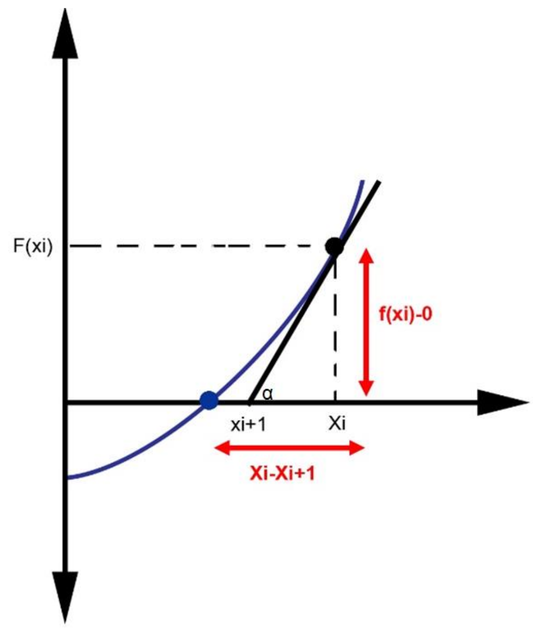
- The intervals are 2.5 and 3.5

iteratio	X(L)	X(R.)	X(U)	F(XL)	F(XR)	F(XU)	Error
1	2.5	3	3.5	-0.78125	-0.75	0.60625	
2	3	3.25	3.5	-0.75	-0.28203	0.60625	7.69E+00
3	3.25	3.375	3.5	-0.28203	0.104004	0.60625	3.70E+00
4	3.25	3.3125	3.375	-0.28203	-0.10284	0.104004	1.89E+00
5	3.3125	3.34375	3.375	-0.10284	-0.00296	0.104004	9.35E-01
6	3.34375	3.359375	3.375	-0.00296	0.049623	0.104004	4.65E-01

4. Newton Raphson Method

Watch the online Video





Newton Method is an open method that uses tangent lines extended from the point $[x_i, f(x_i)]$ to determine an improved estimate of the root.

The Newton-Raphson method can be derived as follows:

$$\tan(\alpha) = f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

Then,

$$\boldsymbol{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}$$

This equation is the iteration equation for the Newton Raphson method, and it is repeated until a solution is reached

Algorithm for the Newton Raphson Method

(1) Inputs:
   a. initial guess of (x)
   b. f(x)
   c. f`(x)
   d. Relative Error
   e. Maximum Number of iteration.
(2) Output:
   a. Improved estimate of (x) or failure message.
(3) Iteration Steps:
   a. for i=1 to iter do steps (b) and (c)

b. set $x = x_0 - \frac{f(x_0)}{f'(x_0)}$
c. if $\left|\frac{x-x_0}{x}\right| < RERR$
d. Output=x
e. Stop

(4) $x_0 = x$
(5) Output Failure Massage

Example:

Write a MATLAB code to find the single equation roots by using Newton Raphson Method.

```
f=input('Insert the equation:  ');
df=input('Insert the derived equation:  ');
xo=input('Insert estimated root:  ');
RERR=input('Insert the relative error:  ');
iter=input('Insert the maximum iteration:  ');

i=1;
while (i<iter);
   x=xo-(f(xo)/df(xo));
   Relerr=abs((x-xo)/x);
   dat(i,:)=[i xo x f(x)];
   if Relerr<=RERR;
      break; end;
   i=i+1;
   xo=x;
end
dat
fprintf('The estimated root of equation is %f and the relative error is %f in %0.0f iterations\n',x,RERR,i)
```

```
New to MATLAB? See resources for Getting Started.
  Insert the equation:  @(x)x^3-2*x^2-5
  Insert the derived equation:  @(x)3*x^2-4*x
  Insert estimated root:  2
  Insert the relative error:  0.1
  Insert the maximum iteration:  5

  dat =

        1.00        2.00        3.25        8.20
        2.00        3.25        2.81        1.41
        3.00        2.81        2.70        0.08      ahmed.elkhatat@qu.edu.qa

  The estimated root of equation is 2.697990 and the relative error is 0.100000 in 3 iterations
f..>>
```

4.1 Advance MatLab Codes

Example 4:

Here we create a bit complicated MatLab script that uses the Newton Raphson method and can derivate the function

Some useful internal functions to be used

- **Diff**: This function can be used to differentiate an equation. For example, write these codes in your MatLab command window

```
>> syms x
>> f=0.9*x^3-5.9*x^2+10.9*x-6
f =(9*x^3)/10 - (59*x^2)/10 + (109*x)/10 - 6
 >> g=diff(f)
         g =(27*x^2)/10 - (59*x)/5 + 109/10
```

- **Subs**: subs(S,OLD,NEW) replaces OLD with NEW in the symbolic expression S. OLD is a symbolic variable, a string representing a variable name, or a string (quoted) expression. NEW is a symbolic or numeric variable or expression.

```
>> subs(f,x,2)
         ans =-3/5
```

- **VPA**: Evaluate each element of the symbolic input x to at least d significant digits.

```
>> f=0.9*x^3-5.9*x^2+10.9*x-6; g=diff(f);

>> subs(f,x,5)

ans = 27/2

>> vpa(ans)

ans =13.5
```

```
clc;
close all;
clear all;
format bank
syms x;
f= input('Enter the Function here eg. x*exp(x):');
g=diff(f);
RERR = input('Enter the relative error:');
x0= input('Enter the initial approximation:');
for i=1:100
    fx=vpa(subs(f,x,x0));
    gx=vpa(subs(g,x,x0));
  x_new=x0-fx/gx;
err=abs((x_new-x0)/x_new);
 dat(i,:)=[i x0 x_new fx];
if err<RERR;
break
end
x0=x_new;
end
disp('i        x0        x_new        f')
disp('_________________________________')
dat
fprintf('The Root is : %f \n',x_new);
fprintf('No. of Iterations : %d\n',i);
```

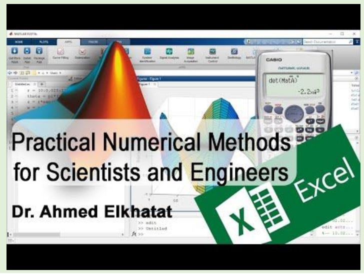
Command Window

```
Enter the Function here eg. x*exp(x):x^3-2*x^2-5
Enter the relative error:0.0001
Enter the initial  approximation:8
i       x0       x_new       f
_________________________________

dat =

[ 1,                                8,                           5.63125,                                  379.0]
[ 2,                          5.63125, 4.1141914071347449890061293451613,                   110.150483642578125]
[ 3, 4.1141914071347449890061293451613, 3.2172401440330869559483362359311,     30.786010822064100898820319497236]
[ 4, 3.2172401440330869559483362359311, 2.7993095662146104285081131674972,     7.5992073967365108654943058099844]
[ 5, 2.7993095662146104285081131674972, 2.6966793909635340411451645022118,     1.2634969063072683611413248282422]
[ 6, 2.6966793909635340411451645022118, 2.6906675180308608375981180635329,   0.066308082662486672447106218243345]
[ 7, 2.6906675180308608375981180635329,  2.6906474482518461741000855216443, 0.00021989262725825977366145513804075]

The Root is : 2.690647
No. of Iterations : 7
>>
```

ahmed.elkhatat@qu.edu.qa

5. Secant Method

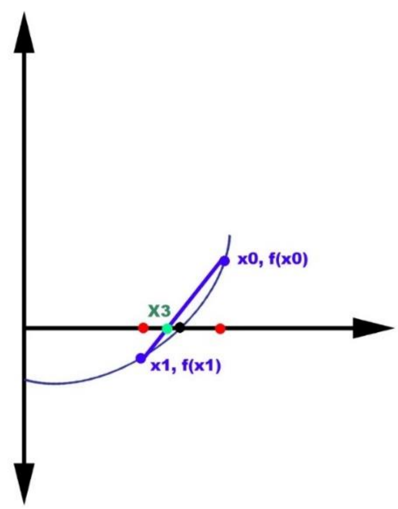
Watch the online Video



The Newtonian method described in the previous section depends on the function derivative. However, it can not be implemented in some functions whose derivatives are complicated. Hence, the Secant method uses the backward finite divided difference instead of the derivative.

It requires two approximations (x_0, X_1), and the connecting line (x_1,$f(x_1)$) and (x_2, $f(x_1)$) will cut x-axis to produce (x_1). by repeating this step, the approximated (x) becomes closer and close, as shown in the following figure.

$$slope = f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$



Use this equation to substitute Newton Raphson

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}}$$

$$\boldsymbol{x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}}$$

This equation is the iteration equation for the Newton Raphson method, and it will be repeated until a solution is reached.

Also, Secant Method requires two initial guesses, but we cannot consider it the bracketing method because it does not require a sign change between the two values.

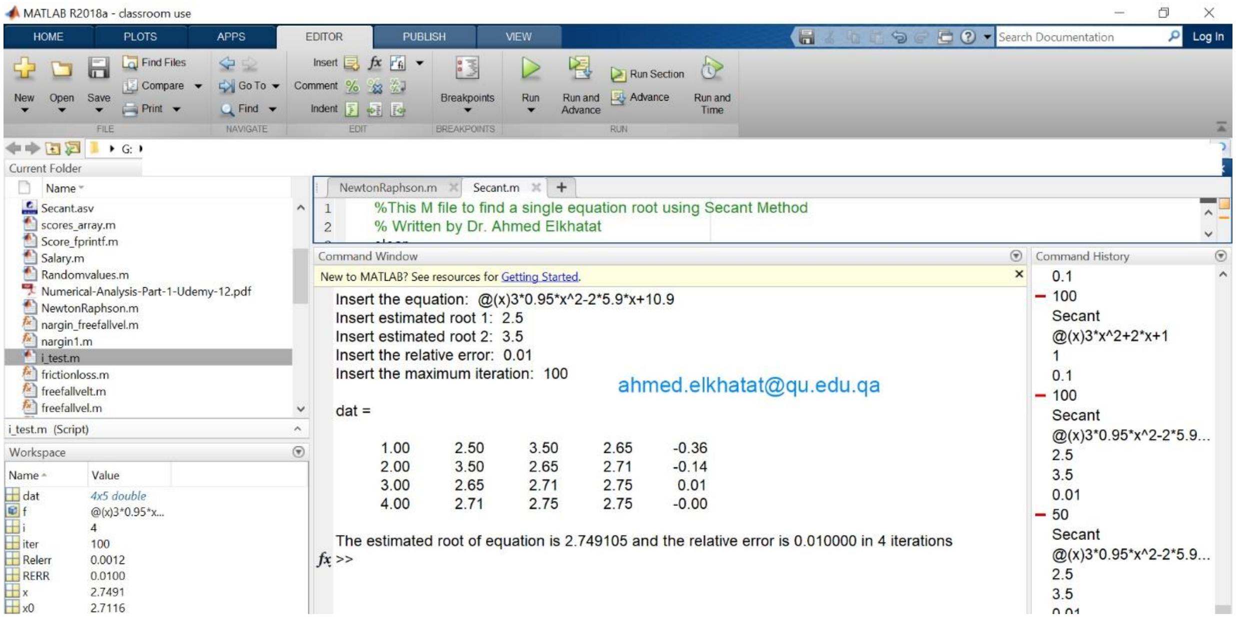
Example:

Write a MATLAB code to find the single equation roots by using Secant Method.

Algorithm for the Secant Method

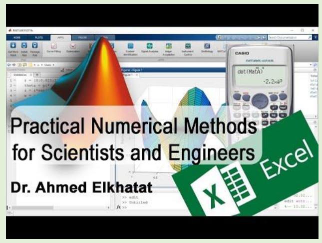
(1) Inputs:
   a. initial guess of (x0) and (x1)
   b. f(x)
   c. Relative Error and Maximum Number of iteration.
(2) Output:
   a. Improved estimate of (x) or failure message.
(3) Iteration Steps:
   a. Set eq0=f(x0) and eq1=f(x1)
   b. for i=1 to iter do steps (b) and (c)
   c. set $x = x_1 - \frac{eq_1(x_0 - x_1)}{(eq_0 - eq_1)}$
   d. if $\left|\frac{x - x_1}{x}\right| < RERR$
   e. Output=x
   f. Stop
(4) $x_0 = x_1;\ eq_0 = eq_1;\ x_1 = x;\ eq_{1=f(x)}$
(5) Output Failure Massage

```
f=input('Insert the equation:  ');
x0=input('Insert estimated root 1:  ');
x1=input('Insert estimated root 2:  ');
RERR=input('Insert the relative error:  ');
iter=input('Insert the maximum iteration:  ');
i=1;
while (i<iter);
   x=x1-f(x1)*(x1-x0)/(f(x1)-f(x0));
   Relerr=abs((x-x1)/x);
   dat(i,:)=[i x0 x1 x f(x)];
   if Relerr<=RERR;
      break; end;
   i=i+1;
   x0=x1; x1=x;
end
dat
fprintf('The estimated root of equation is %f and the relative error is %f in %0.0f iterations\n',x,RERR,i)
```


MATLAB R2018a - classroom use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Log In
Current Folder
NewtonRaphson.m
Secant.m
%This M file to find a single equation root using Secant Method
% Written by Dr. Ahmed Elkhatat
Command Window
New to MATLAB? See resources for Getting Started.
Insert the equation: @(x)3*0.95*x^2-2*5.9*x+10.9
Insert estimated root 1: 2.5
Insert estimated root 2: 3.5
Insert the relative error: 0.01
Insert the maximum iteration: 100
ahmed.elkhatat@qu.edu.qa
dat =
1.00 2.50 3.50 2.65 -0.36
2.00 3.50 2.65 2.71 -0.14
3.00 2.65 2.71 2.75 0.01
4.00 2.71 2.75 2.75 -0.00
The estimated root of equation is 2.749105 and the relative error is 0.010000 in 4 iterations
Command History
Workspace

6. MATLAB built-in functions to find the root of a single equation.

Watch the online Video



fzero and ***roots*** are built-in functions used in MATLAB to find the root of a single equation.

A) fzero Syntax

- fzero(@(x)equation, x0)
  In this syntax, function uses the open method to find the root.
- fzero(@(x)equation, [x0, x1])
  In this syntax, the function uses the bracket method to find the root. So, the two intervals must have different sing solutions. Otherwise, it gives an error message.

  Example: use fzero build-in function to find the roots of the following equation using both open and bracketing methods.

$$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$$

For Open Method: fzero(@(x)0.95*x^3-5.9*x^2+10.9*x-6, 2)

For Bracketing Method: fzero(@(x)0.95*x^3-5.9*x^2+10.9*x-6, [2 4])

```
Command Window
>> fzero(@(x)0.95*x^3-5.9*x^2+10.9*x-6, 2)

ans =

    1.84

>> fzero(@(x)0.95*x^3-5.9*x^2+10.9*x-6, [2 4])

ans =

    3.34

>> fzero(@(x)0.95*x^3-5.9*x^2+10.9*x-6, [0 1])
Error using fzero (line 290)
The function values at the interval endpoints must differ in sign.

fx >>
```


Note that: if the two values are not bracketing, you will receive an error message, as shown above.

If you want to display all details of iteration, use the optimist function as shown in the following syntax.

```
>> Z=optimset ('display', 'iter');
>> fzero(@(x) 0.95*x^3-5.9*x^2+10.9*x-6,2,Z)
```

```
>> Z=optimset ('display', 'iter');
>> fzero(@(x) 0.95*x^3-5.9*x^2+10.9*x-6,2,Z)

Search for an interval around 2 containing a sign change:
 Func-count    a          f(a)             b          f(b)        Procedure
    1               2         -0.2             2          -0.2   initial interval
    3        1.94343     -0.127273       2.05657     -0.274007   search
    5           1.92    -0.0977664         2.08     -0.304794   search
    7        1.88686    -0.0568575      2.11314     -0.348262   search
    9           1.84    -0.0010112         2.16     -0.409229   search
   10        1.77373     0.0729104         2.16     -0.409229   search

Search for a zero in the interval [1.77373, 2.16]:
 Func-count    x          f(x)             Procedure
   10        1.77373     0.0729104        initial
   11        1.83214    0.00809012        interpolation
   12        1.83929  -0.000183972        interpolation
   13        1.83913   7.38887e-07        interpolation
   14        1.83913   6.62777e-11        interpolation
   15        1.83913  -3.55271e-15        interpolation
   16        1.83913             0        interpolation

Zero found in the interval [1.77373, 2.16]

ans =

         1.84
```

B) (roots) Syntax

$$x = [a_3\ a_2\ a_1\ a_0]$$

$$roots(x)$$

Example

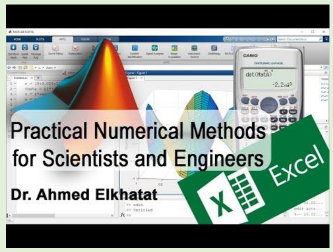
Find the roots of the following equation using the (**roots**) function in MATLAB

$$0.95x^3 - 5.9x^2 + 10.9x - 6$$

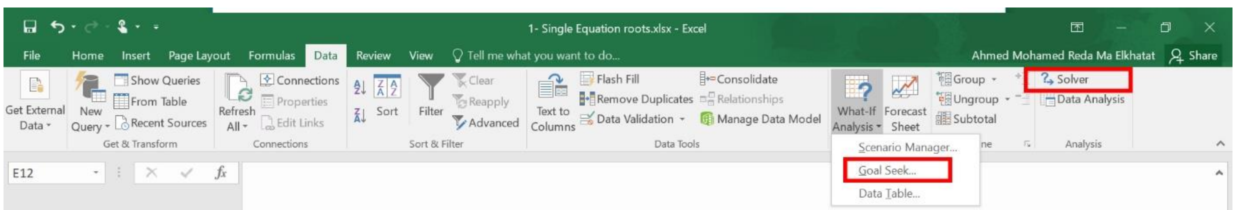
```
Command Window
New to MATLAB? See resources for Getting Started.
>> f=[0.95 -5.9 10.9 -6];
>> roots(f)

ans =

    3.34
    1.84
    1.03

fx >>
```

7. Finding roots of a single equation using Excel Solver and Goal Seek

Watch the online Video

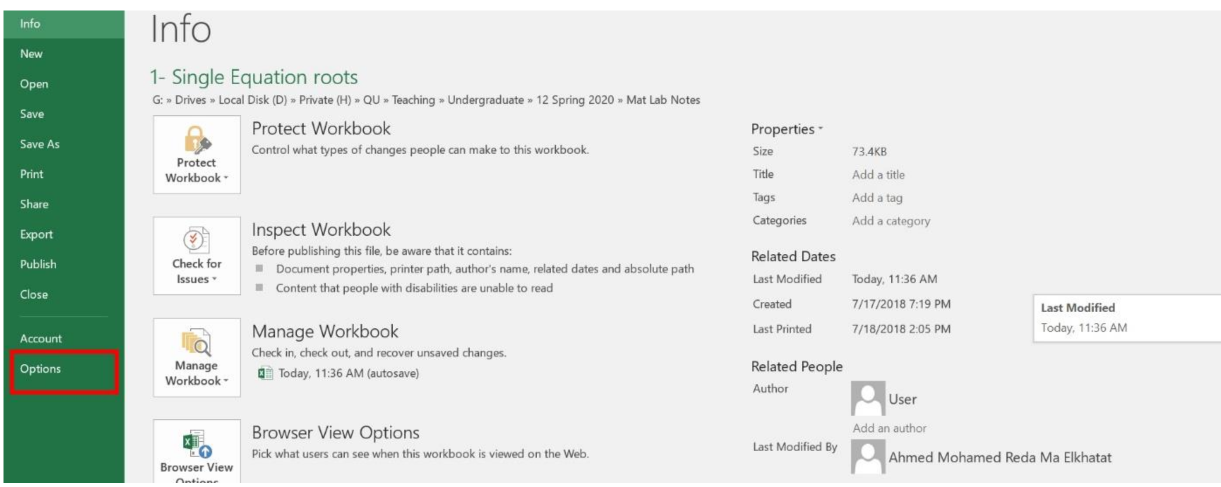


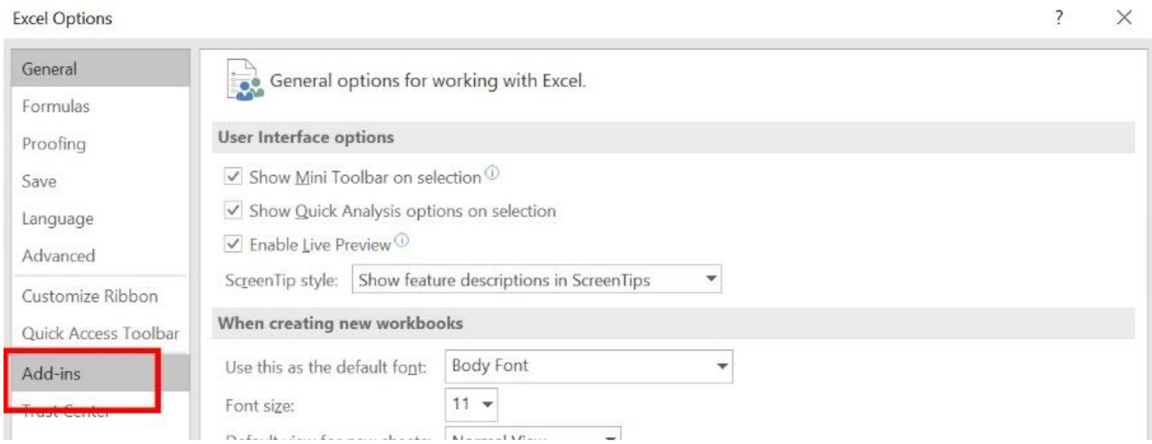
Goal Seek and ***Solver*** are two beneficial functions that can be used in Excel to perform iterations until reaching the solution. They are found in the Data Ribbon of Excel, as shown below.

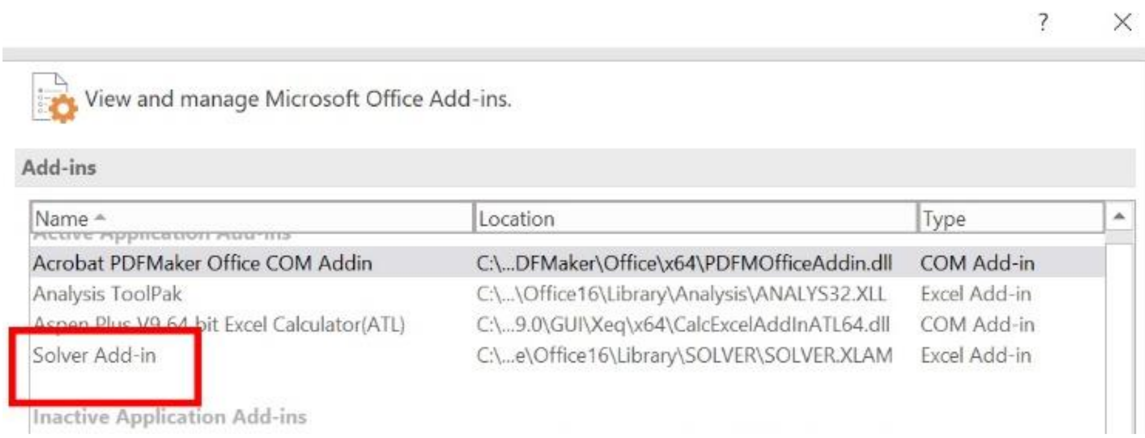


If solver is not installed, you can activated it from

File >> Options >> Add-Ins >> solver Add-In



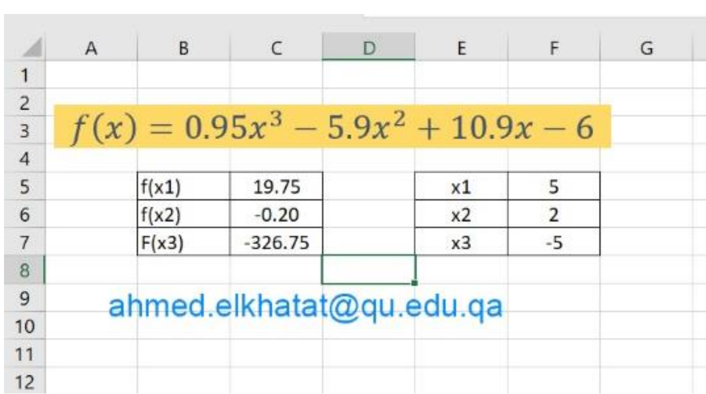


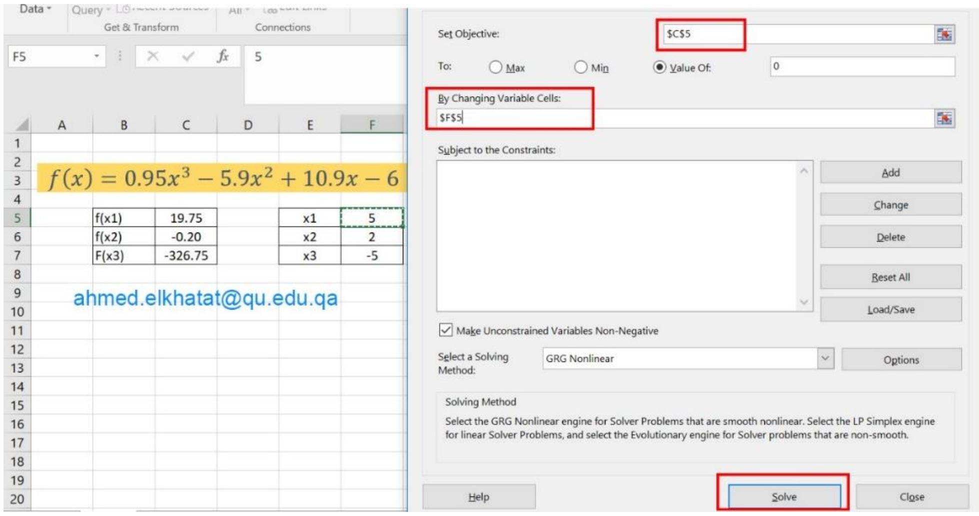


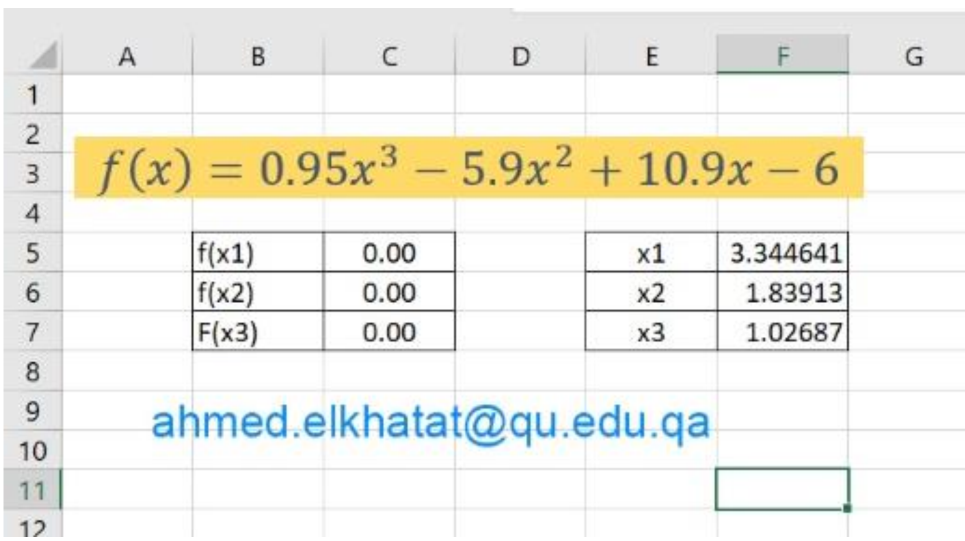
Example

To find the roots of $0.95x^3 - 5.9x^2 + 10.9x - 6$ using Excel Solver,

1) Create multiple cells for (x) and (f(x)) as shown below.
2) Put different initial guess values for (x), thus the f(x) will be calculated automatically according to the related (x) value.
3) use the solver and assign the required value of f(x) to be zero.
4) Repeat the same for f(x2) and f(x3).
5) Thus, the roots of this equation are [1.02, 1.83 and 3.34]

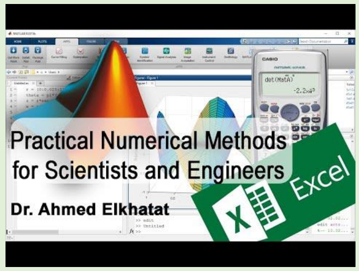



F5
5
$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$
f(x1) 19.75
f(x2) -0.20
F(x3) -326.75
x1 5
x2 2
x3 -5
ahmed.elkhatat@qu.edu.qa
Set Objective:
$C$5
To: Max Min Value Of: 0
By Changing Variable Cells:
$F$5
Subject to the Constraints:
Add
Change
Delete
Reset All
Load/Save
Make Unconstrained Variables Non-Negative
Select a Solving Method:
GRG Nonlinear
Options
Solving Method
Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.
Help
Solve
Close


$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$
f(x1) 0.00
f(x2) 0.00
F(x3) 0.00
x1 3.344641
x2 1.83913
x3 1.02687
ahmed.elkhatat@qu.edu.qa

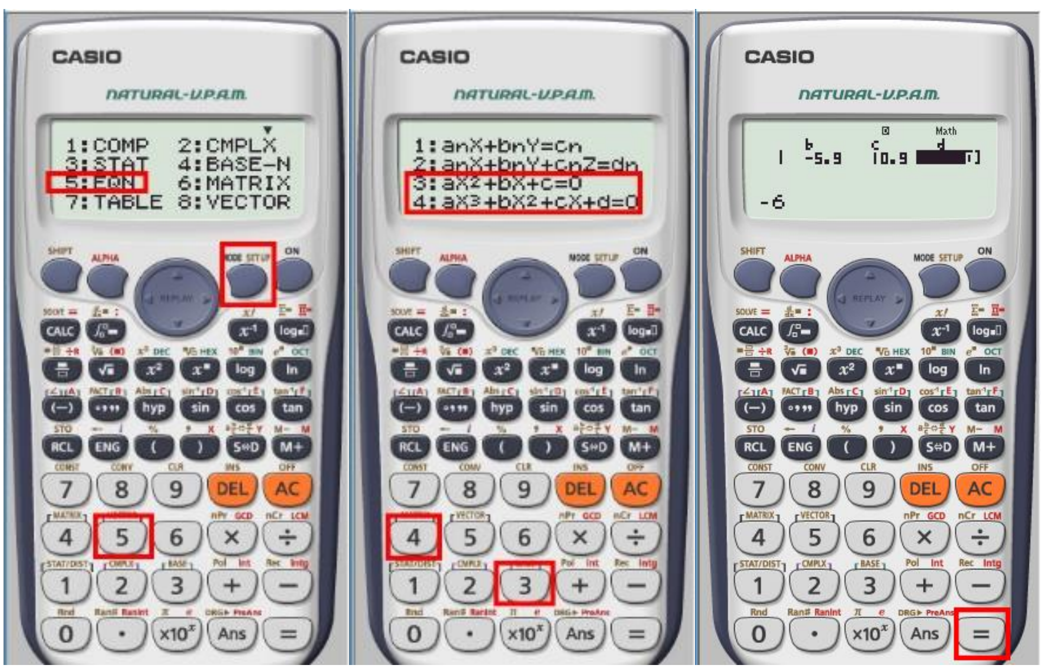
8. Finding roots of a single equation using Calculator (Example: CASIO, fx-991ES PLUS

Watch the online Video



Calculator (Example: CASIO, fx-991ES PLUS can be used to find roots of the single equation as following

$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$



Practice

While operating a spherical tank of radius 7 m to hold 100 m^3 of water in your town, according to the following equation

$$V = \pi h^2 \frac{3R - h}{3}$$

where;

V: water volume in (m^3)

h: water depth in the tank (m)

R: Tank radius (m)

Calculate required water depth to hold the 100 m^3 water using

1) Newton-Raphson Method [MATLAB Code]
2) Secant Method [MATLAB Code]
3) Calculator

Note that by solving the equation using a calculator it gives two roots [20.77 and 2.257] , of course, you will choose the 2.257 m, because 20.77 >D which makes no sense.

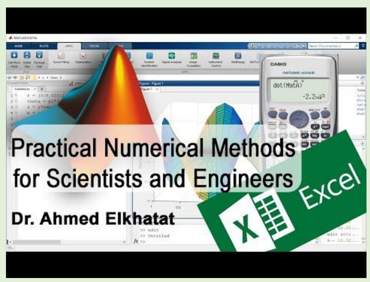
Module (4): Finding Roots of System of Equations using Excel and MATLAB

Part (1): System of Linear Equations

Graphical method, Cramer's method, and elimination methods such as (Gauss, Gauss-Jordon, Gauss-Seidel, and Excel Functions can be used to solve the roots of the system of linear equations.

1. Graphical Method

Watch the online Video



A simple method for estimating the root of $f(x) = 0$ as following

$$a_1 x_1 + a_2 x_2 = b_1$$

$$a_3 x_1 + a_4 x_2 = b_2$$

Both equations can be solved for (x_2)

$$x_2 = -\frac{a_1}{a_2} x_1 + \frac{b_1}{a_2}$$

$$x_2 = -\frac{a_3}{a_4} x_1 + \frac{b_2}{a_4}$$

The values of (x1) and (x2) at the intersection of lines represent the root.

Example 1:

Use the graphical approach to determine the roots of the following linear equations

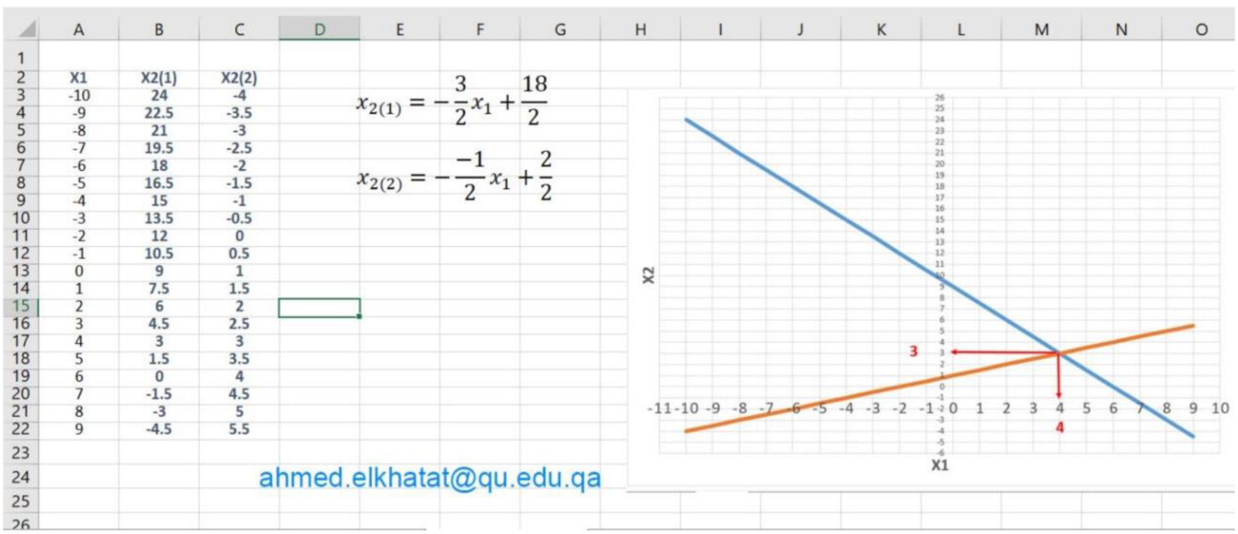
$$3x_1 + 2x_2 - 18 = 0$$

$$-x_1 + 2x_2 - 2 = 0$$

Solve the equations for (x_2)

$$x_{2(1)} = -\frac{3}{2}x_1 + \frac{18}{2}$$

$$x_{2(2)} = -\frac{-1}{2}x_1 + \frac{2}{2}$$



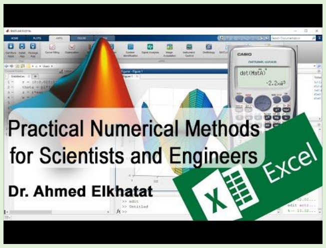
X1	X2(1)	X2(2)
-10	24	-4
-9	22.5	-3.5
-8	21	-3
-7	19.5	-2.5
-6	18	-2
-5	16.5	-1.5
-4	15	-1
-3	13.5	-0.5
-2	12	0
-1	10.5	0.5
0	9	1
1	7.5	1.5
2	6	2
3	4.5	2.5
4	3	3
5	1.5	3.5
6	0	4
7	-1.5	4.5
8	-3	5
9	-4.5	5.5

Note that

a) If the **two lines are parallel**, then there are **no roots** for these equations.
b) If the two lines are coincidental, there is an infinite number of solutions.
c) If the **two lines are very close** to each other, then it causes a problem known as **(ill conditional)**.

2. Cramer's Rule Method

Watch the online Video



Cramer's Rule uses matrices determinants to solve systems of equations provided that having the same number of equations as variables

To Calculate the determent of 2x2 Matrix :

$$A = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \qquad \det(A) = a_1 * b_2 - a_2 * b_1$$

To calculate the determent of 3x3 Matrix:

Method (1)

(i) Assign (+) sign for the first column, (-) sign for the second one and (+) for the third one as shown below.

$$B = \begin{vmatrix} \overbrace{a_1}^{+} & \overbrace{b_1}^{-} & \overbrace{c_1}^{+} \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

(ii) Ignore the first raw, and ignore one column for each product as shown below

$$B = \begin{vmatrix} \overbrace{\cancel{a_1}}^{+} & \overbrace{\cancel{b_1}}^{-} & \overbrace{\cancel{c_1}}^{+} \\ \cancel{a_2} & b_2 & c_2 \\ \cancel{a_3} & b_3 & c_3 \end{vmatrix} \Longrightarrow b_2 * c_3 - b_3 * c_2$$

$$B = \begin{vmatrix} \overbrace{\cancel{a_1}}^{+} & \overbrace{\cancel{b_1}}^{-} & \overbrace{\cancel{c_1}}^{+} \\ a_2 & \cancel{b_2} & c_2 \\ a_3 & \cancel{b_3} & c_3 \end{vmatrix} \Longrightarrow a_2 * c_3 - a_3 * c_2$$

$$B = \begin{vmatrix} \overbrace{\cancel{a_1}}^{+} & \overbrace{\cancel{b_1}}^{-} & \overbrace{\cancel{c_1}}^{+} \\ a_2 & b_2 & \cancel{c_2} \\ a_3 & b_3 & \cancel{c_3} \end{vmatrix} \Longrightarrow a_2 * b_3 - a_3 * b_2$$

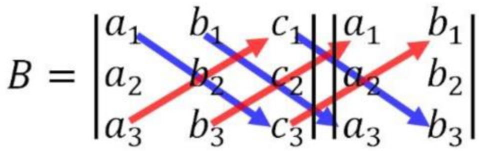
$$\det(B) = [b_2c_3 - b_3c_2] - [a_2c_3 - a_3c_2] + [a_2b_3 - a_3b_2]$$

Method (2)

(i) Matrix B with the first two columns.

$$B = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}$$

(ii) From upper left to lower right: Multiply the entries down the first diagonal. **Add** the result to the product of entries. In addition, from lower left to upper right: **Subtract** the product of entries up the first diagonal.



$$\det(B) = [a_1b_2c_3 + b_1c_2a_3 + c_1a_2b_3] - [a_3b_2c_1 + b_3c_2a_1 + c_3a_2b_1]$$

In the same way, Cramer's method can be used to find the determent of (x, y or z) by replacing that column with the constant column (d). Then, (x, y or z) can be found be dividing D_x/D, D_y/D or D_z/D

Example 2:

Use Cramer's rule method to determine the roots of the following linear equations

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$

$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$

$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

D

$$\begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix} \Longrightarrow D = -2.2 * 10^{-3}$$

Dx_1

$$\begin{vmatrix} -0.01 & 0.52 & 1 \\ 0.67 & 1 & 1.9 \\ -0.44 & 0.3 & 0.5 \end{vmatrix} \Longrightarrow D_{x1} = 0.03278$$

Dx_2

$$\begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix} \Longrightarrow D_{x2} = 0.0649$$

Dx_3

$$\begin{vmatrix} 0.3 & 0.52 & -0.01 \\ 0.5 & 1 & 0.67 \\ 0.1 & 0.3 & -0.44 \end{vmatrix} \Longrightarrow D_{x3} = -0.04356$$

$$x_1 = \frac{Dx_1}{D} = \frac{0.032}{-0.0022} = -14.9$$

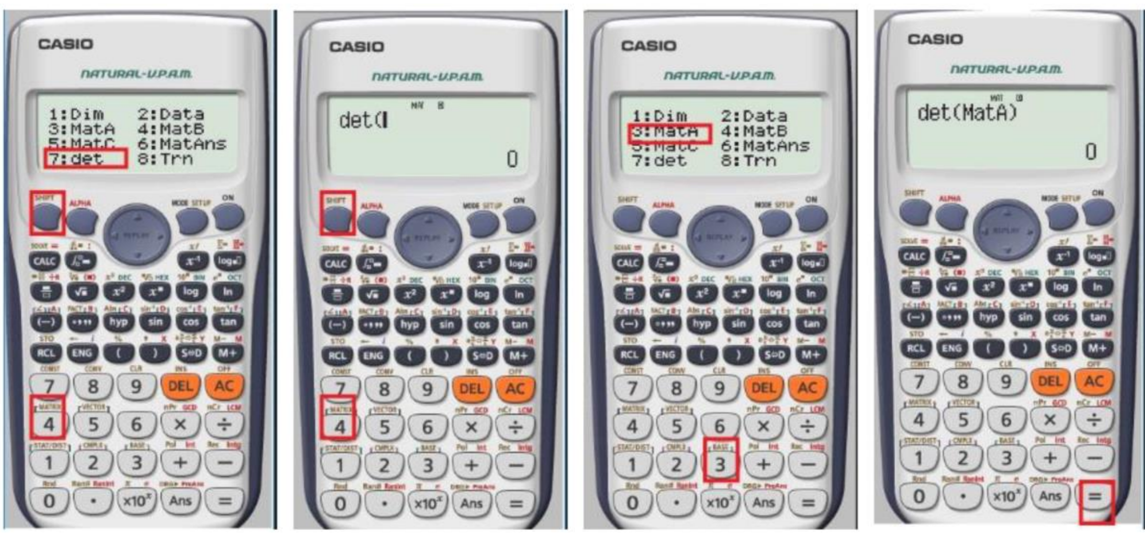
$$x_2 = \frac{Dx_2}{D} = \frac{0.0649}{-0.0022} = -29.5$$

$$x_3 = \frac{Dx_3}{D} = \frac{-0.043}{-0.0022} = 19.8$$

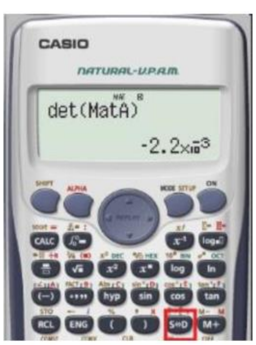
A) Using CASIO (fx-991ES PLUS) to determine the matrix determent

MODE+6 ⟹1:MatA⟹1:3X3⟹Insert data, then AC

Shift+4: 7:det ⟹ Shift+4 ⟹ 3:MatA⟹(=) to show the determent

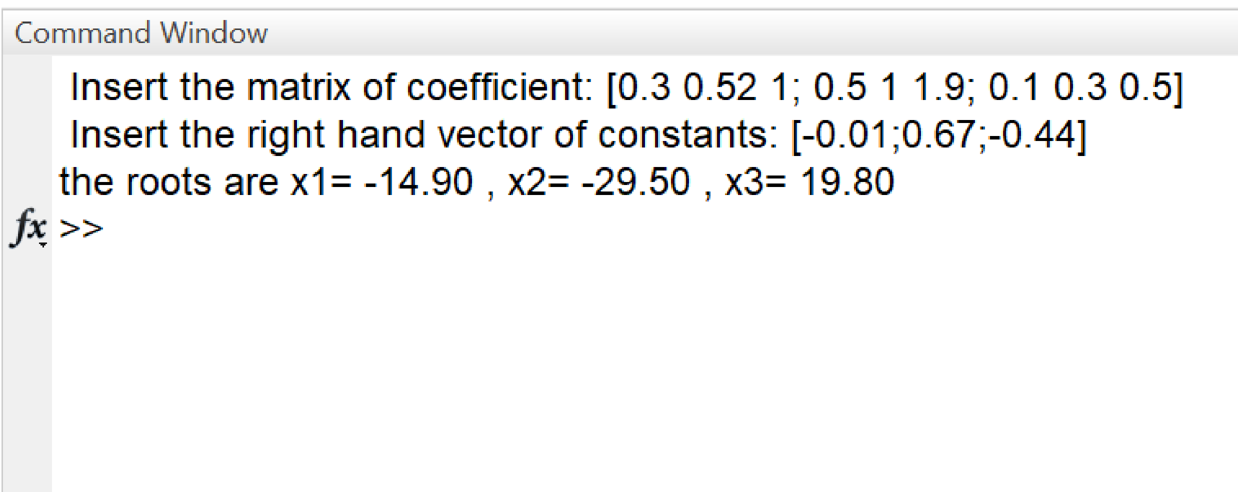


To Change the format $(S \Leftrightarrow D)$



B) Using MatLab to find the roots of system linear equation using Cramer's Rule Method

```
%This Script is for Cramer's Rule By Ahmed Elkhatat
clear; clc;
A=input(' Insert the matrix of coefficient: ');
b=input(' Insert the right hand vector of constants: ');
D=det(A); % to find the determent of (A)
Ax1=A;  % to create another matrix and keep the original one
Ax1(:,1)=b;  % to replace the first column of (A) with constants
Dx1=det(Ax1); % to find the determent of (Ax1)
Ax2=A;
Ax2(:,2)=b;
Dx2=det(Ax2);
Ax3=A;
Ax3(:,3)=b;  %
Dx3=det(Ax3);
x1=Dx1/D; x2=Dx2/D; x3=Dx3/D;
fprintf( 'the roots are x1= %0.2f , x2= %0.2f , x3= %0.2f\n', x1,x2,x3);
```



Using MatLab to find the roots of any number of system of linear equation using Cramer's Rule Method

Cramer2.m

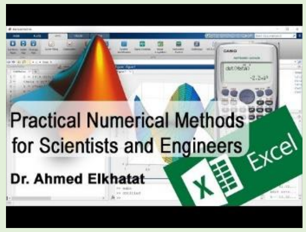
```matlab
%This Script is for Cramer's Rule By Ahmed Elkhatat
clear; clc;
%to insert matrix of coeff.
A=input(' Insert the matrix of coefficient: ');
%check square matrix
[m,n]=size(A);
if m~=n, error('Matrix should be square'); end;
%to insert vector of constants.
b=input(' Insert the right hand vector of constants: ');

% to find the determent of (A)
D=det(A);
% to find determents (x)
for i=1:n;
    AX=A; % to create another matrix and keep the original one
    AX(:,i)=b; % to replace the column of (A) with constants
    DX=det(AX); % to find the determent of (AX)
    X=DX/D;
    Results(i,:)=[i DX X];
end
disp('____________________________________')
disp('               i          D         X')
disp('____________________________________')
Results
```

Command Window

```
Insert the matrix of coefficient: [0.3 0.52 1; 0.5 1 1.9; 0.1 0.3 0.5]
Insert the right hand vector of constants: [-0.01 0.67 -0.44]
____________________________________
             i          D         X
____________________________________

Results =

    1.00      0.03    -14.90
    2.00      0.06    -29.50
    3.00     -0.04     19.80

fx >>
```

3. Naive Gauss Elimination Method

Watch the online Video



The primary strategy of Naive Gauss elimination is a combination of two steps; the first step is to eliminate the unknowns by multiplying the equations by a constant, then by combining the two equations, the result is one equation with one unknown, a step known as (**Forward elimination**). This step is followed by another step known as (**Back Substitution**) in which the equation is solved, and the result is back-substituted into the previous equation until all (x)s are found.

Example:

$$A)\xRightarrow{Pivot\ (Ref)} a_{11}x_1 + a_{12}x_2 = b_1$$

$$B)\xRightarrow{Elimenated\ (target)} a_{21}x_1 + a_{22}x_2 = b_2$$

Suppose that we want to eliminate the second equation to get $a_{21} = 0$ in order to get $a_{22}x_2 = b_2$

$$Pivot\ Coef.* \left(-\frac{\$\$(a)of\ eleiminated\ Coef.}{\$\$(a)of\ pivot\ \ Coef.}\right) + eliminated\ Coef.$$

I.e.

$$A * \left(-\frac{\$\$a_{21}}{\$\$a_{11}}\right) + B$$

so

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$\left(a_{11} * \left\{-\frac{\$\$a_{21}}{a_{11}}\right\} + a_{21}\right)x_1 + \left(a_{12} * \left\{-\frac{\$\$a_{21}}{a_{11}}\right\} + a_{22}\right)x_2 = b_2 * \left\{-\frac{\$\$a_{21}}{a_{11}}\right\} + b_1$$

This leads to

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$0 + a`_{22}x_2 = b_2'$$

Where $a`_{22} = \left(a_{12} * \left\{-\frac{\$\$a_{21}}{a_{11}}\right\} + a_{22}\right) x_2$

For more than two equations, the Forward elimination will be in more than one step as following

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \xrightarrow{FE(1)} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \begin{Bmatrix} b_1 \\ b'_2 \\ b'_3 \end{Bmatrix} \xrightarrow{FE(2)} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \begin{Bmatrix} b_1 \\ b'_2 \\ b''_3 \end{Bmatrix}$$

Practice

Using Naive Gauss elimination, find the roots of the following equation

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$

$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$

$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

A) Using MatLab to find the roots of system linear equation using Naive Gauss elimination Method

```
%Gauss Naive by Ahmed Elkhatat
A=input('Insert the Coefficent Matrix:  ');
b=input('insert the right hand vector: ');

%to check the matrix size
[m,n]= size(A);
if m~=n, error('Matrix A must be square'); end

%to combine A and b in one Matrix
Ab=[A b];

nd=n+1; %to include the right hand vector

% to proceed with the forward elimination, we need to extract the reference coef. and target coef. from the matrix (pivot coeff is numerator, and target is denominator)

for k=1:n-1 % n-1 because we will apply the elimination until the x that is before the last x in A matrix
   for i=k+1:n %k+1, because we will use it  to extract from the target eq. below the ref. eq.

%to proceed with the forward elimination

       factor=-1*Ab(i,k)/Ab(k,k);
       Ab(i,k:nd)=Ab(k,k:nd)*factor+Ab(i,k:nd);
   end
end

%to proceed with the back substitution
x=ones(n,1);  % to identify x that will be used in the loop, zeros(n,1) works as well.
x(n)=Ab(n,nd)/Ab(n,n); % to find the coef of the largest (x) in the matrix

% in order to find the other x (i.e. n-1)
for i=n-1:-1:1;   %because we want to go back from x(n-1) to x1 step by step
     x(i)=(Ab(i,nd)-Ab(i, i+1:n)*x(i+1:n))/Ab(i,i);
end
disp(x)
```

To find the roots of these equations

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$

$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$

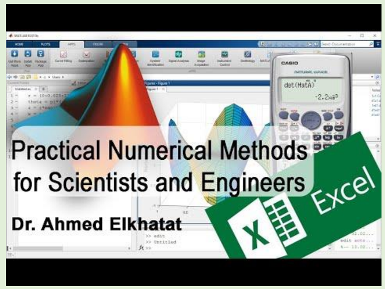
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

Command Window

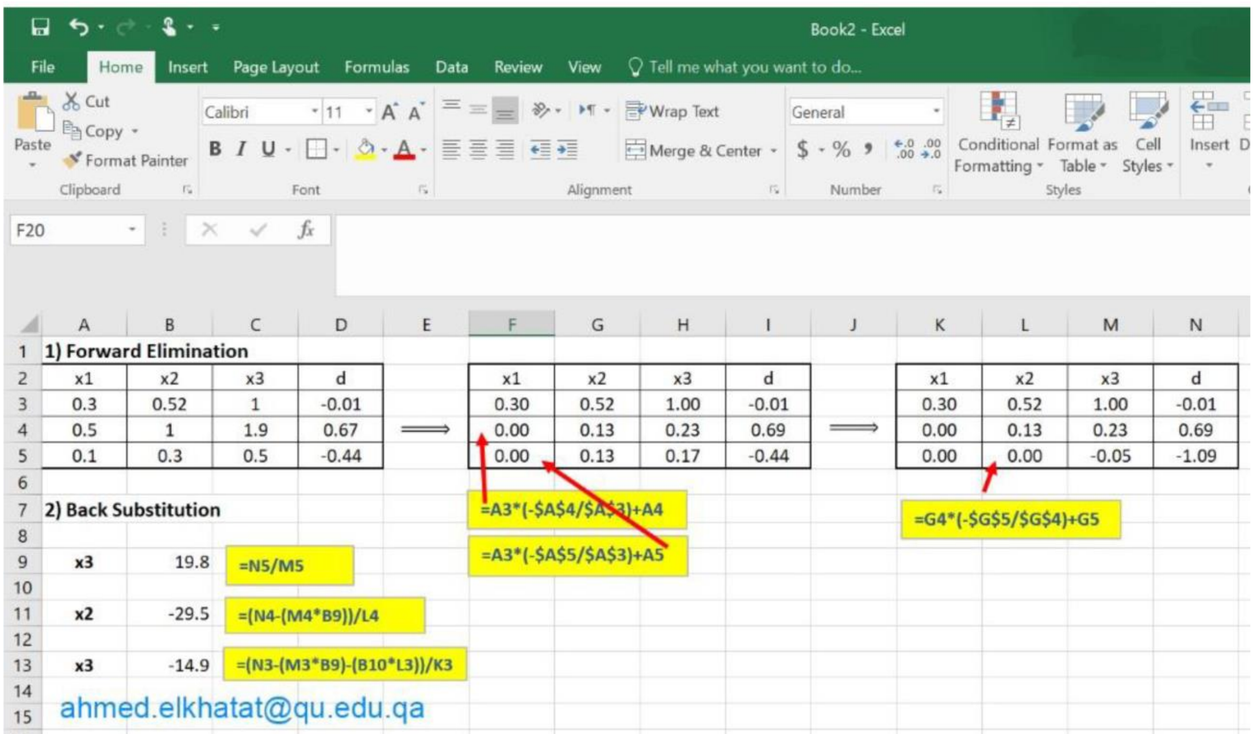
```
>> GaussNaive
Insert the Coefficent Matrix:  [0.3 0.52 1; 0.5 1 1.9; 0.1 0.3 0.5]
insert the right hand vector: [-0.01;0.67;-0.44]
      -14.90
      -29.50
       19.80

fx >>
```

B) Using Excel to find the roots of system linear equation using Naive Gauss elimination Method

Watch the online Video



Excel also can be used to perform Naive Gauss elimination as following



1) Forward Elimination

x1	x2	x3	d
0.3	0.52	1	-0.01
0.5	1	1.9	0.67
0.1	0.3	0.5	-0.44

⟹

x1	x2	x3	d
0.30	0.52	1.00	-0.01
0.00	0.13	0.23	0.69
0.00	0.13	0.17	-0.44

=A3*(-$A$4/$A$3)+A4

=A3*(-$A$5/$A$3)+A5

⟹

x1	x2	x3	d
0.30	0.52	1.00	-0.01
0.00	0.13	0.23	0.69
0.00	0.00	-0.05	-1.09

=G4*(-$G$5/$G$4)+G5

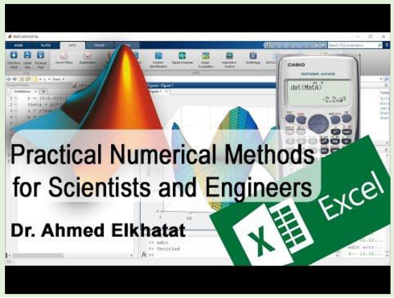
2) Back Substitution

x3	19.8	=N5/M5
x2	-29.5	=(N4-(M4*B9))/L4
x3	-14.9	=(N3-(M3*B9)-(B10*L3))/K3

ahmed.elkhatat@qu.edu.qa

4. LU Factorization Method

Watch the online Video



In this method, the coefficient matrix [A] is a multiplication result of its upper matrix [U] and Lower matrix [L] i.e. [A]=[U]*[L]. Since [A]*{x}={b}, then [U]*[L]*{x}={b}.Recall that [U]*{x}={D}, then [L]*{D}={b}, so {D} can be computed. Last step is that since [U]*{x}={D}, then {x} can be computed as well

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

(1) Find the Upper and Lower Matrices of (A)

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}}_{A} = \left[\underbrace{\begin{matrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{matrix}}_{U}\right] * \left[\underbrace{\begin{matrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{matrix}}_{L}\right]$$

(2) Compute (D) vector

$$\left[\underbrace{\begin{matrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{matrix}}_{L}\right] * \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

(3) Compute (x) vector

$$\left[\underbrace{\begin{matrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{matrix}}_{U}\right] * \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix}$$

Example

Find the roots of these equations using LU factorization

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$

$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$

$$0.3x_1 - 0.2x_2 + 10x_3 = 71$$

$$\underbrace{\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}}_{A} \longrightarrow \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7 & -0.29 \\ 0 & -1.9 & 10.2 \end{bmatrix} \longrightarrow \underbrace{\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7 & -0.29 \\ 0 & 0 & 10.01 \end{bmatrix}}_{U}$$

Since

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix}}_{L} * \underbrace{\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7 & -0.29 \\ 0 & 0 & 10.01 \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}}_{A}$$

Then

$$L_{21} * 3 + 1 * 0 + 0 * 0 = 0.1 \longrightarrow L_{21} = 0.033$$

$$L_{31} * 3 + L_{32} * 0 + 1 * 0 = 0.1 \longrightarrow L_{31} = 0.1$$

$$L_{31} * -0.1 + L_{32} * 7 + 1 * 0 = -0.2 \longrightarrow L_{31} = -0.027$$

Thus

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0.033 & 1 & 0 \\ 0.1 & -0.027 & 1 \end{bmatrix}}_{L}$$

Since

$$\begin{bmatrix} \underbrace{\begin{matrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{matrix}}_{L} \end{bmatrix} * \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

Then

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0.033 & 1 & 0 \\ 0.1 & -0.027 & 1 \end{bmatrix}}_{L} * \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.3 \\ 71 \end{Bmatrix}$$

$$\begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.56 \\ 69.68 \end{Bmatrix}$$

To compute (x) vector

$$\underbrace{\begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}}_{U} * \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix}$$

$$\underbrace{\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0 & 7 & -0.29 \\ 0 & 0 & 10.01 \end{bmatrix}}_{U} * \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 7.85 \\ -19.56 \\ 69.68 \end{Bmatrix}$$

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2.99 \\ -2.5 \\ 6.9 \end{Bmatrix}$$

C) Using MatLab to determine the roots of a system of linear Equation Using LU Factorization Method

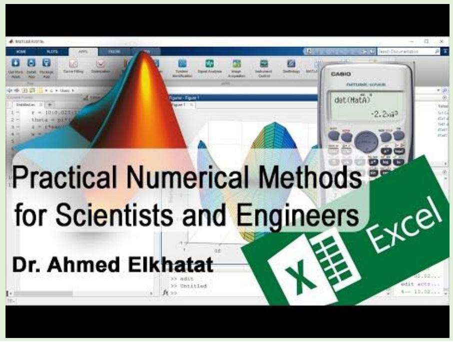
Thanks to (lu) function in MatLab, LU factorization method can be executed quickly with short codes as follows:

```
disp('LU factorization by Ahmed Elkhatat')
disp('_____________________________')
A=input('Insert the Coefficent Matrix:  ');
b=input('insert the right hand vector as vertical array: ');

[L, U]= lu(A);   % to compute the Lower and Upper matrices
D=L\b;  % to compute D array
x=U\D
```

```
Command Window
>> LU_factorization
LU factorization by Ahmed Elkhatat
_____________________________
Insert the Coefficent Matrix:  [0.3 0.52 1; 0.5 1 1.9; 0.1 0.3 0.5]
insert the right hand vector as vertical array: [-0.01;0.67;-0.44]

x =

      -14.90
      -29.50
       19.80

>>
```

5. Gauss-Seidel Method

Watch the online Video



Gauss-Seidel is the most common iterative method used to find roots of a system if linear equations. If the equations are diagonally non-zero (preferably diagonally predominant), then the first equation can be solved for (x_1) and the second equation can be solved for (x_2), and the third equation can be solved for the (x_3) and so on. To start the iteration, all x's are zeros. These zeros are substituted to solve (x1), then this (x1) is used to solve (x_2), then (x_1 and x_2) are used to solve the (x_3), and so on.

To illustrate this

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}}$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

Example

Find the roots of these equations using Gauss-Seidel

$$2x_1 + 3x_2 + 4x_3 = 13$$

$$10x_1 + 5x_2 + \ x_3 = 28$$

$$-3x_1 + 10x_2 + 2x_3 = -10$$

Note that these equations are not diagonally predominant. In order to get them diagonally dominant, equation two should come first, and equation 3 come next.

$$10x_1 + 5x_2 + x_3 = 28$$

$$-3x_1 + 10x_2 + 2x_3 = -10$$

$$2x_1 + 3x_2 + 4x_3 = 13$$

$$x_1 = \frac{28 - 5x_2 - x_3}{10}$$

$$x_2 = \frac{-10 + 3x_1 - 2x_3}{10}$$

$$x_3 = \frac{13 - 2x_1 - 3x_2}{4}$$

Iteration

Iter	x_1	x_2	x_3
1	0	0	0
2	$x_1 = \frac{28 - 5(0) - (0)}{10} = 2.8$	$x_2 = \frac{-10 + 3(2.8) - 0}{10} = -0.16$	$x_3 = \frac{13 - 2(2.8) - 3(-0.16)}{4} = 1.97$

	x_1	x_2	x_3
1	0	0	0
2	2.8	-0.16	1.97
3	2.683	-0.5891	2.350
4	2.860	-0.665	2.350
5	2.898	-0.601	2.252
6	2.875	-0.581	2.252
7	2.865	-0.591	2.260
8	2.869	-0.591	2.259

$\%x_1$	$\%x_2$	$\%x_3$
#	#	#
100	100	100
-4.361	72.840	16.182
6.173	11.436	0
1.313	-10.713	-4.374
-0.777	-3.390	0
-0.344	1.634	0.379
0.138	0.088	-0.071

$$Relative\ Error\% = \left|\frac{x_{n+1} - x_n}{x_{n+1}}\right| * 100$$

A) Using MatLab to determine the roots of linear Equations using Gauss-Seidel.

In order to compute Gauss-Seidel in MatLab, modifications in the equation should be done first

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \longrightarrow \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3$$

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \longrightarrow \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \longrightarrow \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{33}}x_1 - \frac{a_{32}}{a_{33}}x_2$$

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}}_{x_{n+1}} = \underbrace{\begin{Bmatrix} \left(\frac{b_1}{a_{11}}\right) \\ \left(\frac{b_2}{a_{22}}\right) \\ \left(\frac{b_3}{a_{33}}\right) \end{Bmatrix}}_{d} - \underbrace{\begin{bmatrix} 0 & \left(\frac{a_{12}}{a_{11}}\right) & \left(\frac{a_{13}}{a_{11}}\right) \\ \left(\frac{a_{21}}{a_{22}}\right) & 0 & \left(\frac{a_{23}}{a_{22}}\right) \\ \left(\frac{a_{31}}{a_{33}}\right) & \left(\frac{a_{32}}{a_{33}}\right) & 0 \end{bmatrix}}_{C} * \underbrace{\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}}_{x_n}$$

i.e.

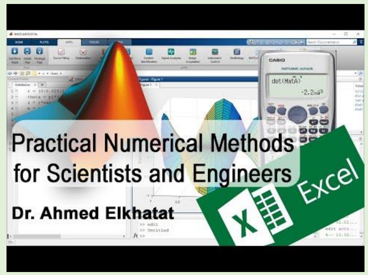
$$\{\boldsymbol{x_{n+1}}\} = \{\boldsymbol{d}\} - |\boldsymbol{C}| * \{\boldsymbol{x_n}\}$$

```
disp("This is written by Ahmed Elkhatat")
A=input('Insert the coefficient matrix: ');
b=input('Insert the right hand sided vector: ');
ERR=input('Insert the relative error: ');
Iter=input('Insert the maximum iteration: ');
%% TO CHECK THE MATRIX
 %to check whether the matrix is square or not.
[m,n]=size(A);
if m~=n, error('Matrix (A) must be square'); end;

%to  check if the matrix A is diagonally dominant
for i = 1:n
    j = 1:n;    % to define an array of the coefficients' elements
    j(i) = [];   % to eliminate the unknown's coefficient from the remaining coefficients
    B = abs(A(i,j));
    C=abs(A(i,i));
    if  C<(max(B))
        fprintf('The matrix is not strictly diagonally dominant at row %2i\n\n',i)
        K=input('If you want to stop press (0): ');
        if K==0 ; return; else continue; end

    end
end
%% To Assign Both (d) and (C)

C=A; % to create another copy of Matrix A and keep the original one
x=zeros(n,1);
ERROR=ones(n,1);
for i=1:n     % to convert the diagonal of matrix C to zero
    C(i,i)=0;
end

for i=1:n   % to get the final form of matrix C
    C(i, 1:n)=C(i,1:n)/A(i,i);
end

for i=1:n     % to get vector d
    d(i)=b(i)/A(i,i);
end
%% To Start the Iteration
iteration=1;
while (1)
    xold=x;
    Tab(iteration,:)=[iteration x' ERROR'];
    for i=1:n
        x(i)=d(i)-C(i,:)*x;
        if x(i)~=0 ERROR(i)=abs((x(i)-xold(i))/x(i))*100; end
    end
     iteration=iteration+1;
      if max(ERROR)<=ERR | iteration>=Iter, break, end;
end
disp("-------------------------------------------------------------------------------")
disp("     #iter       x1        x2       x3   ERROR(x1)    ERROR(x2)    ERROR(3)")
Tab
```

```
Command Window
>> GaussSeidel2022U
This is written by Ahmed Elkhatat
Insert the coefficient matrix: [10 5 1; -3 10 2; 2 3 4]
Insert the right hand sided vector: [28,-10,18]
Insert the relative error: 0.0001
Insert the maximum iteration: 200
-----------------------------------------------------------------------------
     #iter       x1         x2        x3    ERROR(x1)     ERROR(x2)     ERROR(3)

Tab =

    1.0000         0         0         0    1.0000    1.0000    1.0000
    2.0000    2.8000   -0.1600    3.2200  100.0000  100.0000  100.0000
    3.0000    2.5580   -0.8766    3.8784    9.4605   81.7477   16.9771
    4.0000    2.8505   -0.9206    3.7652   10.2599    4.7747    3.0081
    5.0000    2.8838   -0.8879    3.7241    1.1548    3.6764    1.1045
    6.0000    2.8715   -0.8833    3.7267    0.4251    0.5167    0.0719
    7.0000    2.8690   -0.8846    3.7290    0.0889    0.1471    0.0604
    8.0000    2.8694   -0.8850    3.7290    0.0148    0.0364    0.0008
    9.0000    2.8696   -0.8849    3.7289    0.0055    0.0047    0.0030
   10.0000    2.8696   -0.8849    3.7289    0.0003    0.0022    0.0003
   11.0000    2.8696   -0.8849    3.7289    0.0003    0.0001    0.0001
   12.0000    2.8696   -0.8849    3.7289    0.0000    0.0001    0.0000
fx
```

6. Computing the roots using (MatLab Left Division)

Watch the online Video



$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

This system can be expressed as

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} * \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

i.e

$$|A| * \{x\} = \{b\}$$

So,

$$\{x\} = \{b\}/|A|$$

<u>in order to compute it using MatLab</u>

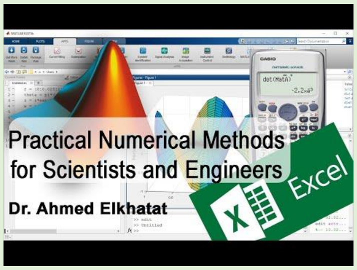
$x = A\backslash b$ or $x = inv(A) * b$

Command Window

```
>> A=[0.3 0.52 1; 0.5 1 1.9; 0.1 0.3 0.5];
b=[-0.01 0.67 -0.44]';
x1=A\b
x2=inv(A)*b

x1 =

  -14.90
  -29.50
   19.80


x2 =

  -14.90
  -29.50
   19.80
```

7. Computing the roots using (Excel Functions)

Watch the online Video



$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} * \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

i.e

$$|A| * \{x\} = \{b\}$$

So,

$$\{x\} = \{b\}/|A|$$

$$\{x\} = \{b\} * inv|A|$$

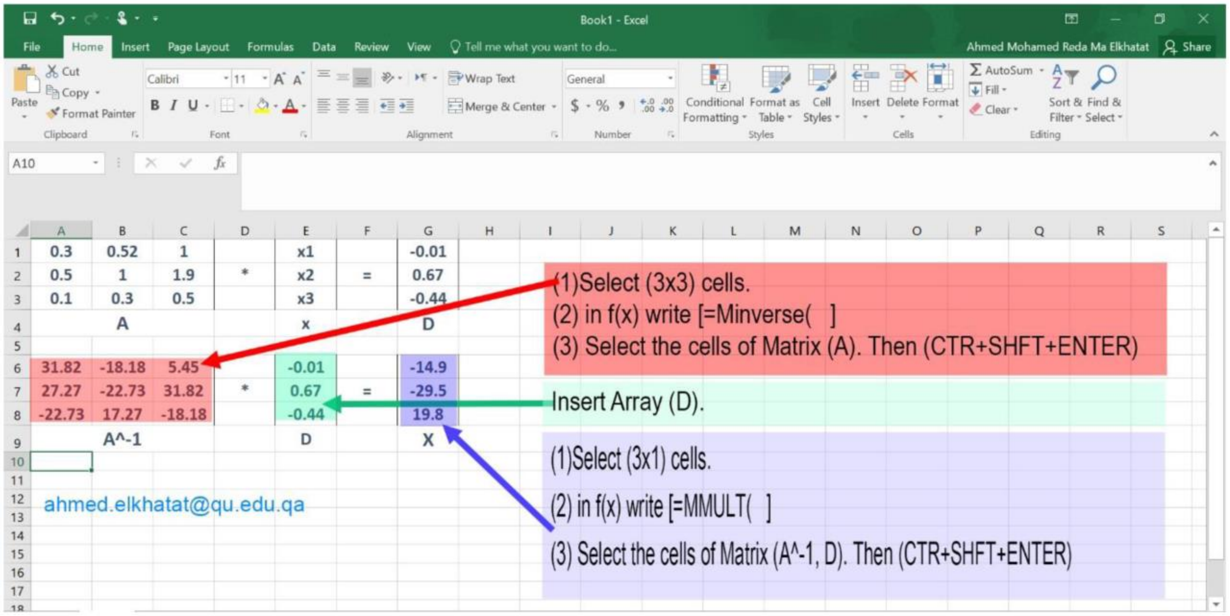
To invert a matrix using Excel (**Minverse**) function is used, and to multiply two matrices in Excel (**MMULTI**) function is used as follows.

To find the roots of these equations

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$

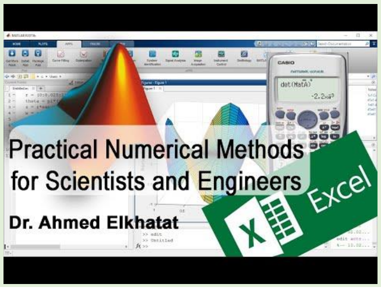
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$

$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$


Book1 - Excel
(1)Select (3x3) cells.
(2) in f(x) write [=Minverse( ]
(3) Select the cells of Matrix (A). Then (CTR+SHFT+ENTER)
Insert Array (D).
(1)Select (3x1) cells.
(2) in f(x) write [=MMULT( ]
(3) Select the cells of Matrix (A^-1, D). Then (CTR+SHFT+ENTER)
ahmed.elkhatat@qu.edu.qa
0.3 0.52 1 * x1 = -0.01
0.5 1 1.9 x2 0.67
0.1 0.3 0.5 x3 -0.44
A x D
31.82 -18.18 5.45 * -0.01 = -14.9
27.27 -22.73 31.82 0.67 -29.5
-22.73 17.27 -18.18 -0.44 19.8
A^-1 D X

8. Computing the roots using (CASIO fx-991ES PLUS) Calculator

Watch the online Video



To find the roots of these equations

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$

$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$

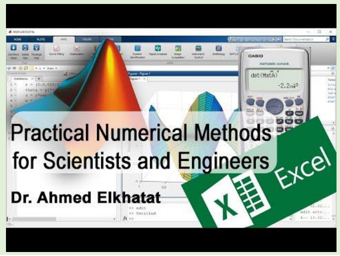
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

MODE >> EQN>> anX+bnY+bnZ=dn


CASIO
NATURAL-V.P.A.M.
1:COMP 2:CMPLX
3:STAT 4:BASE-N
5:EQN 6:MATRIX
7:TABLE 8:VECTOR
1:anX+bnY=cn
2:anX+bnY+cnZ=dn
3:aX2+bX+c=0
4:aX3+bX2+cX+d=0
-0.44
X=
-14.9
Y=
-29.5
Z=
19.8

Part (2): System of Nonlinear Equations

Watch the online Video



In contrast to linear equations, nonlinear equations are curves, and their solution is the intersection of these curves.

Different methods can solve the system of nonlinear equations, but here we will focus on **Newton Raphson** and **Excel Solver** Methods.

1. Computing the Nonlinear roots using Newton Raphson Method

For a single nonlinear equation, Newton Raphson Law is

$$x_{n+1} = x_n - \frac{fx}{f`x}$$

Similarly, a system of nonlinear equations, Newton Raphson Law is

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} - \frac{\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}}{\begin{bmatrix} \left({}^{df_1}/_{dx_1}\right) & \left({}^{df_1}/_{dx_2}\right) \\ \left({}^{df_2}/_{dx_1}\right) & \left({}^{df_2}/_{dx_2}\right) \end{bmatrix}_{(n)}}$$

$$\begin{bmatrix} \left({}^{df_1}/_{dx_1}\right) & \left({}^{df_1}/_{dx_2}\right) \\ \left({}^{df_2}/_{dx_1}\right) & \left({}^{df_2}/_{dx_2}\right) \end{bmatrix}_{(n)} \quad is\ known\ as\ Jacobian$$

Hence,

$$x_{n+1} = x_n - \frac{f(x_n)}{J_n}$$

$$x_{n+1} = x_n - {J_n}^{-1} * f(x_n)$$

$$x_{n+1} = x_n - J_n \backslash f(x_n) \dots \dots .\{in\ MatLab\}$$

A) Manually by Jacobian.

The equation can be rearranged to be

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} - \frac{\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}}{\begin{bmatrix} \left({df_1}/{dx_1} \right) & \left({df_1}/{dx_2} \right) \\ \left({df_2}/{dx_1} \right) & \left({df_2}/{dx_2} \right) \end{bmatrix}_{(n)}}$$

$$\begin{bmatrix} \left({df_1}/{dx_1} \right) & \left({df_1}/{dx_2} \right) \\ \left({df_2}/{dx_1} \right) & \left({df_2}/{dx_2} \right) \end{bmatrix}_{(n)} * \left\{ \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} - \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} \right\} = -\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}$$

$$\begin{bmatrix} \left({df_1}/{dx_1} \right) & \left({df_1}/{dx_2} \right) \\ \left({df_2}/{dx_1} \right) & \left({df_2}/{dx_2} \right) \end{bmatrix}_{(n)} * \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix} = -\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}$$

This can be solve by calculator (MODE →EQN→anx+bnY=Cn) to compute $\begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix}$, Then $\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix} + \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)}$

Example:

$$x_1^2 + x_1 x_2 = 10$$

$$x_2 + 3x_1 x_2^2 = 57$$

$$x_{1(0)} = 1.5;\; x_{2(0)} = 3.5$$

$$\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} = \begin{Bmatrix} x_1^2 + x_1 x_2 - 10 \\ x_2 + 3x_1 x_2^2 - 57 \end{Bmatrix}$$

$$\begin{bmatrix} \left({}^{df_1}/_{dx_1}\right) & \left({}^{df_1}/_{dx_2}\right) \\ \left({}^{df_2}/_{dx_1}\right) & \left({}^{df_2}/_{dx_2}\right) \end{bmatrix} = \begin{vmatrix} (2x_1 + x_2) & (x_1) \\ (3x_2^2) & (1 + 6x_1 x_2) \end{vmatrix}$$

$$\begin{bmatrix} (2x_1 + x_2) & (x_1) \\ (3x_2^2) & (1 + 6x_1 x_2) \end{bmatrix}_{(n)} * \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix} = -\begin{Bmatrix} x_1^2 + x_2 - 10 \\ x_2 + 3x_1 x_2^2 - 57 \end{Bmatrix}_{(n)}$$

@ First Iteration $\boldsymbol{x_{1(0)} = 1.5;\; x_{2(0)} = 3.5}$

$$\begin{bmatrix} (6.5) & (1.5) \\ (36.75) & (32.5) \end{bmatrix}_{(n)} * \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix} = -\begin{Bmatrix} -2.5 \\ 1.625 \end{Bmatrix}_{(n)}$$

This can be solve by calculator (MODE →EQN→anx+bnY=Cn)

$$\begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \end{Bmatrix} = \begin{Bmatrix} 0.54 \\ -0.66 \end{Bmatrix}$$

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \begin{Bmatrix} 0.54 \\ -0.66 \end{Bmatrix} + \begin{Bmatrix} 1.5 \\ 3.5 \end{Bmatrix} = \begin{Bmatrix} 2.04 \\ 2.84 \end{Bmatrix}$$

These values will be used for the second iteration and so on.

B) Manually by Inverse Jacobian.

The equation can be rearranged to be

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} - \begin{bmatrix} \left({}^{df_1}/_{dx_1}\right) & \left({}^{df_1}/_{dx_2}\right) \\ \left({}^{df_2}/_{dx_1}\right) & \left({}^{df_2}/_{dx_2}\right) \end{bmatrix}_{(n)}^{-1} * \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}$$

How to compute the Inverse Jacobian of a matrix?

$$\begin{vmatrix} \left({}^{df_1}/_{dx_1}\right) & \left({}^{df_1}/_{dx_2}\right) \\ \left({}^{df_2}/_{dx_1}\right) & \left({}^{df_2}/_{dx_2}\right) \end{vmatrix}^{-1} = \frac{1}{Matrix.Deter} * \begin{vmatrix} \left({}^{df_2}/_{dx_2}\right) & -\left({}^{df_1}/_{dx_2}\right) \\ -\left({}^{df_2}/_{dx_1}\right) & \left({}^{df_1}/_{dx_1}\right) \end{vmatrix}$$

Example:

$$x_1^2 + x_1 x_2 = 10$$

$$x_2 + 3x_1 x_2^2 = 57$$

$$x_{1(0)} = 1.5;\ x_{2(0)} = 3.5$$

$f_1 = x_1^2 + x_1 x_2 - 10$ $\qquad$ ${}^{df_1}/_{dx_1} = 2x_1 + x_2$ $\qquad$ ${}^{df_1}/_{dx_2} = x_1$

$f_2 = x_2 + 3x_1 x_2^2 - 57$ $\qquad$ ${}^{df_2}/_{dx_1} = 3x_2^2$ $\qquad$ ${}^{df_2}/_{dx_2} = 1 + 6x_1 x_2$

$$J^{-1} = \frac{1}{Det} \begin{vmatrix} \left({}^{df_2}/_{dx_2}\right) & -\left({}^{df_1}/_{dx_2}\right) \\ -\left({}^{df_2}/_{dx_1}\right) & \left({}^{df_1}/_{dx_1}\right) \end{vmatrix} = \frac{1}{165.12} * \begin{vmatrix} 32.5 & -1.5 \\ -36.75 & 6.5 \end{vmatrix} = \begin{vmatrix} 0.21 & -0.01 \\ -0.24 & 0.04 \end{vmatrix}$$

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} 1.5 \\ 3.5 \end{Bmatrix}}_{(n)} - \begin{bmatrix} 0.21 & -0.01 \\ -0.24 & 0.04 \end{bmatrix} * \begin{Bmatrix} -2.5 \\ 1.63 \end{Bmatrix}_{(n)} = \begin{Bmatrix} 2.04 \\ 2.84 \end{Bmatrix}$$

$x_{1(n)}$	$x_{2(n)}$	f_1	f_2	${}^{df_1}/_{dx_1}$	${}^{df_1}/_{dx_2}$	${}^{df_2}/_{dx_1}$	${}^{df_2}/_{dx_2}$	J^{-1}	$x_{1(n+1)}$	$x_{2(n+1)}$
1.5	3.5	-2.5	1.63	6.5	1.5	36.75	32.5	$\begin{matrix} 0.21 & -0.0 \\ -0.24 & 0.04 \end{matrix}$	2.04	2.84
2.04	2.84	-0.04	-4.80	6.92	2.04	24.20	35.76	$\begin{matrix} 0.18 & -0.0 \\ -0.12 & 0.03 \end{matrix}$	2	3

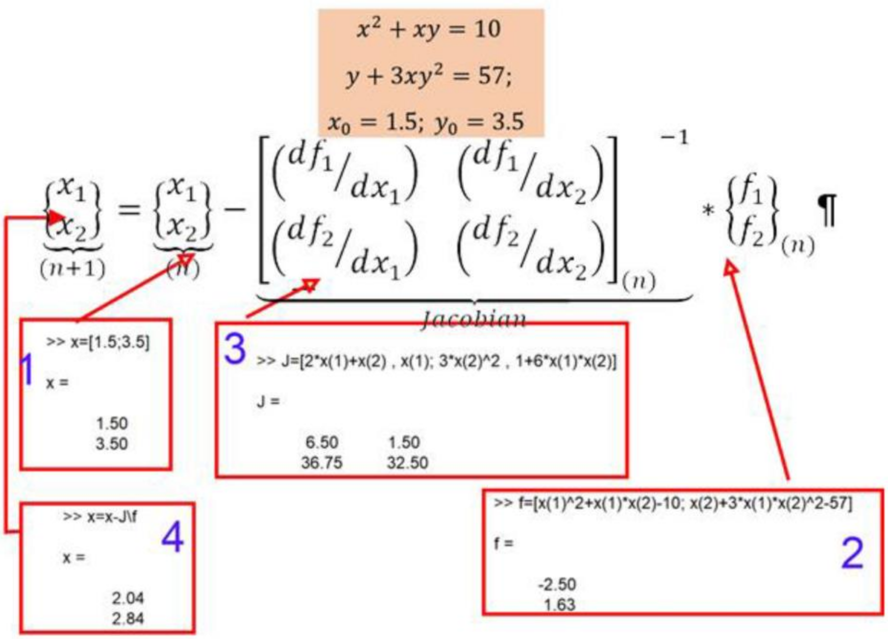
C) Using MATLAB to determine the roots of Nonlinear Equations using Newton Raphson.

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} - \underbrace{\begin{bmatrix} \left({df_1}/{dx_1} \right) & \left({df_1}/{dx_2} \right) \\ \left({df_2}/{dx_1} \right) & \left({df_2}/{dx_2} \right) \end{bmatrix}_{(n)}^{-1}}_{Jacobian} * \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}$$

In MatLab, it can be computed using the left division

$$\underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n+1)} = \underbrace{\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}}_{(n)} - \begin{bmatrix} \left({df_1}/{dx_1} \right) & \left({df_1}/{dx_2} \right) \\ \left({df_2}/{dx_1} \right) & \left({df_2}/{dx_2} \right) \end{bmatrix}_{(n)} \backslash \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}_{(n)}$$

Example



The full script of m. file is as follow:

```
disp('These codes are to find the roots of two nonlinear eqations using Newton Raphson Method,
By Ahmed Elkhatat')
%% INPUTS
x=input('Insert the intial guess of x1 and x2 as a vertical array:  ');
f1=input('insert the 1st non linear eqation eqation (for Example: x(1)^2*x(2)+3x(1)*5x(2):   ');
f2=input('insert the 2nd non linear eqation eqation (for Example: x(1)^2*x(2)+3x(1)*5x(2):   ');
df1x1=input('insert the dereviation of 1st non linear eqation eqation to (x1):   ');
df1x2=input('insert the dereviation of 1st non linear eqation eqation to (x2):   ');
df2x1=input('insert the dereviation of 2nd non linear eqation eqation to (x1):   ');
df2x2=input('insert the dereviation of 2nd non linear eqation eqation to (x2):   ');
RERR=input('Insert the relative error:  ');
itern=input('Insert the maximum iteration:  ');
%% ITERATION
iter=0;
while (1)
   f=[f1;f2];
   J=[df1x1 df1x2; df2x1 df2x2];
   xnew=x-J\f;
   ERR=abs((max(xnew)-max(x))/max(xnew));
   if iter==1
   data(iter, :)=[iter, x(1), x(2), ERR];
   end
  if ERR<=RERR; break ; else
     x=xnew;
     iter=iter+1; end;
     if iter>=itern; break
     end
     data(iter, :)=[iter, x(1), x(2), ERR];
end
 disp('      iter        x(1)          x(2)            Err')
     disp('____________________________________')
    data
```

```
Command Window

>> NewtonRaphson_nonlinear1
These codes are to find the roots of two nonlinear eqations using Newton Raphson Method, By Ahmed Elkhatat
Insert the intial guess of x1 and x2 as a vertical array:  [1.5;3.5]
insert the 1st non linear eqation eqation (for Example: x(1)^2*x(2)+3x(1)*5x(2):   x(1)^2+x(1)*x(2)-10
insert the 2nd non linear eqation eqation (for Example: x(1)^2*x(2)+3x(1)*5x(2):   x(2)+3*x(1)*x(2)^2-57
insert the dereviation of 1st non linear eqation eqation to (x1):   2*x(1)+x(2)
insert the dereviation of 1st non linear eqation eqation to (x2):   x(1)
insert the dereviation of 2nd non linear eqation eqation to (x1):   3*x(2)^2
insert the dereviation of 2nd non linear eqation eqation to (x2):   1+6*x(1)*x(2)
Insert the relative error:  0.1
Insert the maximum iteration:  100
      iter       x(1)        x(2)        Err
____________________________________
data =
        1.00      2.04      2.84      0.11
        2.00      2.57      2.19      0.11
        3.00      3.11      1.53      0.17
        4.00      3.64      0.88      0.15
        5.00      4.18      0.22      0.13
        6.00      4.72     -0.44      0.11
        7.00      5.25     -1.09      0.10
fx >>
```

D) Advanced MatLab to determine the roots of Nonlinear Equations using Newton Raphson

For more information about these codes, revise Module 3: Advanced MatLab to determine the roots of linear equations using Newton Raphson

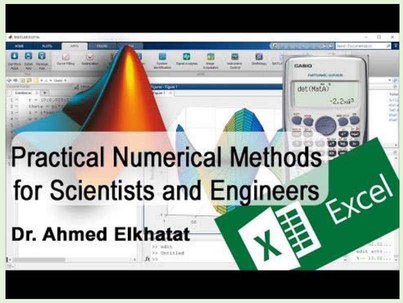
```
% Newton Raphson Nonlinear,  Author: Ahmed Elkhatat
clc;
close all;
clear all;
syms x y;
f1= input('Enter the 1st function here eg. x*exp(x):');
f2= input('Enter the 2nd function here eg. x*exp(x):');
g1x=diff(f1,x);
g1y=diff(f1,y);
g2x=diff(f2,x);
g2y=diff(f2,y);
x0=input('insert initial guesses of [x(1);x(2)] : '); x0=reshape(x0,[],1);
RERR = input('Enter the relative error:');
for i=1:100
     f1x=vpa(subs(f1,x,x0(1))); f1x=vpa(subs(f1x,y,x0(2)));
     f2x=vpa(subs(f2,x,x0(1))); f2x=vpa(subs(f2x,y,x0(2)));
    G1x=vpa(subs(g1x,x,x0(1))); G1x=vpa(subs( G1x,y,x0(2)));
    G1y=vpa(subs(g1y,x,x0(1))); G1y=vpa(subs(G1y,y,x0(2)));
    G2x=vpa(subs(g2x,x,x0(1))); G2x=vpa(subs(G2x,y,x0(2)));
    G2y=vpa(subs(g2y,x,x0(1))); G2y=vpa(subs(G2y,y,x0(2)));

    F=[f1x;f2x];
    J=[G1x G1y;G2x G2y];
  x_new=x0-J\F;
err=abs((max(x_new)-max(x0))/max(x_new));
 dat(i,:)=[i x_new(1) x_new(2) err];
if err<RERR;
break
end
x0=x_new;
end
fprintf('No of Iterations: %0.0f \nX: %0.3f\nY: %0.3f\nRelative Error: %0.3f\n',i, x_new(1),
x_new(2), err)
disp('Do you want to dispaly the whole iterations?')
D=input('If Yes Please insert 1: ');
if D==1;
disp('i       x                         y                                  Err')
disp('___________________________________________________')
dat
end
disp('_________________________')
disp('Thank you, Ahmed Elkhatat')
```

Command Window

```
Enter the 1st function here eg. x*exp(x):x^2+x*y-10
Enter the 2nd function here eg. x*exp(x):y+3*x*y^2-57
insert initial guesses of [x(1);x(2)] : [1.5;3.5]
Enter the relative error:0.01
No of Iterations: 3
X: 2.000
Y: 3.000
Relative Error: 0.001
Do you want to dispaly the whole iterations?
If Yes Please insert 1: 1
i     x                  y                          Err
____________________________________________
dat =
[ 1, 2.0360288230584467574059247397918, 2.8438751000800640512409927942354,   0.23071509009009009009009009009]
[ 2, 1.9987006090558240458905733069004, 3.0022885629245084066009984522318,   0.052764236189920033221158423256303]
[ 3, 1.9999999838762601867444946716248, 2.9999994133889130961352698866948, 0.00076304999440296568148005427494057]
_______________________
Thank you, Ahmed Elkhatat
>>
```

2. Using Excel Solver to determine the roots of Nonlinear Equations

Watch the online Video



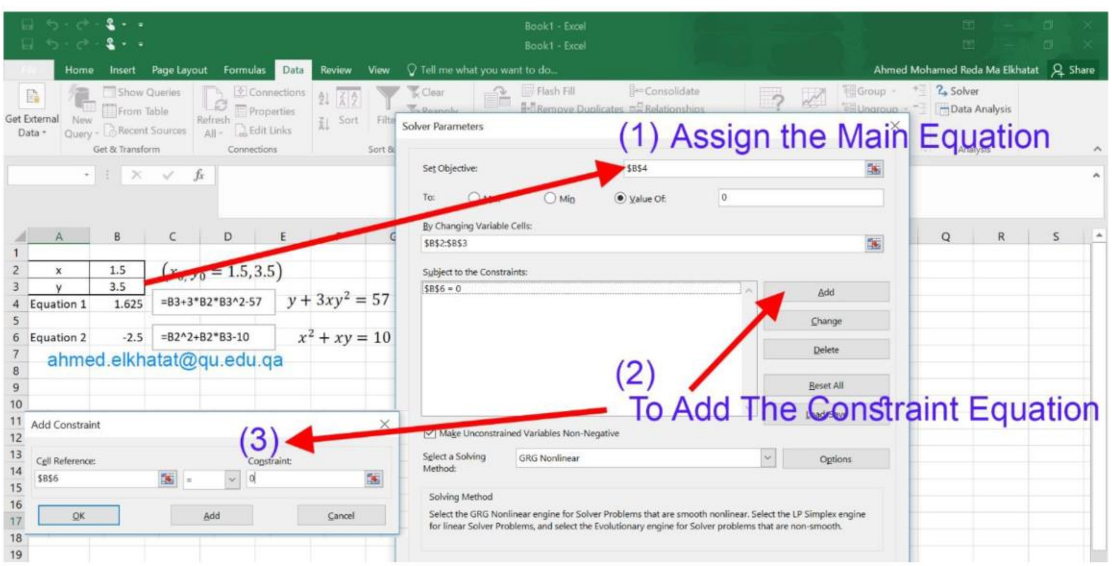
Excel Solver can be used efficiently to find the roots of a system on the nonlinear equation. But in order to solve two equations at once, we have to choose one equation to be the **main equation** while adding the other one to as a **constraint equation=0**

Example:

$$x_1^2 + x_1 x_2 = 10$$

$$x_2 + 3x_1 x_2^2 = 57$$

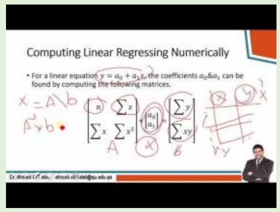
$$x_{1(0)} = 1.5;\ x_{2(0)} = 3.5$$



Module (5): Descriptive Statistics and Regression

I-Descriptive Statistics

Watch the online Video



Descriptive Statistics includes methods to measure the central tendency such as (Arithmetic Mean, Geometric Mean, Logarithmic Mean, Median, Mode), and methods to measure the spread such as (Range, Deviation, Mean Deviation, Variance, and Standard Deviation. In the following table, these measures are explained:

Measure	Description	Formula
Arithmetic Mean	is the sum of a collection of numbers divided by the count of numbers in the collection	$\bar{X} = \frac{x_1 + x_2 + \cdots + x_n}{n}$
Geometric Mean	It is often used when comparing different items	$\bar{X} = \sqrt[n]{x_1 * x_2 * \ldots * x_n}$
Logarithmic Mean	This mean is applicable in engineering problems involving heat and mass transfer.	$X_{LM} = \frac{x_1 - x_2}{ln\frac{x_1}{x_2}}$
Median	The "median" is the "middle" value in the list of numbers. To find the median, your numbers have to be listed in numerical order from smallest to largest, so you may have to rewrite your list before you can find the median.	
Mode	The "mode" is the value that occurs most often. If no number in the list is repeated, then there is no mode for the list.	
Range	The difference between the largest and the smallest value	$R = X_{max} - X_{min}$
Deviation	It is a measure of the difference between the observed value of a variable and the mean.	$Dev = \bar{X} - X_i$
Mean Deviation	It is the sum of deviations divided by the count of their numbers.	$\frac{\sum(\bar{X} - X_i)}{n}$
Sample Variance	It is the sum of squared deviations divided by the degree of freedom.	$Var = \frac{\sum(\bar{X} - X_i)^2}{n-1}$
Sample Standard Deviation	It is the squared root of the variance	$S = \sqrt{\frac{\sum(\bar{X} - X_i)^2}{n-1}}$

Coefficient of Variation	It is the ratio of the standard deviation to the mean. It shows the extent of variability concerning the mean of the population.	$\frac{S}{\overline{X}} * 100$

Using MatLab to Compute (Mean, Median, Mode, Range, Variance and Standard deviation)

Example: Create a MATLAB script file that displays the following statistics for any given input matrix (Mean, Median, Mode, Range, Variance and Standard deviation)

```
A=input('Insert the matrix:');
Mean=mean(A(:));
Median = median(A(:));
Mode = mode(A(:));
Range = range(A(:));
Variance=var(A(:));
S = std(A(:));
fprintf('\nMean: %0.2f\nMedian: %0.2f\nMode: %0.2f\nRange: %0.2f\n',Mean,Median,Mode,Range);
fprintf('Variance: %0.2f\nStandard deviation:%0.2f',Variance,S);
```

- **A(:)** rearrange the elements to be in one column only.
- **(A,[],'all')** can be used also

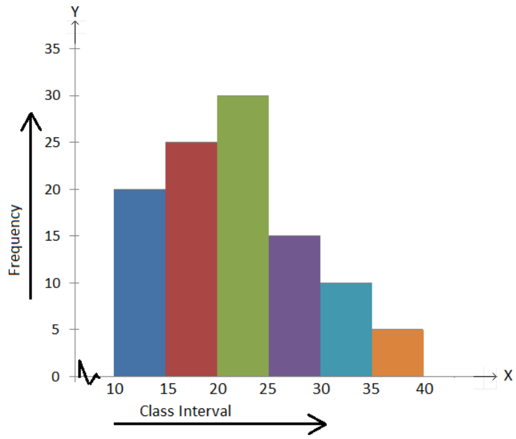
Using Excel to Compute (Mean, Median, Mode, Range, Variance and Standard deviation)

Matrix				
8.8	9.4	10	9.8	10.1
9.5	10.1	10.4	9.5	9.5
9.8	9.2	7.9	8.9	9.6
9.4	11.3	10.4	8.8	10.2
10	9.4	9.8	10.6	8.9

Statistics	Result	formula
Mean	9.65	=AVERAGE(D5:H9)
Median	9.60	=MEDIAN(D5:H9)
Mode	9.40	=MODE(D5:H9)
Variance	0.49	=VAR(D5:H9)
Range	3.40	=MAX(D5:H9)-MIN(D5:H9)
Standard Deviation	0.70	=STDEV(D5:H9)

Histogram

A histogram is a graphical display of data, where these data are grouped into ranges displayed as bars of different heights.



A) Histogram in Excel

For example, the Grades of students in a Lab were as follows, can you categorize them into groups (≤60, 65, 70, 75, 80, 85, 90≥) to help the instructor?

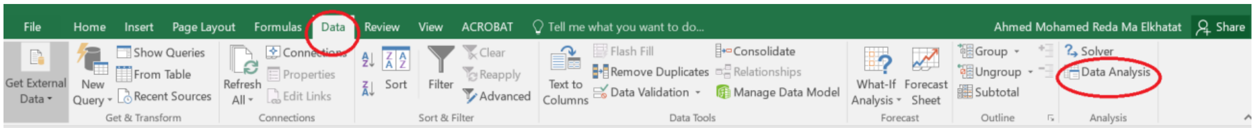
83.71	90.79	90.00	71.36	80.07	75.09	87.00
91.17	83.21	59.91	91.17	75.22	78.96	88.83

Method 1

1) in Excel sheet, create two sets of data; one for the data, while the other one for the upper limit of each group (Bin) as follows:

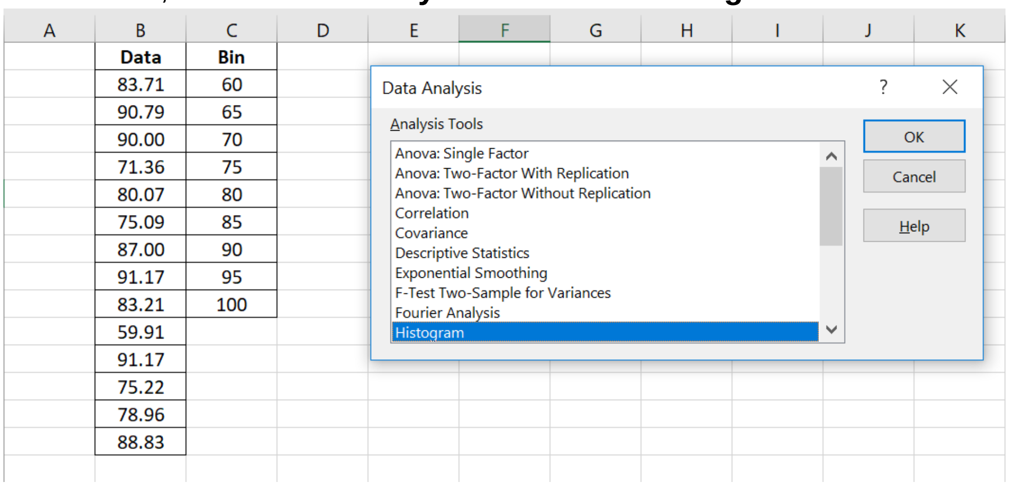
	A	B	C
1		**Data**	**Bin**
2		83.71	60
3		90.79	65
4		90.00	70
5		71.36	75
6		80.07	80
7		75.09	85
8		87.00	90
9		91.17	95
10		83.21	100
11		59.91	
12		91.17	
13		75.22	
14		78.96	
15		88.83	

2) in **Data** tab, select **Data Analysis**:

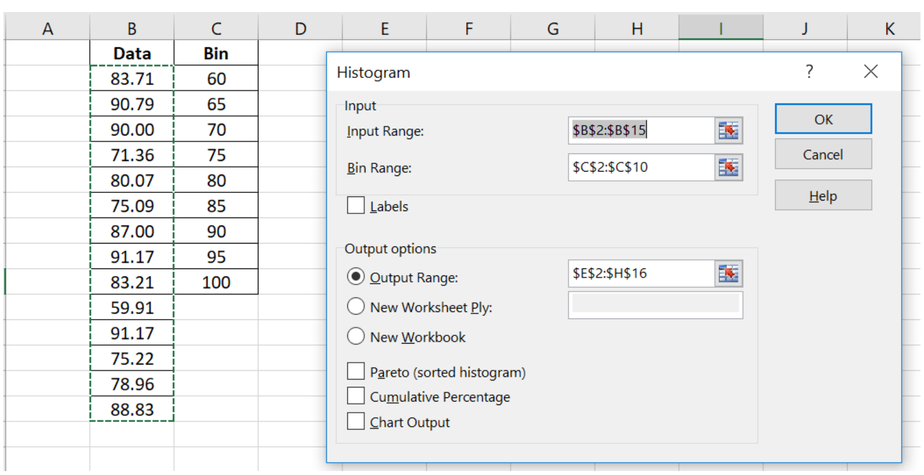


Note: If you can't find the Data Analysis button, click Options on the File tab. Under **Add-ins**, select **Analysis ToolPak** and click on the **Go** button. Check **Analysis ToolPak** and click on **OK**.

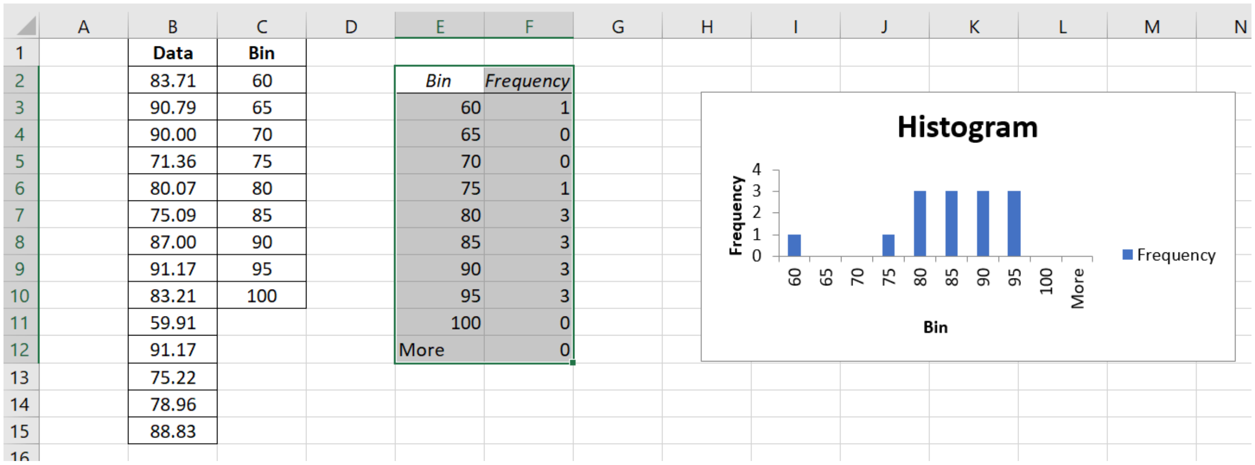
3) in **Data** tab, select **Data Analysis** and select **Histogram**



4) In **Input Range: select the Data Column**, and in Bin Range, select **Bin Column**. If you want to allocate your data in an Excel sheet, then select OUT Range or New Workbook to put them in the new sheet. To plot the Histogram, select **Chart Output**



5) Excel will produce the data as follows.

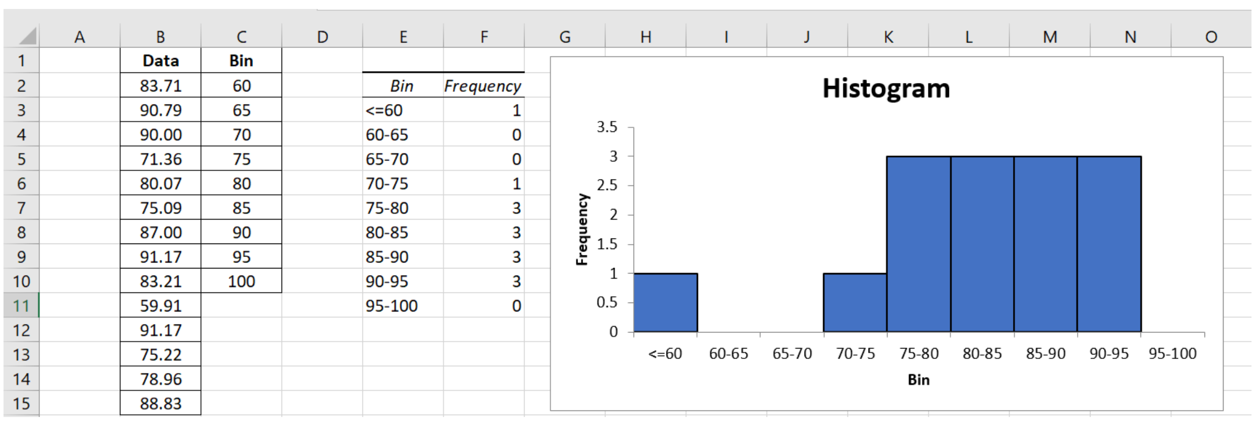


Data	Bin
83.71	60
90.79	65
90.00	70
71.36	75
80.07	80
75.09	85
87.00	90
91.17	95
83.21	100
59.91	
91.17	
75.22	
78.96	
88.83	

Bin	Frequency
60	1
65	0
70	0
75	1
80	3
85	3
90	3
95	3
100	0
More	0

6) You can improve the Histogram by:

- **a.** Removing the gaps between the bars by right click a bar. Click Format Data Series and change the Gap Width to 0%. Excel will produce the data.
- **b.** Editing the Bin column and making it Range.
- **c.** Adding outline to your bars.



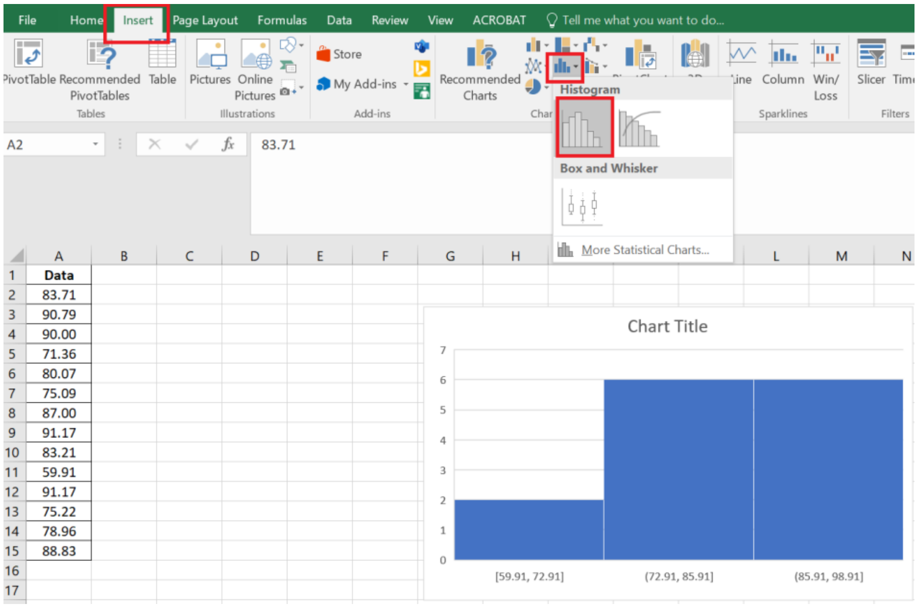
Data	Bin
83.71	60
90.79	65
90.00	70
71.36	75
80.07	80
75.09	85
87.00	90
91.17	95
83.21	100
59.91	
91.17	
75.22	
78.96	
88.83	

Bin	Frequency
<=60	1
60-65	0
65-70	0
70-75	1
75-80	3
80-85	3
85-90	3
90-95	3
95-100	0

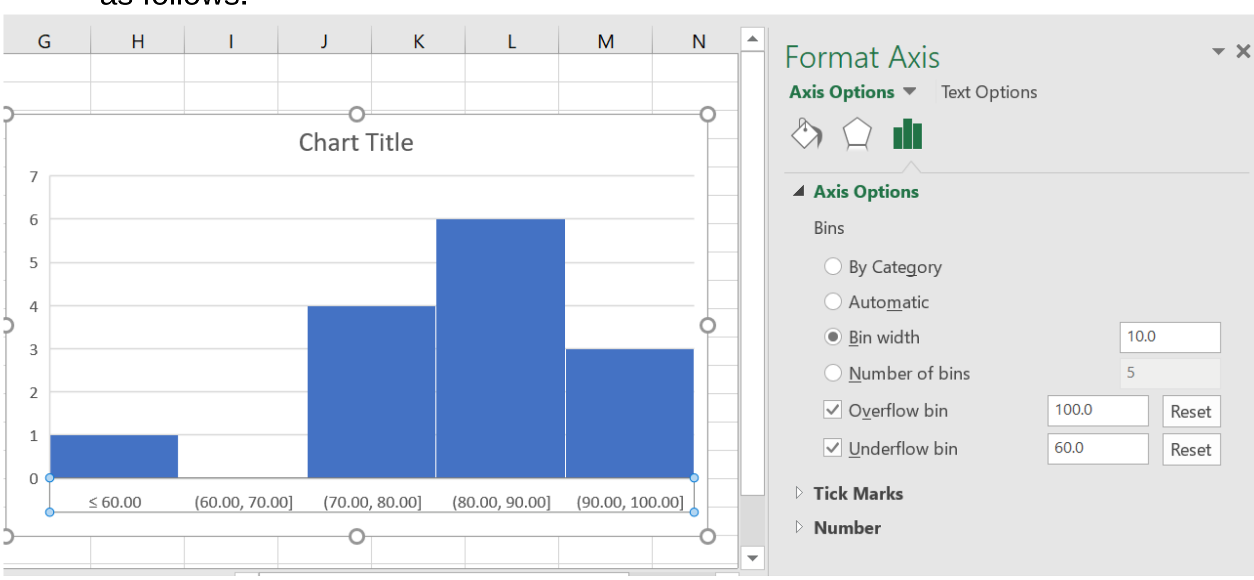
Method 2

In this method, Bins will be created later.

1) In the Excel sheet, select your data, then in the **Insert** tab, select **Histogram** as shown below.



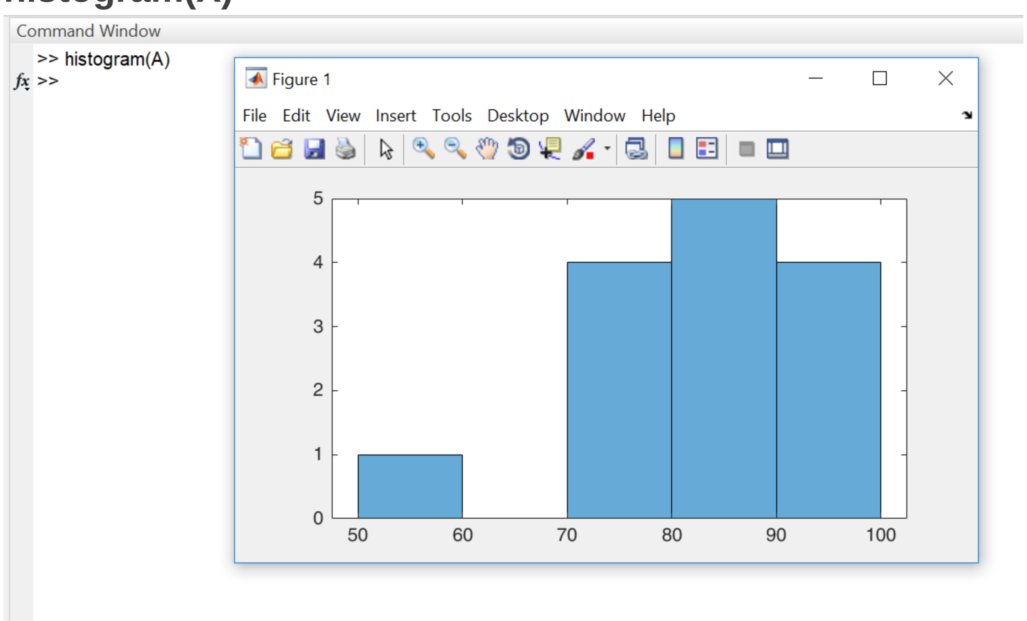
2) To set the Bins, right-click the **x-axis** and select **Format Axis**.
   a. You can either set Bin Width or Number or Bins.
   b. You can also set the Underflow (minimum) and Overflow (maximum) Bins as follows.



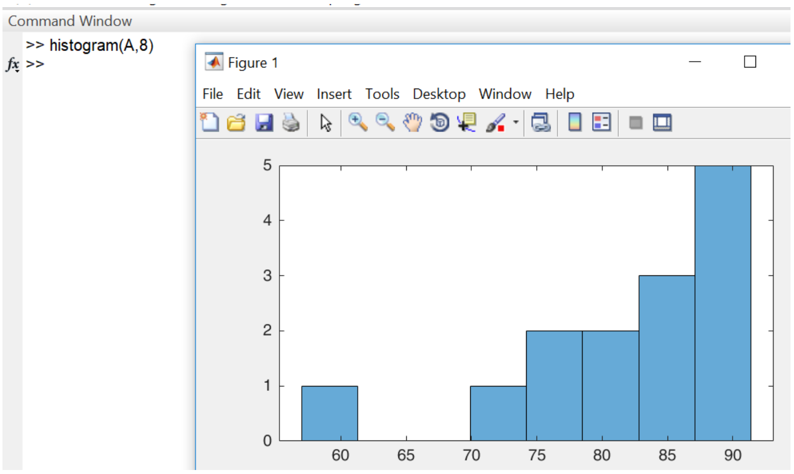
B) Histogram in MatLab

Syntax	Description
histogram(X)	Creates a histogram plot of X. The histogram function uses an automatic binning algorithm that returns bins with a uniform width, chosen to cover the Range of elements in X and reveal the underlying shape of the distribution
histogram(X,nbins)	Uses a number of bins specified by the scalar, nbins.
histogram(X,edges)	Sorts X into bins with the bin edges specified by the vector, edges. Each bin includes the left edge, but does not include the right edge, except for the last bin which includes both edges.
histogram(X,'BinWidth',n)	Sorts X into bins with the bin width specified by a number
histogram('BinEdges',edges,'BinCounts',counts)	Manually specifies bin edges and associated bin counts. Histogram plots the specified bin counts and does not do any data binning.

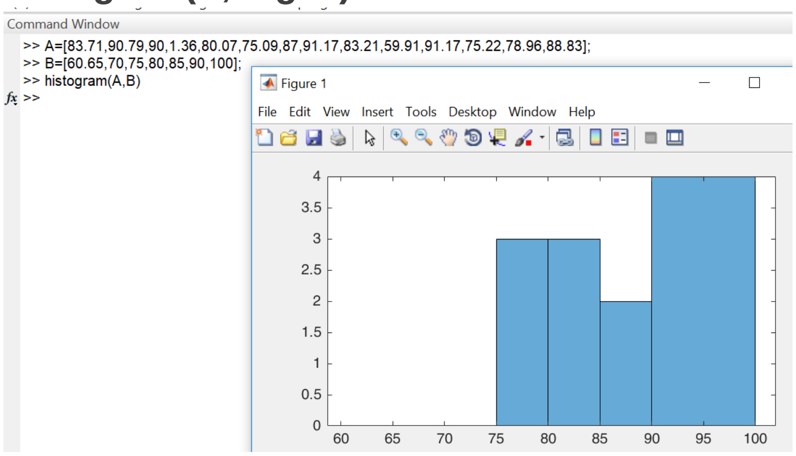
histogram(X)



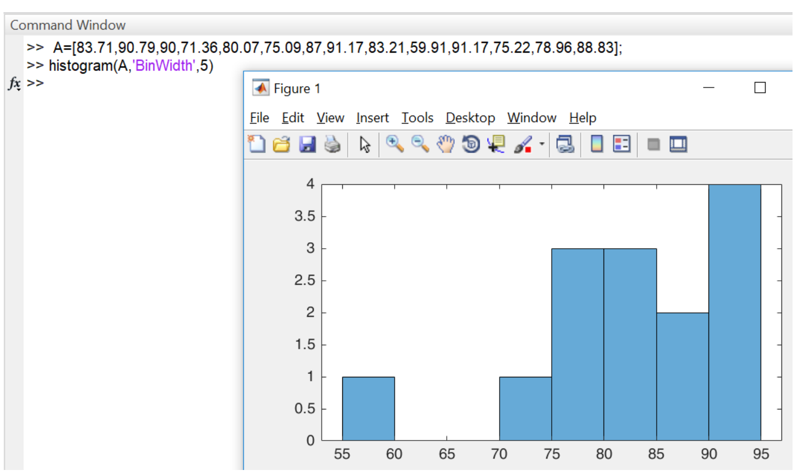
histogram(X,nbins)



histogram(X,edges)



histogram(X,'BinWidth',5)



II-Regression

For a set of (x,y) data, where (x) is the independent variable, while (y) is the dependent one, the regression is conducted to find a function that fits the general trend of (x,y) data with no need to match every point.

Regression equation can be linear (1st order) or polynomial (2nd or 3rd or higher). The maximum order of equation is n-1.

For more complicated nonlinear relationships such as exponential, power and growth rate equation; their relations can be linearized to get a powerful regression.

1. Linear Regression

A) Computing Linear Regressing Numerically

For a linear equation $y = a_0 + a_1 x$, the coefficients $a_0 \& a_1$ can be found by computing the following matrices.

$$\begin{bmatrix} n & \sum x \\ \sum x & \sum x^2 \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum xy \end{bmatrix}$$

Example:

Fit a straight line to the values of the following table.

x	0	1	2	3	4	5
y	3	9	10	15	18	26

n	X	Y	XY	X^2
1	0	3	0	0
2	1	9	9	1
3	2	10	20	4
4	3	15	45	9
5	4	18	72	16
6	5	26	130	25
Sum	15	81	276	55

$$\begin{bmatrix} 6 & 15 \\ 15 & 55 \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 81 \\ 276 \end{bmatrix}$$

Then, $a_0 \& a_1$ can be computed using by a calculator, Excel or Matlab

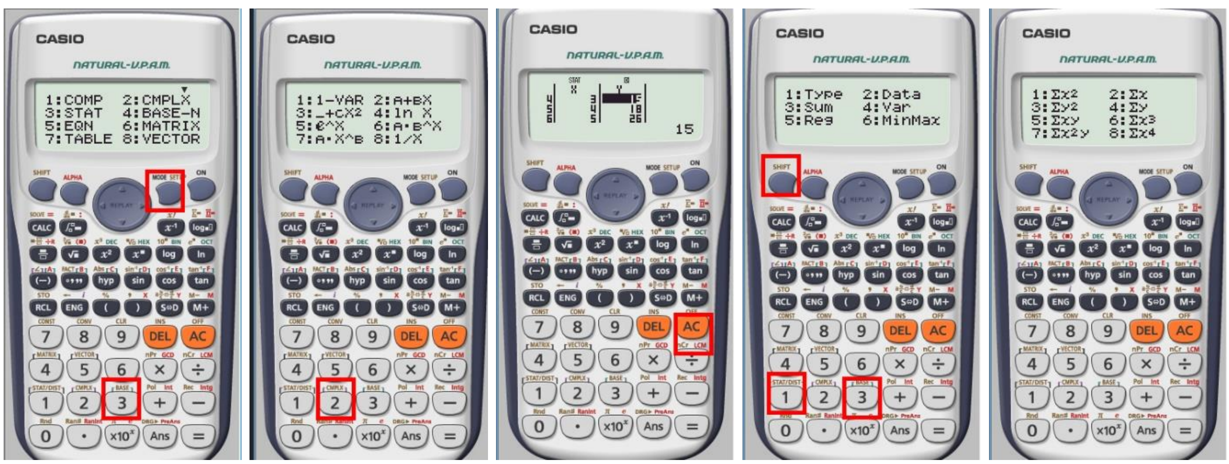
a) By Excel: Using (MINVERSE and MMULT).
b) By Calculator: Using (Mode>>5:EQN>>1:anX+bnY=Cn)

The linear regression equation will be

$$y = 3 + 4.2x$$

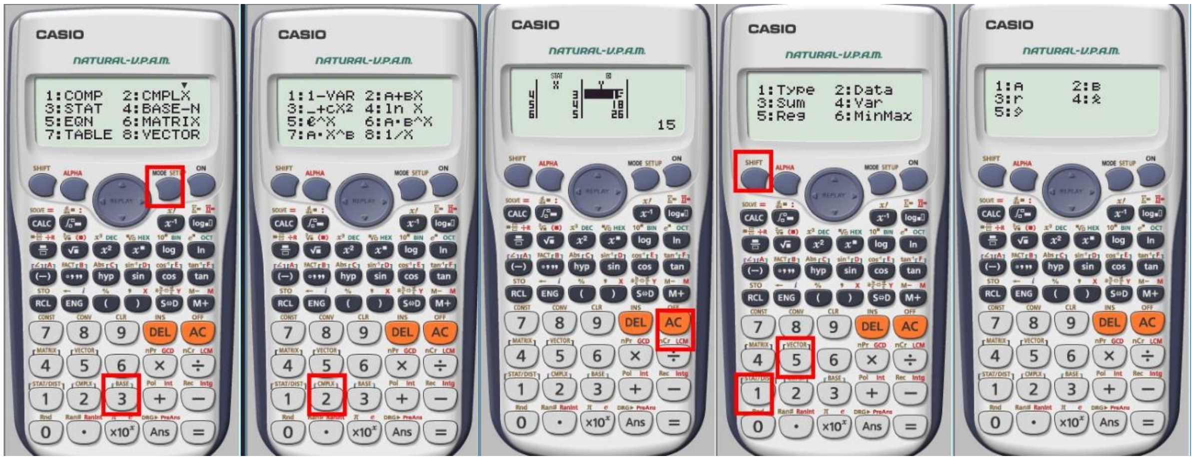
Useful functions to find $\sum \boldsymbol{x}, \sum \boldsymbol{y}, \sum \boldsymbol{x}\,\boldsymbol{y}, \sum \boldsymbol{x}^2$ By Calculator

a) MODE>>3:STAT>>2:A+BX
b) Insert (x, y) variables, then (AC)
c) SHIFT+1>>3:SUM>> Choose the interested summation you want



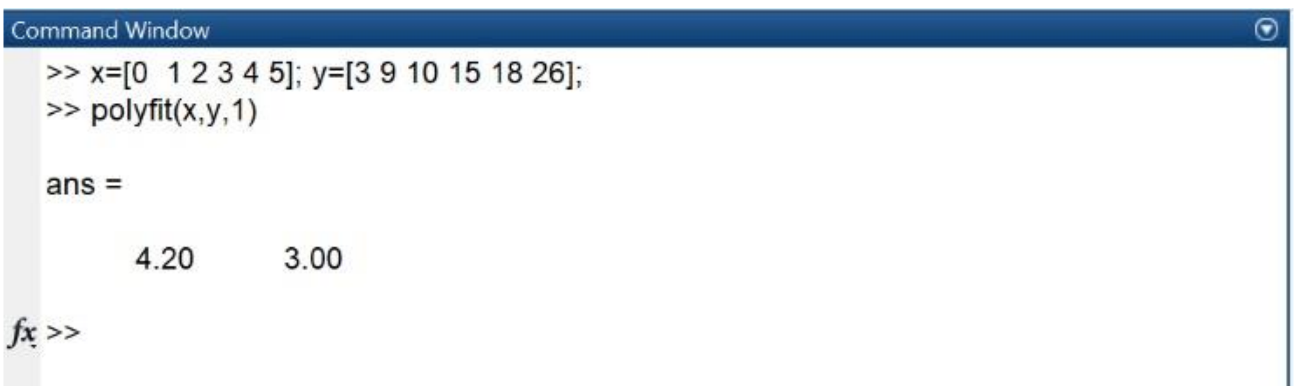
B) Using Calculator to Compute Linear Regressing Equation

a) MODE>>3:STAT>>2:A+BX
b) Insert (x, y) variables, then (AC)
c) SHIFT+1>>5:Reg>> 1:A (for intercept[a_0]) of 2:B(for Slope [a_1])



Using MatLab to find linear line coefficients $a_0 \& a_1$

This can be executed using the build-in function **polyfit(x,y,n)**. The output of this function is [slope, Intercept]



Then 4.20 is the slope and 3.00 is the intercept.

C) Quantification of Error of Linear Regression

Both Coefficient of Determination (R^2) and Correlation Coefficient (R) are used to quantify the error of linear regression.

$$R^2 = \frac{ST - SR}{ST} \qquad R = \sqrt{R^2}$$

where,

ST: (Sum of Squares of True Deviation) measures the deviations of the observations from their mean $\bar{Y} = \frac{\sum Y}{n}$

$$ST = \sum (Y_{true} - \bar{Y})^2$$

SR: (Sum of Squares of Regression Deviation) measures the deviations of observations from their predicted

$$SR = \sum \left(Y_{true} - Y_{reg}\right)^2$$

Example:

Calculate the correlation Coefficient of the linear regression line of the following table.

x	0	1	2	3	4	5
y	3	9	10	15	18	26

$$\bar{Y} = \frac{\sum Y}{n} = \frac{3 + 9 + 10 + 15 + 18 + 26}{6} = 13.5$$

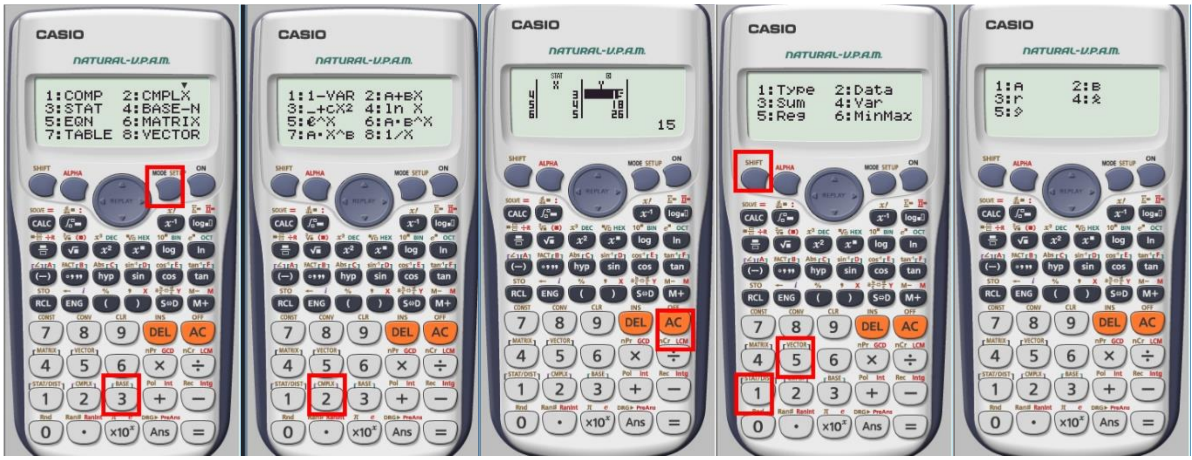
$$ST = \sum (Y_{true} - \bar{Y})^2 = (3 - 13.5)^2 + \cdots + (26 - 13.5)^2 = 321.5$$

$$SR = \sum \left(Y_{true} - Y_{reg}\right)^2 = (3 - 3)^2 + .. + (26 - 24)^2 = 12.5$$

$$R^2 = \frac{321.5 - 12.5}{321.5} = 0.96 \qquad R = \sqrt{R^2} = 0.98$$

Using Calculator to find the Correlation Coefficient

a) MODE>>3:STAT>>2:A+BX
b) Insert (x, y) variables, then (AC)
c) SHIFT+1>>5:Reg>> 3:r



D) Linearization of Non Linear Relationships

Non-linear relationships such as exponential, power and saturation-growth rate equations can be transformed into linear relations as following

Relation	Non Linear Relationships	Linearization
Exponential	$y = ae^{Bx}$	$\ln y = \ln a + Bx$
Power	$y = ax^{B}$	$\log y = \log a + Blogx$
Saturation-growth rate	$y = a\dfrac{x}{B+x}$	$\dfrac{1}{y} = \dfrac{1}{a} + \dfrac{B}{a}.\dfrac{1}{x}$

E) Create Matlab Function to find the coefficients of a linear regression Line

We want to create a MatLab function according to the following conditions:

1) **Inputs**: x (independent variable) and y (dependent variable)
2) **Outputs**: trend line (equation of the regression line), and R (correlation coefficient)
3) **Note**: Number of variables in (x) should be equal to variables at (y).

```
function trendline(x,y)
%this function to create trendline equation and calculate its R and R2
% Written by Ahmed Elkhatat
% to check the length of two vectors
if length (x)~=length (y), error(' x and y must have the same length'); end
% to convert the horizontal vectors into vertical vectors to do th required
% calculations
x=x(:);y=y(:);
% to create the sums
Sx=sum(x); Sy=sum(y); Ay=mean(y) ;
Sx2=sum(x.*x); Sxy=sum(x.*y);
% to create the martices and solve the equations
n=length (x);
MA=[n  Sx; Sx Sx2]; Mb=[Sy;Sxy];
a=MA\Mb;
Yreg=a(2)*x+a(1);
ST=sum((y-Ay).*(y-Ay));
SR=sum((Yreg-y).*(Yreg-y));
R2=(ST-SR)/ST;
R=sqrt(R2);
fprintf('the regression equation is y= (%0.2f) +(%0.2f) * x\n',a(1), a(2))
fprintf('the coefficent of determination (R^2) is %0.2f and the correlation coeficient (R) is  equation is  %0.2
f\n',R2, R)
end
```

(X.X)=(X.^2)

Command Window

```
>> x=[0  1 2 3 4 5]; y=[3 9 10 15 18 26];
>> trendline(x,y)
the regression equation is y= (3.00) +(4.20) * x
the coefficent of determination (R^2) is 0.96 and the correlation coeficient (R) is  equation is  0.98
>>
```

3. Polynomial Regression

A) Computing Polynomial Regressing Numerically

2nd Order Regression

$$\boldsymbol{y = a_0 + a_1 x + a_2 x^2}$$

$$\begin{bmatrix} n & \sum x & \sum x^2 \\ \sum x & \sum x^2 & \sum x^3 \\ \sum x^2 & \sum x^3 & \sum x^4 \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum xy \\ \sum x^2 y \end{bmatrix}$$

This type of matrix can be solved easly using Casio calculator similar to the linear regression

MODE>>5:EQN>>3:ax2+bx+c=0

3rd Order Regression

$$\boldsymbol{y = a_0 + a_1 x + a_2 x^2 + a_3 x^3}$$

$$\begin{vmatrix} n & \sum x & \sum x^2 & \sum x^3 \\ \sum x & \sum x^2 & \sum x^3 & \sum x^4 \\ \sum x^2 & \sum x^3 & \sum x^4 & \sum x^5 \\ \sum x^3 & \sum x^4 & \sum x^5 & \sum x^6 \end{vmatrix} * \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum xy \\ \sum x^2 y \\ \sum x^3 y \end{Bmatrix}$$

In contrast, this type of matrix can not be solved using CASIO, fx-991ES PLUS calculator, as it only can solve up to 3 equations with 3 unknowns. However, other models (CANON: F-792SGA) can solve 4 equations with 4 unknowns

MatLab is a useful tool to compute the regression of n[th] orders, using the function polyfit(x,y,n)

```
Command Window
>> x=[0  1 2 3 4 5]; y=[3 9 10 15 18 26];
>> polyfit(x,y,3)

ans =

        0.30        -1.90        6.65        3.18

fx >>
```

Thus the equation can be written as

$$y = 3.18 + 6.65x - 1.90x^2 + 0.3x^3$$

in order to find the value of (y) at certain (x) using the polynomial coefficient another built-in function is useful.

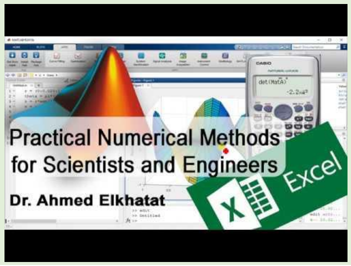
$$polyval([a_{n,} a_{n-1}, \dots, a_1], [x_i, x_{ii}])$$

```
Command Window
>> x=[0  1 2 3 4 5]; y=[3 9 10 15 18 26];
>> P=polyfit(x,y,3)

P =

        0.30        -1.90        6.65        3.18

>> polyval(P,2.3)

ans =

       12.03

>> polyval([0.3 -1.90 6.65 3.18],[0 1 2 3])

ans =

        3.18        8.23        11.28        14.13
```

ahmed.elkhatat@qu.edu.qa

B) Create MatLab Function to find the coefficients of 2nd Order regression Curve.

Watch the online Video




```
function trendline2(x,y)
%this function to create trendline equation and calculate its R and R2
% Written by Ahmed Elkhatat
% to check the length of two vectors
if length (x)~=length (y), error(' x and y must have the same length'); end
% to convert the horizontal vectors into vertical vectors to do th required
% calculations
x=x(:);y=y(:);
% to create the sums
Sx=sum(x); Sy=sum(y); Ay=mean(y) ;
Sx2=sum(x.^2); Sx3=sum(x.^3); Sx4=sum(x.^4); Sxy=sum(x.*y); Sx2y=sum((x.^2).*y);
% to create the martices and solve the equations
n=length (x);
MA=[n  Sx Sx2; Sx Sx2 Sx3;Sx2 Sx3 Sx4 ]; Mb=[Sy;Sxy; Sx2y];
a=MA\Mb;
Yreg=a(3)*(x.^2)+a(2)*x+a(1);
ST=sum((y-Ay).*(y-Ay));
SR=sum((Yreg-y).*(Yreg-y));
R2=(ST-SR)/ST;
R=sqrt(R2);
fprintf('the regression equation is y= (%0.2f) + (%0.2f)*x + (%0.2f)*x^2\n',a(1), a(2), a(3))
fprintf('the coefficent of determination (R^2) is %0.2f and the correlation coeficient (R) is  equation is  %0.2f\n',R2, R)
end
```

ahmed.elkhatat@qu.edu.qa

Command Window

```
>> x=[0  1 2 3 4 5]; y=[3 9 10 15 18 26];
>> trendline2(x,y)
the regression equation is y= (4.07) + (2.59)*x + (0.32)*x^2
the coefficent of determination (R^2) is 0.97 and the correlation coeficient (R) is  equation is  0.99
>>
```

2. Multiple Regression.

Multiple regression formula is used in the analysis of relationship between one dependent variable (y) and multiple independent variables (X1, X2,....xn).
The general formula for linear multiple regression is

$$\boldsymbol{y = a_0 + a_1x_1 + a_2x_2 + \cdots .. + x_nx_n}$$

Formula

$$\boldsymbol{y = a_0 + a_1x_1 + a_2x_2}$$

$$\begin{bmatrix} n & \sum x_1 & \sum x_2 \\ \sum x_1 & \sum {x_1}^2 & \sum x_1x_2 \\ \sum x_2 & \sum x_1x_2 & \sum {x_2}^2 \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum x_1y \\ \sum x_2y \end{bmatrix}$$

$$\boldsymbol{y = a_0 + a_1x_1 + a_2x_2 + a_3x_3}$$

$$\begin{vmatrix} n & \sum x_1 & \sum x_2 & \sum x_3 \\ \sum x_1 & \sum {x_1}^2 & \sum x_1x_2 & \sum x_1x_3 \\ \sum x_2 & \sum x_2x_1 & \sum {x_2}^2 & \sum x_2x_3 \\ \sum x_3 & \sum x_3x_1 & \sum x_3x_2 & \sum {x_3}^2 \end{vmatrix} * \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum x_1y \\ \sum x_2y \\ \sum x_3y \end{Bmatrix}$$

Example

In a corporation HR, they found that the employee's salary is a linear function of both his age and experience. Can you help them to find the regression equation?

Experience	5	7	8	6	7
Age	18	20	22	23	23
Salary	26315	39493	37209	24380	25751

Create MatLab Function to find the coefficients of multi regression Curve.

```
function trendlinemulti(x1,x2,y)
%this function to create multi trendline equation
% Written by Ahmed Elkhatat
% to check the length of two vectors
n=length (x1);
if length (x2)~=n | length (y)~=n, error(' x1, x2 and y must have the same length'); end
% to convert the horizontal vectors into vertical vectors to do th required
% calculations
x1=x1(:); x2=x2(:); y=y(:);
% to create the sums
Sx1=sum(x1); Sx2=sum(x2); Sx1sq=sum(x1.^2); Sx2sq=sum(x2.^2); Sx1x2=sum(x1.*x2);
Sy=sum(y); Sx1y=sum(x1.*y); Sx2y=sum(x2.*y);

% to create the martices and solve the equations

MA=[n  Sx1 Sx2; Sx1 Sx1sq  Sx1x2;  Sx2 Sx1x2 Sx2sq]; Mb=[Sy;Sx1y; Sx2y];
a=MA\Mb;

fprintf('the regression equation is y= (%0.2f) +(%0.2f) * x1 + (%0.2f) * x2\n',a(1), a(2), a(3))
end
```

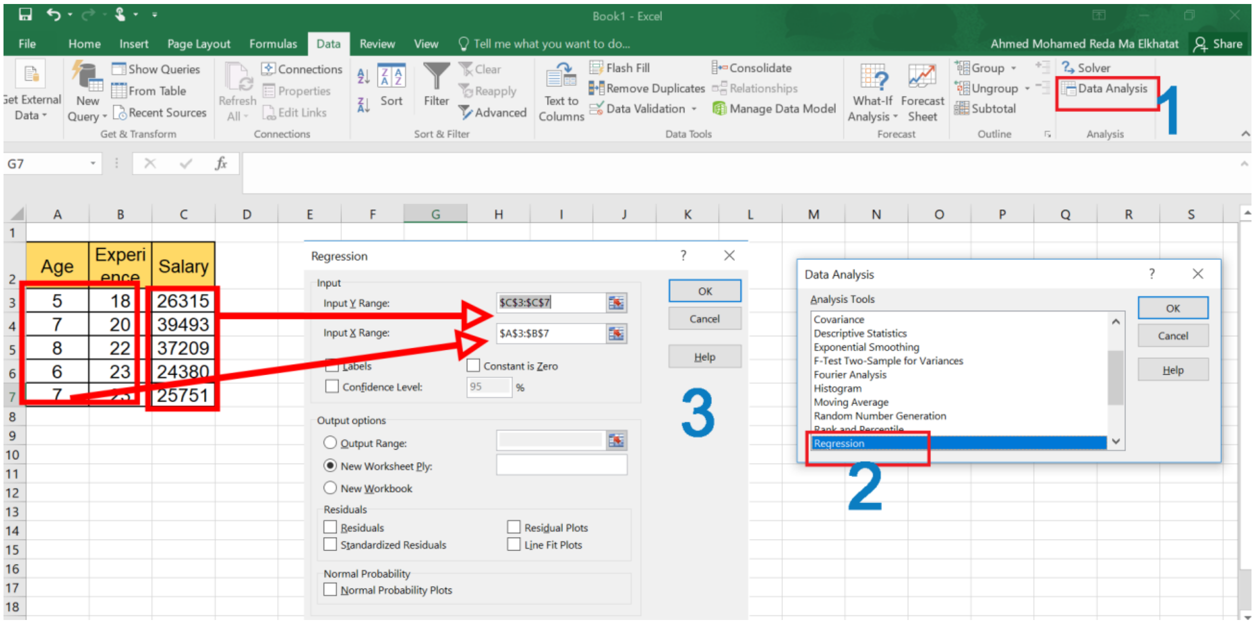
Dr. Ahmed Elkhatat

In order to solve the previous example

```
Command Window
>> x1= [5 7 8 6 7]; x2= [18 20 22 23 23]; y= [26315 39493 37209 24380 25751];
>> trendlinemulti(x1,x2,y)
the regression equation is y= (40237.58) +(6793.52) * x1 + (-2568.17) * x2
fx >>
```

Find the coefficients of multi regression Curve using Excel.

DATA>> DATA Analysis>> Regression



	A	B	C	D	E	F	G	H	I
1	SUMMARY OUTPUT								
2									
3	*Regression Statistics*								
4	Multiple R	0.929221							
5	R Square	0.863451							
6	Adjusted F	0.726902							
7	Standard I	3725.822							
8	Observati	5							
9									
10	ANOVA								
11		*df*	*SS*	*MS*	*F*	*gnificance F*			
12	Regressior	2	1.76E+08	87779439	6.32337	0.136549			
13	Residual	2	27763498	13881749					
14	Total	4	2.03E+08						
15									
16		*Coefficients*	*ndard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
17	Intercept	40237.58	18320.23	2.196347	0.159218	-38588	119063.2	-38588	119063.2
18	X Variable	6793.525	1950.469	3.483022	0.073463	-1598.66	15185.71	-1598.66	15185.71
19	X Variable	-2568.17	1025.798	-2.50359	0.12931	-6981.82	1845.479	-6981.82	1845.479
20									
21									

3. Compute Least Square Curve fitting using Excel Solver

Curve fits modeling is used in Excel to find the best fit line or curve for a series of data points by minimizing the error square between the original data and the values predicted by the equation.

How to Apply The Best Fit Model?

1) Suppose that you want to create a best fit model that has a Saturation-growth rate relationship with the independent variable (x)

x	3.4	6.9	7.4	11.2	13.1	18.9
y	0.981	1.489	1.542	1.846	1.955	2.189

2) We know that the regression curve of the saturation-growth rate relationship equation is $y = a\frac{x}{B+x}$, but the coefficients (a) and (B) are unknowns.

3) Therefore, we can guess initial values for both (a) and (B), and compute the function $a\frac{x}{B+x}$ based on these two initial guesses.

	A	B	C	D	E	F	G	H	I	J
1										
2										
3	x	3.4	6.9	7.4	11.2	13.1	18.9		(a) guess Value	1
4	y	0.9808	1.4892	1.5417	1.8462	1.9552	2.1892		(B) guess Value	5
5	Y Model	0.4048	0.5798	0.5968	0.6914	0.7238	0.7908			
6										
7										
8										
9										
10										
11										

=$J$3*(B3/($J$4+B3))

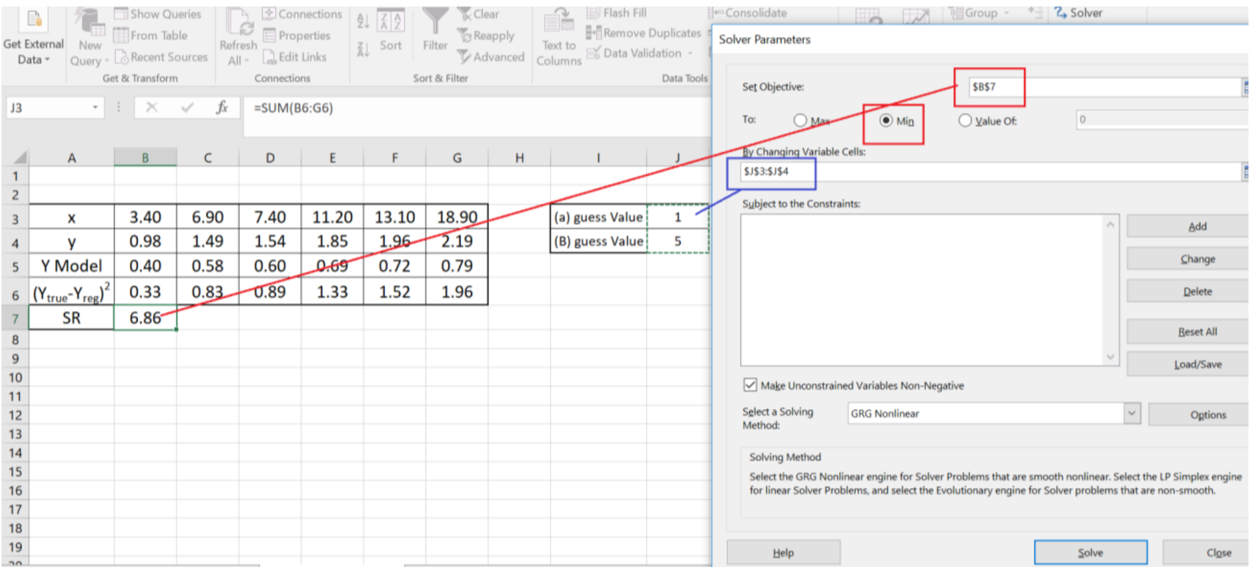
`

4) Create SR: (Sum of Squares of Regression Deviation), where

$$SR = \sum \left(Y_{true} - Y_{reg}\right)^2$$

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3	x	3.40	6.90	7.40	11.20	13.10	18.90		(a) guess Value	1		
4	y	0.98	1.49	1.54	1.85	1.96	2.19		(B) guess Value	5		
5	Y Model	0.40	0.58	0.60	0.69	0.72	0.79					
6	$(Y_{true}-Y_{reg})^2$	0.33	0.83	0.89	1.33	1.52	1.96					
7	SR	6.86										
8												

5) The best regression line is that line has minimum SR value, thus solver can be used to achieve this by changing the guess values (a) and (b)

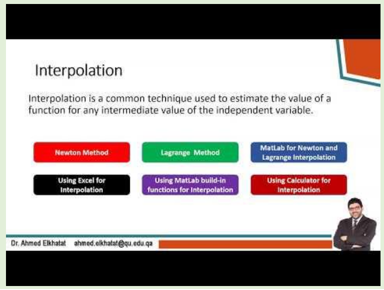


2										
3	x	3.40	6.90	7.40	11.20	13.10	18.90		(a) guess Value	2.999965
4	y	0.98	1.49	1.54	1.85	1.96	2.19		(B) guess Value	6.99978
5	Y Model	0.98	1.49	1.54	1.85	1.96	2.19			
6	$(Y_{true}-Y_{reg})^2$	0.00	0.00	0.00	0.00	0.00	0.00			
7	SR	0.00								
8										

Thus, the best-fit regression equation is $y = 3\frac{x}{7+x}$

Module (6): Interpolation

Watch the online Video

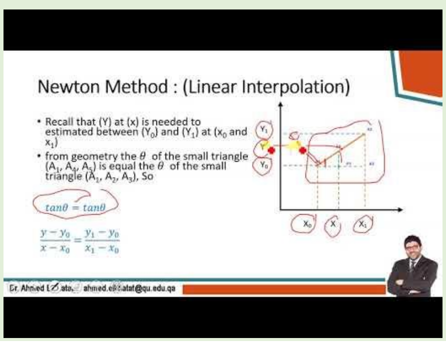


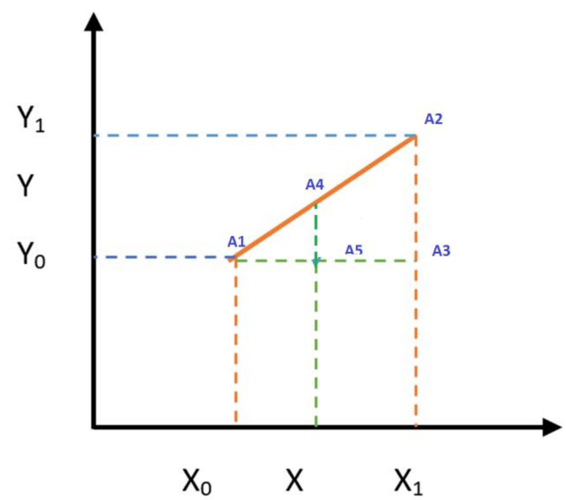
Interpolation is a common technique used to estimate the value of a function for any intermediate value of the independent variable.

1. Newton Method

A) Linear Interpolation

Watch the online Video





Recall that (Y) at (x) is needed to estimated between (Y_0) and (Y_1) at (x_0 and x_1)

from geometry the θ of the small triangle (A_1, A_4, A_5) is equal the θ of the small triangle (A_1, A_2, A_3), So

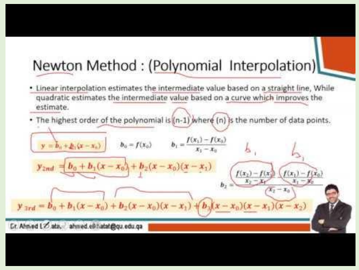
$$tan\theta = tan\theta$$

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = y_{0+}\frac{y_1 - y_0}{x_1 - x_0} * (x - x_0)$$

B) Quadratic (Polynomial) Interpolation

Watch the online Video



Linear interpolation estimates the intermediate value based on a straight line, While quadratic estimates the intermediate value based on a curve which improves the estimate. The highest order of the polynomial is (n-1), where (n) is the number of data points.

$$\boldsymbol{y_{1st} = b_0 + b_1(x - x_0)}$$

$$\boldsymbol{y_{2nd} = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)}$$

$$\boldsymbol{y_{3rd} = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + b_3(x - x_0)(x - x_1)(x - x_2)}$$

$$\boldsymbol{b_0 = f(x_0)}$$

$$\boldsymbol{b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}}$$

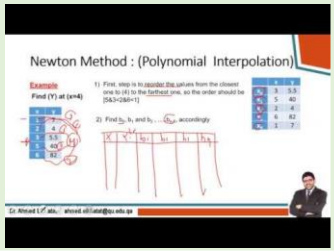
$$\boldsymbol{b_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}}$$

Note: it is very important to reorder the data set from the closest to farthest before applying this method

Example

find (Y) at (x=4)

Watch the online Video



x	y
1	7
2	4
3	5.5
5	40
6	82

(1) The first, step is to reorder the values from the closest one to (4) to the farthest one, so the order should be [5&3<2&6<1]

(2) So, the correct order is

	x	y
x_0	3	5.5
x_1	5	40
x_2	2	4
x_3	6	82
x_4	1	7

(3) Find b_0, b_1 and b_2 , ...b_n, accordingly

	x	b0=y	b1	b2	b3	b4
x_0	3	5.5	17.25	5.25	0.75	0
x_1	5	40	12	7.5	0.75	
x_2	2	4	19.5	4.5		
x_3	6	82	15			
x_4	1	7				

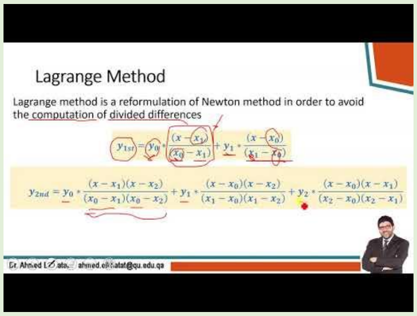
$$y_{1st} = b_0 + b_1(x - x_0)$$
$$= 5.5 + 17.25(4 - 3) = \mathbf{22.5}$$

$$y_{2nd} = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$
$$= 5.5 + 17.25(4 - 3) + 5.25(4 - 3)(4 - 5) = \mathbf{17.25}$$

$$y_{\ 3rd} = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + b_3(x - x_0)(x - x_1)(x - x_2)$$
$$5.5 + 17.25(4 - 3) + 5.25(4 - 3)(4 - 5) + 0.75(4 - 3)(4 - 5)(4 - 2) = \mathbf{15.75}$$

2. Lagrange Method

Watch the online Video



Lagrange method is a reformulation of the Newton method in order to avoid the computation of divided differences

$$y_{1st} = y_0 * \frac{(x - x_1)}{(x_0 - x_1)} + y_1 * \frac{(x - x_0)}{(x_1 - x_0)}$$

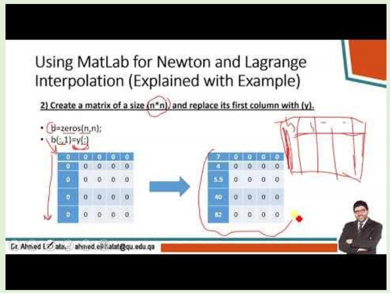
$$y_{2nd} = y_0 * \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 * \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 * \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

$$y\ 3rd = y_0 * \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} + y_1 * \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} + y_2 * \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} + y_3 * \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}$$

$$y4th = y_0 \frac{(x - x_1)(x - x_2)(x - x_3)(x - x_4)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)(x_0 - x_4)} + y_1 \frac{(x - x_0)(x - x_2)(x - x_3)(x - x_4)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} + y_2 \frac{(x - x_0)(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)} + y_3 \frac{(x - x_0)(x - x_1)(x - x_2)(x - x_4)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)} + y_4 \frac{(x - x_0)(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_0)(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)}$$

3. Using MatLab for Newton and Lagrange Interpolation

Watch the online Video



Algorithm for Newton Interpolation Method
General Formulas

$$f_{n-1(x)} = b_1 + b_2(x - x_1) + \cdots . + b_n(x - x_1) \ldots \big((x - x_{n-1})\big)$$

$b_1 = f_{(x1)}$

$b_2 = f[x_2, x_1] = \frac{f_{(x2)} - f_{(x1)}}{x_2 - x_1}$

$b_3 = f[x_3, x_2, x_1] = \frac{f[x_3,x_2] - f[x_2,x_1]}{x_3 - x_1} = \frac{\frac{f_{(x3)} - f_{(x2)}}{x_3 - x_2} - \frac{f_{(x2)} - f_{(x1)}}{x_2 - x_1}}{x_3 - x_1}$

$b_n = f[x_n, x_{n-1}, \ldots, x_1] = \frac{f[x_n,x_{n-1},\ldots,x_2] - f[x_{n-1},\ldots,x_1]}{x_n - x_1}$

$$f[x_i, x_j] = \frac{f_{(xi)} - f_{(xj)}}{x_i - x_j}$$

1) Check that both x and y have the same lenght (n).

- n=length(x); if length(y)~=n, error('x and y must have the same length'); end

x	y
1	7
2	4
3	5.5
5	40
6	82

2) Create a matrix of a size (n*n), and replace its first column with (y). For example,

- b=zeros(n,n);
- b(:,1)=y(:)

7	0	0	0	0
4	0	0	0	0
5.5	0	0	0	0
40	0	0	0	0
82	0	0	0	0

3) Now we want to calculate (b) in this matrix in specific locations as follows

	j=1	j=2	j=3	j=4	j=5
i=1	7	(1,2)	(1,3)	(1,4)	(1,5)
i=2	4	(2,2)	(2,3)	(2,4)	
i=3	5.5	(3,2)	(3,3)		
i=4	40	(4,2)			
i=5	82				

- Create (j) for columns (from 2 and till (n))
  - for each column (j) relate (i) for rows (from 1 and until n-j+1).
  - Then, b(i,j)=(b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i)));
  - To preform this loop in MatLab

```
for j=2:n
    for i=1:n-j+1
        b(i,j)=(b(i+1, j-1)-b(i,j-1))/(x(i+j-1)-x(i))
    end
end
```

7.00	0	0	0	0
4.00	0	0	0	0
5.50	0	0	0	0
40.00	0	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	0	0	0
4.00	0	0	0	0
5.50	0	0	0	0
40.00	0	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	0	0	0
4.00	1.50	0	0	0
5.50	0	0	0	0
40.00	0	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	0	0	0
4.00	1.50	0	0	0
5.50	17.25	0	0	0
40.00	0	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	0	0	0
4.00	1.50	0	0	0
5.50	17.25	0	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0	0
4.00	1.50	0	0	0
5.50	17.25	0	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0	0
4.00	1.50	5.25	0	0
5.50	17.25	0	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0	0
4.00	1.50	5.25	0	0
5.50	17.25	8.25	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0.75	0
4.00	1.50	5.25	0	0
5.50	17.25	8.25	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0.75	0
4.00	1.50	5.25	0.75	0
5.50	17.25	8.25	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

→

7.00	-3.00	2.25	0.75	0
4.00	1.50	5.25	0.75	0
5.50	17.25	8.25	0	0
40.00	42.00	0	0	0
82.00	0	0	0	0

4) Now we want to create loop for $(x - x_1) \dots ((x - x_{n-1}))$ that will be multiplied to (b)

- To perfom this loop in MatLab

```
%To identify xt and yi
xt=1; yi=b(1,1);
for j=1:n-1
xt=xt*(xi-x(j));
yi=yi+b(1,j+1)*xt
end
```

The Full function script is:

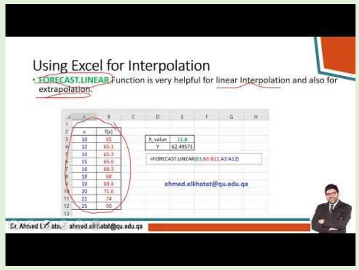
```
Editor
newtoninerpolation.m*
function yi=newtoninerpolation(x, y, xi)
%Newton Interpolation Fucntion
%x: indevendant variable vector.
%y: devendant variable vector.
%yi: debendant variable to be calculated.
%xi: %indebendant variable at which (yi) will be estimated

% To check the length of the two vectors (x and y)
n=length(x); if length(y)~=n, error('x and y must have the same length'); end

b=zeros(n,n); b(:,1)=y(:);
for j=2:n
    for i=1:n-j+1
        b(i,j)=(b(i+1, j-1)-b(i,j-1))/(x(i+j-1)-x(i));
    end
end
% to interpolate xi at given (yi)
xt=1; yi=b(1,1);
for j=1:n-1
xt=xt*(xi-x(j));
yi=yi+b(1,j+1)*xt;
end
```



```
Command Window
>> newtoninerpolation(x, y, 5)

ans =

   40.00

fx >>
```

4. Using Excel for Interpolation

Watch the online Video



A) FORCAST for linear interpolation

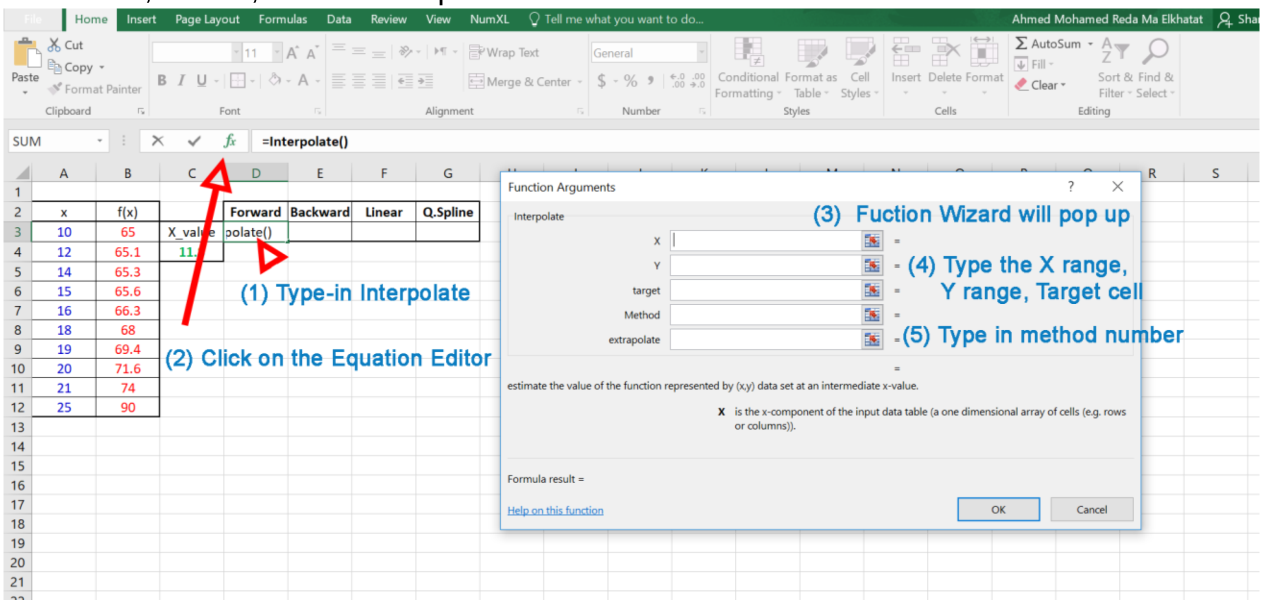
FORECAST.LINEAR function is beneficial for linear interpolation and also for extrapolation. The FORECAST.LINEAR function has the following syntax: =FORECAST.LINEAR (x,known_y's,known_x's)

	A	B	C	D	E	F	G	H
1								
2	x	f(x)						
3	10	65		X_value	11.8			
4	12	65.1		Y	62.49571			
5	14	65.3		=FORECAST.LINEAR(E3,B3:B12,A3:A12)				
6	15	65.6						
7	16	66.3						
8	18	68						
9	19	69.4		ahmed.elkhatat@qu.edu.qa				
10	20	71.6						
11	21	74						
12	25	90						
13								

B) INTEPOLATE for Advanced Interpolations

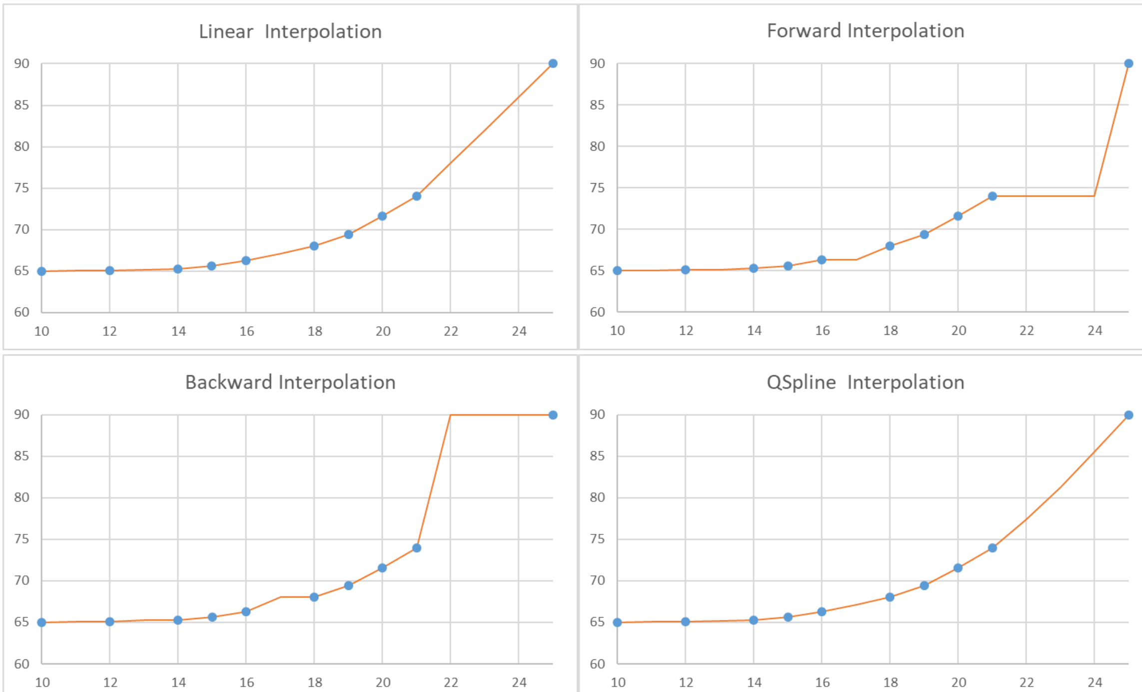
This Interpolate function is not a built-in function in Excel, but it comes with NumXL. You can only see the function after you install NumXL.(https://www.numxl.com/products/numxl)

Interpolate function in Excel offers four methods of Interpolations; Forward, Backward, Linear, and Cubic spline.



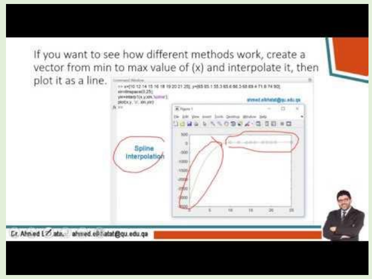
	A	B	C	D	E	F	G
1							
2	x	f(x)		Forward	Backward	Linear	Q.Spline
3	10	65	X_value	65.0000	65.0900	65.0900	65.0872
4	12	65.1	11.8				
5	14	65.3		Forcast Function		62.49571	
6	15	65.6					
7	16	66.3					
8	18	68					
9	19	69.4					
10	20	71.6					
11	21	74					
12	25	90					

You may ask why different interpolation methods give different values ?. The answer is because it depends on how the interpolation method finds f(x) based on (x). The following figures show how each interpolation method can estimate the unknown variable.



5. Using MatLab build-in functions for interpolation

Watch the online Video



MatLab build-in function **Interp1** is an outstanding tool for interpolating a set of data. The general syntax is:

interp1(x, y, x_i)→ x and y are vectors, uses linear interpolation to estimate (x_i)

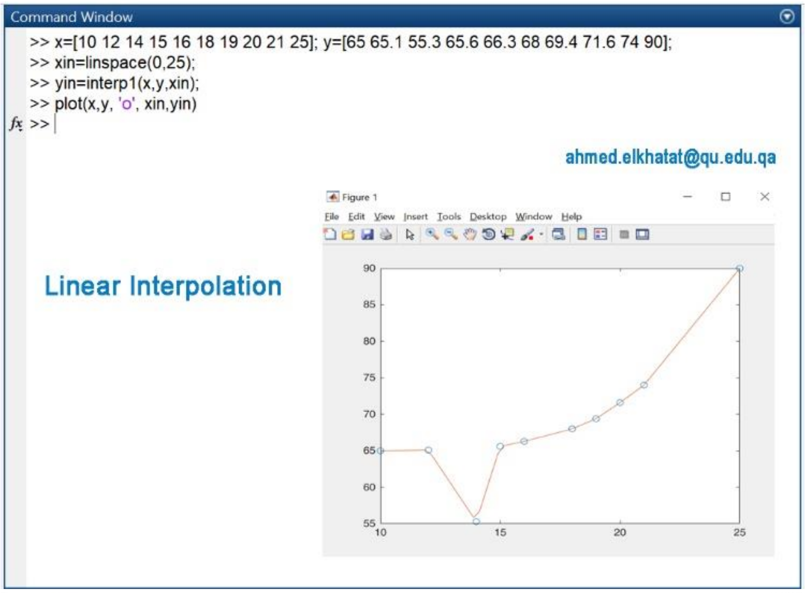
interp1((x, y, x_i, 'method')

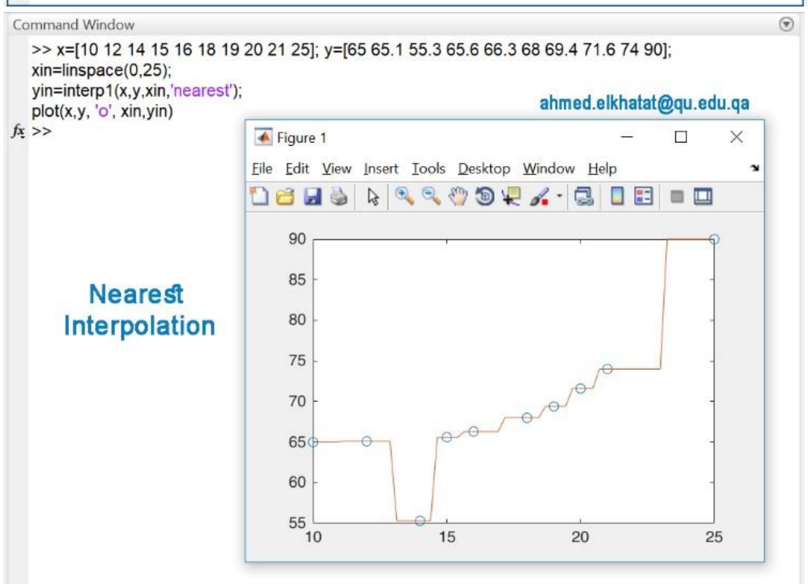
Method	Description
'linear'	(default) linear interpolation
'nearest'	nearest-neighbor interpolation
'next'	next neighbor interpolation
'previous'	previous neighbor interpolation
'spline'	piecewise cubic spline interpolation (SPLINE)
'pchip'	shape-preserving piecewise cubic interpolation
'cubic'	same as 'pchip'
'v5cubic'	the cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.
'makima'	modified Akima cubic interpolation

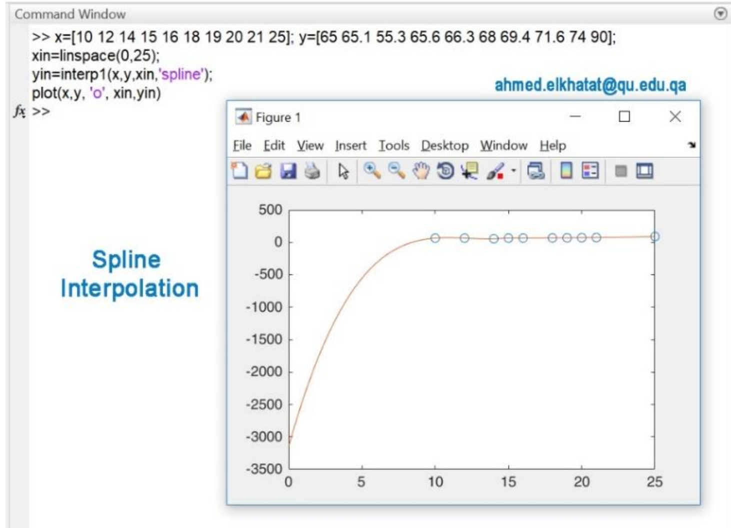
Command Window

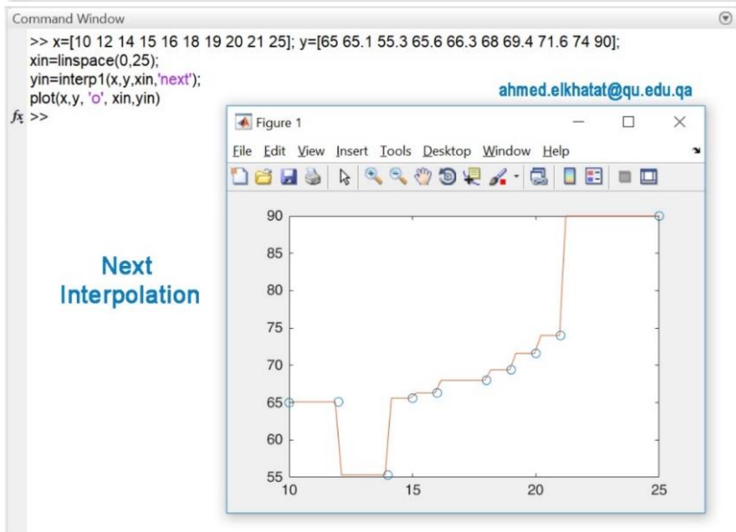
```
>> x=[10 12 14 15 16 18 19 20 21 25]; y=[65 65.1 55.3 65.6 66.3 68 69.4 71.6 74 90];
>> Lin=interp1(x,y,11.8)

Lin =

       65.09

>> Near=interp1(x,y,11.8,'nearest')

Near =

       65.10

>> Spline=interp1(x,y,11.8,'spline')

Spline =

       66.98
```

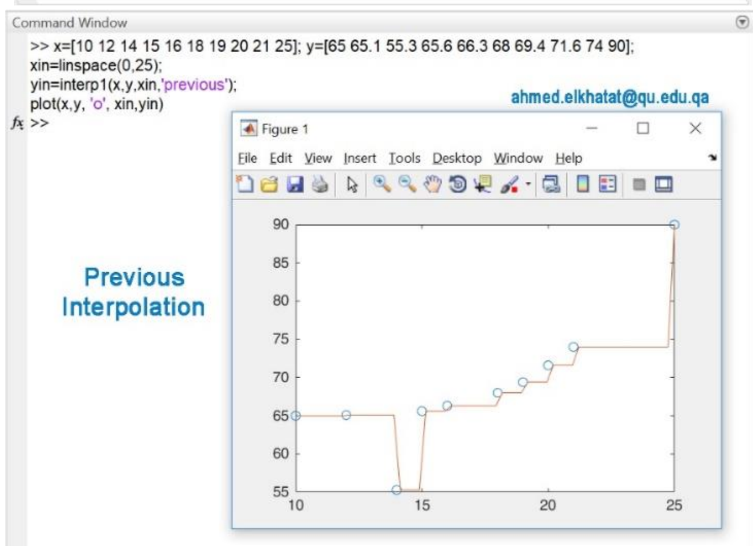
If you want to see how different methods work, create a vector from min to max value of (x), interpret it, and plot it as a line.





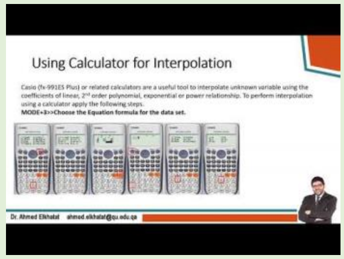

Command Window
>> x=[10 12 14 15 16 18 19 20 21 25]; y=[65 65.1 55.3 65.6 66.3 68 69.4 71.6 74 90];
xin=linspace(0,25);
yin=interp1(x,y,xin,'spline');
plot(x,y, 'o', xin,yin)
>>
ahmed.elkhatat@qu.edu.qa
Figure 1
File Edit View Insert Tools Desktop Window Help
Spline Interpolation
500
0
-500
-1000
-1500
-2000
-2500
-3000
-3500
0
5
10
15
20
25


Command Window
>> x=[10 12 14 15 16 18 19 20 21 25]; y=[65 65.1 55.3 65.6 66.3 68 69.4 71.6 74 90];
xin=linspace(0,25);
yin=interp1(x,y,xin,'next');
plot(x,y, 'o', xin,yin)
>>
ahmed.elkhatat@qu.edu.qa
Figure 1
File Edit View Insert Tools Desktop Window Help
Next Interpolation
90
85
80
75
70
65
60
55
10
15
20
25


Command Window
>> x=[10 12 14 15 16 18 19 20 21 25]; y=[65 65.1 55.3 65.6 66.3 68 69.4 71.6 74 90];
xin=linspace(0,25);
yin=interp1(x,y,xin,'previous');
plot(x,y, 'o', xin,yin)
>>
ahmed.elkhatat@qu.edu.qa
Figure 1
File Edit View Insert Tools Desktop Window Help
Previous Interpolation
90
85
80
75
70
65
60
55
10
15
20
25

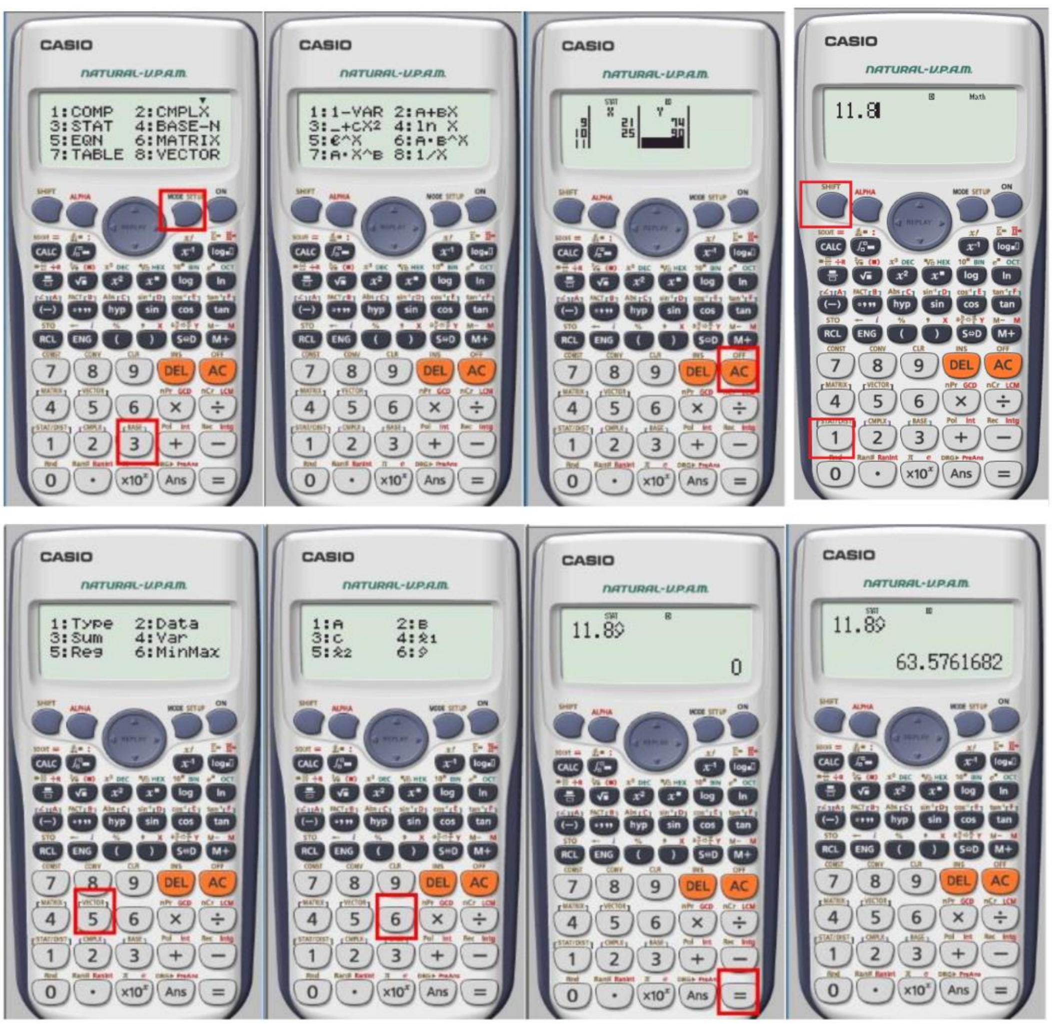
6. Using Calculator for Interpolation

Watch the online Video



Casio (fx-991ES Plus) or related calculators are helpful tools for interpolating unknown variables using the coefficients of linear, 2nd order polynomial, exponential, or power relationships. To perform interpolation using a calculator, apply the following steps.

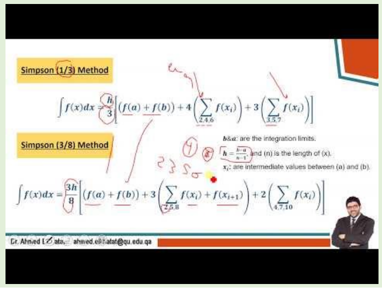
MODE+3>>Choose the Equation formula for the data set.



Module (7): Integration and Differentiation

1. Integration

Watch the online Video



General Formulas

Different methods integrate a function numerically, but Trapezoidal and Simpson Methods are the most commonly used.

The general formulas for these methods are

Composite Trapezoidal Method

The trapezoidal method is useful for approximating the definite integral, which is the average of the left and right sums. It also provides a more accurate approximation than either method on its own

$$\int f(x)dx = \frac{h}{2}[f(a) + f(b)] + h\left[\sum f(x_i)\right]$$

$$\int f(x)dx = \frac{h}{2}\left[f(a) + f(b) + 2\left(\sum f(x_i)\right)\right]$$

where

$b\&a$: are the integration limits.

$h = \frac{b-a}{n-1}$, and (n) is the length of (x).

x_i: are intermediate values between (a) and (b).

Simpson (1/3) Method

Simpson's 1/3 rule uses a quadratic approximation to approximate the 2nd order polynomial. This means that each approximation covers two subintervals. Therefore, it is necessary to have an even number of subintervals. This rule is applied where N is an even number.

$$\int f(x)dx = \frac{h}{3}\left[\left(f(a)+f(b)\right)+4\left(\sum_{2,4,6} f(x_i)\right)+2\left(\sum_{3,5,7} f(x_i)\right)\right]$$

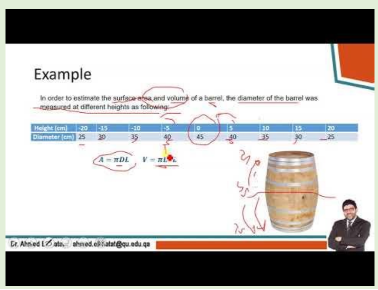
Simpson (3/8) Method

Simpson's 3/8 rule uses a quadratic approximation to approximate the 3rd order polynomial. This means that each approximation covers three subintervals. Therefore, it is necessary to have an even number of subintervals. This rule is applied where N is a multiple of 3.

$$\int f(x)dx = \frac{3h}{8}\left[\left(f(a)+f(b)\right)+3\left(\sum_{2,5,8} f(x_i)+f(x_{i+1})\right)+2\left(\sum_{4,7,10} f(x_i)\right)\right]$$

Example

Watch the online Video



To estimate the surface area and volume of a barrel, the diameter of the barrel was measured at different heights as follows:

Height (cm)	-20	-15	-10	-5	0	5	10	15	20
Diameter (cm)	25	30	35	40	45	40	35	30	25

$$A = \pi DL \qquad V = \pi D^2 L$$

Find both Area and Volume using a) composite trapezoids method, b) Simpson (1/3) method and c) Simpson (3/8) method.

A) Composite Trapezoidal Method

$$\int \boldsymbol{f(x)dx} = \frac{\boldsymbol{h}}{\boldsymbol{2}}[\boldsymbol{f(a)} + \boldsymbol{f(b)}] + \boldsymbol{h}\left[\sum \boldsymbol{f(x_i)}\right]$$

$$h = \frac{20-(-20)}{9-1} = 5$$

$$A = \pi \int DdL$$

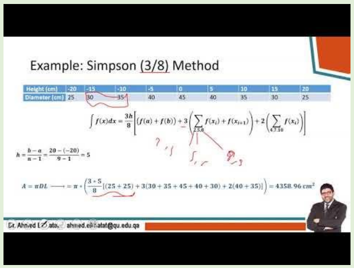
$$A = \pi\left(\frac{5}{2} * [25+25] + 5 * [(30+35+40+45+35+30)]\right) = 4398.23\ cm^2$$

$$V = \pi \int D^2 dL$$

$$V = \pi\left(\frac{5}{2} * [25^2+25^2] + 5 * [(30^2+35^2+40^2+45^2+40^2+\ 35^2+30^2)]\right)$$
$$= 158650.429\ cm^3$$

B) Simpson (1/3) Method

Watch the online Video



$$\int \boldsymbol{f(x)dx} = \frac{\boldsymbol{h}}{\boldsymbol{3}}\left[\left(\boldsymbol{f(a)} + \boldsymbol{f(b)}\right) + \boldsymbol{4}\left(\sum_{\boldsymbol{2,4,6}} \boldsymbol{f(x_i)}\right) + \boldsymbol{2}\left(\sum_{\boldsymbol{3,5,7}} \boldsymbol{f(x_i)}\right)\right]$$

$$A = \pi * \left(\frac{5}{3}[(25+25) + 4(30+40+40+30) + 2(35+45+35)]\right) = \pi * 1400$$
$$= 4{,}398.2\ cm^2$$

$$V = \pi * \left(\frac{5}{3}[(25^2+25^2) + 4(30^2+40^2+40^2+30^2) + 2(35^2+45^2+35^2)]\right) = \pi * 50{,}333$$
$$= 158{,}130\ cm^3$$

C) Simpson (3/8) Method

$$\int f(x)dx = \frac{3h}{8}\left[\left(f(a)+f(b)\right)+3\left(\sum_{2,5,8} f(x_i)+f(x_{i+1})\right)+2\left(\sum_{4,7,10} f(x_i)\right)\right]$$

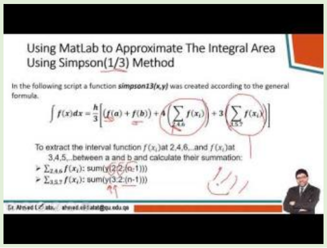
$$A = \pi * \left(\frac{3*5}{8}[(25+25)+3(30+35+45+40+30)+2(40+35)]\right) = \pi * 1387.5 = 4358.96\ cm^2$$

$$V = \pi * \left(\frac{3*5}{8}[(25^2+25^2)+3(30^2+35^2+45^2+40^2+30^2)+2(40^2+35^2)]\right) = \pi * 50343.75 = 158159.55\ cm^3$$

	Area cm^2	Volume cm^3
Composite Trapezoidal Method	4,398.23	158,650.429
Simpson (1/3) Method	4,398.2	158,130
Simpson (3/8) Method	4,358.96	158,159.55

D) Using MatLab to Approximate The Integral Area Using Different Numerical Methods

Watch the online Video

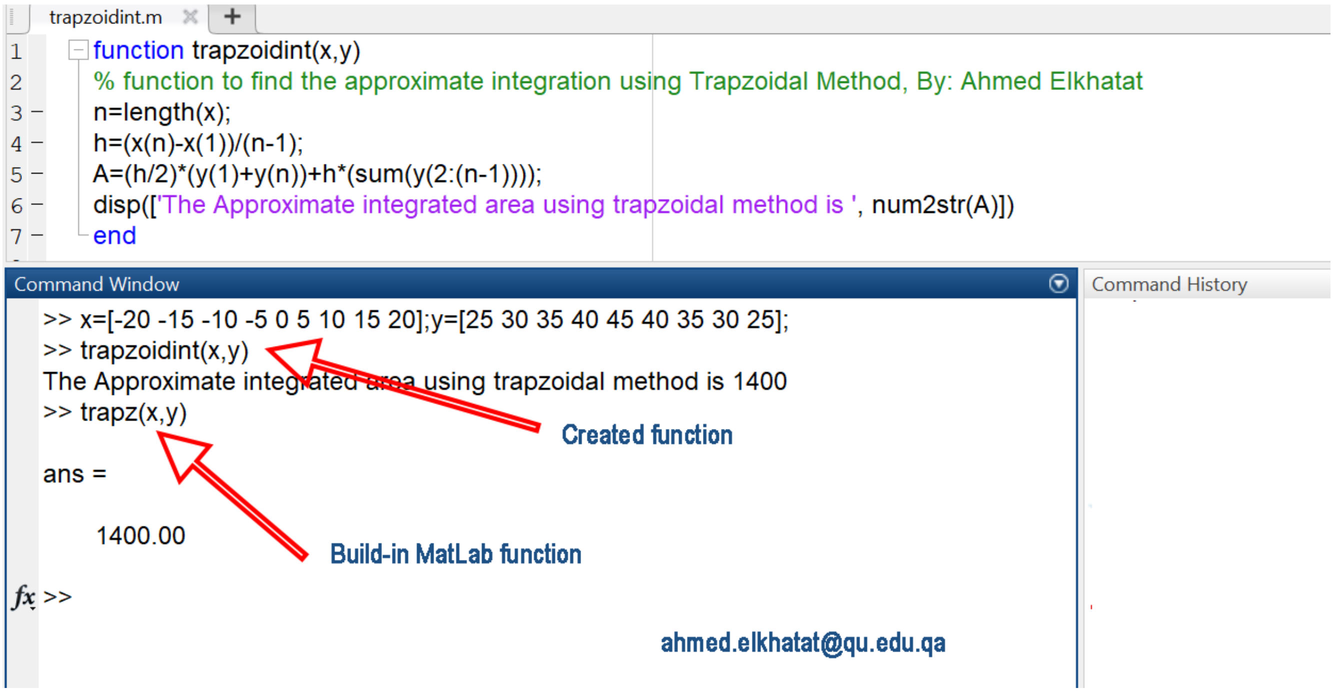


i. Integration Using Trapezoidal Method

In the following script, a function ***trapzoidint(x,y)*** was created according to the general formula.

$$\int \boldsymbol{f(x)dx} = \frac{\boldsymbol{h}}{\boldsymbol{2}}[\boldsymbol{f(a)}+\boldsymbol{f(b)}]+\boldsymbol{h}\left[\sum \boldsymbol{f(x_i)}\right]$$

The result of the created function ***trapzoidint(x,y)*** was compared with a MatLab build-in functions ***trapz(x,y)*** that gives the same value.



Note that: The function here doesn't include (π) or D^2. So your codes must be customized for this example

ii. Integration Using Simpson(1/3) Method

In the following script, a function ***simpson13(x,y)*** was created according to the general formula.

$$\int f(x)dx = \frac{h}{3}\left[\left(f(a)+f(b)\right)+4\left(\sum_{2,4,6} f(x_i)\right)+2\left(\sum_{3,5,7} f(x_i)\right)\right]$$

to extract the interval function $f(x_i)$at 2,4,6,..and $f(x_i)$at 3,4,5,..between a and b and calculate their summation:

- $\sum_{2,4,6} f(x_i)$: sum(y(2:2:(n-1)))
- $\sum_{3,5,7} f(x_i)$: sum(y(3:2:(n-1)))

```
simpson13.m

function simpson13(x,y)
% function to find the approximate integration using Simpson(1/3) Method, By: Ahmed Elkhatat
n=length(x);
h=(x(n)-x(1))/(n-1);
A=(h/3)*((y(1)+y(n))+4*(sum(y(2:2:(n-1))))+2*(sum(y(3:2:(n-1)))));
disp(['The Approximate integrated area using Simpson(1/3) method is ', num2str(A)])
end
```

```
Command Window
New to MATLAB? See resources for Getting Started.
>> x=[-20 -15 -10 -5 0 5 10 15 20];
>> y=[25 30 35 40 45 40 35 30 25];
>> simpson13(x,y)
The Approximate integrated area using Simpson(1/3) method is 1400
>> Area_Barrel=pi*1400

Area_Barrel =
```

Note that: The function here doesn't include (π) or D^2. So your codes must be customized for this example

iii. Integration Using Simpson(3/8) Method

In the following script, a function ***simpson38(x,y)*** was created according to the general formula.

$$\int f(x)dx = \frac{3h}{8}\left[\left(f(a)+f(b)\right)+3\left(\sum_{2,5,8} f(x_i)+f(x_{i+1})\right)+2\left(\sum_{4,7,10} f(x_i)\right)\right]$$

Note that

$$\sum_{2,5,8} f(x_i)+f(x_{i+1}) = \sum_{2,5,8} f(x_i)+\sum_{3,6,9} f(x_{i+1})$$

Thus, to extract the interval functions :

- $\sum_{2,5,8} f(x_i)$: sum(y(2:3:(n-1)))
- $\sum_{3,6,9} f(x_i)$: sum(y(3:3:(n-1)))
- $\sum_{4,7,10} f(x_i)$: sum(y(4:3:(n-1)))

simpson38.m

```
function simpson38(x,y)
% function to find the approximate integration using Simpson(3/8) Method, By: Ahmed Elkhatat
n=length(x);
h=(x(n)-x(1))/(n-1);
A=(3*h/8)*((y(1)+y(n))+3*(sum(y(2:3:(n-1)))+sum(y(3:3:(n-1))))+2*(sum(y(4:3:(n-1)))));
disp(['The Approximate integrated area using Simpson(3/8) method is ', num2str(A)])
end
```

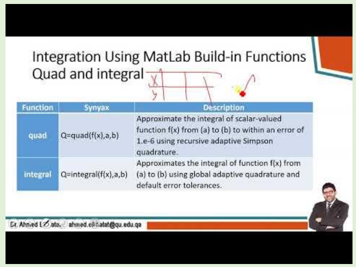
Command Window

```
>> x=[-20 -15 -10 -5 0 5 10 15 20];y=[25 30 35 40 45 40 35 30 25];
>> simpson38(x,y)
The Approximate integrated area using Simpson(3/8) method is 1387.5
>>
```


Note that: The function here doesn't include (π) or D^2. So your codes must be customized for this example

E) Integration Using MatLab Built-in Functions

Watch the online Video



i. Integration using (*Trapz and cumtrapz*)

MatLab offers some useful built-in functions for integration purposes.

1) ***trapz(x,y)***: Computes the integral of Y with respect to X using the trapezoidal method.
2) ***cumtrapz(x,y)*** computes the cumulative integral of Y with respect to X using trapezoidal integration.

Command Window

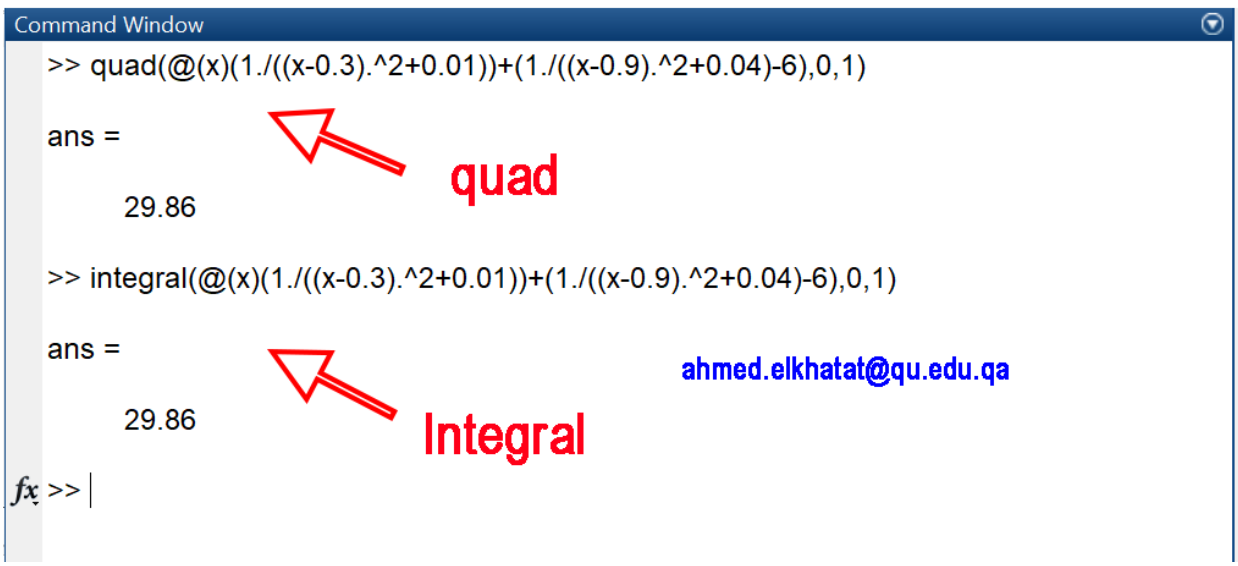
```
>> x=[-20 -15 -10 -5 0 5 10 15 20];y=[25 30 35 40 45 40 35 30 25];
>> trapz(x,y)

ans =

    1400.00

>> cumtrapz(x,y)

ans =

  Columns 1 through 4

         0     137.50     300.00     487.50

  Columns 5 through 8

    700.00     912.50    1100.00    1262.50

  Column 9

    1400.00

fx >>
```

ii. Integration Using MatLab Built-in Functions Quad and integral

Function	Syntax	Description
quad	Q=quad(f(x),a,b)	Approximate the integral of scalar-valued function f(x) from (a) to (b) to within an error of 1.e-6 using recursive adaptive Simpson quadrature.
integral	Q=integral(f(x),a,b)	Approximates the integral of function f(x) from (a) to (b) using global adaptive quadrature and default error tolerances.

Example: Use Quad and Integral MatLab function to integrate the following function

$$\int_0^1 \left(\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6\right)dx$$



(3)**Differentiation:**

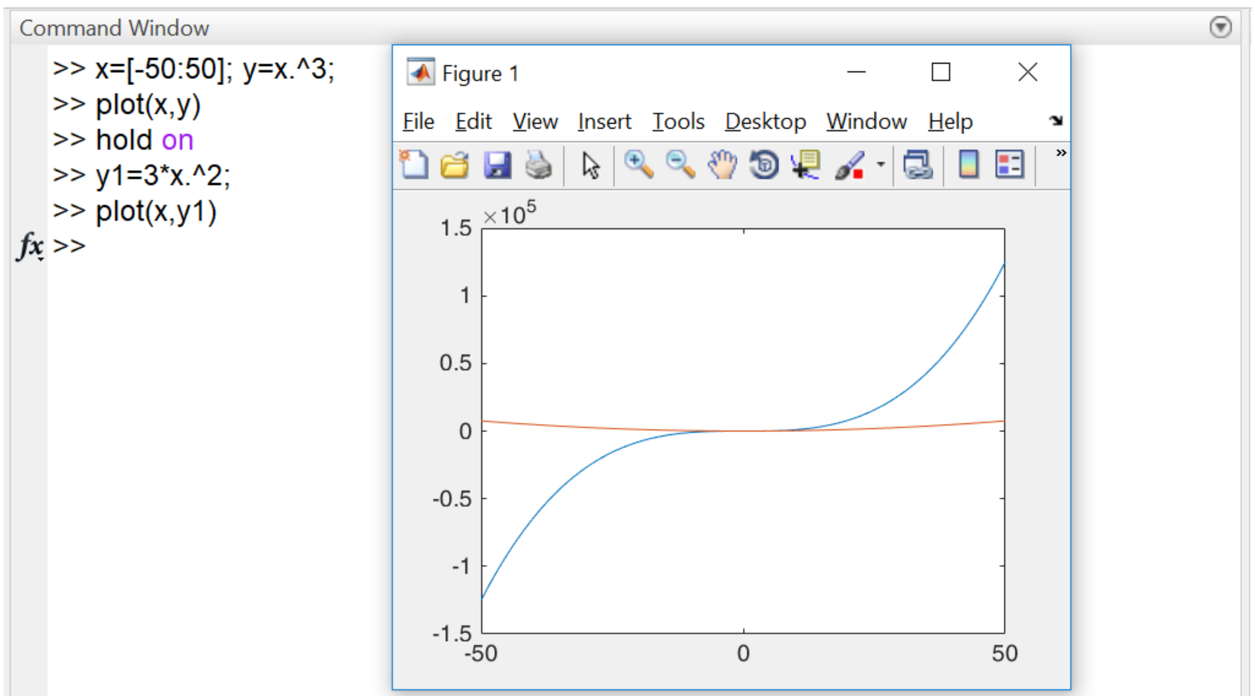
Mathematically, the derivative of a function can be presented as shown in the below equation:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}$$

a. Using MatLab'diff's function to differentiate a function

Note that (diff) command stands for difference, not differentiation. The first derivative of a given function is defined as the change of (y) value with respect to the shift in the (x) values, which is known as (Slope) or (Gradient) of the function at a given location.

For example, the blue curve represents [**y=x³**], where (x) is a vector from -50 to 50. While the red curve represents the real derivation of this equation [**y=3x²**]



$$\frac{dy}{dx} \cong \frac{\Delta y}{\Delta x}$$

diff(y) command in MatLab can compute dy, and the diff(x) command in can compute dx, then $\frac{dy}{dx}$ can be computed as dy./dx

Example: Use 'diff' function to differentiate the following function from (-50) to (50)

$$\boldsymbol{f(x) = x^3}$$

A) Steps to compute the differentiation

- Insert the equation in Matlab Command window as:
  f=@(x) x.^3
- Generate (x) vector from -50 to 50
  x=[-50:50]
- Identify (y)=f(x)

y=f(x)

- Compute the differentiation

dydx=diff(y)./diff(x)

Steps to plot the differentiation

vector 'dydx' contains derivative estimates corresponding to the X(midpoint) between adjacent elements of x.

$$x_{midpoint} = \frac{x_i + x_{i+1}}{2}$$

- So, to generate (x_m) vector corresponding to (dydx)

n=length(x)

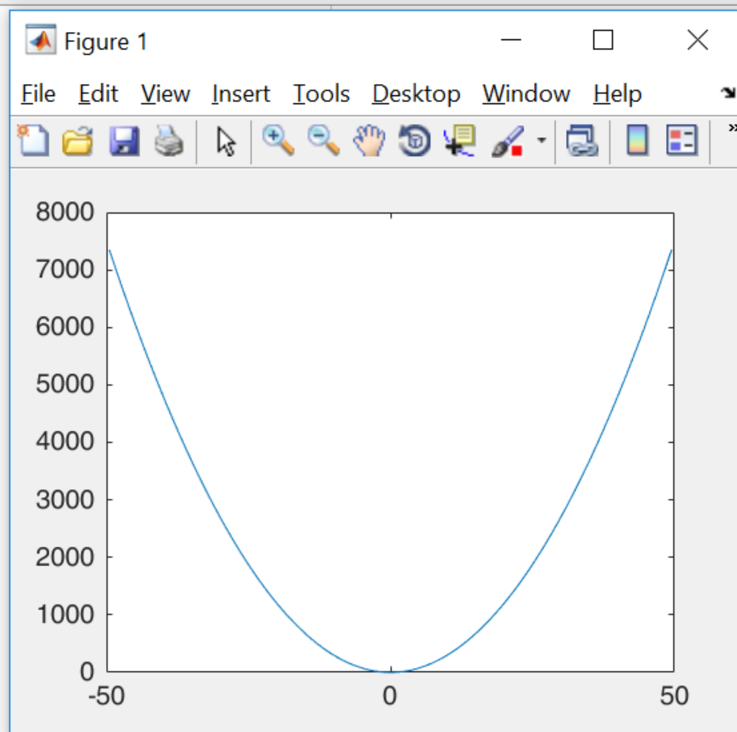
xm=(x(1:n-1)+x(2:n))./2

dydx=diff(y)./diff(x)

plot (xm,dydx)

test.m

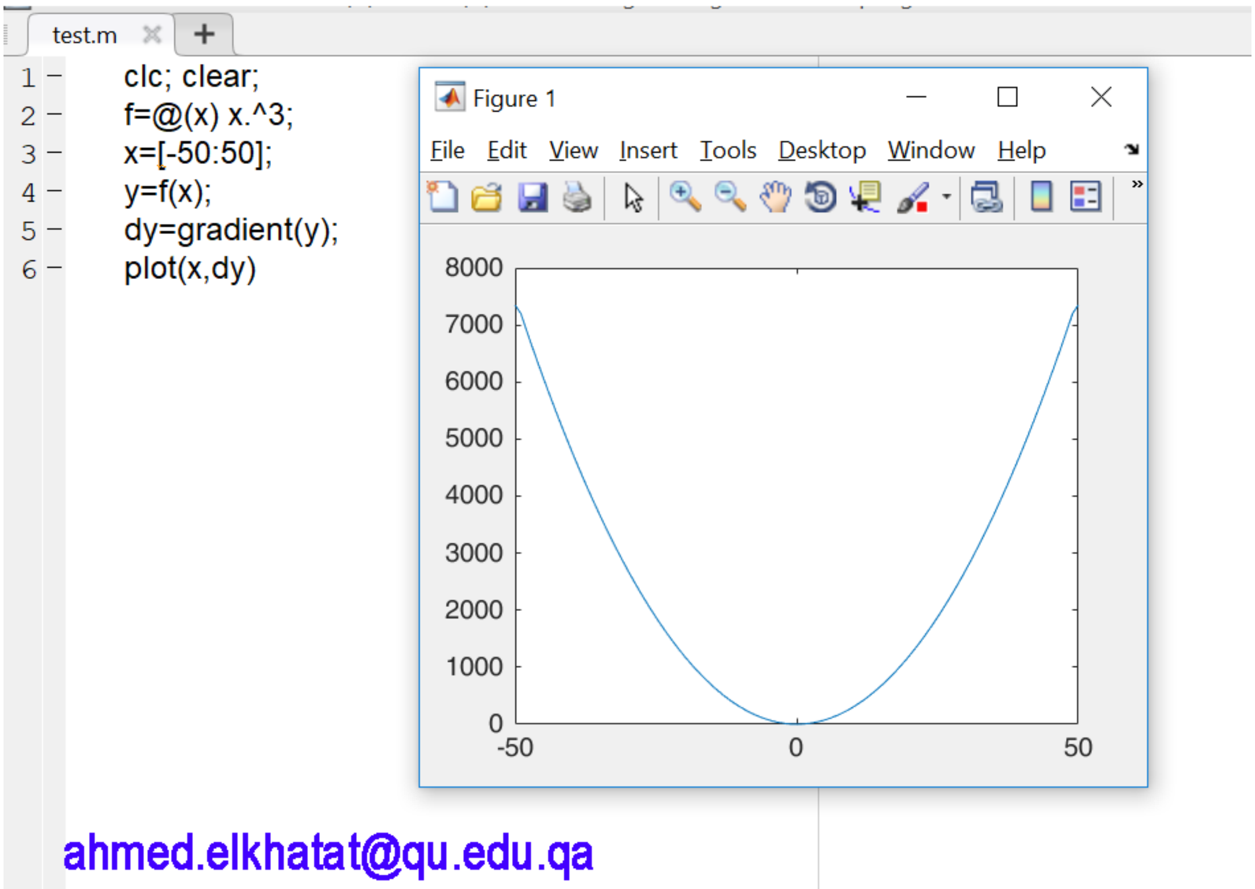
```
clc; clear;
f=@(x) x.^3;
x=[-50:50];
y=f(x);
dy=diff(y);
dx=diff(x);
dydx=dy./dx;
n=length(x);
xm=(x(1:n-1)+x(2:n))./2;
plot(xm,dydx)
```



ahmed.elkhatat@qu.edu.qa

B) Using MatLab 'gradient' function to differentiate a function

The MatLab function "gradient" is similar to 'diff' but evaluates the derivatives at the values themselves rather than in the intervals between the values.



C) Ordinary Differential Equation ODE:

i. Solving Single ODE

MatLab built-in function (ODE) is a helpful tool to solve ordinary differential equations. ODE

ODE23: Uses 2^{nd} and 3rd Runge–Kutta methods to solve ODE and make error estimates for step size judgment.

ODE45: Uses 4^{th} and 5^{th} Runge–Kutta methods to solve ODE and make error estimates for step size judgment. This function is recommended to apply as a first try.

ODE113: It is useful for stringent error tolerances

The simplest syntax of this function is

$$[t, y] = ode45\ (\,odefun, tspan, yo)$$

- (y): is the solution array, where each column is one of the dependent variables and each row corresponds to a time in the column vector (t).
- (odefun): is the name of the function returning a column vector of the RHS of the differential equations.
- (tspan): specifies the integration interval.
- (y0): is a vector containing the initial values.

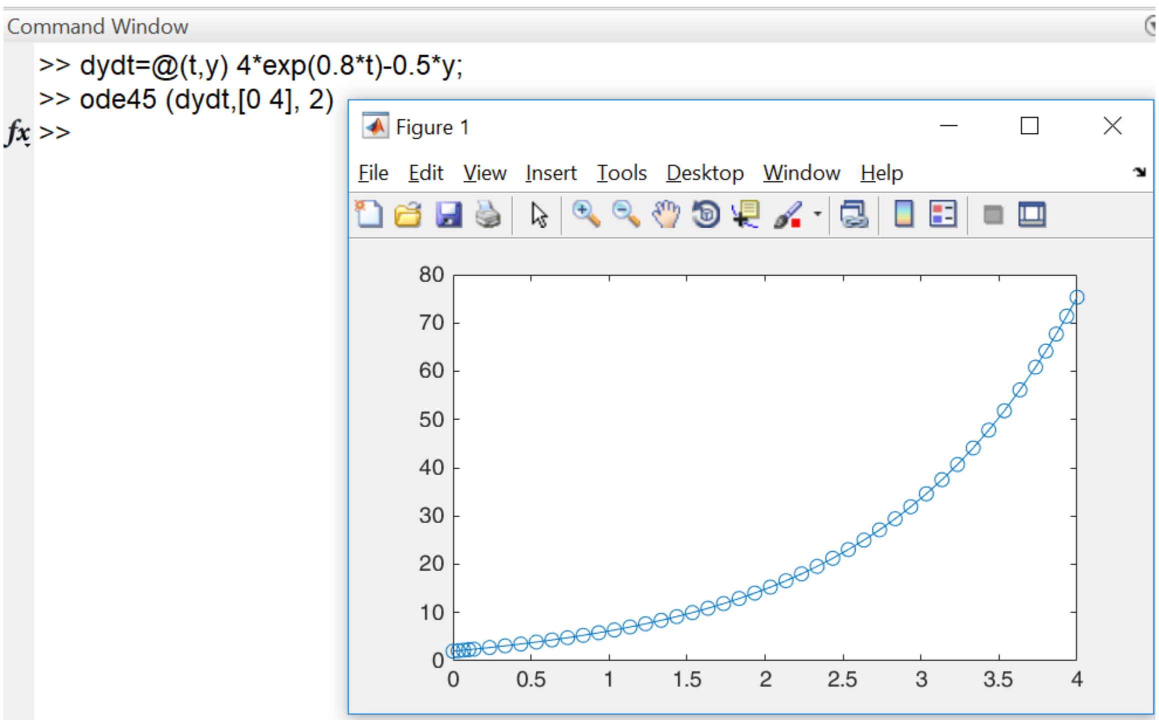
Example: Solve the following ordinary differential equation using MatLab from t =0-4, and initial y =2.

$$\frac{dy}{dt} = 4e^{0.8t} - 0.5y$$

(1) Create the function equation

dydt=@(t,y) 4*exp(0.8*t)-0.5*y;

(2) ode45 function can be employed as folowing



(3) But, too extract the value of (y) at the n^{th} of (t), so employ the ode45 function as folowing

[t,y]=ode45 (dydt,[0 4], 2);

(4) This will create [t,y] as following;

t	0.00	0.03	0.07	0.10	0.13	0.23	0.33	0.43	0.53	0.63	0.73
y	2.00	2.10	2.20	2.31	2.42	2.75	3.11	3.49	3.89	4.33	4.79

↓

t	0.83	0.93	1.03	1.13	1.23	1.33	1.43	1.53	1.63	1.73	1.83
y	5.29	5.82	6.39	7.01	7.68	8.39	9.16	10.00	10.90	11.87	12.91

↓

t	1.93	2.03	2.13	2.23	2.33	2.43	2.53	2.63	2.73	2.83	2.93
y	14.05	15.27	16.59	18.02	19.57	21.25	23.06	25.02	27.14	29.44	31.92

↓

t	3.03	3.13	3.23	3.33	3.43	3.53	3.63	3.73	3.80	3.87	3.93
y	34.62	37.53	40.69	44.10	47.80	51.81	56.15	60.85	64.19	67.71	71.42

↓

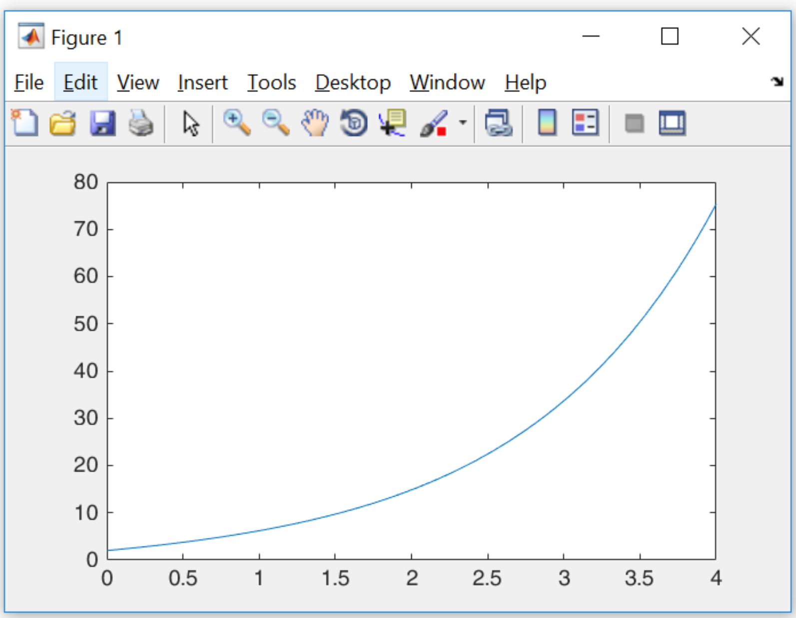
t	4.00
y	75.34

(5) To extract the value of (y) at the nth of (t)

y(length(t))

Command Window

```
>> dydt=@(t,y) 4*exp(0.8*t)-0.5*y;
>> [t,y]=ode45 (dydt,[0 4], 2);
>> y(length(t))

ans =

      75.34

>> plot (t,y)
fx >>
```



ii. Controlling ODE Options

Tolerance, initial step, or maximum step can be adjusted using **odeset** build-in function.
The syntax of this function is as follows

$$[t, y] = ode45\ (\ odefun, tspan, yo, options, p1, p2, \dots .)$$

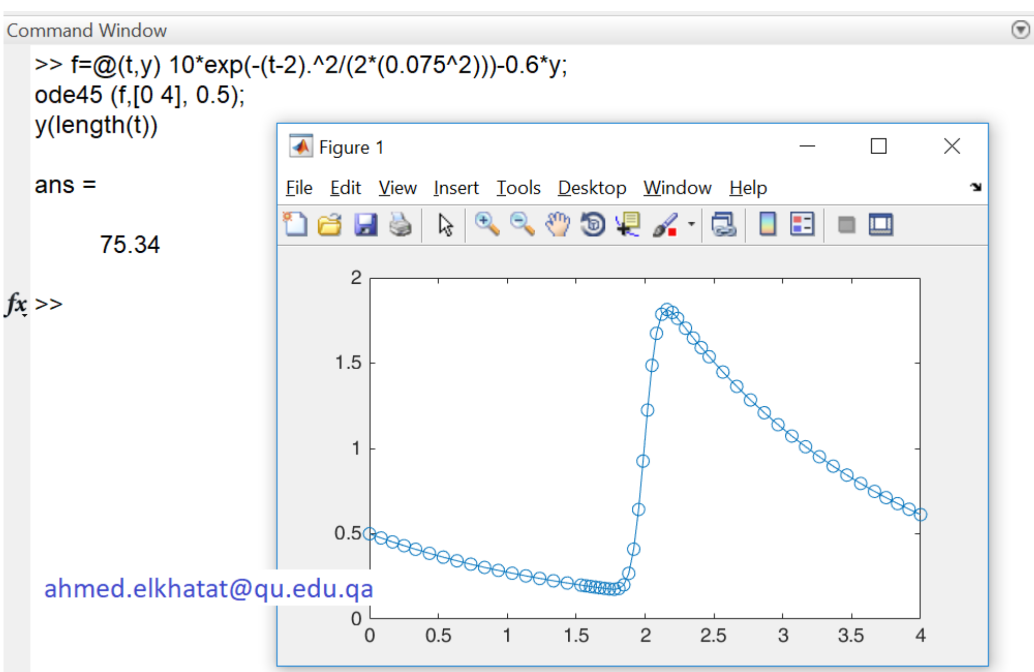
$$options = odest\ ('par_1', val_1, 'par_2', val_2..)$$

Parameter

- **RelTol:** To adjust the relative tolerance.
- **AbsTol:** To adjust the absolute tolerance.
- **InitialStep:** To allow the user to set their own initial step.
- **MaxStep:** To allow the user to set their own maximum step.

Example: Solve the following ordinary differential equation using MatLab from t =0-4, and initial y =0.5

$$\frac{dy}{dt} = 10e^{-\frac{(t-2)^2}{2(0.075)^2}} - 0.6y$$



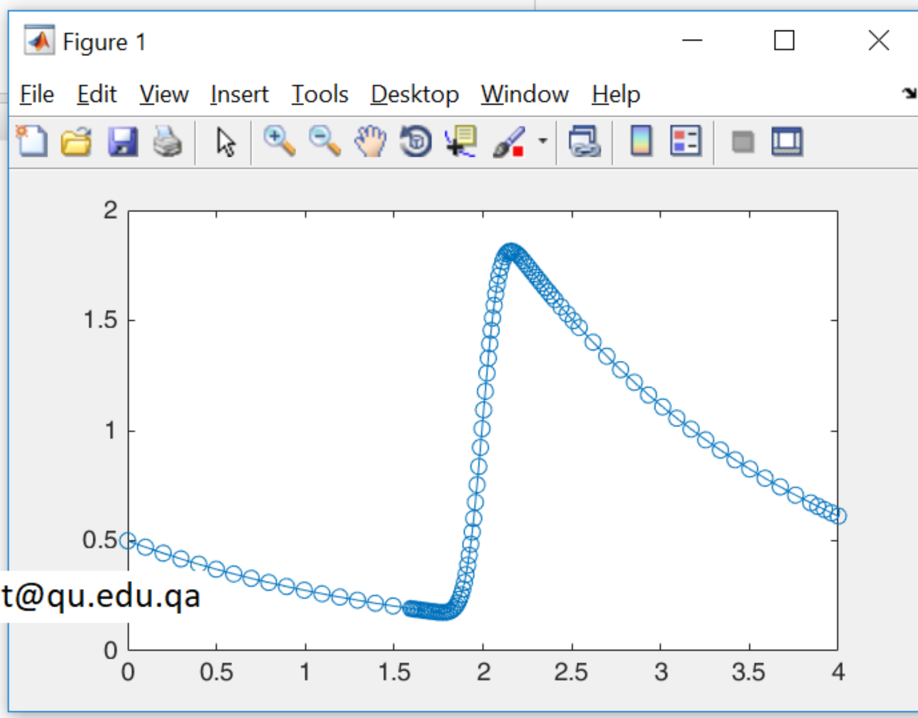
In the previous example the solution has large steps in the smooth region, while has small steps in the rapid region, this increase the relative error. In order to obtain a more accurate result, the ‘odeset’ function can be used to set the relative error tolerance to 10^{-10}

Untitled.m

```
clc
f=@(t,y) 10*exp(-(t-2).^2/(2*(0.075^2)))-0.6*y;
option=odeset('RelTol',1e-10);
ode45 (f,[0 4], 0.5, option);
y(length(t))
```

Command Window

```
ans =

      75.34

fx >>
```



iii. Solving System of ODEs

Example: Solve the following ordinary differential equation using MatLab from t =0-20, and initial y_1 =2 &y_2=1

$$\frac{dy_1}{dt} = 1.2y_1 - 0.6y_1y_2$$

$$\frac{dy_2}{dt} = -0.8y_2 + 0.3y_1y_2$$