Introduction to MATLAB
MTH643

Virtual University of Pakistan

Edited by Mahar Afaq Safdar Muhammadi
MSc Math & MA Islamic Studies
WhatsApp No. 03494873115
Gmail : maharafaq789@gmail.com

What is MATLAB

MATLAB is a programming and numeric computing platform used by engineers and scientists to analyze data, develop algorithms, and create models.

It is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Industry Applications

Automated Driving Systems

Design, simulate, and test automated driving systems

Computational Biology

Analyze, visualize, and model biological data and systems

Control Systems

Design, test, and implement control systems

Image Processing and Computer Vision

Acquire, process, and analyze images and video for algorithm development and system design

Internet of Things

Connect embedded devices to the Internet and gain insight from your data

Machine Learning

Train models, tune parameters, and deploy to production or the edge

Data Science

Explore data; build machine learning models; do predictive analytics

Deep Learning

Data preparation, design, simulation, and deployment for deep neural networks

Embedded Systems

Design, code, and verify embedded systems

Enterprise and IT Systems

Use MATLAB with your IT systems

Mechatronics

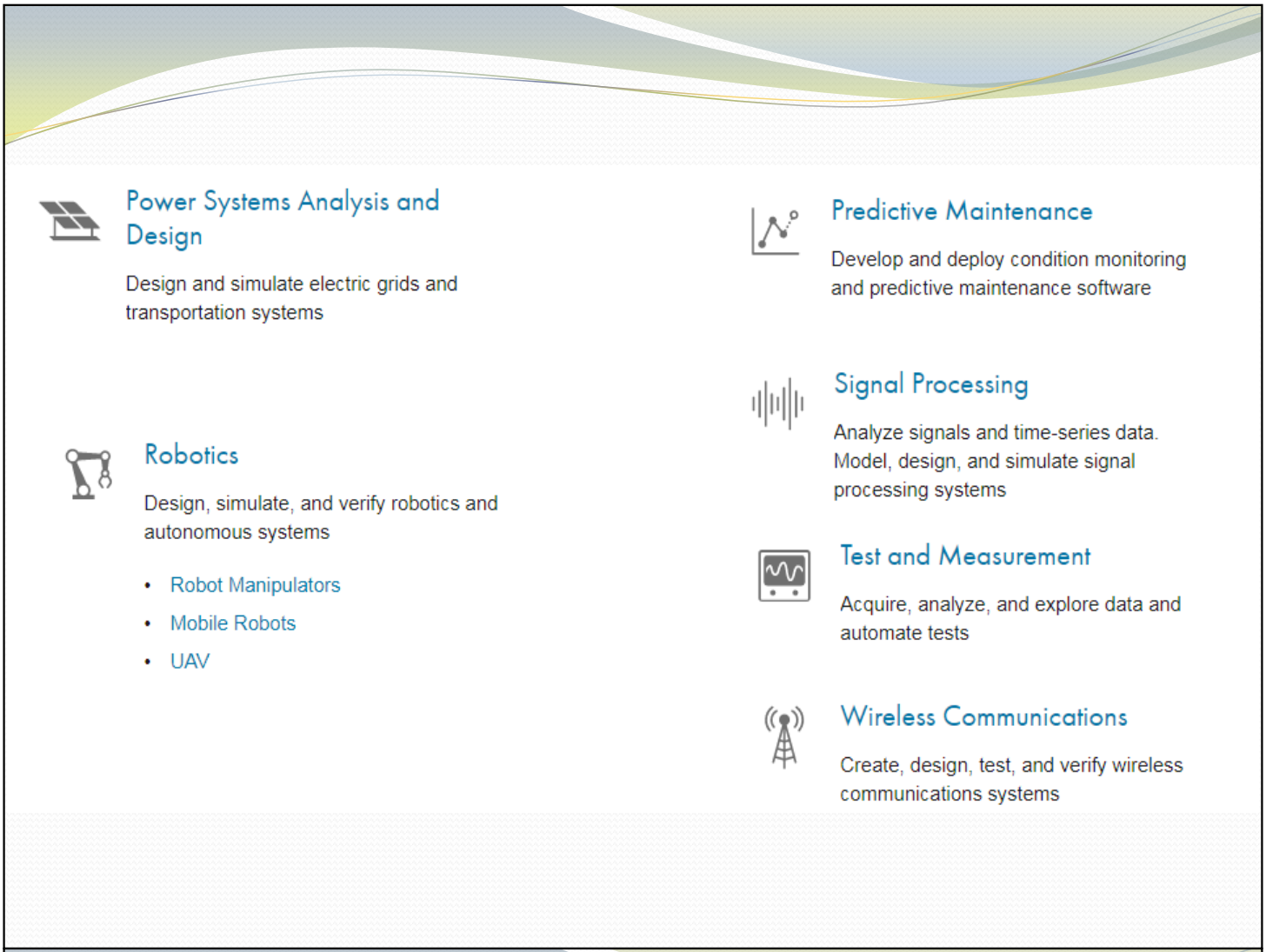
Design, optimize, and verify mechatronic systems

Mixed-Signal Systems

Analyze, design, and verify analog and mixed-signal systems

Power Electronics Control Design

Design and implement digital control for motors, power converters, and battery systems



Toolboxes

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

Course Contents

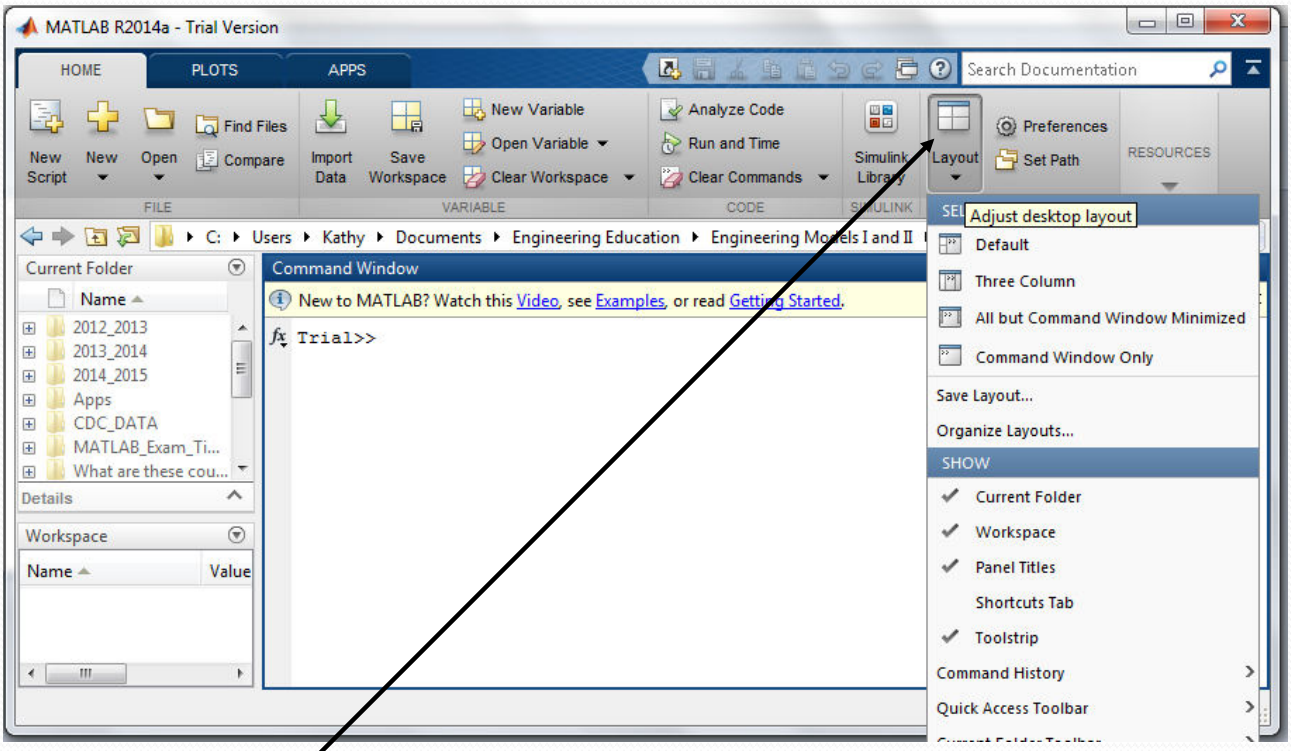
- Variables, and data types
- Script Files
- Conditional Statements
- Loops, Nested Loops
- Arrays
- Functions
- Plotting
- Use of MATLAB to solve Mathematical Problems

MATLAB® Desktop

MATLAB Desktop

- The **Command window** is where you type MATLAB commands following the prompt: >>
- The **Workspace window** shows all the variables you have defined in your current session. Variables can actually be manipulated within the workspace window.
- The **Command History** window displays all the MATLAB commands you have used recently – even includes some past sessions.
- The **Current Folder** window displays all the files in whatever folder you select to be current.

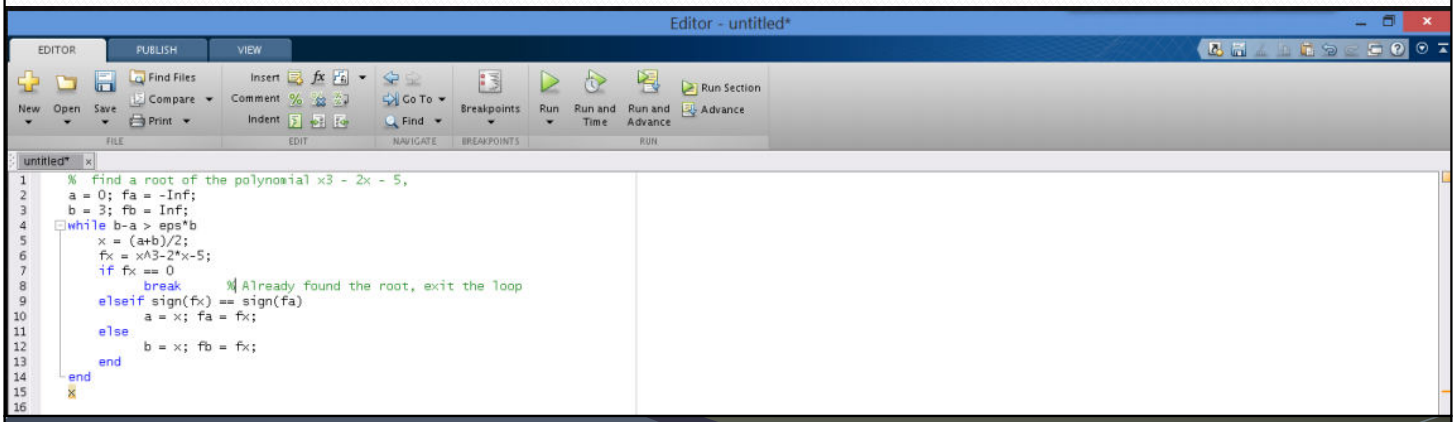
MATLAB Desktop



You can select what is on your desktop by Clicking on Layout. Go down to Command History and select docked.

M-File

- An m-file, or script file, is a simple text file where you can place MATLAB commands.
- Save your work
- Convenient for debugging
- Run directly



Using MATLAB® As a Calculator

Arithmetic Operators and Order of Operations

- Addition (+), Subtraction (-), Multiplication (*), Division (/), Power (^)
- Order of Operations (same rules you should already know from math class and using a calculator)
    1. Complete all calculations inside parenthesis or brackets using the precedent rules below
    2. Powers (left to right)
    3. Multiplication and Division (left to right)
    4. Addition and Subtraction (left to right)

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.
.\	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.
,	Comma; separates statements and elements in a row
;	Semicolon; separates columns and suppresses display.
%	Percent sign; designates a comment and specifies formatting.
=	Assignment operator.

Arithmetic Operators and Order of Operations

Some Examples:

```
>>  10/5*2

>>  5*2^3+4(2)

>> -1^4

>>  8^1/3
```

Exercise 1

Calculate the area and circumference of a circle with a radius of 4 cm.

Area = 50.27 cm^2

Circumference = 25.13 cm

Doing Mathematics with Matlab (I)

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Doing Mathematics with Matlab (I)
Exercise

Outline

1. Doing Mathematics with Matlab (I)
   - Arithmetic
   - Symbolic Computation
   - Substitution in Symbolic Expressions
   - Errors in input
2. Exercise

Imran Talib VU Lecture NO.02 2

Arithmetic

This lecture describes some of the basic MATLAB commands that will be used in this course. For example

- + command is used for addition.
- - command is used for subtraction.
- For division, we use /.
- * command is used for multiplication.
- The MATLAB Desktop looks like the Figure 1 after performing these commands.

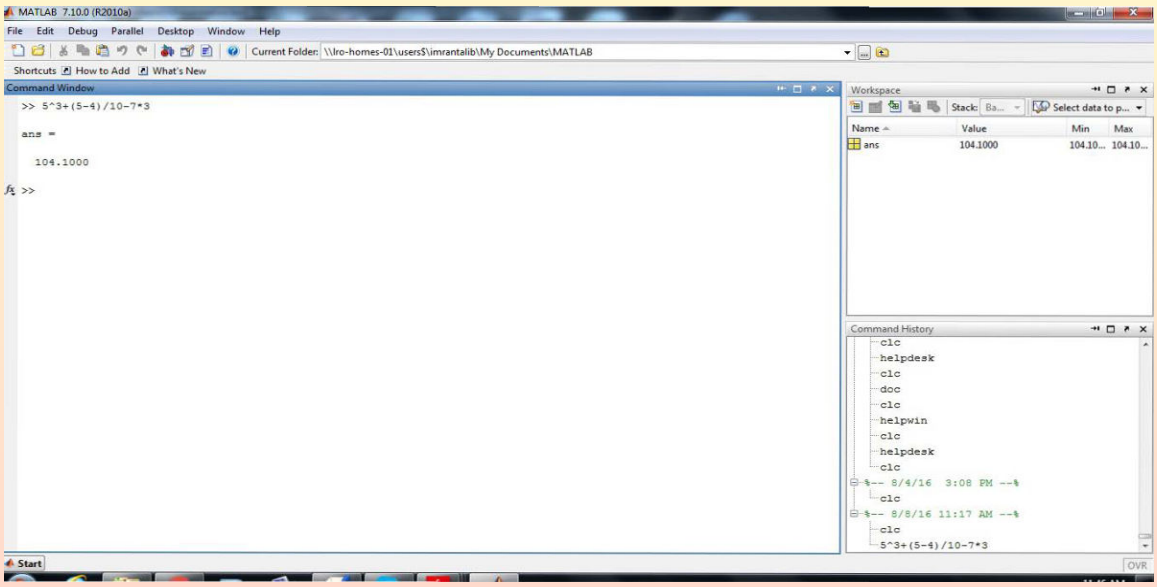


Figure: (1) The Matlab Desktop display after performing some basic arithmetic commands

Arithmetic

- Note that in the Figure 1, MATLAB prints the answer and assigns the value to a variable called **ans**.
- If the further calculation is needed then simply use the variable **ans**, rather than retype the answer.
- It is to be noted that MATLAB assigns a new value to **ans** with each calculation.
- For example the square root of the previous answer can be taken using the command **sqrt(ans)**. The output is displayed in the Figure 2

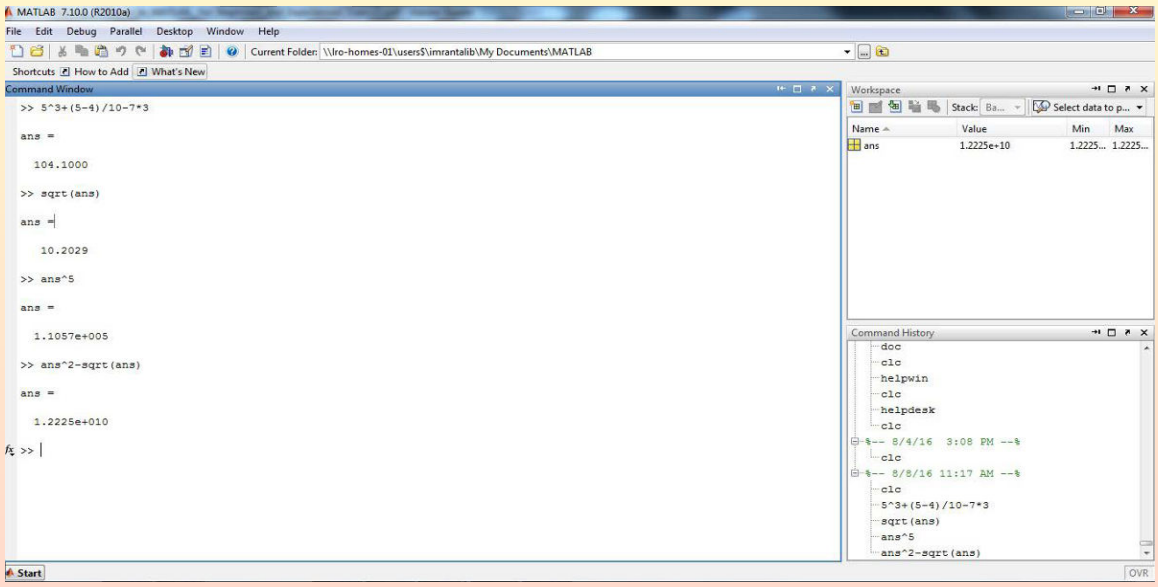


Figure: (2) The Matlab Desktop display after performing some basic arithmetic commands

Arithmetic

- On the other hand, sometimes while doing complex calculations, we prefer to assign the variables of our own choice to computed values.
- So, MATLAB gives us independence to assign variables of our own choice.
- For example, $sin^2(1) + cos^2(1)$ can be computed by assigning a variable to $sin(1)$, and b variable to $cos(1)$. The results are displayed in Figure 3.

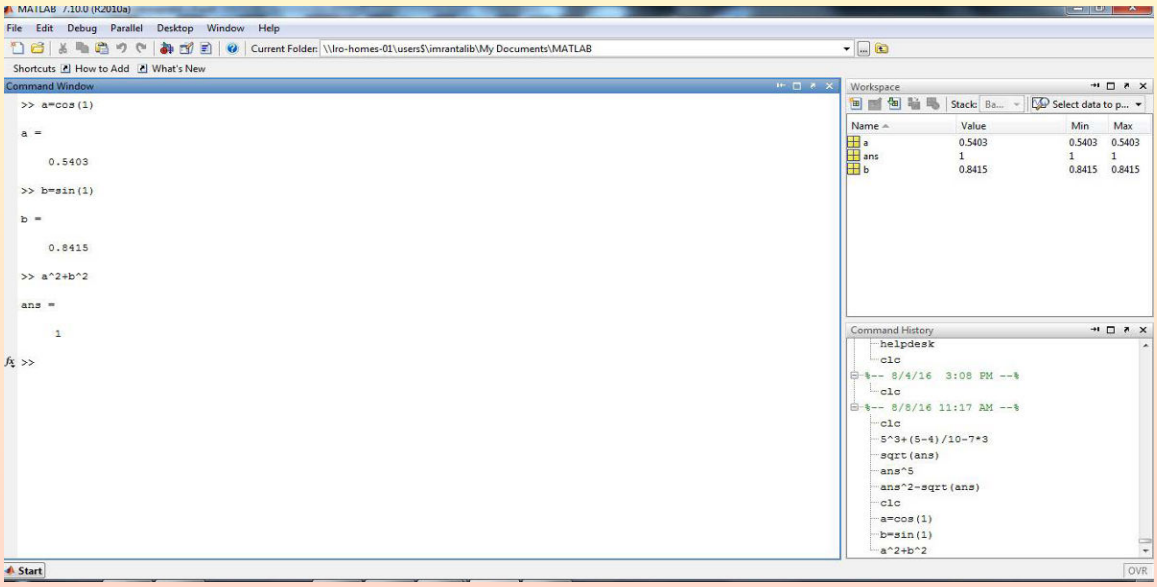


Figure: (3) The Matlab Desktop display after performing some basic arithmetic commands

Symbolic Computation

- To perform symbolic calculations, make sure that **Symbolic Toolbox** is installed on your system.
- For this type **help symbolic** in the Command Window and run this command by pressing enter key.
- If it is installed then the MATLAB desktop looks like the Figure 4, otherwise you have to install it like the installation of the other packages.

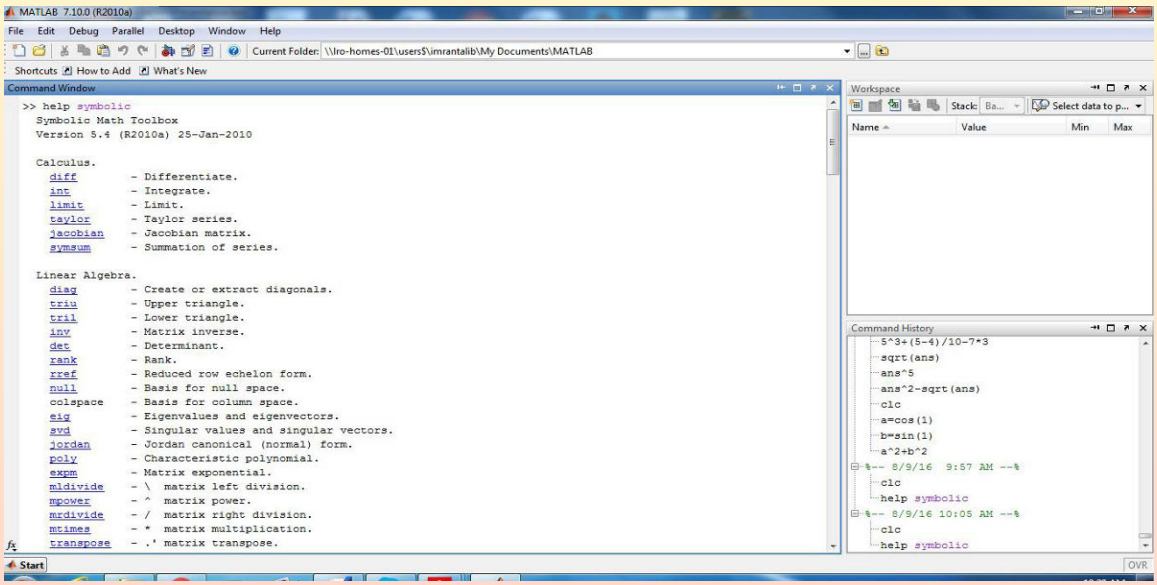


Figure: (4) The Matlab Desktop display with help symbolic command

Symbolic Computation

- To do symbolic computations, you must use syms to declare the variables you plan to use to be symbolic variables.
- Usually this command is used to solve algebraic expressions. For example
- To compute $(x-y)*(x-y)^2$, declare x and y to be symbolic.
- The MATLAB Desktop looks like Figure 5.

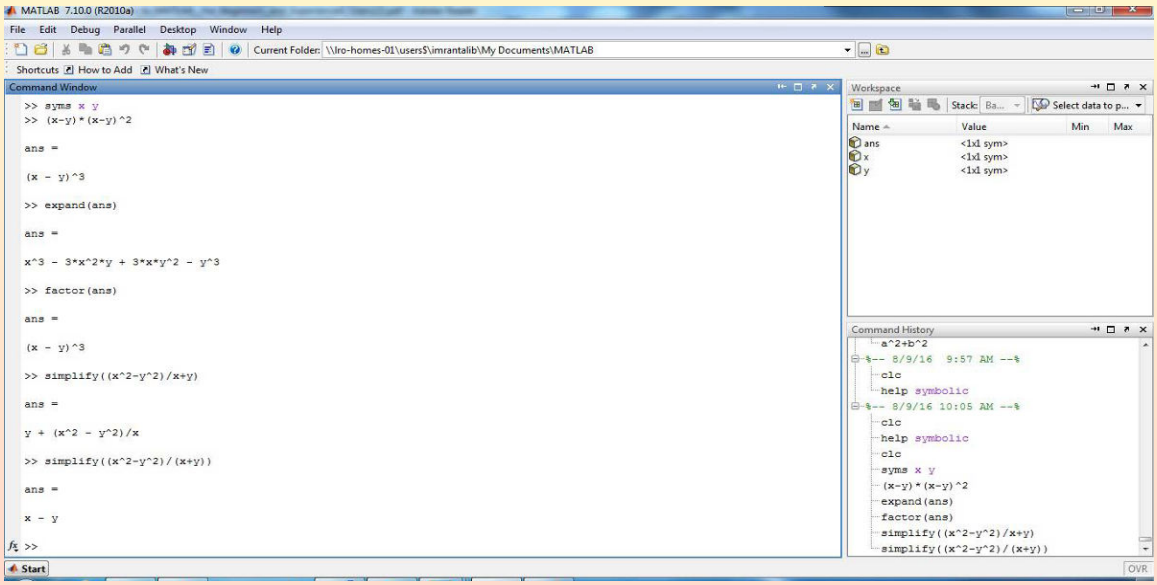


Figure: (5) The Matlab Desktop display after declaring x and y to be symbolic

Symbolic Computation

Remark

It is to be noted that MATLAB uses floating point arithmetic for its calculations. But exact arithmetics can also be done using the Symbolic Math Toolbox with symbolic expressions.
For example, the floating point format of $\cos(pi/2)$ *in MATLAB is* $6.1232e-17$. *However, it is known that* $\cos(pi/2)$ *is really equal to zero. The inaccuracy is due to the fact that typing* ***pi*** *in MATLAB gives an approximation to* π *accurately to about 15 digits. But sym command gives accurate value of* π. *Look at the Figure 6*

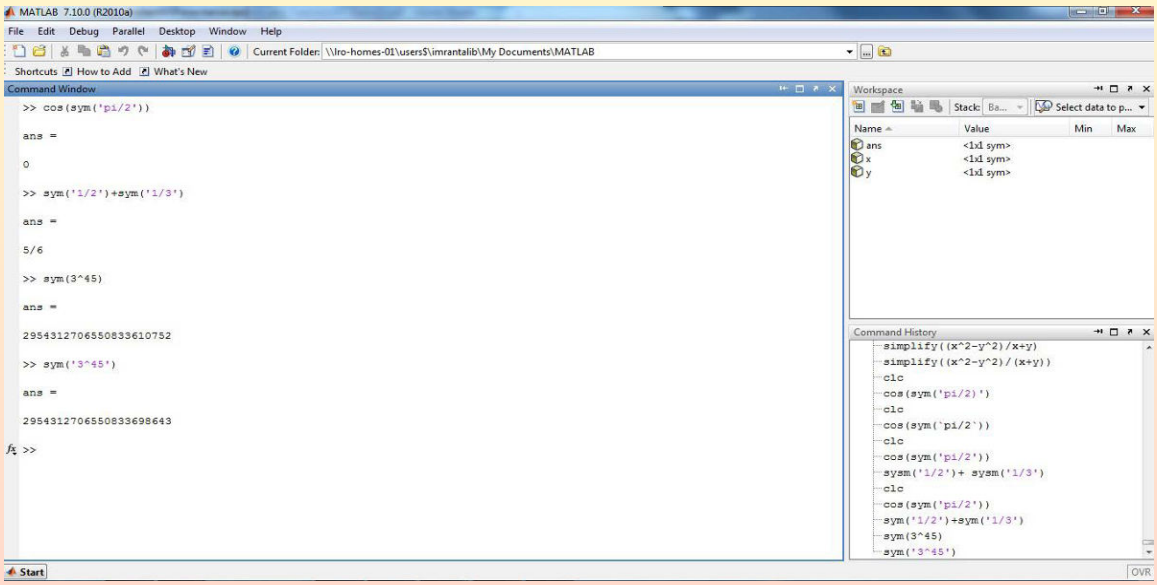


Figure: (6) The Matlab Desktop display with sym command

Substitution in Symbolic Expressions

- While working with symbolic expressions, some times it is necessary to substitute a numerical value, or even another symbolic expression, for one or more of the original variables in the expression.
- **subs** command is used for this purpose.
- For example, we want to substitute the numerical value 2 for the symbolic variable x in the symbolic expression $u = x + y^2$.
- For this we use the command **subs(u,x,2)**.
- Similarly, we can substitute the symbolic expression $y + x$ for the symbolic variable y in the symbolic expression $u = x + y^2$ as, **subs(u,y,y+x)**.
- The MATLAB Desktop looks like the Figure 7.

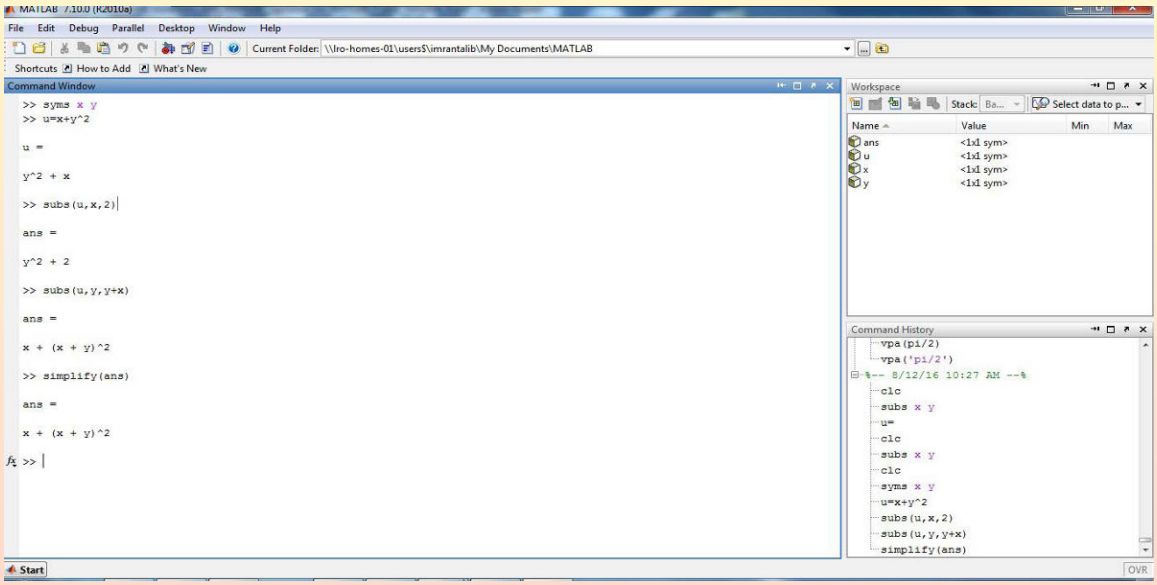


Figure: (7) The Matlab Desktop display with substitution command

Errors in input

- If you make an error while entering input in a Command Window, MATLAB will beep and print an error message.
- For example, note that what happens when you try to evaluate *sin*10:
- » sin10 ??? Undefined function or variable 'sin10'.
- Note that there you did not write the correct command. The correct command is sin(10). Then Matlab gives you the correct output. See Figure 8.

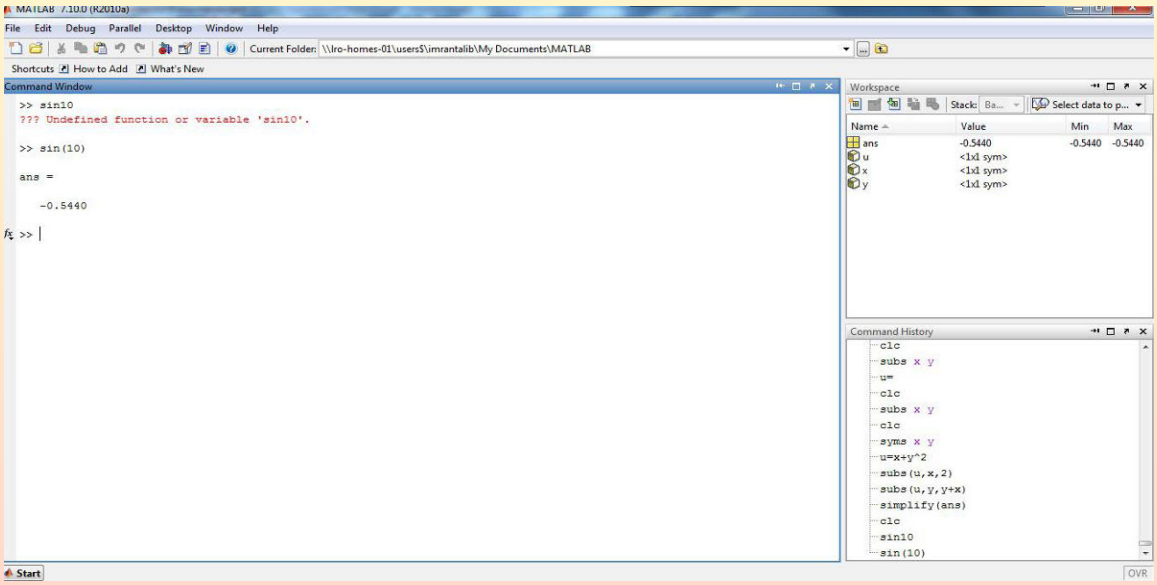


Figure: (8) The Matlab Desktop display while removing error

1. Find the value of the following polynomials at $x = 2$ using **subs** command:
   - $p(x, y) = x^2 + 3y^2x + 10$
   - $u(x) = x^3 + 4x + 10$
2. Find the value of the following trigonometric functions at $x = \frac{\pi}{2}$ using sym command:
   - $\cos(x), \sin(x), tan(x)$.

1. Expand the following algebraic expressions using **expand** command:
   - $(x + y)^{10}$
   - $(x + y)^3 + (x + y)^2$
2. Simplify the following quantities:
   - $2 + 5 - 4 + 10^{10} + 7^2$.
   - $2/5 + 4/10 - 100^2$

Doing Mathematics with Matlab (II)

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Outline

Variables and Assignments

- In MATLAB, = sign is used to assign values to a variable.
- For example, to assign 7 value to a variable x, we proceed as
- » x=7
  x =
  7
- From now to own-ward, whenever MATLAB sees the letter **x**, it will substitute the value 7.
- For example, if y has been defined as a symbolic variable, then
- » x=7
  x =
  7
  » syms y » $x^3 + 2 * x * y + 10 * x$
  ans =
  $14 * y + 413$

Variables and Assignments

- To assign a new value to a variable x, type **clear x** command.
- For example, to assign 8 value, we have to proceed as
- » syms y
  » clear x » x=8
  x =
  8
  » $x^3 + 2 * x * y + 10 * x$
  ans =
  $16 * y + 592$
- The MATLAB Desktop looks like the Figure 1

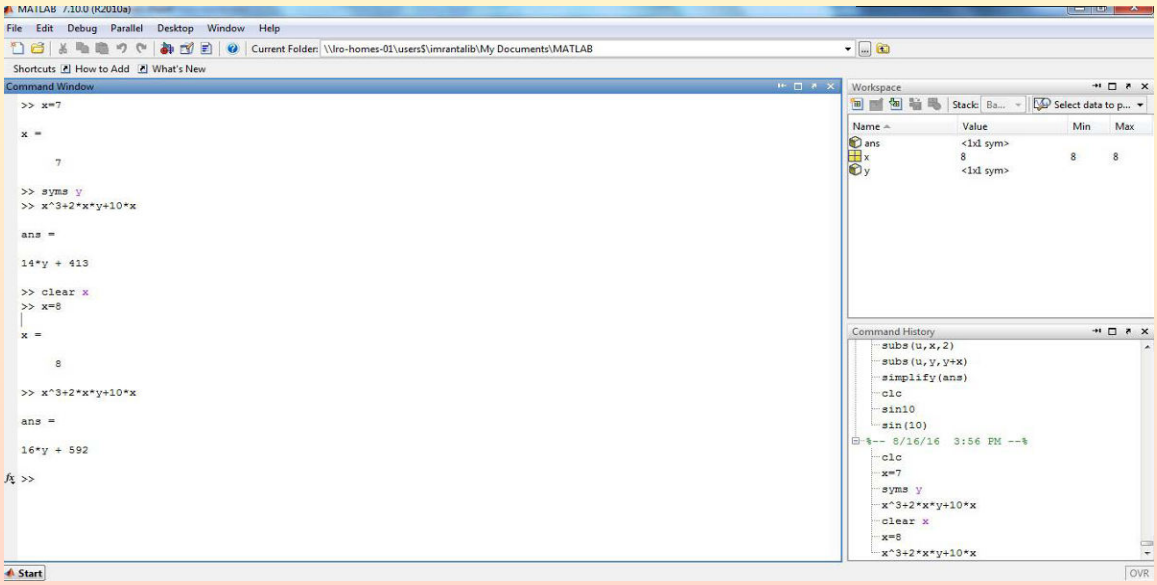


Figure: The Matlab Desktop display with Variable assign command

Vectors

A vector is an ordered list of numbers. A vector of any length can be entered in MATLAB as

- By typing a list of numbers, separated by commas or spaces, inside square brackets. For example
- A vector X with entries 2, 4 and 5 can be generated as
- » X=[2,4,5]
  X =
  2 4 5
- On the same fashion a vector Y with entries 2, 7, and 8 can be generated as
- » Y=[2,7,8]
  Y =
  2 7 8

Vectors

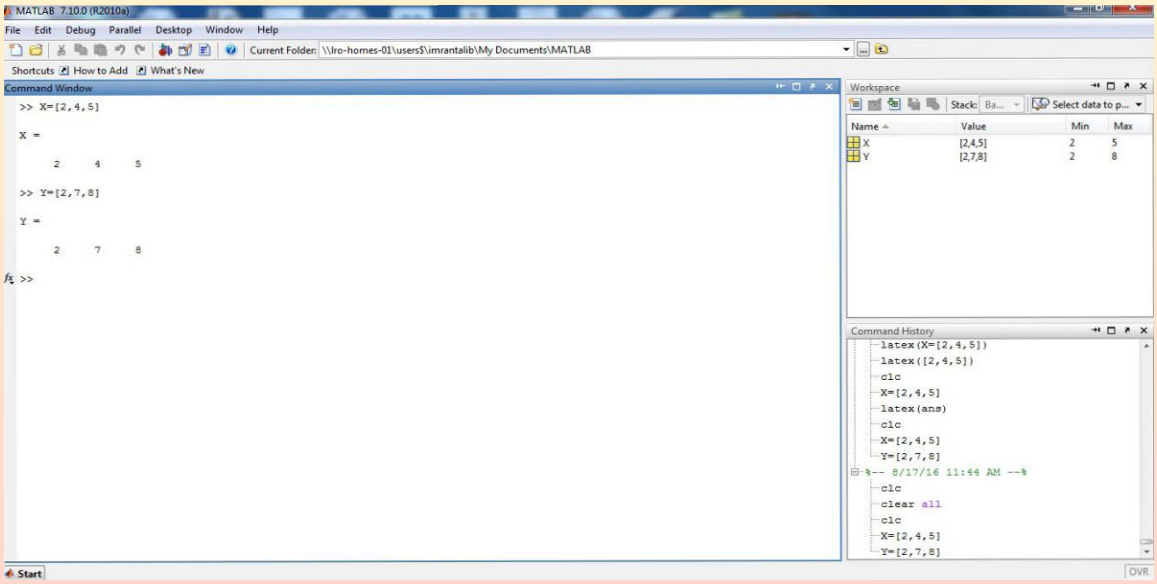


Figure: The Matlab Desktop display with Vector command

Vectors

- Vector can also be created without typing each number. For example
- The vector X with entries 2, 3, 4 and 5 can also be created as
- » X=2:5
  X =
  2 3 4 5
- The notation X=2:5 represents a vector of numbers running form 2 to 5 with increment 1, that increment can also be specified as the second of three arguments as
- » Y=1:2:12
  Y =
  1 3 5 7 9 11

Vectors

- To change a vector Y from a row vector to a column vector, simply put a prime (') after Y, as
- » Y'
  ans =
  1
  3
  5
  7
  9
  11
- You can also extract the entries of vector Y. For example, the entry 5 can be extracted as
- » Y(3)
  ans =
  5

Vectors

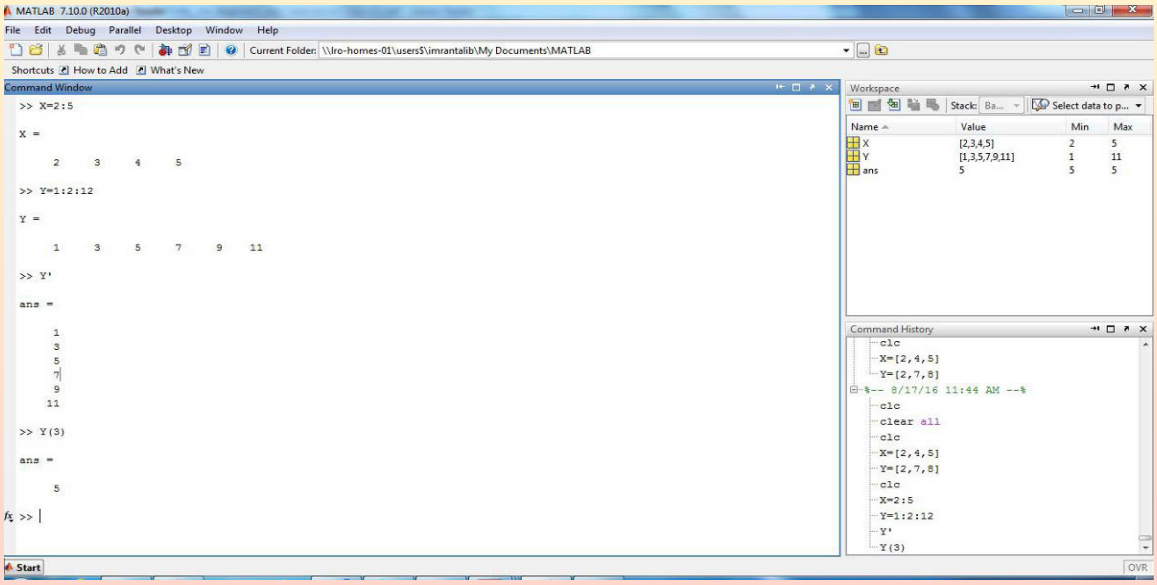


Figure: The Matlab Desktop display with increment command

Vectors

- Mathematical operations can also be performed on vectors. For example, to square the elements of of the vector Y,use the following command
- $Y.^2$
  ans =
  1 9 25 49 81 121
- Similarly you can perform other mathematical operations, see the Figure 4

Vectors

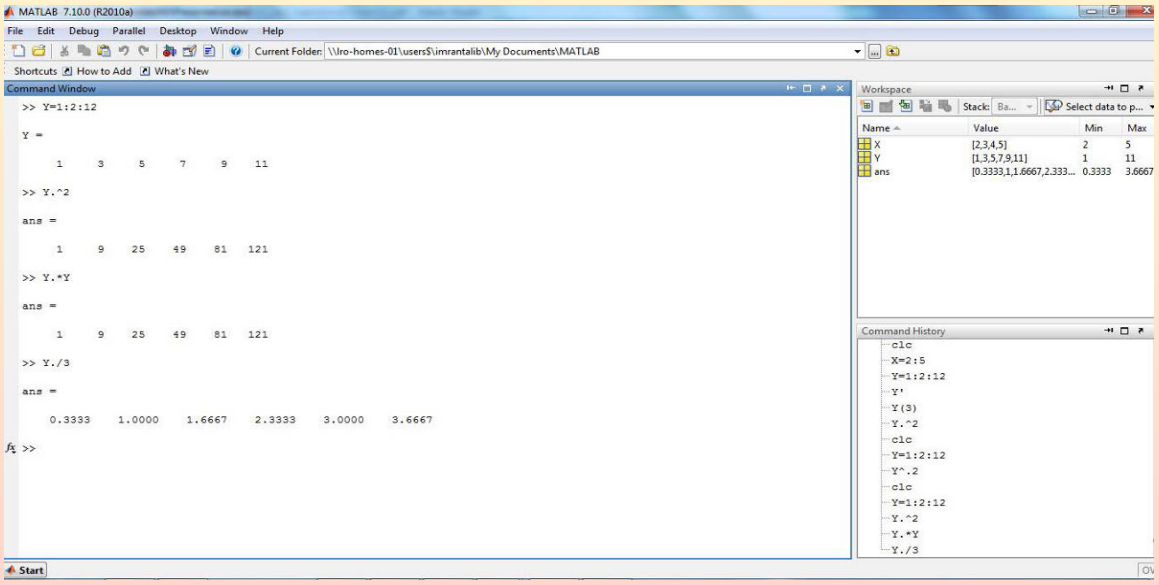


Figure: The Matlab Desktop display with operations on vectors

Built-in Functions

- MATLAB has all the usual **elementary functions** built in. For example,
- **exp(x)** indicates the exponential function of x.
- **log(x)** represents the natural logarithm function of the argument x.
- **sqrt** is used to take the square root of an entity.

User Defined Functions

- We can also define our own functions in MATLAB using the command **inline** or using the operator **@**.
- We prefer the latter command. For example, the function x^3 can be defined as
- » $f = @(x)x^3$
  f =
  $@(x)x^3$

User Defined Functions

- Once the function is defined, it can be evaluated at any real number. For example
- To calculate the value of x^3 at $x = 10$, we will proceed as
- » $f = @(x)x^3$
  f =
  $@(x)x^3$
  » f(10)
  ans =
  1000
- See the Figure 5

User Defined Functions

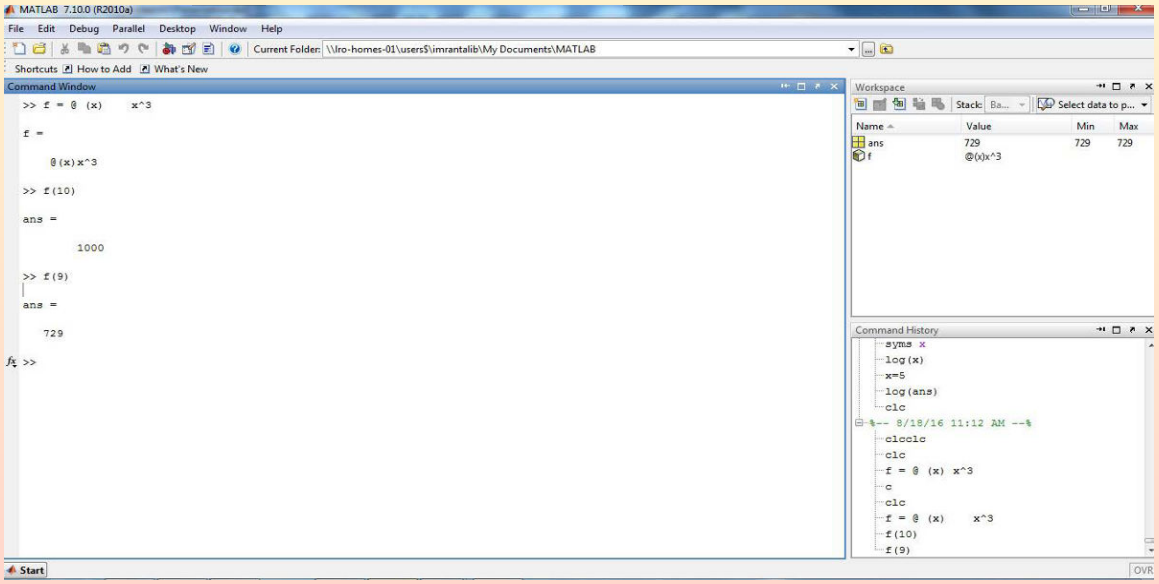


Figure: The Matlab Desktop display with user defined function command

User Defined Functions

- This command can also be used to define functions of several variables. For this we proceed as
- » $h = @(x, y)x^2 + y^2$
  h =
  $@(x, y)x^2 + y^2$
- $h(x, y) = x^2 + y^2$ can also be evaluated at different values of x and y as,
- » h(1,2)
  ans =
  5
- See the Figure 6

User Defined Functions

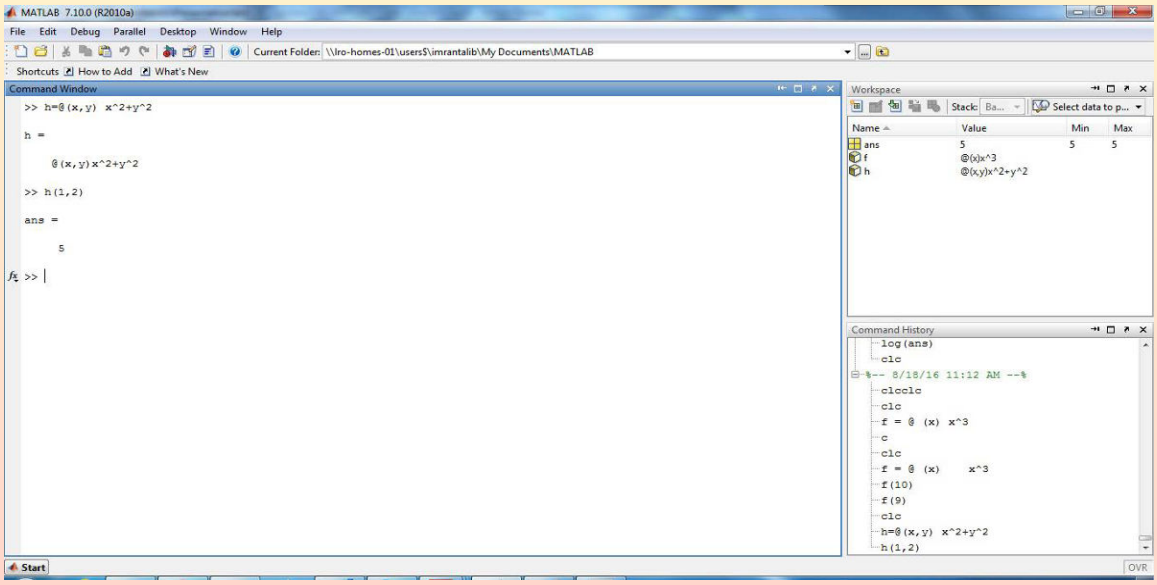


Figure: The Matlab Desktop display with user defined function command

Solving Algebraic Equations

Now, we learn how to solve the Algebraic equations in MATLAB. The command **solve** or **fzero** is used for this purpose. For example,

- The MATLAB solution of the equation $x^2 - 5x + 6 = 0$ can be find as
- $\gg solve('x^2 - 5 * x + 6 = 0')$.
- The MATLAB output is
- ans =
  2
  3
- Note that here the equation to be solved is specified as a string, that is, it is surrounded by a single quotes.
- See the Command Window 7

Solving Algebraic Equations

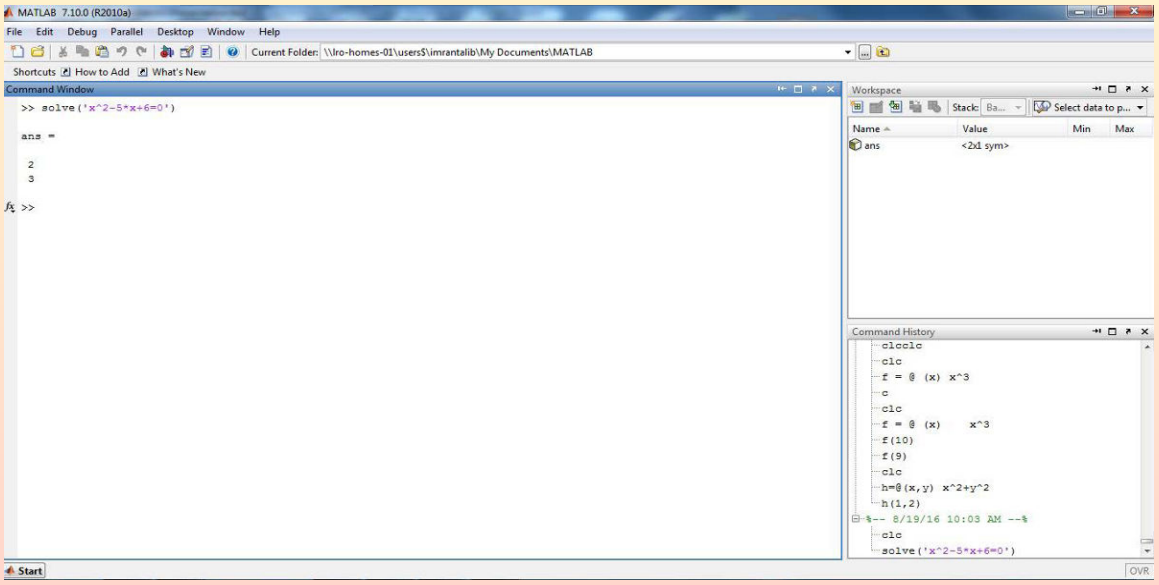


Figure: The Matlab Desktop display with Algebraic equations solver

Solving Algebraic Equations

Note that, the input to **solve** can also be a symbolic expression. For instance

- The MATLAB syntax for solving $x^2 - 3x = -7$ is:
- syms x;
  $solve(x^2 - 3 * x + 7)$
- The MATLAB generated output is
- ans =
  $3/2 - (19^{(1/2)} * i)/2$
  $3/2 + (19^{(1/2)} * i)/2$

Solving Algebraic Equations

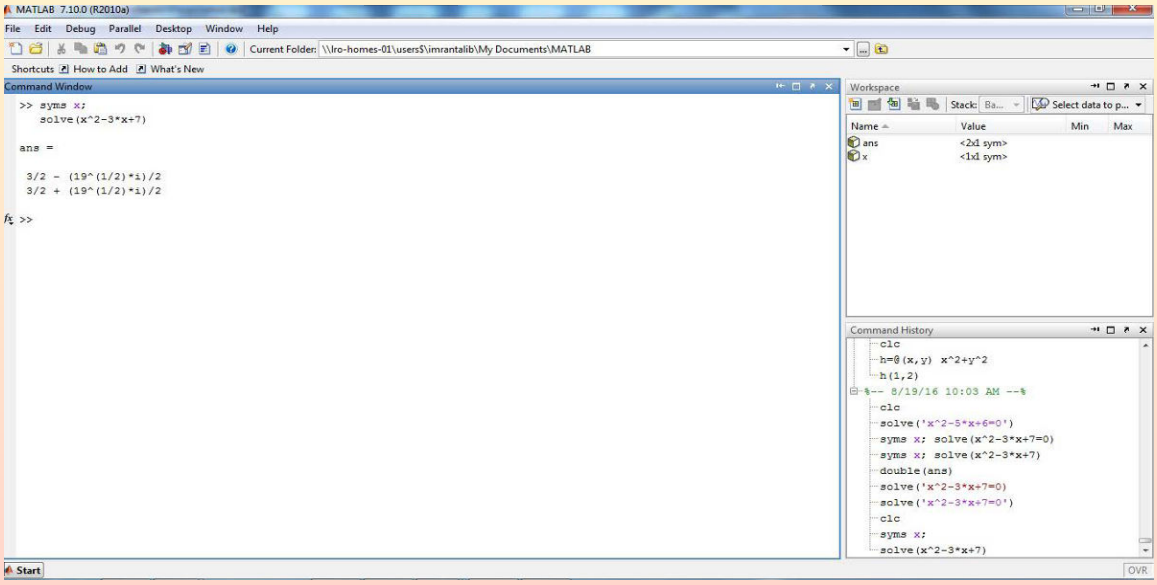


Figure: The Matlab Desktop display with Algebraic equations symbolic command

Solving Algebraic Equations

- The **solve** command can also be used to solve higher degree polynomial equations, as well as many other types of equations.
- It can also be used to solve equation having more than one variables. For example to find the solution of $x^2 - y = 2$ and $y - 2x = 5$, the following syntax is used:
- $\gg [x, y] = solve('x^2 - y = 2', 'y - 2 * x = 5')$
- The MATLAB generated output is:
- x =
  $2 * 2^{(1/2)} + 1$
  $1 - 2 * 2^{(1/2)}$

Solving Algebraic Equations

- Note that, MATLAB reports the solution of the above system of equations by giving two x values and two y values.
- Thus the first solution consists of first value of x together with the first value of y.
- Similar case exists for the second solution.
- The first solution can also be extracted by typing x(1) and y(1) in the Command Window.
- See the Command Window 9

Solving Algebraic Equations

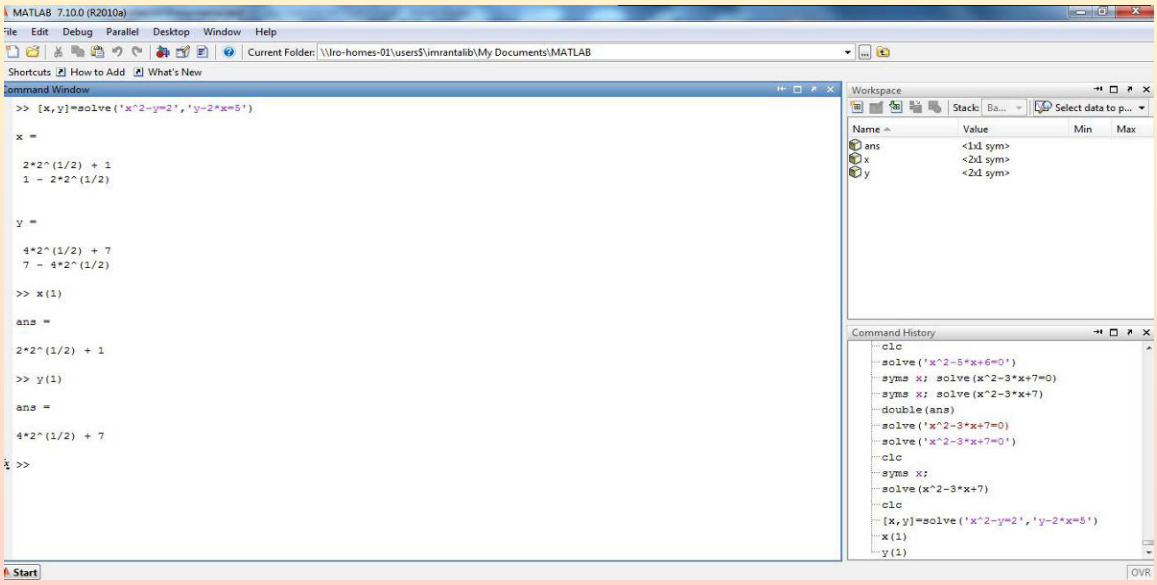


Figure: The Matlab Desktop display with the solution of the system of algebraic equations

Solving Algebraic Equations

- Sometimes an equation have more than one solutions and you may not get what you expected. For example the MATLAB syntax:
- » $solve('exp(-x) = sin(x)')$, gives
- the complex number answer. But we are interested to see its another numerical solution near to 0.5. For this use the following syntax:
- » $h = @(x)exp(-x) - sin(x)$;
  » fzero(h,0.5)
- The required output will be
- ans =
  0.5885

Solving Algebraic Equations

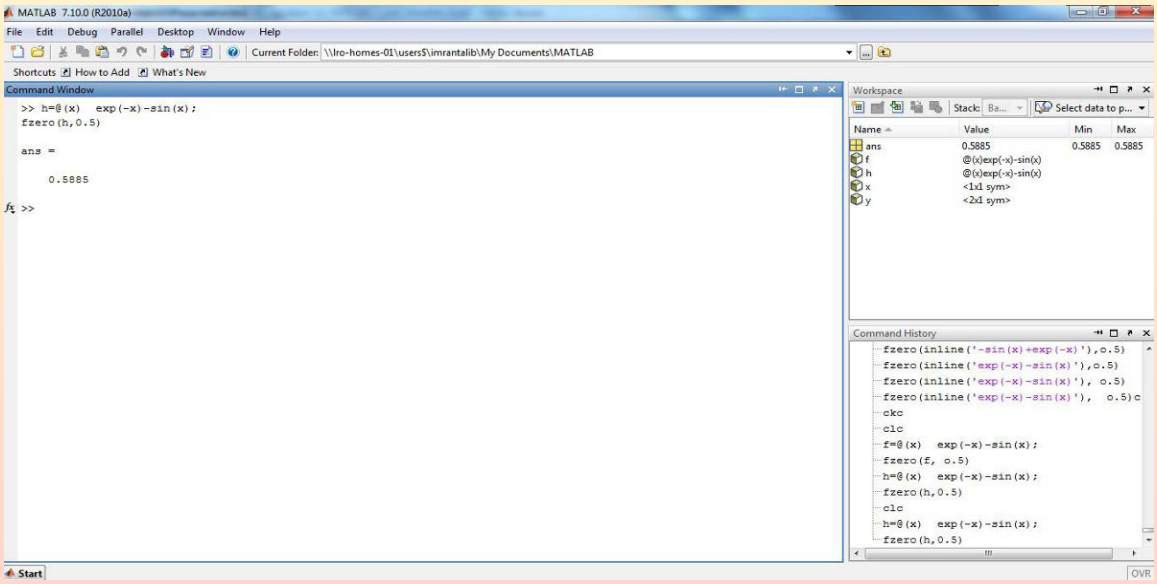


Figure: The Matlab Desktop display with fzero command

Exercise

Solve the following equations using MATLAB **solve** command

- $x^2 + 2x - 4 = 0$.
- $x^3 + 5x^2 + 4x + 3 = 0$

Solve the following equations for x using **solve** command

- $x + log(y) = 3$.
- $x + 10y = 6$.

Find the solution of the following equation using **fzero** command near to $x = 3$

- $exp(-x) = sin(x)$.

Solve the following system of equations using **solve** command

- $x + y^2 = 2,\ y - 3x = 7$.
- $x + y^2 = 2,\ 2y^2 - 3x = 7$.

Graphics Techniques in MATLAB

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

ezplot Command

In this lecture, we learn the use of the basic plotting commands in MATLAB.

- **ezplot** command is used to plot the graph of a function of one variable.
- It expects a string, a symbolic expression or a function representing the function to be plotted.
- For example, the MATLAB command to draw the graph of the function x^3 on the interval $[-2, 2]$, using the string form of **ezplot** is:
- » ezplot('x^3',[-2,2]).
- Note that the result will display in a new window labeled "Figure1". Look at the Figure 1

ezplot Command

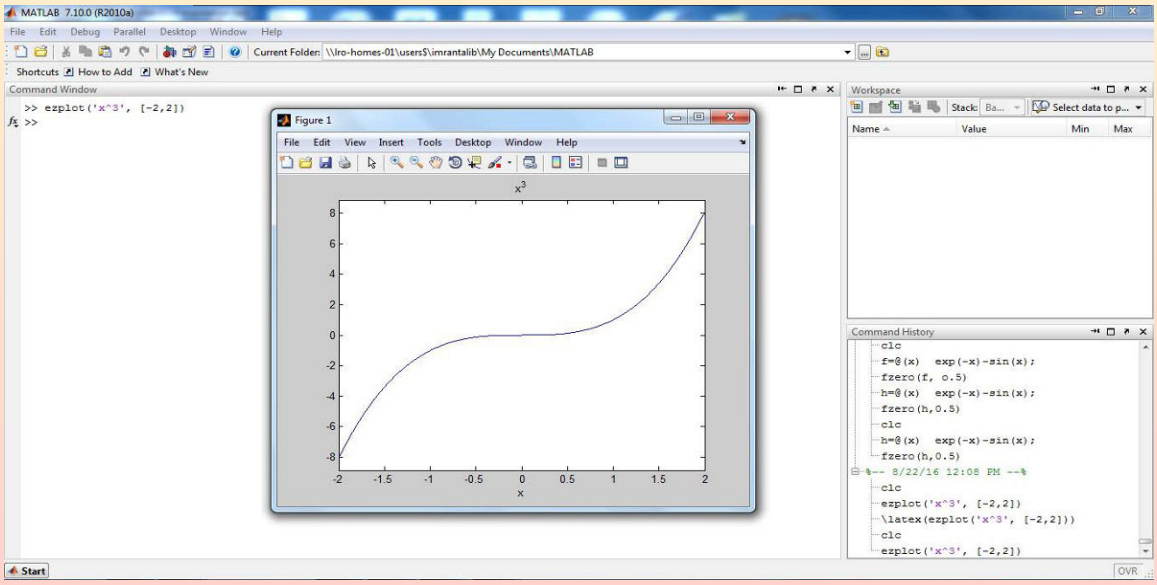


Figure: String form plot with ezplot command

- **ezplot** command can also accepts a symbolic expression. For example
- The graph of x^3 can also be produced using the following input:
- » syms x;
  » ezplot(x^3, [-2,2])
- See the Figure 2

ezplot command with symbolic expression

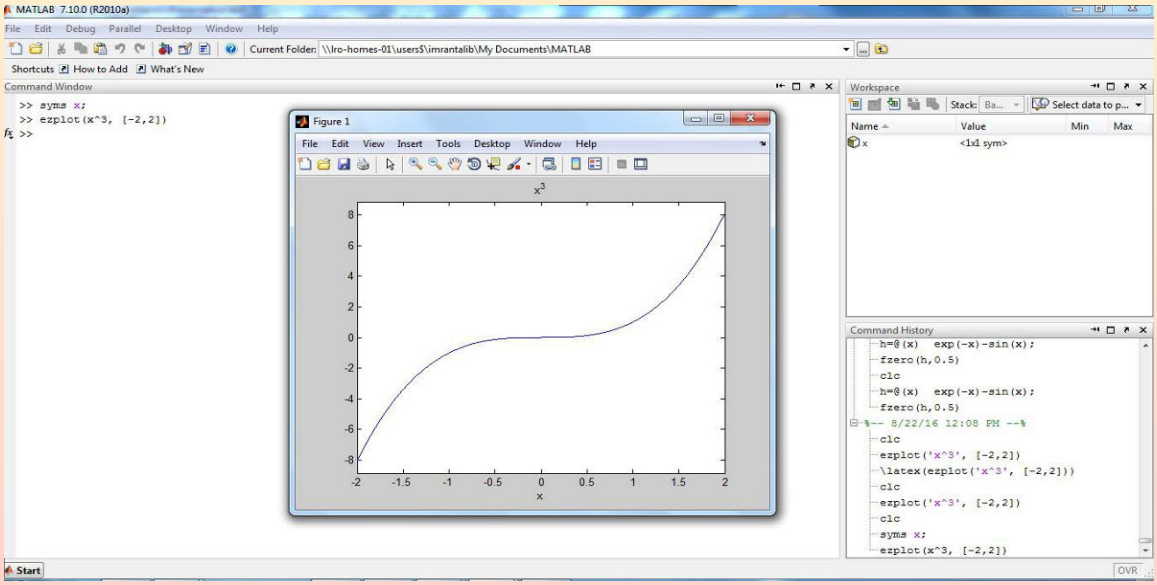


Figure: Symbolic expression form plot with ezplot command

- The graph of function x^3 can also be plotted by using an anonymous function as the argument of **ezplot**. For this case the input will be:
- » ezplot(@(x)x.^3, [-2,2]).
- See the Figure 3

ezplot command on anonymous function

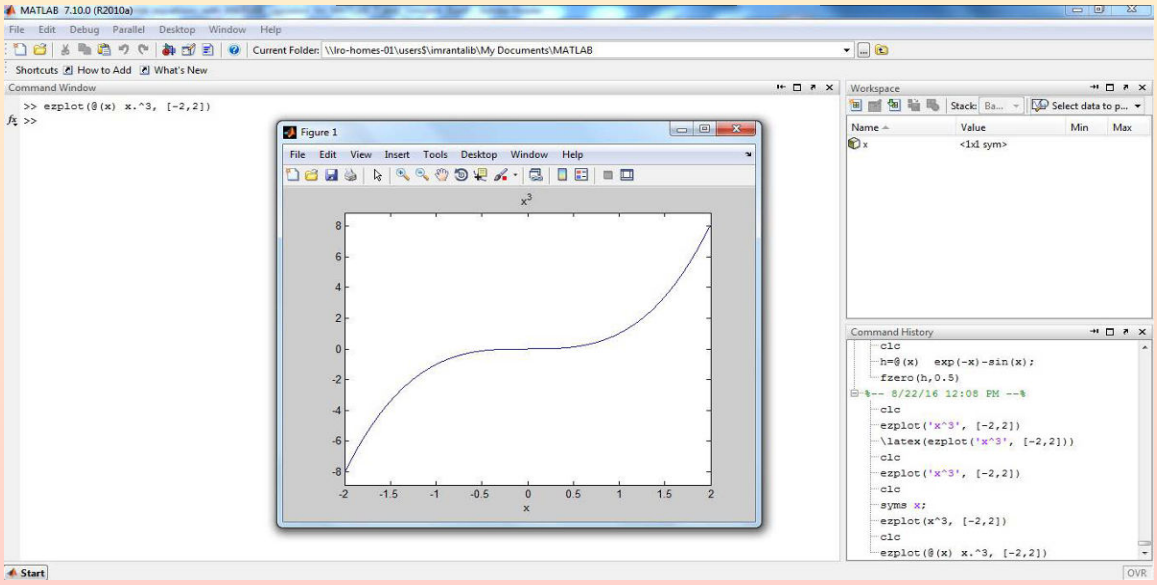


Figure: Anonymous function form plot with ezplot command

- Graphs can be modified in a number of ways. For example
- The title of the graph can be inserted.
- Different types of labels can be given to both axis.
- The region of the Graph can be seen in the form of grid.
- To accomplish this task, simply type the following commands in the Command Window
- syms xy;
  y=ezplot(x^3, [-2,2])
  xlabel('x');
  ylabel('y');
  title('The graph of $y = x^3$');
  grid; ee the output 3

Modification of Graphs

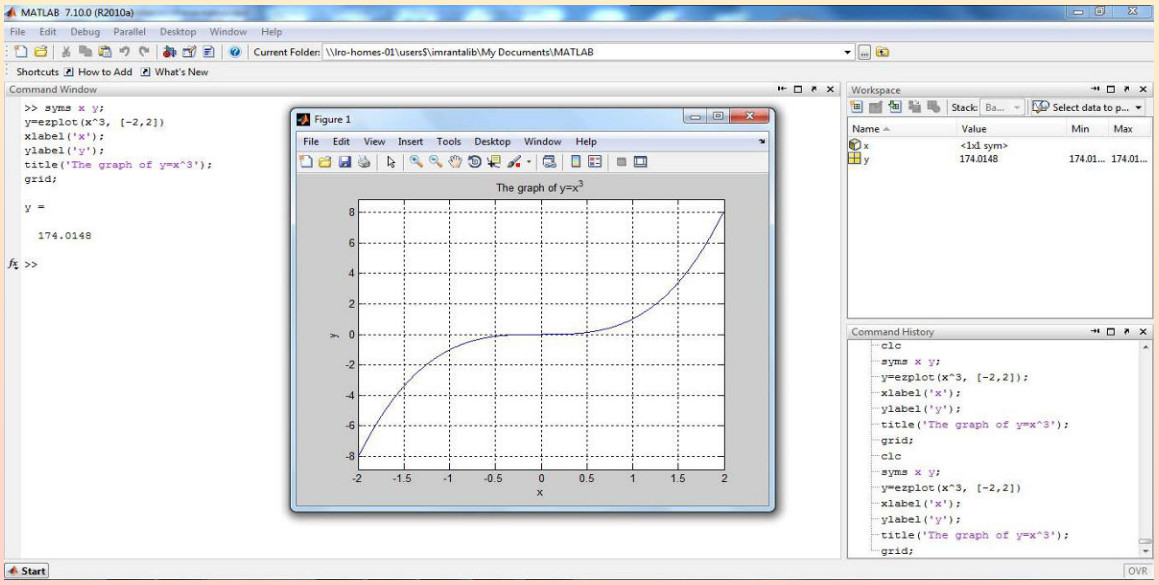


Figure: Modification of the Graph $y = x^3$

Example:

- Plot the graph of the function $y = x^2$ on $[-3, 3]$ using ezplot command considering x and y to be symbolic variables.
- Label the horizontal axis as "VU" and vertical axis as "IT".
- Keep the title of the graph as "Parabola"

Solution:

- The following syntax can be used to attain the required graph:
- » syms x, y;
  y=ezplot(x^2, [-3,3])
  xlabel('VU');
  ylabel('IT');
  title('Parabola');
  grid;
- Also see the Graph 5

Graphical view of the Example

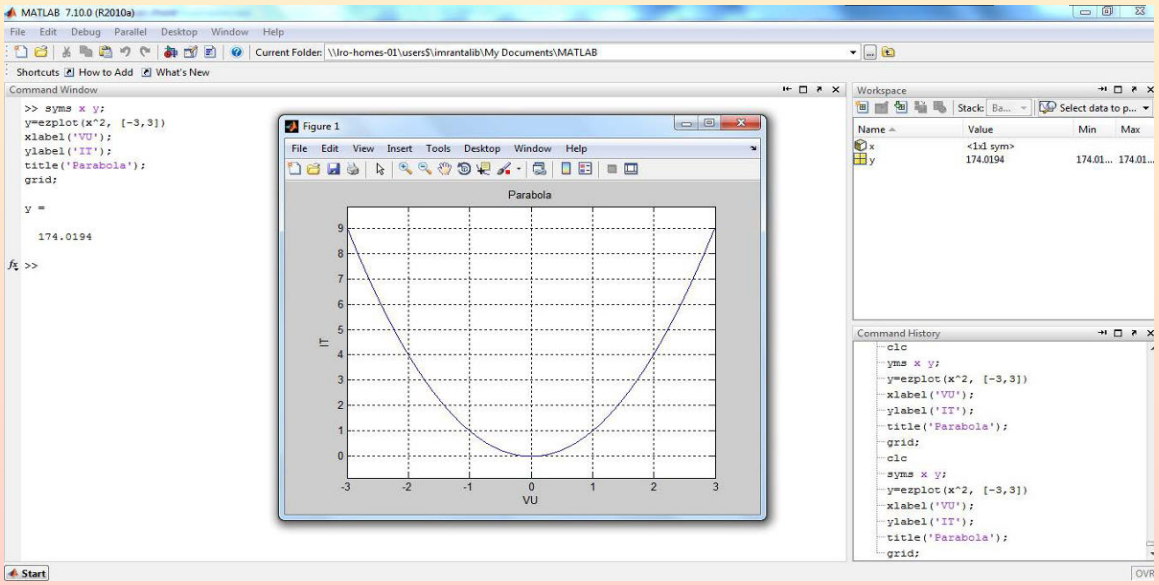


Figure: Modification of the Graph $y = x^3$

plot Command

- This command works on the vectors of numerical data.
- The syntax is **plot(X,Y)**.
- X and Y are the vectors of same length.
- This command considers the vectors X and Y to be lists of x and y coordinates of successive point on a graph, and joins the points with line
- Under **Plot** command the vectors $X = [3\ 5\ 7]$ and $Y = [9\ 11\ 13]$ are connected by line segments as (3,9) to (5,11) to (7,13).
- See their graphical view in Figure 7.

Graphical view of plot command

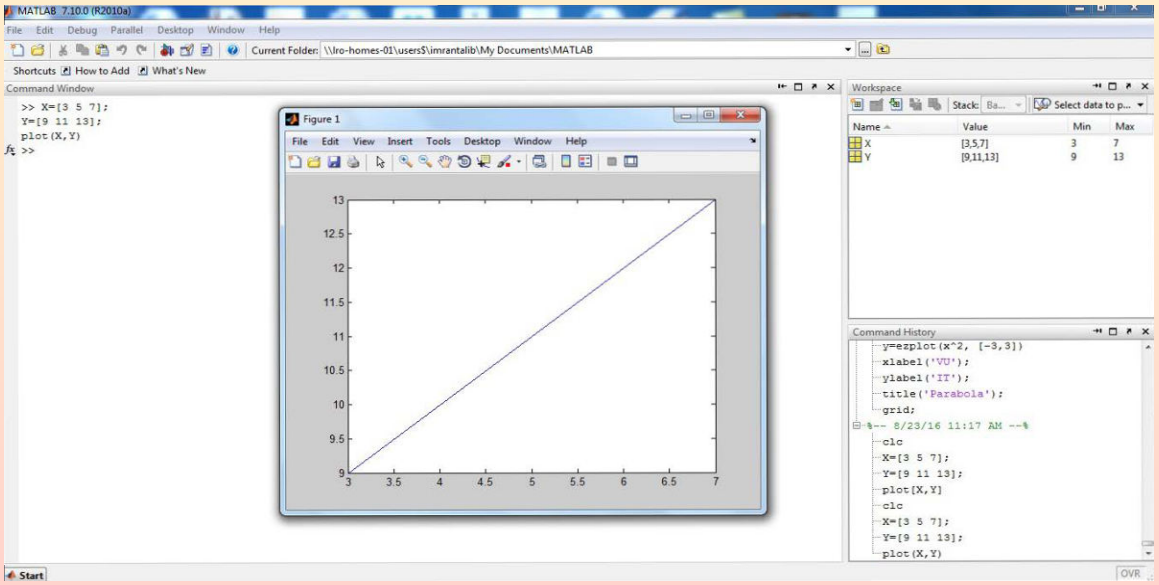


Figure: Plot command view of vectors of same length

Example:

Example:

- Using the plot command, plot the graph of $x^2 + x + 1$ on the interval $[-2, 2]$ by taking increment of 0.1 of the domain set.
- Label the horizontal axis with x and vertical axis with y.
- Finally, gives the title "Parabola" to the graph of $x^2 + x + 1$.

Solutioin:

- Firstly, we make a list X of x values.
- For this we need the enough values of x to get surety about the smoothness of the resulting graph. That is why we are going to subdivide the
- Thus a recipe for graphing the parabola is:

Example:

Solution:

- » X=-2:0.1:2;
  plot($X, X.^2 + X + 1$)
  xlabel('x');
  ylabel('y');
  title('Parabola');
  grid;
- The results appear in Figure 7

Graphical view of Example

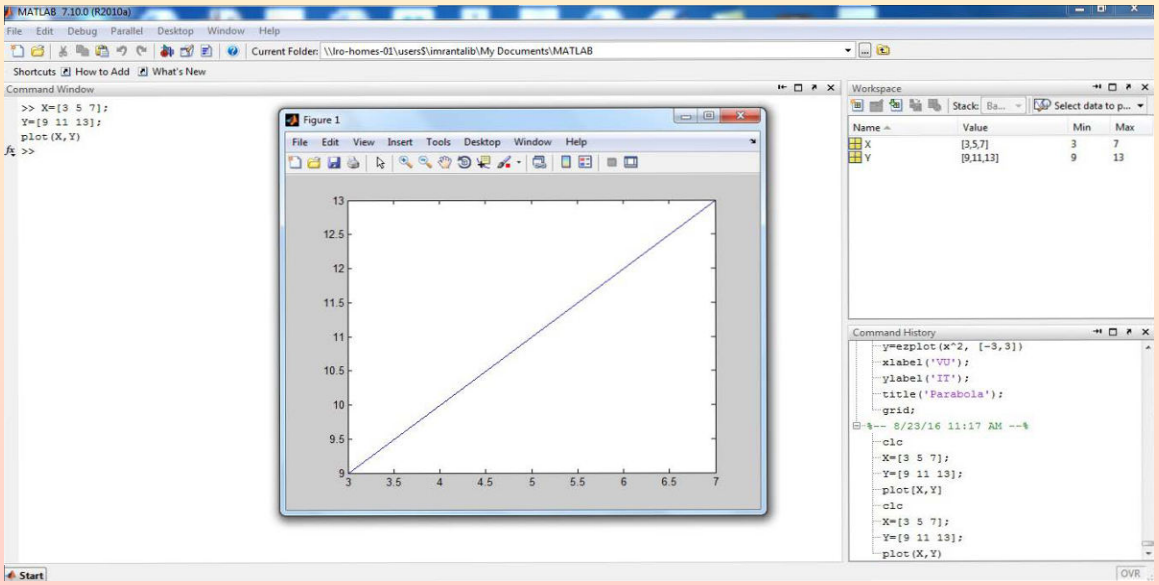


Figure: Plot command view of the Parabola

Graphs of Multiple Curves

- Each time we execute a plotting command, MATLAB erases the old plot and draws a new one.
- But if we want to see the two or more plots, then use command **hold on**.
- This command instructs MATLAB to retain the old graphics and draw any new graphics on top of the old.
- This command remains in effect until we type **hold off**.

Graphs of Multiple Curves

Example:

- Plot the graphs of e^{-x} and $sin(x)$ on the interval $[0, 8]$ using the **ezplot**, **hold on**, and **hold off** commands.
- Label the horizontal axis with x and vertical axis with y.
- Finally, gives the title "Multiple Graphs".

Graphs of Multiple Curves

Solution:

- The recipe for graphing the multiple curves is:
- » syms x;
  ezplot(exp(−x),[0,8])
  hold on
  ezplot(*sin*(x),[0,8])
  hold off
  title('Multiple Graphs')
  xlabel('x');
  ylabel('y');
  grid;
- The results appear in Figure 8

Graphical view of Example

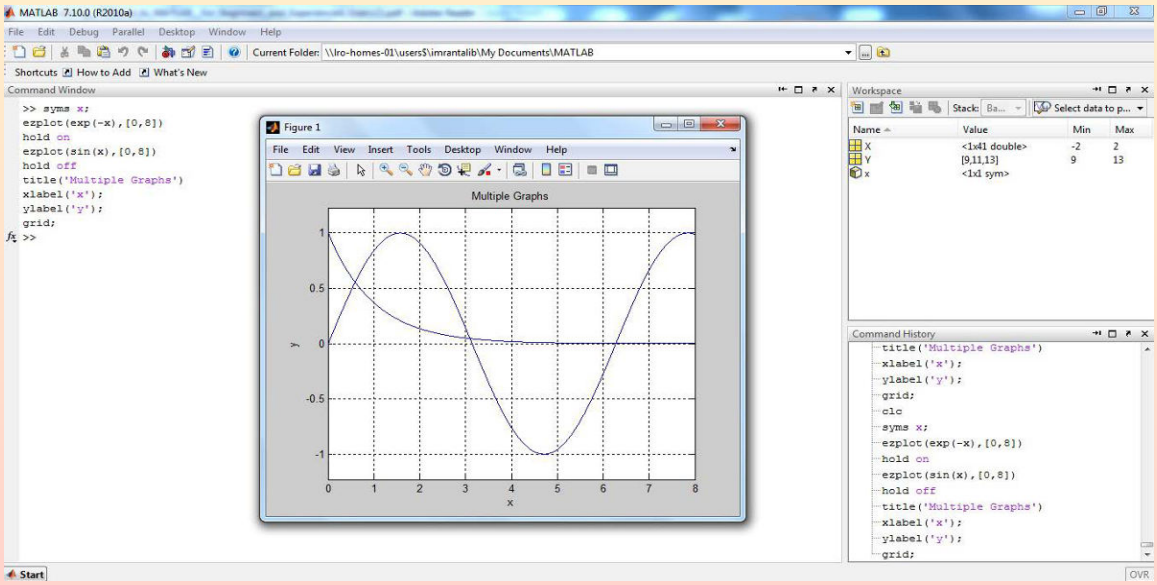


Figure: Ezplot command view of multiple curves

Graphs of Multiple Curves

Example:

- Plot the graphs of e^{-x} and $sin(x)$ on the interval $[0, 8]$ using the **plot**, **hold on**, and **hold off** commands.
- Label the horizontal axis with x and vertical axis with y.
- Finally, gives the title "Multiple Graphs".

Graphs of Multiple Curves

Solution:

- The recipe for graphing the multiple curves is:
- » X=0:0.1:8;
  plot($X, exp(-X)$)
  hold on;
  plot($X, sin(X)$)
  hold off;
  title('Multiple Graphs')
  xlabel('x');
  ylabel('y');
  grid;

Graphical view of Example

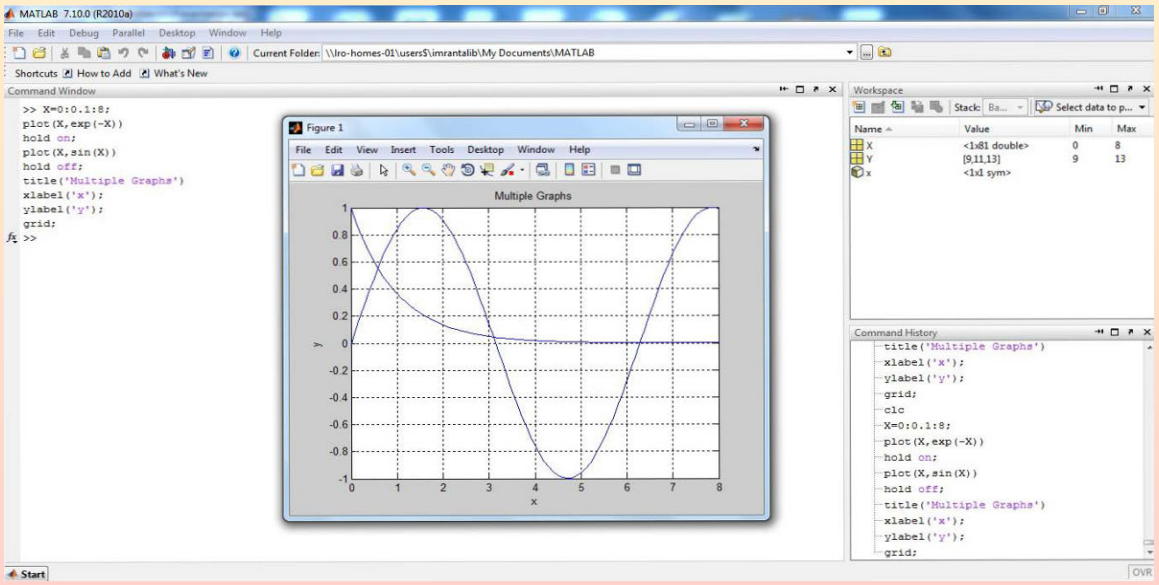


Figure: plot command view of multiple curves

Parametric Plots

- We observe the flexibility of the **plot** command by parametric plot of the circle centered at $(0, 0)$, and radius 1.
- The parametric representation of the circle can be expressed with the aid of equations $x = cos(2\pi t)$ and $y = sin(2\pi t)$, t varies from 0 to
- The recipe for graphing the parametric plots is:
- » t=0:0.1:1;
  » plot(cos(2*pi*t),sin(2*pi*t))
  » xlabel('x');
  » ylabel('y');
  » title('Parametric plot of circle');
  » grid;
  » axis square;

Parametric plot of the circle

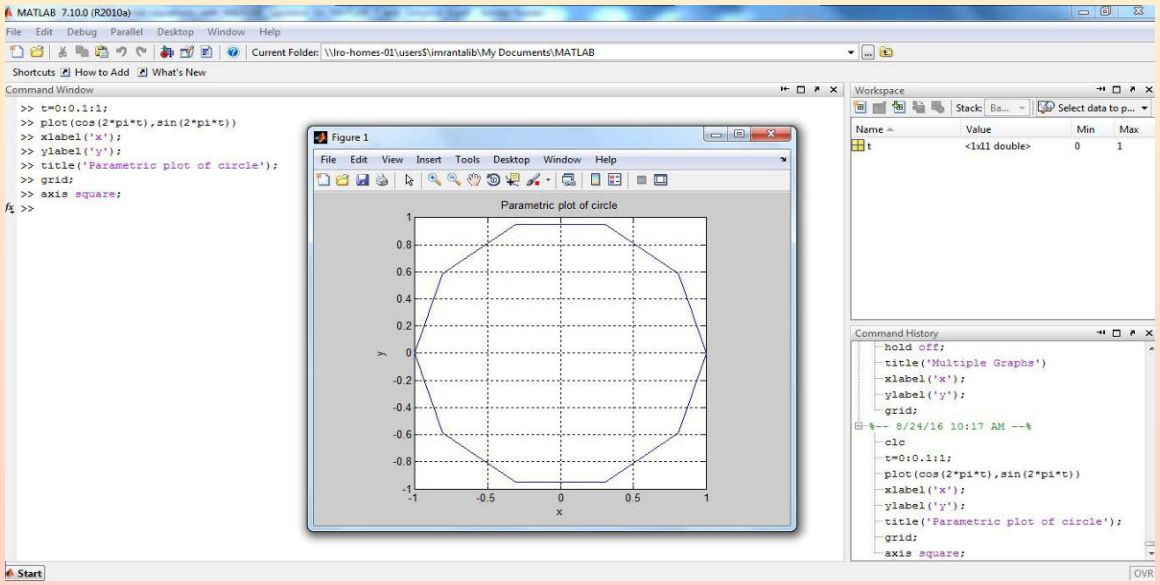


Figure: Parametric plot of the circle using plot command

Parametric Plots

Example:

- Plot the parametric graph of the circle with center at $(0, 0)$, radius 1, and with increment of 0.01 in the domain set.
- Label the horizontal axis with x and vertical axis with y.
- Make the shape of the graph square.
- Keeps the title of the graph "Parametric plot of the circle".

Parametric Plots

Solution:

- The recipe for graphing the parametric plots is:
- » t=0:0.01:1;
  » plot(cos(2*pi*t),sin(2*pi*t))
  » xlabel('x');
  » ylabel('y');
  » title('Parametric plot of circle');
  » grid;
  » axis square;
- The results are presented in Figure 10

Parametric plot of the circle

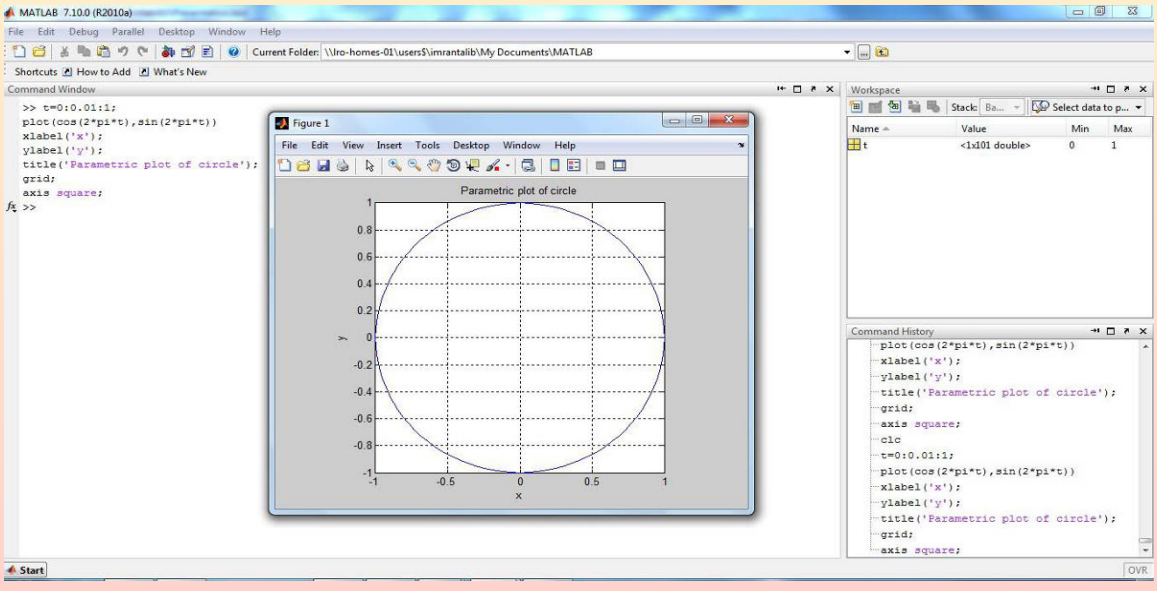


Figure: Parametric plot of the circle using plot command

Contour Plot

- A contour plot of an expression in two variables is a plot of the level curves of the expression.
- For example, the level curves of the expression $x^2 + y^2$ are the circles.
- **meshgrid** and **contour** are the two commands used for the contour plotting.
- **meshgrid** command is used to produce a grid of points in a specified rectangular region, with a specified spacing.
- **contour** command is used to produce a contour plots in the specified regions.

Contour Plots

Example:

- Plot the contour plots of the expression $x^2 + y^2$ using **meshgrid** and **contour** commands.
- Label the horizontal axis with x and vertical axis with y.
- Make the shape of the graph square.
- Keeps the title of the graph "Contour plotting of squares".

Contour Plots

Solution:

- The recipe for graphing the contour plots is:
- » [X,Y] = meshgrid(-2:0.1:2,-2:0.1:2);
  » contour($X, Y, X.^2 + Y.^2$)
  » axis square
  » xlabel('x');
  » ylabel('y');
  » grid;
  » title('Contour plotting of squares');

Contour Plot

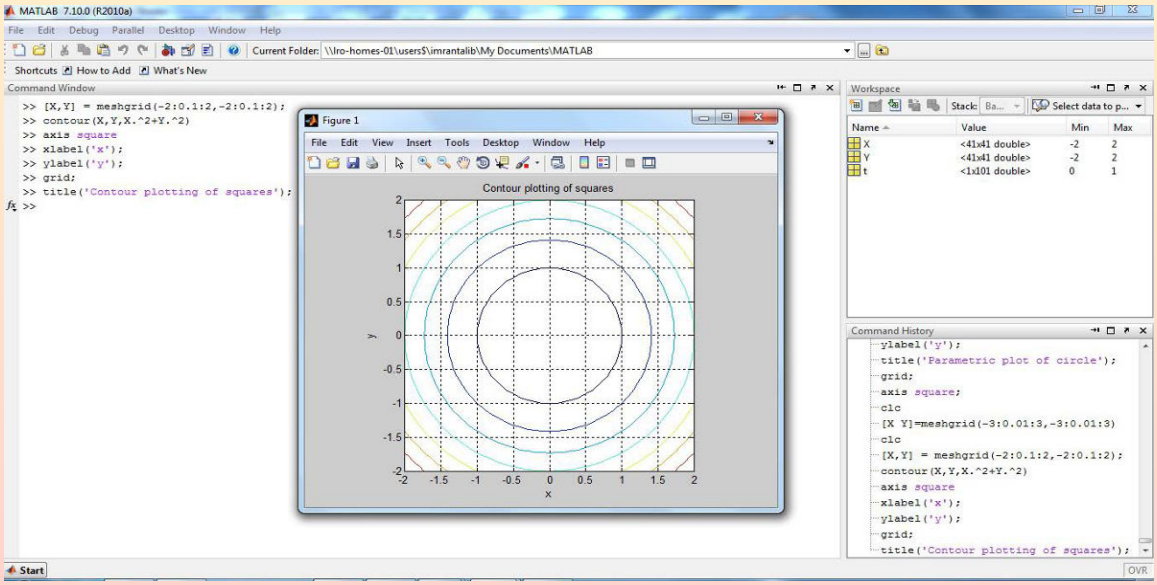


Figure: Contour plotting

Exercise

- Plot the graph of the function $x^3 + x^2 + 4$ on the interval $[-2, 2]$, using **ezplot** command.

  Label the horizontal axis with x and vertical axis with y.
  Make the shape of the graph square.
  Keeps the title of the graph "The Graph of $x^3 + x^2 + 4$ ".
  Keeps the layout of the graph as grid.

Exercise

- Plot the graph of the function $sin(x) + cos(x)$ on the interval $[0, 1]$, using **plot** command.

  Label the horizontal axis with x and vertical axis with y.
  Make the shape of the graph square.
  Keeps the title of the graph "The Graph of $sin(x) + cos(x)$ ".
  Keeps the layout of the graph as grid.

Exercise

- Plot the multiple curves of the functions $sin(x) + cos(x)$ and $e^{-x} + x^2$ on the interval $[0, 4]$, using **ezplot** command.

  Label the horizontal axis with x and vertical axis with y.
  Make the shape of the graph square.
  Keeps the title of the graph "The Graph of Multiple curves using ezplot command ".
  Keeps the layout of the graph as grid.

Exercise

- Plot the multiple curves of the functions $sin(x) + cos(x)$ and $e^{-x} + x^2$ on the interval $[0, 4]$, using **plot** command and without the use of **hold on** and **hold off** commands.

  Label the horizontal axis with x and vertical axis with y.
  Make the shape of the graph square.
  Keeps the title of the graph "The Graph of Multiple curves using plot command ".
  Keeps the layout of the graph as grid.

Exercise

- Plot the contour plots of the circles $x^2 + y^2$ of radius $1,\ 2, \sqrt{2},\ \sqrt{3}$.
  Label the horizontal axis with x and vertical axis with y.
  Make the shape of the graph square.
  Keeps the title of the graph "Contour plots of circles ".
  Keeps the layout of the graph as grid.

Plotting in MATLAB

Ezplot, Plot and FPlot

- We have used different plotting command to plot the curve in MATLAB.

- We will go through these command and check what are the differences between these.

ezplot Command

- ezplot(fun) plots the expression fun(x) over the default domain $-2\pi < x < 2\pi$, where fun(x) is an explicit function of only x.

- ezplot(fun)
- ezplot(fun,[xmin,xmax])
- ezplot(funx,funy)

ezplot

For example

Passing the Function as a Character Vector or String

```
ezplot('x^2')
```

```
Ezplot('x^2-y^4')
```

Passing a Function Handle

```
fh = @(x,y) x.^2 + y.^3 - 2*y - 1;
ezplot(fh)
```

plot Command

- Plot function also plots a 2-D line plot.
- plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.

- plot(X,Y,LineSpec) sets the line style, marker symbol, and color.
- plot(X1,Y1,…,Xn,Yn) plots multiple X, Y pairs using the same axes for all lines.

plot

- plot(X1,Y1,LineSpec1,…,Xn,Yn,LineSpecn) sets the line style, marker type, and color for each line. You can mix X, Y, LineSpec triplets with X, Y pairs.
- For example, plot(X1,Y1,X2,Y2,LineSpec2,X3,Y3).

- x = 0:pi/100:2*pi;
- y = sin(x);
- plot(x,y)

- x = linspace(-2*pi,2*pi);
- y1 = sin(x);
- y2 = cos(x);

- plot(x,y1,x,y2)

- Y = magic(4)
- plot(Y)

Specify Line Style

- x = 0:pi/100:2*pi;
- y1 = sin(x);
- y2 = sin(x-0.25);
- y3 = sin(x-0.5);

- figure
- plot(x,y1,x,y2,'--',x,y3,':')

Specify Line Style, Color, and Marker

- x = linspace(0,10);
- y = sin(x);
- plot(x,y,'-o','MarkerIndices',1:5:length(y))

Line Style

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

Marker

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

Color

Color	Description
y	Yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white

fplot Command

- Plot expression or function
- fplot(f) plots the curve defined by the function y = f(x) over the default interval [-5 5] for x.
- fplot(f,xinterval) plots over the specified interval. Specify the interval as a two-element vector of the form [xmin xmax].

- fplot(funx,funy) plots the curve defined by x = funx(t) and y = funy(t) over the default interval [-5 5] for t.
- fplot(funx,funy,tinterval) plots over the specified interval. Specify the interval as a two-element vector of the form [tmin tmax].
- fplot(___,LineSpec) specifies the line style, marker symbol, and line color. For example, '-r' plots a red line. Use this option after any of the input argument combinations in the previous syntaxes.

Examples

- Plot sin(x) over the default x interval [-5 5].
- fplot(@(x) sin(x))

- Plot Parametric Curve

- xt = @(t) cos(3*t);
- yt = @(t) sin(2*t);
- fplot(xt,yt)

Specify Plotting Interval and Plot Piecewise Functions

```
fplot(@(x) sin(x+pi/5),'Linewidth',2);
hold on
fplot(@(x) sin(x-pi/5),'--or');
fplot(@(x) sin(x),'-.*c')
hold off
```

Plot Multiple Lines in Same Axes

- fplot(@(x) sin(x))
- hold on
- fplot(@(x) cos(x))
- hold off

3D Curve Plotting in MATLAB

fplot3

- 3-D parametric curve plotter.

- fplot3(funx,funy,funz)
- fplot3(funx,funy,funz,tinterval)
- fplot3(___,LineSpec)
- fplot3(___,Name,Value).

Example

- Plot the 3-D parametric line

x=sin(t)

y=cos(t)

z=t

over the default parameter range [-5 5].

```
xt = @(t) sin(t);
yt = @(t) cos(t);
zt = @(t) t;
fplot3(xt,yt,zt)
```

Specify Parameter Range

x=e^(–t/10)sin(5t)

y=e(–t/10)cos(5t)

z=t

over the parameter range [-10 10] by specifying the fourth input argument of fplot3.

```
xt = @(t) exp(-t/10).*sin(5*t);
yt = @(t) exp(-t/10).*cos(5*t);
zt = @(t) t;
fplot3(xt,yt,zt,[-10 10])
```

Specify Line Properties and Display Markers

- fplot3(@(t)sin(t), @(t)cos(t), @(t)t, [0 2*pi], 'LineWidth', 2)
- hold on
- fplot3(@(t)sin(t), @(t)cos(t), @(t)t, [2*pi 4*pi], '--or')
- fplot3(@(t)sin(t), @(t)cos(t), @(t)t, [4*pi 6*pi], '-.*c')
- hold off

plot

- plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn) sets the line style, marker type, and color for each line. You can mix X, Y, LineSpec triplets with X, Y pairs.
- For example, plot(X1,Y1,X2,Y2,LineSpec2,X3,Y3).

- x = 0:pi/100:2*pi;
- y = sin(x);
- plot(x,y)

fsurf

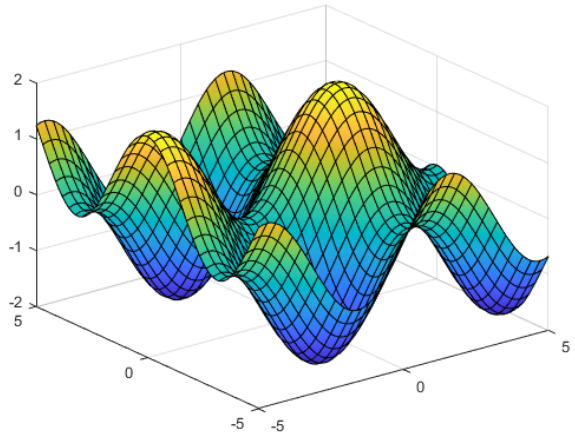
- fsurf(f)

  fsurf(f) creates a surface plot of the function z =f(x,y) over the default interval [-5 5] for x and y.
- fsurf(f,xyinterval)
- fsurf(funx,funy,funz)

  plots the parametric surface defined by x = funx(u,v), y = funy(u,v), z = funz(u,v) over the default interval [-5 5] for u and v.
- fsurf(funx,funy,funz,uvinterval)
- fsurf(___,LineSpec)
- fsurf(___,Name,Value)

- Plot the expression sin(x)+cos(y) over the default interval –5<x<5 and –5<y<5.

  fsurf(@(x,y) sin(x)+cos(y))



Specify Interval of Surface Plot and Plot Piecewise Expression

Plot the piecewise expression

erf(x)+cos(y) $-5<x<0$

sin(x)+cos(y) $0<x<5$

over $-5<y<5$.

Specify the plotting interval as the second input argument of fsurf. When you plot multiple surfaces over different intervals in the same axes, the axis limits adjust to include all the data.

Code

```
f1 = @(x,y) erf(x)+cos(y);
fsurf(f1,[-5 0 -5 5])
hold on
f2 = @(x,y) sin(x)+cos(y);
fsurf(f2,[0 5 -5 5])
hold off
```

Parameterized Surface Plot

Plot the parameterized surface

x=rcos(u)sin(v)
y=rsin(u)sin(v)
z=rcos(v)
wherer=2+sin(7u+5v)

for 0<u<2π and 0<v<π. Add light to the surface using camlight.

Code

```
r = @(u,v) 2 + sin(7.*u + 5.*v);
funx = @(u,v) r(u,v).*cos(u).*sin(v);
funy = @(u,v) r(u,v).*sin(u).*sin(v);
funz = @(u,v) r(u,v).*cos(v);
fsurf(funx,funy,funz,[0 2*pi 0 pi])
camlight
```

fmesh

- fmesh(f)

  creates a mesh plot of the expression z = f(x,y) over the default interval [-5 5] for x and y.
- fmesh(f,xyinterval)
- fmesh(funx,funy,funz)

  plots the parametric mesh defined by x = funx(u,v), y = funy(u,v), z = funz(u,v) over the default interval [-5 5] for u and v.
- fmesh(funx,funy,funz,uvinterval)
- fmesh(___,LineSpec)

Example

- Plot a mesh of the input sin(x)+cos(y) over the default interval $-5<x<5$ and $-5<y<5$.

- fmesh(@(x,y) sin(x)+cos(y))

Parameterized Mesh Plot

Plot the parameterized mesh

x=rcos(s)sin(t)

y=rsin(s)sin(t)

z=rcos(t)

wherer=2+sin(7s+5t)

for 0<s<2π and 0<t<π. Make the mesh partially transparent using alpha.

Code

```
r = @(s,t) 2 + sin(7.*s + 5.*t);
x = @(s,t) r(s,t).*cos(s).*sin(t);
y = @(s,t) r(s,t).*sin(s).*sin(t);
z = @(s,t) r(s,t).*cos(t);
fmesh(x,y,z,[0 2*pi 0 pi]
```

Use Of the M-Files

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Outline

- Up to now, we are working in **Command Window**. But for the complicated problems, **Command Window** is insufficient to produce the results.
- To handle the coding of the complicated problems, a much better approach is to create an **M–File**.
- MATLAB uses two different kinds of **M–Files**. One is script **M–File** and the other is function **M–File**.
- We learn the use of both types of **M–Files**.

- **M–Files** are ordinary text files containing **MATLAB** commands.
- We can create and modify the **MATLAB** commands using **MATLAB's** built in **M–File** editor, known as **Editor/Debugger**.
- This **Editor** automatically checks for certain **MATLAB** syntax requirements, and highlights different parts of the file in special colours.
- This **Editor** can be opened with the **"New M–File"** or **"Open"** icons on the tool bar.
- It can also be opened by typing **edit** at the command prompt.
- In the next slide, the **Editor** is opened writing **edit** in the command prompt.

Editor Window along with Command Window

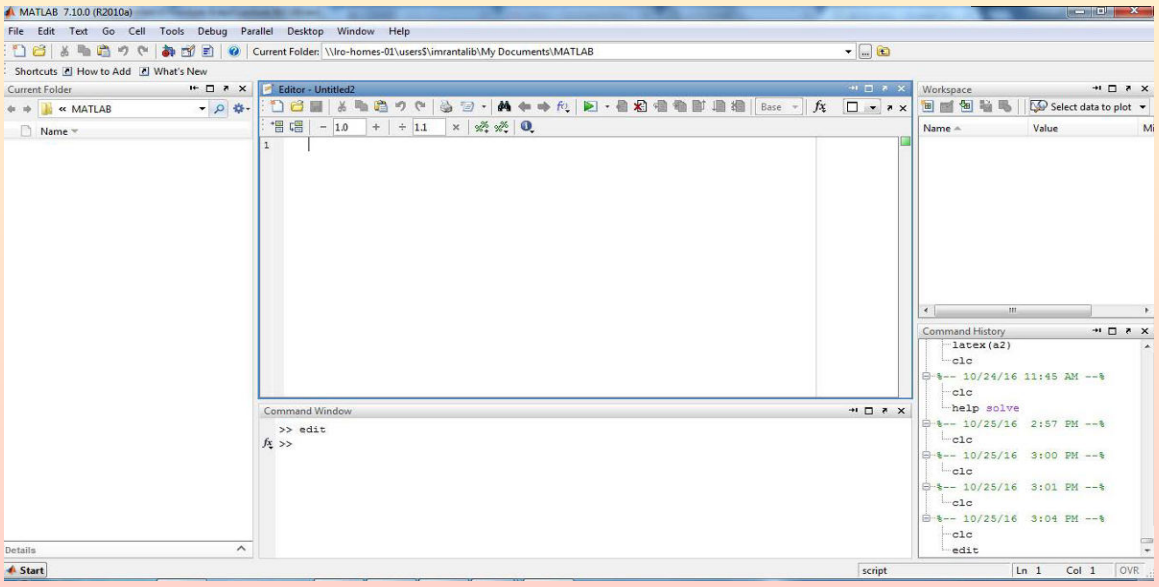


Figure: Display of the MATLAB desktop with Editor Window

- Now, we are going to learn how to construct a script M–File to solve different kind of problems.
- For this we use the **Editor/Debugger** to create the script M–File.
- For more clarity, consider an example

Example:

- Calculate $\frac{sin(x)}{x}$ at $x = 0.1, 0.01, 0.001, 0.0001, 0.00001$ using **Editor/Debugger**. Moreover display the outputs up to 15 digits using **format long** command.

Solution:

- Use **Editor/Debugger** to create the file containing the following lines:
- $x = [0.1, 0.01, 0.001, 0.0001, 0.00001];$
  $y = sin(x)./x.$
- To see the output it is necessary to save and execute the above code.
- The execution and saving procedure is explained in the next slide.

- Now save the file with the name **scriptfile1.m** in your working directory, or in some directory on your path, using the **Save AS**, which is a menu item in the **File** menu of the **editor**.
- Note that you can also save the file with the name what ever you like, but the **".m"** suffix is mandatory to write.
- Note that **editir** add this suffix automatically.

- The Script file can be executed or run by writing **scriptfile1** in the **Command Window**.
- Note that while the execution of the Script file, there is no need to write **scriptfile1.m** in the Command Window. Just write only **scriptfile1**.
- The required out put will be displayed in the **Command Window**.
- Remember that the **Command Window** does not give us liberty of editing.
- But **Editor Window** has advantage of editing.
- From now to onward, we will write our all codes in **Editor Window** instead of **Command Window**.

- Note that the out put will be displayed in the command window not in the **Editor/Debugger** window.
- The desire out put is as follows:
- y =
  0.9983 1.0000 1.0000 1.0000 1.0000
- Above discussion is summarized in the next slide.

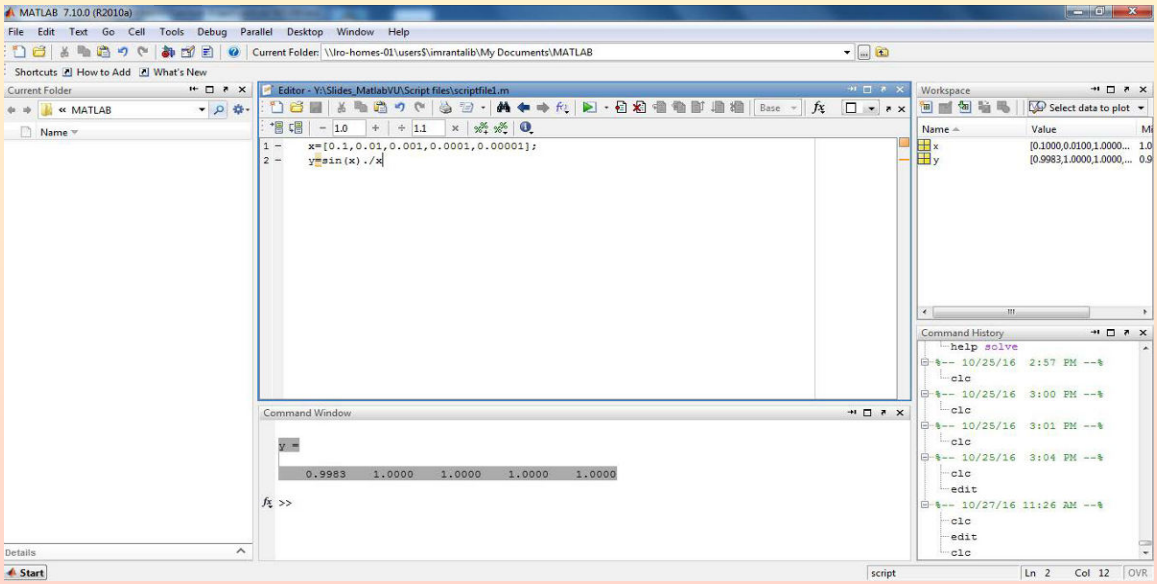


Figure: Computational values of $\frac{sin(x)}{x}$

Echoing Commands

- Note that in the previous discussion we observe that the commands in the **Script M–File** were not automatically be displayed in the **Command Window**. Only results are to be displayed.
- But **echo Command** displays the Commands along with the results.
- **echo** command works as;
- echo on
  format long
  $x = [0.1, 0.01, 0.001, 0.0001, 0.00001]$;
  $y = sin(x)./x$.
  echo off
- For more clarity see the next Figure.

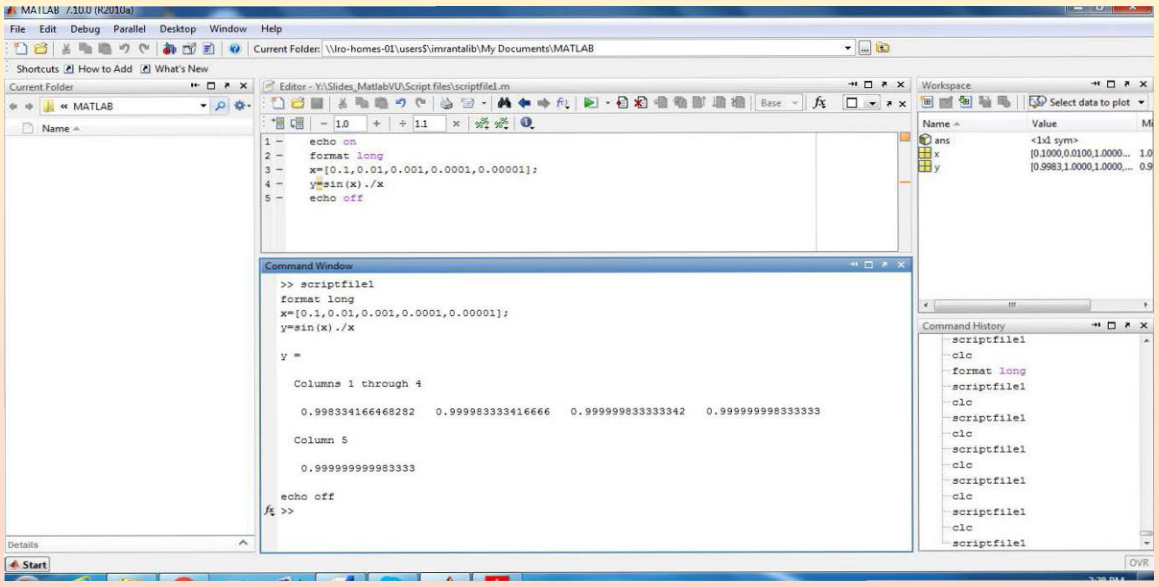


Figure: Display of the Commands along with the results in the Command Window using echo command.

- It is very much important to include comments in the **Script M–File** to explain what is being done in the calculation.
- Any line in a **Script M–File** that begins with a **percent sign** is treated as a comments and is not executed by MATLAB.
- For more clarity see the solution of the above example with comments.

Solution with adding comments

- echo on
- format long
- x=[0.1,0.01,0.001,0.0001,0.00001];
- $y = sin(x)./x$
  echo off
- For more clarity see the next slide

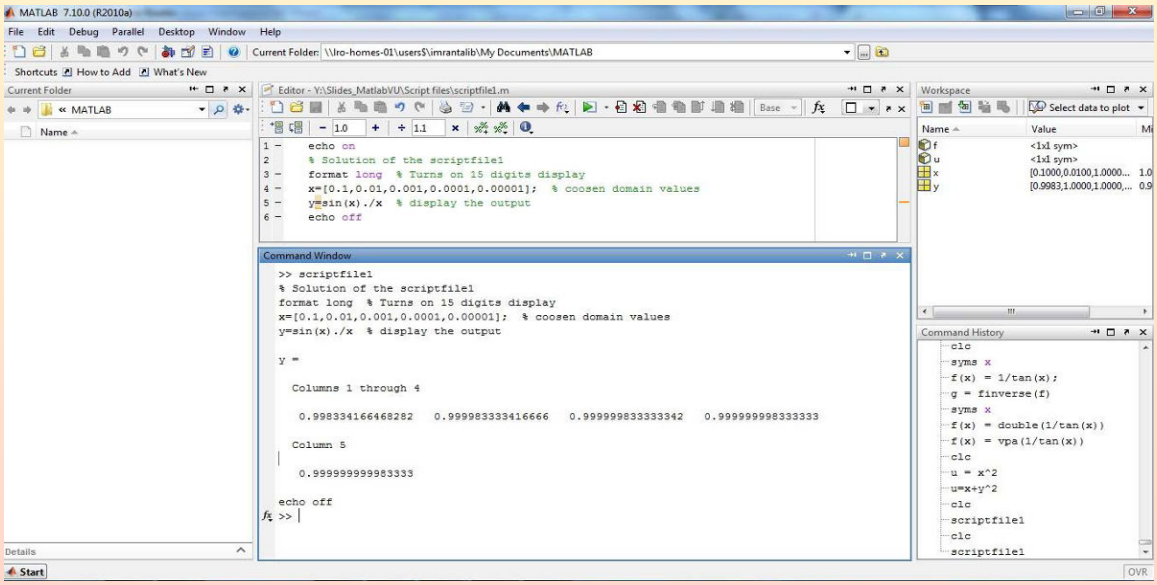


Figure: Computational of the values of $y = sin(x)/x$ at given domain values with adding comments.

- Make sure that your current MATLAB session is not affecting by other variables that had been defined in the last session. That is make sure that your present **Script M–File** is self contained, unaffected by other variables, and uncorrupted by left over graphics.
- For this ensuring, **clear all** command should be used. This command ensure that previous definitions of variables do not affect the results.
- Then in general your **Script M–File** should usually start with the **clear all** command.
- You should also either include the **close all** command at the beginning of an **M–File** that creates graphics.
- This command will automatically will close all **graphics Windows** and start with a clean slate.
- For more clarity observe the new version of the above example with **clear all** and **close all** commands in the next slide.

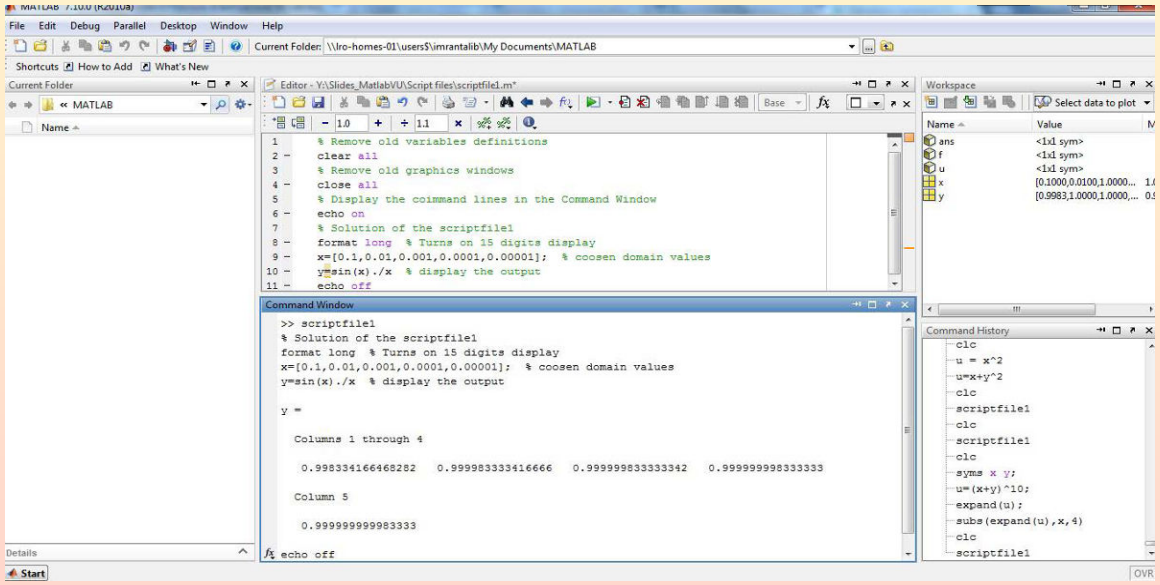


Figure: Structuring of the solution file with clear all and close all command.

- Sometimes, it is necessary to repeat a process several times for different input values of a parameter.
- For instance, we can provide different inputs to a built-in function in order to find an output that meets a given criterion.
- We have already defined our own functions using **@** syntax or the **inline command**.
- In many situations, however, it is more convenient to define a function using an **M–File** instead of an anonymous or inline function.
- **M–File** allows us to extend the functionality of MATLAB by defining our own functions.

Solution of the above example using **sinepower** function

- The solution of the previous example using **M–File** is as under:
- function y = sinepower(c)
  format long
  b=1:c;
  $y = (sin(10.^{(-b)})./10.^{(-b)})'$;
- Now save this **M–File** with sinepower name.
- Then execute this file by typing sinepower(c) in the **Command Window**. But, make sure that the numerical value has been assigned to **c**. Otherwise, it gives execution error.
- For more clarity see the next slide.

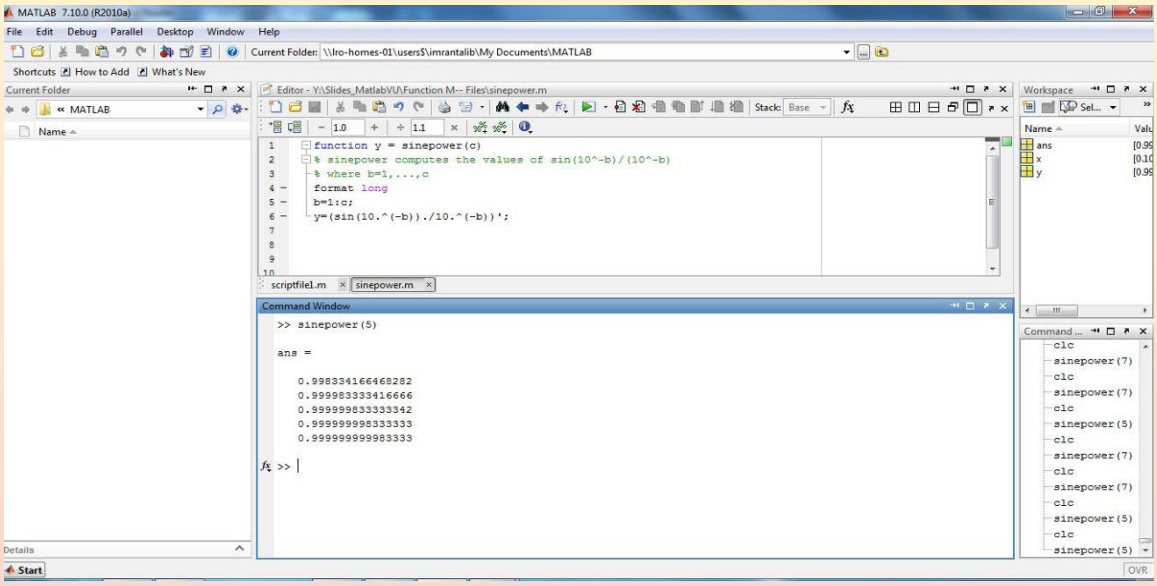


Figure: Solution of the previous example using Function M–File.

- A loop specifies that a command or group of commands should be repeated several times.
- The easiest way to create a loop is to use a **for** statement.
- For more clarity see the upcoming example:

For Loop

Example:

- Compute the values of $sin(x)$ at $x = 0, 1,$ and 2 using **for** loop.

Solution:

- The recipe of the solution is as under
- format long
  for i= 0:2
  $y = sin(i)$
  end
  y
- For more clarity see the next slide.

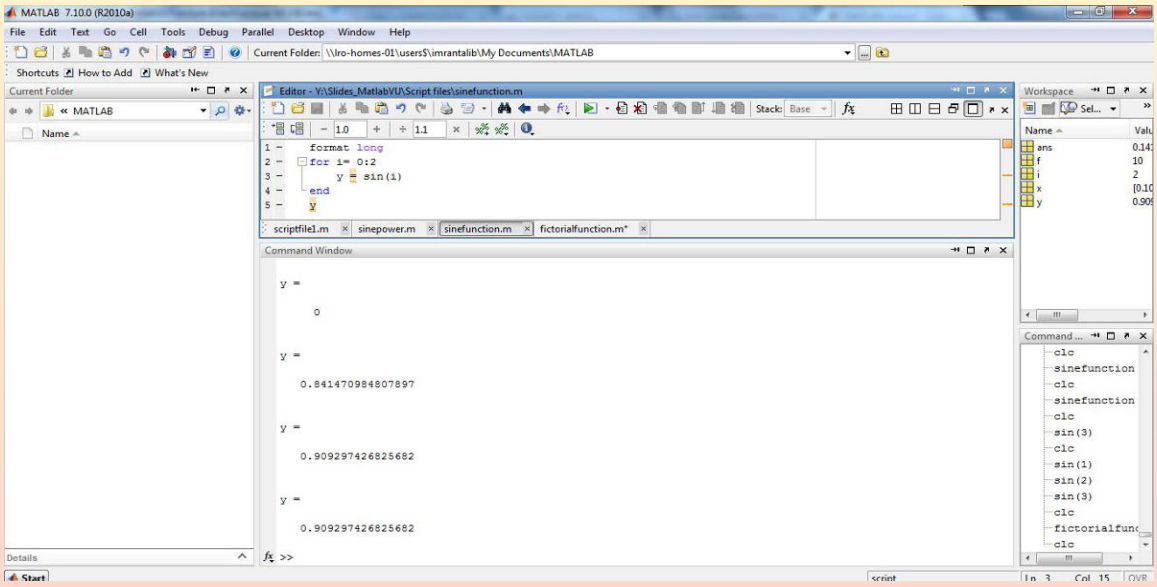


Figure: Computation of the values of sine function using for loop.

Discussion

- The loop begins with the **for** statement and ends with end **statement**.
- The command between those statements is executed a total of three times, once for each value of i from 0 to 2.
- The **Editor** automatically colors the commands **for** and **end** in blue.

Solve the following algebraic equations using **Script M–File**.

- $x^2 + 2x - 4 = 0$.
- $x^3 + 5x^2 + 4x + 3 = 0$

Solve the following equations for x using **Script M–File**.

- $x + log(y) = 3$.
- $x + 10y = 6$.

Solve the following system of equations using **Script M–File**.

- $x + y^2 = 2,\ y - 3x = 7$.
- $x + y^2 = 2,\ 2y^2 - 3x = 7$.

Exercises

Problem:

- Find the first-order, second-order, and fourth-order derivatives of the function $f(x) = x^{10} + 2 * (x^2) + sin(x)$ using **diff** command.
- Plot the multiple graphs of the outputs using **ezplot** command on the interval $[-\pi, \pi]$.
- Label the horizontal axis with x and vertical axis with y.
- Keeps the title "Multiple plot of the function and its derivatives"
- All the coding should be written using **Script M–File**.

Exercises

Problem:

- Find the first-order, fourth-order, and fifth-order derivatives of the function $f(x) = \ln x + sin(x)$ using **diff** command.
- Plot the multiple graphs of the outputs using **plot** command on the interval $[0, 1]$ and without using **hold on** and **hold off** commands
- Label the horizontal axis with x and vertical axis with y.
- Keeps the title "Multiple plot of the function and its derivatives"
- All the coding should be written using **Script M–File**.

Exercises

Problem:

- Find the first-order, fourth-order, and fifth-order derivatives of the function $f(x, y) = \ln x * y + sin(x^2)$ using **diff** command.
- Plot the multiple graphs of the outputs using **plot** command on the interval $[0, 1]$ and without using **hold on** and **hold off** commands
- Label the horizontal axis with x and vertical axis with y.
- Keeps the title "Multiple plot of the function and its derivatives"
- All the coding should be written using **Script M–File**.

Input Statements

Entering input into a program

- There are multiple ways to enter input variables into a MATLAB program. Three most common methods are as follows:

  I. Use hard-coded variables in MATLAB program.
  II. Use input statements within the program to prompt the user in the MATLAB command window to enter input variable.
  III. Use Menu statements within the program to prompt user to click on a button to select an input.

Using Hard Coded Variables

- Whenever, we need a value in the program, we simply create a variable
- >>x=7;
- >>food='Pizza'

- Why not use hard coded variable all the time?
  I. Does not allow the user to choose different values.
  II. Program is only useful for one specific case.
  III. You might not know the needed value beforehand

Using Input Statements for Number

- To allow the user to choose a value is to use an input statement:

  >>variable=input('message')

Using Input Statements for String

- If you want to allow the user to enter a letter or a word instead of a number then the statement

>>variable= input('please enter a letter: ')

Please enter a letter: a

Error using input

Undefined function or variable a

- Add the second argument to the input statement to specify that the value will be string:

>>variable=input('Please enter a letter:', 's')

Examples

>>radius=input('Please enter the radius: ')

Please enter the radius: 3

>>disp(radius);

>>name=input('Please enter the name: ', 's')

Please enter the name: XYZ

Using Menu Statement

- A menu statement can be used to allow the user to select form a set of predefined choices.
- When you use a menu statement, MATLAB will create a new window with a set of buttons.

  >>variable1=menu('Message','opt1','opt2','opt3',...)

- The order you list the option in your message statement determines the order the buttons are displayed.
- The first button has value 1 and second 2 and so on.

Output Statements

Displaying the Outputs of a Program

There are a variety of ways to display the output of a MATLAB program. Three methods will be introduced initially:

1. Do not use ***suppression*** for a given statement in the MATLAB program.
2. Use ***display statements*** (disp) within the program that will display the desired outputs in the MATLAB command window.
3. Use ***fprintf statements*** within the program that will that will display the desired outputs in the MATLAB command window and/or write the resulting outputs to a file.

Unsuppressing Statement

- The simples way to display a value in MATLAB is not to suppress it:
- When you place a ; at the end of a line in MATLAB, you are telling MATLAB that you do not want to display the result in command window.
- If you leave ; at the end of a statement then MATLAB will show you the result.

Using Displaying Statement

- If you want to display additional information then you can use disp('message') command.
- You can use constant or variable in disp command to be displayed.

Example

If we want to display

```
>>radius=input('Please enter the radius: ');
>>Circumference= 2*pi*radius;
>>Area=pi*radius^2;
>>disp('Circumference : '); disp(circumferenece);
>>disp('Area'); disp(Area)
```

fprintf statement

- If we want additional formatting, we can use the fprintf statement:

- Mix string with values
- Specify the number of decimal places
- **Insertion formats**
- %s –insert text
- %f-insert a floting number

Insertion formats

- %f

  Fixed-point notation (Use a precision operator to specify the number of digits after the decimal point.)

- %e

  Exponential notation, such as 3.141593e+00 (Use a precision operator to specify the number of digits after the decimal point.)

- %E

  Same as %e, but uppercase, such as 3.141593E+00 (Use a precision operator to specify the number of digits after the decimal point.)

- %g

  The more compact of %e or %f, with no trailing zeros (Use a precision operator to specify the number of significant digits.)

- %G

  The more compact of %E or %f, with no trailing zeros (Use a precision operator to specify the number of significant digits.)

Special Characters

Special Characters	Representation
Single quotation mark	``
Percent character	%%
Backslash	\\
Backspace	\b
New line	\n
Horizontal tab	\t
Vertical tab	\v

Example

```
>>day=`Wednesday`; weather=`Sunny`; temp=75;

>>fprintf(`Today is %s \n`, day)

>>fprintf(`The weather today is %s  and the temperatue
is %i \n'`,weather,temp)

>>fprintf(`The weather today is %s  and the temperatue
is %0.3f \n'`,weather,temp)
```

Print Double-Precision Values as Integers

- >>a = [1.02 3.04 5.06];
- >>fprintf('%d\n',round(a));

%d in the formatSpec input prints each value in the vector, round(a), as a signed integer. \n is a control character that starts a new line.

Print Literal Text and Array Values

```
>>A1 = [9.9, 9900];
>>A2 = [8.8,  7.7 ];
>>formatSpec = 'X is %4.2f meters or %8.3f mm\n';
>>fprintf(formatSpec,A1,A2)
```

Write Tabular Data to Text File

```
>>x = 0:.1:1;
>>A = [x; exp(x)];

>>fileID = fopen('exptest.txt','w');
>>fprintf(fileID,'%6s %12s\n','x','exp(x)');
>>fprintf(fileID,'%6.2f %12.8f\n',A);
>>fclose(fileID);
```


Matrix Algebra and MATLAB

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Matrices in MATLAB
Matrices
Matrix Algebra
Exercise

Outline

1. Matrices in MATLAB
   - Creating Vector and Matrix Variables
2. Matrices
   - Some useful operations on matrices
3. Matrix Algebra
   - Matrices Multiplication
   - Determinant of a matrix
   - Inverse of a matrix
4. Exercise

Outline

Matrices in MATLAB
Matrices
Matrix Algebra
Exercise

Outline

1. Matrices in MATLAB
   - Creating Vector and Matrix Variables
2. Matrices
   - Some useful operations on matrices
3. Matrix Algebra
   - Matrices Multiplication
   - Determinant of a matrix
   - Inverse of a matrix
4. Exercise

Matrices can be introduced into MATLAB in several ways:

- Entered by an explicit list of elements.
- Generated by so called built–in statements and functions.
- Loaded by external data.
- Created by M–Files.

Generating matrices as a list of its elements. The following basis conventions have to be taken into account:

- The elements of a row are separated by commas or spaces.
- A semi-colon indicates the end of each row.
- The entire list of elements has to be surrounded by brackets [ ].

Row Vector

- Row vector variables can be created in several ways.
- The simplest method is to put the values that you want in the variable in square brackets, separated by either spaces or commas.
- For instance, a row vector with four elements in MATLAB can be generated as:
- rowvec=[2,4,5,6]
- The required output is as under
- rowvec =
  2 4 5 6

Row vector using Colon Operator

- Row vector can also be generated using Colon operator(;)
- The entries in the generated Row vector iterates from the starting to the ending value with a default step of one.
- In this case the square brackets are not necessary.
- For example, a row vector having $2, 3, 4, 5, 6$ entries can be generated as
- » colvec=2:6
- The necessary output is
- colvec =
  2 3 4 5 6

Colon Operator with Step value

- Row vector with Colon operator can also be generated by defining step value for each entry of the matrix.
- For example, the row vector with entries $2, 4, 6, 8, 10$ can be generated as
- » stepvec = 2:2:10
- The desired output is
- stepvec =
  2 4 6 8 10

- Row vector is generated in different ways

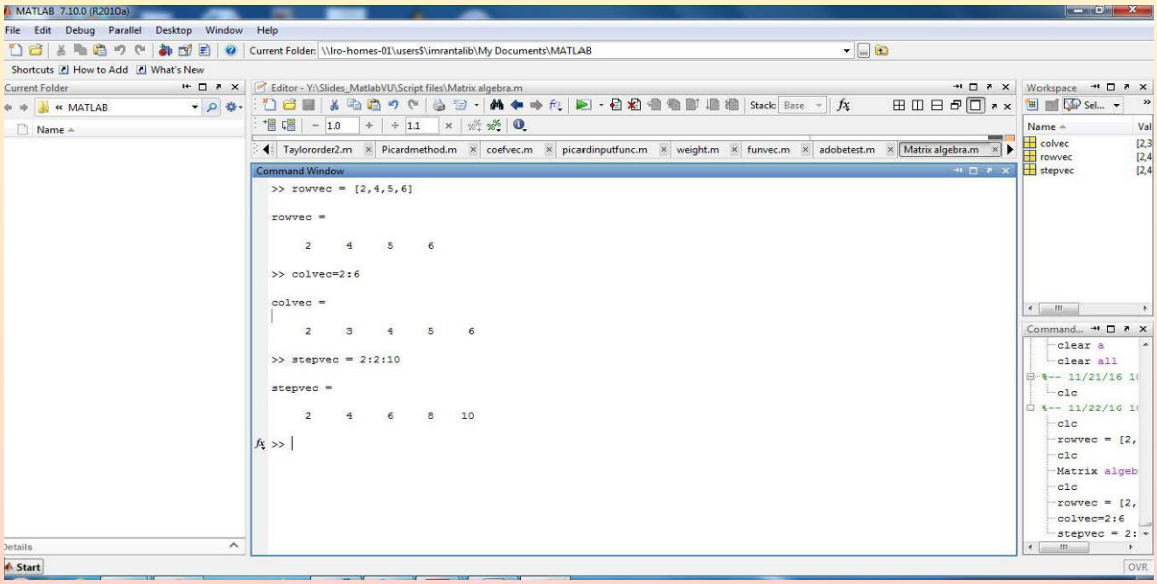


Figure: Display of the Row vector in different ways

Column Vector

- There are two basic methods for creating Column vector:
- Either by putting the values in square brackets, separated by semicolons.
- or by creating a row vector and then transposing it.
- The transpose operator in MATLAB is the apostrophe.

Column Vector with Square bracket and semicolons

- For example, the column vector with entries $2, 5, 10$ can be generated as
- » colvec=[2;5;10]
- The required output is
  colvec =
  2
  5
  10

Column Vector with apostrophe

- Alternatively, the column vector with entries $2, 5, 10$ can also be generated as
- » Rowvec=[2,5,10]
  Rowvec =
  2 5 10
  » colvec=Rowvec'
  colvec =
  2
  5
  10

- Column vector is generated in different ways

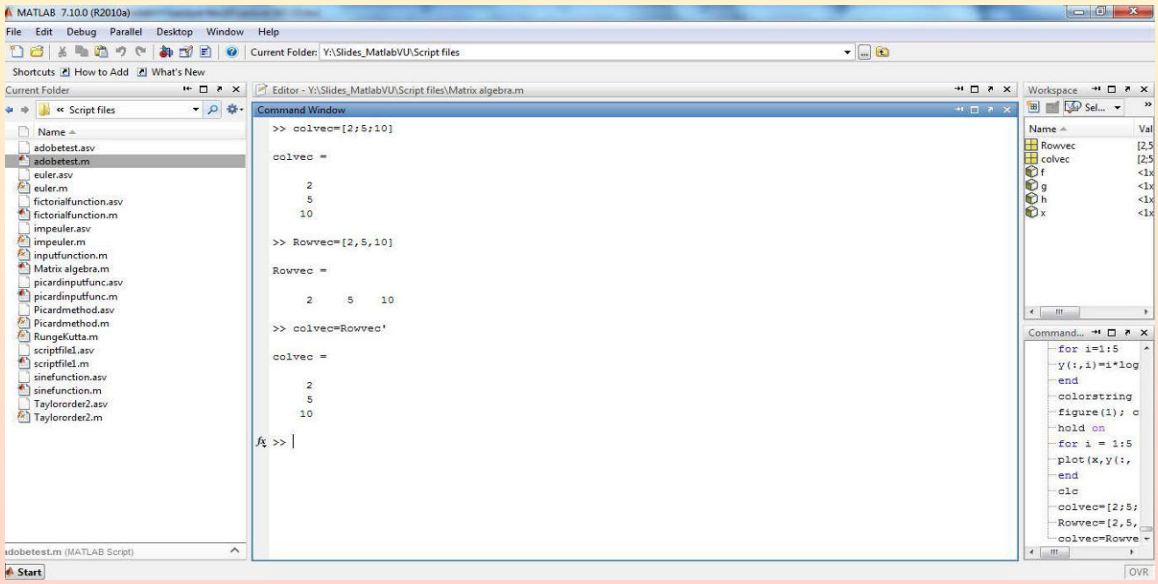


Figure: Display of the Column vector in different ways

- Matrix variables can be created by putting the values in square brackets.
- The values within the rows are separated by either spaces or commas.
- Each individual row is separated by semicolons.
- The must always be the same number of values in every row.
- The colon operator can be used to iterate within the rows.

- For example, the matrix

$$A = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{bmatrix}$$

in MATLAB is generated as

- » A=[5/6,1/6,0;5/6,0,1/6;0,5/6,1/6]
  A =
  0.8333 0.1667 0
  0.8333 0 0.1667
  0 0.8333 0.1667

or

- A=sym([5/6,1/6,0;5/6,0,1/6;0,5/6,1/6])

$$A = \begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$

- Generation of the matrix in MATLAB

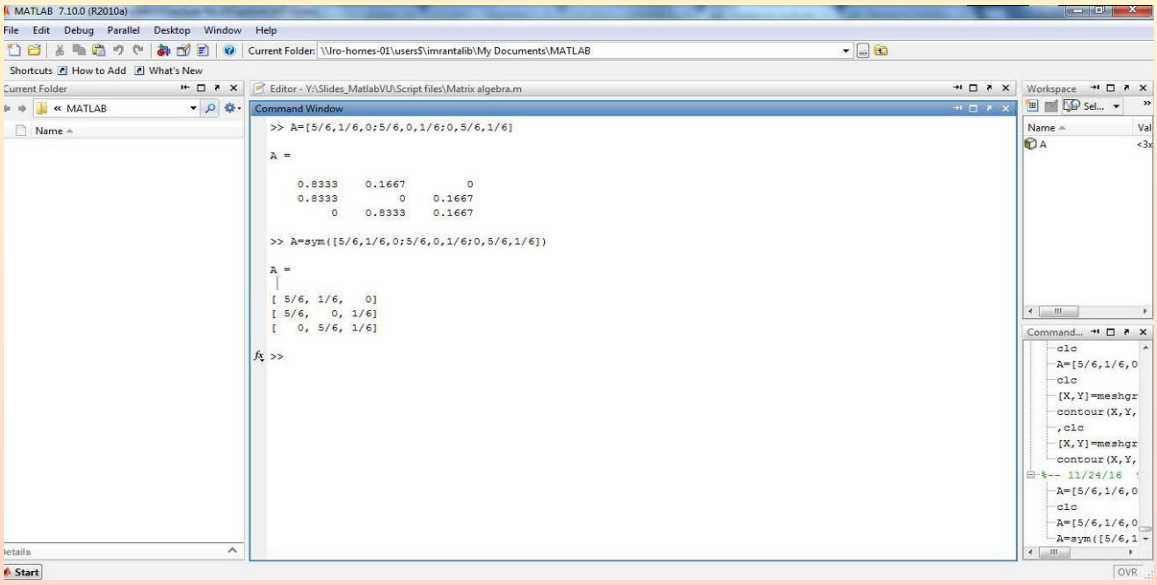


Figure: Graphical view of the matrices

Sum of column vectors of a matrix

- The **sum** command computes a **Row vector** containing the sum of the **columns** of a matrix.
- Note that MATLAB prefers to work with the columns of a matrix.
- For example, the sum of the columns of the matrix A is computed as:
-
- A=sym([5/6,1/6,0;5/6,0,1/6;0,5/6,1/6])

  A =

$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$

  » B = sum(A)

  B =

$$\begin{pmatrix} \frac{5}{3} & 1 & \frac{1}{3} \end{pmatrix}$$

Sum of row vectors of a matrix

- To compute the sum of Row vectors of a matrix, we should take the transpose, then compute the sum and finally transpose the result.
- For example, the sum of the **Row** vectors of the matrix A is calculated in the next slide. Please see it.

- Sum of the entries of the column vectors of A

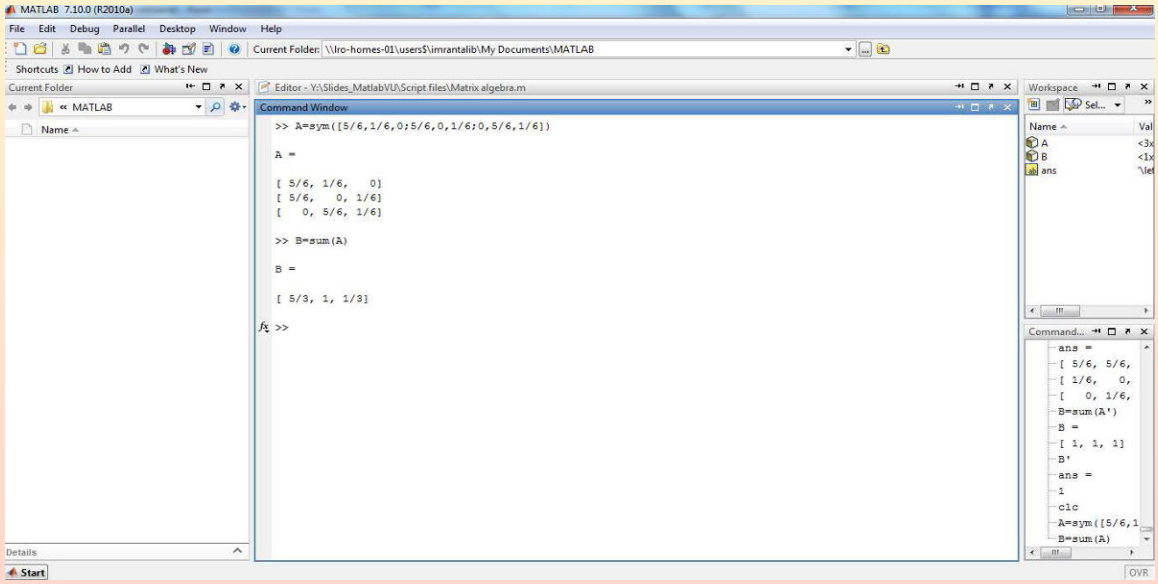


Figure: Display of the sum command

Sum of row vectors of a matrix

- »A = $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$

  » A'= $\begin{pmatrix} \frac{5}{6} & \frac{5}{6} & 0 \\ \frac{1}{6} & 0 & \frac{5}{6} \\ 0 & \frac{1}{6} & \frac{1}{6} \end{pmatrix}$

  » B=sum(A')

  B =

  [1, 1, 1]

  » B'

  ans =

  1

  1

  1.

- Sum of the entries of the row vectors of A

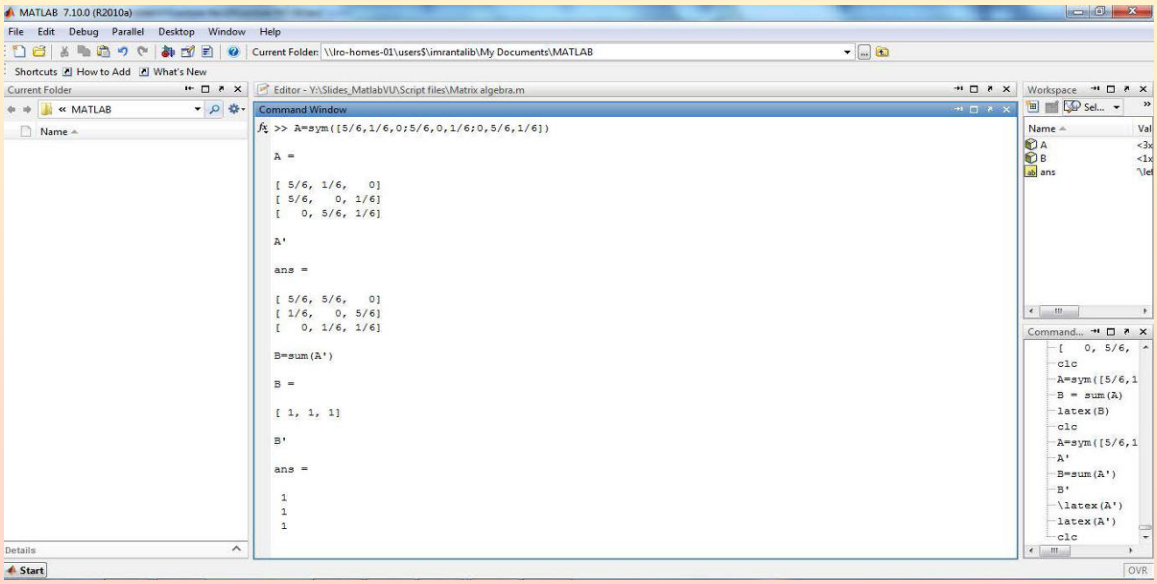


Figure: Display of the sum command

Column vector containing the Diagonal elements of a matrix

- The **diag** command produces a column vector containing the elements of the main diagonal of a matrix.
- For example,
- »A =
$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$
- The MATLAB displays the following result:
- » B=diag(A)
  B =
  5/6
  0
  1/6

Sum of individual row and column

- The collective use of **sum** command and **colon (:)** operator is used to compute the sum of any individual column and row of a matrix.
- The colon operator refers to all elements in a row or column of a matrix.
- The key word **end** refers to last column or row.
- For example, the sum of last column of the matrix A in MATLAB is calculated as
- »A =
$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$
- The MATLAB displays the following result:
- » B=sum(A(:,end))
  B =
  1/3

- Sum of the entries of the last and the second column of A

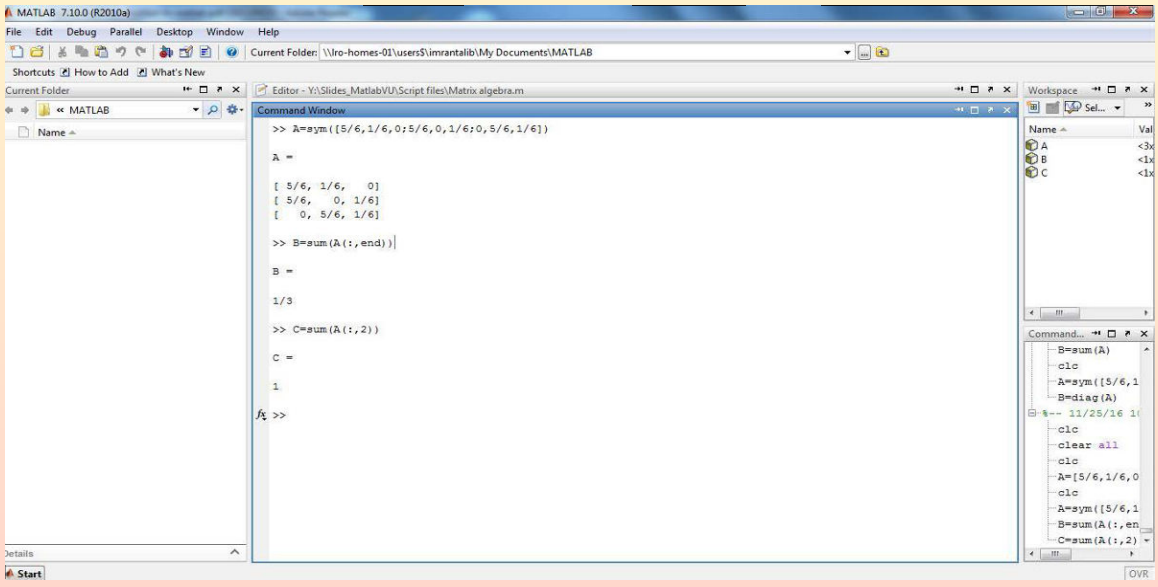


Figure: Display of the sum command with colon operator

Matrices multiplication using for loop

- The matrix multiplication can be defined using MATLAB's **for** loop, colon notation, and vector scalar product.
- For example, the MATLAB syntax for the multiplication of two matrices A and B using for loop is as under:
- clear all;
  close all;
  clc;
  A=[1,2,3;4,5,6];
  B=[1,4;6,7;0,1];
  for i=1:2
  for j=1:2
  C(i,j)=A(i,:)*B(:,j);
  end
  end
  C

- Multiplication of the matrices A and B

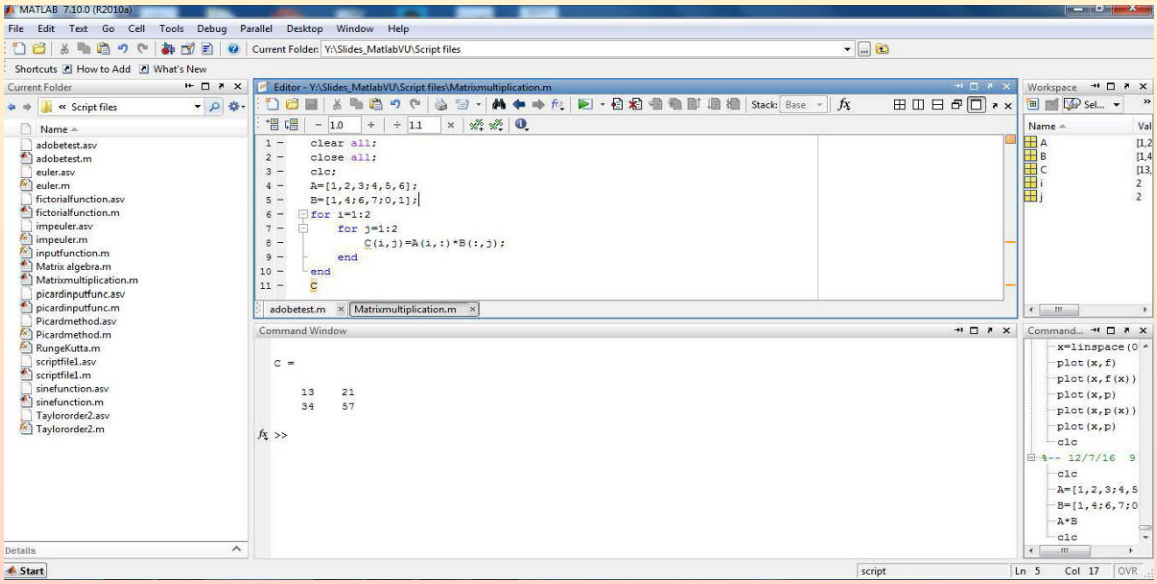


Figure: Matrices multiplication using for loop

Matrices multiplication using asterisk operator

- Single asterisk operator (*) can also be used for the matrices multiplication.
- For example, above defined matrices, A and B can also be multiplied in MATLAB as
- » A=[1,2,3;4,5,6];
  » B=[1,4;6,7;0,1];
  »
  » C=A*B.

- Multiplication of the matrices A and B.

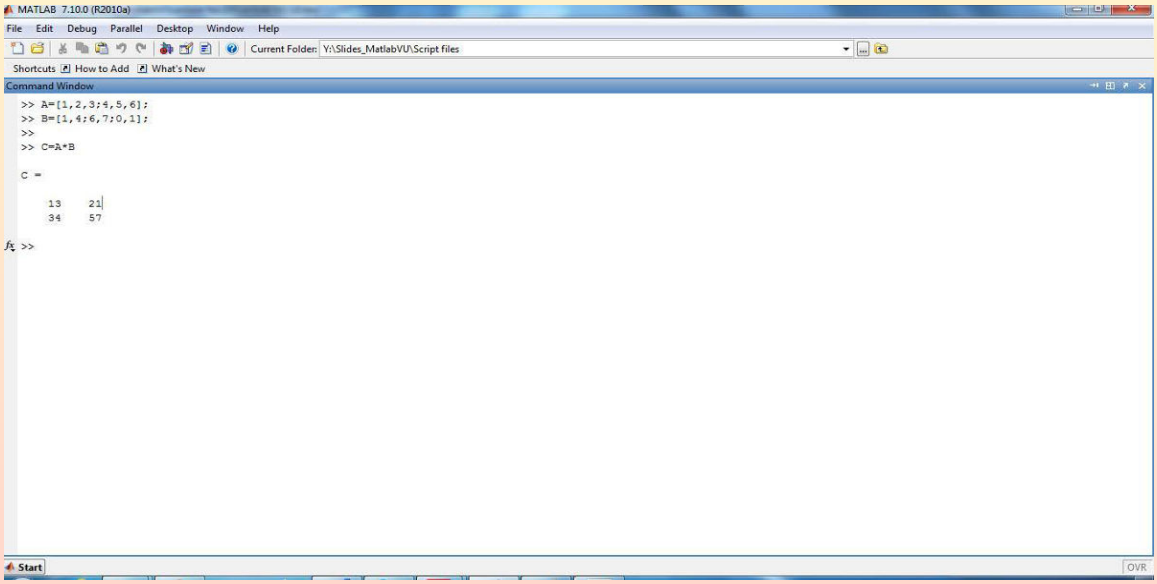


Figure: Matrices multiplication using asterisk operator

- The determinant of a square matrix in MATLAB can be determined using **det** command.
- For example, the determinant of the following matrix is determined as
- »A =
$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$
- » A=sym([5/6,1/6,0;1/6,0,5/6;0,5/6,1])
- » det(A)
- The desired MATLAB output is
  ans =
  -131/216

- The inverse of a square matrix in MATLAB is determined using the **inv** command.
- For example, the inverse of the matrix A is computed as
- » inv(A)
- The required output is as under:
- ans =
$$\begin{pmatrix} \frac{150}{131} & \frac{36}{131} & -\frac{30}{131} \\ \frac{36}{131} & -\frac{180}{131} & \frac{150}{131} \\ -\frac{30}{131} & \frac{150}{131} & \frac{6}{131} \end{pmatrix}$$

- Determinant and inverse of the matrix A.

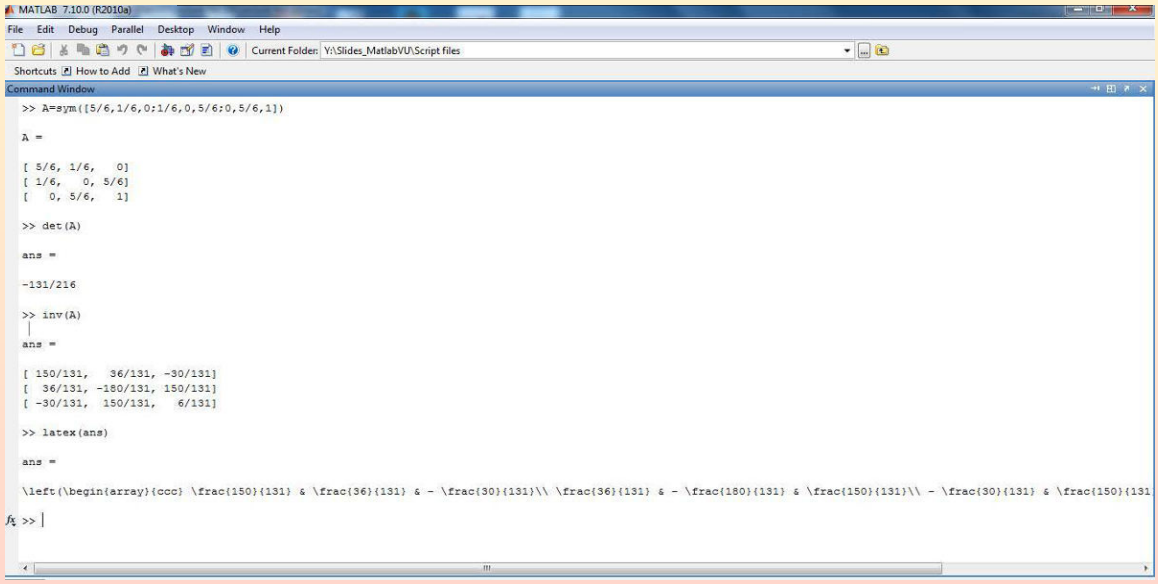


Figure: det and inv command in MATLAB

Compute AB using for loop in MATLAB, if

- A= $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{1}{6} & 0 & \frac{5}{6} \\ 0 & \frac{5}{6} & 1 \end{pmatrix}$
- B= $\begin{pmatrix} \frac{5}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & \frac{5}{3} \\ 0 & \frac{5}{6} & 1 \end{pmatrix}$

Compute the sum of the second column of the following matrix using **sum** command.

- C= $\begin{pmatrix} 1 & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{6} & 2 \\ 0 & 0 & 1 \end{pmatrix}$
- Compute the sum of the second row of the matrix C using **sum** command.

Matrix Algebra and MATLAB

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Outline

Matrices in MATLAB
Matrices
Matrix Algebra
Exercise

Outline

1. Matrices in MATLAB
   - Creating Vector and Matrix Variables
2. Matrices
   - Some useful operations on matrices
3. Matrix Algebra
   - Matrices Multiplication
   - Determinant of a matrix
   - Inverse of a matrix
4. Exercise

Outline

Matrices can be introduced into MATLAB in several ways:

- Entered by an explicit list of elements.
- Generated by so called built–in statements and functions.
- Loaded by external data.
- Created by M–Files.

Generating matrices as a list of its elements. The following basis conventions have to be taken into account:

- The elements of a row are separated by commas or spaces.
- A semi-colon indicates the end of each row.
- The entire list of elements has to be surrounded by brackets [ ].

Row Vector

- Row vector variables can be created in several ways.
- The simplest method is to put the values that you want in the variable in square brackets, separated by either spaces or commas.
- For instance, a row vector with four elements in MATLAB can be generated as:
- rowvec=[2,4,5,6]
- The required output is as under
- rowvec =
  2 4 5 6

Row vector using Colon Operator

- Row vector can also be generated using Colon operator(;)
- The entries in the generated Row vector iterates from the starting to the ending value with a default step of one.
- In this case the square brackets are not necessary.
- For example, a row vector having $2, 3, 4, 5, 6$ entries can be generated as
- » colvec=2:6
- The necessary output is
- colvec =
  2 3 4 5 6

Colon Operator with Step value

- Row vector with Colon operator can also be generated by defining step value for each entry of the matrix.
- For example, the row vector with entries $2, 4, 6, 8, 10$ can be generated as
- » stepvec = 2:2:10
- The desired output is
- stepvec =
  2 4 6 8 10

- Row vector is generated in different ways

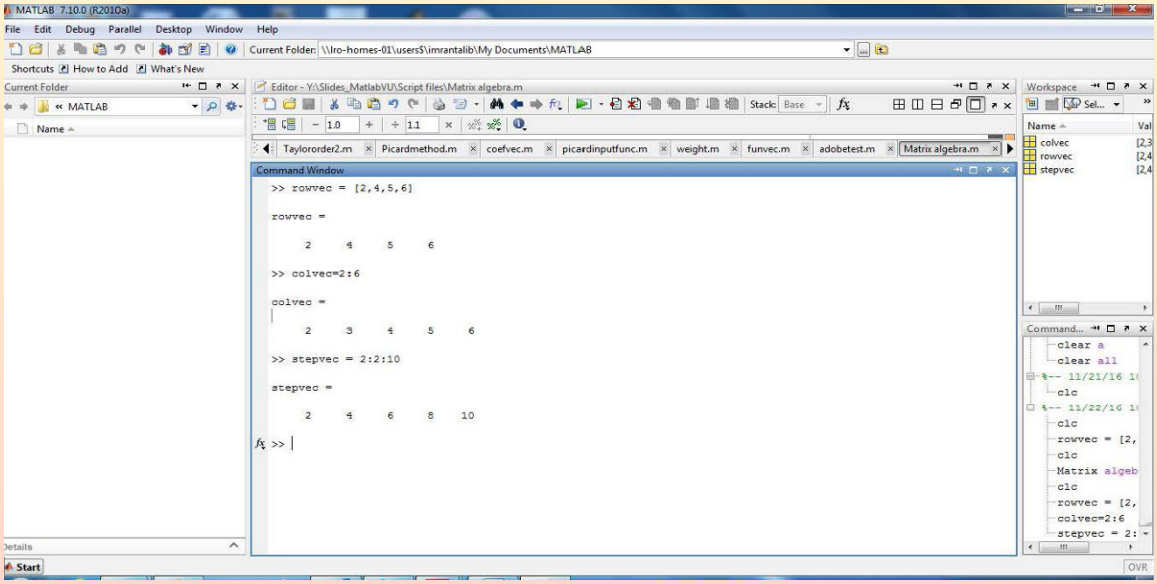


Figure: Display of the Row vector in different ways

Column Vector

- There are two basic methods for creating Column vector:
- Either by putting the values in square brackets, separated by semicolons.
- or by creating a row vector and then transposing it.
- The transpose operator in MATLAB is the apostrophe.

Column Vector with Square bracket and semicolons

- For example, the column vector with entries $2, 5, 10$ can be generated as
- » colvec=[2;5;10]
- The required output is
  colvec =
  2
  5
  10

Column Vector with apostrophe

- Alternatively, the column vector with entries $2, 5, 10$ can also be generated as
- » Rowvec=[2,5,10]
  Rowvec =
  2 5 10
  » colvec=Rowvec'
  colvec =
  2
  5
  10

- Column vector is generated in different ways

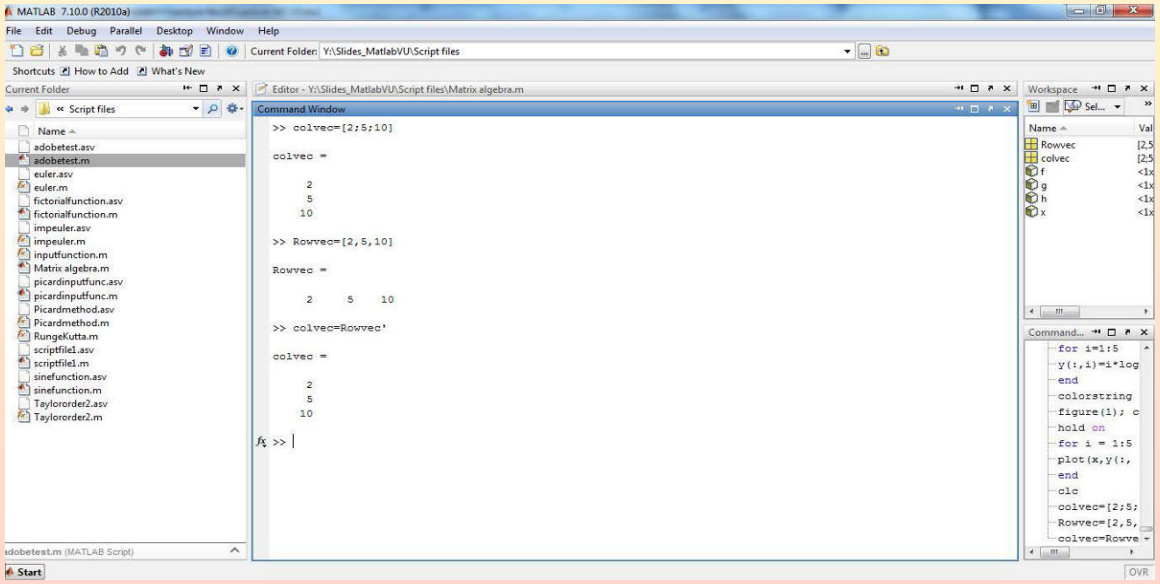


Figure: Display of the Column vector in different ways

- Matrix variables can be created by putting the values in square brackets.
- The values within the rows are separated by either spaces or commas.
- Each individual row is separated by semicolons.
- The must always be the same number of values in every row.
- The colon operator can be used to iterate within the rows.

- For example, the matrix

$$A = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{bmatrix}$$

in MATLAB is generated as

- » A=[5/6,1/6,0;5/6,0,1/6;0,5/6,1/6]
  A =
  0.8333 0.1667 0
  0.8333 0 0.1667
  0 0.8333 0.1667

or

- A=sym([5/6,1/6,0;5/6,0,1/6;0,5/6,1/6])
  A =

$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$

- Generation of the matrix in MATLAB

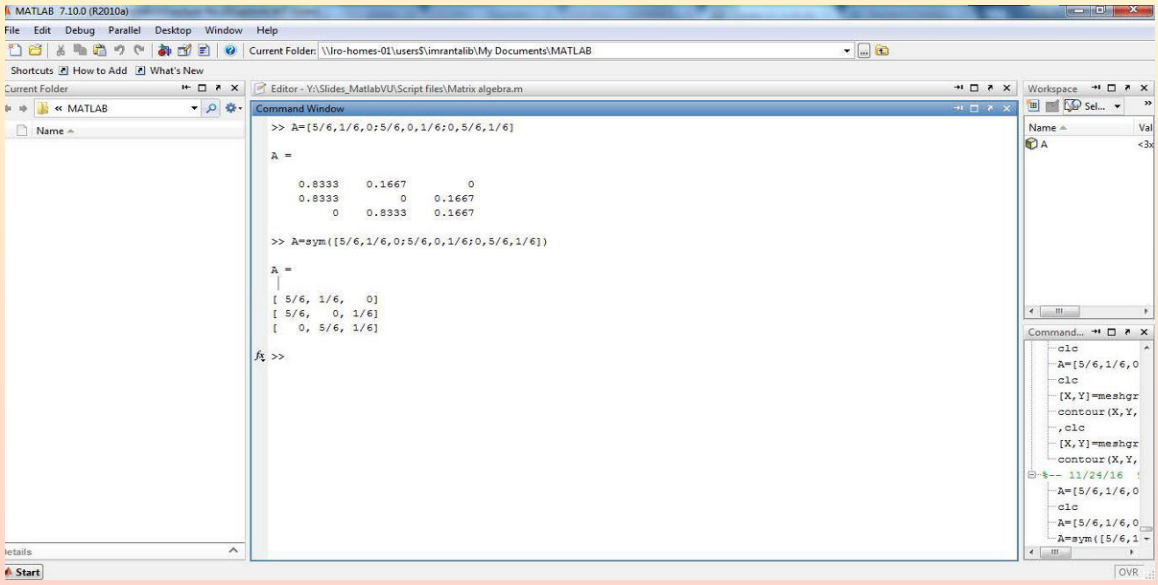


Figure: Graphical view of the matrices

Sum of column vectors of a matrix

- The **sum** command computes a **Row vector** containing the sum of the **columns** of a matrix.
- Note that MATLAB prefers to work with the columns of a matrix.
- For example, the sum of the columns of the matrix A is computed as:
-
- A=sym([5/6,1/6,0;5/6,0,1/6;0,5/6,1/6])

A =

$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$

» B = sum(A)

B =

$$\begin{pmatrix} \frac{5}{3} & 1 & \frac{1}{3} \end{pmatrix}$$

Sum of row vectors of a matrix

- To compute the sum of Row vectors of a matrix, we should take the transpose, then compute the sum and finally transpose the result.
- For example, the sum of the **Row** vectors of the matrix A is calculated in the next slide. Please see it.

- Sum of the entries of the column vectors of A

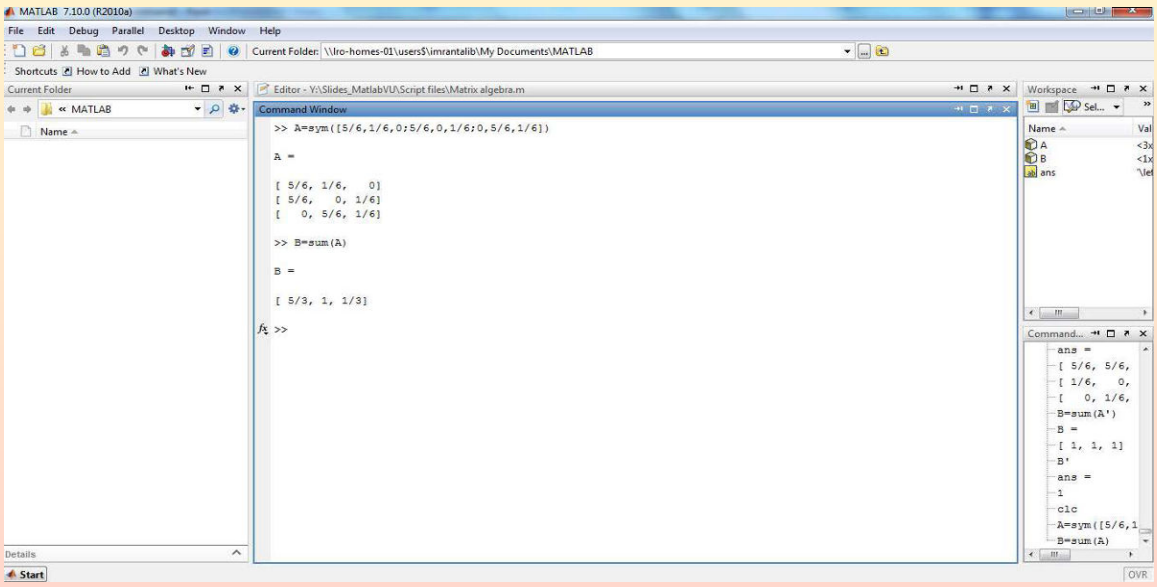


Figure: Display of the sum command

Sum of row vectors of a matrix

- »A = $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$

  » A'= $\begin{pmatrix} \frac{5}{6} & \frac{5}{6} & 0 \\ \frac{1}{6} & 0 & \frac{5}{6} \\ 0 & \frac{1}{6} & \frac{1}{6} \end{pmatrix}$

  » B=sum(A')

  B =

  [1, 1, 1]

  » B'

  ans =

  1

  1

  1.

- Sum of the entries of the row vectors of A

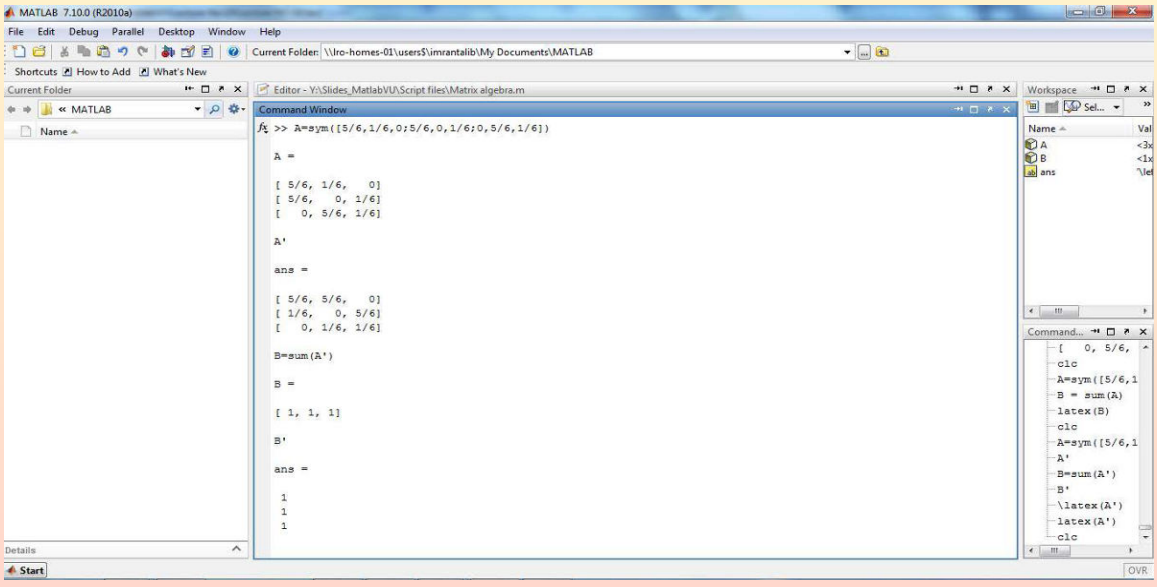


Figure: Display of the sum command

Column vector containing the Diagonal elements of a matrix

- The **diag** command produces a column vector containing the elements of the main diagonal of a matrix.
- For example,
- »A = $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$
- The MATLAB displays the following result:
- » B=diag(A)
  B =
  5/6
  0
  1/6

Sum of individual row and column

- The collective use of **sum** command and **colon (:)** operator is used to compute the sum of any individual column and row of a matrix.
- The colon operator refers to all elements in a row or column of a matrix.
- The key word **end** refers to last column or row.
- For example, the sum of last column of the matrix *A* in MATLAB is calculated as
- »A = $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$
- The MATLAB displays the following result:
- » B=sum(A(:,end))
  B =
  1/3

- Sum of the entries of the last and the second column of A

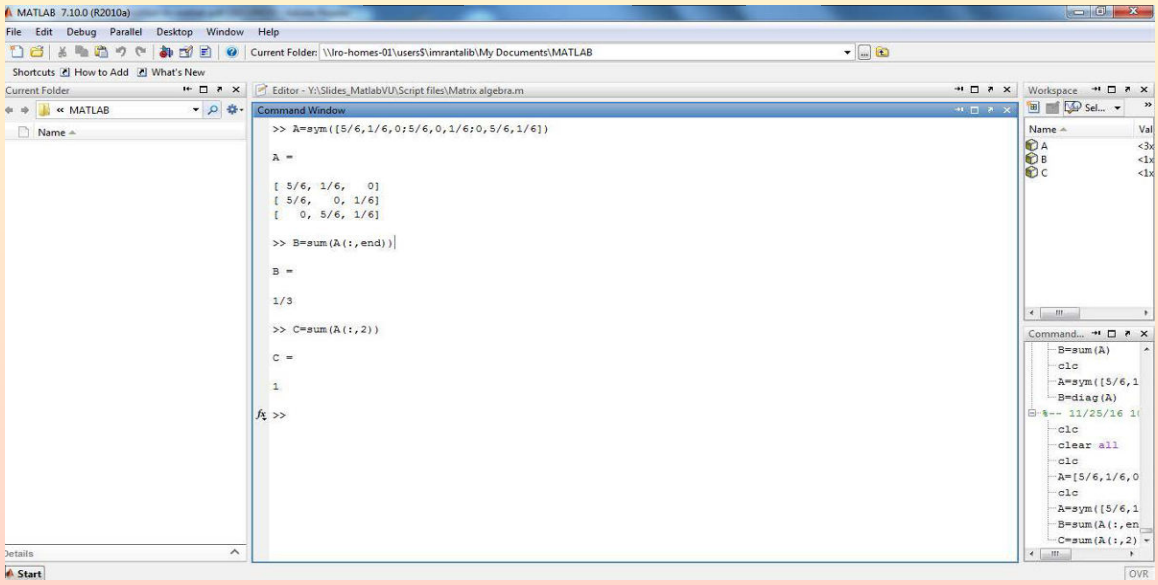


Figure: Display of the sum command with colon operator

Matrices multiplication using for loop

- The matrix multiplication can be defined using MATLAB's **for** loop, colon notation, and vector scalar product.
- For example, the MATLAB syntax for the multiplication of two matrices A and B using for loop is as under:
- clear all;
  close all;
  clc;
  A=[1,2,3;4,5,6];
  B=[1,4;6,7;0,1];
  for i=1:2
  for j=1:2
  C(i,j)=A(i,:)*B(:,j);
  end
  end
  C

- Multiplication of the matrices A and B

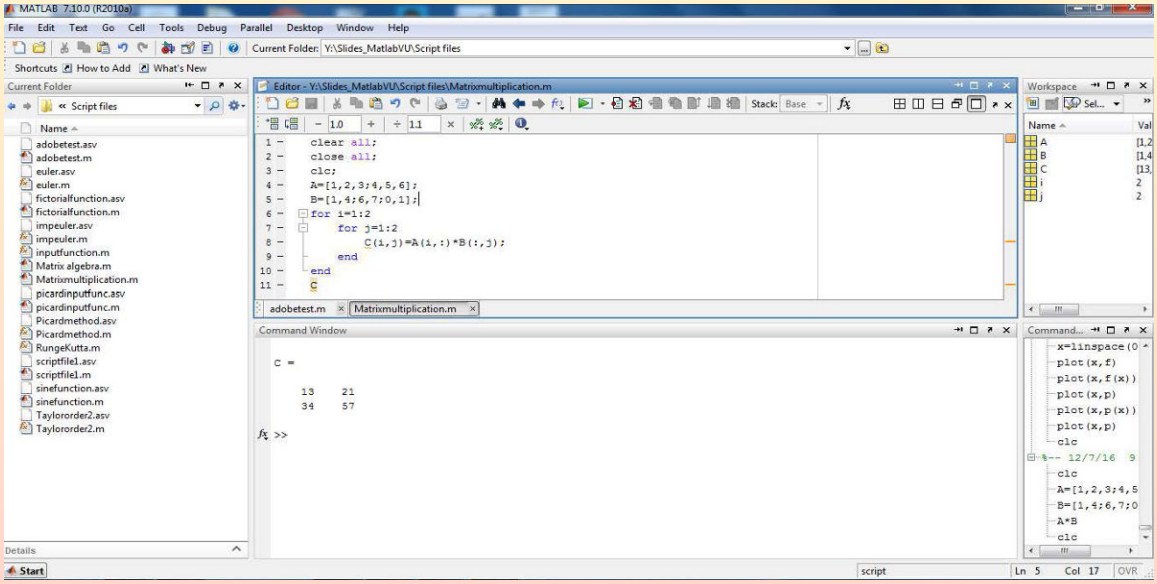


Figure: Matrices multiplication using for loop

Matrices multiplication using asterisk operator

- Single asterisk operator (*) can also be used for the matrices multiplication.
- For example, above defined matrices, A and B can also be multiplied in MATLAB as
- » A=[1,2,3;4,5,6];
  » B=[1,4;6,7;0,1];
  »
  » C=A*B.

- Multiplication of the matrices A and B.

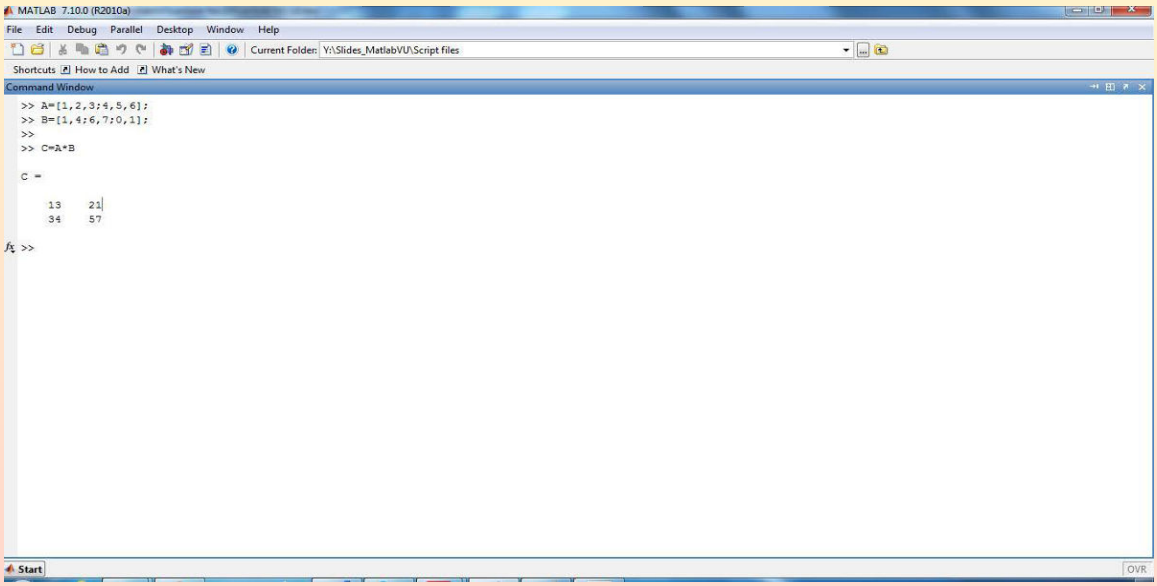


Figure: Matrices multiplication using asterisk operator

- The determinant of a square matrix in MATLAB can be determined using **det** command.
- For example, the determinant of the following matrix is determined as
- »A =
$$\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$$
- » A=sym([5/6,1/6,0;1/6,0,5/6;0,5/6,1])
- » det(A)
- The desired MATLAB output is
ans =
-131/216

- The inverse of a square matrix in MATLAB is determined using the **inv** command.
- For example, the inverse of the matrix A is computed as
- » inv(A)
- The required output is as under:
- ans =

$$\begin{pmatrix} \frac{150}{131} & \frac{36}{131} & -\frac{30}{131} \\ \frac{36}{131} & -\frac{180}{131} & \frac{150}{131} \\ -\frac{30}{131} & \frac{150}{131} & \frac{6}{131} \end{pmatrix}$$

- Determinant and inverse of the matrix A.

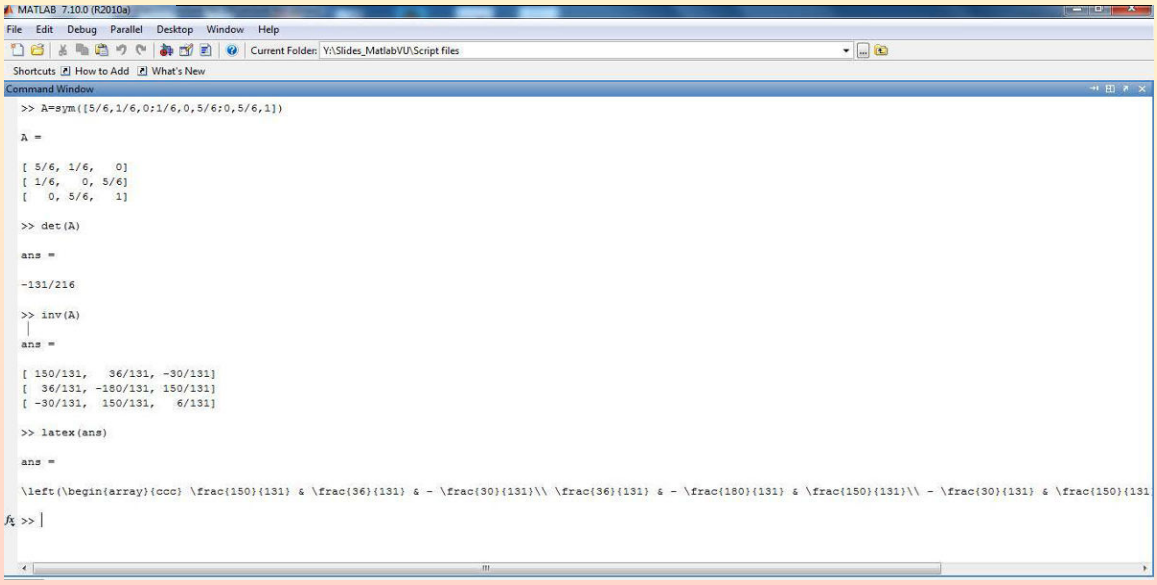


Figure: det and inv command in MATLAB

Compute AB using for loop in MATLAB, if

- A= $\begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{1}{6} & 0 & \frac{5}{6} \\ 0 & \frac{5}{6} & 1 \end{pmatrix}$
- B= $\begin{pmatrix} \frac{5}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & \frac{5}{3} \\ 0 & \frac{5}{6} & 1 \end{pmatrix}$

Compute the sum of the second column of the following matrix using **sum** command.

- C= $\begin{pmatrix} 1 & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{6} & 2 \\ 0 & 0 & 1 \end{pmatrix}$
- Compute the sum of the second row of the matrix C using **sum** command.

Linear Algebra and MATLAB

Department of Mathematics and Statistics
Virtual University of Pakistan, Lahore, Pakistan.

imrantalib@vu.edu.pk

Outline

Outline

Eigen Values and Eigen Vectors
Reduced row echelon form
Solution of system of linear equations
Elementary row operations
Spanning set
Exercise

Outline

1. Eigen Values and Eigen Vectors
2. Reduced row echelon form
3. Solution of system of linear equations
   - Solution of equations when coefficient matrix have inverse
4. Elementary row operations
5. Spanning set
6. Exercise

Imran Talib VU Lecture NO.11 2

Outline

Eigen Values and Eigen Vectors
Reduced row echelon form
Solution of system of linear equations
Elementary row operations
Spanning set
Exercise

Outline

1. Eigen Values and Eigen Vectors
2. Reduced row echelon form
3. Solution of system of linear equations
   - Solution of equations when coefficient matrix have inverse
4. Elementary row operations
5. Spanning set
6. Exercise

Outline

- The **eig** command is used to determine the Eigen value of a square matrix.
- This command with slightly modification also used to determine the associated Eigen vector.
- Consider the following example:

Example:

- Compute the eigen values and eigen vectors for the following matrix:

$$B = \begin{pmatrix} 0 & 1 & 3 \\ 2 & 4 & 1 \\ 6 & 1 & 8 \end{pmatrix}$$

- The recipe of the solution is as under:
- B=[0,1,3;2,4,1;6,1,8];
  eig(B)
  $[P, D] = eig(B)$

Note:

- The columns of the matrix P denotes the eigen vectors of the matrix B.
- The matrix D denotes the diagonal matrix having eigen values on the main diaginal.

- Computation of the eigen values and eigen vectors.

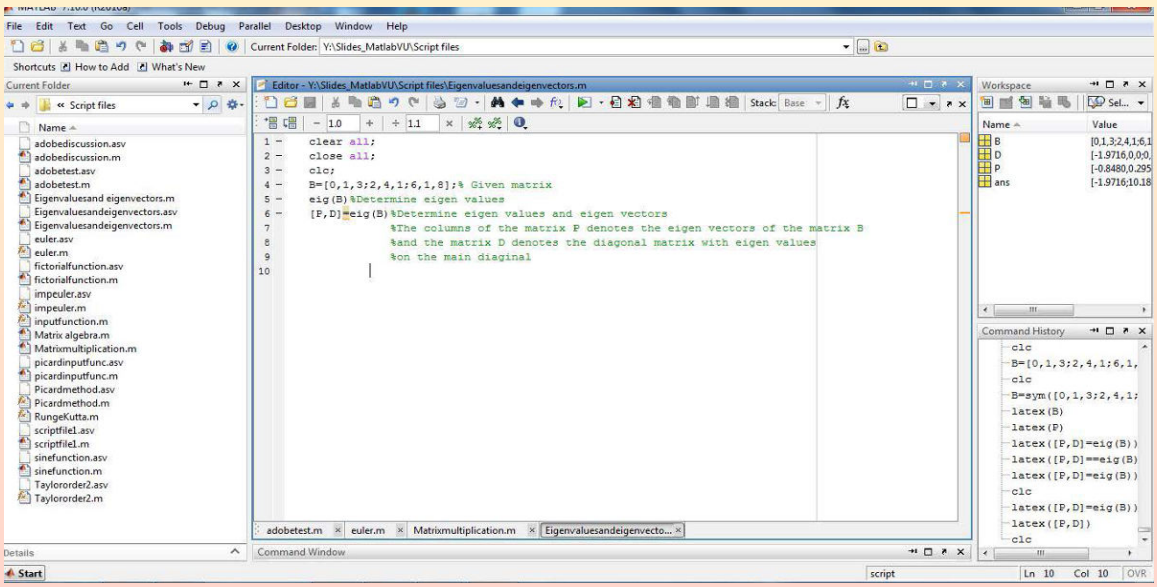


Figure: Editor file having code of eigen values and eigen vectors

- Results of the eigen values and eigen vectors.

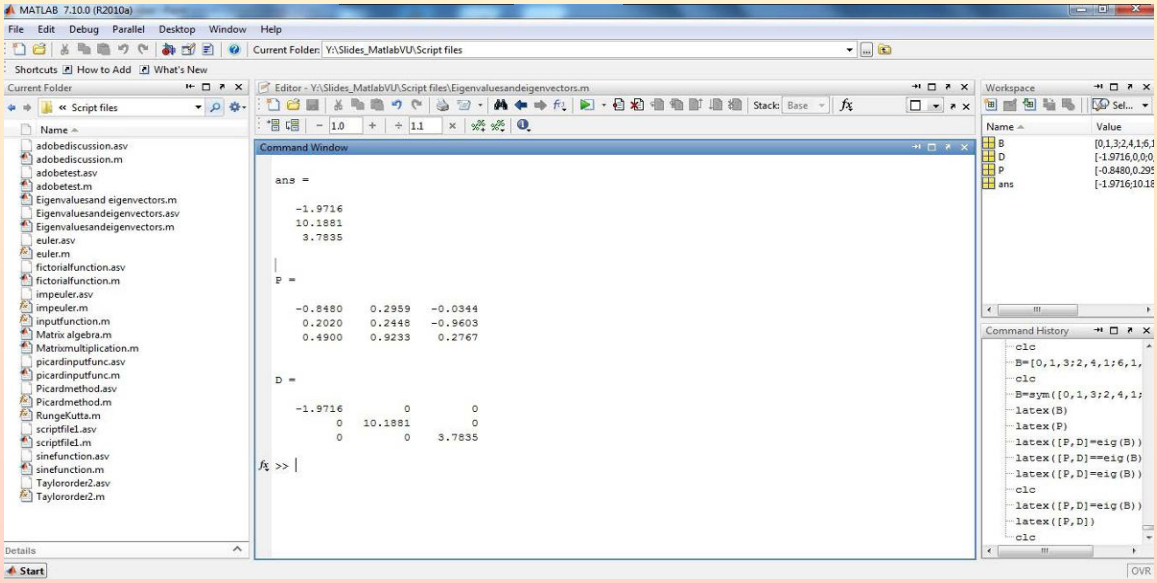


Figure: Command window having outputs of the eigen values and eigen vectors

- MATLAB also gives the reduced echelon form of a matrix.
- For this **rref** command is used.
- Consider the following example:

Example:

- Find the reduced echelon form of the following matrix:

$$B = \begin{pmatrix} 0 & 1 & 3 \\ 2 & 4 & 1 \\ 6 & 1 & 8 \end{pmatrix}$$

- Reduced echelon form.

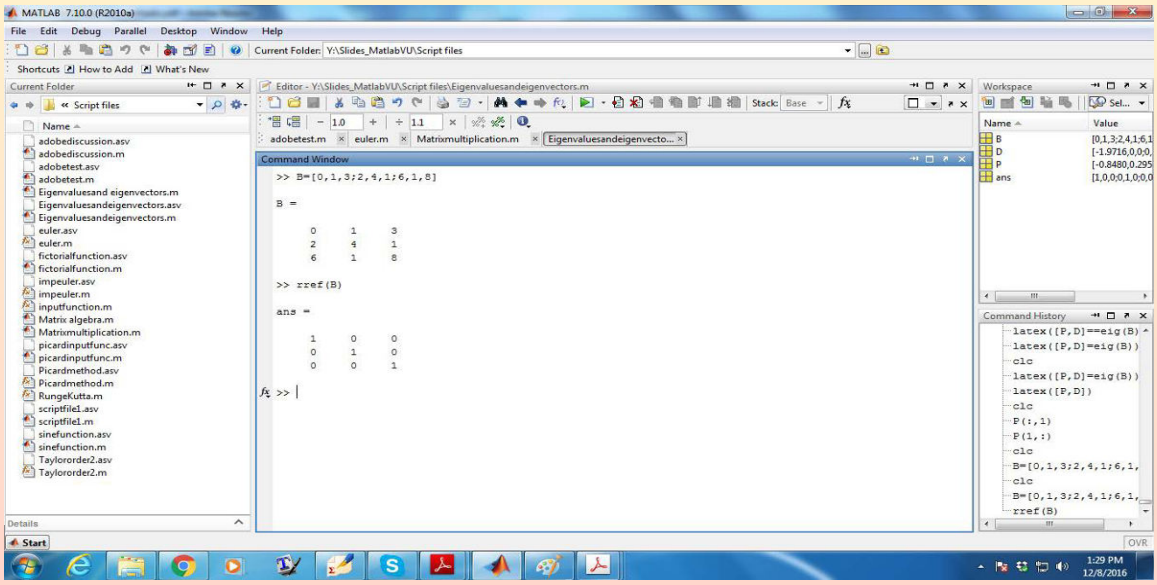


Figure: output of **rref** command

Solution of equations when coefficient matrix have inverse

- While dealing with system of linear equations two situation often occurs:
- Whether the coefficient matrix have inverse or have not?
- First we discuss the case when the coefficient matrix have inverse

Example:

- Find the solution of the following system of linear equations:

$$\begin{aligned} x_1 + 3x_2 &= -2 \\ 2x_1 + 4x_2 &= 1. \end{aligned} \tag{3.1}$$

Solution:

- The coefficient matrix is as under:
  $A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$
- The column matrix is given below:
  $B = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$

- The MATLAB code for the solution of the system is as under:
- A=sym([1,3;2,4]);
  B=sym([-2;1]);
  C=inv(A);
  X=C*B
- For more clarity see the next slide.

- Solution of linear equations in MATLAB when coefficient matrix have inverse.

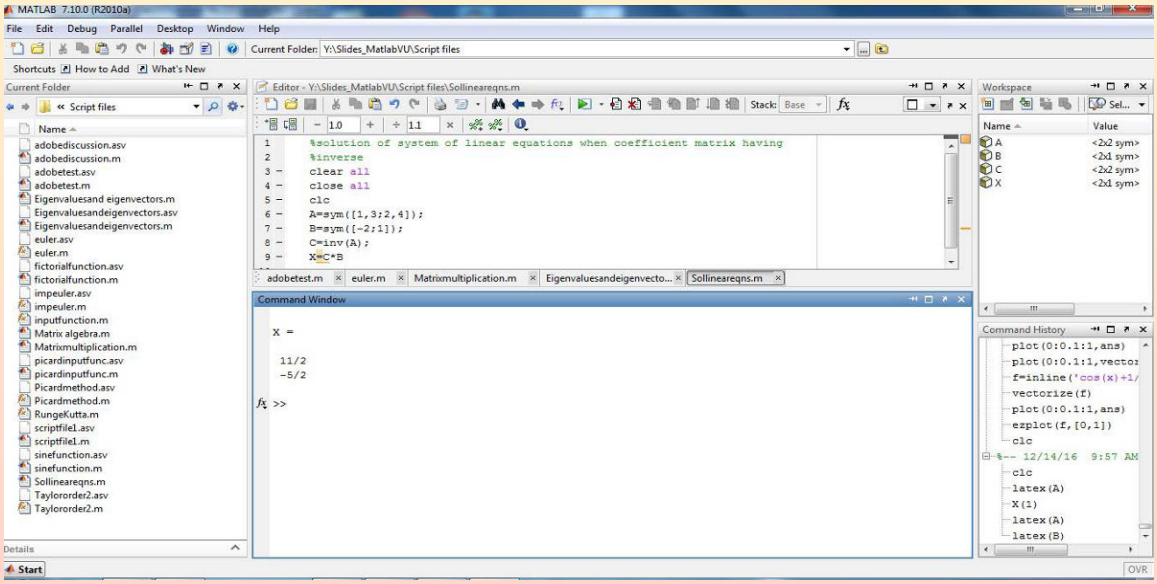


Figure: Display of the MATLAB Editor with code of the solution of linear equations

Example:

- Solve the following system of linear equations using **solve** command:

$$\begin{aligned} 3x + 2y + z &= 1, \\ y &= 2, \\ x + 2y &= 3. \end{aligned} \tag{3.2}$$

Solution:

- The MATLAB code for the solution is:
- eqn1='3*x+2*y+z=1';
  eqn2='y=2';
  eqn3='x+2*y=3';
  sol=solve(eqn1,eqn2,eqn3);
  X=double([sol.x,sol.y,sol.z])'

- Solution of linear equations

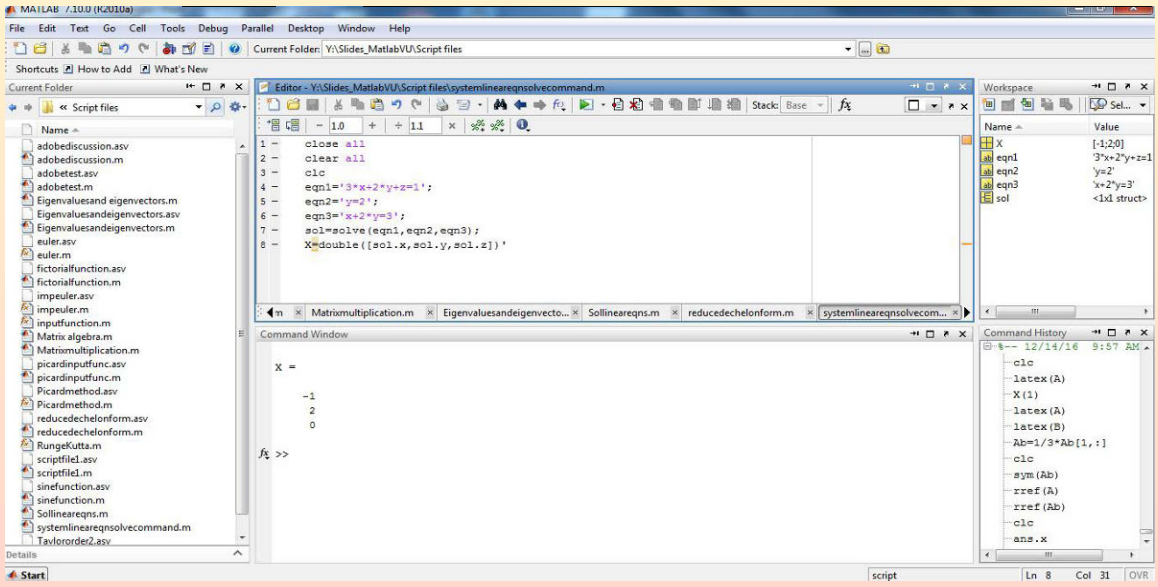


Figure: Display of the MATLAB Editor with code of the solution of linear equations using solve command

- The matrix can also be converted into reduced echelon form using row operations.
- Consider the following example:

Example:

- Consider the system of linear equations:

$$\begin{aligned} 3x + 2y + z &= 1, \\ y &= 2, \\ x + 2y &= 3. \end{aligned} \tag{4.1}$$

- Enter it into MATLAB as an augmented matrix named *Ab*.
- Use elementary row operations to reduce it to reduced-echelon form.

Solution:

- The MATLAB recipe for the solution is as under:
- A=[3,2,1;0,1,0;1,2,0];
  b=[1,2,3]';
  Ab=[A b];
  Ab(1,:)=1/3*Ab(1,:);
  Ab(3,:)=Ab(3,:)-Ab(1,:);
  Ab(1,:)=Ab(1,:)-0.6667*Ab(2,:);
  Ab(3,:)=Ab(3,:)-1.3333*Ab(2,:);
  Ab(3,:)=1/-0.3333*Ab(3,:);
  Ab(1,:)=Ab(1,:)-0.3333*Ab(3,:)

- Reduced echelon form using row operations

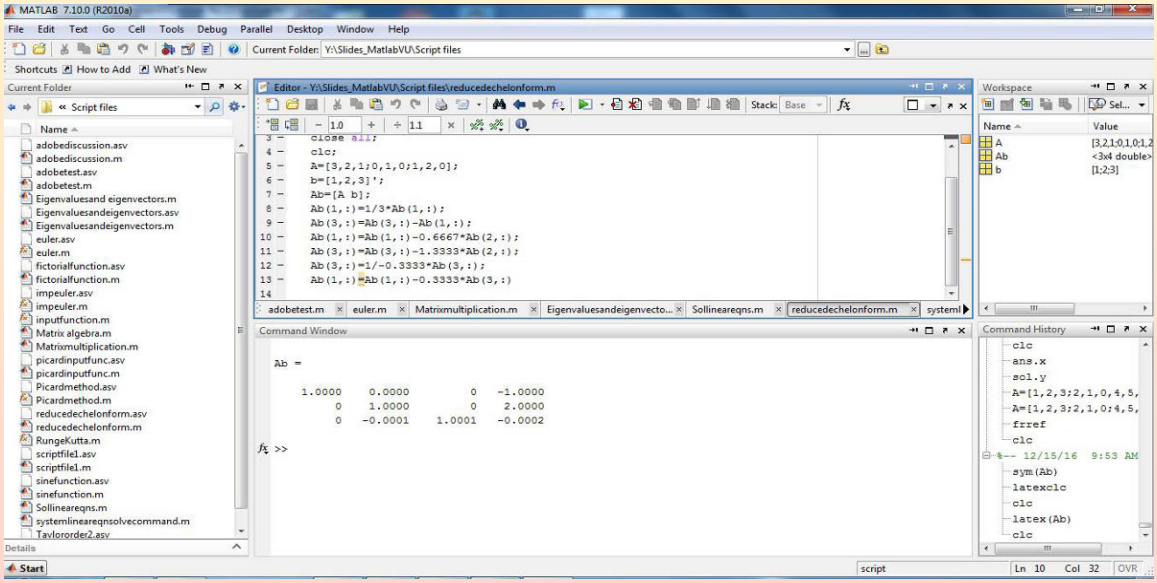


Figure: Reduction of the matrix into reduced echelon form

- **rref** command is also used to check that whether a vector can be written as a linear combination of the elements (vectors) of a set or not?
- Consider the following example:

Example:

- Show that $\begin{pmatrix} a \\ b \end{pmatrix}$ is in the span of $\begin{pmatrix} -2 & 1 \\ 1 & 6 \end{pmatrix}$ for any a and b.

Solution:

- First declare that a and b as symbolic.
- Enter the matrix Ab that can help you to show this.
- Use **rref** on the matrix Ab.
- Explicitly give the weights w_1 and w_2, such that:
$$\begin{pmatrix} a \\ b \end{pmatrix} = w_1 \begin{pmatrix} -2 \\ 1 \end{pmatrix} + w_2 \begin{pmatrix} 1 \\ 6 \end{pmatrix}$$

Solution:

- The MATLAB code for the solution is as under:
- clear all;
close all;
clc;
syms a b
A=[-2,1;1,6];
b=[a;b];
Ab=[A b];
rref(Ab)
- The possible values for the weights are $\begin{pmatrix} \frac{b}{13} - \frac{6\,a}{13} \\ \frac{a}{13} + \frac{2\,b}{13} \end{pmatrix}$

- Application of **rref** command

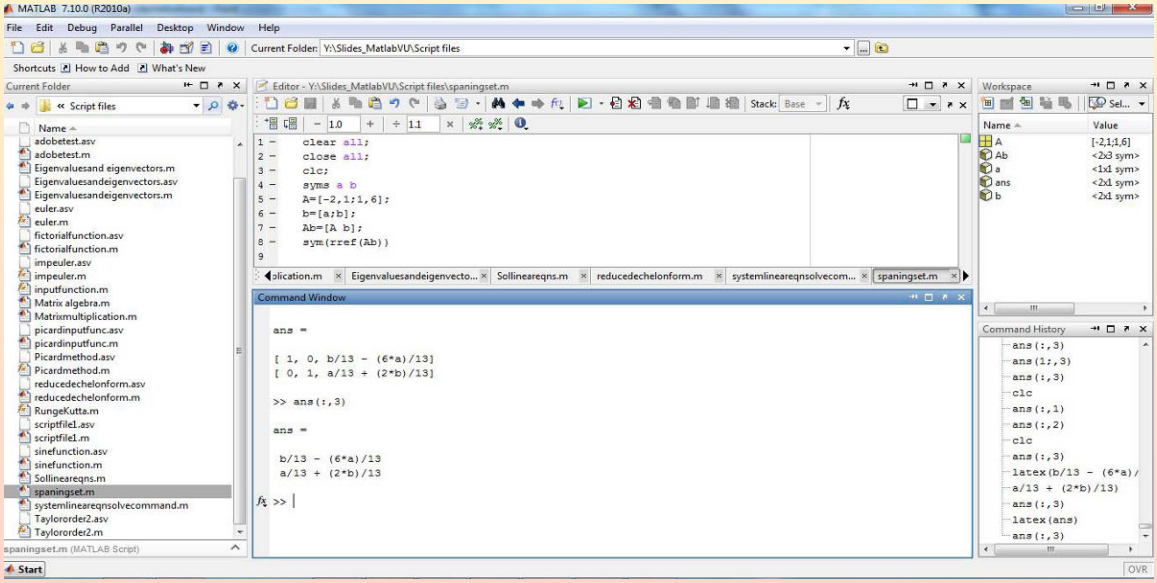


Figure: MATLAB code to determine the weights

- Consider the system of equations

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &= -3, \\ -4x_1 - 5x_2 + 2x_3 &= -2, \\ 2x_1 + 3x_2 - x_3 &= 2. \end{aligned} \tag{6.1}$$

- Enter it into MATLAB as an (augmented) matrix named *Ab*.
- Use elementary row operations to reduce it to reduced echelon form.
- Give the solution of the system. (If the system has no solution say that. Otherwise, write the basic variables (those corresponding to pivot columns) as functions of numbers and free variables, if there are any.)

- Consider the system of equations

$$\begin{aligned} x_1 + 3x_3 + x_5 &= -1, \\ 4x_1 + x_2 + x_3 + 6x_5 &= 1, \\ -3x_1 - 3x_2 - 3x_3 + x_4 - 19x_5 &= 6, \\ 10x_1 - 4x_2 + 38x_3 + 2x_4 - 12x_5 &= 0. \end{aligned} \tag{6.2}$$

- Enter it into MATLAB as an (augmented) matrix named Ab.
- Use elementary row operations to reduce it to reduced echelon form.
- Give the solution of the system. (If the system has no solution say that. Otherwise, write the basic variables (those corresponding to pivot columns) as functions of numbers and free variables, if there are any.)

- Show that $\begin{pmatrix} h \\ k \end{pmatrix}$ is in the span of $\begin{pmatrix} -3 & 2 \\ 4 & 5 \end{pmatrix}$ for any h and k.

LOOPS IN MATLAB

What is a Loop?

A loop is a block of code that repeats itself

A loop can be programmed to repeat a fixed number of times or it can be programmed to repeat until some event occurs

Loop Type Comparison

- Counter Based **For Loop**
  - Runs a predetermined number of times
  - Uses a counter variable
  - Automatically updates the loop variable
- Conditional Based **While Loop**
  - Runs until some condition is met
  - Uses any variable(s)
  - Loop variable(s) must be updated manually

Both are based on a condition, it is simply that the endpoint of the counter based loop is known ahead of time (unless you make a mistake!), whereas the conditional based loop is not

General Items About Loops

- Loop condition is only checked once during each iteration
- Conditional test is based on the ***loop control variable(s)***:
  - Initialized prior entering the loop
  - Updated within the loop
  - When the loop control variable(s) take on value(s) that cause the condition to fail, the loop will exit and continue on with the rest of the program

FOR loops

FOR loops are counter based loops

Sample Construction:

```
for control_variable = begin:step:end
     MATLAB statements
end
```

- The control variable will count from the beginning value to the end value based on the step size specified
- It will repeat the statements included inside the loop (between the `for` and `end`) each time it counts
- Once the control variable is outside of the specified range, the loop will stop repeating

For Loops

- For loops are counter based loops

```
for     k=1:1:10
            MATLAB statement
end
```

Why While Loops?

- Consider the following situation:
  - Someone has been stealing your cookies! To figure out who, you are writing a script to act as a security system for your room. You need to take a measurement from a motion sensor every 10 seconds and take a picture if the motion sensor detects someone near the cookie jar.
- Can you write this program using a For loop?
  - No, you don't know if/when someone will try to steal your cookies!

WHILE loops repeat until some condition is no longer met

Construction:

```
while  conditional statement
    MATLAB statements
end
```

- As long as the condition is true, the MATLAB statements in the while loop will continue to execute
- If the expression is initially false, the while loop will never execute
- If the expression becomes false, the loop will terminate

Example:

```
while  a >= 0
     MATLAB statements
end
```

- a has to be defined in the program before you reach the while loop, otherwise the program won't run
- If a is negative, the while loop will not execute at all
- If a is positive, the while loop will continue to execute as long as it stays positive

What happens if a is positive and none of the statements in the loop ever change a?

Infinite Loop!

Example

Following code will add up each element of a vector, and display the result after ending the for loop.

```
v = 1:10;
sum = 0;
for i = 1:length(v)
   sum = sum + v(i);
end
disp(sum)
```

Example

We want to add only odd numbers from the numbers in 1:10

```
sum = 0;
for i = 1:2:10
    sum = sum + i;
end
disp(sum)
```

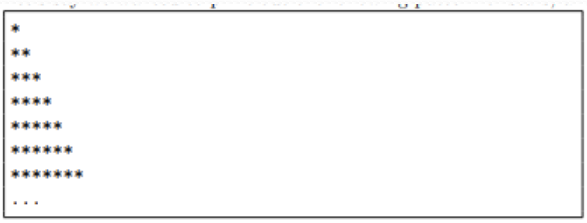
Example

Write a program to ask a number from a user, verify that the number is not negative, and compute its factorial.

```
numb=input('Enter a number: ');
fact=1;
if numb<0
fprintf('the number you have entered is negative');
else
for i=1:numb
fact=fact*i;
end
fact
end
```

Nested Loops

Let's say we wanted to print out the following pattern of stars, allowing the user to specify how many rows:



Example

```
rows=input(How many rows do you want: ');
for R=1:rows
        for s=1:R
                fprintf('*');
         end
     fprintf('\n');
end
```

Solving Differential Equations In MATLAB

Derivatives in MATLAB

```
syms x
f = sin(5*x);
```

The command

diff(f) differentiates f with respect to x:

ans = 5*cos(5*x)

To take the second derivative of g

g = exp(x)*cos(x);

diff(g,2)

Note that to take the derivative of a constant, you must first define the constant as a symbolic expression.
For example, entering

c = sym('5');
diff(c)

Derivatives of Expressions with Several Variables

To differentiate an expression that contains more than one symbolic variable, specify the variable that you want to differentiate with respect to. The diff command then calculates the partial derivative of the expression with respect to that variable. For example, given the symbolic expression

syms s t
*f = sin(s*t);*
the command
diff(f,t)
calculates the partial derivative $\partial f/\partial t$.

Integration in MATLAB

If f is a symbolic expression, then

int(f)

$$\int x^n dx = \begin{cases} \log(x) & \text{if } n = -1 \\ \dfrac{x^{n+1}}{n+1} & \text{otherwise.} \end{cases}$$

syms x

int(x^n) or int(x^n,x)

syms x
f = exp(-x^2);

int(f)
ans = (pi^(1/2)*erf(x))/2

exp(-x^2), there is no formula for the integral involving standard calculus expressions, such as trigonometric and exponential functions. In this case, MATLAB returns an answer in terms of the error function erf.

If MATLAB is unable to find an answer to the integral of a function f, it just returns int(f).

$$\int_0^{\pi/2} \sin(2x)dx = 1$$

syms x
*int(sin(2*x), 0, pi/2) or int(sin(2*x), x, 0, pi/2)*

First-Order Linear ODE

Solve this differential equation.

$dy/dt=ty$.

First, represent y by using syms to create the symbolic function y(t).

syms y(t)

Define the equation using == and represent differentiation using the diff function

*ode = diff(y,t) == t*y.*

dsolve

S = dsolve(eqn) solves the differential equation eqn, where eqn is a symbolic equation.

Use diff and == to represent differential equations. For example, diff(y,x) == y represents the equation dy/dx = y.

So we solve the equation using dsolve.

```
ySol(t) = dsolve(ode)
```

Example

Solve the first-order differential equation

dy/dt=ay

```
syms y(t) a
eqn = diff(y,t) == a*y;
S = dsolve(eqn)
```

Solve Differential Equation with Condition

In the previous solution, the constant *C* appears because no condition was specified.

Solve the equation with the initial condition *y(0) == 2*. The *dsolve* function finds a value of *C* that satisfies the condition.

```
cond = y(0) == 2;
ySol(t) = dsolve(ode,cond)
```

Example

$dy/dt+4y(t)=e^{-t}$,

$y(0)=1$.

```
syms y(t)
ode = diff(y)+4*y == exp(-t);
cond = y(0) == 1;
ySol(t) = dsolve(ode,cond)
```

Nonlinear Differential Equation with Initial Condition

Solve this nonlinear differential equation with an initial condition. The equation has multiple solutions.

$(dy/dt+y)^2=1$,
$y(0)=0$.

```
syms y(t)
ode = (diff(y,t)+y)^2 == 1;
cond = y(0) == 0;
ySol(t) = dsolve(ode,cond)
```

Second-Order ODE with Initial Conditions

- Solve this second-order differential equation with two initial conditions.

$d^2y/dx^2=cos(2x)-y$,
$y(0)=1$,
$y'(0)=0$.

```
syms y(x)
Dy = diff(y);
ode = diff(y,x,2) == cos(2*x)-y;
cond1 = y(0) == 1;
cond2 = Dy(0) == 0;
```

```
conds = [cond1 cond2];
ySol(x) = dsolve(ode,conds);
ySol = simplify(ySol)
```

Example

Solve the following differential Equation

$2x^2d^2y/dx^2+3xdy/dx-y=0.$

```
syms y(x)
ode = 2*x^2*diff(y,x,2)+3*x*diff(y,x)-y == 0;
ySol(x) = dsolve(ode)
```

Example

Solve the initial value problem $\frac{dy}{dt} = 1 + t^2 + y^2 + t^2 y^2, \quad y(0) = 1$

```
syms y(t)
eqn = diff(y,t)==1+t^2+y^2+t^2*y^2;
cond=y(0)==1;
dsolve(eqn,cond)
```

Third-Order ODE with Initial Conditions

Solve this third-order differential equation with three initial conditions.

$d^3u/dx^3 = u,$

$u(0) = 1,$

$u'(0) = -1,$

$u''(0) = \pi.$

```
syms u(x)
Du = diff(u,x);
D2u = diff(u,x,2);

ode = diff(u,x,3) == u;
cond1 = u(0) == 1;
cond2 = Du(0) == -1;
cond3 = D2u(0) == pi;
conds = [cond1 cond2 cond3];

uSol(x) = dsolve(ode,conds)
```

Solve a System of Differential Equations

Solve this system of linear first-order differential equations.

$du/dt = 3u + 4v$,

$dv/dt = -4u + 3v$.

Example

First, represent u and v by using syms to create the symbolic functions u(t) and v(t).

syms u(t) v(t)

Define the equations using == and represent differentiation using the diff function.

```
ode1 = diff(u) == 3*u + 4*v;
ode2 = diff(v) == -4*u + 3*v;
odes = [ode1; ode2]
S = dsolve(odes)
```

To access u(t) and v(t), index into the structure S.

uSol(t) = S.u

vSol(t) = S.v

Alternatively, store u(t) and v(t) directly by providing multiple output arguments.

```
[uSol(t), vSol(t)] = dsolve(odes)
```

The constants C1 and C2 appear because no conditions are specified. Solve the system with the initial conditions u(0) == 0 and v(0) == 0. The dsolve function finds values for the constants that satisfy these conditions.

```
cond1 = u(0) == 0;
cond2 = v(0) == 1;
conds = [cond1; cond2];
[uSol(t), vSol(t)] = dsolve(odes,conds)
```

Visualize the solution using fplot.

```
fplot(uSol)
hold on
fplot(vSol)
grid on
legend('uSol','vSol','Location','best')
```

Find Explicit and Implicit Solutions of Differential Equation

Solve the differential equation

$\partial/\partial t\, y(t)=e^{-y(t)}+y(t)$.

dsolve returns an explicit solution in terms of a Lambert W function that has a constant value.

```
syms y(t)
eqn = diff(y) == y+exp(-y)
sol = dsolve(eqn)
```

To return implicit solutions of the differential equation, set the 'Implicit' option to true.

```
sol = dsolve(eqn,'Implicit',true)
```

Find Implicit Solution When No Explicit Solution Is Found

If dsolve cannot find an explicit solution of a differential equation analytically, then it returns an empty symbolic array. You can solve the differential equation by using MATLAB numerical solver, such as ode45.

```
syms y(x)
eqn = diff(y) == (x-exp(-x))/(y(x)+exp(y(x)));
S = dsolve(eqn)
```

Warning: Unable to find symbolic solution.
S =[empty sym]

Alternatively, you can try finding an implicit solution of the differential equation by specifying the 'Implicit' option to true.

```
S = dsolve(eqn,'Implicit',true)
```

Numerical Solutions of ODEs in MATLAB Using ODE45

Sometimes it is not possible to find the exact solution of certain differential equations, then in this case we are forced to compute their numerical solutions.

To find the numerical solutions of the first–order ordinary differential equations, we use ode45 command.

[t,y] = ode45(odefun,tspan,yo)

where tspan = [to tf], integrates the system of differential equations $y'=f(t,y)$ from t_o to t_f with initial conditions yo. Each row in the solution array y corresponds to a value returned in column vector t.

Example

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.

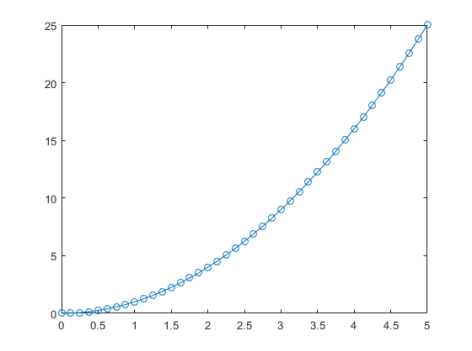
Solve the ODE

$y'=2t$.

Use a time interval of [0,5] and the initial condition yo = 0.

```
tspan = [0 5];
y0 = 0;
[t,y] = ode45(@(t,y) 2*t, tspan, y0);
```

Plot the solution.



Example

Plot the family of the approximate solutions of the following IVPs

$y'(x) = y\text{-}x^2 + 1,\ y(0) = 0{:}5 : 0{:}2 : 3$

```
f= @(x; y)y – x^2 + 1
```

```
ode45(f,[0,2],0.5:0.2:3)
```

Solve ODE with Multiple Initial Conditions

$y'(x) = -2y + 2\cos(t)\sin(2t)$

Create a vector of different initial conditions in the range [−5, 5].

Solve the equation for each initial condition over the time interval [0,3] using ode45.

```
yprime = @(t,y) -2*y + 2*cos(t).*sin(2*t);
tspan = [0 3];
y0 = -5:5;
[t,y] = ode45(yprime,tspan,y0);
```

Plot the results.

```
plot(t,y)
```

Solution of Differential Equations In MATLAB-II

Third-Order ODE with Initial Conditions

Solve this third-order differential equation with three initial conditions.

$d^3u/dx^3 = u,$

$u(0) = 1,$

$u'(0) = -1,$

$u''(0) = \pi.$

```
syms u(x)
Du = diff(u,x);
D2u = diff(u,x,2);

ode = diff(u,x,3) == u;
cond1 = u(0) == 1;
cond2 = Du(0) == -1;
cond3 = D2u(0) == pi;
conds = [cond1 cond2 cond3];

uSol(x) = dsolve(ode,conds)
```

Solve a System of Differential Equations

Solve this system of linear first-order differential equations.

$du/dt = 3u + 4v,$

$dv/dt = -4u + 3v.$

Example

First, represent u and v by using syms to create the symbolic functions u(t) and v(t).

syms u(t) v(t)

Define the equations using == and represent differentiation using the diff function.

```
ode1 = diff(u) == 3*u + 4*v;
ode2 = diff(v) == -4*u + 3*v;
odes = [ode1; ode2]
S = dsolve(odes)
```

To access u(t) and v(t), index into the structure S.

uSol(t) = S.u
vSol(t) = S.v

Alternatively, store u(t) and v(t) directly by providing multiple output arguments.

```
[uSol(t), vSol(t)] = dsolve(odes)
```

The constants C1 and C2 appear because no conditions are specified. Solve the system with the initial conditions u(0) == 0 and v(0) == 0. The dsolve function finds values for the constants that satisfy these conditions.

```
cond1 = u(0) == 0;
cond2 = v(0) == 1;
conds = [cond1; cond2];
[uSol(t), vSol(t)] = dsolve(odes,conds)
```

Visualize the solution using fplot.

```
fplot(uSol)
hold on
fplot(vSol)
grid on
legend('uSol','vSol','Location','best')
```

Explicit and Implicit Solutions

- An **explicit solution** is any solution that is given in the form y=y(t).

  In other words, the only place that y actually shows up is once on the left side and only raised to the first power.

- An **implicit solution** is any solution that isn't in explicit form.

Example

Let's use the example initial-value problem

$$y'y = -x, \qquad y(0) = r, \qquad r \text{ constant}$$

One can derive both an implicit and explicit solution for this DE. The *implicit* solution to this DE is

$$x^2 + y(x)^2 = r^2$$

This solution implicitly defines y(x); all we have here is an equation involving y(x). On the other hand, the explicit solution looks like

$$y(x) = \pm\sqrt{r^2 - x^2}$$

Find Explicit and Implicit Solutions of Differential Equation

Solve the differential equation

$\partial/\partial t\, y(t) = e^{-y(t)} + y(t)$.

dsolve returns an explicit solution in terms of a Lambert W function that has a constant value.

```
syms y(t)
eqn = diff(y) == y+exp(-y)
sol = dsolve(eqn)
```

To return implicit solutions of the differential equation, set the 'Implicit' option to true.

sol = dsolve(eqn,'Implicit',true)

Find Implicit Solution When No Explicit Solution Is Found

If dsolve cannot find an explicit solution of a differential equation analytically, then it returns an empty symbolic array. You can solve the differential equation by using MATLAB numerical solver, such as ode45.

```
syms y(x)
eqn = diff(y) == (x-exp(-x))/(y(x)+exp(y(x)));
S = dsolve(eqn)
```

Warning: Unable to find symbolic solution.
S =[empty sym]

Alternatively, you can try finding an implicit solution of the differential equation by specifying the 'Implicit' option to true.

S = dsolve(eqn,'Implicit',true)

Numerical Solutions of ODEs in MATLAB Using ODE45

Sometimes it is not possible to find the exact solution of certain differential equations, then in this case we are forced to compute their numerical solutions.

To find the numerical solutions of the first–order ordinary differential equations, we use ode45 command.

[t,y] = ode45(odefun,tspan,yo)

where tspan = [to tf], integrates the system of differential equations $y'=f(t,y)$ from t_o to t_f with initial conditions yo. Each row in the solution array y corresponds to a value returned in column vector t.

Example

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.

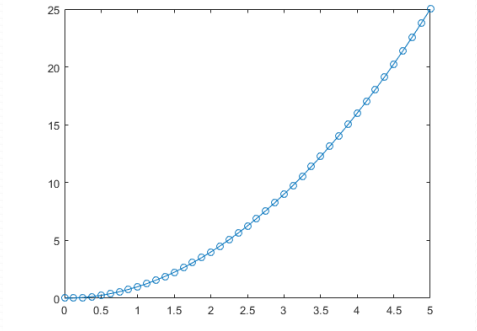
Solve the ODE

$y' = 2t$.

Use a time interval of [0,5] and the initial condition y0 = 0.

```
tspan = [0 5];
y0 = 0;
[t,y] = ode45(@(t,y) 2*t, tspan, y0);
```

Plot the solution.



Example

Plot the family of the approximate solutions of the following IVPs

$y'(x) = y - x^2 + 1,\ y(0) = 0.5 : 0.2 : 3$

```
f= @(x; y)y – x^2 + 1
```

```
ode45(f,[0,2],0.5:0.2:3)
```

Solve ODE with Multiple Initial Conditions

$y'(t) = -2y + 2\cos(t)\sin(2t)$

Create a vector of different initial conditions in the range [−5, 5].

Solve the equation for each initial condition over the time interval [0,3] using ode45.

```
yprime = @(t,y) -2*y + 2*cos(t).*sin(2*t);
tspan = [0 3];
y0 = -5:5;
[t,y] = ode45(yprime,tspan,y0);
```

Plot the results.

```
plot(t,y)
```

Numerical Solution for Differential Equation in MATLAB

Euler's Method

Euler's Method assumes our solution is written in the form of a Taylor's Series. That is, we'll have a function of the form:

$$y(x+h) \approx y(x) + hy'(x) + \frac{h^2 y''(x)}{2!} + \frac{h^3 y'''(x)}{3!} + \frac{h^4 y^{\mathrm{iv}}(x)}{4!} + \dots$$

For **Euler's Method**, we just take the first 2 terms only.

or simply $$y(x+h) \approx y(x) + hy'(x)$$

$$y(x+h) \approx y(x) + hf(x,y)$$

Continue..

The result of using this formula is the value for y, one h step to the right of the current value. Let's call it y_1. So we have:

$$y_1 \approx y_0 + hf(x_0, y_0)$$

where

y_1 is the next estimated solution value;

y_0 is the current value;

h is the interval between steps; and

$f(x_0, y_0)$ is the value of the derivative at the starting point, (x_0, y_0).

Continue...

$$y_2 \approx y_1 + hf(x_1, y_1)$$

where

y_2 is the next estimated solution value;

y_1 is the current value;

h is the interval between steps;

$x_1 = x_0 + h$; and

$f(x_1, y_1)$ is the value of the derivative at the current (x_1, y_1) point.

We continue this process for as many steps as required.

Euler Method on MATLAB

```
f=@(x,y)(x/y)
a=0;
b=2;
ya=1;
n=10;

h=(b-a)/n;      % h is the step
y=zeros(+1,1); % memory allocation, zero column vector
y(1)=ya;
    for j=1:n
    y(j+1)=y(j)+h*f(x(j),y(j)); % Euler's formula
    x(i+1)=x(i)+h;
    end
plot(x,y,'-b')
hold on
syms y(x)
ode= diff(y,x)== x/y ;
cond=y(0)==1;
S=dsolve(ode,cond)
F=matlabFunction(S)
fplot(F,[0,2],'r',"LineWidth",2);
hold off
```

Numerical Integration in MATLAB

Numerical Integration

Sometimes, the evaluation of expressions involving the integrals can become daunting, if not indeterminate. For this reason, a wide variety of numerical methods has been developed to simplify the integral.

Trapezoidal Rule

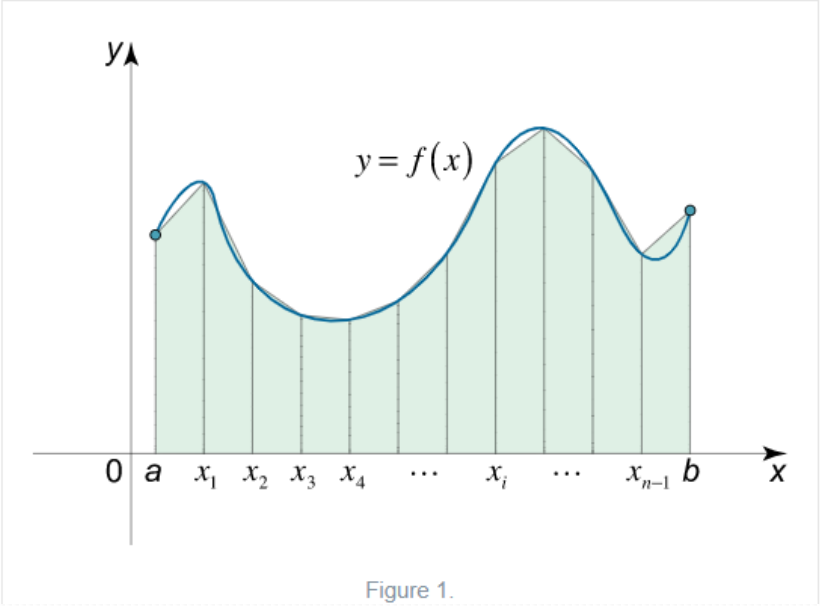


Figure 1.

The Trapezoidal Rule for approximating $\int_a^b f(x)\,dx$ is given by

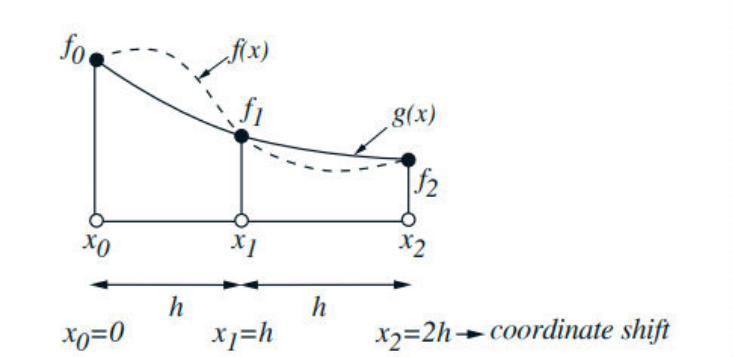
$$\int_a^b f(x)\,dx \approx T_n = \frac{\Delta x}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\right],$$

where $\Delta x = \frac{b-a}{n}$ and $x_i = a + i\Delta x$.

Code

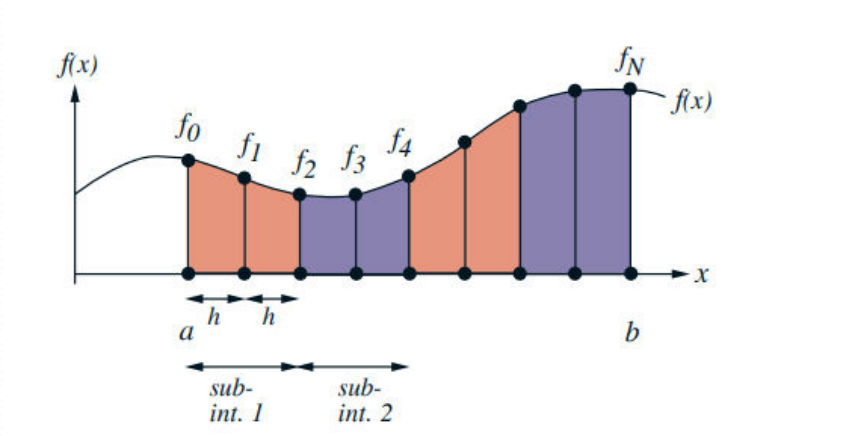
```
clc;
clear all;
f=@(x)1/(1+x); %Change here for different function
 a=input('Enter lower limit a: '); % exmple a=1
 b=input('Enter upper limit b: ');  % exmple b=2
 n=input('Enter the no. of subinterval: ');  % exmple n=10
 h=(b-a)/n;
 sum=0;
for k=1:1:n-1
  x(k)=a+k*h;
  y(k)=f(x(k));
  sum=sum+y(k);
end
% Formula:  (h/2)*[(y0+yn)+2*(y2+y3+..+yn-1)]
answer=h/2*(f(a)+f(b)+2*sum);
fprintf('\n The value of integration is %f',answer);  % exmple The value of integration is
0.410451
```

Simpson's 1/3 Rule



$$I = \frac{h}{3}[f_0 + 4f_1 + f_2]$$

Continue…



Continue…

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

$$\Rightarrow$$

$$I = \frac{h}{3}((f_o + 4f_1 + f_2) + (f_2 + 4f_3 + f_4) + (f_4 + 4f_5 + f_6)$$

$$+ \ldots + (f_{N-4} + 4f_{N-3} + f_{N-2}) + (f_{N-2} + 4f_{N-1} + f_N)) + E_{[a,b]}$$

$$\Rightarrow$$

$$I = \frac{h}{3}[f_o + 4f_1 + 2f_2 + 4f_3 + 2f_4 + 4f_5 + \ldots + 2f_{N-2} + 4f_{N-1} + f_N]$$

Continue

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\sum_{j=1}^{n/2}\left[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})\right]$$

$$= \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right],$$

```
clc;
clear all;
f=@(x)x^3; %Change here for different function
a=input('Enter lower limit a: '); % exmple a=0
b=input('Enter upper limit b: '); % exmple b=3
n=input('Enter the number of sub-intervals n: '); % exmple n=16
h=(b-a)/n;
if rem(n,2)==1
fprintf('\n Enter valid n!!!');
n=input('\n Enter n as even number ');
End
so=0;se=0;

for k=1:1:n-1
x(k)=a+k*h;
for k=1:1:n-1
if rem(k,2)==1
so=so+f(x(k));%sum of odd terms
else
se=se+f(x(k)); %sum of even terms
end
end
answer=h/3*(f(a)+f(b)+4*so+2*se);
fprintf('\n The value of integration is %f',answer);
```

Numerical Integration in MATLAB

Simpson's 3/8 Rule

$$\int_a^b f(x)\,dx \approx \frac{3h}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right]$$

$$= \frac{3h}{8}\left[f(x_0) + 3\sum_{i\neq 3k}^{n-1} f(x_i) + 2\sum_{j=1}^{n/3-1} f(x_{3j}) + f(x_n)\right] \quad \text{for } k \in \mathbb{N}_0.$$

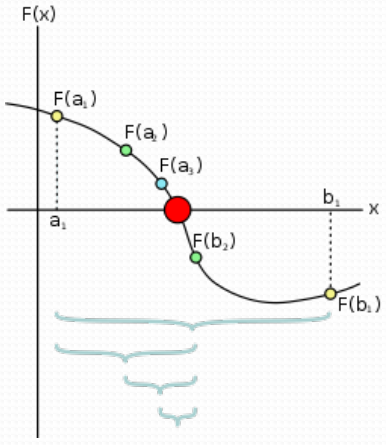
Continue…

```
clc;
clear all;
f=@(x)1/(1+x); %Change here for different function
 a=input('Enter lower limit a: '); % exmple a=1
 b=input('Enter upper limit b: ');  % exmple b=2
 n=input('Enter the  number of sub-intervals n: '); % exmple n=21
 h=(b-a)/n;
if rem(n,3)~=0
   fprintf('\n Enter valid n!!!');
   n=input('\n Enter n as multiple of 3: ');
end
for k=1:1:n
  x(k)=a+k*h;
  y(k)=f(x(k));
end
 s0=0;sm3=0;
 for k=2:1:n-1
    if rem(k,3)==0
       sm3=sm3+y(k); %sum of multiple of 3 terms
     else
     s0=s0+y(k);%sum of others terms
    end
end
answer=(3*h/8)*(f(a)+f(b)+3*s0+2*sm3);
fprintf('\n The value of integration is %f',answer); % exmple The value of integration is 0.381665
```

Bisection Method in MATLAB

Bisection Method

The method is applicable for numerically solving the equation f(x) = 0 for the real variable x, where f is a continuous function defined on an interval [a, b] and where f(a) and f(b) have opposite signs.



Continue…

At each step the method divides the interval in two by computing the midpoint c = (a+b) / 2 of the interval and the value of the function f(c) at that point.

Unless c is itself a root, there are now only two possibilities: either f(a) and f(c) have opposite signs and bracket a root, or f(c) and f(b) have opposite signs and bracket a root. The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step.

Iterative Scheme

The input for the method is a continuous function f, an interval [a, b], and the function values f(a) and f(b). The function values are of opposite sign. Each iteration performs these steps:

- Calculate c, the midpoint of the interval,

c = a + b/2.

- Calculate the function value at the midpoint, f(c).
- If convergence is satisfactory (that is, c - a is sufficiently small, or |f(c)| is sufficiently small), return c and stop iterating.
- Examine the sign of f(c) and replace either (a, f(a)) or (b, f(b)) with (c, f(c)) so that there is a zero crossing within the new interval.

Example

Suppose that the bisection method is used to find the root of the polynomial.

$$f(x) = x^3 - x - 2$$

First, two numbers a and b have to be found such that f(a) and f(b) have opposite signs. For the above function, a=1 and b=2 satisfy this criterion, as

f(1)=-2
f(2)=4

Because the function is continuous, there must be a root within the interval [1, 2].

In the first iteration, the end points of the interval which brackets the root are a1=1 and b1=2, so the midpoint is

c= (a+b)/2 = 1.5

f(c)=-0.125

Because f(c) is negative, a=1 is replaced with a=1.5

Code

```
f=@(x)x^3-x-2;
a=1;
b=2;
 if f(a)*f(b) > 0
    disp('There is no change of sign');
    return
end
 tol=10^-5;
disp('Iter.   a          b        c');
i=1;
while(abs(a-b)>=tol)
c=(a+b)/2;
if(f(c)==0)
       fprintf('Root of function is %f \n',c)
            return
    end
fprintf('%2i \t %f \t %f \t %f \n',i, a, b, c)
    if(f(a)*f(c)>0)
        a=c;
    else
        b=c;
    end
i=i+1;
end
m=(a+b)/2
fprintf('Root lies at c= %f \n',m )
```