



# Lecture NO.01

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

*imrantalib@vu.edu.pk*

# Outline

## 1 Introduction

- Installation
- Starting MATLAB
- Use of the Command Window
- Online Help
- MATLAB Windows
- Ending a Session



# Installation

If you intend to run MATLAB on a PC, particularly the Student Version, it is quite possible that you will have to install it yourself. The following points can be beneficial for installation of the MATLAB

- The task can be easily accomplished using the product CDs.
- Follow the installation instructions as you would with any new software installation.
- During the installation you may be asked which *toolboxes* you wish to install.
- We suggest that you install any that seem of interest to you or that you think you might use at some point in the future.
- But for the purpose of this course , you should be sure to include the *Symbolic Math Toolbox*.



# Starting MATLAB

You start MATLAB as you would any other software application:

- On a PC you access it via the **Start** menu, in the **programs** folder under MATLAB R2010a or MATLAB 7.0.
- Alternatively you may have a desktop icon that enables you to start MATLAB with a simple-double click
- As you start of MATLAB, you will briefly see a window that displays the MATLAB logo alongwith some MATLAB product information, like version of MATLAB etc.
- Then a MATLAB desktop window will launch.
- That window will contain a title bar, a menu bar, a tool bar, and four and embedded windows, one of which is hidden.
- The largest and most important window is the **Command Window** on the left.



# Starting MATLAB

- Later we will go into more detail on the use and manipulation of the other three windows: the *Workspace Browser*, the *Command History Window*, and the *Current Directory Browser*.
- Now, we focus on Command Window in order to get you started issuing MATLAB commands.
- At the top of Command Window, you may see some general information about MATLAB, the most important of all, you will see a command prompt (»).
- If the Command Window is active, its title bar will be dark, and the prompt will be followed by a cursor (a blinking vertical line). That is the place where you will enter your MATLAB Commands.



# Starting MATLAB

- Figure 1 contains an example of a newly launched MATLAB Desktop.

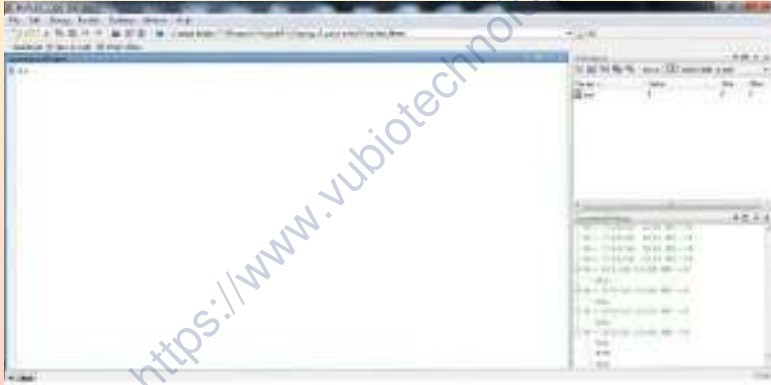


Figure: MATLAB Desktop



# Use of the Command Window

To type the mathematics in a Command Window follow the following points:

- Click in the Command Window to make it active.
- When a Window becomes active, its titlebar darkens and a blinking cursor appear after the prompt.
- Now you in a position to enter simple commands in a Command Window.



# Use of the Command Window

- Type  $2 + 5$ , then press ENTER key, the output will be 7.
- Type `factor(987654321)` and press ENTER key.
- Type `factor(9)`.
- Type `factor(10)`.
- Your MATLAB desktop looks like the Figures 2 and 3.





# Use of Command Window

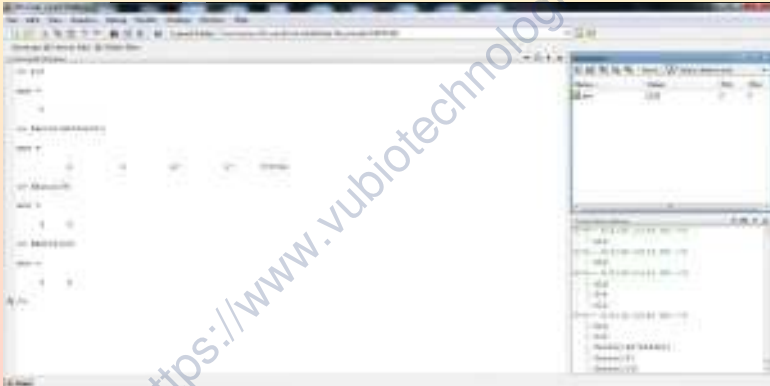
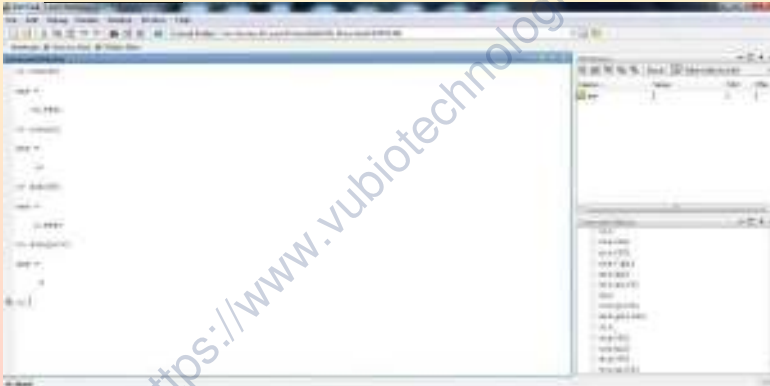


Figure: (2) MATLAB Desktop with several simple commands



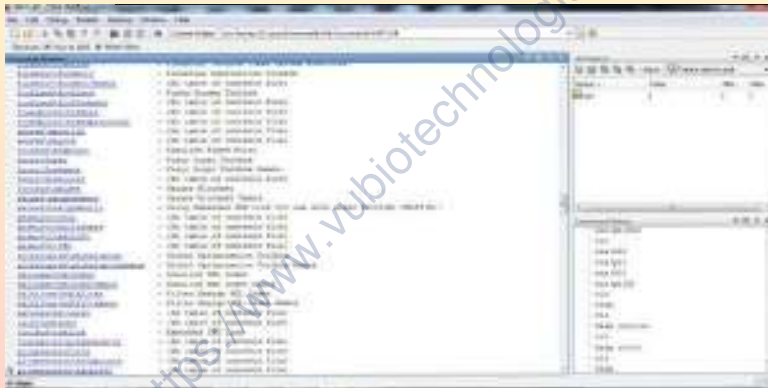


# Online Help

MATLAB has extensive online help that can be gotten in one of the several ways

- Typing **help** at the command prompt will reveal a long list of topics for which help is available.
- To see a general purpose MATLAB Commands, type **help general** will reveal a long list of general help commands.
- To learn about the solve commands, type **help solve**.
- Your MATLAB Desktop looks like the Figures 4, 5, and 6.

# Online Help





# Online Help

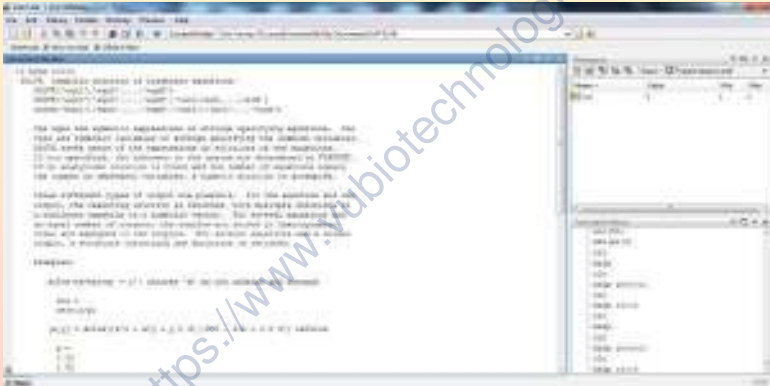


Figure: (6) MATLAB Desktop with help solve commands



# Online Help

There is much more user friendly way to access the online help, namely via the MATLAB **Help Browser**. It can be activated in a several way. For example

- Type helpwin, helpdesk or doc in the Command Window.
- Your MATLAB Desktop looks like the Figures 7, 8, and 9.

# Online Help

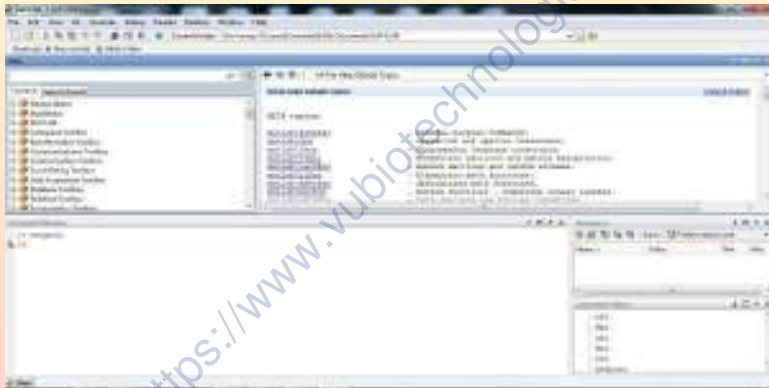


Figure: (7) MATLAB Desktop with helpwin command





# Online Help

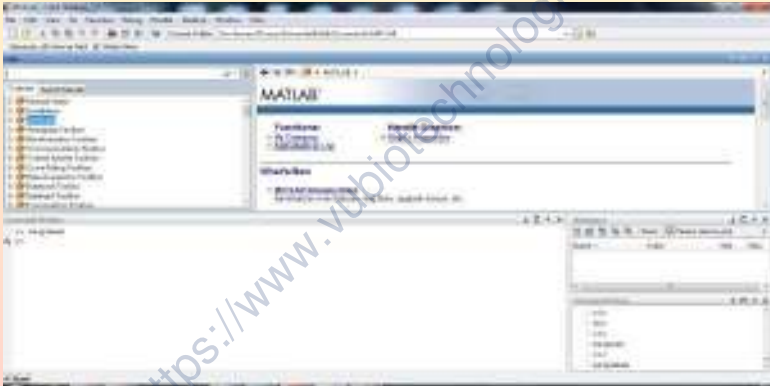


Figure: (8) MATLAB Desktop with helpdesk command



# Online Help

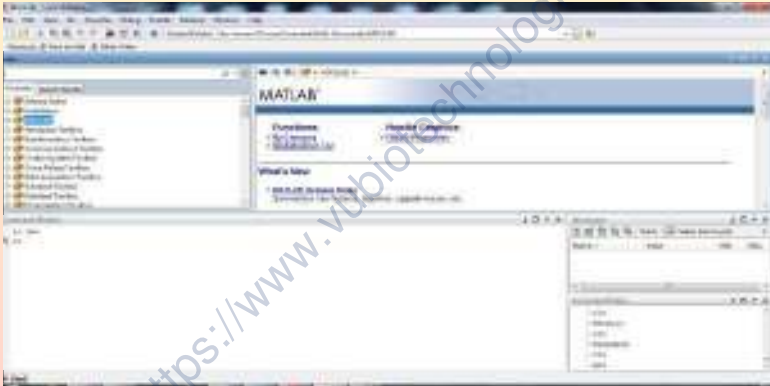


Figure: (9) MATLAB Desktop with doc command



# MATLAB Windows

There are several windows you will encounter as you work with MATLAB, like

- Command Window
- Help Browser
- Command History Window
- Current Dictionary Browser
- Workspace Browser
- Editor (M-files)

Some of these windows launched separately, and some are embedded in the Desktop through the Desktop menu button. We will discuss later in detail the functionality of the Editor.



# MATLAB Windows

The Matlab Desktop display after embedding the different windows inside the Desktop, see the following Figure

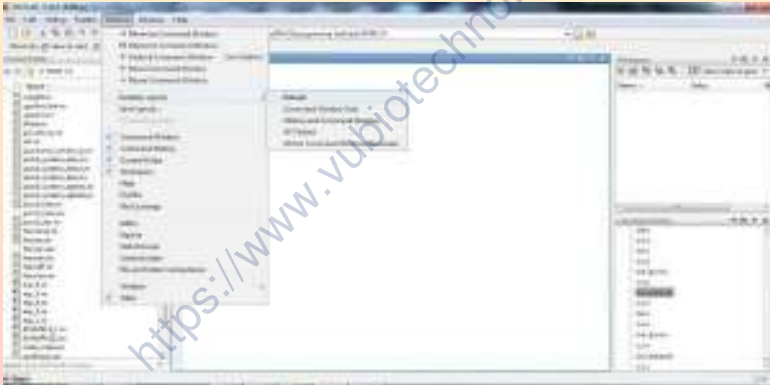


**Figure:** The Matlab Desktop display after embedding the different windows inside the  
[visit our website to get more PDF](https://www.vubiotechnologist.gq) <https://www.vubiotechnologist.gq>



## Ending a Session

- The simplest way to conclude a MATLAB session is to type **quit** in the Command Window.
- Special symbol can also be used (like X in the upper right hand corner) that closes your window.
- **Exit MATLAB** option from the File menu of the Desktop can also be used to close the window.



# Doing Mathematics with Matlab (I)

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

*imrantalib@vu.edu.pk*

# Outline

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

# Outline

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



# 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.

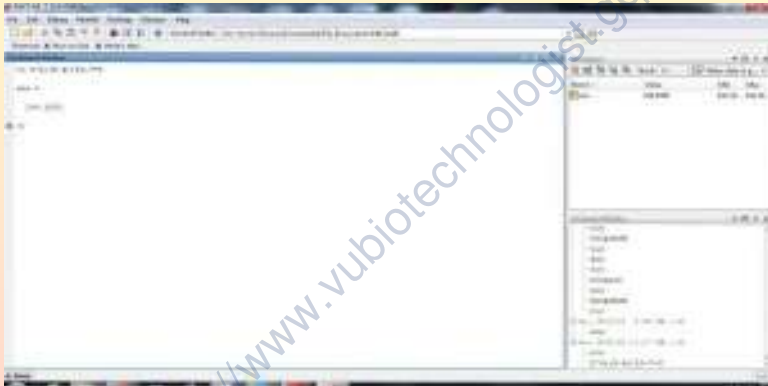


Figure: (1) The Matlab Desktop display after performing some basic arithmetic commands

# 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

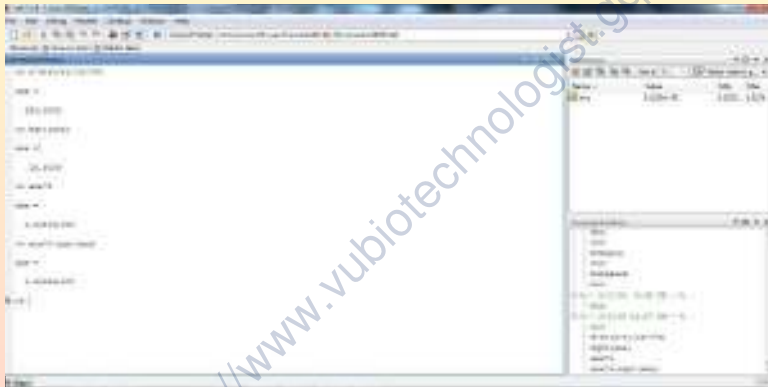


Figure: (2) The Matlab Desktop display after performing some basic arithmetic commands

# 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.

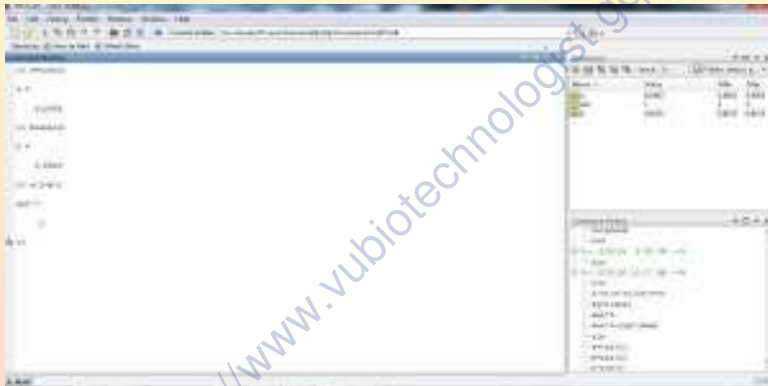
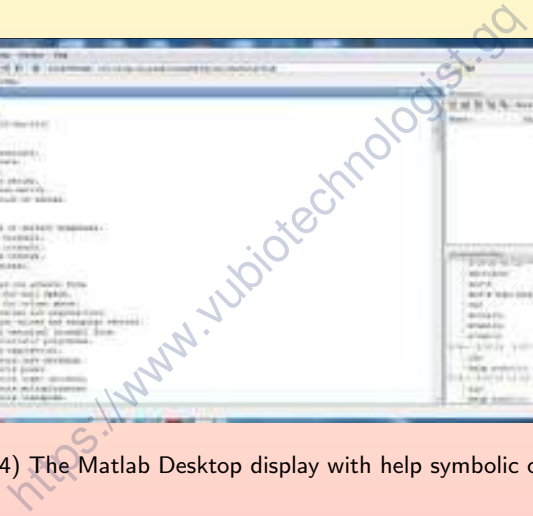


Figure: (3) The Matlab Desktop display after performing some basic arithmetic commands

# 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.



#### 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  $\pi$  accurately to about 15 digits. But sym command gives accurate value of  $\pi$ . 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.

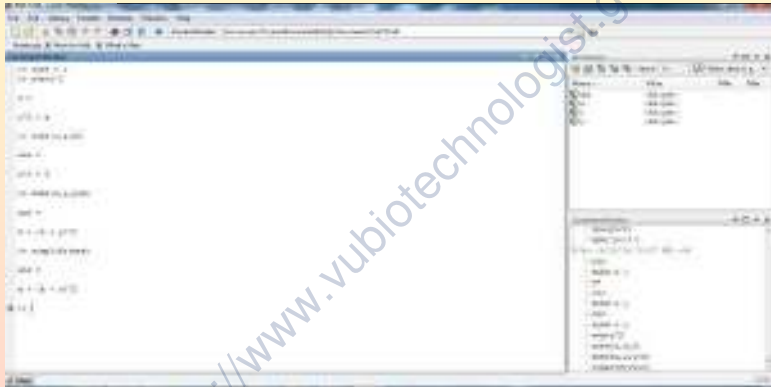


Figure: (7) The Matlab Desktop display with substitution command

## 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 *sin10*:
- » *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.

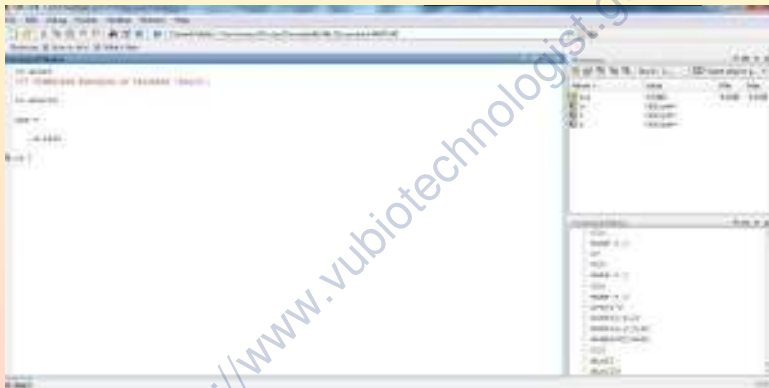


Figure: (8) The Matlab Desktop display while removing error





- ❶ Find the value of the following polynomials at  $x = 2$  using **subs** command:
  - $p(x, y) = x^2 + 3y^2x + 10$
  - $u(x) = x^3 + 4x + 10$
- ❷ 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

## 1 Doing Mathematics with MATLAB II

- Variables and Assignments
- Vectors
- Built-in Functions
- User Defined Functions
- Solving Algebraic Equations

## 2 Exercise

# Outline

## 1 Doing Mathematics with MATLAB II

- Variables and Assignments
- Vectors
- Built-in Functions
- User Defined Functions
- Solving Algebraic Equations

## 2 Exercise

# 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 » x3 + 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

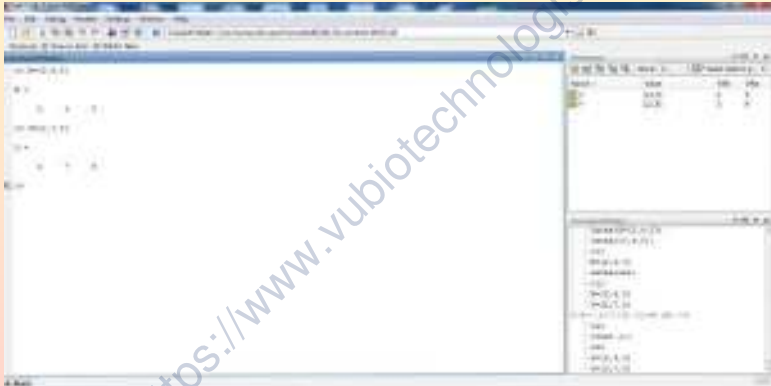


Figure: The Matlab Desktop display with Vector command

# 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 from 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

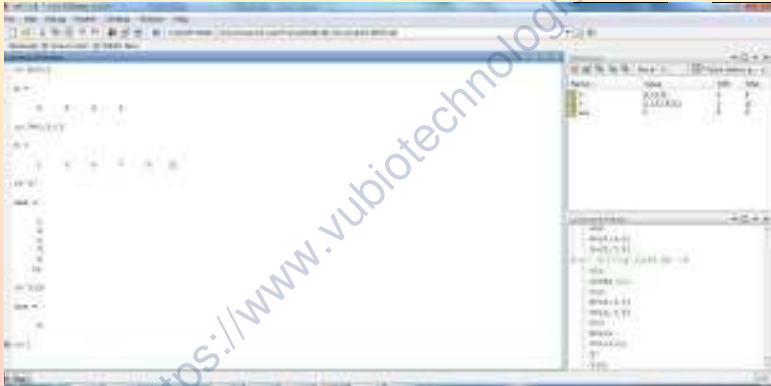


Figure: The Matlab Desktop display with increment command

# Vectors

- Mathematical operations can also be performed on vectors. For example, to square the elements of 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

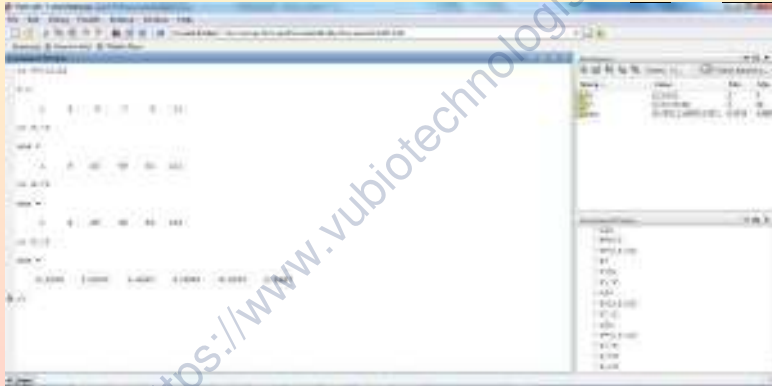


Figure: The Matlab Desktop display with operations on vectors

# 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
- $\gg f = @(x)x^3$   
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

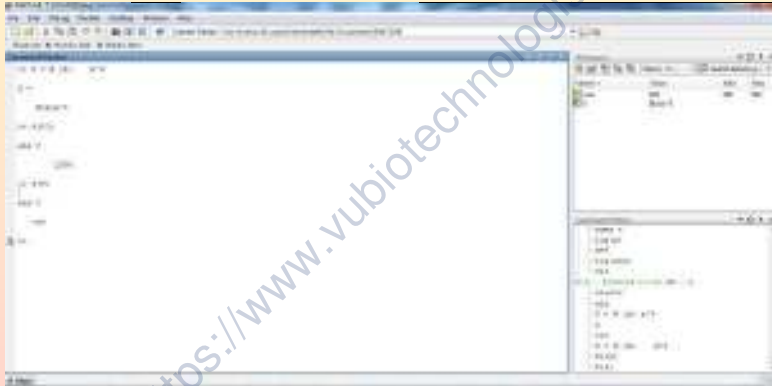


Figure: The Matlab Desktop display with user defined function command

## 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

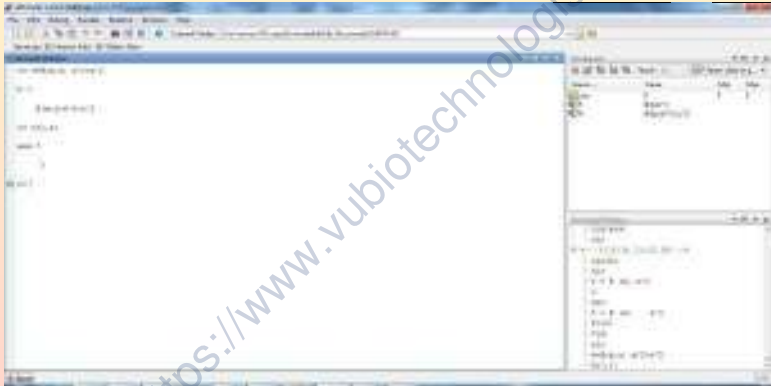


Figure: The Matlab Desktop display with user defined function command

# 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
- » `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

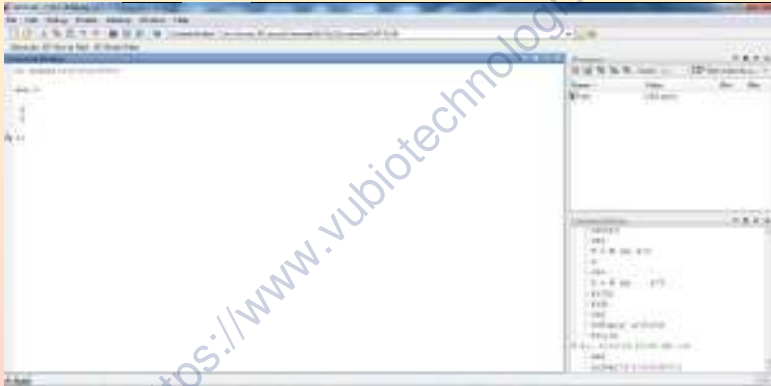


Figure: The Matlab Desktop display with Algebraic equations solver

# 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(x2 - 3 * x + 7)`

- The MATLAB generated output is

- `ans =`  
 $\frac{3}{2} - \frac{(19^{1/2}) * i}{2}$   
 $\frac{3}{2} + \frac{(19^{1/2}) * i}{2}$



# Solving Algebraic Equations

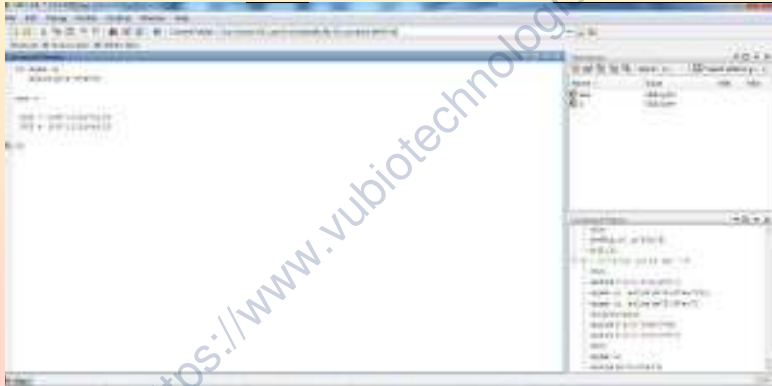


Figure: The Matlab Desktop display with Algebraic equations symbolic command

# 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:
  - `» [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

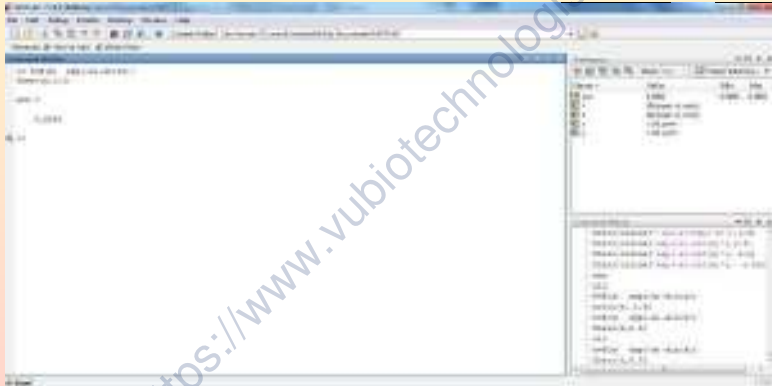


Figure: The Matlab Desktop display with fzero command

## 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

## 1 Graphics Techniques in MATLAB

- ezplot Command
- ezplot command with symbolic expression
- ezplot command on anonymous function
- Modification of Graphs
- Example
- Plot command
- Graphs of Multiple Curves
- Parametric Plots
- Contour Plot
- Exercise

## 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

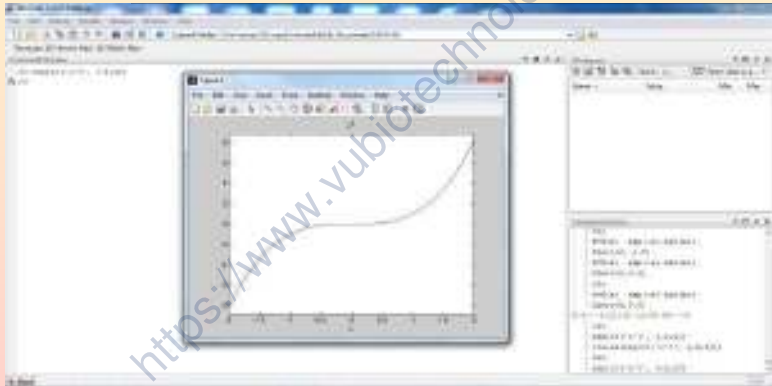


Figure: String form plot with ezplot command

- **ezplot** command can also accept 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

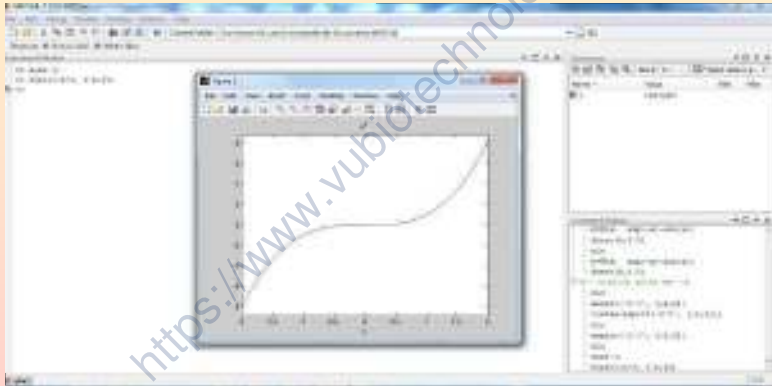


**Figure:** Symbolic expression form plot with ezplot command



- 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

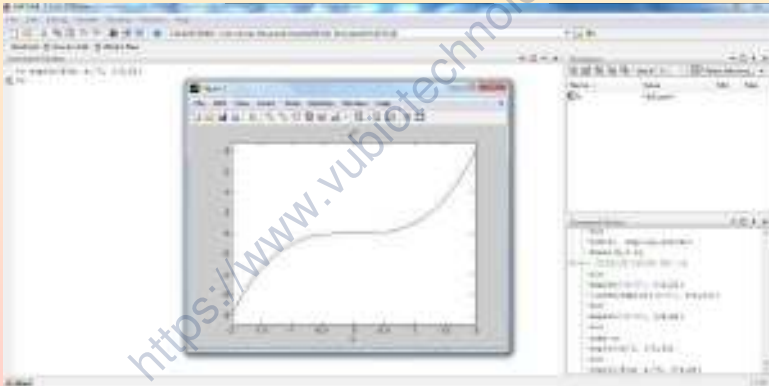


**Figure:** Anonymous function form plot with ezplot command



- 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(x3, [-2,2])  
xlabel('x');  
ylabel('y');  
title('The graph of  $y = x^3$ ');  
grid;
```

 see the output 3





## Modification of Graphs



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



**Example**

Plot command  
Graphs of Multiple Curves  
Parametric Plots  
Contour Plot  
Exercise

## 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

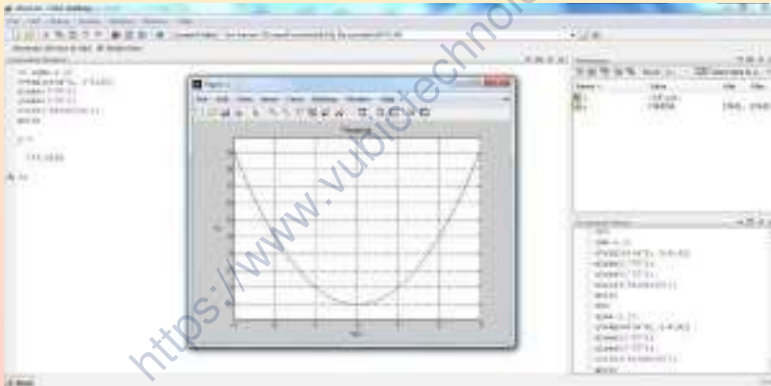


**Figure:** Modification of the Graph  $y = x^3$

## 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

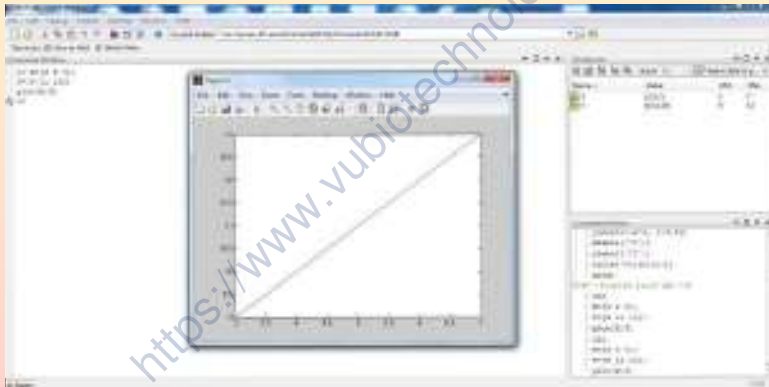


Figure: Plot command view of vectors of same length



## 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  $x^2 + x + 1$ .

### Solution:

- 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

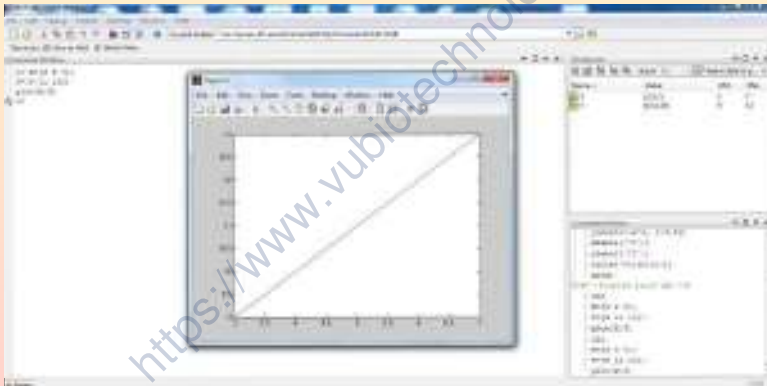


Figure: Plot command view of the Parabola





## 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

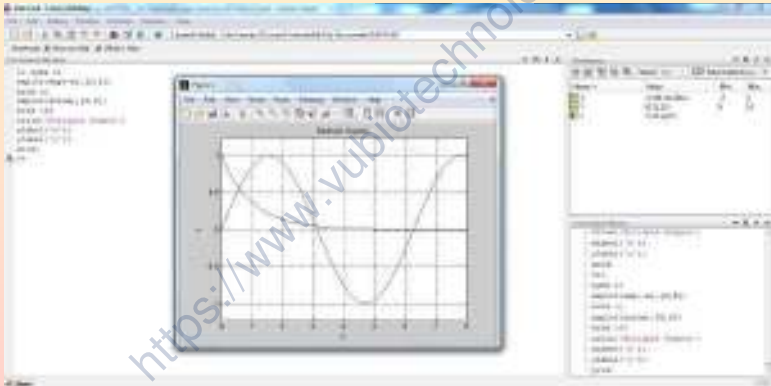


Figure: Ezplot command view of multiple curves



## 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

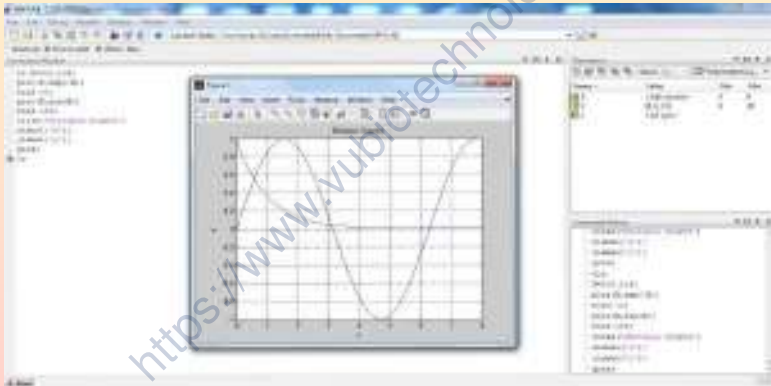


Figure: plot command view of multiple curves

## 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

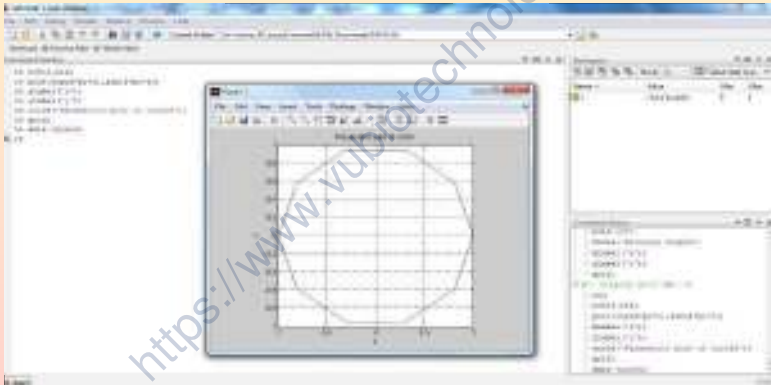


Figure: Parametric plot of the circle using plot command



## 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

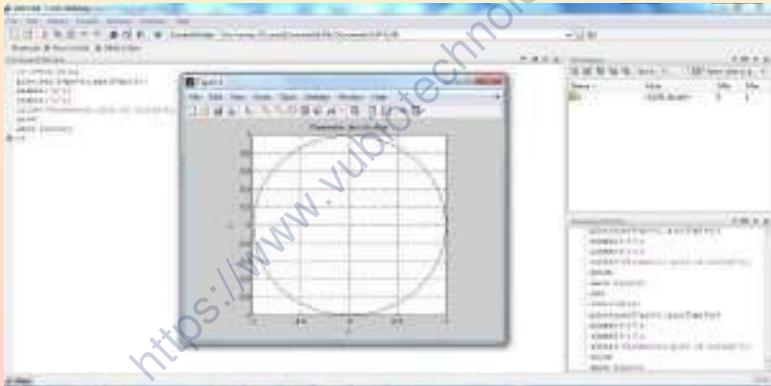


**Figure:** Parametric plot of the circle using plot command

## 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:

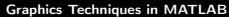
- Plot the contour plots of the expression  $x^2 + y^2$  using **meshgrid** and **contour** commands.
- Label the horizontal axis with  $x$  and vertical axis with  $y$ .
- Make the shape of the graph square.
- Keeps the title of the graph "Contour plotting of squares".



# 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

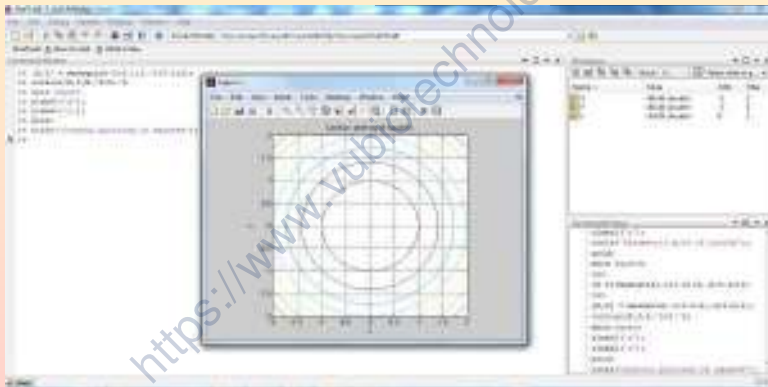


Figure: Contour plotting



## 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.

# Calculus With MATLAB

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)

# Outline

- 1 Doing Calculus with MATLAB
  - Differentiation
  - Differentiation with respect to a particular variable
- 2 Higher Order Derivatives
  - Higher Order Derivatives of Univariate Expression
  - Higher Order Derivatives of Multi-variate Expression
- 3 Integration
  - Indefinite integral
  - Definite Integral
  - Improper Integral
- 4 Exercises

# Outline

## 1 Doing Calculus with MATLAB

- Differentiation
- Differentiation with respect to a particular variable

## 2 Higher Order Derivatives

- Higher Order Derivatives of Univariate Expression
- Higher Order Derivatives of Multi-variate Expression

## 3 Integration

- Indefinite integral
- Definite Integral
- Improper Integral

## 4 Exercises



# Outline

## 1 Doing Calculus with MATLAB

- Differentiation
- Differentiation with respect to a particular variable

## 2 Higher Order Derivatives

- Higher Order Derivatives of Univariate Expression
- Higher Order Derivatives of Multi-variate Expression

## 3 Integration

- Indefinite integral
- Definite Integral
- Improper Integral

## 4 Exercises

# Outline

## 1 Doing Calculus with MATLAB

- Differentiation
- Differentiation with respect to a particular variable

## 2 Higher Order Derivatives

- Higher Order Derivatives of Univariate Expression
- Higher Order Derivatives of Multi-variate Expression

## 3 Integration

- Indefinite integral
- Definite Integral
- Improper Integral

## 4 Exercises

# Differentiation

- To differentiate a symbolic expression, MATLAB uses **diff** command.
- The recipe for differentiation of a symbolic expression is:
  - » `syms x`
  - » `diff(sin(x2), x)`
  - `ans =`
  - `2 * x * cos(x2)`

# Differentiation

## Example:

- Differentiate the following functions using **diff** command.
- Moreover plot the multiple graphs of the given functions and their derivatives on the interval  $[0, 5]$ .
- $f(x) = x^2 + 2x + 10$ .
- $g(x) = \sin(x)$ .

# Differentiation

**Solution:** The recipe of the solution for the function  $f(x)$  is as under:

```
» syms x
diff(x2 + 2 * x + 10, x)
ans =
2 * x + 2
ezplot(x2 + 2 * x + 10, [0,5]);
hold on
ezplot(ans, [0,5]);
hold off
title('Multiple plot of x2 + 2 * x + 10 and its derivative 2 * x + 2');
xlabel('x');
ylabel('y');
grid;
```

# Differentiation

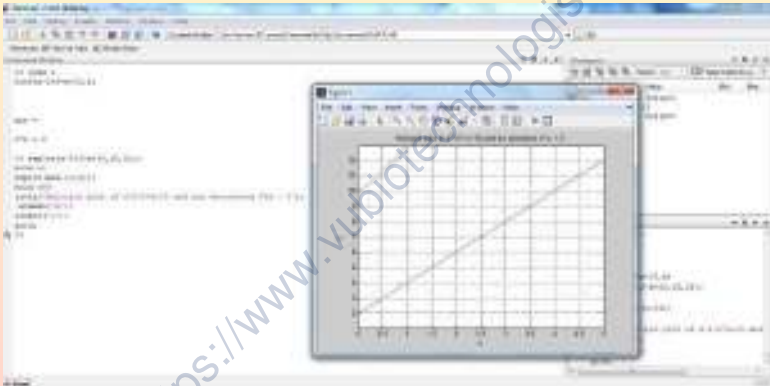


Figure: Multiple plot of the function  $f(x)$  and its derivative.

# Differentiation

**Solution:** The recipe of the solution for the function  $g(x)$  is as under:

```

» syms x
diff(sin(x), x)
ans =
2 * x + 2
ezplot(sin(x),[0,5]);
hold on
ezplot(ans,[0,5]);
hold off
title('Multiple plot of  $\sin(x)$  and its derivative  $\cos(x)$ ');
xlabel('x');
ylabel('y');
grid;

```

# Differentiation

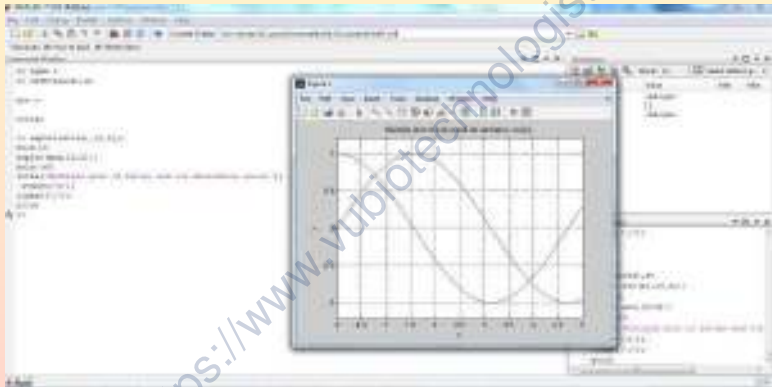


Figure: Multiple plot of the function  $g(x)$  and its derivative.



# Differentiation with respect to a particular variable

- Sometimes there are two or more variables are involved in a function.
- Then it is necessary to specify the differentiation variable.
- For example the MATLAB syntax to find the derivative of the function  $f(x, t) = \sin(x * t^2)$  with respect to  $x$  and  $t$  is as follows

• `syms x t`

`diff(sin(x * t^2), x);`

`diff(sin(x * t^2), t);`

- The following outputs are generated:

• `» syms x t`

`» diff(sin(x * t^2), x)`

`ans =`

`t^2 * cos(t^2 * x)`

`» diff(sin(x * t^2), t)`

`ans =`

`2 * t * x * cos(t^2 * x)`

# Differentiation with respect to a particular variable

## Example:

- Find the derivative of the following function with respect to  $x$ , and  $t$  variables using **diff** command. Moreover plot the multiple graph of the function and their derivatives by defining  $X$  and  $Y$  using **meshgrid** command, Then define  $Z$  and create a surface plot:
- $f(x, t) = xe^{-x^2-t^2}$ ,  $x, t \in [-2, 2]$ .

# Differentiation with respect to a particular variable

## Solution:

- The recipe for the solution is as under:
- » `[X,T]=meshgrid(-2:0.2:2,-2:0.2:2);`  
`Z1=X.*exp(-X.^2-T.^2);`  
`surface(X,T,Z1)`  
`hold on`  
`view(3)`  
`Z2=1./exp(T.^2+X.^2)-(2*X.^2)./exp(T.^2+X.^2);`  
`surface(X,T,Z2)`  
`hold off`  
`view(3)`  
`xlabel('x');`  
`ylabel('y');`  
`title('Surface plot')`

# Differentiation with respect to a particular variable

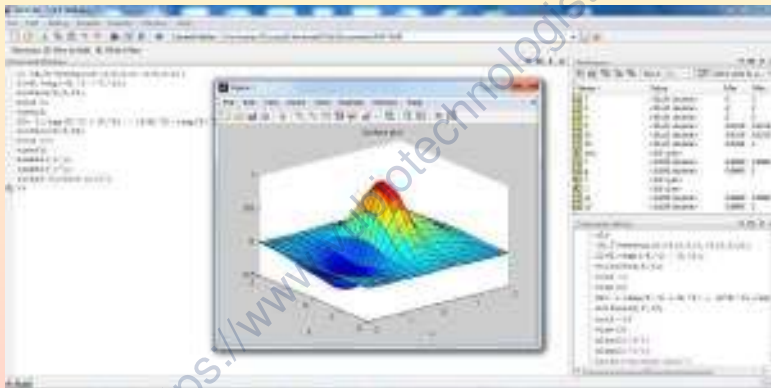


Figure: Surface plot of the function  $f(x, t) = xe^{-x^2-t^2}$ ,  $x, t \in [-2, 2]$ .

# Higher Order Derivatives of Univariate Expression

- MATLAB also facilitate to find the higher order derivatives of univariate and multi-variate expressions.
- The recipe to find the higher order derivatives of univariate expression is as under:
- » `syms x`  
» `diff(x10 + 2 * (x2) + cos(x),order of derivative)`

# Higher Order Derivatives of Univariate Expression

## Example:

- Find the first-order, second-order, and fourth-order derivatives of the function  $f(x) = x^{10} + 2 * (x^2) + \cos(x)$  using **diff** command.
- Plot the multiple graphs of the outputs using **ezplot** command.
- Label the horizontal axis with  $x$  and vertical axis with  $y$ .
- Keeps the title "Multiple plot of the function and its derivatives"

# Higher Order Derivatives of Univariate Expression

## Solution:

- The recipe for the solution is as under:

- » `diff(x10 + 2 * (x2) + cos(x),1)`

ans =

$$4 * x - \sin(x) + 10 * x^9$$

- » `diff(x10 + 2 * (x2) + cos(x),2)`

ans =

$$90 * x^8 - \cos(x) + 4$$

- » `diff(x10 + 2 * (x2) + cos(x),4)`

ans =

$$\cos(x) + 5040 * x^6$$

# Higher Order Derivatives of Univariate Expression

## Solution:

- The recipe for graphical view is as under:
- » `ezplot(x10 + 2 * (x2) + cos(x),[-pi,pi])`  
hold on  
`ezplot(4 * x - sin(x) + 10 * x9,[-2*pi,2*pi])`  
hold on  
`ezplot(90 * x8 - cos(x) + 4,[-3*pi,3*pi])`  
hold on  
`ezplot(cos(x) + 5040 * x6,[-2*pi,2*pi])`  
hold off  
grid  
xlabel('x')  
ylabel('y')  
title('Multiple plot of the function and its derivatives')



# Higher Order Derivatives of Univariate Expression



Figure: Higher order derivatives of univariate expressions.

# Higher Order Derivatives of Univariate Expression

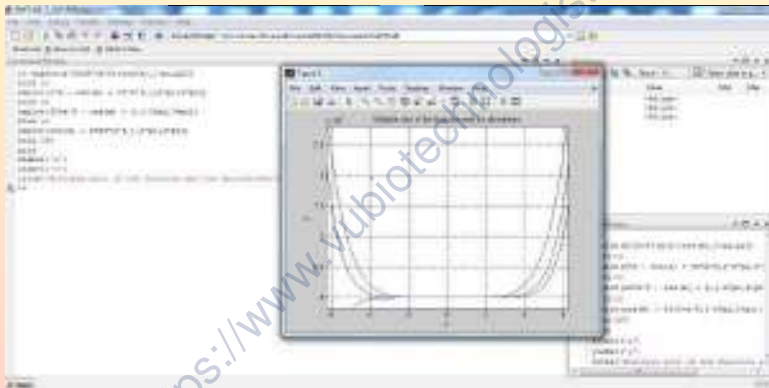


Figure: Multiple plot of the function and its derivatives.

# Higher Order Derivatives of Multi-variate Expression

- MATLAB also facilitate to find the higher order derivatives of multi-variate expressions
- The recipe to find the higher order derivatives is as under:
- » `syms x y`  
  » `diff(2 * x * y + (x * y)3, y, order of derivative)`

# Higher Order Derivatives of Multi-variate Expression

## Example:

- Find the first-order, second-order, and fourth-order derivatives of the function  $f(x) = 2 * x * y + (x * y)^3$  with respect to variable  $y$  using **diff** command.

# Higher Order Derivatives of Multi-variate Expression

## Solution:

- The recipe for the solution is as under:

```
» syms x y
» diff(2 * x * y + (x * y)^3,y,1)
ans =
3 * x^3 * y^2 + 2 * x
» diff(2 * x * y + (x * y)^3,y,2)
ans =
6 * x^3 * y
» diff(2 * x * y + (x * y)^3,y,4)
ans =
0
```

# Higher Order Derivatives of Multi-variate Expression



Figure: Multiple plot of the function and its derivatives.

# Indefinite integral

- Matlab also used to evaluate the indefinite and definite integrals of the functions.
- **int** command is used for this purpose.
- For example, the integral of the function  $x^3$  can be evaluated using the following syntax:
  - » syms x;
  - » int( $x^3$ ,x)
- The output is
- $x^4/4$ .

# Indefinite integral of Multi-variate Expression

- To find the indefinite integral of multi-variable expression, it is necessary to mention the integrable variable.
- For example, the integral of the function  $x/(1 + y^2)$  with respect to variables  $x$  and  $y$  can be evaluated using the following MATLAB syntax:
- » `syms x y`  
 » `int(x/(1 + y^2),x)`  
`ans =`  
 $x^2/(2 * (y^2 + 1))$   
 » `int(x/(1 + y^2),y)`  
`ans =`  
 $x * \text{atan}(y)$



# Indefinite Integral

## Example:

- Find the integral of the function  $f(x) = x^2 + 10x + 4$  using **int** command.
- Plot the multiple graphs of the function and its output using **plot** command.
- Label the horizontal axis with  $x$  and vertical axis with  $y$ .
- Keeps the title "Graph of function and its integral"

# Indefinite Integral

## Solution:

```

» syms x
int(x2 + 10 * x + 4)
ans =
(x * (x2 + 15 * x + 12))/3
» X=-2:0.01:2;
plot(X.2 + 10. * X + 4, X, (X. * (X.2 + 15. * X + 12))./3,X)
» grid;
» xlabel('x');
» ylabel('y');
» title('Graph of function and its integral')

```

# Indefinite Integral

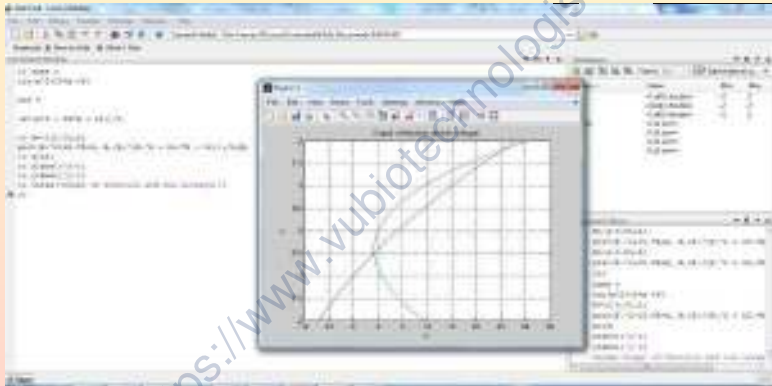


Figure: Multiple plot of the function and its integral.

# Definite Integral of Univariate Expression

- Definite integral can also be evaluated using **int** command.
- For example, the MATLAB syntax to evaluate the definite integral of the function  $x^3$  from 0 to 1 is as under:
  - » syms x
  - » int( $x^3$ ,0,1)
  - ans =
  - 1/4

# Definite Integral of Univariate Expression

- Some integrals can not be evaluated symbolically in MATLAB using **int** command.
- Such type of integrals can be computed numerically using **double** command.
- For example the integral of the function  $\log(1 + x^2) * e^{(-x^2)}$  can not be computed symbolically. it is computed using the following MATLAB
- » syms x  
» hardintegral=int(log(1 + x<sup>2</sup>) \* e<sup>(-x<sup>2</sup>)</sup>,x,0,1)  
» double(hardintegral).
- The out put is:  
ans =  
0.1539

# Definite Integral

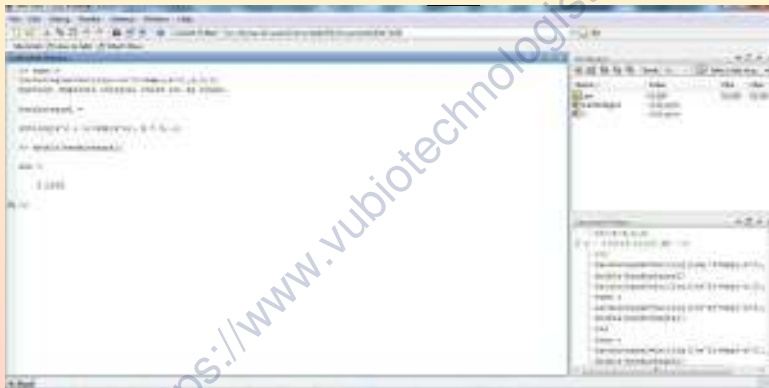


Figure: Evaluation of the definite integral with double command.

# Improper Integral

- MATLAB also have ability to evaluate the improper integrals using **int** command.
- For example the integral of the function  $\sin(x)/x$  from 0 to infinity can be computed using the following MATLAB syntax:
  - » syms x
  - » int( $\sin(x)/x$ , x, 0, inf)
- The output is:  
ans =  
pi/2

# 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"



# 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**
- Label the horizontal axis with  $x$  and vertical axis with  $y$ .
- Keeps the title "Multiple plot of the function and its derivatives"

# 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"



# Use Of the M-Files

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

*imrantalib@vu.edu.pk*



# Outline

## 1 M-Files

- Attributes of the M-Files
- Script M-files
- File saving procedure
- Execution or run of the Script file
- Display of the commands with results in Command Window
- Adding Comments
- Structuring Script M-File
- Function M-Files
- Loops

## 2 Exercise



# Outline

## 1 M-Files

- Attributes of the M-Files
- Script M-files
- File saving procedure
- Execution or run of the Script file
- Display of the commands with results in Command Window
- Adding Comments
- Structuring Script M-File
- Function M-Files
- Loops

## 2 Exercise



- 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

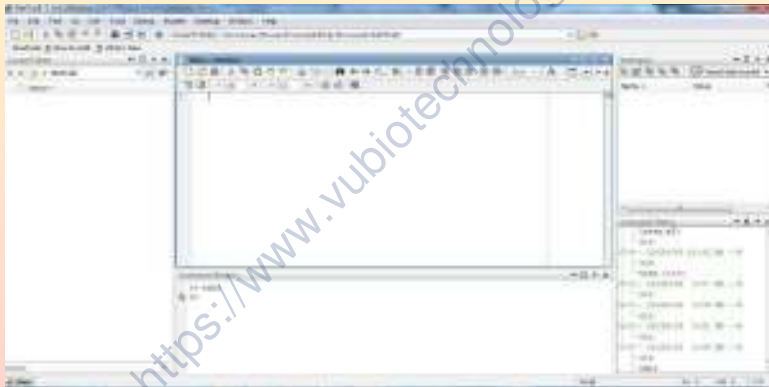


Figure: Display of the MATLAB desktop with Editor Window





- 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.

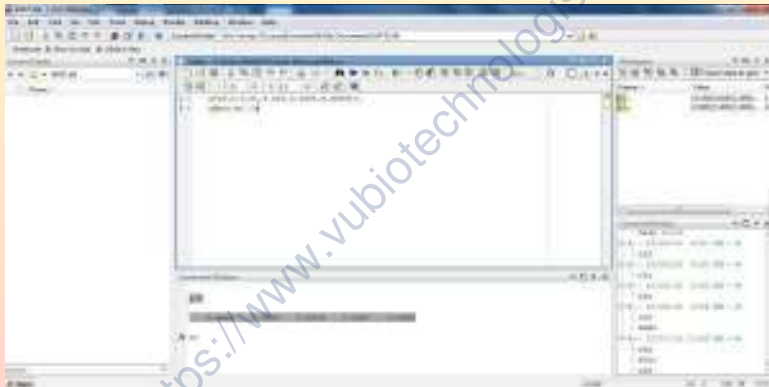


Figure: Computational values of  $\frac{\sin(x)}{x}$



## 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.

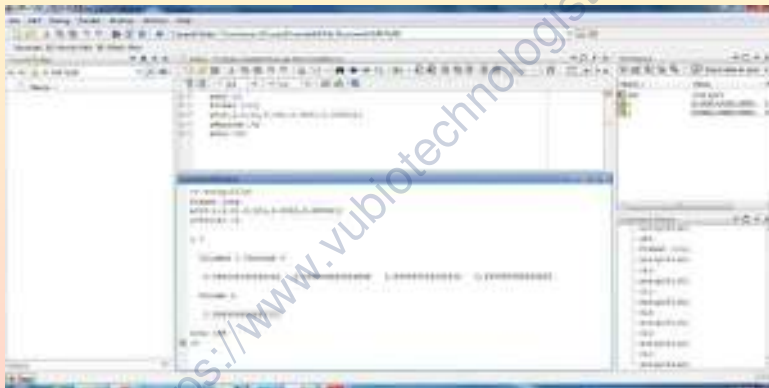


**Figure:** Display of the Commands along with the results in the Command Window using echo command.





- 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

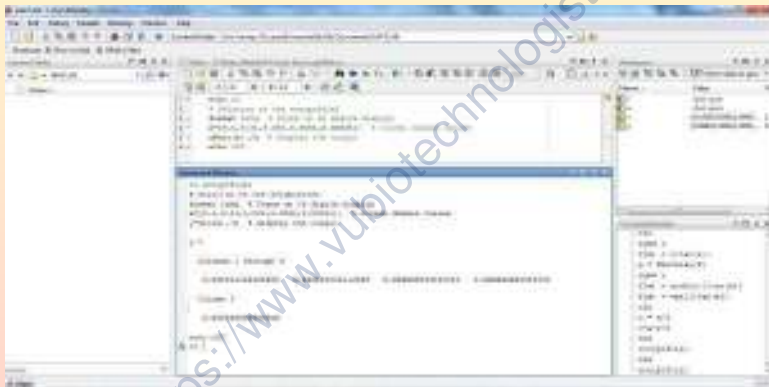


**Figure:** Computational of the values of  $y = \sin(x)/x$  at given domain values with adding comments.



- 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.

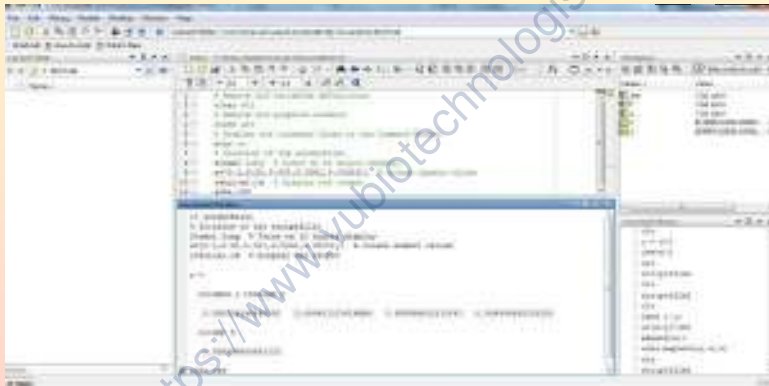


Figure: Structuring of the solution file with clear all and close all command.



- 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.

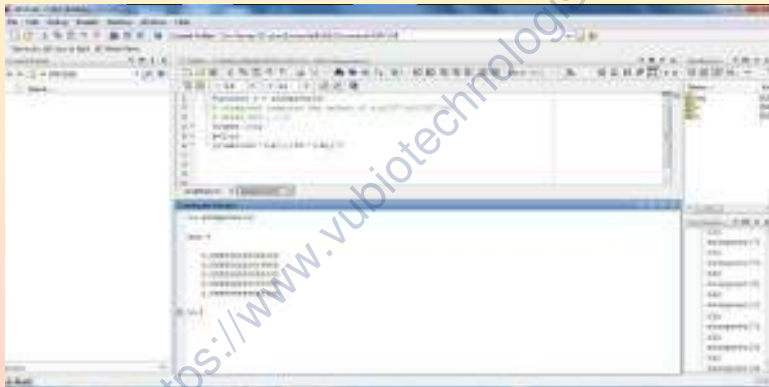


Figure: Solution of the previous example using Function M-File.





- 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.

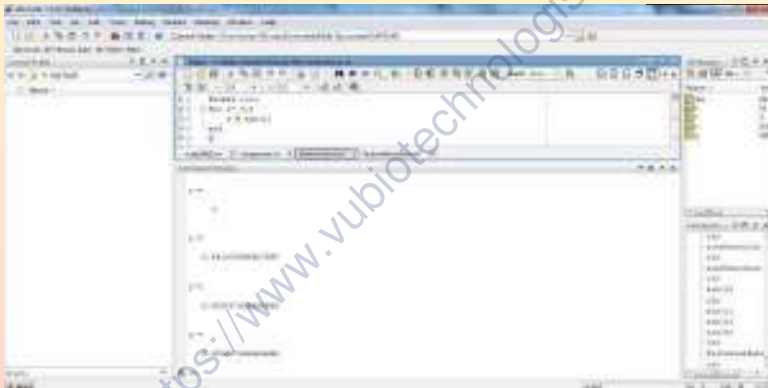


**Figure:** Computation of the values of sine function using for loop.



## 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**.



# Solving Ordinary Differential Equations (ODEs) in MATLAB

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)

# Outline

- 1 Solving Ordinary Differential Equations (ODEs) in MATLAB
  - Explicit solution of the first order equations
  - Graphical Views of the First-order Equations
  - Graphical view of the family of solutions
- 2 Second-order Ordinary Differential Equations in MATLAB
  - Explicit Solution of Second-order ODEs
  - Examples
- 3 Solution Of System Of Differential Equations
  - Example
- 4 Exercise

# Outline

- 1 Solving Ordinary Differential Equations (ODEs) in MATLAB
  - Explicit solution of the first order equations
  - Graphical Views of the First-order Equations
  - Graphical view of the family of solutions
- 2 Second-order Ordinary Differential Equations in MATLAB
  - Explicit Solution of Second-order ODEs
  - Examples
- 3 Solution Of System Of Differential Equations
  - Example
- 4 Exercise

# Outline

- 1 Solving Ordinary Differential Equations (ODEs) in MATLAB
  - Explicit solution of the first order equations
  - Graphical Views of the First-order Equations
  - Graphical view of the family of solutions
- 2 Second-order Ordinary Differential Equations in MATLAB
  - Explicit Solution of Second-order ODEs
  - Examples
- 3 Solution Of System Of Differential Equations
  - Example
- 4 Exercise

# Outline

- 1 Solving Ordinary Differential Equations (ODEs) in MATLAB
  - Explicit solution of the first order equations
  - Graphical Views of the First-order Equations
  - Graphical view of the family of solutions
- 2 Second-order Ordinary Differential Equations in MATLAB
  - Explicit Solution of Second-order ODEs
  - Examples
- 3 Solution Of System Of Differential Equations
  - Example
- 4 Exercise

# First order equations

- MATLAB is primarily a numerical package, it can certainly solve differential equations symbolically.
- For this purpose MATLAB uses **dsolve** command.
- For example, the input and output for solving  $\frac{dy}{dx} = x * y$  in MATLAB is given below:
- » `dsolve('Dy = x * y','x')`  
ans =  
 $C2 * \exp(x^2/2)$
- Note that MATLAB uses capital letter **D** to indicate the derivative and requires that the entire equation appears in single quote.

## First order equations

- Alternatively, if we are interested in using the  $\frac{dy}{dx} = x * y$  a number of times, then define it as a variable, say, eqn1
- The input and output for this purpose is given below:
- » eqn1 = 'Dy = x \* y'  
eqn1 =  
Dy = x \* y  
» y=dsolve(eqn1,'x')  
y =  
C2 \* exp(x<sup>2</sup>/2)

## First order equations

- MATLAB also has ability to find the particular solution of the differential equations corresponding to certain initial and boundary conditions.
- For example, the particular solution of the first-order problem  $\frac{dy}{dx} = x * y$  subject to initial condition  $y(1) = 1$  in MATLAB can be found as
- » `eqn1 = 'Dy = x * y'`  
`eqn1 =`  
`Dy = x * y`  
» `y=dsolve(eqn1,'y(1)=1','x')`  
`y =`  
`exp(x^2/2)/exp(1)^1/2)`



## First order equations

- Suppose, we want to observe the behaviour of the solution of the differential equation  $\frac{dy}{dx} = x * y$ .
- For this purpose, the **ezplot** command can be used to see the graphical view of the solution subject to certain initial condition.
- The input and output command for seeing the graphical view is as under:
- ```
» eqn1 = 'Dy = x * y'
eqn1 =
Dy=x * y
» y=dsolve(eqn1,'y(1)=1','x')
y =
exp(x^2/2)/exp(1)^(1/2)
» ezplot(y,[0,2]).
```

## First order equations

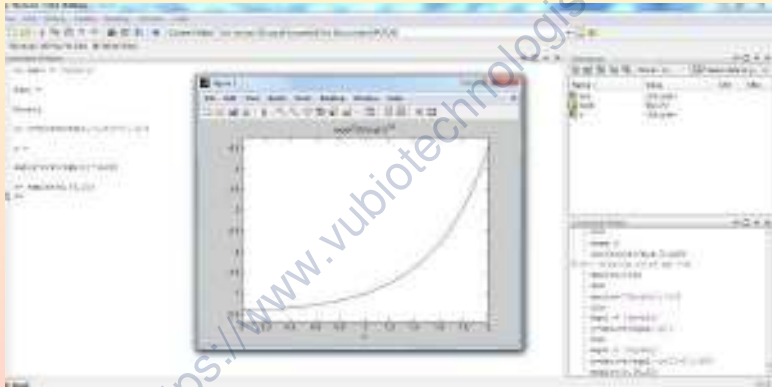


Figure: Graphical view of the solution of initial value problem on the interval  $[0,2]$ .

# First order equations

## Example:

- Find the solution of the following initial value problem
- $\frac{dy}{dx} = x^2 + y, y(1) = 3.$
- Moreover, plot the graph of the solution using **ezplot** command on the interval  $[0,2]$ .

# First order equations

## Solution:

- The recipe for the solution is as under:

- » `eqn1 = 'Dy = x^2 + y'`

`eqn1 =`

`Dy = x^2 + y`

- » `y=dsolve(eqn1,'y(1)=3','x')`

`y =`

`(8 * exp(x))/exp(1) - 2 * x - x^2 - 2`

- » `ezplot(y,[0,3])`

- » `xlabel('x')`

- » `ylabel('y')`

- » `grid`

- » `title('Graphical view of the initial')`

## First order equations



Figure: Graphical view of the solution of initial value problem on the interval  $[0,3]$ .

## First order equations

- Sometimes, we often want to see the family of solutions of the differential equations obtained by varying the initial conditions.
- Here is a natural way to do this in MATLAB.
- Begin by solving the given differential equation with a generic initial value.
- For example,
  - »  $\text{eqn1} = 'Dy = x^2 + y'$
  - $\text{eqn1} =$
  - $Dy = x^2 + y$
  - »  $y = \text{dsolve}(\text{eqn1}, 'y(0)=c', 'x')$
  - $y =$
  - $\exp(x) * (c + 2) - x^2 - 2 * x - 2$

## First order equations

- The solution formula  $\exp(x) * (c + 2) - x^2 - 2 * x - 2$  is in terms of the initial value  $c$ .
- Now let we want to plot the solution curves with initial values  $y(0) = -2, -1, 0, 2, -2$ , on the interval  $[0, 2]$ .
- one approach is to execute the command using for loop as explained in the next slide:

## First order equations

```
» eqn1 = 'Dy = x^2 + y'
eqn1 =
Dy = x^2 + y
» y=dsolve(eqn1,'y(0)=c','x')
y =
exp(x) * (c + 2) - x^2 - 2 * x - 2
» figure, hold on;
» syms x;
» for cval = -2:2
ezplot(subs(y,'c',cval),[0,2])
end
» axis tight
» xlabel('x')
» ylabel('y')
» title('Plot of family curves of  $Dy = x^2 + y$  at different initial conditions.
')
```



## First order equations

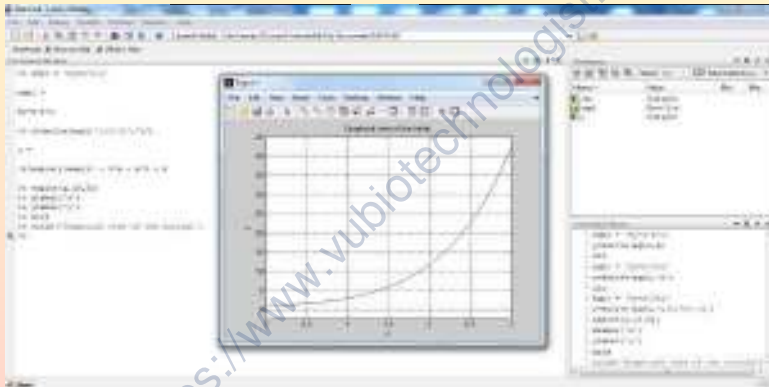


Figure: Plot of the family of curves of  $Dy = x^2 + y$  at different initial conditions on the interval  $[0,2]$  .

# First order equations

## Example:

- Plot the family of curves of the following differential equation on the interval  $[0,3]$ , with initial values  $y(0) = -3 : 3$ ,
- $\frac{dy}{dx} = x * y + 2 * x^2$ .

# First order equations

## Solution:

- The recipe for the solution is as under:
- » `eqn1 = 'Dy = x * y + 2 * x^2'` `eqn1 =`  
 $Dy = x * y + 2 * x^2$
- » `y=dsolve(eqn1,'y(0)=c','x')`
- `y =`  

$$c * \exp(x^2/2) - \exp(x^2/2) * ((2 * x) / \exp(x^2/2) - (pi^{1/2} * x^3) / (2 * (x^2/2)^{3/2})) + (pi^{1/2} * x^3 * \operatorname{erfc}((x^2/2)^{1/2})) / (2 * (x^2/2)^{3/2}))$$
- » `figure; hold on`
- » `for cval = -3:3`
- `ezplot(subs(y,'c',cval),[0,3])`
- `end`
- » `axis tight`
- » `xlabel('x')`
- » `ylabel('y')`
- `title('Solutions of Dy = x * y + 2 * x^2 with initial conditions`  
`y(0)=-3,-2,...,3').`

## First order equations

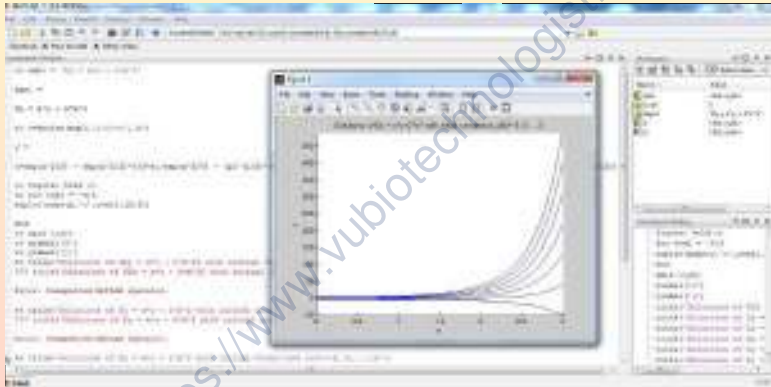


**Figure:** Plot of the family of curves of  $Dy = x^2 + y$  at different initial conditions on the interval  $[0,2]$  .

## Second-order ODEs

- **dsolve** command can also be used to find the solution of certain second-order ODEs.
- For example, the MATLAB syntax for solving the  $y''(x) + 8y'(x) + 2y(x) = \cos(x)$  is as under:
- » `eqn='D2y+8*Dy+2*y=cos(x)';`  
  » `dsolve(eqn,'x')`

## Example

- Solve the following second-order differential equation
- $y''(x) + y'(x) - 6y = 0$
- subject to the following initial conditions
- $y(0)=1, y'(0)=0$ .
- Moreover plot the solution curve on the interval  $[0, 2]$ .

## Solution

- The recipe for the solution is as under:
- »  $eqn = 'D^2y + Dy - 6 * y = 0';$   
initialcondns= 'y(0)=1,Dy(0)=0'; dsolve(eqn,initialcondns,'x')  
ans =  
 $(3 * \exp(2 * x))/5 + 2/(5 * \exp(3 * x))$   
»  $ezplot((3*\exp(2*x))/5 + 2/(5*\exp(3*x)),[0,2])$   
»  $xlabel('x');$   
»  $ylabel('y');$   
»  $grid$   
»  $title('The graph of (3 * \exp(2 * x))/5 + 2/(5 * \exp(3 * x))')$  .

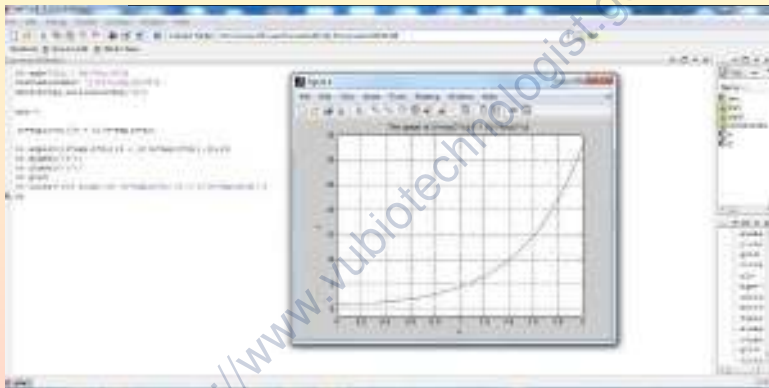


Figure: Graphical view of the solution curve of  $y''(x) + y'(x) - 6y = 0$ .



## Example

- Solve the following second-order differential equation
- $y''(x) + y'(x) - 6y = 0$
- subject to the following initial conditions
- $y(0)=1, y'(1)=0$ .
- Moreover plot the solution curve on the interval  $[0, 3]$ .

## Solution

- The recipe for the solution is as under:
- »  $eqn = 'D^2y + Dy - 6 * y = 0';$   
initialcondns= 'y(0)=1,Dy(1)=0'; dsolve(eqn,initialcondns,'x')  
 $y =$   
 $(2 * \exp(2))/( \exp(3 * x) * (2 * \exp(2) + 3/\exp(3))) + (3 * \exp(2 * x))/( \exp(3) * (2 * \exp(2) + 3/\exp(3)))$   
» ezplot(y,[0,2])  
» xlabel('x');  
» ylabel('y');  
» grid  
» title('The graph of the solution curve y')



Figure: Graphical view of the solution curve of  $y''(x) + y'(x) - 6y = 0$ .

- **dsolve** command can also be used to find the solution of system of differential equations.
- For example, we want to find the solution of the following system of differential equations
- $x'(t) = x(t) + 2y(t) - z(t),$
- $y'(t) = x(t) + z(t),$
- $z'(t) = 4x(t) - 4y(t) + 5z(t).$
- The MATLAB syntax with **dsolve** command is mentioned in the next slide:

- »  $eqn1 = ' Dx = x + 2 * y - z';$
- »  $eqn2 = ' Dy = x + z';$
- »  $eqn3 = ' Dz = 4 * x - 4 * y + 5 * z';$
- »  $[x,y,z]=dsolve(eqn1,eqn2,eqn3).$
- The desire output is as under:
- $x =$   
 $-(C16 * exp(t))/2 - (C17 * exp(2 * t))/2 - (C18 * exp(3 * t))/4,$   
 $y =$   
 $(C16 * exp(t))/2 + (C17 * exp(2 * t))/4 + (C18 * exp(3 * t))/4 ,$   
 $z =$   
 $C16 * exp(t) + C17 * exp(2 * t) + C18 * exp(3 * t) .$

- Find the solution of the following system of differential equations
- $x'(t) = x(t) + 2y(t) - z(t),$   
 $y'(t) = x(t) + z(t),$   
 $z'(t) = 4x(t) - 4y(t) + 5z(t).$  corresponding to the following conditions  
 $x(0)=1,$   
 $y(0)=2,$   
 $z(0)=3.$
- Moreover plot the solution curves using the plot command.

## Solution:

- The recipe for the solution is as under:
- »  $eqn1 = 'Dx = x + 2 * y - z';$   
 $eqn2 = 'Dy = x + z';$   
 $eqn3 = 'Dz = 4 * x - 4 * y + 5 * z';$   
 $initialcondns = 'x(0) = 1, y(0) = 2, z(0) = 3';$   
 $[x, y, z] = dsolve(eqn1, eqn2, eqn3, initialcondns)$   
 $x =$   
 $6 * \exp(2 * t) - (5 * \exp(3 * t))/2 - (5 * \exp(t))/2$   
 $y =$   
 $(5 * \exp(3 * t))/2 - 3 * \exp(2 * t) + (5 * \exp(t))/2$   
 $z =$   
 $10 * \exp(3 * t) - 12 * \exp(2 * t) + 5 * \exp(t)$

## Solution:

- » `t=linespace(0,.5,25);`
- » `t=linspace(0,.5,25);`
- » `xx=eval(vectorize(x));`
- » `yy=eval(vectorize(y));`
- » `zz=eval(vectorize(z));`
- » `plot(t,xx,t,yy,t,zz)`
- » `xlabel('x');`
- » `ylabel('y');`
- » `grid;`
- » `title('Graph of the solution curves of system of differential equations')`



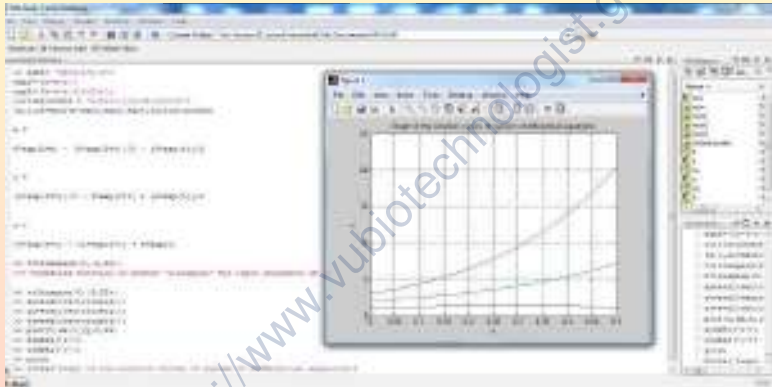


Figure: Graphical view of the solution curve of the system of differential equations.

- Find the solution of the following system of differential equations
- $x''(t) = x'(t) + 2y'(t) - z'(t),$   
 $y''(t) = x'(t) + z'(t),$   
 $z''(t) = 4x'(t) - 4y'(t) + 5z'(t).$   
corresponding to the following conditions  $x(0)=1, x'(0)=2$   
 $y(0)=1, y'(0)=4$   
 $z(0)=3, z'(0)=4.$
- Moreover plot the solution curves using the plot command.

## Solution:

- The recipe for the solution is as under:
- »  $eqn1 = 'D2x = Dx + 2 * Dy - Dz';$   
 $eqn2 = 'D2y = Dx + Dz';$   
 $eqn3 = 'D2z = 4 * Dx - 4 * Dy + 5 * Dz';$   
 »  $initialconditns = 'x(0) = 1, Dx(0) = 2, y(0) = 1, Dy(0) = 4, z(0) = 3, Dz(0) = 4';$   
 »  $[x, y, z] = dsolve(eqn1, eqn2, eqn3, initialconditns)$   
 $x =$   
 $7/3 - (exp(t) * (16 * exp(2 * t) - 72 * exp(t) + 72))/12$   
 $y =$   
 $(exp(t) * (32 * exp(2 * t) - 72 * exp(t) + 144))/24 - 10/3$   
 $z =$   
 $(exp(t) * (32 * exp(2 * t) - 72 * exp(t) + 72))/6 - 7/3$

## Solution:

```
» t=linespace(0,1,20);  
» xx=eval(vectorize(x));  
» yy=eval(vectorize(y));  
» zz=eval(vectorize(z));  
» plot(t,xx,t,yy,t,zz)  
» xlabel('x');  
» ylabel('y');  
» grid;  
» title('Graph of the solution curves of system of differential equations')
```

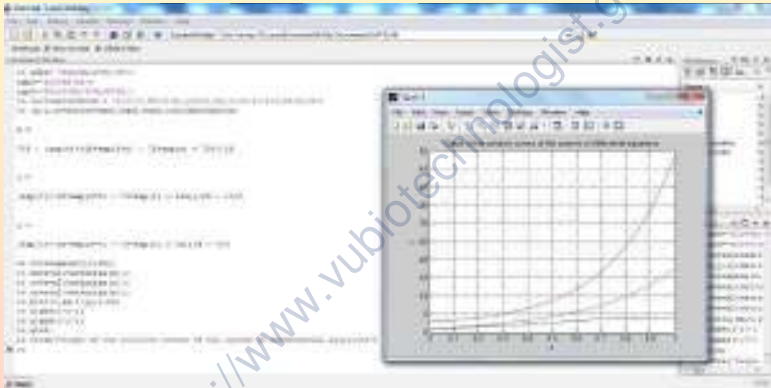


Figure: Graphical view of the solution curve of the system of differential equations.

- Find the general solution of the following differential equation

$$y'(x) = 2xy + 2x^3.$$

Also plot the family of solution curves on the interval  $[-4, 4]$  of the above differential equation obtained using the following initial conditions,  
 $y(0) = -3, -2, 0, 1, 2, 3$

- Find the general solution of the following boundary value problem  $y''(x) + 8y'(x) + 2y(x) = \cos(x)$ .  
Then plot the solution curve on the interval  $[0, 4]$  corresponding to following boundary conditions  $y(0)=4, y(1)=5$ .

- Find the solution of the following system of differential equations

- $x''(t) = 2x'(t) + 2y'(t) - z'(t),$

$$y''(t) = 3x'(t) - 3z'(t),$$

$$z''(t) = 5x'(t) - 4y'(t) + 5z'(t),$$

corresponding to the following conditions

$$x(0)=2, x'(0)=1$$

$$y(0)=3, y'(0)=4$$

$$z(0)=3, z'(0)=5.$$

- Moreover plot the solution curves using the plot command.



- Find the solution of the following system of differential equations

- $x'(t) = 2x(t) + 2y(t) - 3z(t),$

$$y'(t) = 2x(t) + 3z(t),$$

$$z'(t) = x(t) - y(t) + z(t),$$

corresponding to the following conditions

$$x(0)=2,$$

$$y(0)=1,$$

$$z(0)=4.$$

- Moreover plot the solution curves using the plot command.



# MATLAB coding of some numerical methods

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)



# Outline

- 1 Euler's Method
  - Application of the Method
- 2 Improved Euler Method
  - Application of the Method
- 3 Runge-Kutta Method
  - Application of the Method
- 4 Higher Order Taylor's Method
  - Taylor's method of second order
  - Application of the Method
- 5 Exercises



# Outline

- 1 Euler's Method
  - Application of the Method
- 2 Improved Euler Method
  - Application of the Method
- 3 Runge–Kutta Method
  - Application of the Method
- 4 Higher Order Taylor's Method
  - Taylor's method of second order
  - Application of the Method
- 5 Exercises



# Outline

- 1 Euler's Method
  - Application of the Method
- 2 Improved Euler Method
  - Application of the Method
- 3 Runge–Kutta Method
  - Application of the Method
- 4 Higher Order Taylor's Method
  - Taylor's method of second order
  - Application of the Method
- 5 Exercises



# Outline

- 1 Euler's Method
  - Application of the Method
- 2 Improved Euler Method
  - Application of the Method
- 3 Runge–Kutta Method
  - Application of the Method
- 4 Higher Order Taylor's Method
  - Taylor's method of second order
  - Application of the Method
- 5 Exercises



# Outline

- 1 Euler's Method
  - Application of the Method
- 2 Improved Euler Method
  - Application of the Method
- 3 Runge–Kutta Method
  - Application of the Method
- 4 Higher Order Taylor's Method
  - Taylor's method of second order
  - Application of the Method
- 5 Exercises



- Consider the general first order differential equation

$$\frac{dy}{dx} = f(x, y); \quad y(x_0) = y_0. \quad (1.1)$$

- Suppose we want to find its numerical solution on the interval of  $x$  - values  $[x_0, x_n]$ .
- Our goal will be to approximate the value of the solution  $y(x)$  at each of the  $x$  values in a partition  $P = [x_0, x_1, \dots, x_n]$ .
- To achieve this goal, the following formula is used:

$$y_{n+1} \approx y_n + f(x_n, y_n) \Delta x. \quad (1.2)$$

- Where  $\Delta x$  represents the length of each subinterval of equal length.





- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Euler's Formula for  $n = 10$

$$\frac{dy}{dx} = \sin(xy); \quad y(0) = \pi. \quad (1.3)$$



## Solution

- The recipe of the solution is as under:
- function [xvalues,yvalues]= euler(f,x0,y0,xn,n)

format long

n=10;

x0=0;

xn=1;

y0=pi;

x(1)=x0;

y(1)=y0;

xn=1;

dx=(xn-x0)/n;

for k=1:n

x(k+1)=x(k)+dx;

y(k+1)=y(k)+sin(x(k)\*y(k))\*dx;

end

xvalues=x';

yvalues=y'

plot(xvalues,yvalues)



- The coding of Euler's formula in Editor (Function M-File)

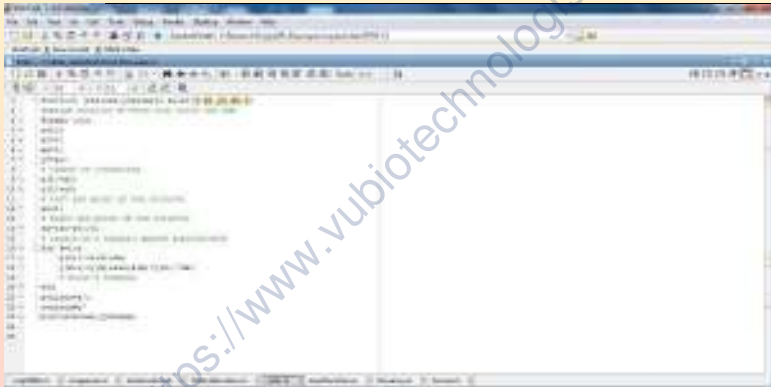


Figure: Coding of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Euler's Formula



- Output of the above coding shown in **Command Window**.



- Graphical view of the results of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .

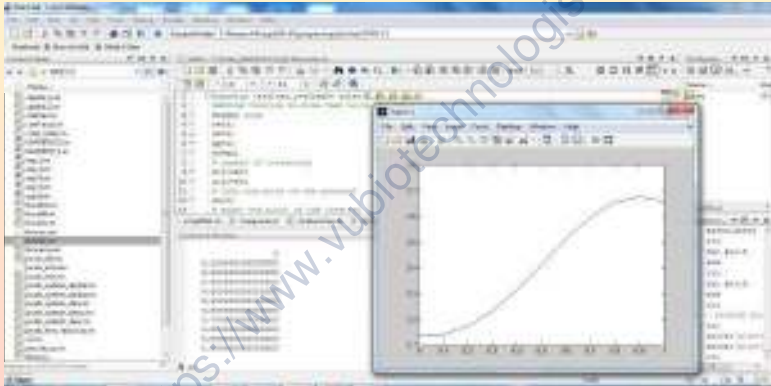


**Figure:** Graphical view along with the command window and function M-File of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .



- The general form of improved Euler's method is as under:

$$y_{k+1} = y_k + \frac{\Delta x}{2} (y'_k + f(x_{k+1}, y_k + \Delta x y'_k)), \quad i = 0, 1, 2, \dots, n-1. \quad (2.1)$$

- I will not go into the detail study of this method.
- The detail is mentioned in **MTH603**.



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using improved Euler's Formula for  $n = 10$

$$\frac{dy}{dx} = \sin(xy); \quad y(0) = \pi. \quad (2.2)$$



## Solution

- The recipe of the solution is as under:
- ```
function [xvalues,yvalues] = impeuler(f,x0,x1,xn,n)  
format long  
f = @(x,y)sin(x.*y);  
n=10;  
x0=0;  
y0=pi;  
xn=1;  
x(1)=x0;  
y(1)=y0;  
dx=(xn-x0)/n;  
for k=1:n x(k+1)=x(k)+dx;  
y(k+1)= y(k)+dx/2*(f(x(k),y(k))+f(x(k),y(k)+dx*y(k)));  
end  
xvalues=x'  
yvalues=y';  
plot(xvalues,yvalues)
```





- The coding of Improved Euler's formula in Editor (Function M-File)

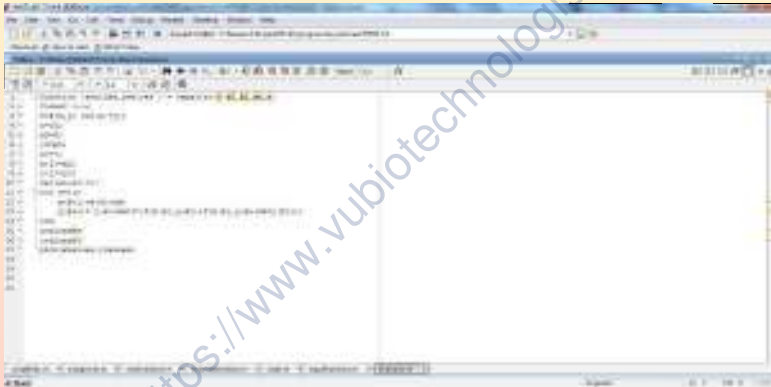


Figure: Coding of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Improved Euler's Formula



- Output of the above coding shown in **Command Window**.



**Figure:** Numerical solution (output) of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Improved Euler's Formula

- Graphical view of the results of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .

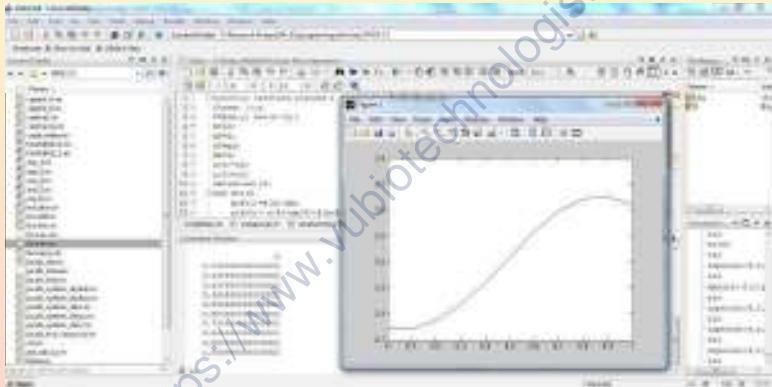


**Figure:** Graphical view along with the command window and function M-File of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .



- The recursive relation of fourth-order Runge-Kutta Method is as under:

$$y_{k+1} = y_k + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4),$$

where

$$\begin{aligned} K_1 &= f(x_k, y_k), \\ K_2 &= f\left(x_k + \frac{\Delta x}{2}, y_k + \frac{\Delta x}{2} K_1\right), \\ K_3 &= f\left(x_k + \frac{\Delta x}{2}, y_k + \frac{\Delta x}{2} K_2\right), \\ K_4 &= f(x_{k+1}, y_k + \Delta x K_3). \end{aligned} \tag{3.1}$$



### Example:

- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Runge-Kutta method of fourth order for  $n = 10$

$$\frac{dy}{dx} = \sin(xy); \quad y(0) = \pi. \quad (3.2)$$



## Solution:

- The coding of the solution is as under:
- `function [xvalues, yvalues] = RungeKutta(x0,xn,f,n,y0)`  
format long  
`f = @(x,y)sin(x.*y);`  
`n=10;`  
`x0=0;`  
`xn=1;`  
`y0=pi;`  
`x(1)=x0;`  
`y(1)=y0;`  
`dx=(xn-x0)/n;`  
for `k=1:n`  
`x(k+1)=x(k)+dx;`  
`k1=f(x(k),y(k));`



## Solution Continuing

### Solution:

```
k2=f(x(k)+dx/2,y(k)+dx/2*(k1));  
k3=f(x(k)+dx/2,y(k)+dx/2*(k2));  
k4=f(x(k+1),y(k)+dx*(k3));  
y(k+1)=y(k)+dx/6*(k1+2*k2+2*k3+k4);  
end  
xvalues =x';  
yvalues=y'  
plot(xvalues,yvalues)  
title('Runge Kutta method od order 4');  
xlabel('x');  
ylabel('y');
```



- The coding of Runge-Kutta method in Editor (Function M-File)

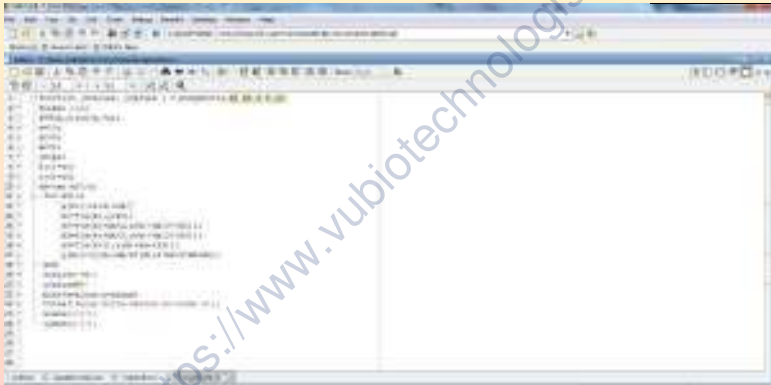


Figure: Coding of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Runge-Kutta Formula



- Output of the above coding shown in **Command Window**.



**Figure:** Numerical solution (output) of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Runge-Kutta Formula

- Graphical view of the results of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .

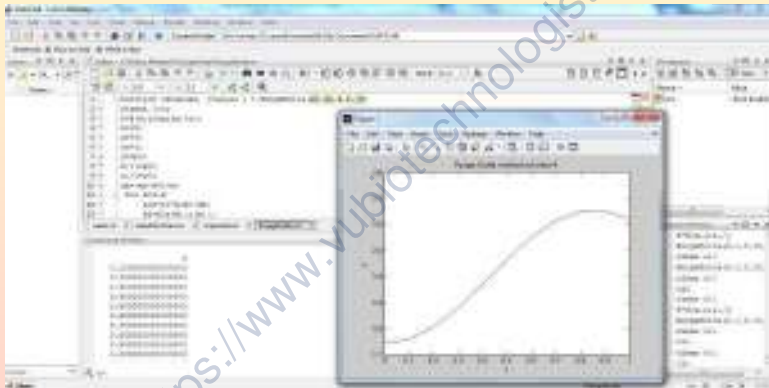


**Figure:** Graphical view along with the command window and function M-File of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .



- The basic idea of Euler's method can be improved in a straight forward manner by using higher order Taylor's polynomials.
- Remember that while deriving Euler's method, we approximate  $y(x)$  with a first order polynomial.
- More generally, if we use a Taylor polynomial of order  $n$ , we obtain the Taylor method of order  $n$ .
- For example, if we use a Taylor polynomial of order 2, we get a Taylor method of order 2.
- Similarly, the idea can be extended to higher order Taylor methods by increasing the order of Taylor polynomial.



- I will not go into the detail study of the second order Taylor's method.
- Simply, I will write down the iterative relation of the method, which is as under:

$$y_{k+1} \approx y_k + f(x_k, y_k) \Delta x + \left( \frac{\partial f}{\partial x}(x_k, y_k) + \frac{\partial f}{\partial y}(x_k, y_k) f(x_k, y_k) \right) \frac{(\Delta x)^2}{2} \quad (4.1)$$



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Taylor's method of the second order for  $n = 10$

$$\frac{dy}{dx} = \sin(xy); \quad y(0) = \pi. \quad (4.2)$$



## Solution:

- The MATLAB coding of the Method is given as under:
- `function [xvalues,yvalues] = Taylororder2(x0,xn,f,n,y0)`  
`format long`  
`n=10;`  
`f = @(x,y)sin(x.*y);`  
`x0=0;`  
`xn=1;`  
`y0=pi;`  
`x(1)=x0;`  
`y(1)=y0;`



## Solution:

```
• dx=(xn-x0)/(n);  
  for k=1:n  
    x(k+1)=x(k)+dx;  
    y(k+1) = y(k) + f(x(k), y(k)) * dx + (y(k) * cos(x(k) * y(k)) + x(k) *  
    cos(x(k) * y(k)) * f(x(k), y(k))) * (dx)^2/2;  
  end  
  xvalues=x';  
  yvalues=y'  
  plot(xvalues,yvalues)  
  xlabel('x');  
  ylabel('y'); title('Taylor's method of second Order.')
```

- The MATLAB coding of Taylor's method of second Order in Editor (Function M-File)

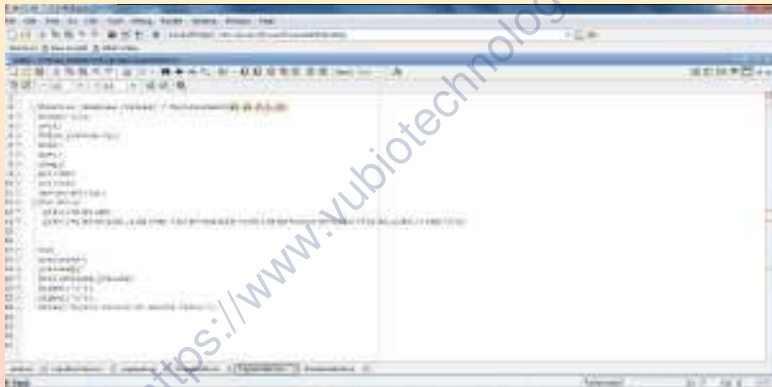


Figure: Coding of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Taylor's method of second Order.



- Output of the above coding shown in **Command Window**.



**Figure:** Numerical solution (output) of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$  using Taylor's method of second Order.

- Graphical view of the results of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .

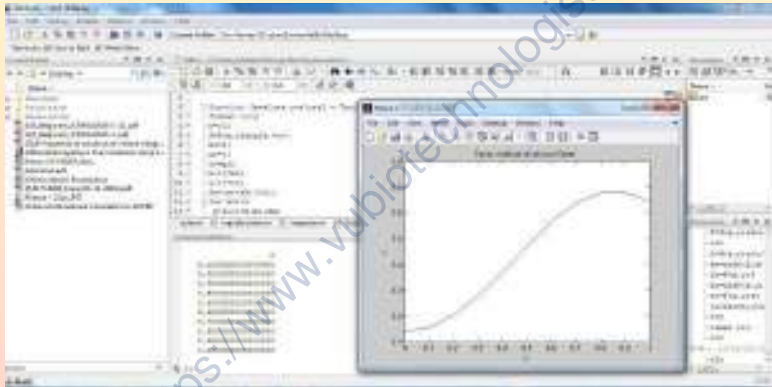


**Figure:** Graphical view along with the command window and function M-File of  $\frac{dy}{dx} = \sin(xy)$ ;  $y(0) = \pi$ .



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Euler's method for  $n = 20$

$$\frac{dy}{dx} = \frac{y - x}{y + x}; \quad y(0) = 0.5. \quad (5.1)$$



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Improved Euler's method for  $n = 20$

$$\frac{dy}{dx} = \frac{y - x}{y + x}; \quad y(0) = 0.5. \quad (5.2)$$



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Runge-Kutta method of fourth order for  $n = 20$

$$\frac{dy}{dx} = \frac{y - x}{y + x}; \quad y(0) = 0.5. \quad (5.3)$$



- Find the numerical solution of the following differential equation on the interval  $[0, 1]$  using Taylor's method of second order for  $n = 20$

$$\frac{dy}{dx} = \frac{y - x}{y + x}; \quad y(0) = 0.5. \quad (5.4)$$



# Numerical Solutions of ODEs in MATLAB

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

*imrantalib@vu.edu.pk*

# Outline

## 1 First-Order initial value problems

- Examples
- Family of approximate solutions
- Extraction of the numerical values at some particular domain values
- Comparison between exact and approximate solutions
- Limitations of the ode45 command
- ode45 command when initial condition point lies between the interval



# First-Order initial value problems (IVPs)

- 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.
- This command attempts to produce an approximate solution with error less than or equal to  $10^{-3}$ .
- This command produces the plot of the approximate solution by default.
- To make the picture more clear, consider the following example:



### Example1:

- Find the numerical plot of the following initial value problem on  $[0, 2]$ :

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

### Solution:

- Define the right hand side as an anonymous function, like that
- $f = @(x,y)y - x.^2 + 1.$
- The dot in front of the power sign indicates that the function must be able to operate on vectors.
- The recipe of the solution is given in the next slide.



- »  $f = @(x,y)y - x.^2 + 1$   
f =  
 $@(x,y)y - x.^2 + 1$   
» ode45(f,[0,2],0.5)  
» xlabel('x');  
» ylabel('y');  
» title('Graphical view of the numerical solution of  $y'(x) = y - x^2 + 1$ ')  
• The graphical view of the approximate solution is given in the next slide

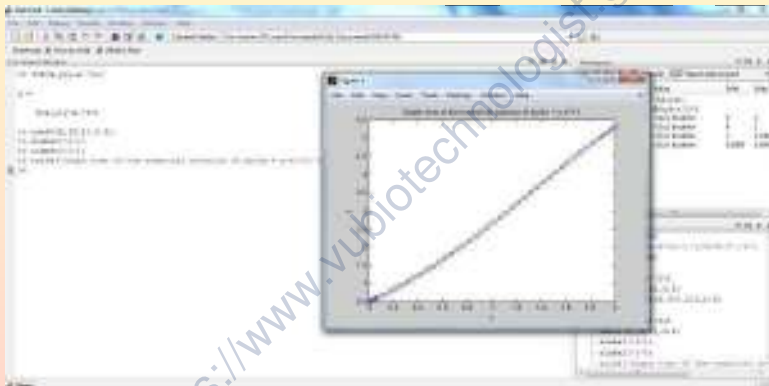


Figure: Graphical view of the numerical solution of  $y'(x) = y - x^2 + 1$

- Using **ode45** command, we can also plot the family of solutions of the above initial value problem.
- For this, we have to define the initial condition  $y(0)$  at different values. For example
- $y(0) = 0.5 : 0.2 : 3$ .
- That is we have to change the third argument (0.5) of the command **ode45(f,[0,2],0.5)** with 0.5:0.2:3.
- Study the following example:



## Example2:

- Plot the family of the approximate solutions of the following IVPs

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

## Solution:

- The recipe of the solution is as under:
- »  $f = @(x,y)y - x.^2 + 1$   
 $f =$   
 $@(x,y)y - x.^2 + 1$   
» `ode45(f,[0,2],0.5:0.2:3)`  
» `xlabel('x');`  
» `ylabel('y');`  
» `title('Graphical view of the family of the approximate solutions of  $y'(x) = y - x^2 + 1$ ')`
- See also the following figure.



Figure: Graphical view of the numerical solution of  $y'(x) = y - x^2 + 1'$



## Extraction of the numerical values at some particular domain values

- Up to now, we see the results of the approximate solutions in the form of graphs.
- But using **ode45** command, we can also extract the numerical values at some specific domain values.
- For this, we have to make some changes in the second argument of the command **ode45(f,[0,2],0.5)**.
- Study the next example.





### Example3:

- Compute the numerical values of the approximate solution of the following IVP at  $x = 0:0.2:2$

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

### Solution:

- The recipe of the solution is as under:

- $\gg f = @(x,y)y - x.^2 + 1$

$f =$

$@(x,y)y - x.^2 + 1$

$\gg [x,ya]=ode45(f,0:0.2:2,0.5)$

$x =$

0

0.2000000000000000

0.4000000000000000

0.6000000000000000

0.8000000000000000

1.0000000000000000

1.2000000000000000



## Solution:

- $x = 1.6000000000000000$

$$1.8000000000000000$$

$$2.0000000000000000$$

- $y_a =$

$$0.5000000000000000$$

$$0.829298806102213$$

$$1.214087852350423$$

$$1.648940818452941$$

$$2.127229773242783$$

$$2.640859343211129$$

$$3.179941816703143$$

$$3.732400315281681$$

$$4.283484106130412$$

$$4.815176603278762$$

$$5.305472351203669$$

## Solution:

- Note that, MATLAB stores the domain values in vector  $x$ .
- The corresponding approximate values are stored in vector  $ya$ .
- The graphical view corresponding to these specific domain values is given in the next slide.

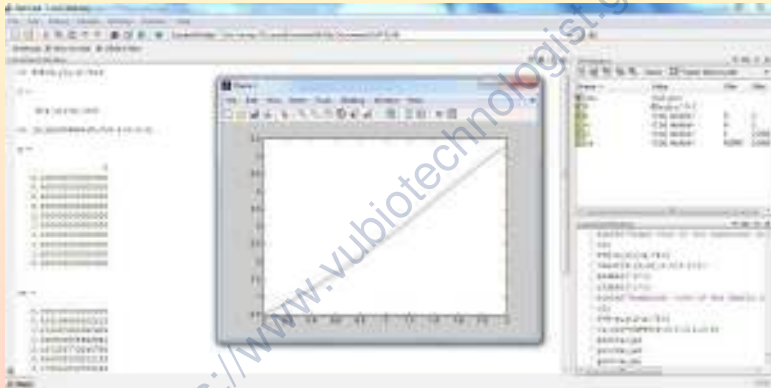


**Figure:** Graphical view of the numerical solution of  $y'(x) = y - x^2 + 1$  at some specific domain values

- We can also compare the exact and approximate solutions to observe the accuracy of the **ode45** command.
- For this purpose, we have to find the exact solution of the under-discussion differential problem.
- The methodology is discussed in Example 4.

#### Example4:

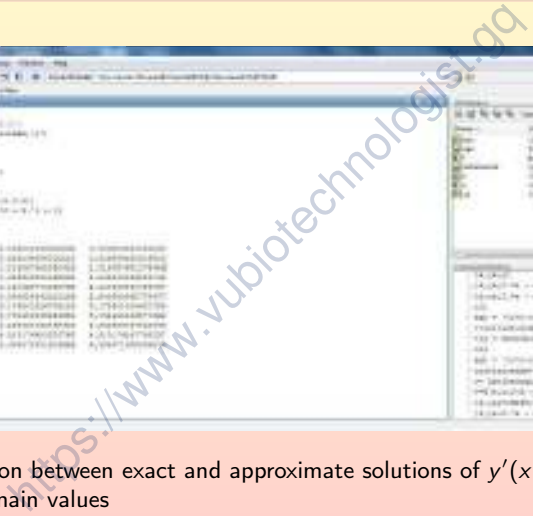
- Find the exact solution of the following IVP then compute its approximate solution at  $x = 0:0.2:2$  using **ode45** command. Finally, makes the table of the values of  $t$ ,  $ya$ , and  $y$ (exact solution).

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



## Solution:

- The recipe of the solution is as under:
- »  $\text{eqn} = 'Dy = y - x^2 + 1';$   
»  $\text{initialcondn} = 'y(0)=0.5';$   
»  $y = \text{dsolve}(\text{eqn}, \text{initialcondn}, 'x')$   
 $y =$   
 $2 * x - \exp(x)/2 + x^2 + 1$   
»  $f = @(x,y)y - x.^2 + 1;$   
»  $[x, ya] = \text{ode45}(f, 0:0.2:2, 0.5);$   
»  $[x, ya, 2.*x - \exp(x)./2 + x.^2 + 1]$
- The output is displayed in the next slide



on between exact and approximate solutions of  $y'(x) = y - x^2 + 1$  at main values

## Solution:

- Note that, MATLAB stores the domain values in vector  $x$ .
- The corresponding approximate values are stored in vector  $ya$ .
- And the corresponding values of the exact solution are stored in  $y$  (Third Column).
- It is to be noted that the approximate solution is correct to at least 4 digits in all cases





## Limitations of the ode45 command

- **ode45** command will only compute a solution on an interval with the initial condition is given at one of the end points.
- We have discussed the case when initial condition is given at the left end point in the above discussed examples.
- But when the initial condition is given at the right end point of the interval, then enter the values in decreasing order.
- For example, to plot the approximate solution of  $y'(x) = y - x^2 + 1$ ,  $y(0) = 0.5$  on the interval  $[-1, 0]$ , type the following syntax in the command
- `ode45(f,[0,-1],0.5).`



## Example5:

- Plot the approximate solutions of the following IVP at some specific domain elements,  $x = -0.2, -0.4, -0.6, -0.8, -1, 0$  in the interval  $[-1, 0]$  using **ode45** command.

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

## Solution:

- The recipe of the solution is as under:
- »  $f = @(x,y)y - x.^2 + 1;$   
» `ode45(f,0:-0.2:-1,0.5)`  
» `xlabel('x');`  
» `ylabel('y');`  
» `title('Graphical view of the concerned IVP, when initial condition is mentioned at the right end pont.')`
- The graphical view of the approximate solution is given in the next slide.



**Figure:** Graphical view of the concerned IVP, when initial condition is mentioned at the right end pont.

- Sometimes we are interested in finding the approximate solution of the IVP, when initial condition is defined at the end points of the interval but the end points of the interval are other than the initial point. Then in such situation, we subdivide the interval
- For example, if the initial condition is defined at 0, and the domain of interest is  $[-1, 2]$ , then the possible subdivision of the domain set is  $[-1, 2] = [-1, 0] \cup [0, 2]$ .
- Then we compute the approximate solution separately on each subinterval.
- The procedure is explained in the coming slide.

### Example6:

- Plot the family of the approximate solutions of the following IVP on the interval  $[-1, 2]$ . using **ode45** command.

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



## Solution:

- The recipe of the solution is as under:
- »  $f = @(x,y)y - x.^2 + 1;$   
» `[x,y]=ode45(f,[0,-1],1:0.2:3);`  
» `plot(x,y)`  
» hold on  
» `[x1,y1]=ode45(f,[0,2],1:0.2:3);`  
» `plot(x1,y1)`  
» hold off  
» `xlabel('x');`  
» `ylabel('y');`  
» `title('Graphical view of the approximate solution by subdividing the interval')`
- The graphical view is shown in the following slide

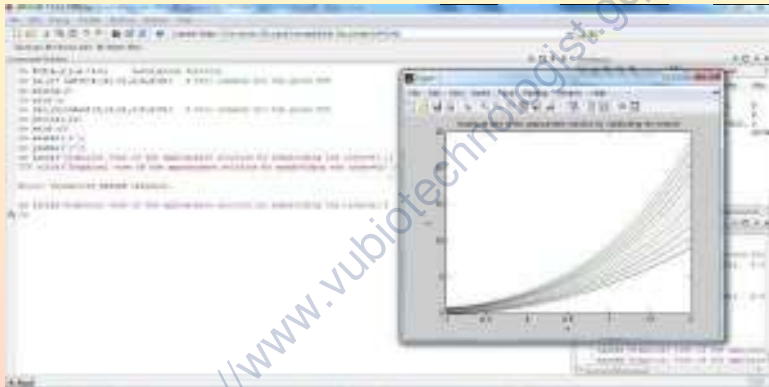


**Figure:** Graphical view of the family of approximate solutions by subdividing the interval.

$$\Gamma(\tau^2 + 1 + i)$$

# Matrix Algebra and MATLAB

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)



# 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

- 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

- 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

- 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

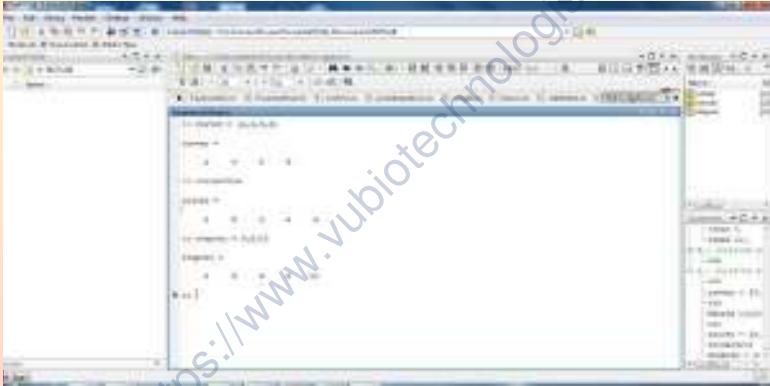


Figure: Display of the Row vector in different ways

# 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

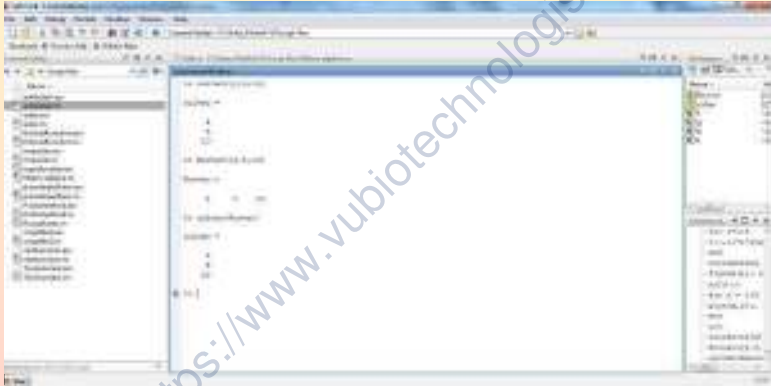


Figure: Display of the Column vector in different ways

- 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.
- There 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



- $A = \text{sym}([5/6, 1/6, 0; 5/6, 0, 1/6; 0, 5/6, 1/6])$

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

- 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:
- 
- $A = \text{sym}([5/6, 1/6, 0; 5/6, 0, 1/6; 0, 5/6, 1/6])$

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

$$\gg B = \text{sum}(A)$$

$$B =$$

$$\left( \frac{5}{3} \quad 1 \quad \frac{1}{3} \right)$$

## 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

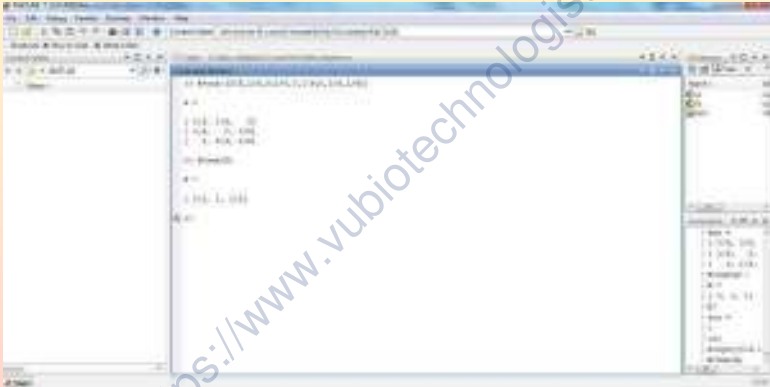


Figure: Display of the sum command

# Sum of row vectors of a matrix

```

» A =
    ( 5/6  1/6  0 )
    ( 5/6  0  1/6 )
    ( 5/6  0  1/6 )
    ( 0  5/6  5/6 )
    ( 0  5/6  0 )
    ( 0  1/6  1/6 )

» A' =
    ( 5/6  1/6  0 )
    ( 5/6  0  5/6 )
    ( 1/6  1/6  1/6 )

» B = sum(A')
B =
    [ 1, 1, 1 ]

» B'
ans =
    1
    1
    1.
    
```

- Sum of the entries of the row vectors of  $A$

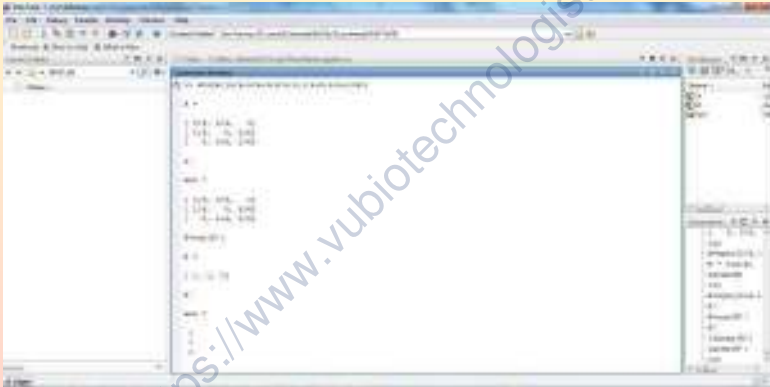


Figure: Display of the sum command

# 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,

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

- The MATLAB displays the following result:

- $\gg B = \text{diag}(A)$

$B =$

$\frac{5}{6}$

$0$

$\frac{1}{6}$



## 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

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

- The MATLAB displays the following result:

$$\gg B = \text{sum}(A(:, \text{end}))$$

$$B =$$

$$1/3$$

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

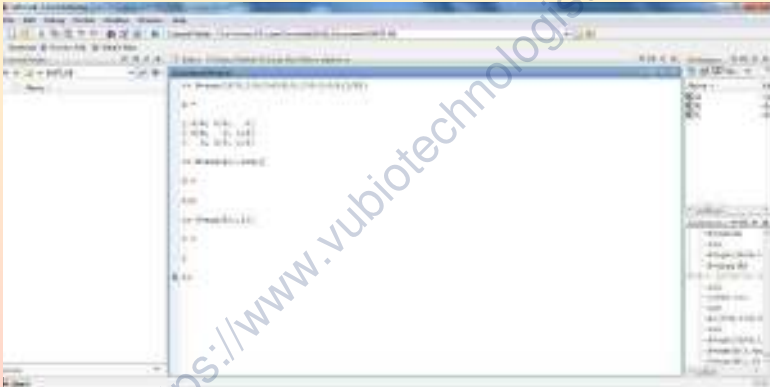


Figure: Display of the sum command with colon operator

# 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  $B$

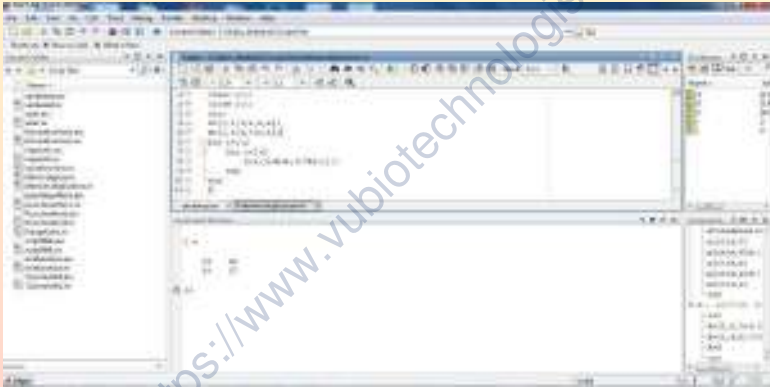


Figure: Matrices multiplication using for loop

# 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

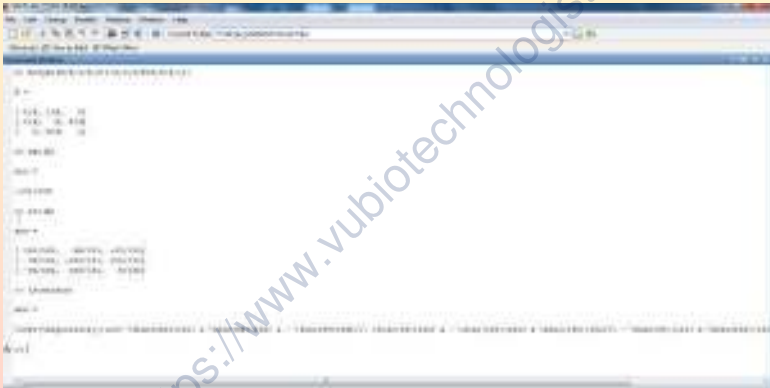
- The determinant of a square matrix in MATLAB can be determined using **det** command.
- For example, the determinant of the following matrix is determined as
- »  $A = \begin{pmatrix} \frac{5}{6} & \frac{1}{6} & 0 \\ \frac{5}{6} & 0 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}$
- » `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:

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



- Determinant and inverse of the matrix A.



```
A = [1 2 3; 4 5 6; 7 8 9];  
det(A)  
ans =  
0  
inv(A)  
ans =  
-1.0000e+001  3.0000e+000  -2.0000e+000  
  2.0000e+000 -1.0000e+000  1.0000e+000  
 -1.0000e+000  1.0000e+000 -1.0000e+000
```

Figure: det and inv command in MATLAB



Compute  $AB$  using for loop in MATLAB, if

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

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

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

- 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

- 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

- 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

- 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

- 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

- 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



- 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:
- $B = [0, 1, 3; 2, 4, 1; 6, 1, 8];$   
 $\text{eig}(B)$   
 $[P, D] = \text{eig}(B)$

**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 diagonal.

- Computation of the eigen values and eigen vectors.

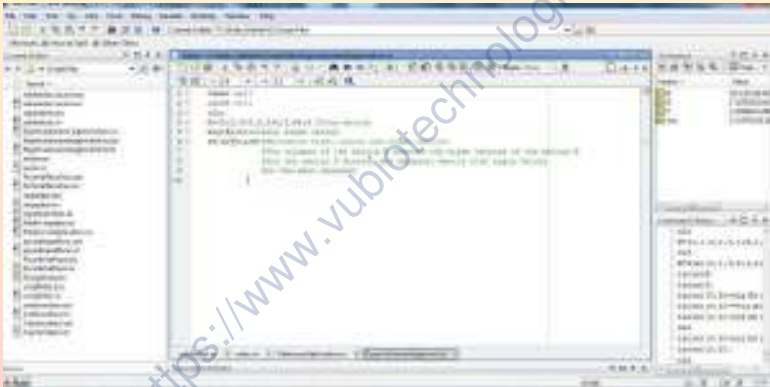


Figure: Editor file having code of eigen values and eigen vectors

- Results of the eigen values and eigen vectors.

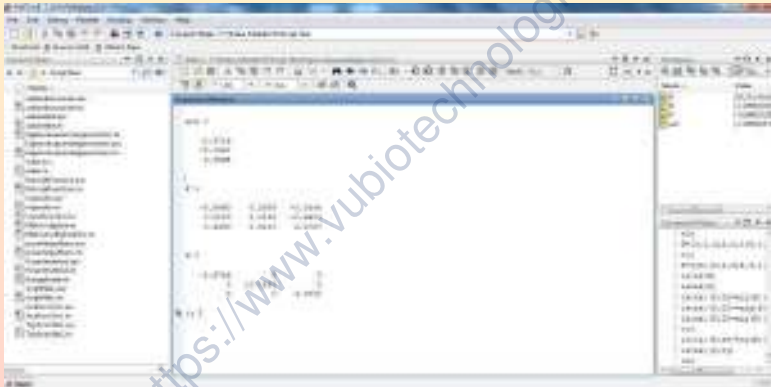


Figure: Command window having outputs of the eigen values and eigen vectors



- 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.

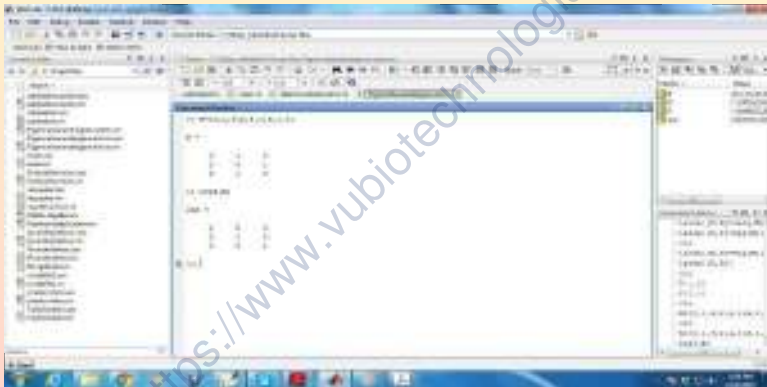


Figure: output of `rref` command





- 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:
- $A = \text{sym}([1,3;2,4]);$   
 $B = \text{sym}([-2;1]);$   
 $C = \text{inv}(A);$   
 $X = C * B$
- For more clarity see the next slide.

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

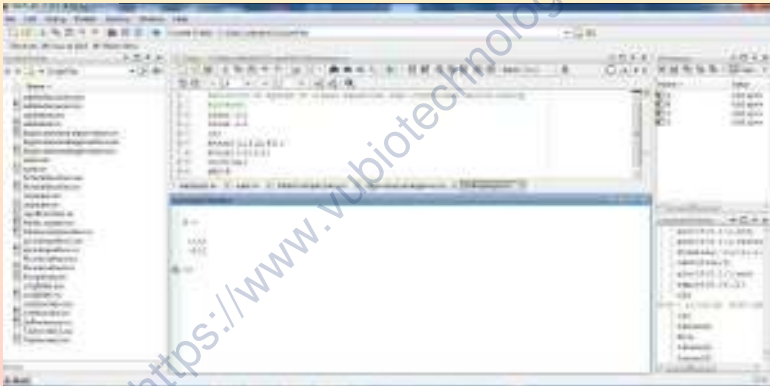


Figure: Display of the MATLAB Editor with code of the solution of linear equations



### 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

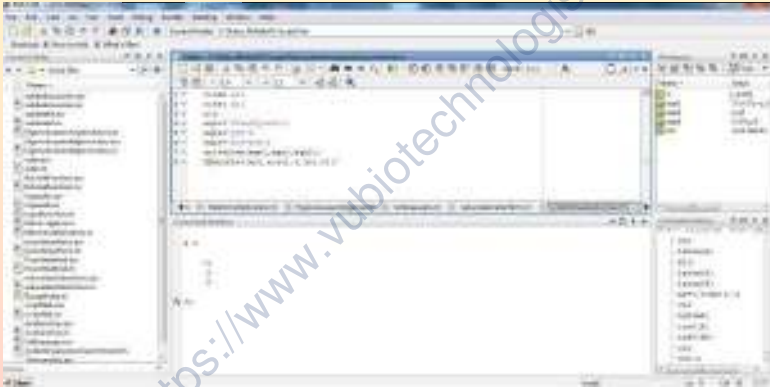


**Figure:** Display of the MATLAB Editor with code of the solution of linear equations using solve command

- 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

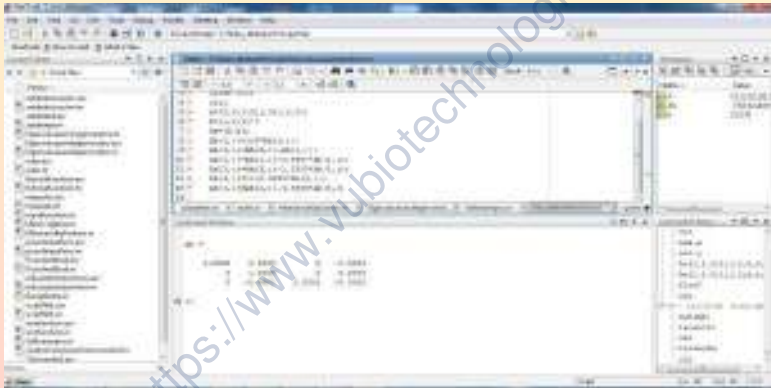


Figure: Reduction of the matrix into reduced echelon form



- **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  $\left( \begin{array}{c} \frac{b}{13} - \frac{6a}{13} \\ \frac{a}{13} + \frac{2b}{13} \end{array} \right)$

- Application of **rref** command

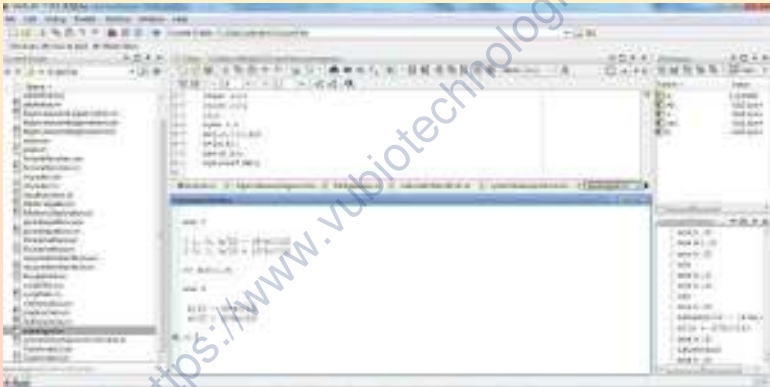


Figure: MATLAB code to determine the weights



- 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  $k$ .



# Numerical Methods for Integration

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)

# Outline

- 1 Riemann Sum
  - Brief introduction
  - Examples
  - Random selection of the points
- 2 Numerical Methods for Integration Part-I
  - Trapezoidal Rule
  - Simpson's one-Third Rule
  - Simpson's Three-Eight Rule
- 3 Exercise

# Outline

- 1 Riemann Sum
  - Brief introduction
  - Examples
  - Random selection of the points
- 2 Numerical Methods for Integration Part-I
  - Trapezoidal Rule
  - Simpson's one-Third Rule
  - Simpson's Three-Eight Rule
- 3 Exercise

# Outline

- 1 Riemann Sum
  - Brief introduction
  - Examples
  - Random selection of the points
- 2 Numerical Methods for Integration Part-I
  - Trapezoidal Rule
  - Simpson's one-Third Rule
  - Simpson's Three-Eight Rule
- 3 Exercise



- The MATLAB built-in-function **quad** is used to approximate definite integrals that could not be evaluated by Fundamental Theorem of Calculus.
- In order to understand the type of calculation carried out by **quad**, we will develop a variety of algorithms for approximating values of definite integrals.
- In this section we will introduce the Right End Point Method, Mid Point Method.
- First, we get the idea of Riemann sums

- Let  $f(x)$  be a continuous function on interval  $[a, b]$ , and suppose this interval is partitioned by the values  
 $a = x_0 < x_1 < x_2 < \dots < x_{n-1} < x_n = b$ .
- Any sum of the form

$$R = \sum_{k=1}^n f(c_k) \Delta x_k, \quad (1.1)$$

where  $\Delta x_k = x_k - x_{k-1}$ , and  $c_k \in [x_{k-1}, x_k]$  is referred to as a Riemann sum of  $f$ . Let

$$\|P\| := \max(\Delta x_1, \Delta x_2, \dots, \Delta x_n) \quad (1.2)$$

denote the norm of the partition, then we say  $f$  is Riemann integrable if

$$\lim_{\|P\| \rightarrow 0} \sum_{k=1}^n f(c_k) \Delta x_k \quad (1.3)$$

exists.

**Example:** Compute the Riemann sum of the function  $f(x) = e^{-x^2}$  for  $n=4,6,8,10,100$ , and  $1000$  by choosing  $c_K$  as the right end point of the interval  $[0,2]$ .

**Solution:**

- If we take a regular partition with  $n$  intervals, then each interval has length  $\Delta x = \frac{b-a}{n}$ .
- The  $k$ th end point is  $x_k = a + k\Delta x$ .
- The Riemann sum becomes

$$R = \sum_{k=1}^n f(c_k = a + k\Delta x) \Delta x_k, \quad (1.4)$$

- Particularly, for  $n = 4$ , it becomes

$$R = \sum_{k=1}^4 f(c_k = 0 + k0.5)0.5, \quad (1.5)$$



# MATLAB code

- The MATLAB code for the solution is as under:

- `function value=riemannsum1(f,a,b,n)`

```
clear all;
```

```
clc;
```

```
close all;
```

```
a=0;
```

```
b=2;
```

```
n=100;
```

```
value=0;
```

```
f=inline('exp(-x^2)');
```

```
dx=(b-a)/n;
```

```
for k=1:n
```

```
    c=a+k*dx;
```

```
    value=value+f(c);
```

```
end
```

```
value=dx*value
```

- Riemann sum when  $c_k$  is chosen as the right end point of each sub-interval.

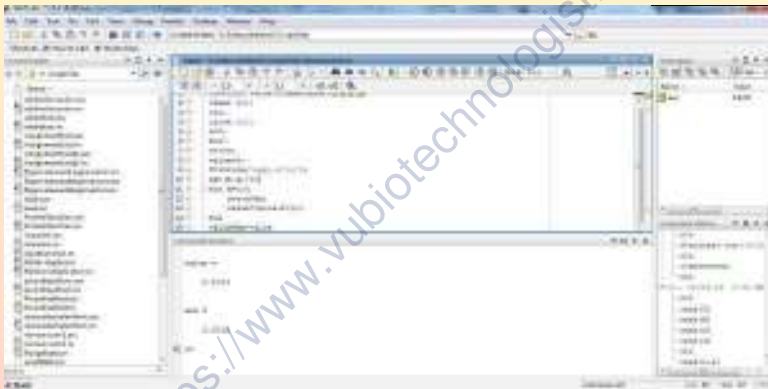


Figure: Riemann sum of the function  $e^{-x^2}$  on  $[0,2]$  for  $n=100$ .



- One interesting aspect of the Riemann sum is that the point  $c_k$  need not be chosen in the same place on each interval.
- For example, suppose we partition the interval  $[0, 1]$  with  $x_0 < x_1 = \frac{2}{3} < x_2 = \frac{4}{3} < x_3 = 2$ . If we choose  $c_1$  as the left point of the interval  $[0, 2/3]$ ,  $c_2$  as the right end point of the interval  $[2/3, 4/3]$ , and  $c_3$  as the left end point of the interval  $[4/3, 2]$ . Then the possible Riemann sum in this case is:

$$f(0)\frac{2}{3} + f(2/3)\frac{2}{3} + f(4/3)\frac{2}{3}. \quad (1.6)$$

**Example:** Compute the Riemann sum of the function  $f(x) = e^{-x^2}$  for  $n = 100$  by choosing  $c_K$  is randomly selected on the interval  $[0, 2]$ .

**Solution:**

- To choose randomly the points  $c_K$ , we know that the points  $c = (1 - r)a + rb$  moves from  $a$  to  $b$  as  $r$  goes from 0 to 1.
- Our generic interval is  $[a + (k - 1)\Delta x, a + k\Delta x]$  and **rand** denotes a value randomly chosen between 0 and 1.
- Therefore,  $c = (1 - \text{rand})(a + (k - 1)\Delta x) + \text{rand}(a + k\Delta x)$  will give the formula that we use to develop the MATLAB code for randomly selection of the points.

## Solution:

- The MATLAB code for the solution is as under:

- `function value=riemannsum2(f,a,b,n)`

```
clear all;
```

```
clc;
```

```
close all;
```

```
a=0;
```

```
b=2;
```

```
n=100;
```

```
value=0;
```

```
f=inline('exp(-x^2)');
```

```
dx=(b-a)/n;
```

```
for k=1:n
```

```
c=c+dx*rand+(a+(k-1)*dx);
```

```
value=value+f(c);
```

```
end
```

```
value=dx*value
```

- Riemann sum when the points  $c_k$  is selected randomly.

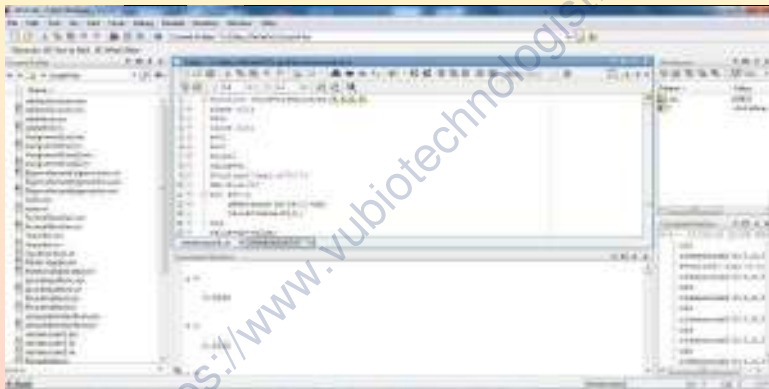


Figure: Riemann sum of the function  $e^{-x^2}$  on  $[0, 2]$  for  $n=100$ .

- The important observation for evaluation of the definite integrals using Riemann sum is that if a function is increasing then approximation by evaluation at right end points always overestimates the area, while approximation by evaluation at left end point always underestimates the area.
- Precisely the opposite is true for decreasing functions.
- So, in order to address this Trapezoidal Rule takes an average of these two approximations. That is

$$\begin{aligned}
 T_n &\approx \frac{\sum_{k=1}^n f(x_k)\Delta x_k + \sum_{k=1}^n f(x_{k-1})\Delta x_k}{2}, \\
 &\approx \frac{b-a}{n} \left( \frac{f(b) + f(a)}{2} + \sum_{k=1}^{n-1} f(x_k) \right), \quad \Delta x_k = \frac{b-a}{n}.
 \end{aligned} \tag{2.1}$$

**Example:** Compute the definite integral of  $f(x) = e^{-x^2}$  using Trapezoidal Rule considering hundred sub-intervals (that is,  $n = 100$ ) on the interval  $[0, 2]$ .

**Solution:**

- Here,  $\Delta x_k = \frac{b-a}{n} = 0.02$ .
- $\sum_{k=1}^{99} f(x_k) = f(1) + f(2) + \dots + f(99)$ .
- Then Trapezoidal approximation for this case becomes:
- 

$$T_n \approx \frac{b-a}{n} \left( \frac{f(b) + f(a)}{2} + \sum_{k=1}^{n-1} f(x_k) \right), \quad \Delta x_k = \frac{b-a}{n}, \quad (2.2)$$

$$T_{100} \approx \frac{2-0}{100} \left( \frac{f(2) + f(0)}{2} + f(1) + f(2) + \dots + f(99) \right).$$



- The MATLAB code for the solution is as under:

- `function value=trapezoidalrule(f,a,b,n)`

`clear all;`

`clc;`

`close all;`

`sum=0;`

`a=0;`

`b=2;`

`n=100;`

`f = inline('exp(-x^2)');`

`dx = (b - a)/n;`

`Averagevalue=(f(b)+f(a))/2;`

`for k=1:(n-1)`

`c = a + k * dx;`

`sum=sum+f(c);`

`end`

`Answer= (sum+Averagevalue)*dx`

- Trapezoidal approximation of definite integral.

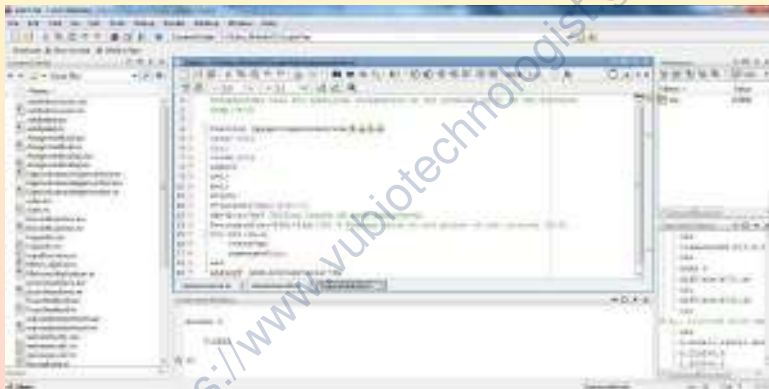


**Figure:** Evaluation of the definite integral of the function  $e^{-x^2}$  using Trapezoidal Rule on  $[0,2]$  for  $n=100$ .

- The idea behind Simpson's Rule is to proceed by approximating  $f(x)$  by quadratic polynomials rather than by linear one (lines).
- More generally, any order of polynomial can be used to approximate  $f$ , and higher orders give more accurate method.
- The general form of Simpson one-Third Rule is given as under:

$$\int_{x_0}^{x_{n+h}} f(x) dx \approx \frac{\Delta}{3} \left[ (y(x_0) + y(x_n)) + 4(y(x_1) + y(x_3) + \dots + y(x_{n-1})) + 2(y(x_2) + y(x_4) + \dots + y(x_{n-2})) \right]. \quad (2.3)$$

### Example:

- Compute the definite integral of  $f(x) = e^{-x^2}$  on the interval  $[0, 2]$  using Simpson's one-Third Rule by dividing the interval  $[0, 2]$  into 6 sub-intervals

### Note:

- While using Simpson one-Third Rule, the given interval of integration must be divided into an even number of sub-intervals, since we find the area over two sub-intervals at a time.

## Solution:

- For this case,  $n = 6$  (must be even).
- Calculate the width,  $\Delta x = \frac{2-0}{6}$ .
- Calculate the values of  $x_0$  to  $x_6$  as  
 $x_0 = 0, x_1 = 0 + \Delta x, x_2 = 0 + 2\Delta x, \dots, x_6 = 0 + 6\Delta x = 2$ .
- Now find the values of  $f(x) = e^{-x^2}$  from  $y(x_0)$  to  $y(x_6)$  for the corresponding  $x_0$  to  $x_6$ .
- Finally, substitute all the above found values in the Simpson one-Third Rule to calculate the approximate integral value.

# MATLAB code for the above example

- The MATLAB recipe for the Simpson's one-Third Rule is as under:

- `function simpson=Simpsononethird(f,a,b,n)`

`format long`

`clear all;`

`clc;`

`close all;`

`sum1=0;`

`sum2=0;`

`a=0;`

`b=2;`

`n=6;`

`f = inline('exp(-x^2)');`

`dx=(b-a)/n;`

`value=f(b)+f(a);`

## MATLAB code for the above example

```
for k=1:n-1
c=a+k*dx;
if mod(k,2)==0
sum1=sum1+2*f(c);
else
sum2=sum2+4*f(c);
end
end
sum=sum1+sum2;
Answer=(value+sum)*(dx/3)
```

- Simpson's one-Third Rule to approximation the area.

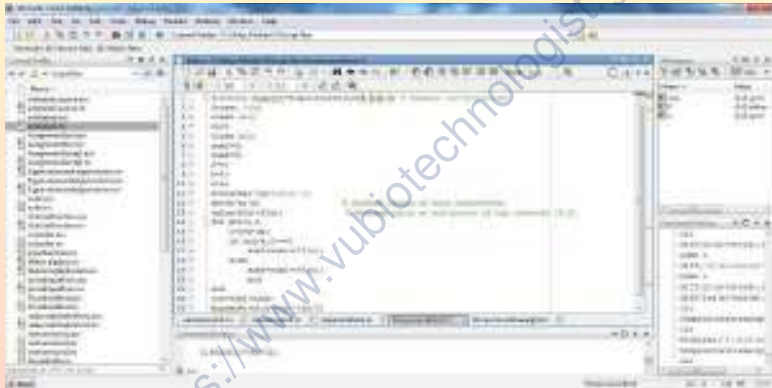


Figure: Evaluation of the definite integral of the function  $e^{-x^2}$  using Simpson's one-Third Rule on  $[0,2]$  for  $n=6$ .





- While using Simpson's Three-Eight Rule, the given interval of integration must be divided into sub-intervals whose number  $n$  is a multiple of 3.
- The general form of Simpson's Three-Eight Rule is given as under:

$$\int_{x_0}^{x_{n+h}} f(x) dx \approx \frac{3\Delta}{8} \left[ (y(x_0) + y(x_n)) + 3(y(x_1) + y(x_2) + \dots + y(x_{n-1})) + 2(y(x_3) + y(x_6) + \dots + y(x_{n-3})) \right]. \quad (2.4)$$

## Example:

- Compute the definite integral of  $f(x) = e^{-x^2}$  on the interval  $[0, 2]$  using Simpson's Three-Eight Rule by dividing the interval  $[0, 2]$  into 6 sub-intervals .

## Solution:

- For this case,  $n = 6$  (must be a multiple of 3).
- Calculate the width,  $\Delta x = \frac{2-0}{6}$ .
- Calculate the values of  $x_0$  to  $x_6$  as  
 $x_0 = 0, x_1 = 0 + \Delta x, x_2 = 0 + 2\Delta x, \dots, x_6 = 0 + 6\Delta x = 2$ .
- Now find the values of  $f(x) = e^{-x^2}$  from  $y(x_0)$  to  $y(x_6)$  for the corresponding  $x_0$  to  $x_6$ .
- Finally, substitute all the above found values in the Simpson's Three-Eight Rule to calculate the approximate integral value.

# MATLAB code for the above example

- The MATLAB recipe for the Simpson's Three-Eight Rule is as under:
- ```
function simpson38=Simpsononethreeeight(f,a,b,n)
format long
clear all;
clc;
close all;
sum1=0;
sum2=0;
a=0;
b=2;
n=6;
f = inline('exp(-x^2)');
dx=(b-a)/n;
value=f(b)+f(a);
```

## MATLAB code for the above example

```
for k=1:n-1
c=a+k*dx;
if mod(k,3)==0
sum1=sum1+2*f(c);
else
sum2=sum2+3*f(c);
end
end
sum=sum1+sum2;
Answer=(value+sum)*(3*dx/8)
```

- Simpson's Three-Eight Rule to approximation the area.

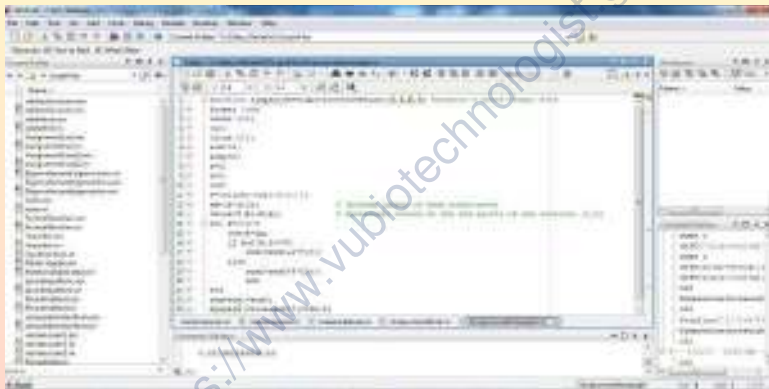


Figure: Evaluation of the definite integral of the function  $e^{-x^2}$  using Simpson's Three-Eight Rule on  $[0,2]$  for  $n=6$ .



- For the following integral

$$\int_1^5 \sqrt{1+x^3} dx. \quad (3.1)$$

- Compute the Riemann sum approximation using 100 equally spaced sub-intervals and left end points.
- Compute the Riemann sum approximation using 100 equally spaced sub-intervals and right end points.
- Compute the Riemann sum approximation using 100 equally spaced sub-intervals and mid points.

## Exercise

- Compute the following integral

$$\int_1^5 \sqrt{(1+x^3)} dx \quad (3.2)$$

- Using the Trapezoidal Rule considering 10 equally spaced sub-intervals.
- Using the Trapezoidal Rule considering 100 equally spaced sub-intervals.
- Using the Trapezoidal Rule considering 1000 equally spaced sub-intervals.

## Exercise

- Compute the following integral

$$\int_1^5 \sqrt{1+x^3} dx \quad (3.3)$$

- Using the Simpson's One-Third Rule considering 6 equally spaced sub-intervals.
- Using the Simpson's One-Third Rule considering 30 equally spaced sub-intervals.
- Using the Simpson's One-Third Rule considering 300 equally spaced sub-intervals.



## Exercise

- Compute the following integral

$$\int_1^5 \sqrt{1+x^3} dx \quad (3.4)$$

- Using the Simpson's Three-Eight Rule considering 6 equally spaced sub-intervals.
- Using the Simpson's Three-Eight Rule considering 30 equally spaced sub-intervals.
- Using the Simpson's Three-Eight Rule considering 300 equally spaced sub-intervals.



# Orthogonal Polynomials

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

*imrantalib@vu.edu.pk*



# Outline

## 1 Importance of orthogonal polynomials

- Orthogonal Functions
- Orthonormal Functions
- Orthogonal Polynomials
- Orthonormal Polynomials
- Properties of Orthogonal Polynomials

## 2 Some Important Orthogonal Polynomials

- Orthogonal Shifted Jacobi Polynomials
- Exercise
- Weight function
- Exercise
- Orthogonality Condition
- Exercise

## 3 Function Approximation using Jacobi Polynomials

- MATLAB code to determine the constant factor  $R_T^{(\alpha, \beta)}$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

- MATLAB code to determine the series coefficients  $C_j$ ,  $j = 0, 1, 2, \dots$



# Outline

## 1 Importance of orthogonal polynomials

- Orthogonal Functions
- Orthonormal Functions
- Orthogonal Polynomials
- Orthonormal Polynomials
- Properties of Orthogonal Polynomials

## 2 Some Important Orthogonal Polynomials

- Orthogonal Shifted Jacobi Polynomials
- Exercise
- Weight function
- Exercise
- Orthogonality Condition
- Exercise

## 3 Function Approximation using Jacobi Polynomials

- MATLAB code to determine the constant factor  $R_T^{(\alpha, \beta)}$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

- MATLAB code to determine the series coefficients  $C_j$ ,  $j = 0, 1, 2, \dots$



# Outline

## 1 Importance of orthogonal polynomials

- Orthogonal Functions
- Orthonormal Functions
- Orthogonal Polynomials
- Orthonormal Polynomials
- Properties of Orthogonal Polynomials

## 2 Some Important Orthogonal Polynomials

- Orthogonal Shifted Jacobi Polynomials
- Exercise
- Weight function
- Exercise
- Orthogonality Condition
- Exercise

## 3 Function Approximation using Jacobi Polynomials

- MATLAB code to determine the constant factor  $R_{T,j}^{(\alpha,\beta)}$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise
- MATLAB code to determine the series coefficients  $C_j$ ,  $j = 0, 1, 2, \dots$



# Outline

## 1 Importance of orthogonal polynomials

- Orthogonal Functions
- Orthonormal Functions
- Orthogonal Polynomials
- Orthonormal Polynomials
- Properties of Orthogonal Polynomials

## 2 Some Important Orthogonal Polynomials

- Orthogonal Shifted Jacobi Polynomials
- Exercise
- Weight function
- Exercise
- Orthogonality Condition
- Exercise

## 3 Function Approximation using Jacobi Polynomials

- MATLAB code to determine the constant factor  $R_{T,j}^{(\alpha,\beta)}$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise
- MATLAB code to determine the series coefficients  $C_j$ ,  $j = 0, 1, 2, \dots$



- Orthogonal polynomials are very much important in view of the development of the numerical schemes of certain types of differential equations.
- The numerical solution of many fractional and integer order ordinary and partial differential equations is based on the orthogonal polynomials. A number of research articles are available in the existing literature in which the numerical schemes have been developed using orthogonal polynomials.
- These polynomials exhibit the numerical solution in a series form.
- Frequently used orthogonal polynomials are Legendre polynomials, Jacobi polynomials, Bernstein polynomials, and Hermite polynomials.
- There we get the precise idea of these polynomials.



- Two functions  $f_i(x)$  and  $f_j(x)$  are orthogonal on an interval  $[a, b]$ , if

$$\int_a^b f_i(x)f_j(x)dx = 0 \quad (1.1)$$

- Mathematically, we say that the inner product of the functions  $f_i(x)$  and  $f_j(x)$  is zero.





- Two functions  $f_i(x)$  and  $f_j(x)$  are said to be orthonormal, if

$$\int_a^b f_i(x) f_j(x) dx = \delta_{ij}, \quad (1.2)$$

where

$$\begin{aligned} \delta_{ij} &= 1, & i &= j \\ &= 0, & i &\neq j \end{aligned} \quad (1.3)$$



- A set of orthogonal polynomials is an infinite sequence of polynomials,  $p_0(x), p_1(x), p_2(x), \dots, p_n(x), \dots$
- Where  $p_n(x)$  is a polynomial of degree  $n$ .
- Any two polynomials in the set are orthogonal to each other, that is,

$$\int_a^b p_i(x)p_j(x)dx = 0, \quad i \neq j \quad (1.4)$$



- The set of polynomials is orthonormal, if

$$\int_a^b p_i(x)p_j(x)dx = \delta_{ij}. \quad (1.5)$$

**Note:**

- The interval  $[a, b]$  is known as the interval of orthogonality and may be infinite at one or both ends.
- Any infinite sequence of polynomials  $p_n$ , with  $p_n$  having degree  $n$  forms a basis for the infinite-dimensional vector space of all polynomials.



- Any polynomial  $g(x)$  of degree  $n$  can be expressed as a linear combination of orthogonal polynomials:

$$g(x) = \sum_{i=0}^n c_i p_i(x). \quad (1.6)$$

- Where  $c_i$  is constant and can be determined easily.
- Given an orthogonal set of polynomials  $p_0(x), p_1(x), p_2(x), \dots, p_n(x), \dots$ , each polynomial,  $p_k(x)$ , is orthogonal to any polynomial of degree  $< k$ .
- Each polynomial in  $p_0(x), p_1(x), p_2(x), \dots, p_n(x), \dots$  has all  $n$  of its roots real distinct and strictly within the interval of orthogonality. That is not on its ends.



- Jacobi polynomials was first introduced by Carl Gustav Jacob Jacobi (1804–1851).
- These polynomials are solutions of the second order homogeneous differential equations of the form:

$$(1 - x^2)y''(z) + (\beta - \alpha - (\beta + \alpha + 2))y'(z) + n(n + \alpha + \beta + 1)y(z) = 0. \quad (2.1)$$

- The two parametric shifted Jacobi polynomials defined on  $[0, \tau]$  in one dimension is defined as:

$$P_n^{(\alpha, \beta)}(x) = \sum_{k=0}^i \frac{(-1)^{(i-k)} \Gamma(i + \beta + 1) \Gamma(i + k + \alpha + \beta + 1)}{\Gamma(k + \beta + 1) \Gamma(i + \alpha + \beta + 1) \Gamma(i - k + 1) \Gamma(k + 1)} \tau^k, \quad (2.2)$$

- where  $i = 0, 1, 2, \dots$



## Orthogonal Shifted Jacobi Polynomials Contd.

- The two parametric orthogonal shifted Jacobi polynomials defined in (2.2) is also known as the function vector in one dimension.

### Example:

- Construct the function vector  $P_n^{(\alpha, \beta)}(x)$  (Orthogonal Shifted Jacobi Polynomials) defined on  $[0, T]$  of degree 0, 1, 2, and 3 using equation (2.2).

### Solution:

- For  $i = 0$ ,  $k$  is also equal to zero, and we get  $P_0(x) = 1$ .
- For  $i = 1$ ,  $k$  varies from 0 to 1, and we get  $P_1(x) = \frac{2x}{T} - 1$ .
- For  $i = 2$ ,  $k$  varies from 0 to 2, and we get  $P_2(x) = \frac{6x^2}{T^2} - \frac{6x}{T} + 1$ .
- For  $i = 3$ ,  $k$  varies from 0 to 3, and we get  $P_3(x) = \frac{12x}{T} - \frac{30x^2}{T^2} + \frac{20x^3}{T^3} - 1$ .



## Solution Contd.

- In matrix form (vector form), the required polynomials can be written as:

$$\begin{pmatrix} P_0(x) \\ P_1(x) \\ P_2(x) \\ P_3(x) \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{2x}{T} - 1 \\ \frac{6x^2}{T^2} - \frac{6x}{T} + 1 \\ \frac{12x}{T} - \frac{30x^2}{T^2} + \frac{20x^3}{T^3} - 1 \end{pmatrix}$$



## Solution Contd.

- The MATLAB code for the construction of function vector,  $P_n^{(\alpha,\beta)}(x)$  (Orthogonal Shifted Jacobi Polynomials) is as under:
- ```
function P=Jacobfunvec(M,alpha,beta)
clc;
clear all;
close all;
syms x Tk
alpha=0;
beta=0;
M=4;
for i=0:M-1
    a1 = (-1)^(i - k);
    a2 = gamma(i + beta + 1);
    a3 = gamma(i + k + alpha + beta + 1);
```





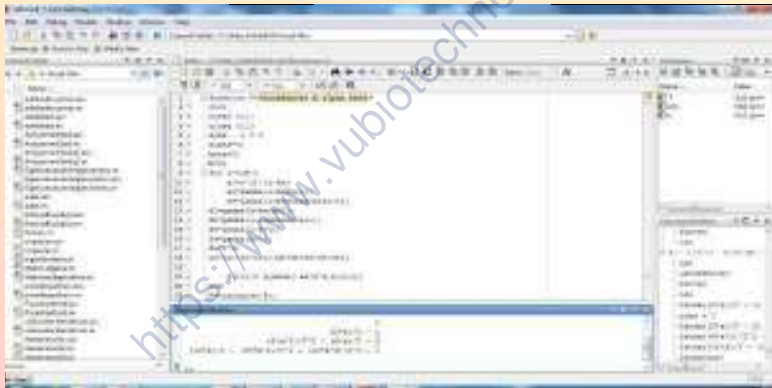
## Solution Contd.

```
d1 = gamma(k + beta + 1);  
d2 = gamma(i + alpha + beta + 1);  
d3 = gamma(i - k + 1);  
d4 = gamma(k + 1);  
d5 = Tk;  
aa = (a1 * a2 * a3)/(d1 * d2 * d3 * d4 * d5);  
P(i + 1) = symsum(aa * xk, k, 0, i);  
end  
P = transpose(P);
```



## Solution Contd.

- Function vector generated by using orthogonal shifted Jacobi polynomial,  $P_n^{(\alpha, \beta)}(x)$ .





- Construct the function vectors (Orthogonal Shifted Jacobi Polynomials,  $P_n^{(\alpha, \beta)}(x)$ ) defined in (2.2) on the interval  $[0, 1]$  by taking  $\alpha = 1$ ,  $\beta = 2$ , and scale level  $M = 10$ .

**Note:**

- Scale level  $M$  means, the parameter  $i$  varies from  $0 : 10$ . In this case, 10 function vectors of different degrees will be generated.



- The weight function  $W_T^{(\alpha,\beta)}(x)$  is defined as under:

$$W_T^{(\alpha,\beta)}(x) = (T-x)^\alpha x^\beta, \quad (2.3)$$

where  $\alpha$  and  $\beta$  are the parameters.

- The MATLAB code for defining the weight function is as follow:

- function `W=weightfuncjacob(x,alpha,beta,T)`

```
clc;
```

```
clear all;
```

```
close all;
```

```
syms T x alpha beta
```

```
W = xbeta * (T - x)alpha;
```



- Construct the weight function,  $W_T^{(\alpha, \beta)}(x)$  defined in (2.3) on the interval  $[0, 1]$  by taking  $\alpha = 1, \beta = 2$ .
- Construct the weight function,  $W_T^{(\alpha, \beta)}(x)$  defined in (2.3) on the interval  $[0, 1]$  by taking  $\alpha = 0, \beta = 0$ .



- The orthogonality condition for the Jacobian Polynomial with respect to weight function is as under:

$$\int_0^T P_{T,i}^{(\alpha,\beta)}(x) P_{T,j}^{(\alpha,\beta)}(x) W_{T,j}^{(\alpha,\beta)}(x) dx = R_{T,j}^{(\alpha,\beta)} \delta_{(i,j)}, \quad (2.4)$$

where

$$R_{T,j}^{(\alpha,\beta)} = \frac{T^{\alpha+\beta+1} \Gamma(j+\alpha+1) \Gamma(j+\beta+1)}{(2j+\alpha+\beta+1) \Gamma(j+1) \Gamma(j+\alpha+\beta+1)} \quad (2.5)$$



## Exercise:

- Check the orthogonality of orthogonal shifted Jacobi polynomial,  $P_n^{(\alpha, \beta)}(x)$  on  $[0, T]$ , by choosing  $\alpha = 1$ ,  $\beta = 1$  with respect to the weight function  $W_T^{(\alpha, \beta)}(x)$ .



- The orthogonality relation defined above allows us to approximate  $f(x) \in C([0, T])$  in terms of Jacobi series as follows:

$$f(x) = \sum_{i=0}^{\infty} C_i P_i^{(\alpha,\beta)}(x), \quad (3.1)$$

where  $C([0, T])$  is a space of continuous functions defined on  $[0, T]$ .





## Function Approximation using Jacobi Polynomials Contd.

- The series coefficients  $C_i$  can be easily determined using the orthogonality relationship of Jacobi Polynomials. That is

$$C_i = \frac{1}{R_{T,j}^{(\alpha,\beta)}} \int_0^T f(x) W_T^{(\alpha,\beta)}(x) P_i^{(\alpha,\beta)}(x) dx. \quad (3.2)$$



- The constant factor  $R_{T,j}^{(\alpha,\beta)}$  defined in (2.5) is computed in MATLAB as
- ```
function R=testcaseorthogonalitycheck(alpha,beta,T,M)
clc;
clear all;
close all;
syms i alpha beta T
M=4;
alpha=1;
beta=1;
T=1;
for i=0:M-1
a1 = T^(alpha+beta+1);
a2=gamma(i+alpha+1);
a3=gamma(i+beta+1);
```



# MATLAB code to determine the constant factor $R_{T,j}^{(\alpha,\beta)}$ contd.

```
d1=(2*i+alpha+beta+1);  
d2=gamma(i+1);  
d3=gamma(i+alpha+beta+1);  
R(i+1)=(a1*a2*a3)/(d1*d2*d3);  
end  
R=transpose(R)
```



- Constant factor involved in orthogonality condition of Jacobian Polynomial.

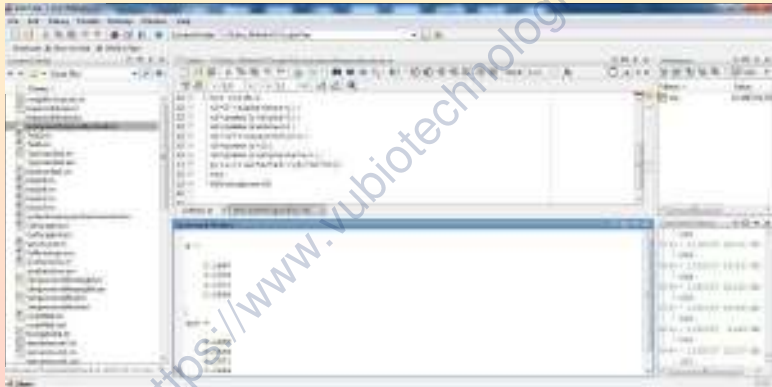


Figure: MATLAB code for generating the constant factor  $R_{T,j}^{(\alpha,\beta)}$ .



- Compute the constant factor defined in (2.5) for  $j = 0 : 5$  and for  $\alpha = 2$ , and  $\beta = 3$ .
- Compute the constant factor defined in (2.5) for choosing scale level  $M = 7$  and  $\alpha = 3$ , and  $\beta = 1$ .



- The series coefficients defined in (3.2) can be computed using MATLAB using the following code:
- ```
function C=coefvec(f,x,M,alpha,beta,T)
syms x f alpha beta T
alpha=1;
beta=1;
T=1;
M=4;
W=weightfuncjacob(x,alpha,beta,T);
P=Jacobfunvec(x,M,alpha,beta,T);
PW=f*P*W;
for i=0:M-1
R=orthogonal(i,alpha,beta,T);
aa=PW(i+1);
C(i+1)=(1/R)*int(aa,x,0,T);
end
C=transpose(C)
```



- Computation of the series coefficients  $C_i$ ,  $i = 0, 1, 2, 3, 4$  defined in (3.2).

The screenshot shows a MATLAB script in the editor and its corresponding Command Window output. The script defines a function `computeCoefficients` that calculates the series coefficients  $C_i$  for  $i = 0, 1, 2, 3, 4$ . The output in the Command Window displays the values of these coefficients.

```
function C = computeCoefficients(alpha, beta, n)
% Compute the series coefficients C_i for i = 0, 1, 2, 3, 4
% using the Jacobi polynomial basis.
% Inputs: alpha, beta (parameters of the Jacobi polynomial)
%         n (number of coefficients to compute)
% Output: C (vector of coefficients)

C = zeros(1, n);
for i = 0:n-1
    C(i+1) = ... % Calculation of C_i
end
```

Command Window Output:

```
C =
    1.0000    0.0000    0.0000    0.0000    0.0000
```

Figure: MATLAB code with generated output to compute the series coefficients  $C_i$ ,  $i = 0, 1, 2, 3, 4$ .



- Compute the series coefficients  $C_i$  of Jacobi Polynomials for  $i = 0 : 10$  and  $\alpha = 1, \beta = 2$ .
- Compute the series coefficients  $C_i$  of Jacobi Polynomials for scale level  $M = 6$  and  $\alpha = 2, \beta = 1$ .





# Orthogonal Polynomials II

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

[imrantalib@vu.edu.pk](mailto:imrantalib@vu.edu.pk)



# Outline

## 1 Orthogonal Legendre Polynomials

- Applications of the Legendre's differential equation
- Recurrence Relation to generate the Legendre Polynomials
- Some Legendre Polynomials
- MATLAB code to compute the Legendre Polynomials
- Exercise

## 2 Orthogonal Shifted Legendre Polynomials

- Example:
- Exercise:
- Orthogonality Condition
- MATLAB code to determine the constant factor  $R_j$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

## 3 Function Approximation using Legendre Polynomials

- MATLAB code to determine the series coefficients  $C_i$ ,  $i = 0, 1, 2, \dots$

## 4 Exercise



# Outline

## 1 Orthogonal Legendre Polynomials

- Applications of the Legendre's differential equation
- Recurrence Relation to generate the Legendre Polynomials
- Some Legendre Polynomials
- MATLAB code to compute the Legendre Polynomials
- Exercise

## 2 Orthogonal Shifted Legendre Polynomials

- Example:
- Exercise:
- Orthogonality Condition
- MATLAB code to determine the constant factor  $R_j$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

## 3 Function Approximation using Legendre Polynomials

- MATLAB code to determine the series coefficients  $C_i$ ,  $i = 0, 1, 2, \dots$

## 4 Exercise



# Outline

## 1 Orthogonal Legendre Polynomials

- Applications of the Legendre's differential equation
- Recurrence Relation to generate the Legendre Polynomials
- Some Legendre Polynomials
- MATLAB code to compute the Legendre Polynomials
- Exercise

## 2 Orthogonal Shifted Legendre Polynomials

- Example:
- Exercise:
- Orthogonality Condition
- MATLAB code to determine the constant factor  $R_j$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

## 3 Function Approximation using Legendre Polynomials

- MATLAB code to determine the series coefficients  $C_i$ ,  $i = 0, 1, 2, \dots$

## 4 Exercise



# Outline

## 1 Orthogonal Legendre Polynomials

- Applications of the Legendre's differential equation
- Recurrence Relation to generate the Legendre Polynomials
- Some Legendre Polynomials
- MATLAB code to compute the Legendre Polynomials
- Exercise

## 2 Orthogonal Shifted Legendre Polynomials

- Example:
- Exercise:
- Orthogonality Condition
- MATLAB code to determine the constant factor  $R_j$ ,  $j = 0, 1, 2, \dots, n, \dots$
- Exercise

## 3 Function Approximation using Legendre Polynomials

- MATLAB code to determine the series coefficients  $C_i$ ,  $i = 0, 1, 2, \dots$

## 4 Exercise

- These polynomials arise as the solutions of the following differential equation known as Legendre's differential equation:

$$(1 - x^2) \frac{\partial^2 P(x)}{\partial x^2} - 2x \frac{\partial P}{\partial x} + \lambda P(x), \quad \lambda = n(n + 1). \quad (1.1)$$

- These polynomials are orthogonal on the interval  $[-1, 1]$ .



- Legendre's differential equation is frequently encountered in Physics, particularly when spherical polar coordinates are used and where the problems have cylindrical symmetry.
- Moreover, Legendre polynomials form part of the quantum mechanical solution for rotational motion.



- The Legendre Polynomials defined on  $[-1, 1]$  are given by the following recurrence relation:

$$L_{i+1}(z) = \frac{2i+1}{i+1}zL_i(z) - \frac{i}{i+1}L_{i-1}(z), \quad \text{where } L_0(z) = 1, \quad L_1(z) = z \quad (1.2)$$

- $i = 1, 2, 3, \dots$





- Some Legendre Polynomials for different values of  $i$  are mentioned below:

$$P_0(z) = 1,$$

$$P_1(z) = z,$$

$$P_2(z) = \frac{1}{2}(3z^2 - 1),$$

$$P_3(z) = \frac{1}{2}(5z^3 - 3z),$$

$$P_4(z) = \frac{1}{8}(35z^4 - 30z^2 + 3),$$

and so on...

(1.3)



- The MATLAB code to compute the Legendre Polynomials defined in (1.3) using the relation (1.2) is given as under:
- ```
function P=Lagendrfucvector(M,T,value1,value2)
syms z
format short
M=2;
value(1)=z;
value(2)=sym((3/2)) * z^2 - sym(1/2);
for i=1:M
value(i+2)=((2 * i + 3)/(i + 2)) * z * value(i + 1) - ((i + 1)/(i + 2)) * value(i);
end
P=simplify(transpose(value));
```



- After execution of the above code, the following outputs are generated:

- $$\begin{pmatrix} P_1(z) \\ P_2(z) \\ P_3(z) \\ P_4(z) \end{pmatrix} = \begin{pmatrix} z \\ \frac{3z^2}{2} - \frac{1}{2} \\ z \left( \frac{5z^2 - 3}{2} \right) \\ \frac{35z^4}{8} - \frac{15z^2}{4} + \frac{3}{8} \end{pmatrix}$$

- Function vector generated by using orthogonal Legendre polynomial,  $P_n(x)$ .

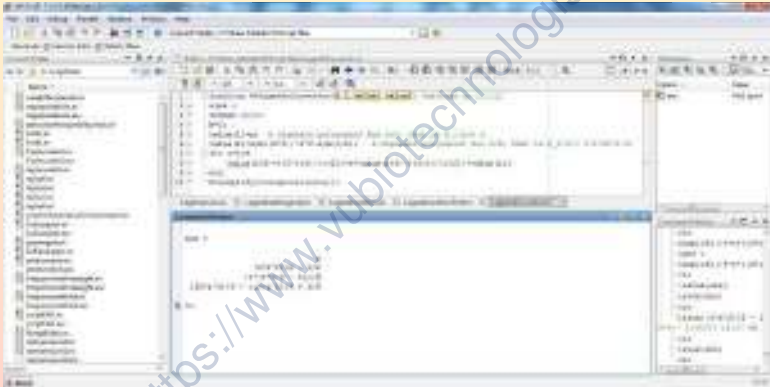


Figure: Orthogonal Legendre Polynomials generated by taking  $i = 0, 1, 2, 3$



- Determine the Legendre Polynomials using the relation (1.2) for  $i = 1 : 20$  using the above mentioned MATLAB code with slightly modification. That is we have to be determined  $P_3(z), P_4(z), P_5(z), \dots, P_{20}(z)$  using  $P_1(z) = z$ , and  $P_2(z) = \frac{3z^2}{2} - \frac{1}{2}$ .
- Moreover compute the values of the above Legendre Polynomials at  $z = 0, z = 1$ , and  $z = 5$ .



- The transformation  $x = \frac{(z+1)}{2}$  in relation (1.2) transforms the interval  $[-1, 1]$  to  $[0, 1]$ , then the Orthogonal shifted Legendre polynomials are given by

$$P_i(x) = \sum_{k=0}^i (-1)^{(i+k)} \frac{\Gamma(i+k+1)}{\Gamma(i-k+1)(\Gamma(k+1))^2}, \quad i = 0, 1, 2, \dots \quad (2.1)$$

- It can be easily checked that  $P_i(0) = (-1)^i$ , and  $P_i(1) = 1$ .



- Construct the function vector  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) defined on  $[0, T]$  of degree 0, 1, 2, and 3 using relation (2.1).

### Solution:

- For  $i = 0$ ,  $k$  is also equal to zero, and we get  $P_0(x) = 1$ .
- For  $i = 1$ ,  $k$  varies from 0 to 1, and we get  $P_1(x) = 2x - 1$ .
- For  $i = 2$ ,  $k$  varies from 0 to 2, and we get  $P_2(x) = 6x^2 - 6x + 1$ .
- For  $i = 3$ ,  $k$  varies from 0 to 3, and we get  $P_3(x) = 20x^3 - 30x^2 + 12x - 1$ .



## Solution Contd.

- In matrix form (vector form), the required polynomials can be written as:

$$\begin{pmatrix} P_0(x) \\ P_1(x) \\ P_2(x) \\ P_3(x) \end{pmatrix} = \begin{pmatrix} 1 \\ 2x - 1 \\ 6x^2 - 6x + 1 \\ 20x^3 - 30x^2 + 12x - 1 \end{pmatrix}$$





## Solution Contd.

- The MATLAB code for the construction of function vector,  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) is as under:

- function P=Lagdrfunvec(M,T)

```
syms x T k
```

```
M = 4;
```

```
for i = 0 : M - 1
```

```
    a1 = (-1)^(i+k);
```

```
    a2 = gamma(i + k + 1);
```

```
    d1 = gamma(i - k + 1);
```

```
    d2 = (gamma(k + 1))^2;
```

```
    aa = (a1 * a2)/(d1 * d2);
```

```
    P(i + 1) = symsum(aa * x^k, k, 0, i);
```

```
end
```

```
P = transpose(P);
```

## Solution Contd.

- Function vector generated by using orthogonal shifted Legendre polynomial,  $P_n(x)$ .

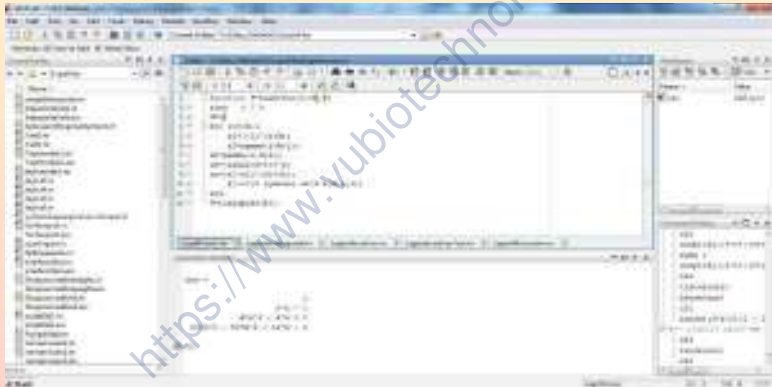


Figure: Orthogonal shifted Legendre Polynomials generated by taking  $i = 0, 1, 2, 3$   
visit our website to get more PDF <https://www.vubiotechnologist.gq>



- Construct the function vector  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) defined on  $[0, T]$  for  $i = 0 : 10$  using relation (2.1).
- Construct the function vector  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) defined on  $[0, T]$  for the scale level  $M = 15$  using relation (2.1).



- The orthogonality condition for the Legendre Polynomial is as under:

$$\int_0^1 P_i(x)P_j(x)dx = R_j\delta_{(i,j)}, \quad (2.2)$$

where

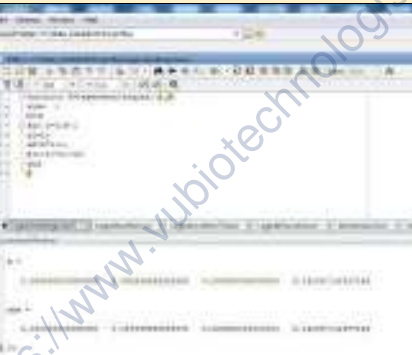
$$R_j = \frac{1}{2j+1} \quad (2.3)$$

and

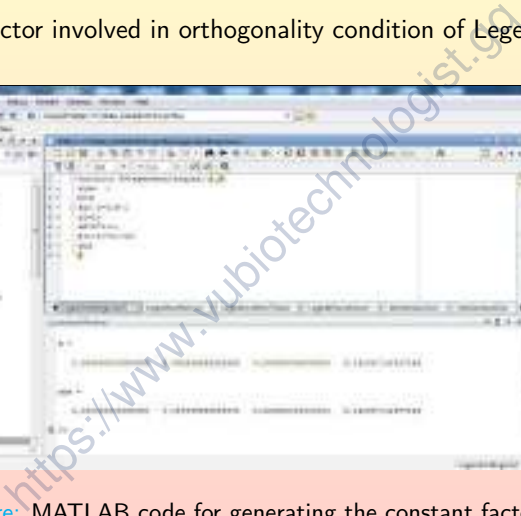
$$\begin{aligned} \delta_{ij} &= 1, & i &= j \\ &= 0, & i &\neq j \end{aligned} \quad (2.4)$$



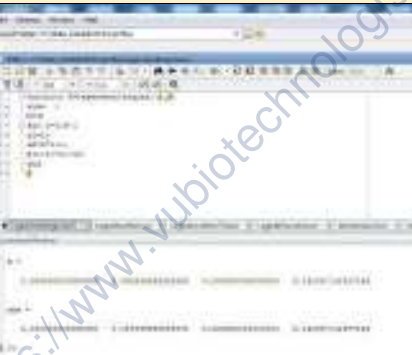
- The constant factor  $R_j$  defined in (2.5) is computed in MATLAB as
- ```
function R=Lagendrorthogonal(i,M)
syms i
M=4;
for i=0:M-1
a1=1;
a2=2*i+1;
R(i+1)=a1/a2;
end
R
```

- factor involved in orthogonality condition of Legendre polynomials
- 
- The image shows a MATLAB script in a code editor. The script defines a function `LegendrePolynomial` that takes two inputs: `n` (the order of the polynomial) and `xi` (the value of  $x$ ). The function uses a loop to calculate the Legendre polynomial  $P_n(x)$  using the recurrence relation. The code is as follows:
- ```
function P = LegendrePolynomial(n,xi)
% LegendrePolynomial: Calculate the Legendre polynomial P_n(x)
% Input: n (order of the polynomial), xi (value of x)
% Output: P (Legendre polynomial)

P = zeros(1,n+1);
P(1) = 1;
for k = 2:n+1
    P(k) = ((2*k-1)*xi*(P(k-1)) - (k-1)*P(k-2)) / k;
end
P = P(1:n+1);
```
- The script also includes a main section that calls the function for  $n=0$  to  $n=5$  and  $xi=0.5$ , and displays the results using `disp` and `format f`.
- Source: <https://www.vubiotecnologist.com>
- MATLAB code for generating the constant factor



factor involved in orthogonality condition of Legendre polynomials



The image shows a MATLAB script in a code editor. The script defines a function `LegendrePolynomial` that takes two inputs: `n` (the order of the polynomial) and `xi` (the value of  $x$ ). The function uses a loop to calculate the Legendre polynomial  $P_n(x)$  using the recurrence relation. The code is as follows:

```
function P = LegendrePolynomial(n,xi)
% LegendrePolynomial: Calculate the Legendre polynomial P_n(x)
% Input: n (order of the polynomial), xi (value of x)
% Output: P (Legendre polynomial)

P = zeros(1,n+1);
P(1) = 1;
for k = 2:n+1
    P(k) = ((2*k-1)*xi*(P(k-1)) - (k-1)*P(k-2)) / k;
end
P = P(1:n+1);
```

The script also includes a main section that calls the function for  $n=0$  to  $n=5$  and  $xi=0.5$ , and displays the results using `disp` and `format f`.

Source: <https://www.vubiotecnologist.com>

MATLAB code for generating the constant factor



- Compute the constant factor defined in (2.3) for  $j = 0 : 6$ .
- Compute the constant factor defined in (2.3) for choosing scale level  $M = 10$ .



- The orthogonality relation defined in (2.2) allows us to approximate  $f(x) \in C([0, 1])$  in terms of Legendre series as follows:

$$f(x) = \sum_{i=0}^{\infty} C_i P_i(x), \quad (3.1)$$

where  $C([0, 1])$  is a space of continuous functions defined on  $[0, 1]$ .





## Function Approximation using Legendre Polynomials Contd.

- The series coefficients  $C_i$  can be easily determined using the orthogonality relationship of Legendre Polynomials. That is

$$C_i = \frac{1}{R_j} \int_0^1 f(x) P_i(x) dx. \quad (3.2)$$



- The series coefficients defined in (3.2) can be computed in MATLAB using the following code:

- `function C=Lagendrcoefvec(M,T,f,i)`

```
syms x T i f
```

```
M=4;
```

```
P=Lagdrfunvec(M,T);
```

```
R=Lagendrorthogonal(i,M);
```

```
Kw=f*P;
```

```
for i=0:M-1
```

```
aa=Kw(i+1);
```

```
C(i+1)=(1/(R(i+1)))*int(aa,x,0,T);
```

```
end
```

```
C=transpose(C);
```



- Computation of the series coefficients  $C_i$ ,  $i = 0, 1, 2, 3, 4$  defined in (3.2).

The image shows a MATLAB script in the Editor window and its corresponding Command Window output. The script defines a function `LegendreCoefficients` that calculates the coefficients  $C_i$  for  $i = 0, 1, 2, 3, 4$ . The output in the Command Window shows the calculated values for each coefficient.

```
function C = LegendreCoefficients(f, N)
% LegendreCoefficients: Compute the series coefficients C_i for i = 0, 1, 2, ..., N
% Input: f - function handle or symbolic expression
% Input: N - number of coefficients to compute
% Output: C - vector of coefficients C_0, C_1, ..., C_N

syms x
L = legendre(N, x);
C = zeros(1, N+1);
for i = 0:N
    C(i+1) = (2*i+1)/2 * integral(f(x)*L(i,x), -1, 1);
end
```

Command Window Output:

```
C =  
1.0000  
0.5000  
-0.5000  
0.0000  
0.0000
```

Figure: MATLAB code with generated output to compute the series coefficients  $C_i$ ,  $i = 0, 1, 2, 3, 4$ .



- Compute the series coefficients  $C_i$  for  $i = 0 : 10$  using relation (3.2).
- Compute the series coefficients  $C_i$  for scale level  $M = 6$  and using relation (3.2) .



## Exercise

- Determine the Legendre Polynomials using the relation (1.2) for  $i = 1 : 10$ . That is we have to be determined  $P_3(z)$ ,  $P_4(z)$ ,  $P_5(z)$ , ...  $P_{10}(z)$  using  $P_1(z) = z$ , and  $P_2(z) = \frac{3z^2}{2} - \frac{1}{2}$ .
- Moreover compute the values of the above Legendre Polynomials at  $z = 3$ ,  $z = 4$ , and  $z = 6$ .



## Exercise

- Construct the function vector  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) defined on  $[0, 1]$  for  $i = 0 : 20$  using relation (2.1). Moreover compute the values of  $P_n(x)$  at  $x = 2$ .
- Construct the function vector  $P_n(x)$  (Orthogonal Shifted Legendre Polynomials) defined on  $[0, 1]$  for the scale level  $M = 30$  using relation (2.1). Moreover compute the values of  $P_n(x)$  at  $x = 0$ .



## Exercise

- Compute the constant factor defined in (2.3) for  $j = 0 : 10$  .
- Compute the constant factor defined in (2.3) for choosing scale level  $M = 40$ .