SWAMI DAYANANDA COLLEGE OF ARTS & SCIENCE, MANJAKKUDI.

(Affiliated to Bharathidasan University, Tiruchirappalli)

UGC Recognized under section 2(f) & 12(B)

DEPARTMENT OF MATHEMATICS

16SCCMM9:

NUMERICAL METHODS WITH MATLAB PROGRAMMING

CLASS:

III– B.Sc., MATHEMATICS

Prepared by:

Dr. S. VINOTH KUMAR,

M.Sc., M.Phil., Ph.D., M.Ed., B.Ed., (Special Education)

Assistant Professor of Mathematics

Unit-1 (Predefined Function)

- Arithmetic expressions often require computations other than addition, subtraction, multiplication, division and exponentiation.
- Many expressions require the use of logarithms, exponentials, and trigonometric functions. MATLAB includes a built-in library of these useful functions.
- For example, if we want to compute the square root of x and store the result in b, we can use the following command:

```
b = sqrt(x);
```

this statement is valid if x is a scalar or a matrix.

Unit-1 (Predefined Function)

- If x is a matrix, the function will be applied element by element to the values in the matrix.

for example,

```
x = 9;
b = sqrt(x)
```

returns

```
b =
    3
```

but if x is a matrix.

```
x = [4,    9,    16]
b = sqrt(x)
```

returns.

```
b =
     2    3     4
```

Unit-1 (Predefined Function)

- All functions can be thought of as having three components; a name, input and output.
- Some functions require multiple inputs. For example, the remainder function requires two inputs – a dividend and a divisor.

  for example,

  ```
  rem(10,3)
  ```

  returns

  ```
  ans =
      1   (calculates the remainder of 10 divided by 3)
  ```

The size command is an example of a function that returns two outputs. The size command determines the number of rows and columns in a matrix. Thus,

Unit-1 (Predefined Function)

```
d = [1,    2,    3; 4,    5,    6];
f = size(d)
```

returns

```
f =

    2    3
```

- When one function is used to compute the argument of another function. Be sure to enclose the argument of each function in its own set of parentheses. The nesting of functions is also called composition of functions.
- MATLAB includes extensive help tools. Which are especially useful for interpreting function syntax. A command line help function (help), a separate windowed help function (helpwin).

Unit-1 (Predefined Function)

- To use the command line help function, type help in the command window:

    help

  A list of help topics will appear, as follows:

    HELP topics:

    matlab\general – general purpose commands

    matlab\ops - operations and special characters

    matlab\lang - programming language constructs

    matlab\elmat - elementary matrices and matrix manipulation

    matlab\elfun - elementary math functions

    matlab\specfun – specialized math functions

- To get help on a particular topic,

    type 'help <topic>'.

Unit-1 (Elementary Math Functions)

Math function	Meaning	Example	Result
abs(x)	Computes the absolute value of x	abs(-3)	ans = 3
sqrt(x)	Computes the square root of x	sqrt(85)	ans = 9.2195
round(x)	Rounds x to the nearest integer	round(8.6)	ans = 9
fix(x)	Rounds x to the nearest integer toward zero	fix(8.6)	ans = 8
floor(x)	Rounds x to the nearest integer toward -∞	floor(-8.6)	ans = -9
ceil(x)	Rounds x to the nearest integer toward +∞	ceil(-8.6)	ans = -8
sign(x)	Returns a value of -1 if x is less than zero, a value of 0 if x equals zero, and a value of +1 if x is greater than zero.	sign(-8)	ans = -1
rem(x,y)	Computes the remainder of x/y	rem(25,4)	ans = 1
exp(x)	Computes the value of e^x, where e is the base for natural logarithms or 2.7182 (app)	exp(10)	ans = 2.2026e+004
log(x)	Computes the In(x), the natural logarithm of x to the base e	log(10)	ans = 2.3026
log10(x)	Computes the $\log_{10} x$, the common logarithm of x to the base 10	log10(10)	ans = 1

Unit-1 (Trigonometric Functions)

- The trigonometric functions assume that angles are represented to radians.

  To convert radians to degrees or degrees to radians,

  angle_degrees = angle_radians*(180/pi);

  angle_radians = angle_degrees*(pi/180);

In trigonometric calculations, the value of π is often needed, so a constant, pi is built into MATLAB.

MATLAB Command	Meaning (Where x is radian measure)	Example	Result
sin(x)	Computes the sine of x	sin(0)	ans = 0
cos(x)	Computes the cosine of x	cos(0)	ans = 1
tan(x)	Computes the tangent of x	tan(pi)	ans = -1.2246e-016
asin(x)	Computes the arcsine of x (or inverse)	asin(-1)	ans = -1.5708
sinh(x)	Computes the hyperbolic sine of x	sinh(pi)	ans = 11.5487

Unit-1 (Data Analysis Functions)

- Analysing data is an important part of evaluating test results. MATLAB contains a number of functions that make it easier to evaluate and analyze data.
- Simple Analysis

Maximum and Minimum:

This is of functions can be used to determine maximum and minimum.

1. The command

```
max(x)
```

returns the largest value in a vector x.

for example, if x = [1 5 3], the maximum value is 5.

```
x = [1,    5,    3]
max(x)
ans =
    5
```

returns a row vector containing the maximum element from each column of a matrix.

for example, if $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the maximum value in column 1 is 2, in column 2 is 5 and in column 3 is 6.

```
x = [1,   5,    3;2,    4,    6]
max(x)
ans =
   2    5    6
```

2. The command

```
[a,b] = max(x)
```

returns both the largest value in a vector x and its location in vector x. for x = [1   5   3], the maximum value is named a and is found to be 5. the location of the maximum value is element 2

and is named b.

```
x = [1,    5,    3];
[a,b] = max(x)
a =
   5
b =
   2
```

Returns a row vector containing the maximum element from each column of a matrix x, and returns a row vector of the location of the maximum in each column of matrix x.

for example, if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the maximum value in column 1 is 2, in column 2 is 5, and in column 3 is 6. These maxima occur in row 2, row 1, and row 2 respectively.

```
x = [1,    5,    3;2,    4,    6];
[a,b] = max(x)
```

Unit-1 (Data Analysis Functions)

a =

2 5 6

b =

2 1 2

3. The command, max(x,y)

returns a matrix the same size as x and y. Each element in the resulting matrix contains the maximum value from the corresponding positions in x and y.

for example if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, y = $\begin{bmatrix} 10 & 2 & 4 \\ 1 & 8 & 7 \end{bmatrix}$, the resulting matrix will be

ans = $\begin{bmatrix} 10 & 5 & 4 \\ 2 & 8 & 7 \end{bmatrix}$

```
x = [1,   5,    3;2,    4,    6];
y = [10,  2,    4;1,    8,    7];
max(x,y)
ans =
    10    5    4
     2    8    7
```

Unit-1 (Data Analysis Functions)

4. The command

```
min(x)
```

returns the smallest value in a vector x.

for example, if x = [1 5 3], the minimum value is 1.

```
x = [1,    5,    3]
min(x)
ans =
    1
```

returns a row vector containing the minimum element from each column of a matrix. for example, if $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the minimum value in column 1 is 1, in column 2 is 4 and in column 3 is 3.

```
x = [1,   5,    3;2,    4,    6]
min(x)
ans =
    1    4    3
```

Unit-1 (Data Analysis Functions)

5. The command

[a,b] = min(x)

returns both the smallest value in a vector x and its location in vector x. for x = [1 5 3], the minimum value is named a and is found to be 1. the location of the minimum value is element 1 and is named b.

```
x = [1,   5,   3];
[a,b] = min(x)
a =
    1
b =
    1
```

returns a row vector containing the minimum element from each column of a matrix x, and returns a row vector of the location of the minimum in each column of matrix x.

Unit-1 (Data Analysis Functions)

for example, if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the minimum value in column 1 is 1, in column 2 is 4, and in column 3 is 3. These minima occur in row 1, row 2, and row 1 respectively.

```
x = [1,    5,    3;2,    4,    6];
[a,b] = min(x)
a =
   1   4   3
b =
   1   2   1
```

6. The command, min(x,y)

returns a matrix the same size as x and y. Each element in the resulting matrix contains the minimum value from the corresponding positions in x and y. for example if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, y = $\begin{bmatrix} 10 & 2 & 4 \\ 1 & 8 & 7 \end{bmatrix}$, the resulting matrix will be ans = $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \end{bmatrix}$

Unit-1 (Data Analysis Functions)

```
x = [1,   5,    3;2,    4,    6];
y = [10,  2,    4;1,    8,    7];
min(x,y)
ans =
      1    2    3
      1    4    6
```

- Mean and Median:
- The mean of a group of values is the average of the values.
- The median is the value in the middle of the group, assuming that the values are sorted. If there is an odd number of values, the median is the value in the middle position. If there is an even number of values, then the median is the mean of the two middle values.

1) The command

```
mean(x)
```

Computes the mean value of a vector x.

Unit-1 (Data Analysis Functions)

for example, if x = [1   5   3], the mean value is 3.

```
x = [1,   5,   3]
mean(x)
ans =
    3.0000
```

returns a row vector containing the mean value from each column of a matrix x.

for example, if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the mean value of column 1 is 1.5, column 2 is 4.5 and column 3 is 4.5.

```
x = [1, 5, 3;2, 4, 6]
mean(x)
ans =
    1.5   4.5   4.5
```

The command

```
median(x)
```

Finds the median of the elements of a vector x.

Unit-1 (Data Analysis Functions)

For example, if x = [1   5   3], the median value is 3.

```
x = [1    5    3]
median(x)
ans =
    3
```

Returns a row vector containing the median value from each column of a matrix x, for example $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \\ 3 & 8 & 4 \end{bmatrix}$, the median value from column 1 is 2, column is 2 is 5 and column 3 is 4.

```
x = [1 5 3;2 4 6;3 8 4];
median(x)
ans =
    2    5    4
```

Unit-1 (Data Analysis Functions)

- Sums and Products
- MATLAB contains the functions for computing the sums and products of vectors (or of the columns in a matrix) and functions for computing the cumulative sums and products of vectors (or the elements of a matrix):

1) The command

```
sum(x)
```

Computes the sum of the elements of a vector x, for example, if x = [1 5 3] the sum is 9

```
x = [1    5    3]
sum(x)
ans =
    9
```

Computes a row vector containing the sum of the elements in each column of a matrix x. for example, $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the sum of column 1 is 3, column 2 is 9 and column 3 is 9.

Unit-1 (Data Analysis Functions)

```
x = [1   5   3;2   4   6]
sum(x)
ans =
    3    9    9
```

2) The command

```
prod(x)
```

Computes the product of the elements of a vector x, for example, if x = [1 5 3] is 15

```
x = [1    5    3]
prod(x)
ans =
    15
```

Computes a row vector containing the product of the elements in each column of a matrix x, for example if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ the product of column 1 is 2, column 2 is 20 and column 3 is 18.

```
x = [1    5    3;2    4    6]
prod(x)
ans =
    2    20    18
```

3) The command

```
cumsum(x)
```

Computes a vector of the same size containing cumulative sums of the elements of a vector x, for example, if x = [1   5   3], the resulting vector is x = [1   6   9]

```
x = [1    5    3]
cumsum(x)
ans =
    1    6    9
```

Computes a matrix containing the cumulative sum of the elements in each column of a matrix x,

Unit-1 (Data Analysis Functions)

For example, x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the resulting matrix is x = $\begin{bmatrix} 1 & 5 & 3 \\ 3 & 9 & 9 \end{bmatrix}$.

x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$

cumsum(x)

ans =

x = $\begin{bmatrix} 1 & 5 & 3 \\ 3 & 9 & 9 \end{bmatrix}$

4) The command

cumprod(x)

Computes a vector of the same size containing cumulative products of the elements of a vector x, for example, if x = [1 5 3], the resulting vector is x = [1 5 15].

x = [1 5 3]

cumprod(x)

ans =

[1 5 15]

Unit-1 (Data Analysis Functions)

Computes a matrix containing the cumulative product of the elements in each column of a matrix x, for example, x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the resulting matrix is x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 20 & 18 \end{bmatrix}$.

x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$

cumprod(x)

ans =

x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 20 & 18 \end{bmatrix}$.

- Sorting values
- The sort command arrange the values of a vector x into ascending order. If x is a matrix, the command sorts each column into ascending order:

1) The command

sort(x)

Sorts the elements of a vector x into ascending order. For example, if x = [1 5 3],

Unit-1 (Data Analysis Functions)

The resulting vector is x = [1   3   5]

```
x = [1   5   3]
sort(x)
ans =
    1   3   5
```

Sorts the elements in each column of a matrix x into ascending order, for example, if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the resulting matrix is x = $\begin{bmatrix} 1 & 4 & 3 \\ 2 & 5 & 6 \end{bmatrix}$

```
x = [1   5   3;2   4   6];
sort(x)
ans =
    1   4   3
    2   5   6
```

Note: if your data analysis requires you to evaluate data in rows, the data must be transposed. In other words, the rows must become column and the column must become rows.

Unit-1 (Data Analysis Functions)

- Determining Matrix Size:
- MATLAB offers two functions that allow us to determine how big a matrix is, size and length:

1) The command

```
size(x)
```

Determine the number of rows and columns in matrix x

```
x = [1    5    3;2    4     6]
size(x)
ans =
    2    3
```

2) The command

```
[a,b] = size(x)
```

Determine the number of rows and columns in matrix x and assigns the number of rows to a and the number of rows to b

```
[a,b] = size(x)
```

Unit-1 (Data Analysis Functions)

```
a =
    2
b =
    3
```

3) The command

```
length(x)
```

Determines the largest dimension of a matrix x.

```
x = [1   5   3;2   4   6];
length(x)
ans=
    3
```

- Variance and Standard deviation:
- Two of the most important statistical measurements of a set of data are the variance and standard deviation.
- The variance is the average squared deviation of the data from the mean.
- The standard deviation is defined as the square root of the variance.

Unit-1 (Data Analysis Functions)

The command

```
std(x)
```

Computes the standard deviation of the values in a vector x, for example, if x = [1 5 3], the standard deviation is 2.

```
x = [1    5    3];
std(x)
ans =
    2
```

Returns a row vector containing the standard deviation calculated for each column of a matrix x, for example, if x = $\begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the standard in column 1 is 0.7071, column 2 is 0.7071 and column 3 is 2.1213.

Unit-1 (Manipulating Matrices)

- A matrix can be defined by typing in a list of numbers enclosed in square brackets. The numbers can be separated by spaces or commas at the user's discretion.

For example:

```
A = [3.5]
B = [-1  0  0;1  1  0;0   0   2]
```

Note: A matrix can also be defined by listing each row on a separate line.

MATLAB also allows you to define a matrix by using another matrix that has already been defined.

For example,

```
if S = [3.0    1.5    3.1], then T = [1    2    3; S]
```

Returns

```
T =
    1    2    3
    3   1.5  3.1
```

-

We can also change values in a matrix or include additional values, by using a reference to specific locations. Thus, the command:

S(2) = -1.0

Returns

S =

3.0 -1.0 3.1

We can also extend a matrix by defining new elements. If we execute the command S(8) by define the eight element in matrix S:

S(8) = 9.5;

In matrix S will have eight values, and the values of S(4),S(5),S(6) and S(7) will be set to 0.

Thus S returns,

S =

3.0 -1.0 3.1 0 0 0 0 9.5

Unit-1 (Manipulating Matrices)

- Using the Colon Operator:

The colon operator is a very powerful operator for defining new matrices and modifying existing matrices.

An evenly spaced matrix can be defined with the colon operator.

Thus,

```
H = 1:8
```

Returns

```
H =
   1   2   3   4   5   6   7   8
```

Default spacing is 1. however, when colons are used to separate three numbers, the middle value becomes the spacing. For example,

```
time = 0.0:0.5:2.0
```

Returns

```
time =
   0   0.5000   1.0000   1.5000   2.0000
```

Unit-1 (Manipulating Matrices)

The colon operator can also be used to extract data from matrices, which becomes very useful in data analysis. When a colon is used in a matrix reference in place of a specific subscript, the colon represents the entire row or column.

If we define

```
M = [1 2 3 4 5;2 3 4 5 6;3 4 5 6 7];
```

Then we can extract column 1 from matrix M with the command

```
x = M(:,1)
```

Which returns

```
x =
    1
    2
    3
```

You can extract any of the columns in similar manner, so that

```
y = M(:,4)
```

returns

Unit-1 (Manipulating Matrices)

```
y =
    4
    5
    6
```

Similarly, to extract a row, use

```
z = M(1,:)
```

Which returns

```
z =
    1    2    3    4    5
```

You don't have to extract an entire row or an entire column. The colon operator can also be used to mean "from row_to row_" or "from column_to column_". To extract the two bottom rows of the M matrix, type

```
w = M(2:3,:)
```

Which returns

Unit-1 (Manipulating Matrices)

```
w =
   2   3   4   5   6
   3   4   5   6   7
```

Similarly, to extract just the four numbers in the lower right-hand corner of matrix M, use

```
w = M(2:3,4:5)
```

Which returns

```
w =
   5   6
   6   7
```

In MATLAB it is valid to have a matrix that is empty.

```
a = [ ];
```

Using the matrix name with a single colon, as in

```
M(:)
```

Transforms M into one long column matrix.

Unit-1 (Manipulating Matrices)

To find the value on row 2, column 3, use the following commands:

```
M(2,3)
ans =
    4
```

Alternatively, you can use a single index number. The value on row 2. column 3 of matrix M is element number 8. that is,

```
M(8)
ans =
    8
```

- Computational Limitations
- For example, suppose that we execute the following commands.

```
x = 2.5e200;
y = 1.0e200;
z = x*y
```

MATLAB responds with, z = Inf, because the answer (2.5*e400) is outside of the allowable range. This error is called exponent overflow.

Exponent underflow is similar error. But, the result of and exponent underflow is zero. MATLAB may also print a warning telling you that division by zero is not possible.

- A MATLAB function typically requires inputs called arguments to compute a result. However, some functions don’t require any input arguments. Although used as if they were scalar constants, the following functions do not require any input:

Unit-1 (Manipulating Matrices)

MATLAB Constants	Meaning
pi	Represents the mathematical constant π
i,j	Represents an imaginary number, the $\sqrt{-1}$
Inf	Represents infinity, which typically occurs as a result of division by zero.
NaN	Represents not-a-number and typically occurs when an expression is undefined, as in the division of zero by zero
clock	Represents the current time in a six-element row vector containing year, month, day, hour, minute and seconds.
date	Represents the current date in a character-string format, such as 28-July-2020
eps	Represents the epsilon floating-point precision for the computer being used. This epsilon precision is the smallest amount with which two values can differ in the computer.
ans	Represents a value computed by and expression, but not stored in a variable name.

Unit-1 (Commands and Functions)

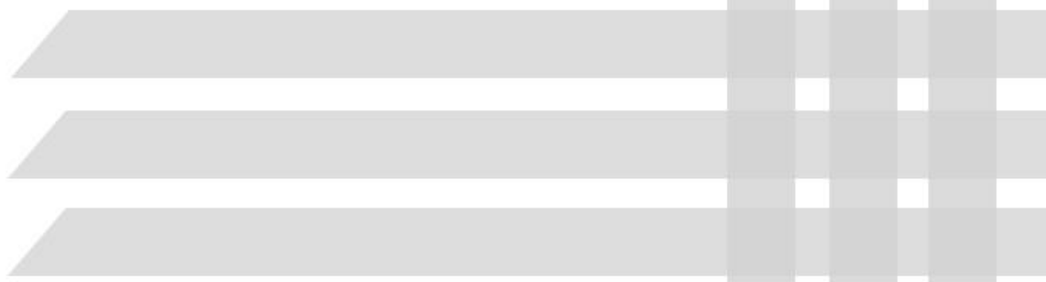
Commands	Meanings
abs	Computes the absolute value
asin	Computes the inverse sine or arcsine
ceil	Rounds to the nearest integer toward positive infinity
cos	Computes the cosine
cumprod	Computes a cumulative product of the values in an array
cumsum	Computes a cumulative sum of the values in an array
erf	Calculates the error functions
exp	Computes the value of e^x
fix	Rounds to the nearest integer toward zero
floor	Rounds to the nearest integer toward minus infinity
help	Opens the help function
length	Determines the largest dimension of an array
log	Computes the natural log
log10	Computes the log base 10
log2	Computes the log base 2

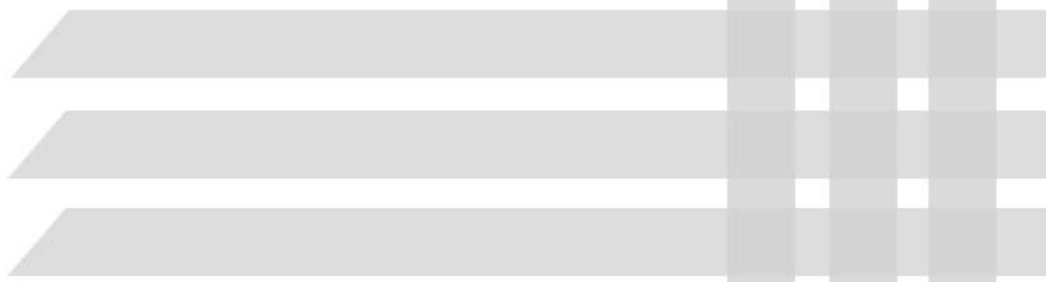
Unit-1 (Commands and Functions)

Command	Meaning
max	Finds the maximum value in an array
min	Finds the minimum value in an array
mean	Computes the average of the elements in an array
median	Finds the median of the elements in an array
prod	Multiplies the values in an array
sum	Sums the values in an array
sin	Computes the sine
tan	Computes the tangent
rand	Calculates evenly distributed random numbers
randn	Calculates normally distributed random numbers
rem	Calculates the remainder in a division problem
round	Rounds to the nearest integer
sign	Determines the sign
sinh	Computes the hyperbolic sine

Unit-1 (Commands and Functions)

Command	Meanings
size	Determines the number of rows and columns in an array
sort	Sorts the elements of a vector in to ascending order
sqrt	Calculates the square root of a number
std	Determines the standard deviation
Unit – 1 over	





UNIT – II

Plotting

Programming in MATLAB

Unit – II (Plotting)

- Introduction:

Large tables of data are difficult to interpret. Engineers use graphing techniques to make the information more accessible. With a graph it is easy to identify trends, pick out highs and lows, and isolate data points that may be measurement or calculation errors. A graph can also be used as a quick check to determine whether or not a computer solution is yielding expected results.

Unit – II (Plotting)

- Two – Dimensional Plots :

  The most common plot used by engineers is the x-y plot. The data that we plot are usually read from a data like or computed in programs and stored in vectors that we will call x and y. Generally, the x values represent the independent variable and the y values represent the dependent variable. The y values can be computed as a function of x, or the x and y values might be measured in an experiment.

Unit – II (Plotting)

- Basic Plotting

Once the x and y vectors have been defined, MATLAB makes it easy to create plots.

Assume that the trial values are stored in a vector called x and the distance values are stored in a vector called y.

```
x = [1:10];
y = [58.5,63.8,64.2,67.3,71.5,88.3,90.1,89.5,
90.5];
```

To plot these points, we use the plot command , with x and y as arguments.

```
plot(x,y)
```

A graphics window automatically opened.

The following commands add a title, x and y axis labels, and background grid.

```
plot(x,y)
title('Laboratory Experiment')
xlabel('Trial')
ylabel('Distance, ft')
grid on
```

These commands generated the plot.

As you type these commands into MATLAB, notice that the type colour changes to red when you enter a single quote (‘). This alerts you that you are starting a string. The colour changes to purple when you type the final single quote (‘). Indicating that you have completed the string.

If you are working in the command window, the graphics window will open on top of the other windows. To continue working, either click in the command window or minimize the graphics window. You can also resize the graphics window.

Unit – II (Plotting)

If you are working in an M-file, when you request a plot and then continue on with more computations. MATLAB will generate and display the graphics window and then return immediately to execute the rest of the commands in the program.

If you request a second plot, the graph you created will be overwritten. You can layer plots on top of one another by using the 'hold on' command. MATLAB will continue to layer the plots until the 'hold off' command is executed. In the default mode, MATLAB uses a blue line for the first set of data plotted and changes the colour for subsequent sets of data.

By typing 'help plot' in the command window, you can determine what choices are available to you. You can select solid, dashed, dotted, and dash-dot styles. The choices include plus sign, stars, circles, and x-marks among others. There are seven different colour choice.

Unit – II (Plotting)

Line Type	Indicator	Point Type	Indicator	Colour	Indicator
solid	-	point	·	blue	b
dotted	:	circle	o	green	g
Dash-dot	-.	x-mark	x	red	r
dashed	--	plus	+	cyan	c
		star	°	magenta	m
		square	s	yellow	y
		diamond	d	black	k
		Triangle down	V		
		Triangle up	^		
		Triangle left	<		
		Triangle right	>		
		pentagram	p		
		hexagram	h		

- Axes Scaling

MATLAB automatically scales the axes to fit the data values. However, you can over-ride this scaling with the axis command.

axis – freezes the current axis scaling for subsequent plots.

axis(v) – specifies that the axis being used is a four-element vector [xmin,xmax,ymin,ymax]

- Polar plots

MATLAB provides plotting capability with polar coordinates

Polar(theta,rho) - generates a polar plot of angle theta (radians) and radical distance (rho).

Unit – II (Plotting)

- Logarithmic plots

The MATLAB commands for generating linear and logarithmic plots of the vectors x and y:

plot(x,y) – generates a linear plot of the vectors x and y.

semilogx(x,y) – generates a plot of the values of x and y using a logarithmic scale for x and a linear scale for y.

semilogy(x,y) – generates a plot of the values of x and y using a linear scale for x and a logarithmic scale for y

loglog(x,y) – generates a plot of the vectors x and y using a logarithmic scale for both x and y.

Unit – II (Plotting)

- Bar graphs and Pie charts:
- bar(x) – when x is a vector, bar generates a vertical bar graph, When x is a two-dimensional matrix, bar groups the data by row.
- barh(x) – when x is a vector, barh generates a horizontal bar graph. When x is a two-dimensional matrix, barh groups the data by row.
- bar3(x) – generates a three-dimensional bar chart.
- barh3(x) – generates a three-dimensional horizontal bar chart.
- pie(x) – generates a pie chart. Each element in the matrix is represented as a slice of the pie.
- pie3(x) – generates a three-dimensional pie chart. Each element in the matrix is represented as a slice of the pie.

- Histograms:
- In MATLAB, the histogram computes the number of values falling in 10 bins that are equally spaced between the minimum and maximum values, from the set of values. For example,

```
x = [100,95,74,87,22,78,34,35,93,88,86,42,55,48];
hist(x)
```

The default number of bins is 10, but if we have a large data set we may want to divide up the data into more categories(bins).

Unit – II (Plotting)

- Three – Dimensional line plots:
- The plot3 function is similar to the plot function, expect that it accepts data in three dimensions. plot3(x,y,z) these ordered ‘triples’ are then plotted in three-spaces and connected with straight lines.
- Mesh plots:
- There are several ways to use mesh plots. They can be used to good effect with a single two-dimensional matrix. In the application, the value in the matrix represents the z-value in the plot. The x and y values are based on the matrix dimensions.
- Surf plots:
- Surf plots are similar to mesh plots, but surf creates a three dimensional coloured surface instead of a mesh. The colours vary depending on the value of z.

Unit – II (Plotting)

- Contour plots:

Contour plots are two-dimensional representations of three-dimensional surfaces. Maps often represent elevations with contours.

- Editing Plots from the Menu Bar:

In MATLAB, you also edit a plot once you have created it. To annotate the graph, select the insert menu, you can insert labels, titles, legends, text boxes, etc., all by using this menu. The Tools Menu allows you to change the way the plot looks by zooming in or out, changing the aspect ratio, etc., the figure toolbar underneath the menu tool bar offers icons that allow you to do the same thing. Similarly, labels, a title, and a colour bar were added using the insert menu option on the menu bar. Editing your plot in this manner is more interactive and allows you.

- Creating plots from the Workspace window

  A great feature of MATLAB is the ability to inter-actively create plots from the workspace window. In the workspace window, select a variable, then select the dropdown menu on the plotting icon. MATLAB will list the plotting options it thinks are reasonable for the data stored in your variable. Simply select the appropriate option, and your plot is create in the current figure window. If you don’t like any of the suggested plot types, choose ‘More plots...’ From the dropdown menu.

Programming in MATLAB

Unit – II (Programming in MATLAB)

- Problems with two variables:

Consider the following MATLAB statements,

```
x = 3
y = 5
A = x*y
```

Since x and y are scalars, it’s an easy calculation, x*y = 15. thus

```
A =
    15
```

Now let’s see what happens if x is a matrix and y is still a scalar:

```
x= 1:5;
```

Returns five values of x. Because y is still a scalar with only one value,

```
A = x*y
```

Returns

```
A =
     5     10     15     20     25
```

But what happens if y is also a vector?

Then

```
y = 1:3;
A = x*y
```

returns an error statement. This error statement reminds us that the asterisk is the operator for matrix multiplication, which is not what we want. We want the dot-asterisk operator(.*), which will perform an element-by-element multiplication.

```
y = linspace(1,3,5)
```

creates a new y with five evenly spaced elements:

```
y =
    1.0000    1.5000    2.0000    2.5000    3.0000
A = x .* y
A =
      1    3    6    10    15
```

Unit – II (Programming in MATLAB)

In order for your answer, 'A' to be a two-dimensional matrix, the input vectors have to be two-dimensional matrices. MATLAB has a built-in function called meshgrid that will help you accomplish this and x and y don't even have to be the same size. First let's change y back to a three-element vector:

```
y = 1:3;
```

Then we'll use meshgrid to create a new two-dimensional version of both x and y that we'll call new_x and new_y:

```
[new_x, new_y] = meshgrid(x,y)
```

The meshgrid command takes the two input vectors and creates two 2-D matrices.

```
new_x =
   1   2   3   4   5
   1   2   3   4   5
   1   2   3   4   5
```

```
new_y =
    1    1    1    1    1
    2    2    2    2    2
    3    3    3    3    3
```

We really want:

```
A = new_x .* new_y
A =
    1    2    3    4    5
    2    4    6    8   10
    3    6    9   12   15
```

Unit – II (Programming in MATLAB)

- In put / Out put:

The input command pauses the program and prompts the user for input. While the ‘disp’ and ‘fprintf’ commands provide output to the command window. The pause command stops the program execution until any key is typed.

- User Defined Input:

The input function allows us to do this. It displays a text string in the command window, then waits for the user to provide the requested input. For example,

```
z = input('Enter a value')
```

Displays

Enter a value

In the command window. If the user enters a value such as

5

The program assigns the value of 5 to the variable z.

If the input command does not end with a semicolon, the value entered is displayed on the screen:

```
z =
    5
```

The same approach can be used to enter a one or two-dimensional matrix. The user must provide the appropriate brackets and delimiters (commas and semicolons). For example:

```
z = input ( 'Enter values for z in brackets' )
```

requests the user to input a matrix such as

```
[1, 2, 3;4, 5, 6]
```

and responds

```
z =
    1   2   3
    4   5   6
```

this user input value of z can then be used in subsequent calculations by the script M-file.

- Output Options

The simplest way is enter the name of the matrix, without a semicolon. The name of the matrix will be repeated and the values of the matrix will be displayed, starting on the next line. For example, first define a matrix x:

```
x = 1:5;
```

Because there is a semicolon at the end of the assignment statement, the values in x are not repeated in the command window. However, if later in your program you want to display x, simply type in the variable name, which returns

```
x =
    1    2    3    4    5
```

MATLAB offers two other approaches to displaying results:

The ‘disp’ function and the ‘fprintf’ function.

- Display function:

The display function can be used to display the contents of a matrix without printing the matrix's name:

disp(x)

returns

1 2 3 4 5

The display command can also be used to display a string.

display ('The values in the x matrix are: ');

disp(x);

returns

the values in the x matrix are:

1 2 3 4 5

The semicolon at the end of the disp function is optional.

You can get around this by creating a matrix of your two outputs, using the 'num2str' function:

disp(['The values in the x array are: 'num2str(x)])

which returns

the values in the x array are: 1   2   3   4   5

The disp function requires a matrix as input. It can be a string. A variable that represents a matrix (or) the programmer can define the matrix inside square brackets, using the standard matrix definition rules. The num2str function changes an array of numbers into a string.

You can see the resulting matrix by typing

A = ['The values in the x array are: 'num2str(x)]

which returns

A =

the values in the x array are: 1   2   3   4   5

Unit – II (Programming in MATLAB)

- Formatted Output:

The fprintf function gives you even more control over the output than you have with the disp function. In addition to displaying both text and matrix values, you can specify the format to be used in displaying the values, and you can specify when to skip to a new line.

The general form of this command contains two arguments – a string and a list of matrices:

fprintf (format-string, var,….)

Consider the following example:

```
temp = 98.6;
fprintf ( ‘ The temperature is %f degrees F ‘, temp)
```

The string, which is the first argument inside the fprintf function, contains a place holder (%) where the value of the variable will be inserted. This place holder also contains formatting information.

MATLAB allows you to specify an exponential format, %e, or let you allow MATLAB to choose whichever is shorter, fixed point or exponential %g.

MATLAB does not automatically start a new line after an fprintf function is executed, so if the following new commands are issued.

To cause MATLAB to start a new line, you'll need to use \n, called a linefeed, at the end of the string:

```
temp = 98.6;
fprintf('The temperature is %f degrees F \n', temp)
ftemp = 100.1
fprintf(' The temperature is %f degrees F \n', temp)
```

Which returns

```
The temperature is 98.6000 degrees F
The temperature is 100.1000 degrees F
```

You can further control how the variables are displayed by using the

optional ‘width field’ and ‘precision field’ with the format command.

The width field controls the minimum number of characters to be printed. It must be a positive decimal integer.

The precision field is preceded by a period (.) and specifies the number of decimal places after the decimal point for exponential and fixed point types. For example,

```
fprintf(‘The temperature is %8.2f degrees F\n’, temp);
```

returns

```
The temperature is 100.10 degrees F
```

Where %8.2 specifies that the minimum total width available to display your result is 8 digits, two of which are after the decimal point.

Many times when you use the fprintf function, your variable will be a matrix. For example,

```
temp = [98.6, 100.1, 99.2];
```

MATLAB will repeat the string in the fprintf command until it uses all of the values in the matrix:

fprint(‘ The temperature is %8.2f degrees F\n’, temp);

returns

The temperature is 98.60 degrees F

The temperature is 100.10 degrees F

The temperature is 99.20 degrees F

If the variable is a two dimensional matrix, MATLAB uses the values one column at a time, going down the first column, then the second,etc.

patient = 1:3;

temp = [98.6, 100.1, 99.2];

Combine these two matrices:

history = [patient ; temp]

Returns

history =

```
    1.0000      2.0000      3.0000
   98.6000    100.1000     99.2000
```

Now we can use the fprintf function to create a table that is easier to interpret:

fprintf('Patient %4f had a temperature of %8.2f \n' ,history)

Returns

Patient 1 had a temperature of 98.60

Patient 2 had a temperature of 100.10

Patient 3 had a temperature of 99.20.

Unit – II (Programming in MATLAB)

TYPE FIELD FORMAT	
TYPE FIELD	RESULT
%f	Fixed point or decimal notation
%e	Exponential notation
%g	Whichever is shorter %f or %e

SPECIAL FORMAT COMMAND	
FORMAT COMMAND	RESULTING ACTION
\n	Line feed
\r	Carriage return
\t	tab
\b	Back space

UNIT – II (Programming in MATLAB)

- Functions

The MATLAB programming language is built around functions. A function is simply a piece of computer code that accepts an input argument from the user and provides output to the program. Functions allow us to program efficiently, since we don't need to rewrite the computer code for calculations that are performed frequently. User defined functions are stored as M-files, and can be accessed by MATLAB if they are in the current directory.

UNIT – II (Programming in MATLAB)

User defined functions are stored as M-files, and can be accessed by MATLAB if they are in the current directory.

- Syntax

User defined MATLAB functions are written in M-files. Access a new function M-file the same way a script M-file is created:

File → New →M-file from the menu bar.

For example,

```
function s = f(x)
% A function that adds 3 to every member of  an array x
s = x+3;
```

These lines of code define a function called f. notice that the first line starts with the word function. This is a requirement for all user defined functions. Next, an output variable that we've named s is set equal to the function name, with the input arguments enclosed in parentheses (x). Finally, the output s, is defined as x+3.

UNIT – II (Programming in MATLAB)

More complicated functions can be written that require more than one input argument. For example,

```
function output = g(x,y)
%This function multiplies x and y together
%Be sure that x and y are the same size matrices
a = x .* y;
output = a;
```

- Local Variables

The variable used in function M-files are known as local variable. The only way that a function can communicate with the workspace is through input arguments and the output returned. Any variables defined within the function only exist for the function to use.

```
function output = g(x,y)
%This function multiplies x and y together
%Be sure that x and y are the same size matrices
a = x .* y;
output = a;
```

The inside the variable a is a local variable. It can be used for additional calculations inside the g function, but it is not stored in the workspace.

- Rules for writing and using Function M-files

Writing and using a function M-file requires user to follow very specific rules, and a format when writing it. These rule are summarized as follows:

- The function must begin with a line containing the word function, which is followed by the output argument, an equals sign, and the name of the function. The input arguments to the function follow the name of the function and are enclosed in parentheses. This line distinguishes the function file from a script M-file.

  function output_name = function_name(input)

- The first few lines of the function should be comments because they will be displayed if help is requested for the function name:

  % comment your function so users will know how to use it

- The only information returned from the function is contained in the output arguments, which are of course matrices. Always check to be sure that the function includes a statement that assigns a value in the output argument.
- A function that has multiple input arguments must list the arguments in the function statement.

  function error = mse(w,d)
- A function that is going to return more than one value should show all values to be returned as a vector in the function statement as in

  function [dist, vel, accel] = motion(x)

  all output values need to be computed within the function.
- The same matrix names can be used in both a function and the program that references it. No confusion occurs as to which matrix is referenced. Because the function and the program are completely separate.
- The special function nargin and nargout can be used to determine the number of input arguments and the number of output arguments for a function. Both require a string containing the function name as input.

UNIT – II (Programming in MATLAB)

- Statement Level Control Structures:

One way to think of computer programs is to consider how the statements that compose the program are organized. Usually, sections of computer code can be categorized into on of three structures:

sequences, selection structures and repetition structures.

- Sequences: Sequences are lists of commands that are executed one after another.
- Selection structure: A selection structure allows the programmer to execute one command or group of commands if some criteria is true and a second set of commands if the criteria is false. A selection statement provides the means of choosing between these paths based on a logical condition. The conditions that are evaluated often contain relational and logical operators or functions.
- Repetition structure: A repetition structure or loop, causes a group of statements to be executed zero, one, or more times. The number of times a loop is executed depends on either a counter or the evaluation of a logical condition.

UNIT – II (Programming in MATLAB)

- Relational and Logical Operators:

MATLAB has six relational operators for comparing two matrices of equal size,

Comparisons are either true or false and most computer programs use the number 1 for true and 0 for false. If we define two scalars

```
x = 5;
y = 1;
```

and use a relational operator such as <,

```
x < y
```

The result is either true or false. In this case, x is not less than y. so MATLAB responds

```
ans =
     0
```

Indicating the comparison was not true. MATLAB uses this answer in selection statements and to repetition structures to make decisions.

Of course, variables in MATLAB usually represent entire matrices. If we redefine x and y.

we can see how MATLAB handles comparisons between matrices:

```
x = 1:5;
y = x-4;
x<y
```

returns

```
ans =
    0    0    0    0    0
```

MATLAB compares corresponding elements and creates and answer matrix of zeros and ones. In the previous example, x was greater than y for every element comparison, so every comparison was false, and the answer was a string of zeros. If instead,

```
x = [ 1, 2, 3, 4, 5];
y = [-2, 0, 2, 4, 6];
x<y
ans =
    0    0    0    0    1
```

Which tells us that the comparison was false for the first four elements

UNIT – II (Programming in MATLAB)

But true for the last. In order for MATLAB to decide that a comparison is true for an entire matrix. It must be true for every element in the matrix.

MATLAB also allows us to combine comparisons with logical operators; and, not, and or.

Consider the following:

```
x = [1, 2, 3, 4, 5];
y = [-2, 0, 2, 4, 6];
z = [8, 8, 8, 8, 8];
z>x & z>y
```

returns

```
ans =
    1   1   1   1   1
```

Because z is greater than both x and y for every element. The statement

```
x>y | x>z
```

Is read as “x is greater than y or x is greater than z” and returns

UNIT – II (Programming in MATLAB)

Relational Operator	Interpretation
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to
~ =	Not equal to

Logical Operator	Interpretation
&	and
~	not
|	or

```
ans =
   1   1   1   0   0
```

This result is interpreted to mean that the condition is true for the first three elements and false for the last two.

- Selection structures

MATLAB offers two kinds of selection structures:

find and a family of if structures.

- Find

The find command is unique to MATLAB, and can often be used instead of both if and loop structures. It returns a vector composed of the indices of the nonzero elements of a vector x. those indices can then be use in subsequent commands. The usefulness of the find command is best described with examples.

Assume that you have a list of temperatures measured in a manufacturing process. If the temperature is less than 95 degree F, the

widgets produced will be faulty:

```
temp = [100, 98, 94, 101, 93];
```

Use the find function to determine which widgets are family:

```
find(temp<95)
```

Returns a vector of element numbers:

```
ans =
    3    5
```

Which tells us that items 3 and 5 will be faulty.

MATLAB first evaluated temp<95, which resulted in a vector of zeros and ones. Which returns a vector indicating when the comparison was true (1) and when it was false (0):

```
ans =
    0    0    1    0    1
```

When the find command is used with a two-dimensional matrix, a single element number is returned. For example, consider the following two-dimensional matrix and use find to determine the

UNIT – II (Programming in MATLAB)

Location of all elements greater than 9:

```
x = [1, 2, 3;10, 5, 1;3, 12, 2;8, 3,1]
element = find(x>9)
[row, column] = find(x>9)
```

returns

```
x =
    1    2    3
   10    5    1
    3   12    2
    8    3    1
element =
    2
    7
row =
    2
`   3
```

```
column =
        1
        2
```

Notice that the numbers 10 and 12 are the only two values greater than 9. by counting down the columns, we see that they are elements 2 and 7 respectively, using the alternative designation, 10 is in row 2, column 1 and 12 is in row 3, column 2.

- If, else and elseif

Most of the time, the find command can and should be used instead of an ‘if’.

- If:

A simple if statement has the following form:

```
if comparison
        statements
end
```

If the logical expression is true, the statements between the if statement and the end statement are executed.

If the logical expression is false, program control jumps immediately to the statement following the end statement. It is good programming practice to indent the statements within an if structure for readability. For example,

```
if G<50
        count = count + 1;
        disp(G);
end
```

This statement is easy to interpret if G is a scalar. For example, if G has a value of 25, then count is incremented by 1 and G is displayed on the screen. However, if G is not a scalar, then the if statement considers the comparison true, if only is true for every element. If G is defined from 0 and 80.

```
G = 0:10:80;
```

Then the comparison is false, and the statement inside the if statement are not executed. If statement work best with scalars.

- Else

The else clause allows us to execute one set of statements if the comparison is true, and a different set of statement if the comparison is false. Assume that we have a variable interval. If the value of interval is less than 1, we set the value of x_increment to interval/10. otherwise, we set the value of x_increment to 0.1.

```
if interval < 1
        x_increment = interval/10;
else
        x_increment = 0.1;
end
```

When interval is a scalar, this is easy to interpret. However, when interval is a matrix, the comparison is only true if it is true for every element in the matrix. So, if

```
interval = 0:0.5:2;
```

The elements in the matrix are not all less than 1. therefore, MATLAB skips to the else portion of the statement of the statement, and all

Values in the x_increment vector are set equal to 0.1. Again, if/else statements are probably best confined to use with scalars, although you may find limited use with vectors.

- Elseif:

In these cases, the elseif clause is often used to clarify the program logic, for example,

```
if temperature >100
        disp('Too hot-equipment malfuntioning')
elseif temperature >90
        disp('Normal operating temperature')
elseif temperature >50
        disp('Temperature below desired operating range')
else
        disp('Too cold-turn off equipment')
end
```

In this example, temperature above 90 and below or equal to 100 are in the normal operating range.

Temperature outside of this range generate an appropriate message. Notice that a temperature of 101 does not trigger all of the responses. Also notice that the final else does not require a comparison. In order for the computation to reach the final else, the temperature must be less than or equal to 50.

Again this structure is easy to interpret if temperature is a scalar. If it is a matrix, the comparison must be true for every element in the matrix. If you had a temperature matrix

temperature = [90, 95, 101]

The first comparison would be false. The second comparison would also be false. Finally, the third comparison would be true, since all of the temperature are above 50. As before, elseif structures work well for scalars, but find is probably a better choice for matrices.

- Loops

A loop is a structure that allows you to repeat a set of statements. In general, you should avoid loops in MATLAB because they are seldom needed, and they can significantly increase the execution time of a program.

UNIT – II (Programming in MATLAB)

- For Loops

In general, it is possible to use either a for or a while loop in any situation that requires a repetition structure. However, for loops are the easier choice when you know how many times you want to repeat a set of instructions. The general format is

```
for     index = expression
            statements
end
```

Usually, the expression is a vector, and the statements are repeated as many times as there are columns in the expression matrix. For example, to find 5 raised to the 100th power, first initialize a running total:

```
total = 1;
for k = 1:100
        total = total*5;
end
```

The first time through the loop, total = 1, so total×5 = 5. the next line

Through the loop, the value of total is updated to 5×5 equals 25. after 100 times through the loop, the final value is found, and corresponds to 5^{100}. Notice that the value of total was suppressed, so it won't print out each time through the loop. To recall the find value of total, type

```
total
```

Which returns

```
total =
        7.8886e+069
```

- The rules for writing and using a for loop are the following:

a. The index of a for loop must be a variable. Although k is often used as the symbol for the index, any variable name can be used. The use of k is strictly a style issue.

b. If the expression matrix is the empty matrix, the loop will not be executed. Control will pass to the statement following the end statement.

a. If the expression matrix is a scalar, the loop will be executed one time, with the index containing the value of the scalar.
b. If the expression is a row vector, each time through the loop the index will contain the next column in the matrix.
c. If the expression matrix, each time through the loop the index will contain the next column in the matrix. This means that the index will be a column vector.
d. Upon completion of a for loop, the index contains the last value used.
e. The colon operator can be used to define the expression matrix using the following format:

```
for k = initial : increment : limit
```

UNIT – II (Programming in MATLAB)

- While loops:

The while loop is a structure used for repeating a set of statements as long as specified condition is true. The general format for this control structure is

```
while expression
        statements
end
```

The statements in the while loop are executed as long as the real part of the expression has all nonzero elements. The expression is usually a comparison using relational and logical operators. When the result of a comparison is true, the result is 1, and therefore ‘nonzero’. The loop will continue repeating as long as the comparison is still true. When the expression is evaluated as false, control skips to the statement following the end statement . Consider the following example:

First initialize a:

```
a = 0;
```

Then find the smallest multiple of 3 that is less than 100:

```
while( a < 100 )
        a = a+3;
end;
```

The last time through the loop a will start out as 99, then will become 102 when 3 is added to 99. The smallest multiple then becomes

```
a – 3
```

Which returns

```
ans =
   99
```

The variable modified in the statements inside the loop should include the variables in the expression, or else the value of the expression will never change. If the expression is always true, then the loop will execute an infinite number of times.

UNIT – II (Programming in MATLAB)

COMMANDS AND FUNCTIONS	
COMMAND	MEANINGS
clock	Determines the current time on the CPU clock
disp	Displays matrix or text
else	Defines the path if the result of an if statement is false
elseif	Defines the path if the result of an if statement is false, and specifies a new logical test
end	Identifies the end of a control structure
etime	Finds elapsed time
find	Determines which elements in a matrix meet the input criteria
for	Generates a loop structure
fprintf	Prints formatted information
function	Identifies an M-files as a function

UNIT – II (Programming in MATLAB)

COMMANDS AND FUNCTIONS	
COMMAND	MEANING
if	Tests a logical expression
input	Prompts the user to enter a value
meshgrid	Maps two input vectors onto two 2-D matrices
nargin	Determine the number of input arguments in a function
nargout	Determines the number of output arguments from a function
num2string	Converts an array into a string
ones	Creates a matrix of ones
tic	Starts a timing sequence
toc	Stops a timing sequence
while	Generates a loop structure
zeros	Creates a matrix of zeros

UNIT - III

NUMERICAL TECHNIQUES

Unit – III (Numerical Techniques)

- Interpolation:

Interpolation is a technique by which we estimate a variable's value between two known values. There are a number of different techniques for this, but in this section we present the two most common types of interpolation:

(i) linear interpolation

(ii) cubic-spline interpolation

In both techniques, we assume that we have a set of data points which represents a set of xy-coordinates for which y is a function of x:

that is, $y = f(x)$.

We then have a value of x that is not part of the data set for which we want to find the y value.

Unit – III (Numerical Techniques)

- Linear Interpolation:

Linear interpolation is one of the most common techniques for estimating data values between two given data points. With this technique, we assume that the function between the points can be estimated by a straight line drawn between the points. If we find the equation of a straight line defined by the two known points, we can find y for any value of x. the closer together the points are the more accurate our approximation is likely to be. We could use the equation to extrapolate points past out collected data. This is rarely wise, however, and often leads to large errors.

Unit – III (Numerical Techniques)

- Cubic-Spline Interpolation:

A cubic spline is a smooth curve constructed to go through a set of points. The curve between each pair of points is a third-degree polynomial which is computed so that it provides a smooth curve between the two points and a smooth transition from the third-degree polynomial between the previous pair of points.

- Interp1 Function:

The MATLAB function that performs interpolation, interp1, has two forms. Each form assumes that vectors x and y contain the original data values and that another vector x_new contains the new point or points for which we want to compute interpolated y_new values. For example, the points were generated with the following commands:

```
x = 0:5;
y = [0, 20, 60, 68, 77, 110];
```

Suppose we would like to find a value for y, if x = 1.5. unfortunately 1.5 is not one of the elements in the x vector, so we'll need to perform an interpolation:

```
interp1(x,y,1.5)
```

returns

```
ans =
   40
```

We can see from this answer corresponds to a linear interpolation between the x,y points (1,20) and (2,60). The function interp1 defaults to linear interpolation unless otherwise specified.

If, instead of a scalar value of new_x values, we define an array of new_x values, the function returns an array of new_y values:

```
new_x = 0:0.2:5
new_y = interp1(x,y,new_x)
```

The new calculated points are plotted. They all full on a straight line connecting the original data points. The commands to generate the graph are

```
plot(x,y,new_x,new_y, ’o’)
axis([-1, 7, -20, 120])
title(‘Linear Interpolation Plot’)
xlabel(‘x values’)
ylabel(‘y vaues’)
```

If we wish to use a cubic spline interpolation approach, we must add a fourth argument to the interp1 function. The argument must be a string. To find the value of y at x = 1.5 using a cubic spline, type

```
interp1(x, y, 1.5, 'spline')
```

Which returns

```
ans =
   42.2083
```

- Curve Fitting:
- Linear Regression

Linear regression is the name given to the process that determines the linear equation which is the best fit to a set of data points, in terms of minimizing the sum of the squared distances between the line and the data points. For example,

```
x = 0:5;
y = [0, 20, 60, 68, 77, 110];
```

If we plot these points, it appears that a good estimate of a line through the points is y = 20x.

(Note: this process is sometimes called "eyeballing it" – meaning that no calculations were done, but it looks like a good fit)

Looking at the plot, we can see that the first two points appear to fall exactly on the line, but the other points are off by varying amounts. To compare the quality of the fit of this line to other possible estimates, we find the difference between the actual y value and the value calculated from the estimate.

If we sum the differences, some of the positive and negative values would cancel each other out and give a sum that is smaller than it should be. To avoid this problem, we could add the absolute value of the differences, or we could square them. The least squared technique uses the squared values. Therefore, the measure of the quality of the fit of this linear estimate is the sum of the squared distances between the points and the linear estimates. This sum can be calculated by the following command:

sum_sq = sum((actual y – calculated y)^2)

If we drew another line through the points we could compute the sum of square that corresponds to the new line. Of the two lines, the better fit is provided by the line with the smaller sum of squared distances.

We call it linear regression when we derive the equation of a straight line, but more generally it is called polynomial regression.

- Polynomial Regression:

Linear regression is a special case of the polynomial regression technique.

The degree of a polynomial is equal to the largest value used as an exponent. Therefore, the general form of a cubic polynomial is $a_0x^3 + a_1x^2 + a_2x^1 + a_3x^0$. We plot the original set of data points that we used in the linear regression, along with plots of the best-fit polynomials with degrees two through five. As the degree of the polynomial increases, the number of points that fall on the curve also increases. if a set of n+1 points is used to determine an nth degree polynomial, all n points will fall on the polynomial.

Unit – III (Numerical Techniques)

- Polyfit and Polyval Function:

The MATLAB function for computing the best fit to a set of data with a polynomial is "polyfit". The function has three argument:

    the x coordinate of the data points,
    the y coordinate of the data points and
    the degree n of the polynomial.

The function returns the coefficients, in descending powers of x, of the nth degree polynomial used to model the data. For example,

```
x = 0:5
y = [0, 20, 60, 68, 77, 110]
```

The function

```
polyfit(x,y,1)
```

Returns

```
ans =
    20.8286    3.7619
```

So the first order polynomial that best fits our data is

f(x) = 20.8286x + 3.7619

Similarly, we can find other polynomial to fit the data by specifying a higher order in the polyfit equation. Thus,

polyfit(x,y,4)

Returns

ans =

1.5625 -14.5231 38.6736 -3.4511 -0.3770

Which corresponds to a fourth-order polynomial:

f(x) = $1.5625x^4 - 14.5231x^3 + 38.6736x^2 - 3.4511x - 0.3770$

we could use the coefficients to create equation to calculate new values of y. for example,

```
y_first_order_fit = 20.8286.*x + 3.7619;
```

Or we could use the function polyval provided by MATLAB to accomplish the same thing.

The polyval function is used to evaluate a polynomial at a set of data points. The first argument of the polyval function is a vector containing the polynomial and the second argument is the vector of x values for

Which we want to calculate corresponding y values.

polyfit(x,y,n)	Returns a vector of n+1 coefficients that represents the best fit polynomial of degree n for the x and y coordinates provided. The coefficient order corresponds to decreasing powers of x.
polyval(coef,x)	Returns a vector of polynomial values f(x) that correspond to the x vector values. The order of the coefficients corresponds to decreasing powers of x.

- Using the Interactive Fitting Tools:

In MATLAB, interactive plotting tools that allow you to annotate your plots without using the command windows. The software also includes basic curve fitting more complicated curve fitting and statistics tools.

Unit – III (Numerical Techniques)

- Basic Fitting Tools:

To access the basic fitting tools, first create a figure:

```
x = 0:5;
y = [0, 20, 60, 68, 77, 110]
plot(x,y, 'o')
axis([-1, 7, -20, 120])
```

To activate the curve fitting tools, select Tools → Basic Fitting from the menu bar on the figure. The Basic Fitting window opens on top of the plots. By checking linear and cubic and show equations, the plot is generated.

Checking the plot residuals box generates a second plot, showing how far each data point is form the calculated line.

In the lower right-hand corner of the Basic Fitting window is an arrow button. Selecting that button twice opens the rest of the Basic Fitting window.

The centre panel of the window shows the results of the curve fit and offers the select x values and calculate y values based on the equation displayed in the centre panel.

In addition to the Basic Fitting window, you can also access the Data Statistics window from the figure menu bar. Select Tools → Data Statistics from the figure window. This window allows you to calculate statistical functions interactively, such as mean and standard deviation, based on the data in the figure, and allows you to save the results to the workspace.

- Curve Fitting Toolbox:

In addition to the basic fitting utility, MATLAB contains toolboxes to help you perform more specialized statistical and data fitting operations. In particular, the Curve Fitting toolbox contains a Graphical User Interface (GUI) that allows you to fit with more tools than just polynomials.

Before you access the curve fitting toolbox, you'll need a set of data to analyze.

```
x = 0:5;
y = [0, 20, 60, 68, 77, 110];
```

To open the curve fitting toolbox, type

```
cftool
```

This launches the curve fitting tool window. Now you'll need to tell the curve fitting tool what data to use. Select the data button, which will open a data window. The data window has access to the workspace and will let you select an independent(x) and dependent(y) variable from a drop-down list.

From the drop-down lists, you should choose x and y. At this point you can close the data window.

Going back to the Curve Fitting Tool window, you now select the Fitting button, which offers you choices of fitting algorithms. Select New Fit, and select a fit type from the type of fit list. You can experiment with fitting choices to find the best one for your graph. We choose an interpolated scheme, which forces the plot through all points, and a third order polynomial.

Unit – III (Numerical Techniques)

- Numerical Integration:

The integral of a function f(x) over the interval [a,b] is defined to be the area under the curve of f(x) between a and b.

For many function, this integral can be computed analytically. However, for a number of function, the integral cannot easily be computed analytically and thus requires a numerical technique to estimate its value. The numerical evaluation of an integral is also called quadrature.

The numerical integration techniques estimate the function f(x) by another function g(x). Then, the better the estimate of g(x) to f(x). The better will be the estimate of the integral of f(x). Two of the most common numerical integration techniques estimate f(x) with a set of piecewise linear functions or with a set of piecewise parabolic functions.

If we estimate the function with piecewise linear functions, we can then compute the area of the trapezoids that compose the area under the piecewise linear function, this technique is called the Trapezoidal rule. If we estimate the function with piecewise quadratic functions, we can then compute and add the areas of these components , this technique is called Simpson's rule.

Unit – III (Numerical Techniques)

- Trapezoidal Rule:

If the area under a curve is represented by trapezoids and if the interval [a,b] is divided into n equal sections, then the area can be approximated by the formula:

$$K_T = \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\right)$$

where the x_i values represent the endpoints of the trapezoids and where x_0 = a, x_n = b and h = $\frac{b-a}{n}$

- Simpson's rule:

If the area under a curve is represented by area under quadratic sections of a curve, and if the interval [a,b] is divided into 2n equal sections, then the area can be approximated by the formula:

$$K_s = \frac{h}{3}\begin{pmatrix} [f(x_0) + f(x_n)] + 4[f(x_1) + f(x_3) + \cdots f(x_{2n-1})] \\ +2[f(x_2) + f(x_4) + \cdots + f(x_{2n-2})] \end{pmatrix}$$

where the x_i values represent the endpoints of the Simpson's and where x_0 = a, x_n = b and h = $\frac{b-a}{n}$

Unit – III (Numerical Techniques)

If the piecewise components of the approximating function are higher degree functions the integration techniques are referred to as Newton-Cotes integration techniques.

The estimate of an integral improves as we use more components to approximate the area under a curve.

If we attempt to integrate a function with a singularity, we may not be able to get a satisfactory answer with a numerical integration technique.

- MATLAB Quadrature Functions

MATLAB has two quadrature functions for performing numerical function integration. The quad function uses an adaptive form of Simpson’s rule. The quad1 function is better at handling functions with certain types of singularities.

The simplest form of the quad and quad1 functions requires three arguments. The first argument is the name of the MATLAB function that returns a vector of values of f(x) when given a vector of input values x.

Unit – III (Numerical Techniques)

This function name can be the name of another MATLAB function. The second and third arguments are the integral limits a and b.

quad('function',a,b)	Returns the area of the 'function' between a and b, assuming that 'function' is a MATLAB function
quad1('function',a,b)	Returns the area of the 'function' between a and b, assuming that 'function' is a MATLAB function

These integration techniques can handle some singularities that occur at one or the other interval endpoints. But they cannot handle singularities that occur within the interval. For these cases, you should consider dividing the interval into subintervals and providing estimates of the singularities using other results, such as L'Hopital's rule.

Unit – III (Numerical Techniques)

- Numerical Differentiation:

The derivative of a function f(x) is defined to be a function $f'(x)$ that is equal to the rate change of f(x) with respect to x. the derivative can be expressed as a ratio, with the change in f(x) indicated by df(x) and the change in x indicated by dx, giving

$$f'(x)=\frac{df(x)}{dx}$$

The derivative $f'(x)$ can be described graphically as the slope of the function f(x), where the slope of f(x) is defined to be the slope of the tangent line to the function at the specified point.

Points with derivatives of zero are called critical points and can represent either a horizontal region, a local maximum or a local minimum of the function.

If we evaluate the derivative of a function at several points in an interval and we observe that the sign of the derivative changes, then a local maximum or local minimum occurs in the interval.

Unit – III (Numerical Techniques)

The second derivative can be used to determine whether or not the critical points represent local maxima or local minima. More specifically, if the second derivative of an extrema point is positive, then the value of the function at the extreme point is a local minimum. If the second derivative of an extrema point is negative, then the value of the function at the extrema point is local maximum.

- Difference Expressions:

Numerical differentiation techniques estimate the derivative of a function at a point x_k by approximating the slope of the tangent line at x_k using values of the function at points near x_k. The approximation of the slope of the tangent line can be done in several ways.

The derivative at x_k is estimated by computing the slope of the line between $f(x_{k-1})$ and $f(x_k)$ is

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

This type of derivative approximation is called backward difference approximation.

Assume that the derivative at x_k is estimated by computing the slope of the line between $f(x_{k+1})$ and $f(x_k)$ is

$$f'(x_k) = \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}$$

This type of derivative approximation is called forward difference approximation.

Assume that the derivative at x_k is estimated by computing the slope of the line between $f(x_{k+1})$ and $f(x_{k-1})$ is

$$f'(x_k) = \frac{f(x_{k+1}) - f(x_{k-1})}{x_{k+1} - x_{k-1}}$$

This type of derivative approximation is called central difference approximation.

The quality of all of these types of derivative computations depends on the distance between the points used to estimate the derivative.

Unit – III (Numerical Techniques)

- diff function:

The diff function computes differences between adjacent values in a vector, generating a new vector with on fewer value. If the diff function is applied to a matrix, it operates on the columns of the matrix as if each column were a vector. A second, optional argument specifies the number of times to recursively apply diff. Each time diff is applied, the length of the vector is reduced in size. A third, optional argument specifies the dimensions in which to apply the function. The forms of diff as follows:

diff(x)

for a vector x, diff returns

[x(2)-x(1), x(3)-x(2), … x(n)-x(n-1).

diff(x)

for a matrix x, diff returns the matrix of column difference

[x(2:m,:) – x(1:m-1,:)]

diff(x,n,dim)

Unit – III (Numerical Techniques)

The general form of diff returns the nth difference function along dimension dim(a scalar). If n >= the length of dim, the diff returns an empty array. We define vectors x, y, and z as follows;

```
x = [0 1 2 3 4 5];
y = [2 3 1 5 8 10];
z = [1 3 5;1 5 10];
```

Then the vector generated by diff(x) is

```
diff(x)
ans =
    1    1    1    1    1
```

The vector generated by diff(y) is

```
diff(y)
ans =
    1   -2    4    3    2
```

If you execute diff twice, the length of the returned vector is 4:

```
diff(y,2)
ans =
   -3    6    -1    -1
```

The diff function can be applied to either dimension of matrix z:

```
diff(z,1,1)
ans =
    0    2    5
diff(z,1,2)
ans =
2    2
4    5
```

An approximate derivative dy can be computed by using diff(y)./diff(x). Note that these values of dy are correct for both the forward difference equation and the backward difference equation. The distinction between the two methods for computing the derivative is determined by the values of the vector xd, which correspond to the derivative dy.

Unit – III (Numerical Techniques)

COMMANDS AND FUNCTIONS	
COMMANDS	MEANING
cftool	Opens the curve fitting graphical user interface
diff	Computes the differences between adjacent values
interp1	Computes linear and cubic interpolation
polyfit	Computes a least-squares polynomial
polyval	Evaluates a polynomial
quad	Computes the integral under a curve
quad1	Computes the integral under a curve

END OF FIRST 3 UNITS

Thank you