ATME College of Engineering

13th KM mile Stone, Mysuru-Kanakapura Rd, Mysuru-28

Department of Mechanical Engineering

BASICS OF MATLAB

(BME657A)

MANUAL
For
VI SEMESTER

2025-2026

Name of the Student: ______________________________

University Seat No. : ________________________________

Semester: _________________***Batch No.*** : ______________

DEPARTMENT OF MECHANICAL ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

ATMECOLLEGE ENGINEERING

13th K.M Mile stone, Mysuru-Kanakapura Road, Mysuru

LABORATORY CERTIFICATE

This is to certify that Mr. /Miss.. Bearing USN... has satisfactorily completed the course of experiments in practical **BASICSOFMATLAB (BME657A)**prescribed by the Visvesvaraya Technological University for the 6th semester B.E course during the year 20........20........

SESSIONAL MARKS AWARDED		
Conduction		
Record & Viva-voce		
Internal Assessment		
Maximum Mark		

Signature of staff **Signature of H.O.D**

Basics of Matlab		Semester	**6th**
Course Code	**BME657A**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2*:0	SEE Marks	50
Total Hours of Pedagogy	12-14 sessions	Total Marks	100
Credits	01	Exam Hours	03
Examination nature (SEE)	**Practical**		

* Additional one hour may be considered for instructions, if required

Course objectives:

- To know about fundamentals of MATLAB tool.
- To provide an overview to program curve fitting & solve Linear and Nonlinear Equations.
- To understand the concept and importance of Fourier transforms.
- To gain knowledge about MATLAB Simulink & solve engineering problems.

Sl.NO	**Experiments**
1	Introduction to MATLAB Programming: Basics of MATLAB Programming, array operations in MATLAB, loops and execution of control, working with files: Scripts and functions, plotting and programming output, examples.
2	
3	Numerical Methods and their applications: Curve Fitting: Straight line fit, Polynomial fit.
4	
5	Numerical Integration and Differentiation: Trapezoidal method, Simpson method.
6	
7	Linear and Nonlinear Equations: Eigen values, Eigen vectors, Solution of linear algebraic equations using Gauss Elimination and LU decomposition, Solution of nonlinear equation in single variable using Gauss-Siedal and Newton-Raphson method.
8	
9	Ordinary Differential Equations: Introduction to ODE's, Euler's method, second order RungaKutta method, MATLAB ode45 algorithm in single variable and multivariable. Transforms: Discrete Fourier Transforms,
10	
11	Application of MATLAB to analyse problems in basic engineering mechanics, mechanical vibrations, control system, statistics and dynamics of different circuits.
12	MATLAB Simulink: Introduction to MATLAB Simulink, Simulink libraries, development of basic models in Simscape Power Systems

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Implement loops, branching, control instruction and functions in MATLAB programming environment.
- Programming for curve fitting, numerical differentiation and integration, solution of linear equations in MATLAB and solve engineering problems.
- Understand implementation of ODE using ode 45 and execute Solutions of nonlinear equations and DFT in MATLAB.
- Simulate MATLAB Simulink examples.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/

course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

Text Books:

1. Agam Kumar Tyagi, “**MATLAB and Simulink for Engineers**”, OXFORD Higher Education.
2. Dr. Shailendra Jain, “**Modelling & Simulation using MATLAB – Simulink**”, Wiley – India.

Reference Books:

1. Won Y.Tang, Wemun Cao, Tae-Sang Ching and John Morris, “**Applied Numerical Methods Using MATLAB**”, A John Wiley & Sons.
2. Steven T. Karris, “**Introduction to Simulink with Engineering Applications**”, Orchard Publications.

INTRODUCTION TO MAT LAB

The name MATLAB stands for **MATrix LABoratory**. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

MATLAB is a high-performance language for technical computing. It integrates *computation*, *visualization*, and *programming* environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated *data structures*, contains built-in editing and *debugging tools*, and supports *object-oriented programming*. These factors make MATLAB an excellent tool for teaching and research.

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

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

Features of MATLAB

Following are the basic features of MATLAB:

- It is a high-level language for numerical computation, visualization and application development.
- It also provides an interactive environment for iterative exploration, design and problem solving.
- It provides vast library of mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration and solving ordinary differentialequations.
- It provides built-in graphics for visualizing data and tools for creating custom plots.
- MATLAB's programming interface gives development tools for improving code quality, maintainability, and maximizing performance.

- It provides tools for building applications with custom graphical interfaces.
- It provides functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET and Microsoft Excel.

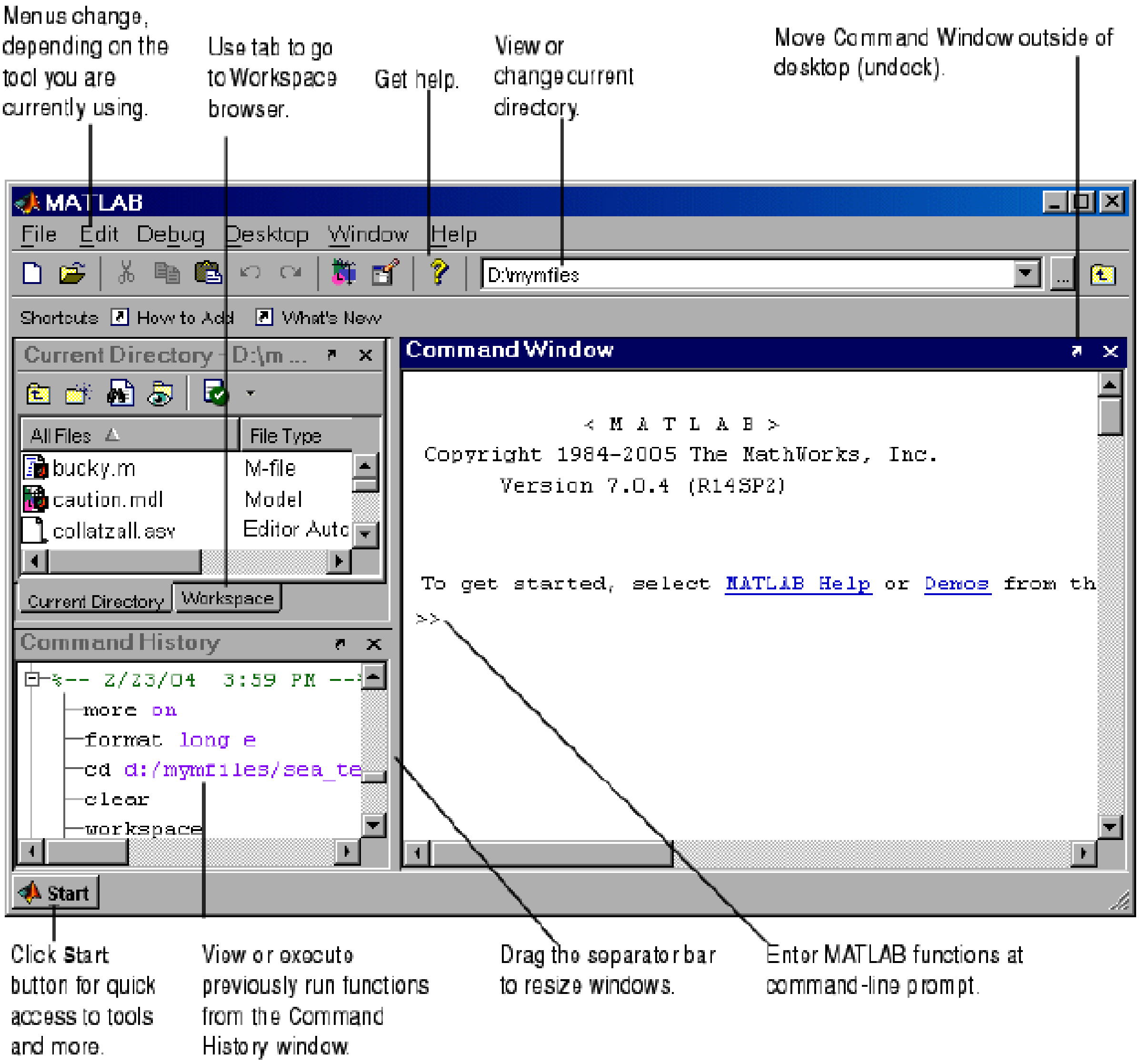
Uses of MATLAB

- MATLAB is widely used as a computational tool in science and engineering encompassing the fields of physics, chemistry, math and all engineering streams. It is used in a range of applications including:
- signal processing and Communications
- image and video Processing
- control systems
- test and measurement
- computational finance
- computational biology

Starting MATLAB

After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut *icon* (MATLAB 17.0.4) on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains *other* windows. The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser
- The Start button



```
          < M A T L A B >
Copyright 1984-2005 The MathWorks, Inc.
      Version 7.0.4 (R14SP2)


To get started, select MATLAB Help or Demos from th
>>
```

```
%-- 2/23/04   3:59 PM --
   more on
   format long e
   cd d:/mymfiles/sea_te
   clear
   workspace
```

Menus change, depending on the tool you are currently using.

Use tab to go to Workspace browser.

Get help.

View or change current directory.

Move Command Window outside of desktop (undock).

Click Start button for quick access to tools and more.

View or execute previously run functions from the Command History window.

Drag the separator bar to resize windows.

Enter MATLAB functions at command-line prompt.

Figure 1.1: The graphical interface to the MATLAB workspace

Desktop Tools

The following tools are managed by the MATLAB desktop, although not all of them appear by default when you first start. If you prefer a command line interface, you can use functions to perform most of the features found in the MATLAB desktop tools. Instructions for using these function equivalents are provided with the documentation for each tool.

- **Command Window** – Run MATLAB functions.
- **Command History** – View a log of the functions you entered in the Command Window, copy them, and execute them.
- **Launch Pad** – Run tools and access documentation for all of your MathWorks products.
- **Current Directory Browser** – View MATLAB files and related files, and perform file operations such as open, and find content.
- **Help Browser** – View and search the documentation for the full family of MATLAB

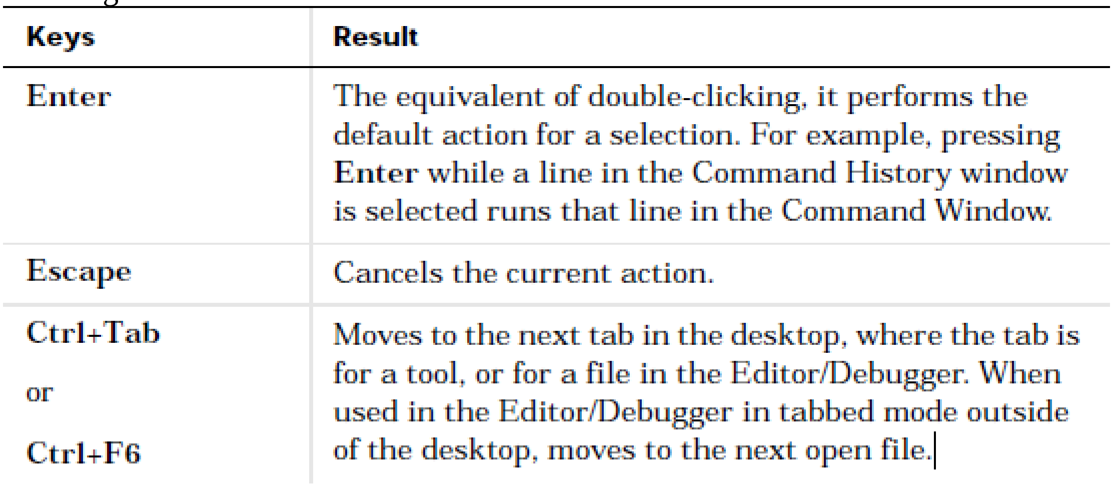
products.

- **Workspace Browser** – View and make changes to the contents of the workspace.
- **Array Editor** – View array contents in a table format and edit the values.
- **Editor/Debugger** – Create, edit, and debug M-files (files containing MATLAB functions).

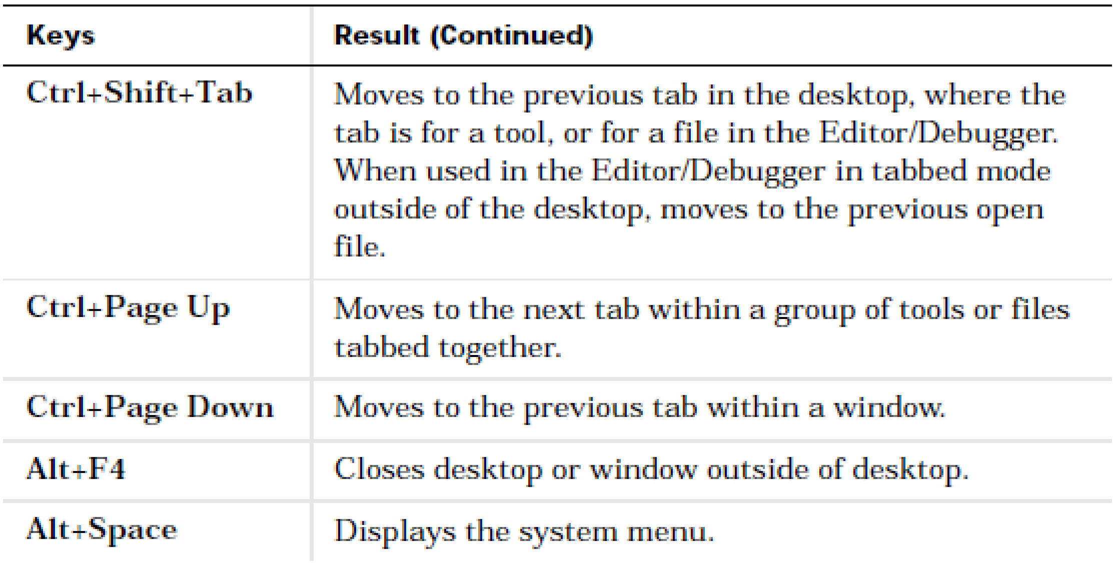
Keyboard Shortcuts and Accelerators

You can access many of the menu items using keyboard shortcuts or accelerators for your platform, such as using **Ctrl+X** to perform a **Cut** on Windows platforms, or **Alt+F** to open the **File** menu. Many of the shortcuts and accelerators are listed with the menu item. For example, on Windows platforms, the **Edit** menu shows **Cut Ctrl+X**, and the **File** menu shows the **F** in **File** underlined, which indicates that **Alt+X** opens it. Many standard shortcuts for your platform will work but are not listed with the menu items.

Following are some additional shortcuts that are not listed on menu items.



Keys	Result
Enter	The equivalent of double-clicking, it performs the default action for a selection. For example, pressing Enter while a line in the Command History window is selected runs that line in the Command Window.
Escape	Cancels the current action.
Ctrl+Tab or Ctrl+F6	Moves to the next tab in the desktop, where the tab is for a tool, or for a file in the Editor/Debugger. When used in the Editor/Debugger in tabbed mode outside of the desktop, moves to the next open file.



Keys	Result (Continued)
Ctrl+Shift+Tab	Moves to the previous tab in the desktop, where the tab is for a tool, or for a file in the Editor/Debugger. When used in the Editor/Debugger in tabbed mode outside of the desktop, moves to the previous open file.
Ctrl+Page Up	Moves to the next tab within a group of tools or files tabbed together.
Ctrl+Page Down	Moves to the previous tab within a window.
Alt+F4	Closes desktop or window outside of desktop.
Alt+Space	Displays the system menu.

.

MATLAB Windows

Window	Purpose
Command Window	Main window, enters variables, runs programs
Figure Window	Contains output from graphic commands
Editor Window	Creates and debugs script and function files
Help Window	Provides help information
Launch Pad Window	Provides access to tools, demos, and documentation
Command History Window	Logs commands entered in the Command Window
Workspace Window	Provides information about the variables that are used
Current Directory Window	Shows the files in the current directory

Working in the Command Window

Notation	Purpose
>>	Command Prompt
↵ (Enter key)	Out put of the command is executed
↑ ↓ (Upper and down arrow keys)	Recalled the previously typed command to the command prompt
… (Three periods)	Command is continued to the next line
; (Semicolon)	Out put of the command is not displayed
% (Enter key)	It indicates that the line is designated as a comment
clc	clears the command window
clear	Removes all the variables from the memory
who	Displays a list of the variables currently in the memory
whos	Displays a list of the variables currently in the memory and their size together with information about their bytes and class

Lab Program – 1&2

Introduction to MATLAB Programming: Basics of MATLAB Programming, array operations in MATLAB, loops 2 and execution of control, working with files: Scripts and functions, plotting and programming output, examples.

MATLAB environment behaves like a super-complex calculator. You can enter commands at the >> command prompt.

MATLAB is an interpreted environment. In other words, you give a command and MATLAB executes it right away.

Commonly used Operators and Special Characters

MATLAB supports the following commonly used operators and special characters:

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

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

Hands on Practice

Type a valid expression, for example,

```
5 + 5
```

And press ENTER

When you click the Execute button, or type Ctrl+E, MATLAB executes it immediately and the result returned is:

```
ans = 10
```

Let us take up few more examples:

```
3 ^ 2       % 3 raised to the power of 2
```

When you click the Execute button, or type Ctrl+E, MATLAB executes it immediately and the result returned is:

```
ans = 9
```

Another example,

sin(pi /2) % sine of angle 90^{0}

When you click the Execute button, or type Ctrl+E, MATLAB executes it immediately and the result returned is:

```
ans = 1
```

Semicolon (;) indicates end of statement. However, if you want to suppress and hide the MATLAB output for an expression, add a semicolon after the expression.

For example,

```
x = 3;
```

```
y = x + 5
```

When you click the Execute button, or type Ctrl+E, MATLAB executes it immediately and the result returned is:

```
y = 8
```

Some more Examples

```
>> 7+8/2  ← Type and press enter
 ( →  8/2 is executed first )
   ans  = 11
```

```
>> (7+8)/2 ← Type and press enter
→  7+8 is executed first
    ans = 7.5000
```

```
>> 4 + 5/3 + 2
→  5/3 is executed first
    ans =  7.6667
```

```
>> 5 ^3/2
→  5^3 is executed first, /2 is executed next
   ans =  62.5000
```

```
>> 27^(1/3)+32^0.2
→ 1/3 is executed first, 27^(1/3) and 32^0.2 are executed next, and + is executed
last
```

```
Ans = 5
```

```
>> 27^1/3+32^0.2
→ 27^1/3 and 32^0.2 are executed first, 1/3 is executed next, and + is executed
last
Ans =  11
```

```
>> 0.7854 (0.7854)^3/(1*2*3)+0.7854^5/(1*2*3*4*5)...
type three periods … and press enter to continue the expression on the next line
-0.7854^7/(1*2*3*4*5*6*7)
Ans =  0.7071
```

The format Command

By default, MATLAB displays numbers with four decimal place values. This is known as ***short format***.

However, if you want more precision, you need to use the **format** command.

The **format long** command displays 16 digits after decimal.

For example:

```
format long
x = 7 + 10/3 + 5 ^ 1.2
```

MATLAB will execute the above statement and return the following result:

```
x = 17.231981640639408
```

```
format short
x = 7 + 10/3 + 5 ^ 1.2
```

MATLAB will execute the above statement and return the following result:

```
x = 17.2320
```

The **format bank** command rounds numbers to two decimal places. For example,

```
format bank
daily_wage = 177.45;
weekly_wage = daily_wage * 6
```

MATLAB will execute the above statement and return the following result:

```
weekly_wage = 1064.70
```

MATLAB displays large numbers using exponential notation.

The **format short e** command allows displaying in exponential form with four decimal places plus the exponent. For example,

```
format short e
4.678 * 4.9
```

MATLAB will execute the above statement and return the following result:

```
ans = 2.2922e+01
```

The **format long e** command allows displaying in exponential form with four decimal places plus the exponent. For example,

```
format long e
x = pi
```

MATLAB will execute the above statement and return the following result:

```
x = 3.141592653589793e+00
```

The **format rat** command gives the closest rational expression resulting from a calculation. For example,

```
format rat
4.678 * 4.9
```

MATLAB will execute the above statement and return the following result:

```
Ans =2063/90
```

Some more Examples

Command	Description	Examples
format short	Fixed-point with 4 decimal digits	>> format short >> 290/7 **ans =41.4286**
format long	Fixed-point with 14 decimal digits	>> format long >> 290/7 **Ans = 41.42857142857143**
format short e	Scientific notation with 4 decimal digits	>> format short e >> 290/7 **Ans = 4.1429e+001**
format long e	Scientific notation with 15 decimal digits	>> format long e >> 290/7 **Ans = 4.142857142857143e+001**
format short g	Best of 5-digit fixed or floating point	>> format short g >> 290/7 **Ans = 41.429**
format long g	Best of 15-digit fixed or floating point	>> format long g >> 290/7 **Ans = 41.4285714285714**
format bank	Two decimal digits	>> format bank >> 290/7 **Ans = 41.43**

Using the sqrt built-in function

```
>> sqrt(64)  ← Argument is a number

ans =      8
```

```
>> sqrt(50+14*3) ← Argument is an expression

ans =     9.5917
```

```
>> sqrt(54+9*sqrt(100)) ← Argument includes a function

ans =     12
```

```
>> (15+600/4)/sqrt(121) ← function is included in an expression

ans =    15
```

Elementary math functions

cos(x) - Cosine	abs(x) Absolute value
sin(x) - Sine	sign(x) - Signum function
tan(x) - Tangent	max(x) - Maximum value
acos(x) - Arc cosine	min(x) - Minimum value
asin(x) - Arc sine	ceil(x) - Round towards + *infinite*
atan(x) - Arc tangent	floor(x) - Round towards - *infinite*
exp(x) - Exponential	round(x) - Round to nearest integer
sqrt(x) - Square root	rem(x,number) - Remainder after division
log(x) - Natural logarithm	angle(x) - Phase angle
log10(x) - Common logarithm	conj(x) - Complex conjugate

Function	**Description**	**Examples**
exp(x)	exponential	>> exp(5) **Ans = 148.4132**
sqrt(x)	Square root	>> sqrt(81) **Ans = 9**
abs(x)	absolute value	>> abs(-24) **Ans = 24**
log(x)	Natural logarithm. Base e logarithm(ln)	>> log(1000) **Ans = 6.9078**
log10(x)	Base 10 logarithm	>> log10(1000) **Ans = 3**
factorial(x)	The factorial function x! (x must be a positive integer)	>> factorial(5) **ans = 120**

rem(x,number)	Finding the reminder of given value x.	>>rem(11,2)
		ans=1

Trigonometric math functions

Function	**Description**	**Examples**
sin(x)	Sine of angle x (x in radians)	>> sin(pi/6) **Ans = 0.5000**
sin(x)	Sine of angle x (x in Degrees)	>> sind(30) **Ans = 0.5000**
cos(x)	Cosine of angle x (x in radians)	>> cos(pi/6) **Ans** = 0.8660
tan(x)	tangent of angle x (x in radians)	>> tan(pi/6) **Ans = 0.5774**
cot(x)	cotangent of angle x (x in radians)	>> cot(pi/6) **Ans = 1.7321**

Rounding functions

Function	Description	Examples
round(x)	Round to the nearest integer	>> round(17/5) **Ans = 3**
fix(x)	Round towards zero	>> fix(13/5) **Ans = 2**
ceil(x)	Round towards infinity	>> ceil(11/5) **Ans = 3**
floor(x)	Round towards minus infinity	>> floor(-9/4) **Ans = - 3**
rem(x,y)	Returns the remainder after x is divided by y	>> rem(13,5) **Ans = 3**
sign(x)	Signum function. Returns 1 if x>0, -1 if x<0, and o if x=0	>> sign(5) **Ans = 1**

Example: the value of the expression $A = e^{-a}\sin(x) + 10\sqrt{y}$ for a = 5, x = 2, and y = 8 is computed by

```
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y = 28.2904
```

The Assignment operator
Hierarchy of arithmetic operations

Precedence	Mathematical operations
First	The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward.
Second	All exponentials are evaluated, working from left to right
Third	All multiplications and divisions are evaluated, working from left to right
Fourth	All additions and subtractions are evaluated, starting from left to right

Example:

1) >> x=15   ← The number 15 assigned to the variable x

x = 15   ← MATLAB displays the variable and its assigned value

>> x=3*x - 12

← A new value is assigned to x. The new value is 3 times the previous value of x minus 12.

Ans x = 33

```
 >> a=12   ← Assign 12 to a
a = 12
>> b=4      ← Assign 4 to b
b = 4
>> c=(a-b)+40-a/b*10 ← Assign the value of the expression on the right-hand
side to the variable c.
     c =18
```

```
 >> a=12;  ← The variables a, b and c are defined but are not displayed since a
>> b=4;          semicolon is typed  at the end of each statement
>> c=(a-b)+40-a/b*10;
>> c             ← The value of the variable c is displayed by typing the name of the
                     variable
    c =18
```

```
>> a=12, b=4; c=(a-b)+40-a/b*10  ← The variable b is not displayed because
 a = 12                                      semicolon is typed at the end of the assignment
 c = 18
```

A variable that already exists can be reassigned a new value.

Example: >> abb=72;

```
>> abb=9;
>> abb
abb = 9
```

Once a variable is defined it can be used as an argument in functions.

Example:
```
>> x=0.75;
>> E=sin(x)^2+cos(x)^2
E = 1
```

Rules about variable names:

1. Can contain letters, digits, and the underscore character.
2. Must begin with a letter.
3. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. For examples, AA, Aa, aA and aa are the names of four different variables.
4. Avoid using the names of a built in function for a variable i.e. sin, exp etc.

Predefined variables: i.e. pi, esp, inf, I or j.

Show that at x=π/5.

Solution:
```
x = pi/5;

>> LHS=cos(x/2)^2
LHS =    0.9045
```

```
>> RHS=(tan(x)+sin(x))/(2*tan(x))
RHS =    0.9045
```

Creating Vectors

A vector is a one-dimensional array of numbers. MATLAB allows creating two types of vectors:

Row vectors

Column vectors

Variable name=[type vector element]

Row vector: To create a row vector type the elements with a space or a comma between the elements inside the square brackets.

Column vector: To create a column vector type the left square bracket [and then enter the elements with a semicolon between them, or press the Enter key after each element. Type the right square bracket] after the last element.

Row vectors are created by enclosing the set of elements in square brackets, using space or comma to delimit the elements.

For example,

```
r = [7 8 9 10 11]
```

MATLAB will execute the above statement and return the following result:

```
r =
Columns 1 through 4
7    8     9     10
Column 5
```

```
 11
```

Another example

```
r = [7 8 9 10 11];
t = [2, 3, 4, 5, 6];
res = r + t
```

MATLAB will execute the above statement and return the following result:

```
res =
Columns 1 through 4
9    11     13    15
Column 5
17
```

Column vectors are created by enclosing the set of elements in square brackets, using semicolon (;) to delimit the elements.

```
c = [7; 8; 9; 10; 11]
```

MATLAB will execute the above statement and return the following result:

```
c =
         7
         8
         9
        10
        11
```

Creating a vector with constant spacing by specifying the first term, the spacing, and the last term

Variable_name= [m:q:n] or variable_name= m:q:n

Examples :

```
>> x=[1:2:13]   First element 1, spacing 2, last element 13
x =
     1     3     5     7     9    11    13
```

```
>> y=[1.5:0.1:2.1]        First element 1.5, spacing 0.1, last element 2.1
 y =
    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000    2.1000
```

```
>> z=[-3:7]              First element -3, last element 7. If spacing is omitted, the
default is 1
 z =
    -3    -2    -1     0     1     2     3     4     5     6     7
```

```
>> xa=[21:-3:6]        First element 21, spacing -3, last element 6
xa =
    21    18    15    12     9     6
```

Creating a vector with constant spacing by specifying the first and last terms and the number of terms

Variable_name=linspace(xi,xf,n)

Examples :

```
>> va= linspace(0,8,6)        6 elements, first element 0, last element 8
 va =  0    1.6000    3.2000    4.8000    6.4000    8.0000
```

```
>> vb= linspace(30,10,11)            11 elements, first element 30, last element
10
vb =    30    28    26    24    22    20    18    16    14    12    10
```

>> u=linspace(49.5,0.5) **when the number of elements is omitted, the default is 100**

>> b=[7 2 76 33 8 **The enter key is pressed a new line is entered**

```
        1 98 6 25 6
        5 54 68 9 0]
```

```
u =  Columns 1 through 9
49.5000   49.0051   48.5101   48.0152   47.5202   47.0253   46.5303   46.0354
45.5404
Ans
……………
Columns 91 through 99
 4.9545     4.4596     3.9646     3.4697     2.9747     2.4798     1.9848     1.4899
0.9949
```

```
Column 100
 0.5000
```

Creating a Two-dimensional array (matrix)

In MATLAB, a matrix is created by entering each row as a sequence of space or comma separated elements, and end of a row is demarcated by a semicolon. For example, let us create a 3-by-3 matrix as:

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

MATLAB will execute the above statement and return the following result:

```
m =
1   2   3
4   5   6
7   8   9
```

```
>> cd=6; e=3; h=4;          Three variables are defined
 >> mat=[e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]
   Elements are defined  by Mathematical expressions
mat =
     3.0000   24.0000    0.5000
   16.0000    1.6330   14.0000
```

```
      1     3      5     7       9     11
       0     5     10    15    20    25
     10    20    30    40   50    60
     67     2    43    68     4    13
```

The zeros, ones and eye Commands

The zeros(m,n) and the ones(m,n) commands create a matrix with m rows and n columns, in which all the elements are the numbers o and 1 respectively. The eye(n) command creates a square matrix with n rows and n columns in which the diagonal elements are equal to 1, and the rest of the elements are 0. This matrix is called the identity matrix. Examples are:

```
>> zr=zeros(3,4)
 zr =
      0     0     0     0
    0     0     0     0
    0     0     0     0
```

```
>> a=zeros(4)
 a =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
>> ne=ones(4,3)
 ne =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

```
>> idn=eye(5)
 idn =
    1     0     0     0     0
    0     1     0     0     0
    0     0     1     0     0
    0     0     0     1     0
    0     0     0     0     1
```

The Transpose Operator

The transpose operator is applied by typing a single quote ‘ following the variable to be transposed.

Examples are:

```
>> aa=[3 8 1]         Define a row vector aa
aa =
     3     8     1
```

```
>> bb=aa‘             Define a row vector bb as the transpose of vector aa
bb =
     3
     8
     1
```

```
>> D=C'                   Define a matrix D as the transpose of matrix C
D =
     2    21    41
    55     5    64
    14    32     9
     8    11     1
```

Array Addressing Vector:

```
>> S=[35 46 78 23 5 14 81 3 55]          Define a vector
 S =
    35    46    78    23     5    14    81     3    55
```

```
>> S(4)               Display the fourth element
 ans =
    23
```

```
>> S(6)=273          Assign a new value to the sixth element
 S =
   35    46    78    23     5   273    81     3    55
```

```
>> S(2)+S(8)         Use the vector elements in mathematical expressions
 ans = 49

 >>  S(5)^S(8)+sqrt(S(7))             Use  the  vector  elements  in  mathematical
expressions
 ans = 134
```

Matrix operations

```
>> A=[3 11 6 5; 4 7 10 2; 13 9 0 8]     Create a 3x4 matrix
 A =
    3    11     6     5
    4     7    10     2
   13     9     0     8

>> A(3,1)=20         Assign a new value to the (3,1) element
 A =
    3    11     6     5
    4     7    10     2
   20     9     0     8

>> A(2,4)-A(1,2)       Use elements in a mathematical expression
```

```
ans = -9
```

Using Colon : in addressing Arrays

```
>> v=[4 15 8 12 34 2 50 23 11]            A vector v is created
 v =
     4    15     8    12    34     2    50    23    11


>> u=v(3:7)  A vector u is created from the elements 3 through 7 of vector v
 u =
     8    12    34     2    50       A vector u is created from the elements 3
through 7 of vectors v
```

For a matrix

A(:,n)	Refers to the elements in all the rows of column n of the matrix A
A(n,:)	Refers to the elements in all the columns of row n of the matrix A
A(:,m:n)	Refers to the elements in all the rows between columns m and n of the matrix A
A(m:n,:)	Refers to the elements in all the columns between rows m and n of the matrix A
A(m:n,p:q)	Refers to the elements in rows m through n and columns p through q of the matrix A

Examples

```
>> A=[1 3 5 7 9 11; 2 4 6 8 10 12; 3 6 9 12 15 18; 4 8 12 16 20 24;
     5 10 15 20 25 30]
 A =
     1    3    5    7    9   11
    2    4    6    8   10   12
    3    6    9   12   15   18
    4    8   12   16   20   24
    5   10   15   20   25   30
```

Using a colon in addressing arrays

Examples:

```
>> B=A(:,3)          Define a column vector B from the elements in all the rows of
column 3 in Matrix A
 B =
     5
    6
    9
   12
   15
```

>> **C=A(2,:)** Define a row vector C from the elements in all the columns of row 2 in matrix A

```
 C =
     2    4    6    8   10   12
```

>> **E=A(2:4,:)** Define a matrix E from the elements in rows 2 through 4 and all the columns in matrix A

```
E =
     2    4    6    8   10   12
     3    6    9   12   15   18
     4    8   12   16   20   24
```

>> **F=A(1:3,2:4)** Create a matrix f from the elements in rows 1 through 3 and columns 2 through 4 In matrix A

```
F =
    3    5    7
    4    6    8
    6    9   12
```

>> **V=4:3:34** Create a vector V with 11 elements

```
 V =
     4    7   10   13   16   19   22   25   28   31   34
```

>> **u=V([3,5,7:10])** Create a vector u from the 3rd, the 5th, and 7th through 10 elements of V

```
 u =
    10   16   22   25   28   31
```

```
>> B=[5 7 2]         Define vector B with 3 elements
 B =
     5     7     2

>> B(8)=4           Assign a value to the 8th element
B =
    5     7     2     0     0     0     0     4
  (MATLAB assigns zeros to the 4th through 7th elements )

 >> P(5)=24
P =
    0     0     0     0    24
```

```
>> A=[3 8 1 24]          Define vector A with 4 elements
A =
    3     8     1    24
>> B=4:3:16         Define vector B with 5 elements
B =
    4     7    10    13    16
>> C=[A B]         Define a new vector C by appending A and B
C =
    3     8     1    24     4     7    10    13    16
>> D=[A';B']        Create a new column vector D by appending A’ and B’
 D =
     3
    8
    1
   24
```

```
    4
    7
   10
```

```
>> A=[1 2 3 4; 5 6 7 8]        Define a 2 x 4 matrix
 A =
      1     2     3     4
     5     6     7     8


>> A(3,:)=[10:4:22]            Add the vector 10 14 18 22 as the third row of A
 A =
      1     2     3     4
     5     6     7     8
    10    14    18    22
 >> B=eye(3)
 B =
      1     0     0
     0     1     0
     0     0     1
C=[A B]      Append the matrix B to matrix A. The number of rows in B and A
must  be the same
 C =
      1     2     3     4     1     0     0
     5     6     7     8      0     1     0
    10   14   18    22     0     0     1
  >> A=[3 6 9; 8 5 11]
  A =
      3     6     9
     8     5    11
  >> A(4,5) =17       Assign a value to the (4,5) element

 A =
      3     6     9     0     0   MATLAB changes the matrix size to 4X5, and
     8     5    11     0     0   assigns zeros to the new elements
     0     0     0     0     0
     0     0     0     0    17

 >> Q(3,4)=15          Assign a value to the (3,4) element of a new matrix
  Q =
      0     0     0     0
     0     0     0     0
     0     0     0    15
```

Deleting Elements

```
 >> kt=[2 8 40 3 55 23 15 75 80]           Define a vector with 10 elements
 kt =
     2     8    40     3    55    23    15    75    80
 >> kt(6)=[ ]          Eliminate the sixth element
 kt =
     2     8    40     3    55    15    75    80
 >> kt(3:6)=[ ]       Eliminate elements 3 through 6
 kt =
     2     8    75    80
 >> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 31]        Define 3x5 matrix
 mtr =
      5    78     4    24     9
      4     0    36    60    12
     56    13     5    89    31
>> mtr(:,2:4)=[ ]       Eliminate all the rows of columns 2 through 4
 mtr =
      5     9
      4    12
     56    31
```

Built-in-functions for handling arrays

Function	Description	Examples
length(A)	Returns the number of elements In the vector A	>> A=[5 9 2 4]; >> length(A) **ans = 4**
Size(A)	Returns a row vector [m,n], where m and n are the size m x n of the array A	>> A=[6 1 4 0 12; 5 19 6 8 2] A = 6 1 4 0 12 5 19 6 8 2 >> size(A) **ans =** **2 5**
Reshape(A,m,n)	Rearrange a matrix A that has r rows and s columns to have m rows and n columns. R	>> A=[5 1 6; 8 0 2] A = 5 1 6

	times s must be equal to m times n	8 0 2 >> B=reshape(A,3,2) = 5 0 8 6 1 2
diag(v)	When v is a vector, creates a square matrix with the elements of v in the diagonal	>> v=[7 4 2]; >> A=diag(v) A = 7 0 0 0 4 0 0 0 2
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A	>> A=[1 2 3; 4 5 6; 7 8 9] A = 1 2 3 4 5 6 7 8 9 >> vec=diag(A) vec = 1 5 9

Mathematical Operations with Arrays

```
>> A=[5 -3 8; 9 2 10]
A =
    5   -3    8
    9    2   10
```

Define two 2X3 matrices A and B

```
 >> B=[10 7 4; -11 15 1]
B =
    10     7     4
   -11    15     1
 >> A-B   ← Subtracting matrix B from matrix A
ans =
    -5   -10     4
    20   -13     9
 >> C=A+B ← Define a matrix C that is equal to A + B
C =
    15     4    12
    -2    17    11
 >> C-8 ← The number 8 is subtracted from the matrix C
ans =
     7    -4     4
   -10     9     3
```

```
>> A=[1 4 2; 5 7 3; 9 1 6; 4 2 8] ← Define a 4X3 matrix A
A =
     1     4     2
     5     7     3
     9     1     6
     4     2     8
 >> B=[6 1; 2 5; 7 3] ← Define a 3X2 matrix B
 B =
      6     1
     2     5
     7     3
 >> C=A*B ← Multiply matrix A by matrix B and assign the result to
variable C
 C =
    28   27
   65   49
   98   32
   84   38
 >> D=B*A ← Trying to multiply B by A, B*A, gives an error since the
number of columns in B is 2 and the number of rows in A is 4
??? Error using ==> *
Inner matrix dimensions must agree.
```

```
>> F=[1 3; 5 7]
F =
     1     3                    Define two 2 X 2 matrices F and G
     5     7
 >> G=[4 2; 1 6]
 G =
      4     2
     1     6
 >> F*G  ← Multiply F*G
ans =
     7    20
    27    52
 >> G*F  ← Multiply G*F
ans =
    14    26
    31    45
```

Note: The answer of F*G is not the same as the answer G*F

```
>> AV=[2 5 1] ← Define a three-element column vector AV
AV =
     2     5     1
>> BV=[3; 1; 4] ← Define a three-element column vector BV
BV =
     3
     1
     4
 >> AV*BV ← Multiply AV by BV. The answer is a scalar. (dot product of
two vectors)
ans =
    15
 >> BV*AV ← Multiply AV by BV. The answer is a 3 X 3 matrix
ans =
     6    15     3
     2     5     1
     8    20     4
```

```
>> A=[2 5 7 0; 10 1 3 4; 6 2 11 5] ← Define a 3 X 4 matrix A
A =
     2     5     7     0
    10     1     3     4
     6     2    11     5
 >> b=3← Assign the number 3 to the variable b
b =
     3
```

```
>> b*A ← Multiply the matrix A by b. This can be done by either typing
b*A or A*b
ans =
     6    15    21     0
    30     3     9    12
    18     6    33    15

>> A*b
ans =
     6    15    21     0
    30     3     9    12
    18     6    33    15

>> C=A*5 ← Multiply the matrix A by 5 and assign the result to a new
variable C
C =
    10    25    35     0
    50     5    15    20
    30    10    55    25

>>C= 5*A
C =
    10    25    35     0
    50     5    15    20
    30    10    55    25
```

Inverse of the Matrix

```
>> A=[7 3 8; 4 11 5; 6 7 10]
A =
     7     3     8
     4    11     5
     6     7    10
 >> I=eye(3)
I =
     1     0     0
     0     1     0
     0     0     1
 >> A*I
ans =
     7     3     8
     4    11     5
     6     7    10
 >> I*A
ans =
     7     3     8
     4    11     5
     6     7    10
```

```
>> A=[2 1 4; 4 1 8; 2 -1 3] ← Creating the matrix A
A =
      2     1     4
     4     1     8
     2    -1     3
>> B=inv(A) ← Use the inv function to find the inverse of A and assign it to B
 B =
    5.5000   -3.5000    2.0000
    2.0000   -1.0000         0
   -3.0000    2.0000   -1.0000
```

Use matrix operations to solve the following system of linear equations

Solution: Using the rules of linear algebra, the above system of equations can be written in the matrix form A X = B or in the form X C = D:

The solution of both forms is shown below:

```
>> A=[4 -2 6; 2 8 2; 6 10 3]; ← Solving the form A X = B
>> B=[8; 4; 0];
>> X=A\B   ← Solving by using left division X = A \ B
X =
   -1.8049
    0.2927
    2.6341

>> Xb= inv(A)*B ← Solving by using the inverse of A ( i.e. X= B)


Xb =
   -1.8049
    0.2927
    2.6341
>> C=[4 2 6; -2 8 10; 6 2 3]; ← Solving the form X C = D
>> D=[8 4 0];
>> Xc= D/C ← Solving by using right division X= D / C
Xc =
   -1.8049    0.2927    2.6341

>> Xd=D*inv(C) ← Solving by using the inverse of C, X=D.
Xd =
   -1.8049    0.2927    2.6341
```

The M Files

MATLAB allows writing two kinds of program files:

Scripts - script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.

Functions - functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.

You can use the MATLAB editor or any other text editor to create your **.m** files.

In this section, we will discuss the script files. A script file contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line

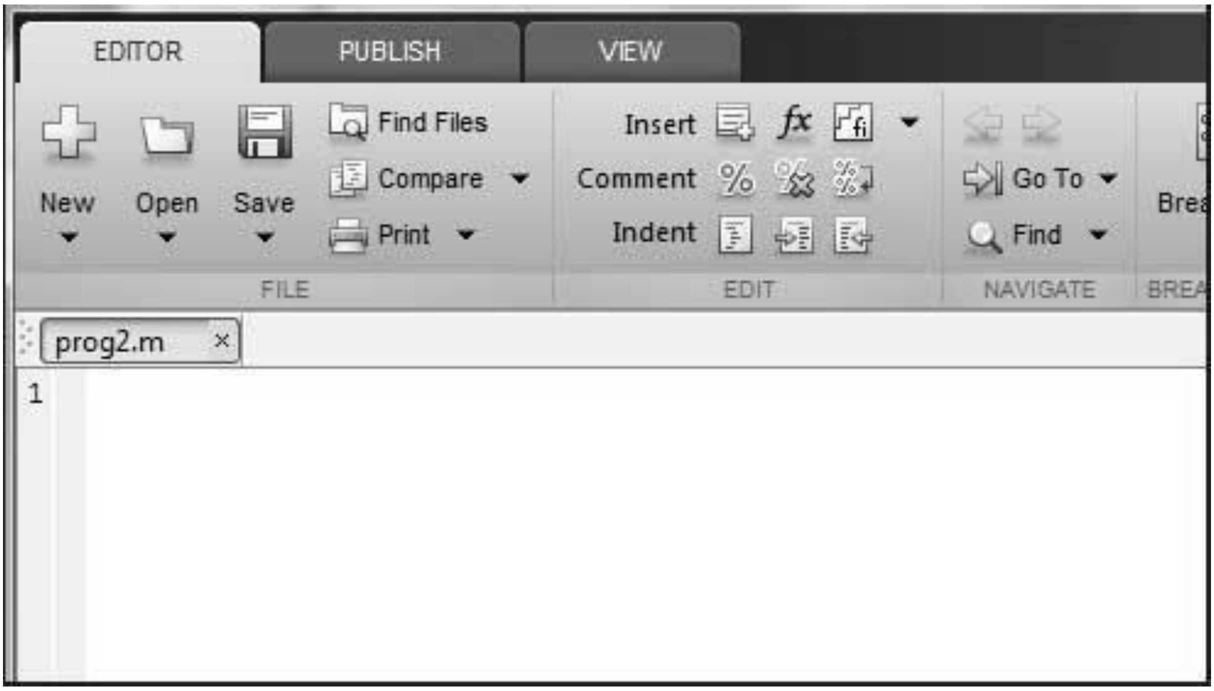
To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

- Using the command prompt
- Using the IDE

If you are using the **command prompt**, type edit in the command prompt. This will open the editor. You can directly type edit and then the filename (with .m extension)

```
edit
Or
edit <filename>
```

Alternatively, if you are using the **IDE**, choose NEW -> Script. This also opens the editor and creates a file named Untitled. You can name and save the file after typing the code.



Type the following code in the editor:

```
No of Students = 6000;
Teaching Staff = 150;
Non Teaching Staff = 20;
Total = No of Students + Teaching Staff ...
+ Non Teaching Staff;
disp(Total);
```

After creating and saving the file, you can run it in two ways:

- Clicking the Run button on the editor window or
- Just typing the filename (without extension) in the command prompt: >> prog1

The command window prompt displays the result:

```
6170
```

Example 1

Create a script file, and type the following code:

```
a = 5; b = 7;
c = a + b
d = c + sin(b)
e = 5 * d
f = exp(-d)
```

When the above code is compiled and executed, it produces the following result:

```
c = 12
d =  12.6570
e =  63.2849
f = 3.1852e-06
```

Example 2

Create a script file with the following code:

```
str = 'Hello World!'
n = 2345
d = double(n)
un = uint32(789.50)
rn = 5678.92347
c = int32(rn)
```

When the above code is compiled and executed, it produces the following result:

```
=
Hello World!
n =
2345
d =
2345
un =
790
rn =
5.6789e+03
c =
5679
```

Example 3

The following examples show the use of arithmetic operators on scalar data. Create a script file with the following code:

```
a = 10;
b = 20;
c = a + b
d = a - b
e = a * b
f = a / b
g = a \ b
x = 7;
```

```
y = 3;
z = x ^ y
```

When you run the file, it produces the following result:

```
c = 30
d =  -10
e =  200
f = 0.5000
g = 2
z = 343
```

Example 4

Create a script file and type the following code:

```
a = 100;
b = 200;
if (a >= b)
max = a
else
max = b
end
```

When you run the file, it produces following result:

```
max = 200
```

Flow control (Loops)

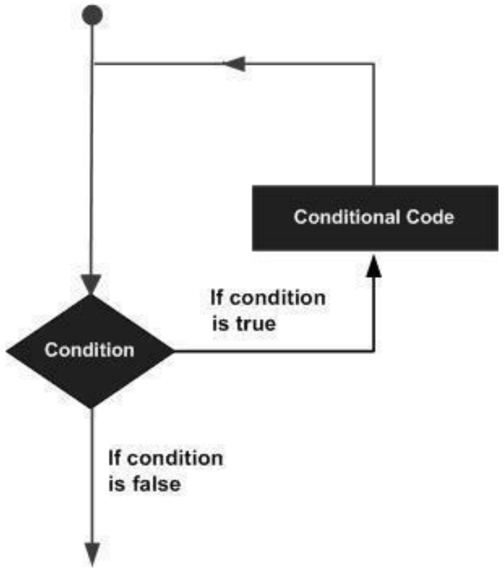
There may be a situation when you need to execute a block of code several times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The drawing shows the general form of a loop statement for most programming languages.

Matlab provides various types of loops to handle looping requirements including: while loops, for loops, and nested loops. If you are trying to declare or write your own loops, you need to make sure that the loops are written as scripts and not directly in the Command Window.

To start a new script, locate the button in the upper left corner of the window labeled **New Script**.

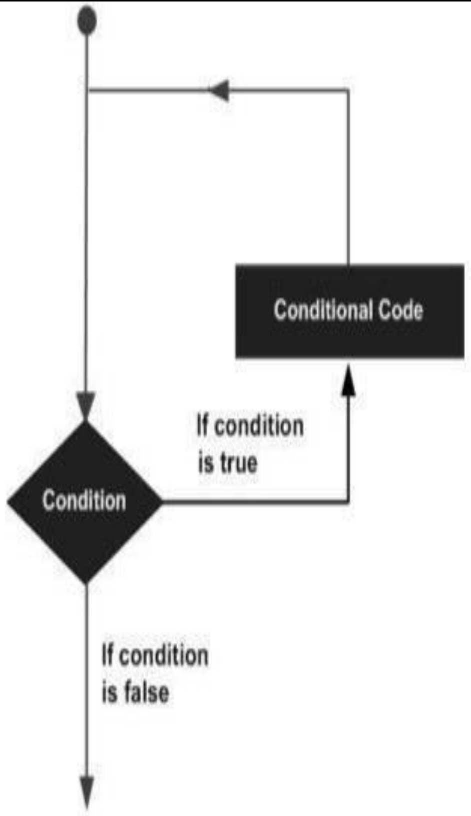


MAT LAB has five flow control statements

- If statement

- Switch statement
- For loops
- While loops
- Break statements

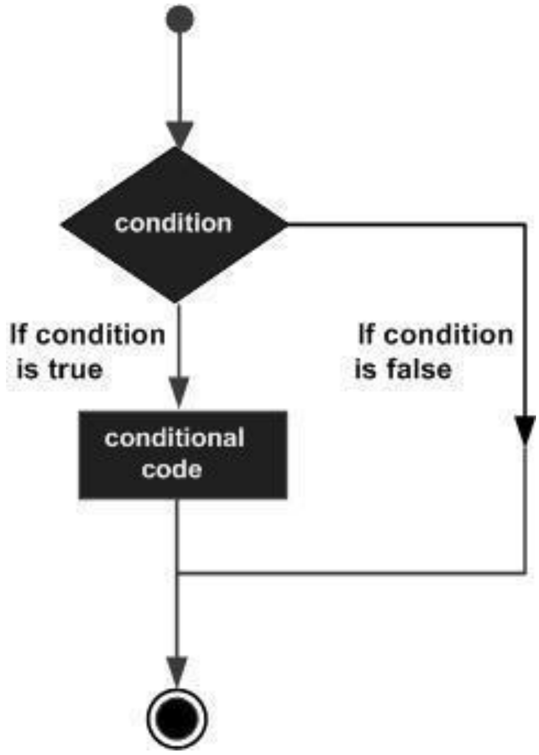
MATLAB provides following types of loops to handle looping requirements

Loop Type	Description	
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.	 Conditional Code If condition is true Condition If condition is false
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.	
nested loops	You can use one or more loops inside any another loop.	
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.	

Decision making Structures

Decision making structures require that the programmer should specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



MATLAB provides following types of decision making statements.

if ... end statement	An if ... end statement consists of a boolean expression followed by one or more statements.
if...else...end statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

If... elseif...elseif...else...end statements	An if statement can be followed by one (or more) optional elseif... and an else statement, which is very useful to test various conditions.
nested if statements	You can use one if or elseif statement inside another if or elseif statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).

if….end Statement

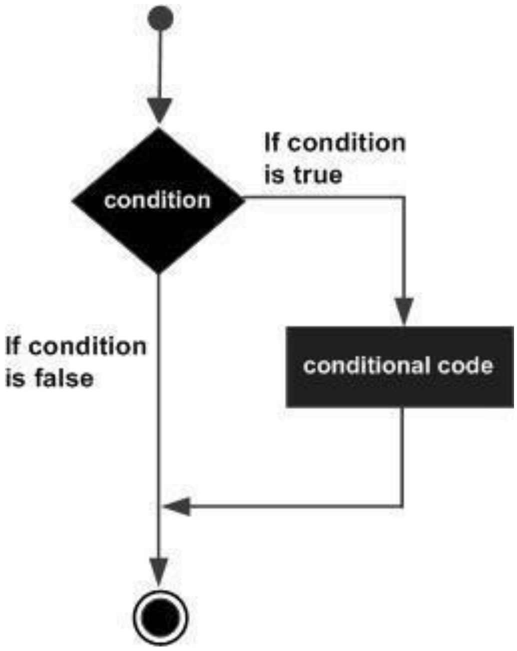
An **if ... end** statement consists of an **if** statement and a boolean expression followed by one or more statements. It is delimited by the **end** statement.

Syntax

The syntax of an if statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statements>
end
```

If the expression evaluates to true, then the block of code inside the if statement will be executed. If the expression evaluates to false, then the first set of code after the end statement will be executed.



Example

Create a script file and type the following code:

```
a = 10;
% check the condition using if statement
if a < 20
% if condition is true then print the following
fprintf('a is less than 20\n' );
end
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result:

```
a is less than 20
value of a is : 10
```

if...else...end Statement

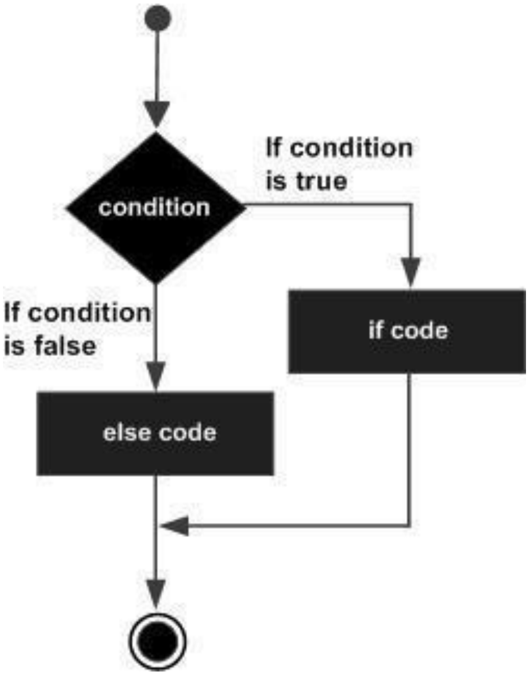
An if statement can be followed by an optional else statement, which executes when the expression is false.

Syntax

The syntax of an if...else statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statement(s)>
else
<statement(s)>
% statement(s) will execute if the boolean expression is false
end
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.



Example

Create a script file and type the following code:

```
a = 100;
% check the boolean condition
if a < 20
% if condition is true then print the following
fprintf('a is less than 20\n' );
else
% if condition is false then print the following
fprintf('a is not less than 20\n' );
end
fprintf('value of a is : %d\n', a);
```

When the above code is compiled and executed, it produces the following result:

```
a is not less than 20
```

```
value of a is : 100
```

if...elseif...elseif...else...end Statements

An **if** statement can be followed by one (or more) optional **elseif...** and an **else** statement, which is very useful to test various conditions.

When using if... elseif...else statements, there are few points to keep in mind:

An if can have zero or one else's and it must come after any elseif's.

An if can have zero to many elseif's and they must come before the else.

Once an else if succeeds, none of the remaining elseif's or else's will be tested.

Syntax

```
if <expression 1>
% Executes when the expression 1 is true
<statement(s)>
elseif <expression 2>
% Executes when the boolean expression 2 is true
<statement(s)>
Elseif <expression 3>
% Executes when the boolean expression 3 is true
<statement(s)>
else
% executes when the none of the above condition is true
<statement(s)>
end
```

Example

Create a script file and type the following code in it:

```
a = 100;
% check the boolean condition
if a = = 10
% if condition is true then print the following
fprintf('Value of a is 10\n' );
elseif ( a = = 20 )
% if else if condition is true
fprintf ('Value of a is 20\n' );
elseif a = = 30
% if else if condition is true
fprintf('Value of a is 30\n' );
else
% if none of the conditions is true '
fprintf('None of the values are matching\n');
fprintf('Exact value of a is: %d\n', a );
end
```

When the above code is compiled and executed, it produces the following result:

```
None of the values are matching
Exact value of a is: 100
```

The Nested if Statements

It is always legal in MATLAB to nest if-else statements which means you can use one if or elseif statement inside another if or elseif statement(s).

Syntax

The syntax for a nested if statement is as follows:

```
if <expression 1>
% Executes when the boolean expression 1 is true
if <expression 2>
% Executes when the boolean expression 2 is true
end
end
```

You can nest elseif...else in the similar way as you have nested if statement.

Example

Create a script file and type the following code in it:

```
a = 100;
b = 200;
% check the boolean condition
if( a == 100 )
% if condition is true then check the following
if( b == 200 )
% if condition is true then print the following
fprintf('Value of a is 100 and b is 200\n' );
end
end
fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

While Loop

The while loop repeatedly executes statements while a specified condition is true.

The syntax of a while loop in MATLAB is as following:

```
 while <expression>
<statements>
end
```

The while loop repeatedly executes a program statement(s) as long as the expression remains true.

An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example:1

```
a = 10;     % while loop execution
```

```
while( a < 20 )
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

When the code above is executed, the result will be:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example 2:

```
x=3
while (x<100)
x=x*3;
end
```

When the code above is executed, the result will be:

Output

```
x=243
```

Remark: one can think of while loop as a combination of for loop and an if statement. Here, the looping will keep going indefinitely as long as the condition (x<100), is satisfied. Therefore, the value of x progresses from 3,9,27,81, to 243 when loop is terminated.

Example 3:

```
x = input(‘Enter a Number:’);
count = 0;
while x > 1
x = x/2;
count = count + 1;
end
display (count);
```

When the code above is executed, the result will be:

```
Enter a Number:
130
Count =
8
```

Example : 4

```
x = input(‘Enter an integer:’);
Fact = 1;
While x > 1;
Fact = fact * x;
End
Display(fact);
```

When the code above is executed, the result will be:

```
Enter a Integer:
12
Fact =
479001600
```

For loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

The syntax of a for loop in MATLAB is as following:

```
for   index = values
<program statements>
...
end
```

Example: 1

```
b = 3
for k = 1:5
b = b^k
end
```

When the code above is executed, the result will be:

```
Output
3
9
27
81
243
```

Remark: the outputs are 3^1,3^2,3^3,3^4 and 3^5. The value of “k” keeps changing as we go through the loop.

Example: 2

```
sum1 = 0;
for k = 1:9
sum1 = sum1+ k;
end
```

When the code above is executed, the result will be:

Output

```
45
```

Remark: this program performs the summation of 1+2+3+4+5+6+7+8+9 = 45

Example: 3

```
sum1 = 0;
for k = 1:2:9
sum1 = sum1+ k;
end
sum1
```

When the code above is executed, the result will be:

Output

```
25
```

Remark: this program performs the summation of 1+3+5+9 = 45. This command “for k = 1:2:9” means we go through the loop only 5 times. First time with k=1, second time with k=1+2(=3), third time with k = 1+2+2(=5), and so on. The looping stops once k reaches 9.

Example: 4 using array

```
b = [3 8 9 4 7 5];
sum1 = 0;
for k = 1:4
sum1 = sum1+ b(k);
end
sum1
```

When the code above is executed, the result will be:

```
Output
24
```

Remark: this program performs the summation of sum1= b(1)+b(2)+b(3)+b(4) = 3+8+9+4 = 24

Example: 5 using array with step 2

```
b = [3 8 9 4 7 5];
sum1 = 0;
for k = 1:2:5
sum1 = sum1+ b(k);
end
sum1
```

When the code above is executed, the result will be:

```
Output
```

```
19
```

Remark: this program performs the summation of sum1= b(1)+b(3)+b(5) = 3+9+7 = 19

Example: 6

```
for a = 10:20
fprintf(‘value of a:%d\n’,a);
end
```

When the code above is executed, the result will be:

```
Output
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Example: 7 using two for loops

```
sum1 = 0;
for n = 1:2
for m = 1:3
sum1 = sum1+ n * m;
end
    end
sum1
```

When the code above is executed, the result will be:

Output

```
18
```

Remark: this program performs the summation of sum1= 1*1+1*2+1*3+2*1+2*2+2*3 = 18

Example: 8 using two for loops with printf

```
for n = 1:2
  for m = 1:3
fprintf(‘n = %3u m = %3u \r’, n, m)
end
    end
```

When the code above is executed, the result will be:

Output

```
n = 1         m = 1
n = 1         m = 1
```

```
n = 1        m = 1
n = 2        m = 2
n = 2        m = 2
n = 2        m = 2
```

Example: 9 More complicated use of loop and index

```
b = [3 5 7 4 9 8 3];
c = [2 3 5 7];
sum1 = 0;
for k = 1:4
sum1 = sum1+ b (c(k));
end
sum1
```

When the code above is executed, the result will be :

```
Output
24

Remark: this program performs the summation of
sum1= b(c(1)) + b(c(2)) + b(c(3)) + b(c(4))
      = b(2) + b(3) + b(5) + b(7)
      = 5+7+9+3
      =  24
```

Lab Program – 3&4

Numerical methods and their applications: Curve fitting: Straight line fit, Polynomial fit

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning mathematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

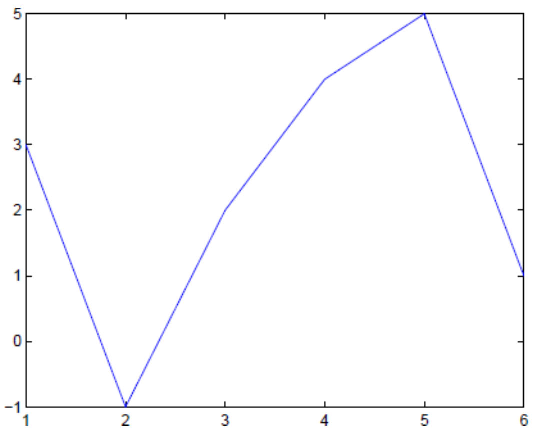
Creating simple plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of *x*- coordinates, x = $(x1; : : : ; xN)$, and a vector of *y*-coordinates, y = $(y1; : : : ; yN)$, locate the points $(xi; yi)$, with $i = 1; 2; : : : ; n$ and then join them by straight lines. You need to prepare *x* and *y* in an identical array form; namely, *x* and *y* are both row arrays and column arrays of the *same* length.

Example:1

Plot a graph considering the vectors, x = (1*;* 2*;* 3*;* 4*;* 5*;* 6) and y = (3*;* -1*;* 2*;* 4*;* 5*;* 1)

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

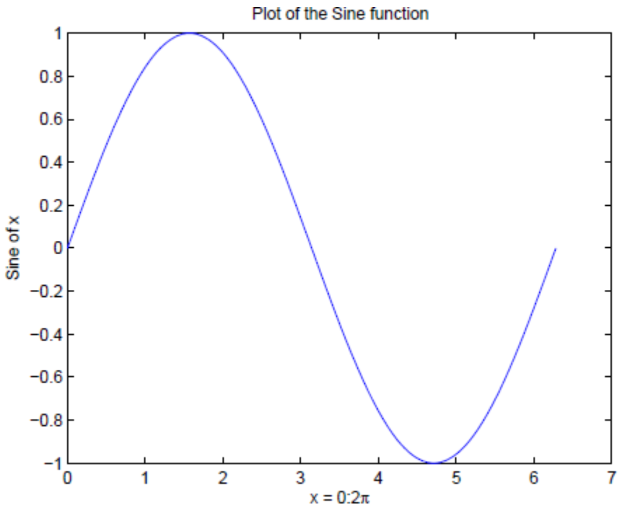


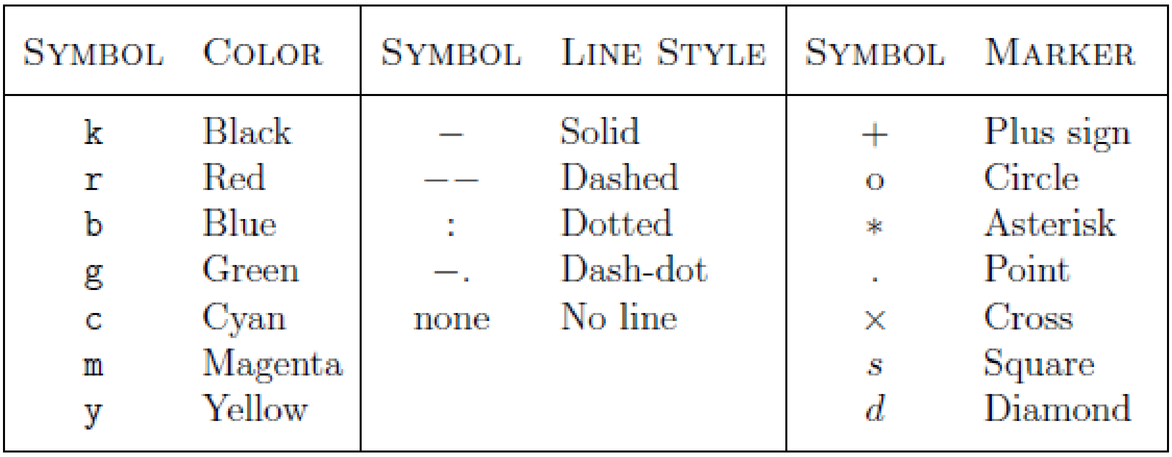
Note: The plot functions have different forms depending on the input arguments. If y is a vector plot(y) produces a piecewise linear graph of the elements of y versus the index of the elements of y. If we specify two vectors, as mentioned above, plot(x,y) produces a graph of y versus x.

Example: 2

Create x as a vector of linearly spaced values between 0 and 2π. Use an increment of $\pi/100$ between the values. Create y as sine values of x. Create a line plot of the data.

```
 x = 0:pi/100:2*pi;
y = sin(x);
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
>>plot(x,y)
```



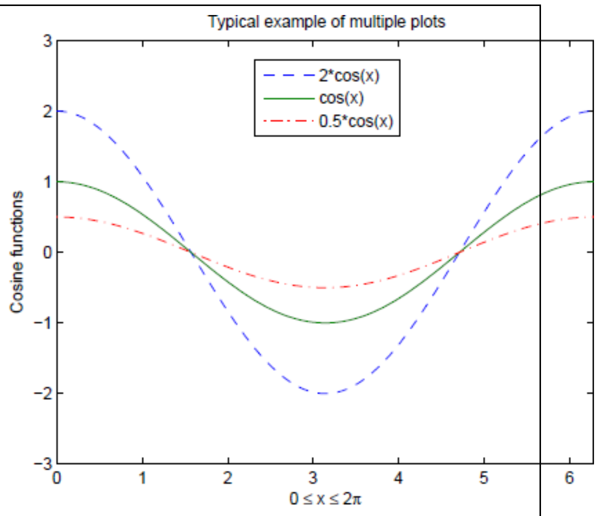


SYMBOL	COLOR	SYMBOL	LINE STYLE	SYMBOL	MARKER
k	Black	–	Solid	+	Plus sign
r	Red	– –	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			*s*	Square
y	Yellow			*d*	Diamond

Multiple data sets in one plot

Multiple (*x; y*) *pairs* arguments create *multiple* graphs with a single call to *plot*.

For **example,** these statements plot three related functions of *x*: *y*1 = 2 cos(*x*), *y*2 = cos(*x*), and *y*3 = 0:5* cos(*x*), in the interval 0 ≤ *x* ≤ 2*π*

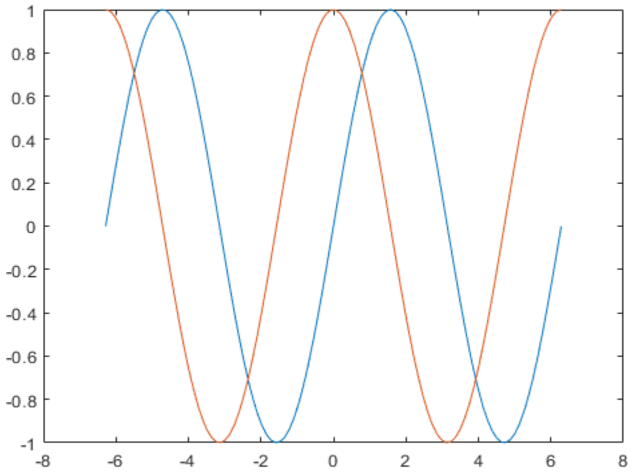
>> x = 0:pi/100:2*pi; >> y1 = 2*cos(x); >> y2 = cos(x); >> y3 = 0.5*cos(x); >> plot(x,y1,'--',x,y2,'-',x,y3,':') >> xlabel('0 \leq x \leq 2\pi') >> ylabel('Cosine functions') >> legend('2*cos(x)','cos(x)','0.5*cos(x)') >> title('Typical example of multipleplots') >> axis([0 2*pi -3 3])	

The result of multiple data sets in one graph plot is shown in Figure.

Example

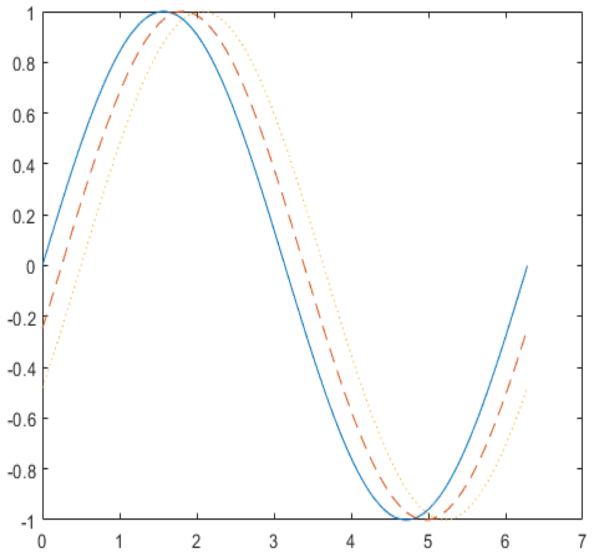
Define x as 100 linearly spaced values between -2π and 2π. Define y1 and y2 as sine and cosine values of x. Create a line plot of both sets of data.

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



Three Curves in single Plot

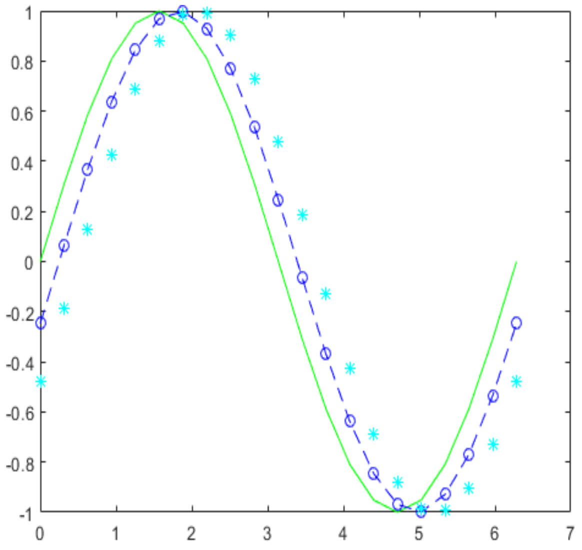
Plot three sine curves with a small phase shift between each line. Use the default line style for the first line. Specify a dashed line style for the second line and a dotted line style for the third line.

x = 0:pi/100:2*pi;
y1 = sin(x);
y2 = sin(x-0.25);
y3 = sin(x-0.5);
 	
%figure
plot(x,y1,x,y2,'--',x,y3,':')
	

Specify Line Style, Color, and Marker

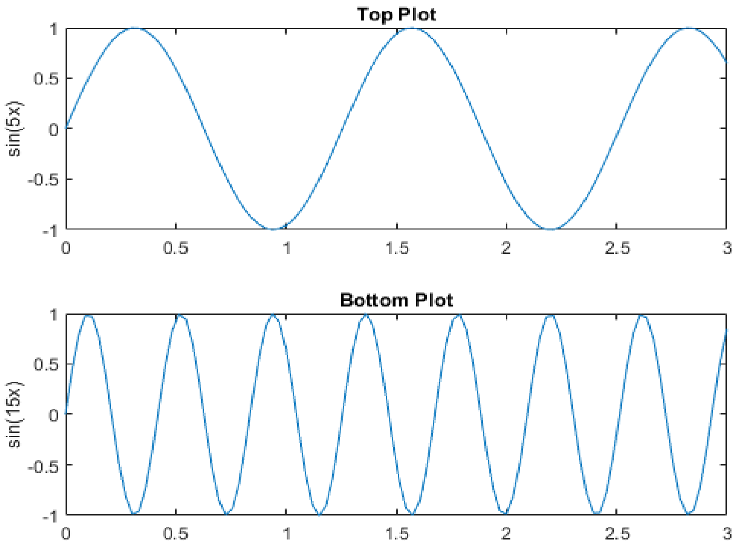
Plot three sine curves with a small phase shift between each line. Use a green line with no markers for the first sine curve. Use a blue dashed line with circle markers for the second sine curve. Use only cyan star markers for the third sine curve

```
x = 0:pi/10:2*pi;
y1 = sin(x);
y2 = sin(x-0.25);
y3 = sin(x-0.5);
plot(x,y1,'g',x,y2,'b--o',x,y3,'c*')
```



Bifurcation Plots

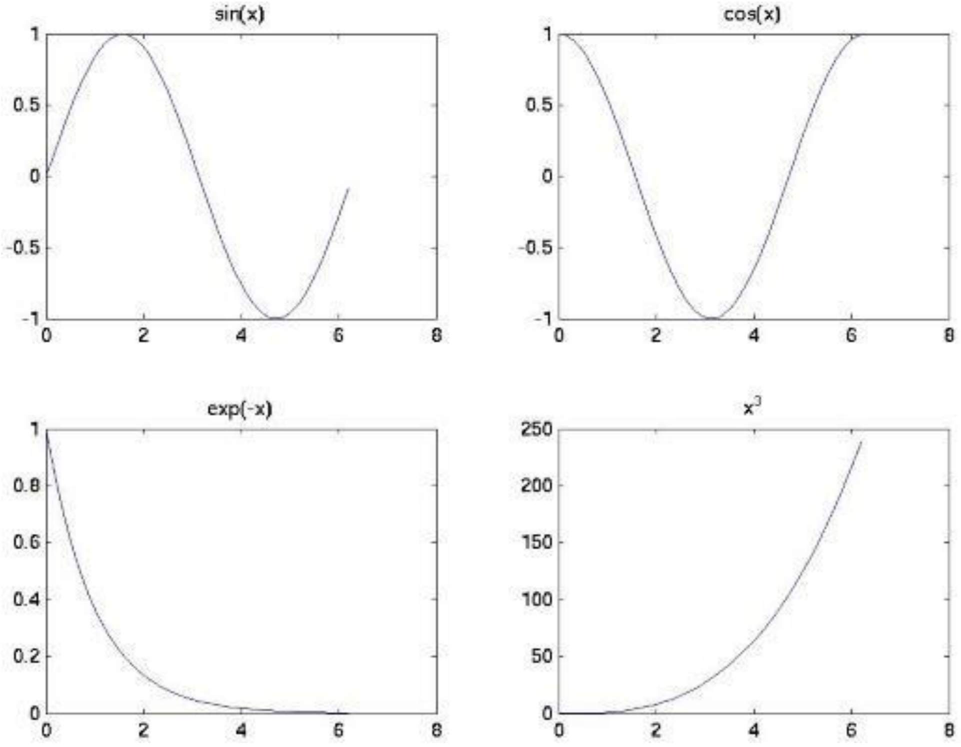
```
x = linspace(0,3);
y1 = sin(5*x);
y2 = sin(15*x);
subplot(2,1,1);
plot(x,y1)
title('Top Plot')
ylabel('sin(5x)')
subplot(2,1,2);
plot(x,y2)
title('Bottom Plot')
ylabel('sin(15x)')
```



Subplots

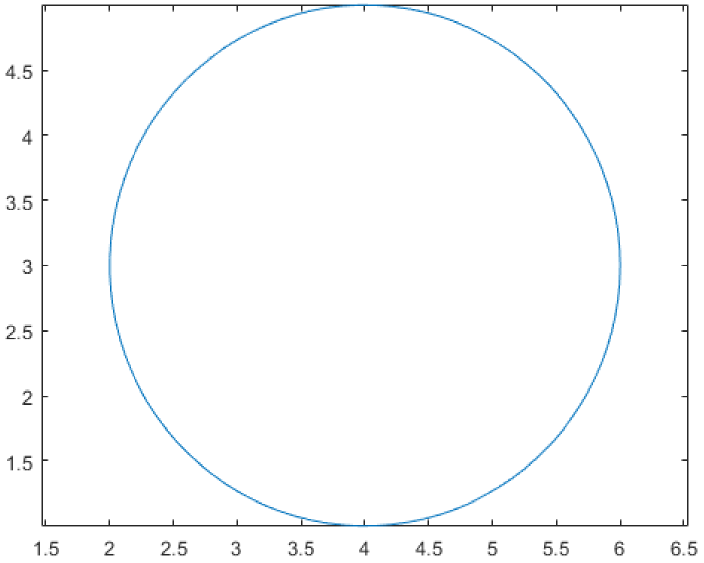
```
x = linspace(0,6);
```

```
subplot(2,2,1);   % plot sine function

plot(x,sin(x));

subplot(2,2,2);   % plot cosine function

plot(x,cos(x));

subplot(2,2,3);    % plot negative   exponential function

plot(x,exp(-x));

subplot(2,2,4);   % plot x^3

plot(x, x.^3);
```



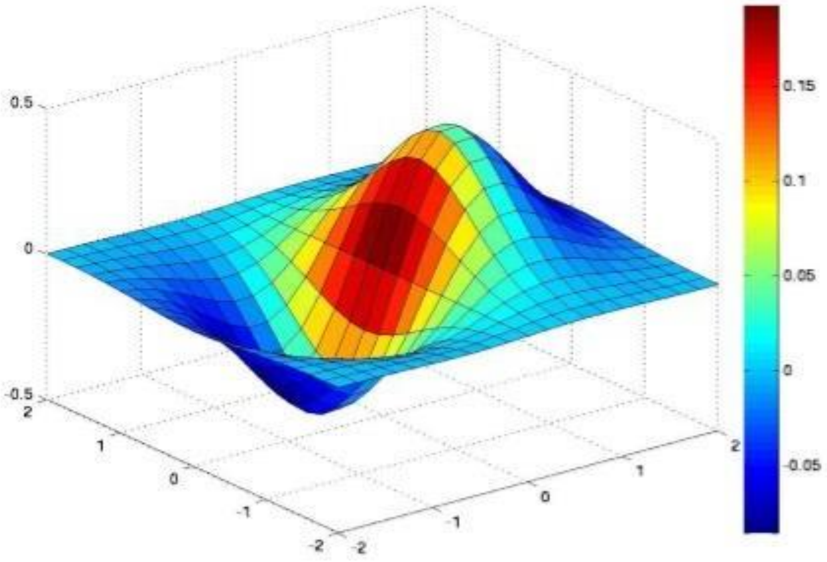
Plots Circle

```
r = 2;
xc = 4;
yc = 3;
theta =
linspace(0,2*pi);
x = r*cos(theta) + xc;
y = r*sin(theta) + yc;
plot(x,y) axis equal
```



Plotting in 3-D

```
[x,y] = meshgrid([-2:.2:2]);   % set up 2-D plane
Z = x.*exp(-x.^2-y.^2);         % plot 3rd dimension on plane figure
surf(x,y,Z,gradient(Z))        % surface plot, with gradient(Z)
                                  % determining color distribution
colorbar                        % display color scale, can adjust
                                % location similarly to legend
```



Polynomials in MATLAB

Evaluating Polynomials

```
p = [1 7 0  -5 9];
polyval (p,4)
```

```
Ans = 693
```

```
p = [1 7 0  -5 9];
X = [1 2 -3 4; 2 -5 6 3; 3 1 0 2; 5 -7 3 8];
polyvalm(p, X)
```

```
ans=
          2307     -1769     -939     4499

          2314     -2376     -249     4695
          2256     -1892     -549     4310
```

Finding the roots of Polynomials

Solve the equation $3x^2 - 2x - 4$

Create the vector to represent the polynomial, then find the roots

```
P=[3 -2 -4];
r = roots (p)
```

```
Ans r = 2 x 1
1.532
-0.8685
```

Solve the equation $x^4 + 7x^3 - 5x + 9 = 0$

Create a vector to represent the polynomial, then find the roots

```
p = [1 7  0 -5  9];
r = roots(p)
```

```
r =
  -6.8661 + 0.0000i
  -1.4247 + 0.0000i
  0.6454 + 0.7095i
  0.6454 - 0.7095i
```

The function **poly** is an inverse of the roots function and returns to the polynomial coefficients. For example:

p2 = poly(r)

```
p2 =
  Columns 1 through 3:
     1.00000 + 0.00000i   7.00000 + 0.00000i   0.00000 + 0.00000i
   Columns 4 and 5:
    -5.00000 - 0.00000i   9.00000 + 0.00000i
```

Polynomial Curve Fitting

The **polyfit** function finds the coefficients of a polynomial that fits a set of data in a least-squares sense. If x and y are two vectors containing the x and y data to be fitted to a n-degree polynomial, then we get the polynomial fitting the data by writing **p = polyfit(x,y,n)**

Lab-Program 3: Numerical methods and their applications: Curve fitting: Straight line fit.

Example 1 *By the method of least squares, fit a straight line to the following data:*

x :	1	2	3	4	5
y :	14	13	9	5	2

▸ Here, $n = 5$ values of (x, y) are given. We first form the following Table.

*Here and in expressions that follow, $\sum$ denotes $\sum_{i=1}^{n}$.

*Note that $\sum a = a + a + a \ldots$ (n terms) $= na$

x_i	y_i	x_i^2	x_iy_i
1	14	1	14
2	13	4	26
3	9	9	27
4	5	16	20
5	2	25	10
$\sum x_i = 15$	$\sum y_i = 43$	$\sum x_i^2 = 55$	$\sum x_iy_i = 97$

From the details available in the Table, we find that the normal equations
the line of best fit are*

$$43 = 5a + 15b,$$

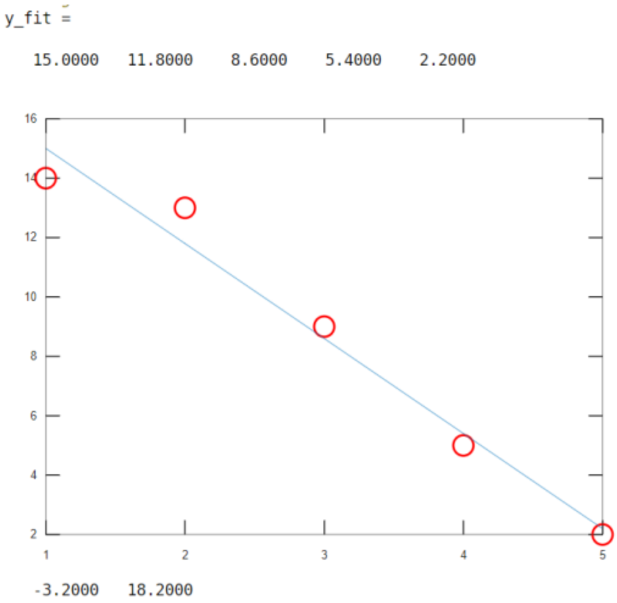
$$97 = 15a + 55b.$$

Solving these equations, we get $a = 91/5$ and $b = -(16/5)$. Hence, for the
line of best fit is

$$y = a + bx = \frac{91}{5} - \frac{16}{5}x, \quad \text{or} \quad 5y = 91 - 16x$$

Lab Program -3 code with degree1 for strainght line

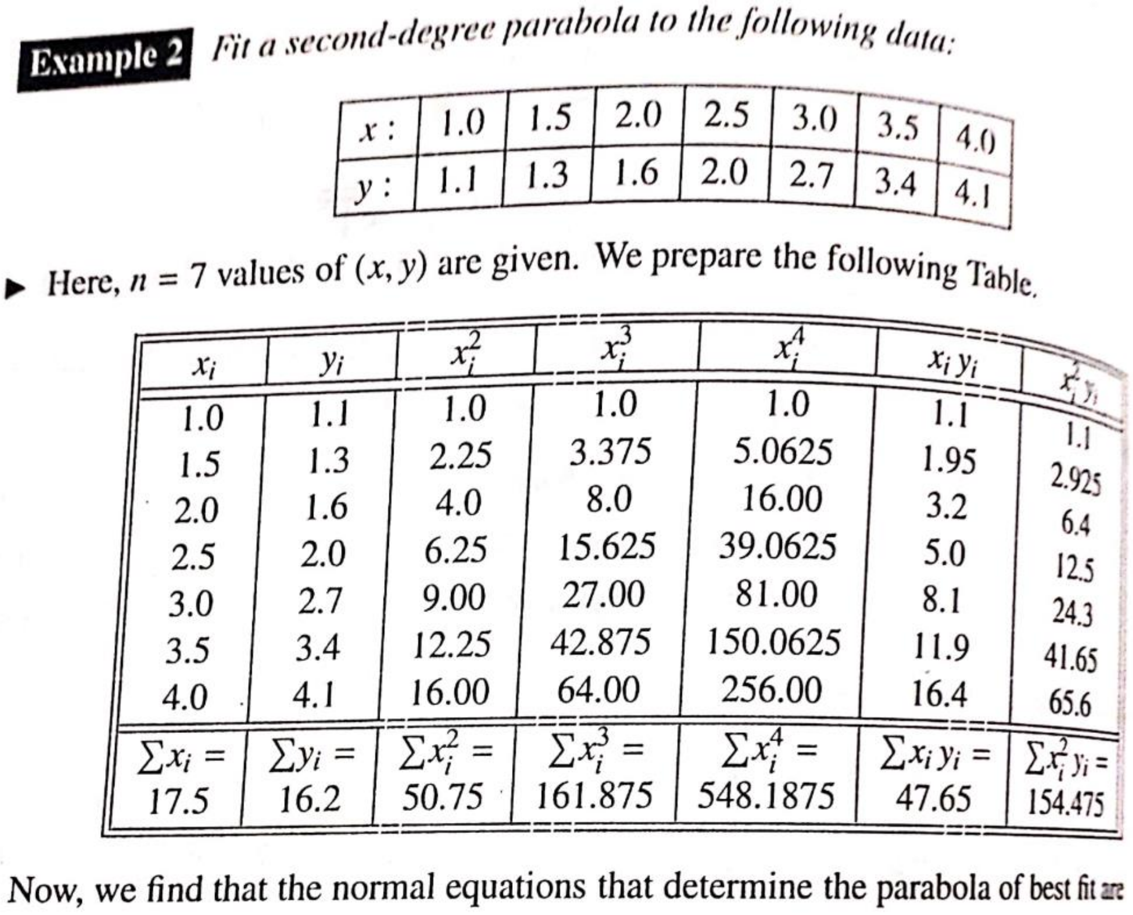
```
x=[1:5];
y=[14,13,9,5,2];
p=polyfit(x,y);
p=polyfit(x,y,1);
y_fit=polyval(p,x)
plot(x,y,'ro',x,y_fit)
disp(p)
```



```
y_fit =

   15.0000   11.8000    8.6000    5.4000    2.2000
```

```
   -3.2000   18.2000
```

Program with degree 2



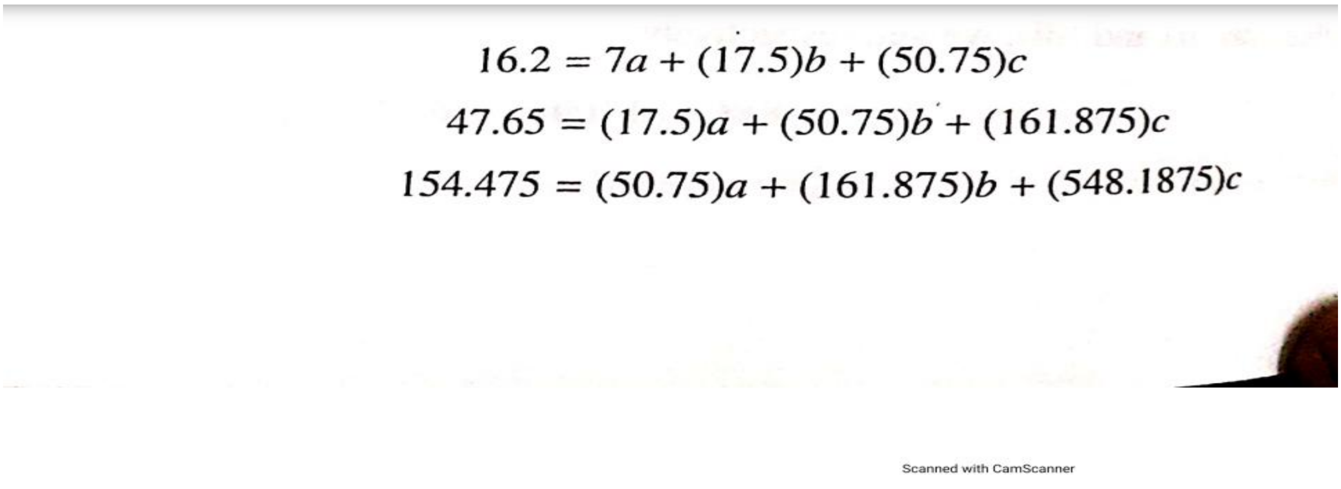
Example 2 *Fit a second-degree parabola to the following data:*

$x:$	1.0	1.5	2.0	2.5	3.0	3.5	4.0
$y:$	1.1	1.3	1.6	2.0	2.7	3.4	4.1

▶ Here, $n = 7$ values of (x, y) are given. We prepare the following Table.

x_i	y_i	x_i^2	x_i^3	x_i^4	$x_i y_i$	$x_i^2 y_i$
1.0	1.1	1.0	1.0	1.0	1.1	1.1
1.5	1.3	2.25	3.375	5.0625	1.95	2.925
2.0	1.6	4.0	8.0	16.00	3.2	6.4
2.5	2.0	6.25	15.625	39.0625	5.0	12.5
3.0	2.7	9.00	27.00	81.00	8.1	24.3
3.5	3.4	12.25	42.875	150.0625	11.9	41.65
4.0	4.1	16.00	64.00	256.00	16.4	65.6
$\sum x_i =$ 17.5	$\sum y_i =$ 16.2	$\sum x_i^2 =$ 50.75	$\sum x_i^3 =$ 161.875	$\sum x_i^4 =$ 548.1875	$\sum x_i y_i =$ 47.65	$\sum x_i^2 y_i =$ 154.475

Now, we find that the normal equations that determine the parabola of best fit are

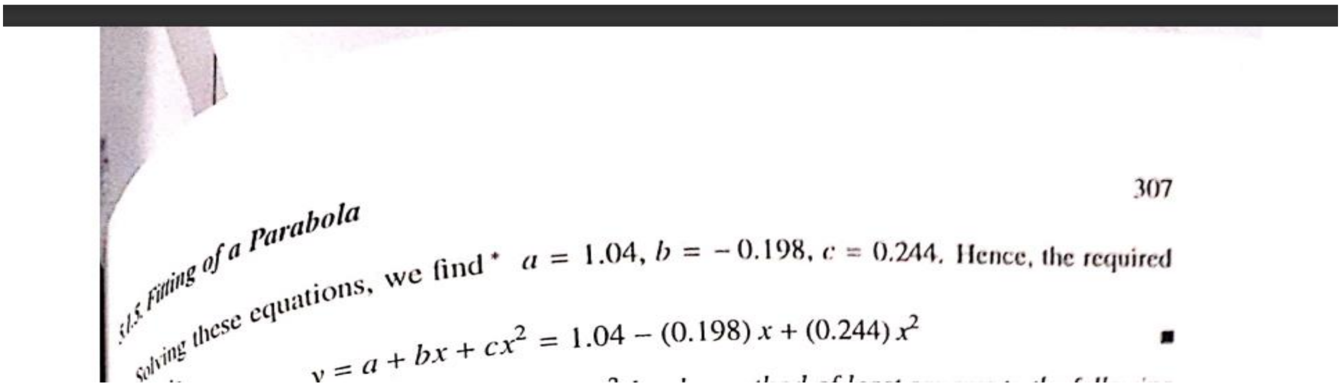


$$16.2 = 7a + (17.5)b + (50.75)c$$

$$47.65 = (17.5)a + (50.75)b + (161.875)c$$

$$154.475 = (50.75)a + (161.875)b + (548.1875)c$$

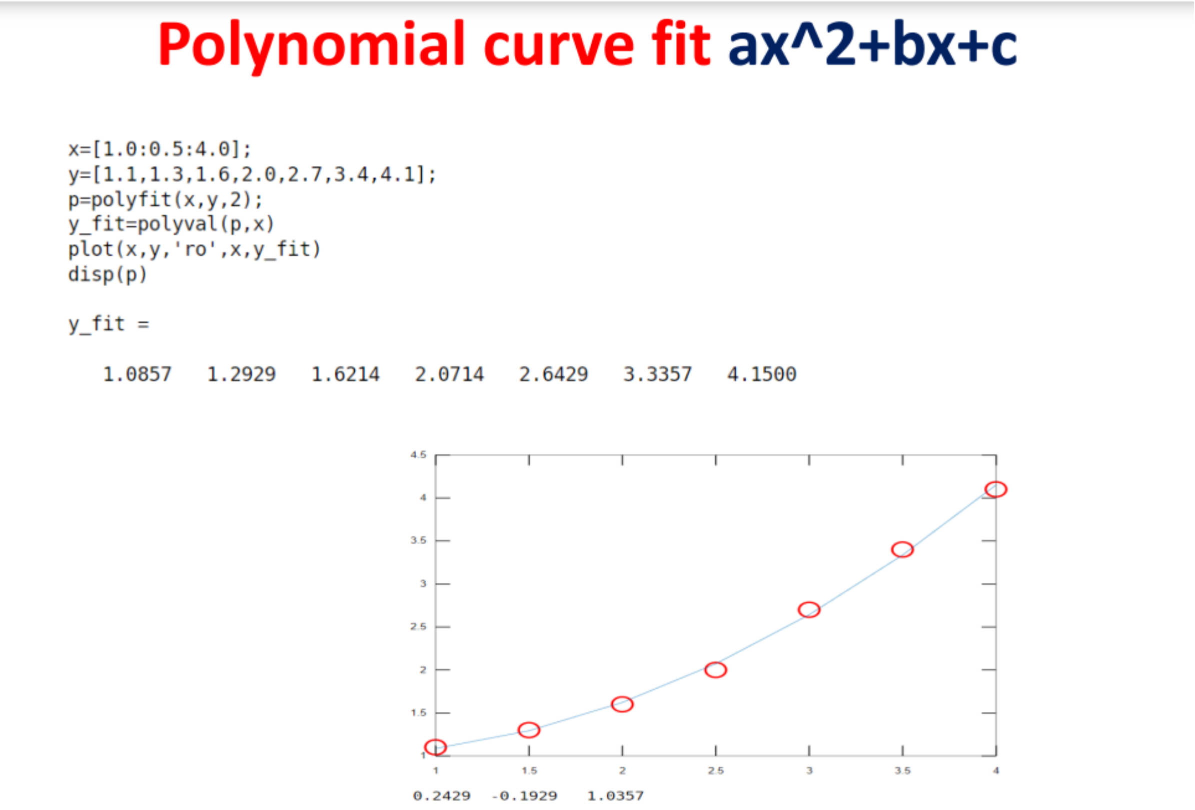
Scanned with CamScanner



Solving these equations, we find* $a = 1.04$, $b = -0.198$, $c = 0.244$. Hence, the required

$$y = a + bx + cx^2 = 1.04 - (0.198)\,x + (0.244)\,x^2$$ ■

Lab Program 4: Program code with degree2 polynomial curve fit



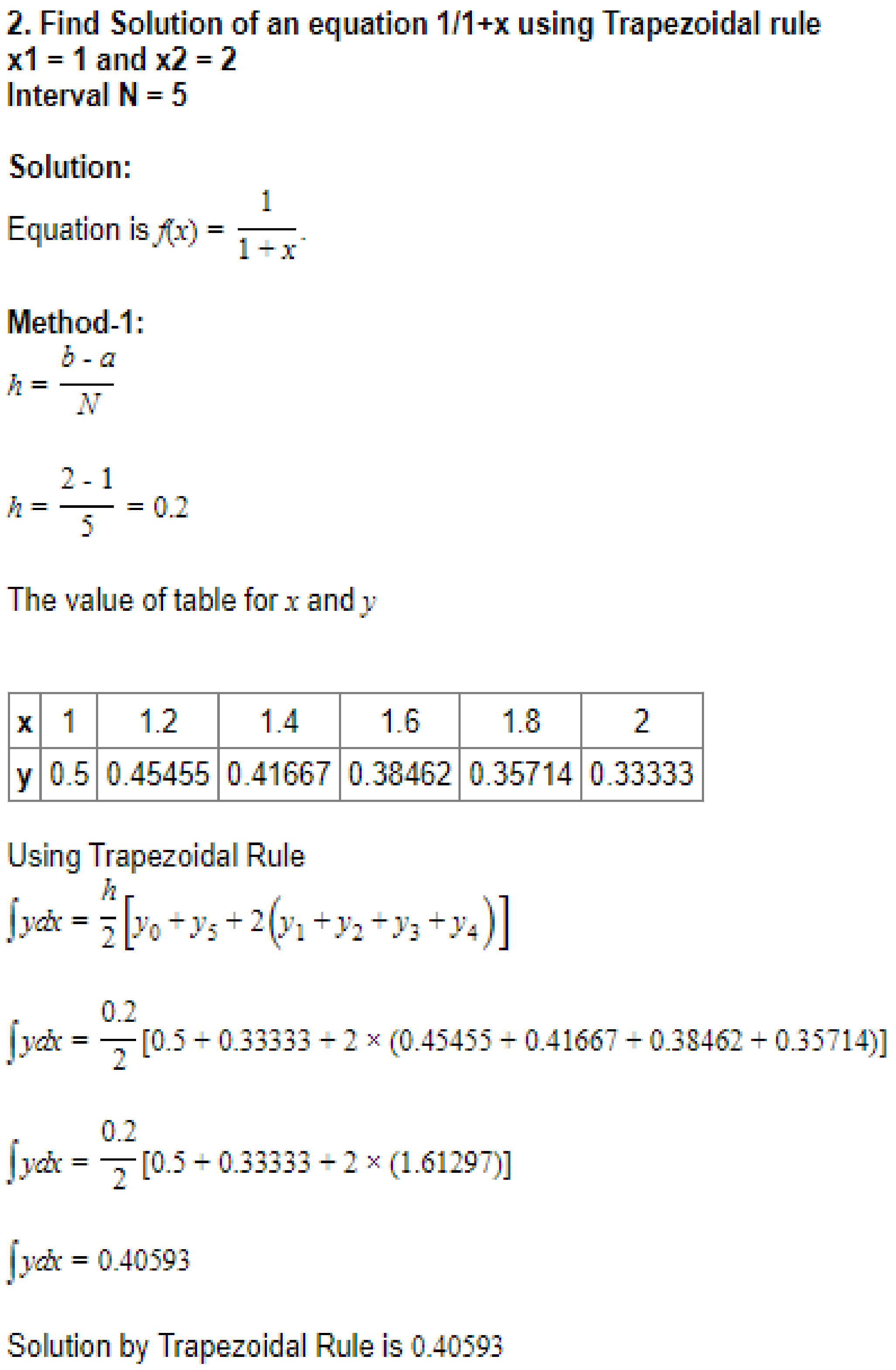
Polynomial curve fit ax^2+bx+c

```
x=[1.0:0.5:4.0];
y=[1.1,1.3,1.6,2.0,2.7,3.4,4.1];
p=polyfit(x,y,2);
y_fit=polyval(p,x)
plot(x,y,'ro',x,y_fit)
disp(p)

y_fit =

   1.0857    1.2929    1.6214    2.0714    2.6429    3.3357    4.1500
```

```
0.2429   -0.1929   1.0357
```

Lab Program - 5

Numerical Integration and Differentiation: Trapezoidal method,Simpson method.



2. Find Solution of an equation 1/1+x using Trapezoidal rule
x1 = 1 and x2 = 2
Interval N = 5

Solution:

Equation is $f(x) = \frac{1}{1+x}$.

Method-1:

$$h = \frac{b - a}{N}$$

$$h = \frac{2 - 1}{5} = 0.2$$

The value of table for x and y

x	1	1.2	1.4	1.6	1.8	2
y	0.5	0.45455	0.41667	0.38462	0.35714	0.33333

Using Trapezoidal Rule

$$\int y dx = \frac{h}{2}\left[y_0 + y_5 + 2\left(y_1 + y_2 + y_3 + y_4\right)\right]$$

$$\int y dx = \frac{0.2}{2}[0.5 + 0.33333 + 2 \times (0.45455 + 0.41667 + 0.38462 + 0.35714)]$$

$$\int y dx = \frac{0.2}{2}[0.5 + 0.33333 + 2 \times (1.61297)]$$

$$\int y dx = 0.40593$$

Solution by Trapezoidal Rule is 0.40593

Lab Program 5-Numerical Integration and Differentiation: Trapezoidal method.

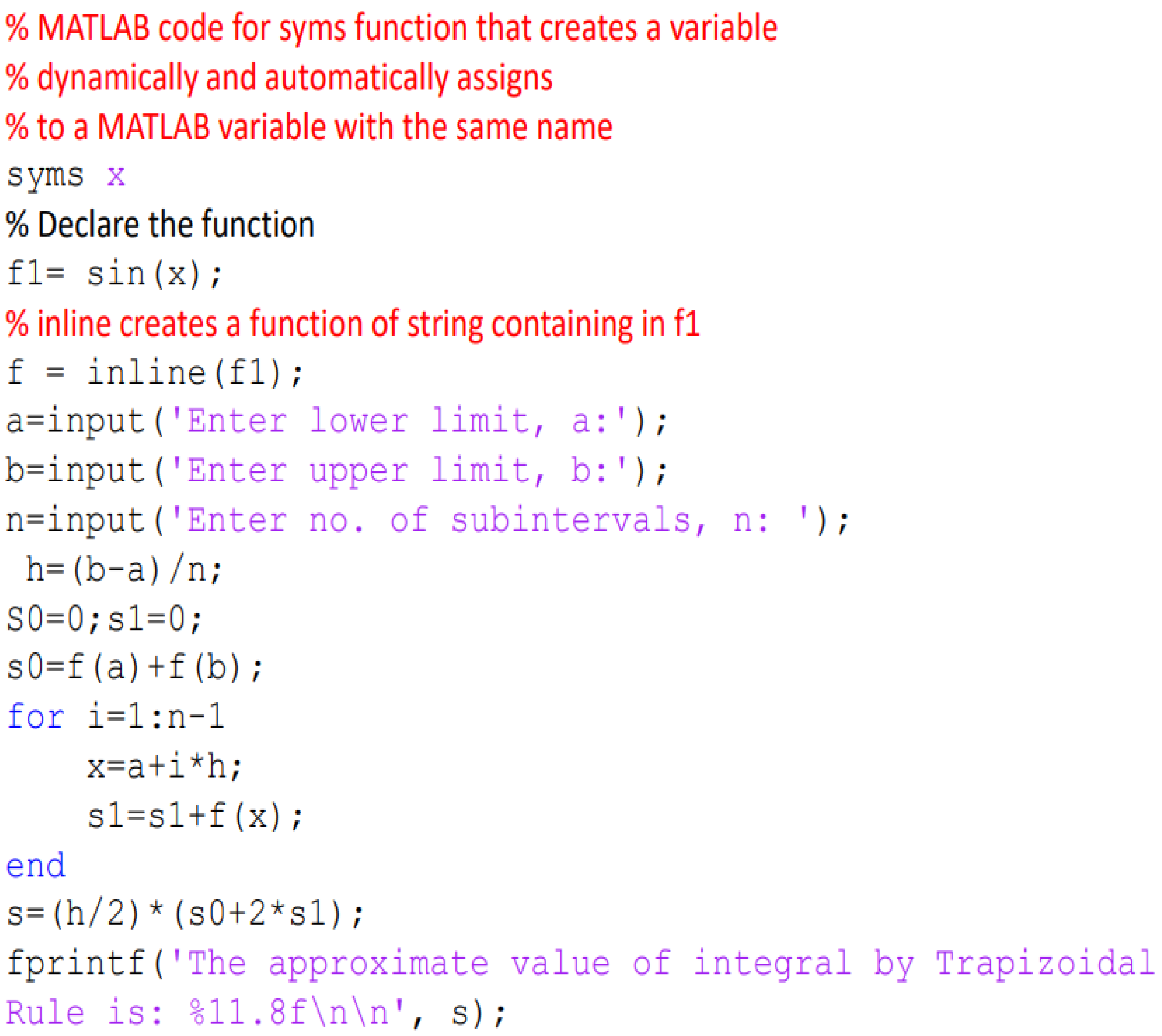
Program Code With Example 1:f=1/(1+x)

```
% MATLAB code for syms function that creates a variable
% dynamically and automatically assigns
% to a MATLAB variable with the same name
syms x
% Declare the function
f1=  1/(1+x);
% inline creates a function of string containing in f1
f = inline(f1);
a=input('Enter lower limit, a:');
b=input('Enter upper limit, b:');
n=input('Enter no. of subintervals, n: ');
 h=(b-a)/n;
S0=0;s1=0;
s0=f(a)+f(b);
for i=1:n-1
    x=a+i*h;
    s1=s1+f(x);
end
s=(h/2)*(s0+2*s1);
fprintf('The approximate value of integral by
Trapizoidal Rule is: %11.8f\n\n', s);
```

OUTPUT

```
Enter lower limit, a:1

Enter upper limit, b:2

Enter no. of subintervals, n: 5

The approximate value of integral by Trapizoidal Rule is: 0.40592741
```

Program with Example 2:f=sin(x)



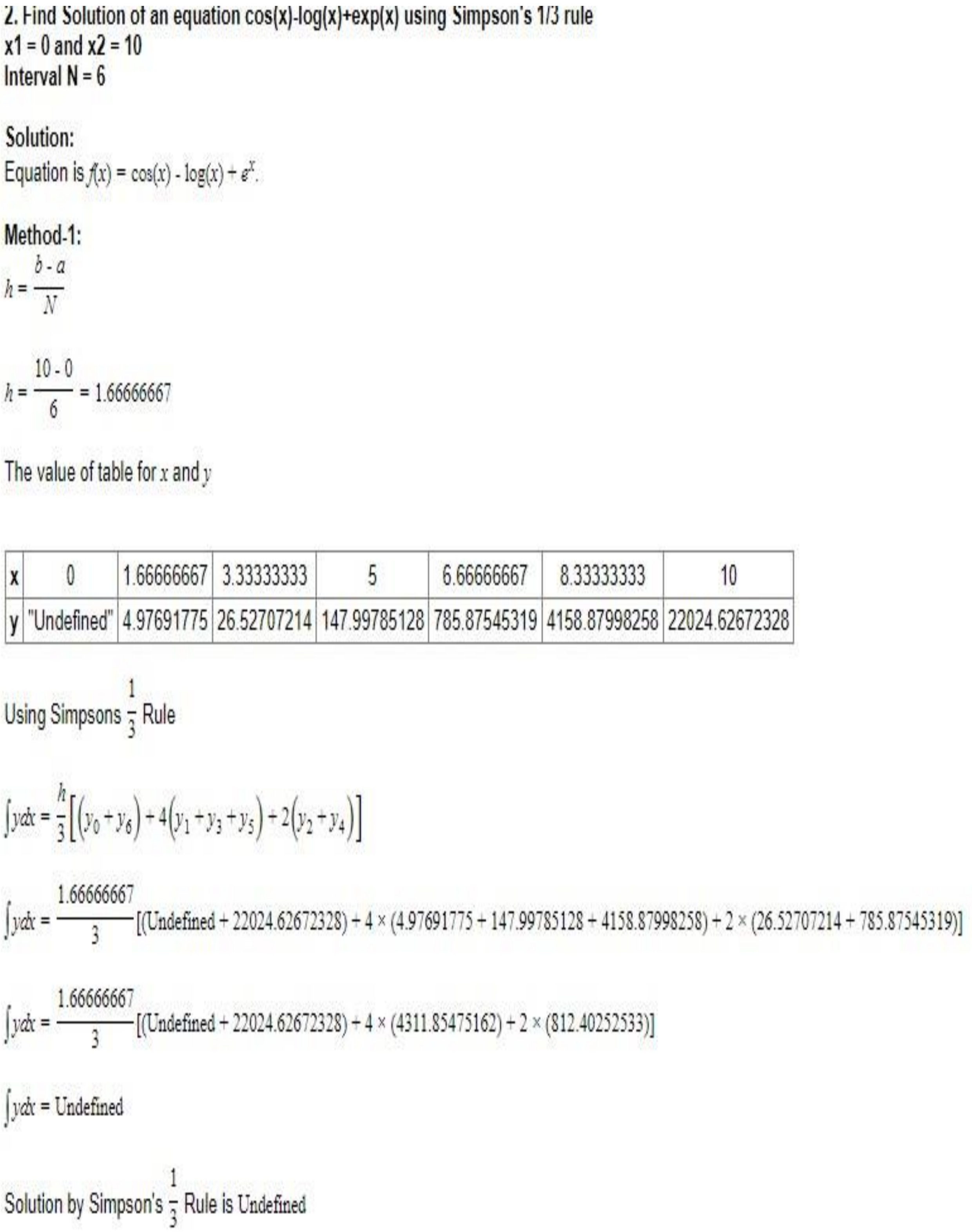
```
% MATLAB code for syms function that creates a variable
% dynamically and automatically assigns
% to a MATLAB variable with the same name
syms x
% Declare the function
f1= sin(x);
% inline creates a function of string containing in f1
f = inline(f1);
a=input('Enter lower limit, a:');
b=input('Enter upper limit, b:');
n=input('Enter no. of subintervals, n: ');
 h=(b-a)/n;
S0=0;s1=0;
s0=f(a)+f(b);
for i=1:n-1
    x=a+i*h;
    s1=s1+f(x);
end
s=(h/2)*(s0+2*s1);
fprintf('The approximate value of integral by Trapizoidal
Rule is: %11.8f\n\n', s);
```

OUTPUT

```
Enter lower limit, a:2

Enter upper limit, b:8

Enter no. of subintervals, n: 5

The approximate value of integral by Trapizoidal Rule is: -0.23736200
```

Lab Program 6-Numerical Integration and Differentiation:Simpson’s method

Simpson’s 1/3 rule



2. Find Solution of an equation cos(x)-log(x)+exp(x) using Simpson's 1/3 rule
x1 = 0 and x2 = 10
Interval N = 6

Solution:

Equation is $f(x) = \cos(x) - \log(x) + e^x$.

Method-1:

$$h = \frac{b - a}{N}$$

$$h = \frac{10 - 0}{6} = 1.66666667$$

The value of table for x and y

x	0	1.66666667	3.33333333	5	6.66666667	8.33333333	10
y	"Undefined"	4.97691775	26.52707214	147.99785128	785.87545319	4158.87998258	22024.62672328

Using Simpsons $\frac{1}{3}$ Rule

$$\int y dx = \frac{h}{3}\left[\left(y_0 + y_6\right) + 4\left(y_1 + y_3 + y_5\right) + 2\left(y_2 + y_4\right)\right]$$

$$\int y dx = \frac{1.66666667}{3}[(\text{Undefined} + 22024.62672328) + 4 \times (4.97691775 + 147.99785128 + 4158.87998258) + 2 \times (26.52707214 + 785.87545319)]$$

$$\int y dx = \frac{1.66666667}{3}[(\text{Undefined} + 22024.62672328) + 4 \times (4311.85475162) + 2 \times (812.40252533)]$$

$$\int y dx = \text{Undefined}$$

Solution by Simpson's $\frac{1}{3}$ Rule is Undefined

Program Code Example 1 : SIMPSON 1/3 RULE FOR cos(x)-log(x)+exp(x)

SIMPSON 1/3 RULE FOR cos(x)-log(x)+exp(x)

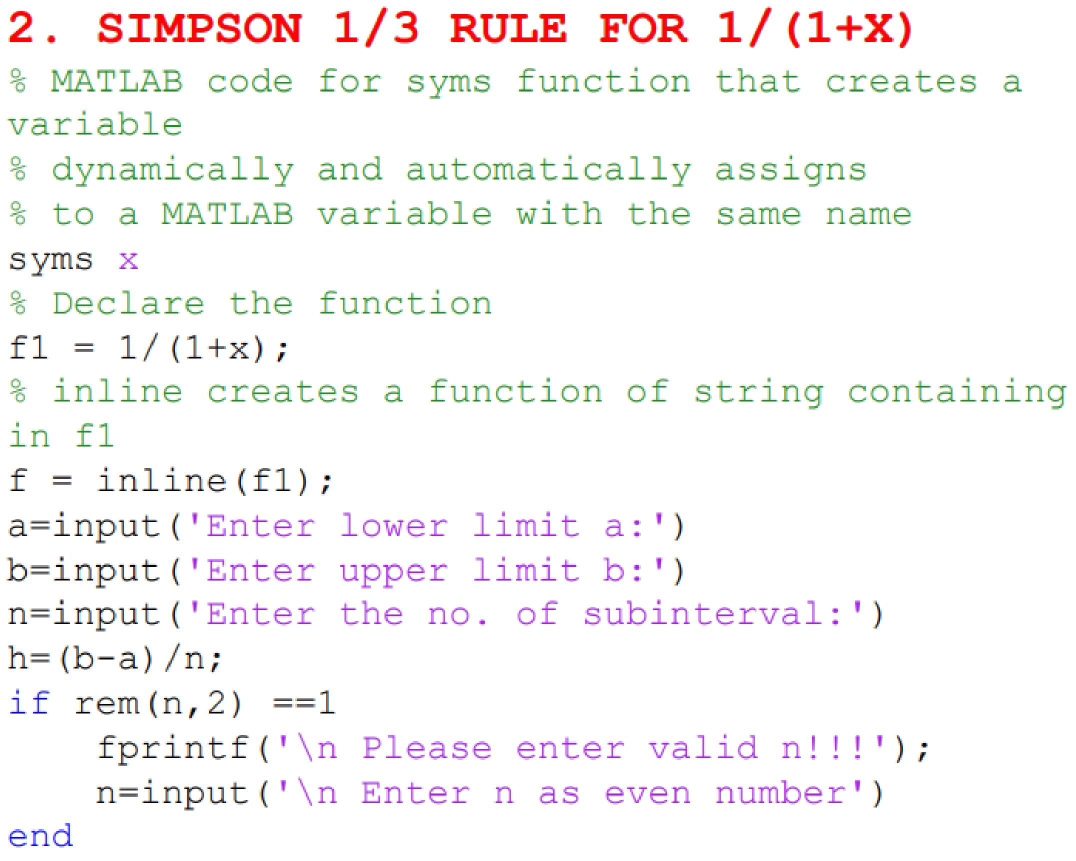
```
% MATLAB code for syms function that creates a
variable
% dynamically and automatically assigns
% to a MATLAB variable with the same name
syms x
% Declare the function
f1 = cos(x)-log(x)+exp(x);
% inline creates a function of string containing
in f1
f = inline(f1);
a=input('Enter lower limit a:')
b=input('Enter upper limit b:')
n=input('Enter the no. of subinterval:')
h=(b-a)/n;
if rem(n,2) ==1
    fprintf('\n Please enter valid n!!!');
    n=input('\n Enter n as even number')
end
```

```
% X stores the summation of first and last segment
X = f(a)+f(b);

% variables Odd and Even to store
% summation of odd and even
% terms respectively
Odd = 0;
Even = 0;
for i = 1:2:n-1
    xi=a+(i*h);
    Odd=Odd+f(xi);
end
for i = 2:2:n-2
    xi=a+(i*h);
    Even=Even+f(xi);
end
% Formula to calculate numerical integration
% using Simpsons 1/3 Rule
I = (h/3)*(X+4*Odd+2*Even);

disp('The approximation value of integral by Simsons
1/3 Rule is: ');
```

OUTPUT

```
Enter lower limit a:0

a =0

Enter upper limit b:10

b =10

Enter the no. of subinterval:5

n =5

 Please enter valid n!!!

Enter n as even number6

n =6

The approximation value of integral by Simsons 1/3 Rule is:    Inf
```

Program code Example 2: SIMPSON 1/3 RULE FOR 1/(1+X)



2. SIMPSON 1/3 RULE FOR 1/(1+X)

```
% MATLAB code for syms function that creates a
variable
% dynamically and automatically assigns
% to a MATLAB variable with the same name
syms x
% Declare the function
f1 = 1/(1+x);
% inline creates a function of string containing
in f1
f = inline(f1);
a=input('Enter lower limit a:')
b=input('Enter upper limit b:')
n=input('Enter the no. of subinterval:')
h=(b-a)/n;
if rem(n,2) ==1
    fprintf('\n Please enter valid n!!!');
    n=input('\n Enter n as even number')
end
```

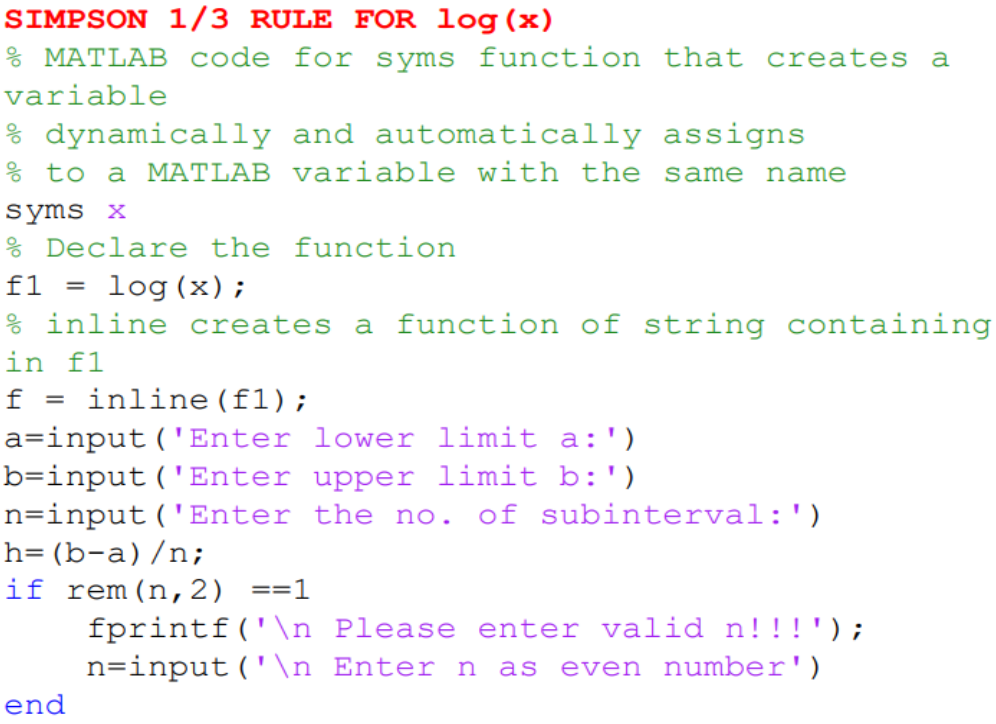
SIMPSON 1/3 RULE FOR 1/(1+X) (Cont….)

```
% X stores the summation of first and last segment
X = f(a)+f(b);
  % variables Odd and Even to store
% summation of odd and even
% terms respectively
Odd = 0;
Even = 0;
for i = 1:2:n-1
    xi=a+(i*h);
    Odd=Odd+f(xi);
end
for i = 2:2:n-2
    xi=a+(i*h);
    Even=Even+f(xi);
end
% Formula to calculate numerical integration
% using Simpsons 1/3 Rule
I = (h/3)*(X+4*Odd+2*Even);
  disp('The approximation value of integral by
Simsons 1/3 Rule is: ');
disp(I)
```

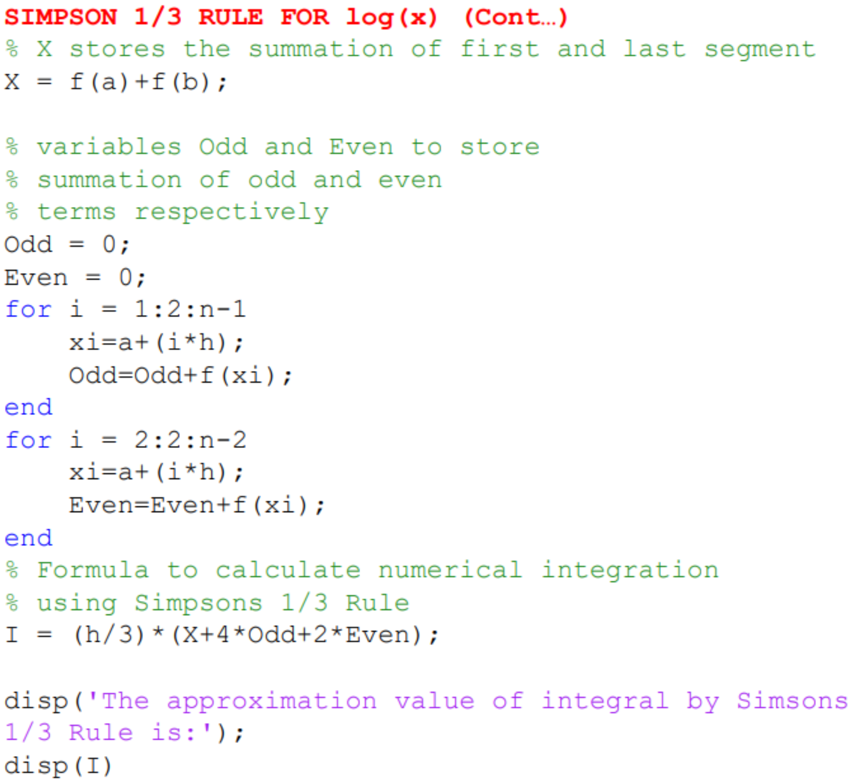
OUTPUT

```
Enter lower limit a:0

a =0

Enter upper limit b:10

b =10

Enter the no. of subinterval:6

n =6

The approximation value of integral by Simsons 1/3 Rule is:   2.4492
```

Program code Example 3: SIMPSON 1/3 RULE FOR log(x)



```
SIMPSON 1/3 RULE FOR log(x)
% MATLAB code for syms function that creates a
variable
% dynamically and automatically assigns
% to a MATLAB variable with the same name
syms x
% Declare the function
f1 = log(x);
% inline creates a function of string containing
in f1
f = inline(f1);
a=input('Enter lower limit a:')
b=input('Enter upper limit b:')
n=input('Enter the no. of subinterval:')
h=(b-a)/n;
if rem(n,2) ==1
     fprintf('\n Please enter valid n!!!');
     n=input('\n Enter n as even number')
end
```

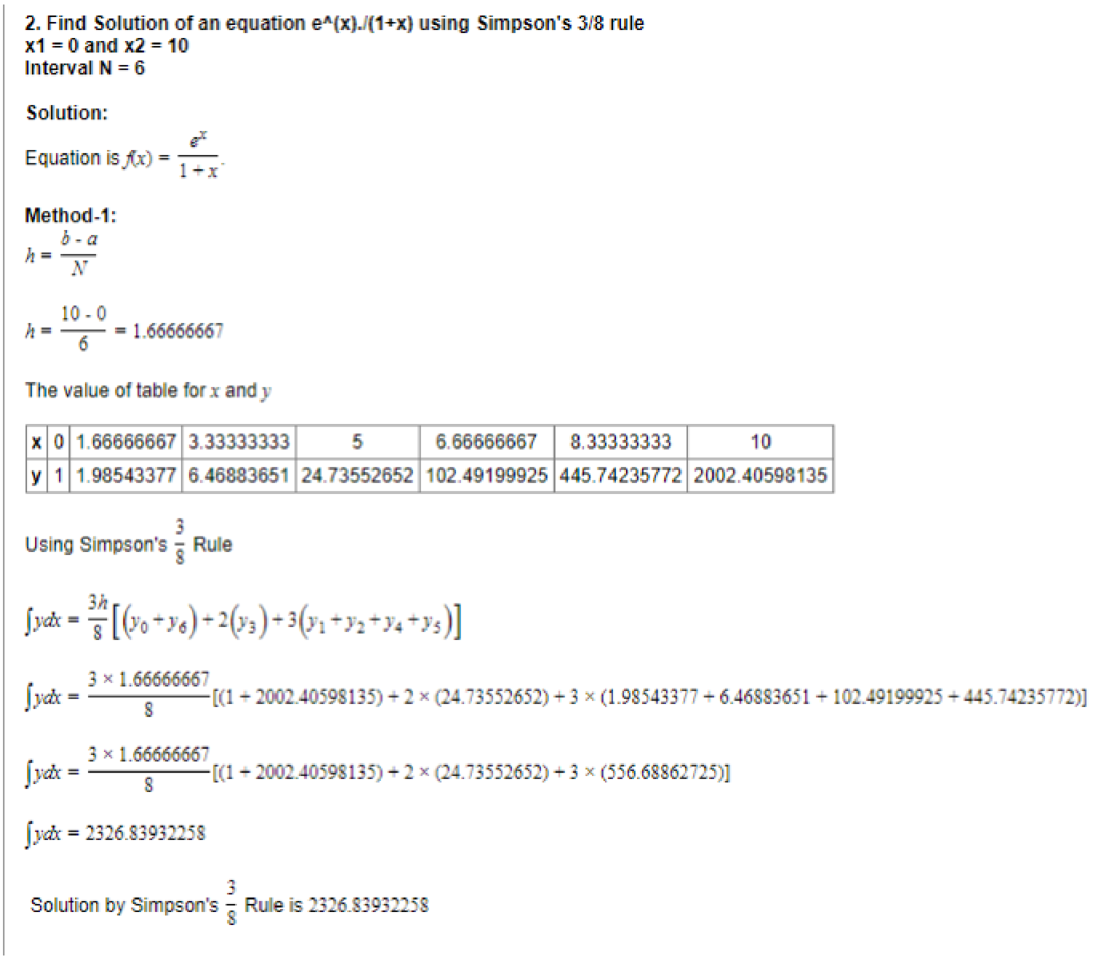


```
SIMPSON 1/3 RULE FOR log(x) (Cont…)
% X stores the summation of first and last segment
X = f(a)+f(b);

% variables Odd and Even to store
% summation of odd and even
% terms respectively
Odd = 0;
Even = 0;
for i = 1:2:n-1
    xi=a+(i*h);
    Odd=Odd+f(xi);
end
for i = 2:2:n-2
    xi=a+(i*h);
    Even=Even+f(xi);
end
% Formula to calculate numerical integration
% using Simpsons 1/3 Rule
I = (h/3)*(X+4*Odd+2*Even);

disp('The approximation value of integral by Simsons
1/3 Rule is:');
disp(I)
```

Output:

```
simsons1
Enter lower limit a:4
a =4

Enter upper limit b:5.2
b =5.2000

Enter the no. of subinterval:6
n =6

The approximation value of integral by Simsons 1/3 Rule is:
   1.8278
```

SIMPSON'S 3/8TH RULE



2. Find Solution of an equation e^(x)./(1+x) using Simpson's 3/8 rule
x1 = 0 and x2 = 10
Interval N = 6

Solution:

Equation is $f(x) = \frac{e^x}{1+x}$.

Method-1:

$$h = \frac{b - a}{N}$$

$$h = \frac{10 - 0}{6} = 1.66666667$$

The value of table for x and y

x	0	1.66666667	3.33333333	5	6.66666667	8.33333333	10
y	1	1.98543377	6.46883651	24.73552652	102.49199925	445.74235772	2002.40598135

Using Simpson's $\frac{3}{8}$ Rule

$$\int y dx = \frac{3h}{8}\left[\left(y_0 + y_6\right) + 2\left(y_3\right) + 3\left(y_1 + y_2 + y_4 + y_5\right)\right]$$

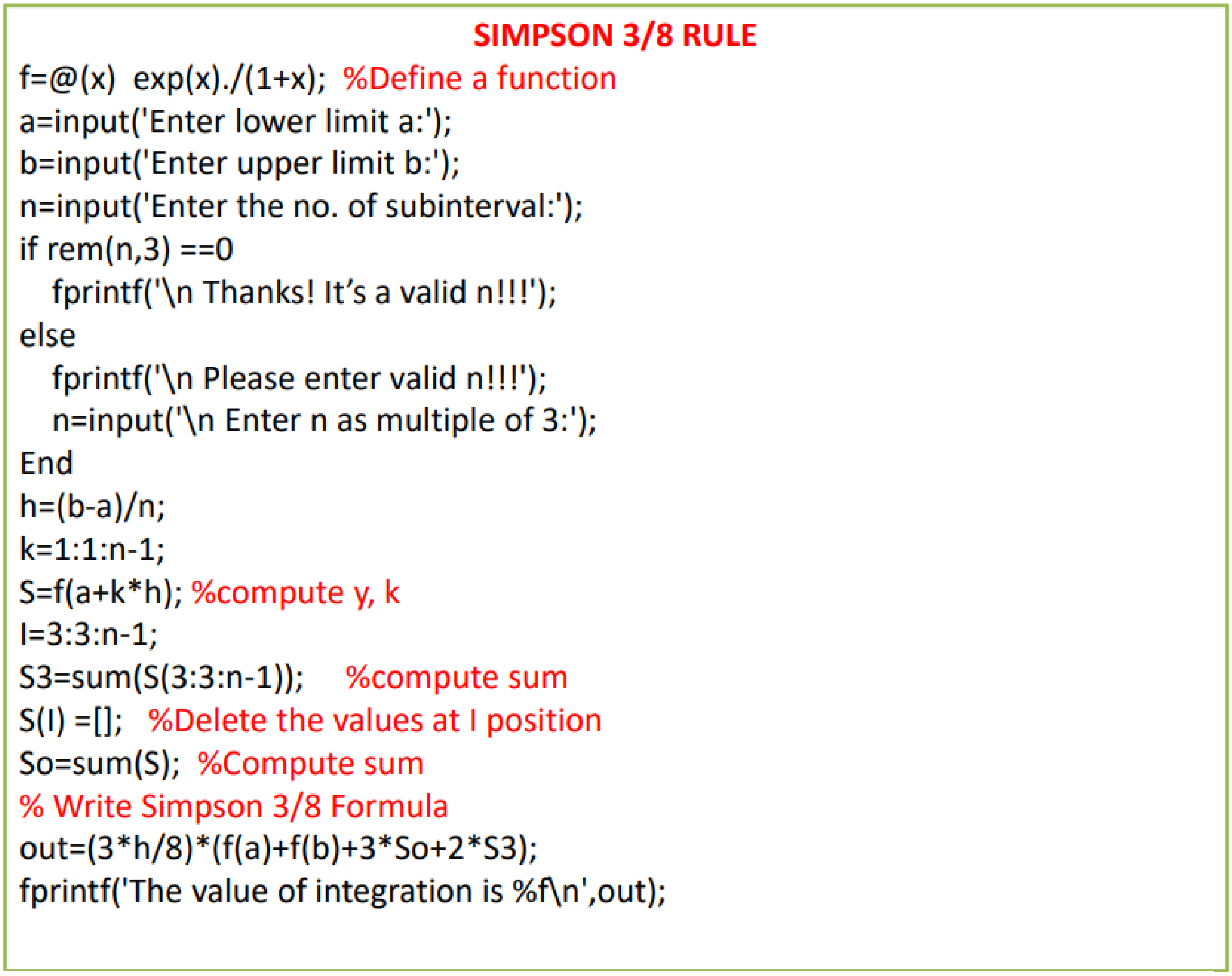
$$\int y dx = \frac{3 \times 1.66666667}{8}[(1 + 2002.40598135) + 2 \times (24.73552652) + 3 \times (1.98543377 + 6.46883651 + 102.49199925 + 445.74235772)]$$

$$\int y dx = \frac{3 \times 1.66666667}{8}[(1 + 2002.40598135) + 2 \times (24.73552652) + 3 \times (556.68862725)]$$

$$\int y dx = 2326.83932258$$

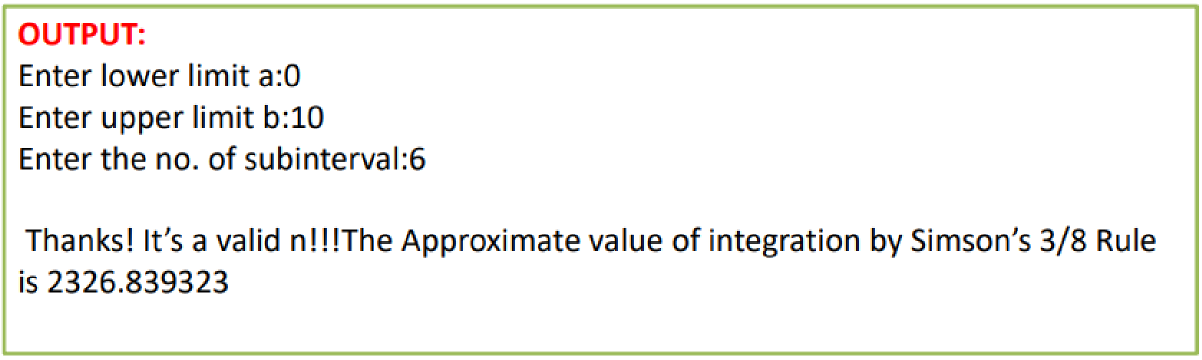
Solution by Simpson's $\frac{3}{8}$ Rule is 2326.83932258

Program Code for Simson’s 3/8 Rule



SIMPSON 3/8 RULE

```
f=@(x)  exp(x)./(1+x);  %Define a function
a=input('Enter lower limit a:');
b=input('Enter upper limit b:');
n=input('Enter the no. of subinterval:');
if rem(n,3) ==0
   fprintf('\n Thanks! It’s a valid n!!!');
else
   fprintf('\n Please enter valid n!!!');
   n=input('\n Enter n as multiple of 3:');
End
h=(b-a)/n;
k=1:1:n-1;
S=f(a+k*h); %compute y, k
l=3:3:n-1;
S3=sum(S(3:3:n-1));     %compute sum
S(l) =[];   %Delete the values at l position
So=sum(S);  %Compute sum
% Write Simpson 3/8 Formula
out=(3*h/8)*(f(a)+f(b)+3*So+2*S3);
fprintf('The value of integration is %f\n',out);
```



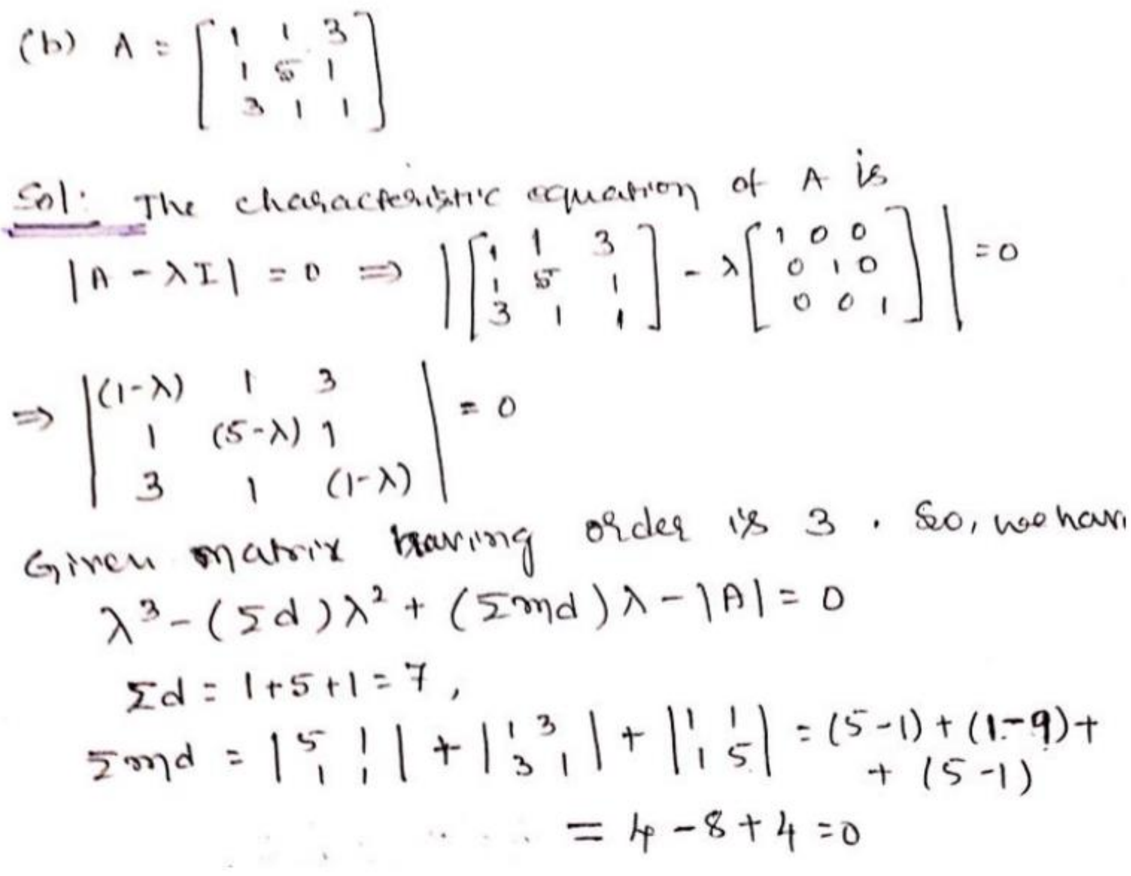
OUTPUT:

```
Enter lower limit a:0
Enter upper limit b:10
Enter the no. of subinterval:6

 Thanks! It’s a valid n!!!The Approximate value of integration by Simson’s 3/8 Rule
is 2326.839323
```

Lab Program 7&8

Linear and Nonlinear Equations: Eigen values, Eigen vectors, Solution of linear algebraic equations using Gauss Elimination and LU decomposition, Solution of nonlinear equation in single variable using Gauss-Siedal and Newton-Raphson method.

Eigen Values and Eigen Vectors:



(b) $A = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 1 \end{bmatrix}$

Sol: The characteristic equation of A is

$$|A - \lambda I| = 0 \Rightarrow \left| \begin{bmatrix} 1 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right| = 0$$

$$\Rightarrow \begin{vmatrix} (1-\lambda) & 1 & 3 \\ 1 & (5-\lambda) & 1 \\ 3 & 1 & (1-\lambda) \end{vmatrix} = 0$$

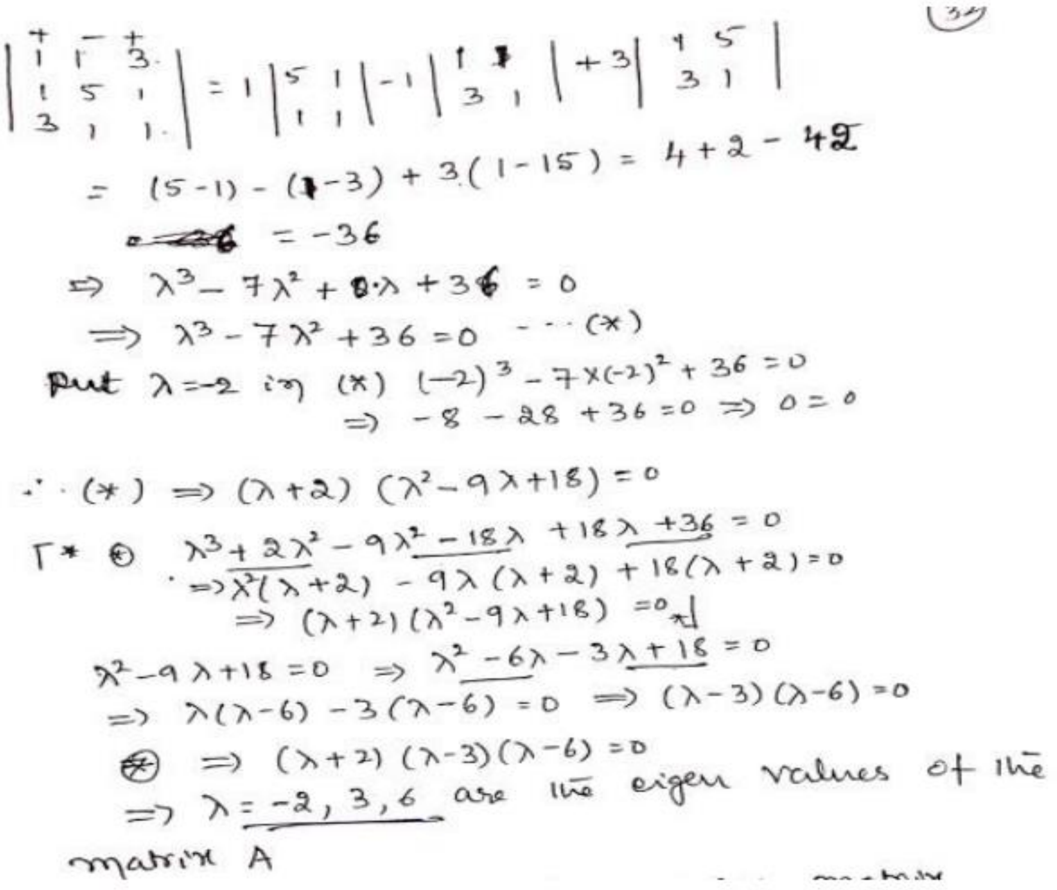
Given matrix having order is 3. So, we have

$$\lambda^3 - (\Sigma d)\lambda^2 + (\Sigma md)\lambda - |A| = 0$$

$\Sigma d = 1+5+1 = 7$,

$$\Sigma md = \begin{vmatrix} 5 & 1 \\ 1 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 3 \\ 3 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 1 \\ 1 & 5 \end{vmatrix} = (5-1) + (1-9) + + (5-1)$$

$$= 4 - 8 + 4 = 0$$



$$\begin{vmatrix} \overset{+}{1} & \overset{-}{1} & \overset{+}{3} \\ 1 & 5 & 1 \\ 3 & 1 & 1 \end{vmatrix} = 1\begin{vmatrix} 5 & 1 \\ 1 & 1 \end{vmatrix} - 1\begin{vmatrix} 1 & 1 \\ 3 & 1 \end{vmatrix} + 3\begin{vmatrix} 1 & 5 \\ 3 & 1 \end{vmatrix}$$

$$= (5-1) - (1-3) + 3(1-15) = 4 + 2 - 42$$

$$\cancel{-26} = -36$$

$$\Rightarrow \lambda^3 - 7\lambda^2 + 0\cdot\lambda + 36 = 0$$

$$\Rightarrow \lambda^3 - 7\lambda^2 + 36 = 0 \quad \cdots (*)$$

Put $\lambda = -2$ in $(*)$ $(-2)^3 - 7\times(-2)^2 + 36 = 0$

$$\Rightarrow -8 - 28 + 36 = 0 \Rightarrow 0 = 0$$

$$\therefore (*) \Rightarrow (\lambda+2)(\lambda^2 - 9\lambda + 18) = 0$$

$$[* \circledast \quad \lambda^3 \underline{+ 2\lambda^2} - 9\lambda^2 \underline{- 18\lambda} + 18\lambda \underline{+ 36} = 0$$

$$\Rightarrow \lambda^2(\lambda+2) - 9\lambda(\lambda+2) + 18(\lambda+2) = 0$$

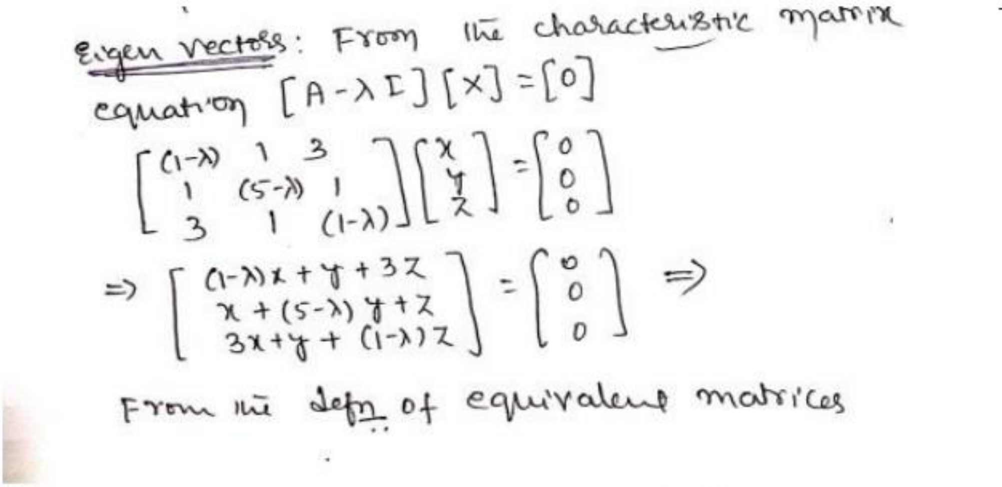
$$\Rightarrow (\lambda+2)(\lambda^2 - 9\lambda + 18) = 0 *]$$

$$\lambda^2 - 9\lambda + 18 = 0 \Rightarrow \lambda^2 \underline{-6\lambda} - 3\underline{\lambda + 18} = 0$$

$$\Rightarrow \lambda(\lambda-6) - 3(\lambda-6) = 0 \Rightarrow (\lambda-3)(\lambda-6) = 0$$

$$\circledast \Rightarrow (\lambda+2)(\lambda-3)(\lambda-6) = 0$$

$\Rightarrow \lambda = \underline{-2, 3, 6}$ are the eigen values of the matrix A

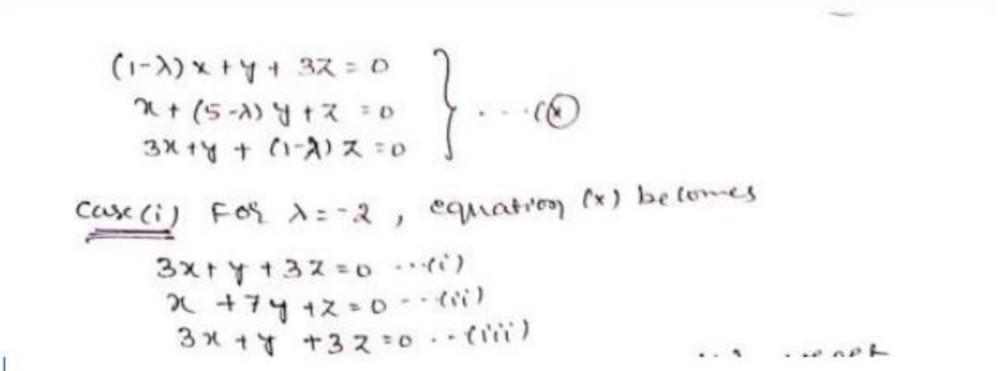


Eigen vectors: From the characteristic matrix equation $[A-\lambda I][X]=[0]$

$$\begin{bmatrix} (1-\lambda) & 1 & 3 \\ 1 & (5-\lambda) & 1 \\ 3 & 1 & (1-\lambda) \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} (1-\lambda)x+y+3z \\ x+(5-\lambda)y+z \\ 3x+y+(1-\lambda)z \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow$$

From the defn of equivalent matrices



$$\left.\begin{aligned} (1-\lambda)x+y+3z&=0 \\ x+(5-\lambda)y+z&=0 \\ 3x+y+(1-\lambda)z&=0 \end{aligned}\right\} \cdots \boxed{(x)}$$

Case (i) For $\lambda=-2$, equation (x) becomes

$3x+y+3z=0 \cdots(i)$

$x+7y+z=0 \cdots(ii)$

$3x+y+3z=0 \cdots(iii)$

By cross multiplication on (i) & (ii), we get
⟨ ∵ equations (i) & (iii) are same ⟩

$$\frac{x}{(1\times1-7\times3)}=\frac{-y}{(3\times1-1\times3)}=\frac{z}{(3\times7-1\times1)}$$

$$\Rightarrow \frac{x}{-20}=\frac{-y}{0}=\frac{z}{20} \Rightarrow \frac{x}{-1}=\frac{y}{0}=\frac{z}{1}$$

$X_1=\begin{bmatrix}-1\\0\\1\end{bmatrix}$ is the eigen vector corresponding to $\lambda=-2$

Case(ii) For $\lambda=3$, equation (*) becomes

$-2x+y+3z=0$ ---(iv)

$x+2y+z=0$ ---(v)

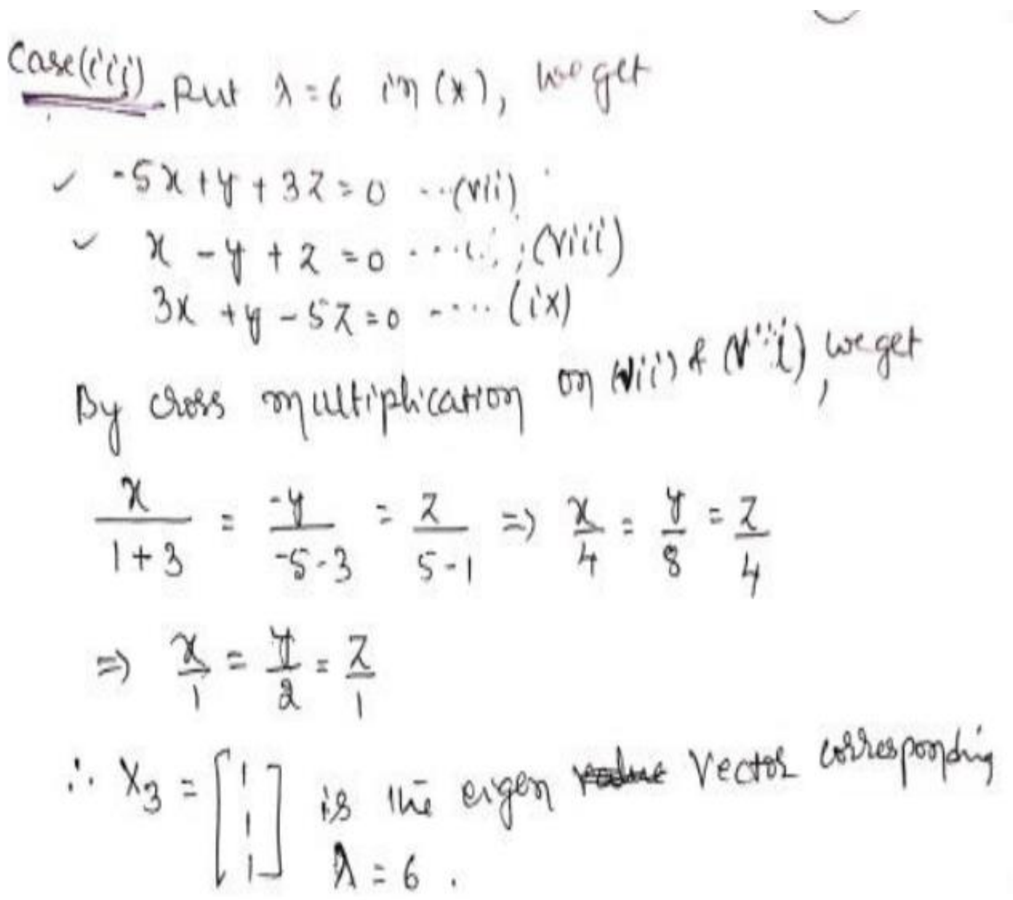
$3x+y-2z=0$ ---(vi)

By cross multiplication on (iv) & (v)

$$\frac{x}{1-6}=\frac{-y}{-2-3}=\frac{z}{-4-1}$$

$$\frac{x}{-5}=\frac{y}{5}=\frac{z}{-5} \Rightarrow \frac{x}{1}=\frac{y}{-1}=\frac{z}{1}$$

$X_2=\begin{bmatrix}1\\-1\\1\end{bmatrix}$ is the eigen ~~vect~~ vector corresponding to $\lambda=3$.



Case(iii) Put $\lambda=6$ in (x), we get

✓ $-5x+y+3z=0$ ···(vii)

✓ $x-y+z=0$ ······(viii)

$3x+y-5z=0$ ····(ix)

By cross multiplication on (vii) & (viii), we get

$$\frac{x}{1+3}=\frac{-y}{-5-3}=\frac{z}{5-1} \Rightarrow \frac{x}{4}=\frac{y}{8}=\frac{z}{4}$$

$$\Rightarrow \frac{x}{1}=\frac{y}{2}=\frac{z}{1}$$

$\therefore X_3=\begin{bmatrix}1\\1\\1\end{bmatrix}$ is the eigen ~~value~~ vector corresponding $\lambda=6$.

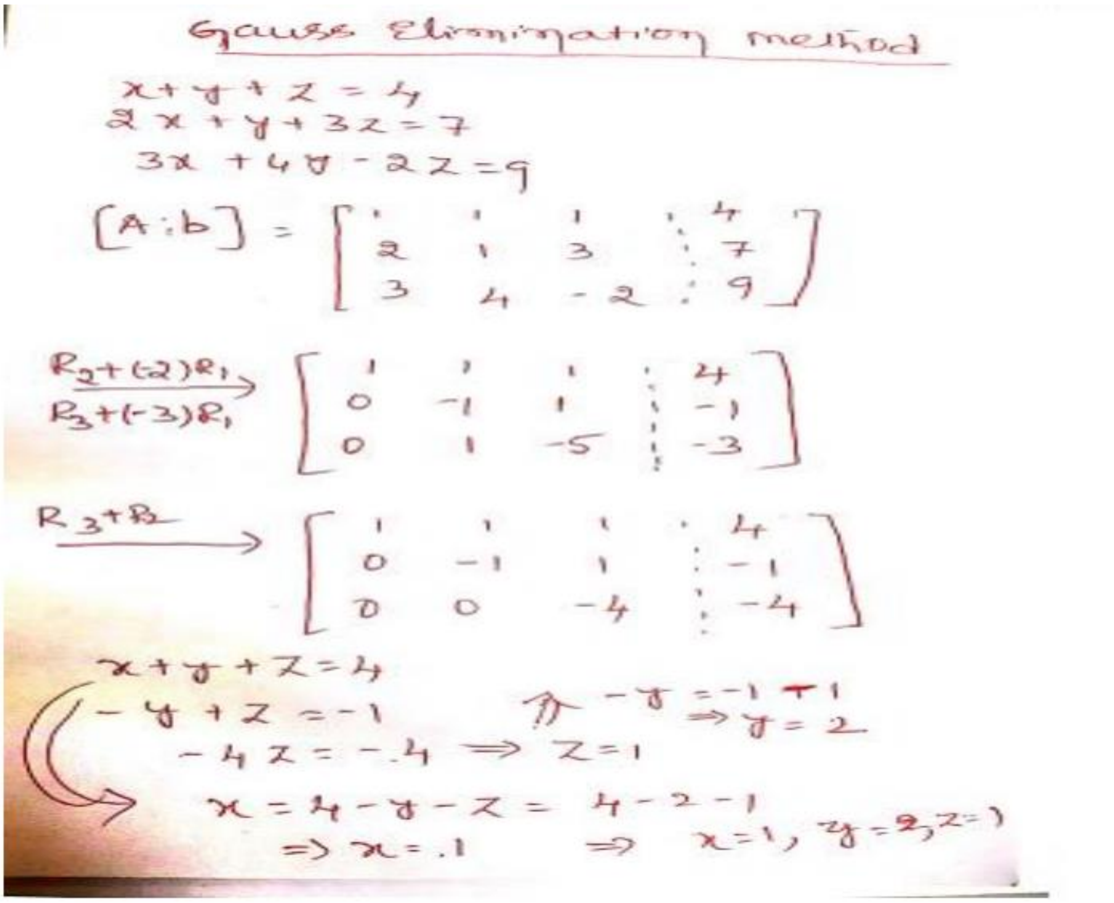
Lab Program 7: Program code for Egien values and Eigen vectors

```
A=[8  -6   2;
     -6   7  -4;
      2  -4    3];
 disp("Matrix");
 disp(A)
 %Eigenvalues and right eigenvectors of matrix A
 [V,D]=eig(A)
```

OUTPUT:

```
Matrix
     8    -6     2
    -6     7    -4
     2    -4     3
V =
    0.3333    0.6667   -0.6667
    0.6667    0.3333    0.6667
    0.6667   -0.6667   -0.3333
D =
    0.0000         0         0
         0    3.0000         0
         0         0   15.0000
```

Gauss Elimination Method:



Gauss Elimination method

$x+y+z=4$

$2x+y+3z=7$

$3x+4y-2z=9$

$$[A:b] = \left[\begin{array}{ccc:c} 1 & 1 & 1 & 4 \\ 2 & 1 & 3 & 7 \\ 3 & 4 & -2 & 9 \end{array}\right]$$

$$\xrightarrow[R_3+(-3)R_1]{R_2+(-2)R_1} \left[\begin{array}{ccc:c} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & 1 & -5 & -3 \end{array}\right]$$

$$\xrightarrow{R_3+R_2} \left[\begin{array}{ccc:c} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & -4 & -4 \end{array}\right]$$

$x+y+z=4$

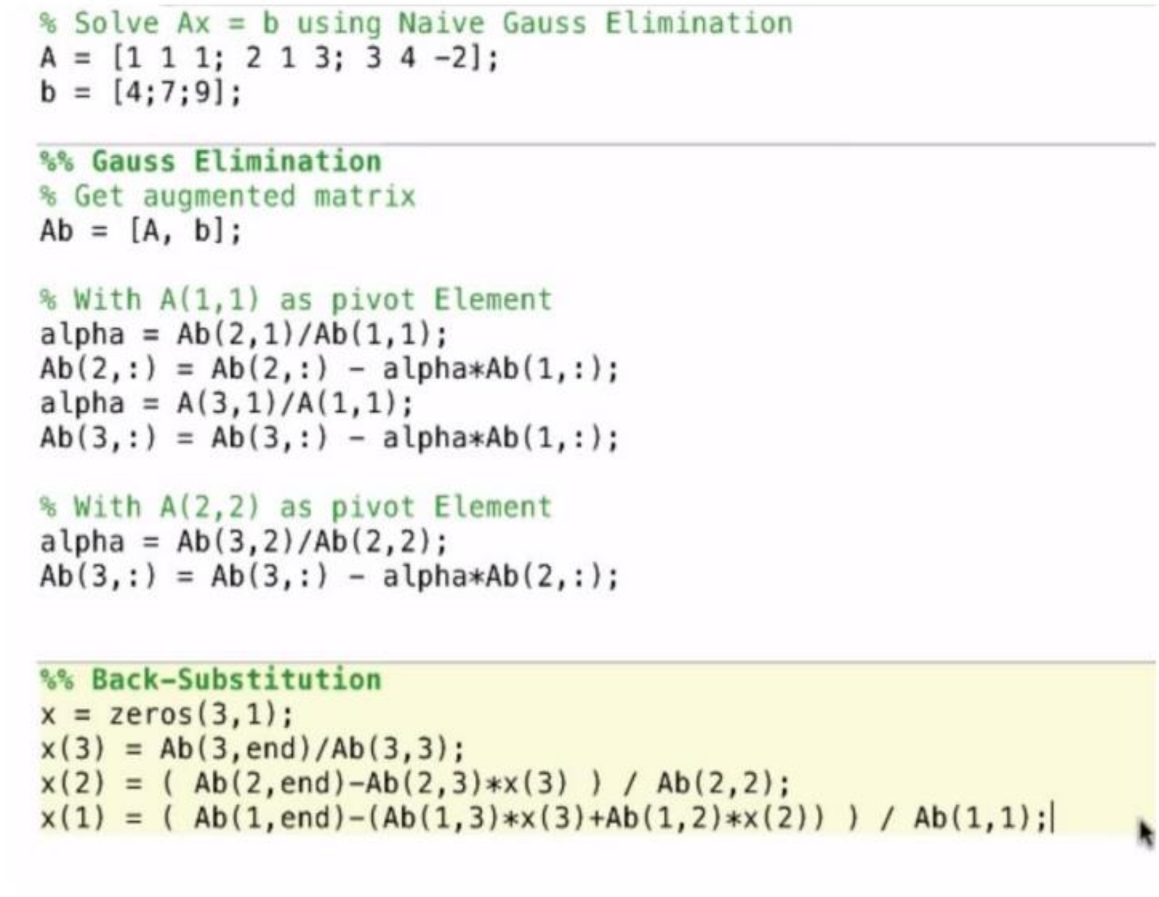
$-y+z=-1$ $\quad\Uparrow\quad -y=-1-1 \Rightarrow y=2$

$-4z=-4 \Rightarrow z=1$

$x=4-y-z = 4-2-1$

$\Rightarrow x=1 \quad \Rightarrow x=1,\ y=2,\ z=1$

Lab Program 8a: Guass Elimination Method using back substitution

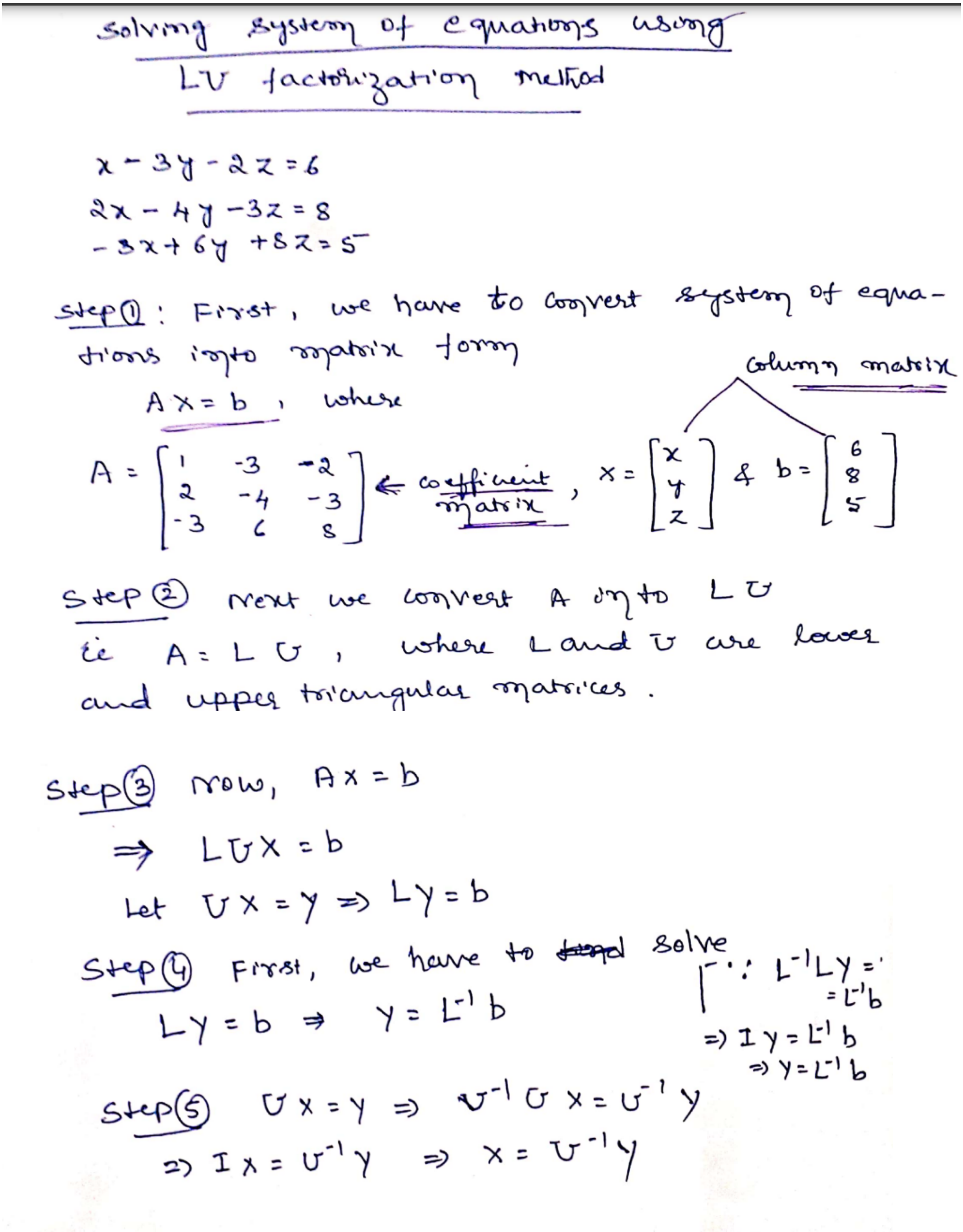


```
% Solve Ax = b using Naive Gauss Elimination
A = [1 1 1; 2 1 3; 3 4 -2];
b = [4;7;9];

%% Gauss Elimination
% Get augmented matrix
Ab = [A, b];

% With A(1,1) as pivot Element
alpha = Ab(2,1)/Ab(1,1);
Ab(2,:) = Ab(2,:) - alpha*Ab(1,:);
alpha = A(3,1)/A(1,1);
Ab(3,:) = Ab(3,:) - alpha*Ab(1,:);

% With A(2,2) as pivot Element
alpha = Ab(3,2)/Ab(2,2);
Ab(3,:) = Ab(3,:) - alpha*Ab(2,:);


%% Back-Substitution
x = zeros(3,1);
x(3) = Ab(3,end)/Ab(3,3);
x(2) = ( Ab(2,end)-Ab(2,3)*x(3) ) / Ab(2,2);
x(1) = ( Ab(1,end)-(Ab(1,3)*x(3)+Ab(1,2)*x(2)) ) / Ab(1,1);
```

OUTPUT:

```
x =

    1
    2
    1
```

Lab Program 8b: LU Decomposition



Solving system of equations using LU factorization method

$$x - 3y - 2z = 6$$
$$2x - 4y - 3z = 8$$
$$-3x + 6y + 8z = 5$$

Step ①: First, we have to convert system of equations into matrix form

$Ax = b$, where

Column matrix

$$A = \begin{bmatrix} 1 & -3 & -2 \\ 2 & -4 & -3 \\ -3 & 6 & 8 \end{bmatrix} \Leftarrow \text{coefficient matrix}, \quad x = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \ \& \ b = \begin{bmatrix} 6 \\ 8 \\ 5 \end{bmatrix}$$

Step ② next we convert A into LU

ie $A = LU$, where L and U are lower and upper triangular matrices.

Step ③ now, $Ax = b$

$$\Rightarrow LUx = b$$

Let $Ux = y \Rightarrow Ly = b$

Step ④ First, we have to ~~[illegible]~~ solve

$Ly = b \Rightarrow y = L^{-1}b$

$\because L^{-1}Ly = L^{-1}b$

$\Rightarrow Iy = L^{-1}b$

$\Rightarrow y = L^{-1}b$

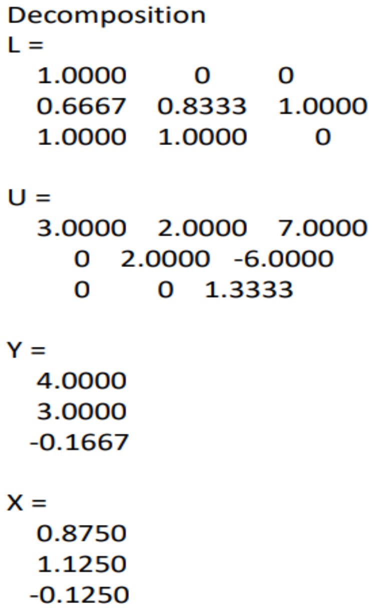
Step ⑤ $Ux = y \Rightarrow U^{-1}Ux = U^{-1}y$

$\Rightarrow Ix = U^{-1}y \Rightarrow x = U^{-1}y$

Program Code 8b: for LU Decomposition

```
% A = coefficient matrix
% b = right hand side vector
% X = solution vector
A = [1 2 3; 4 5 6; 7 8 9];
b=[4; 5; 6];
[m,n]=size(A);
if m~=n
    fprintf ("Matrix A must be square")
end
[L, U] = lu(A);  % LU factorization
disp("L =  ");
disp(L)
disp("U =  ");
disp(U)
Y = L\b ;                   % solution of Y
disp("Y =   ");
disp(Y)
X=U\Y ;                            % solution of X
disp("X =  ");
disp(X)
```

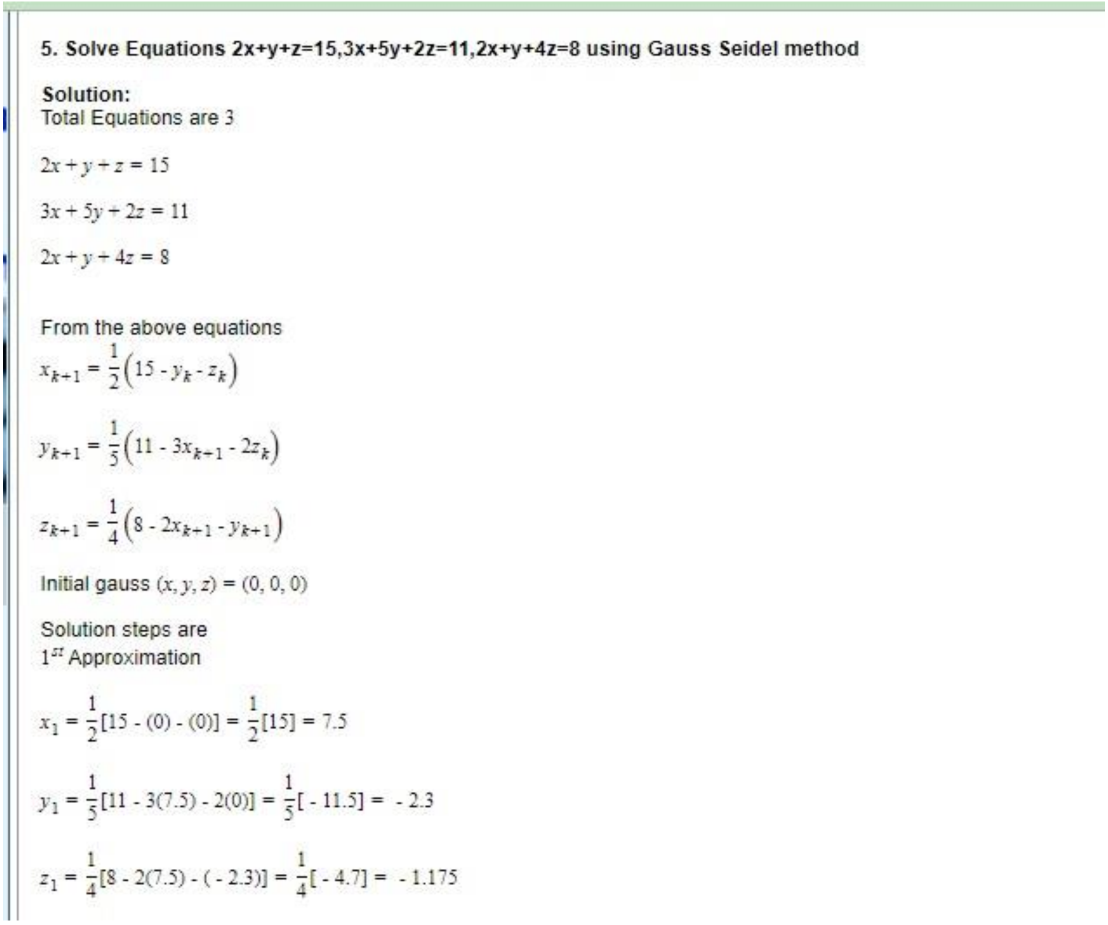
OUTPUT



```
Decomposition
L =
    1.0000         0         0
    0.6667    0.8333    1.0000
    1.0000    1.0000         0

U =
    3.0000    2.0000    7.0000
         0    2.0000   -6.0000
         0         0    1.3333

Y =
    4.0000
    3.0000
   -0.1667

X =
    0.8750
    1.1250
   -0.1250
```

Lab Program 8c: Solution of nonlinear equation in single variable using Gauss-Siedal

Gauss Seidal Method



5. Solve Equations 2x+y+z=15,3x+5y+2z=11,2x+y+4z=8 using Gauss Seidel method

Solution:
Total Equations are 3

$2x + y + z = 15$

$3x + 5y + 2z = 11$

$2x + y + 4z = 8$

From the above equations

$$x_{k+1} = \frac{1}{2}\left(15 - y_k - z_k\right)$$

$$y_{k+1} = \frac{1}{5}\left(11 - 3x_{k+1} - 2z_k\right)$$

$$z_{k+1} = \frac{1}{4}\left(8 - 2x_{k+1} - y_{k+1}\right)$$

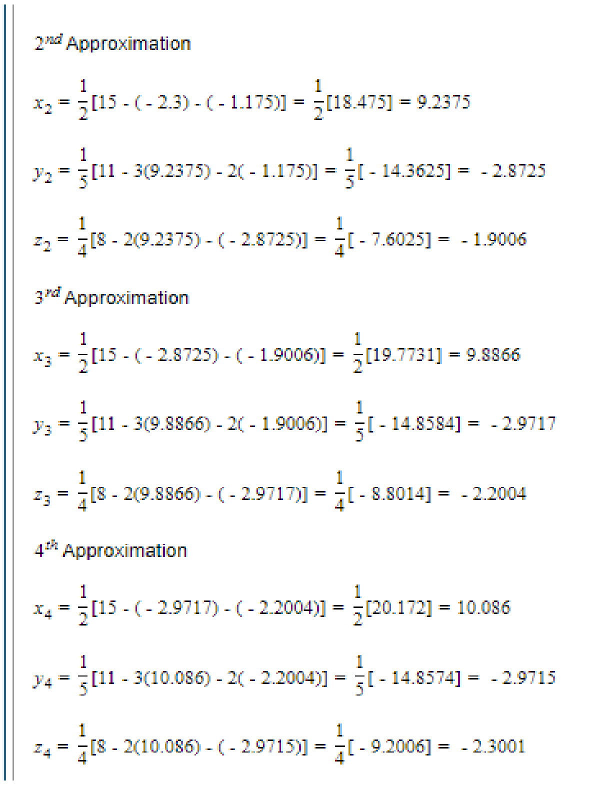
Initial gauss $(x, y, z) = (0, 0, 0)$

Solution steps are
1^{st} Approximation

$$x_1 = \frac{1}{2}[15 - (0) - (0)] = \frac{1}{2}[15] = 7.5$$

$$y_1 = \frac{1}{5}[11 - 3(7.5) - 2(0)] = \frac{1}{5}[-11.5] = -2.3$$

$$z_1 = \frac{1}{4}[8 - 2(7.5) - (-2.3)] = \frac{1}{4}[-4.7] = -1.175$$



2^{nd} Approximation

$$x_2 = \frac{1}{2}[15 - (-2.3) - (-1.175)] = \frac{1}{2}[18.475] = 9.2375$$

$$y_2 = \frac{1}{5}[11 - 3(9.2375) - 2(-1.175)] = \frac{1}{5}[-14.3625] = -2.8725$$

$$z_2 = \frac{1}{4}[8 - 2(9.2375) - (-2.8725)] = \frac{1}{4}[-7.6025] = -1.9006$$

3^{rd} Approximation

$$x_3 = \frac{1}{2}[15 - (-2.8725) - (-1.9006)] = \frac{1}{2}[19.7731] = 9.8866$$

$$y_3 = \frac{1}{5}[11 - 3(9.8866) - 2(-1.9006)] = \frac{1}{5}[-14.8584] = -2.9717$$

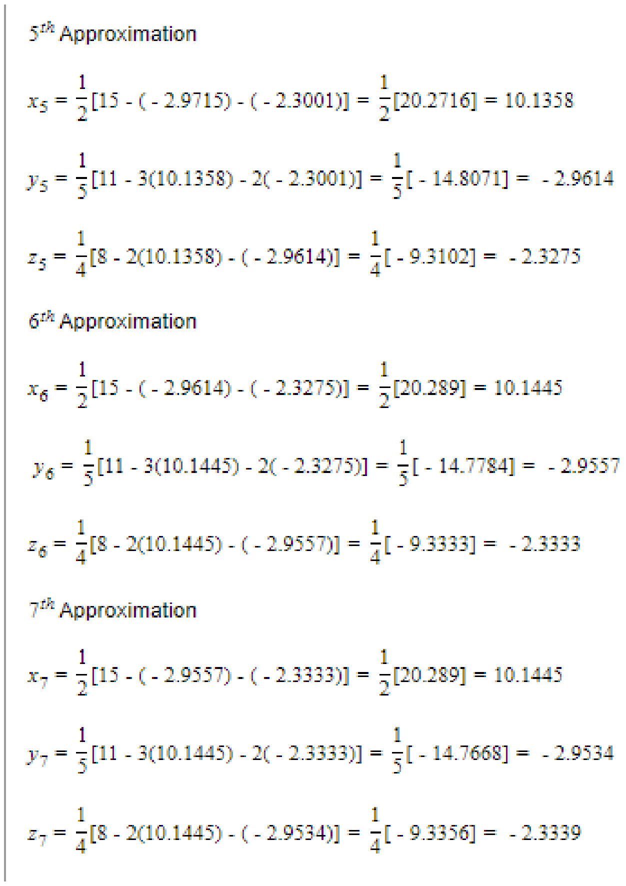
$$z_3 = \frac{1}{4}[8 - 2(9.8866) - (-2.9717)] = \frac{1}{4}[-8.8014] = -2.2004$$

4^{th} Approximation

$$x_4 = \frac{1}{2}[15 - (-2.9717) - (-2.2004)] = \frac{1}{2}[20.172] = 10.086$$

$$y_4 = \frac{1}{5}[11 - 3(10.086) - 2(-2.2004)] = \frac{1}{5}[-14.8574] = -2.9715$$

$$z_4 = \frac{1}{4}[8 - 2(10.086) - (-2.9715)] = \frac{1}{4}[-9.2006] = -2.3001$$



5^{th} Approximation

$$x_5 = \frac{1}{2}[15 - (-2.9715) - (-2.3001)] = \frac{1}{2}[20.2716] = 10.1358$$

$$y_5 = \frac{1}{5}[11 - 3(10.1358) - 2(-2.3001)] = \frac{1}{5}[-14.8071] = -2.9614$$

$$z_5 = \frac{1}{4}[8 - 2(10.1358) - (-2.9614)] = \frac{1}{4}[-9.3102] = -2.3275$$

6^{th} Approximation

$$x_6 = \frac{1}{2}[15 - (-2.9614) - (-2.3275)] = \frac{1}{2}[20.289] = 10.1445$$

$$y_6 = \frac{1}{5}[11 - 3(10.1445) - 2(-2.3275)] = \frac{1}{5}[-14.7784] = -2.9557$$

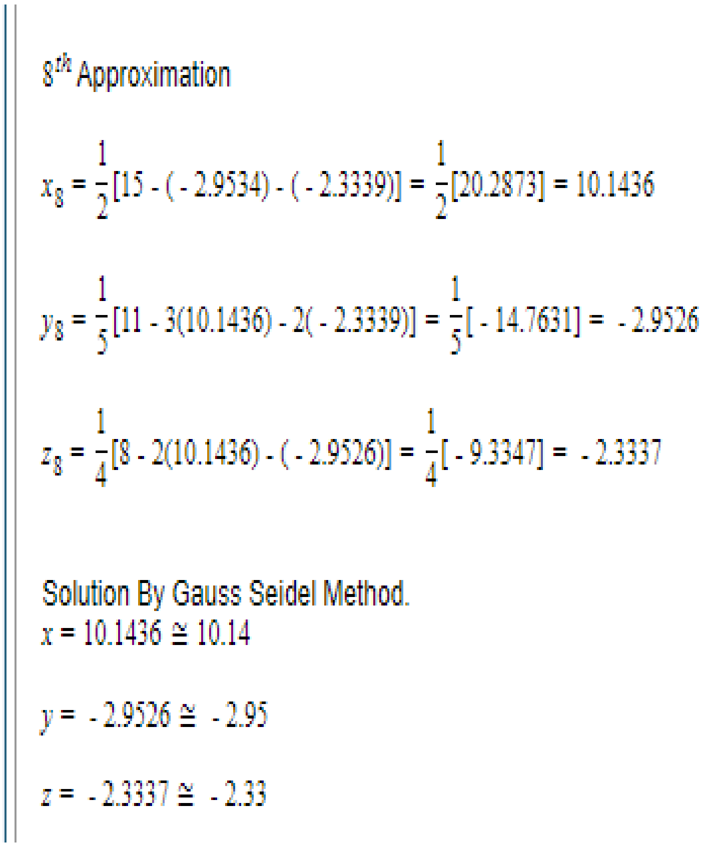
$$z_6 = \frac{1}{4}[8 - 2(10.1445) - (-2.9557)] = \frac{1}{4}[-9.3333] = -2.3333$$

7^{th} Approximation

$$x_7 = \frac{1}{2}[15 - (-2.9557) - (-2.3333)] = \frac{1}{2}[20.289] = 10.1445$$

$$y_7 = \frac{1}{5}[11 - 3(10.1445) - 2(-2.3333)] = \frac{1}{5}[-14.7668] = -2.9534$$

$$z_7 = \frac{1}{4}[8 - 2(10.1445) - (-2.9534)] = \frac{1}{4}[-9.3356] = -2.3339$$



8^{th} Approximation

$$x_8 = \frac{1}{2}[15 - (-2.9534) - (-2.3339)] = \frac{1}{2}[20.2873] = 10.1436$$

$$y_8 = \frac{1}{5}[11 - 3(10.1436) - 2(-2.3339)] = \frac{1}{5}[-14.7631] = -2.9526$$

$$z_8 = \frac{1}{4}[8 - 2(10.1436) - (-2.9526)] = \frac{1}{4}[-9.3347] = -2.3337$$

Solution By Gauss Seidel Method.

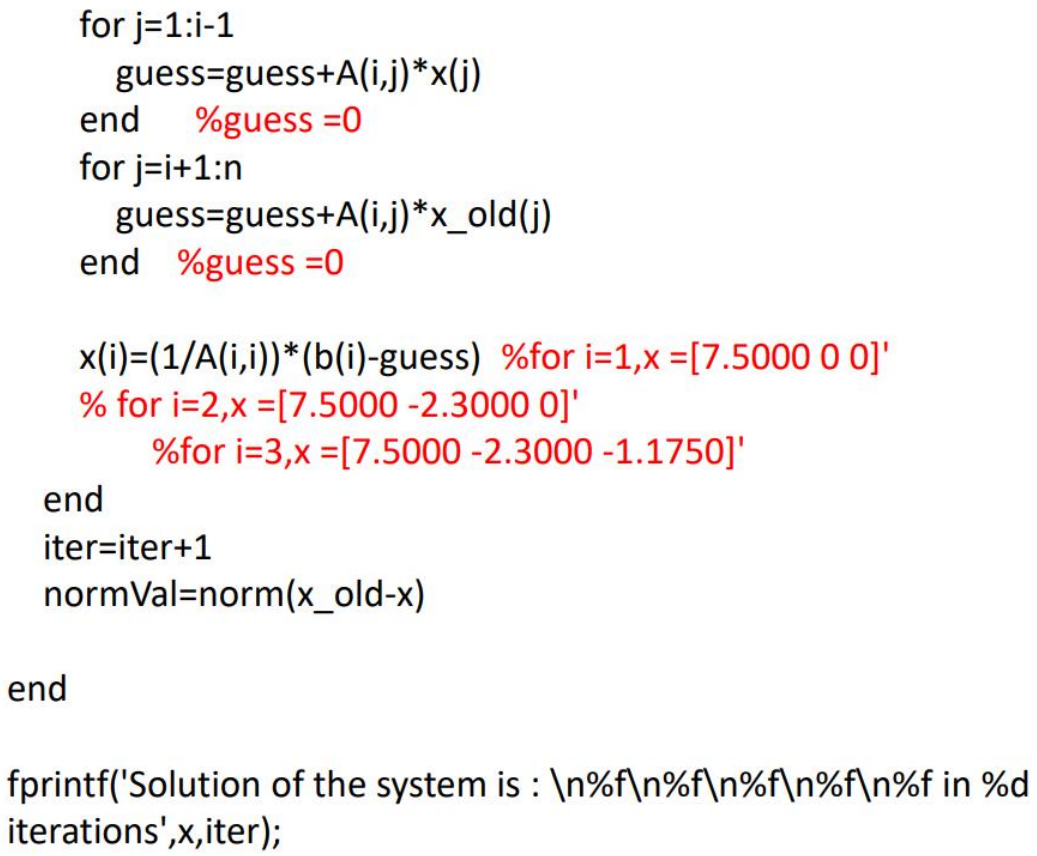
$x = 10.1436 \cong 10.14$

$y = -2.9526 \cong -2.95$

$z = -2.3337 \cong -2.33$

Lab Program 8c: Code for Gauss-Siedal

```
clc
clear
close all
A=[2 1 1;3 5 2;2 1 4];
b=[15 11 8]';
x=linspace(0,0,length(A))'
%x =[0 0 0]'
n=size(x,1) %n=3
normVal=Inf
nmax=1000 %number of maximum iterations which can be reached%
tol=1e-3 % Tolerence for method%
iter=0
while normVal>tol && iter<nmax
   x_old=x;

   for i=1:n

      guess=0
```

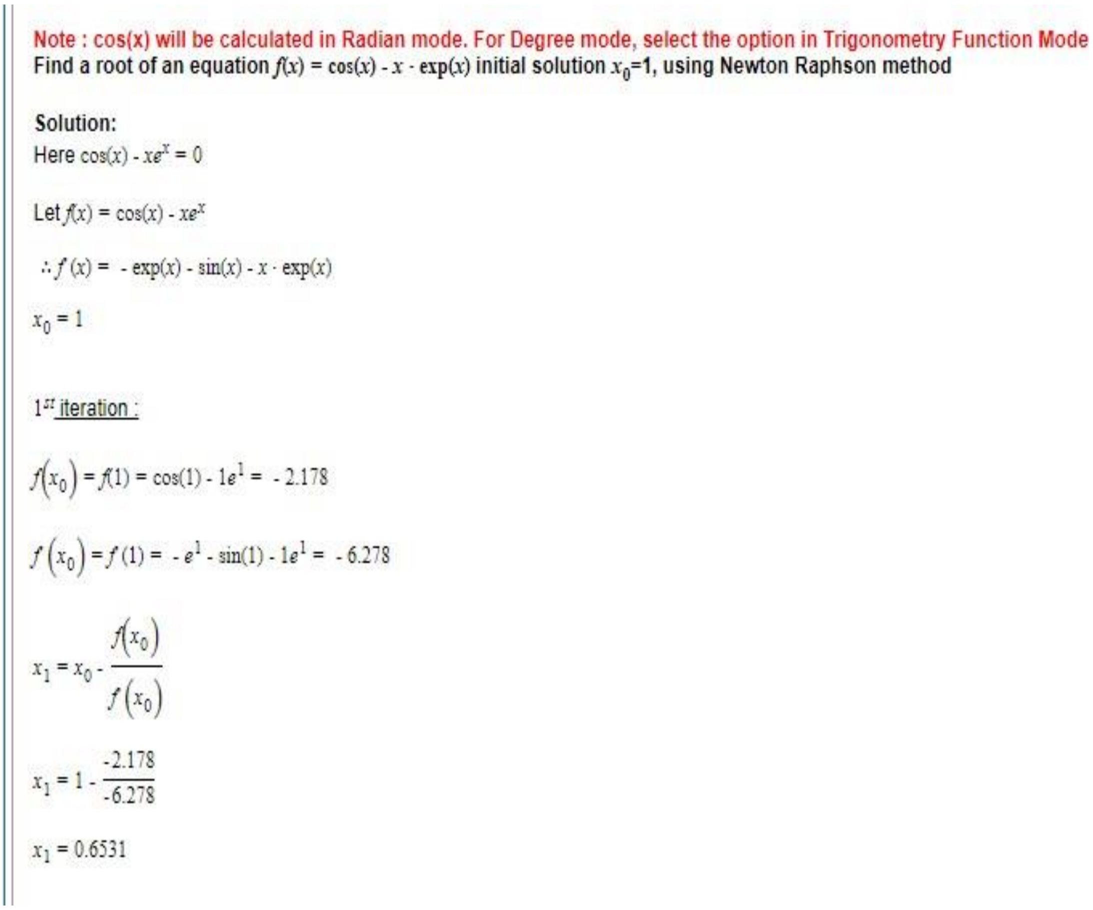


```
        for j=1:i-1
           guess=guess+A(i,j)*x(j)
        end     %guess =0
        for j=i+1:n
           guess=guess+A(i,j)*x_old(j)
        end   %guess =0

        x(i)=(1/A(i,i))*(b(i)-guess)  %for i=1,x =[7.5000 0 0]'
        % for i=2,x =[7.5000 -2.3000 0]'
               %for i=3,x =[7.5000 -2.3000 -1.1750]'
    end
    iter=iter+1
    normVal=norm(x_old-x)

end

fprintf('Solution of the system is : \n%f\n%f\n%f\n%f\n%f in %d
iterations',x,iter);
```

OUTPUT:

```
Solution of the system is :
10.143142
-2.952420
-2.333466
9.000000
```

Lab-Program 8d: Solution of nonlinear equation in single variable using Newton Raphson Method



Note : cos(x) will be calculated in Radian mode. For Degree mode, select the option in Trigonometry Function Mode

Find a root of an equation $f(x) = \cos(x) - x \cdot \exp(x)$ initial solution x_0=1, using Newton Raphson method

Solution:

Here $\cos(x) - xe^x = 0$

Let $f(x) = \cos(x) - xe^x$

$\therefore f'(x) = - \exp(x) - \sin(x) - x \cdot \exp(x)$

$x_0 = 1$

1^{st} iteration :

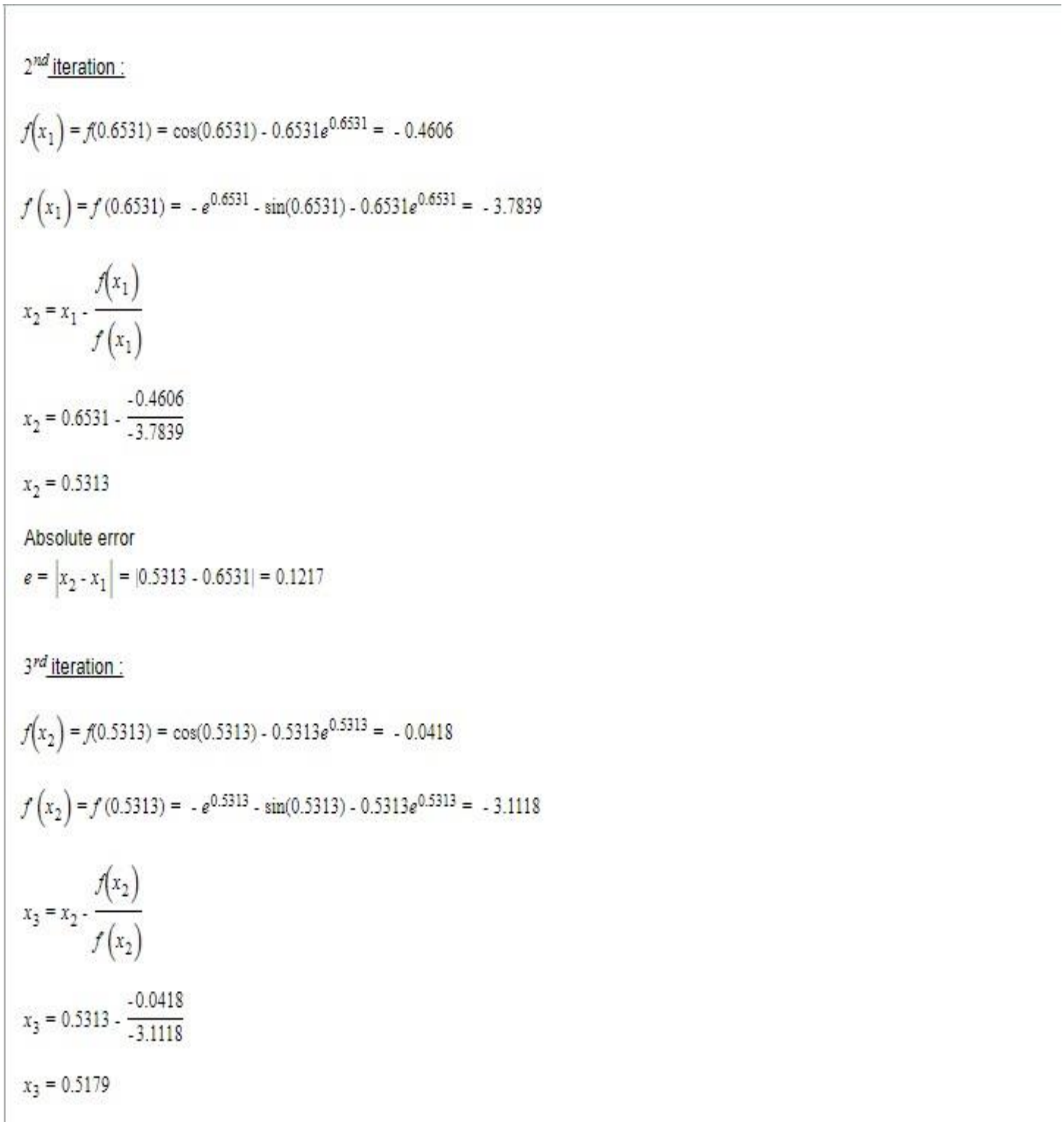
$f(x_0) = f(1) = \cos(1) - 1e^1 = - 2.178$

$f'(x_0) = f'(1) = - e^1 - \sin(1) - 1e^1 = - 6.278$

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

$$x_1 = 1 - \frac{-2.178}{-6.278}$$

$x_1 = 0.6531$



2^{nd} iteration :

$$f\left(x_1\right) = f(0.6531) = \cos(0.6531) - 0.6531e^{0.6531} = -0.4606$$

$$f\left(x_1\right) = f(0.6531) = -e^{0.6531} - \sin(0.6531) - 0.6531e^{0.6531} = -3.7839$$

$$x_2 = x_1 - \frac{f\left(x_1\right)}{f\left(x_1\right)}$$

$$x_2 = 0.6531 - \frac{-0.4606}{-3.7839}$$

$$x_2 = 0.5313$$

Absolute error

$$e = \left|x_2 - x_1\right| = |0.5313 - 0.6531| = 0.1217$$

3^{rd} iteration :

$$f\left(x_2\right) = f(0.5313) = \cos(0.5313) - 0.5313e^{0.5313} = -0.0418$$

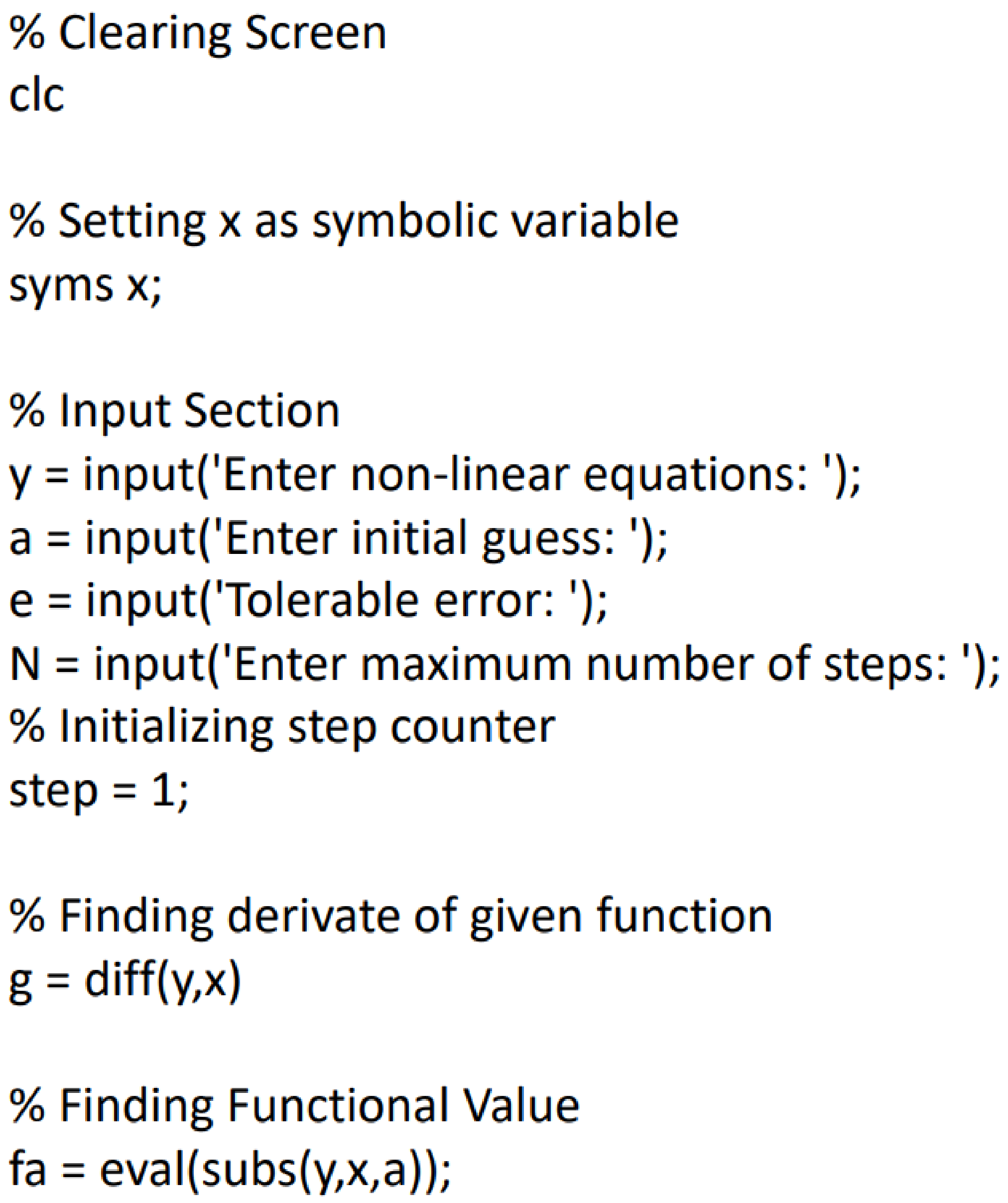
$$f\left(x_2\right) = f(0.5313) = -e^{0.5313} - \sin(0.5313) - 0.5313e^{0.5313} = -3.1118$$

$$x_3 = x_2 - \frac{f\left(x_2\right)}{f\left(x_2\right)}$$

$$x_3 = 0.5313 - \frac{-0.0418}{-3.1118}$$

$$x_3 = 0.5179$$

Lab Program -8d: Code using Newton Raphson Method



```
% Clearing Screen
clc

% Setting x as symbolic variable
syms x;

% Input Section
y = input('Enter non-linear equations: ');
a = input('Enter initial guess: ');
e = input('Tolerable error: ');
N = input('Enter maximum number of steps: ');
% Initializing step counter
step = 1;

% Finding derivate of given function
g = diff(y,x)

% Finding Functional Value
fa = eval(subs(y,x,a));
```

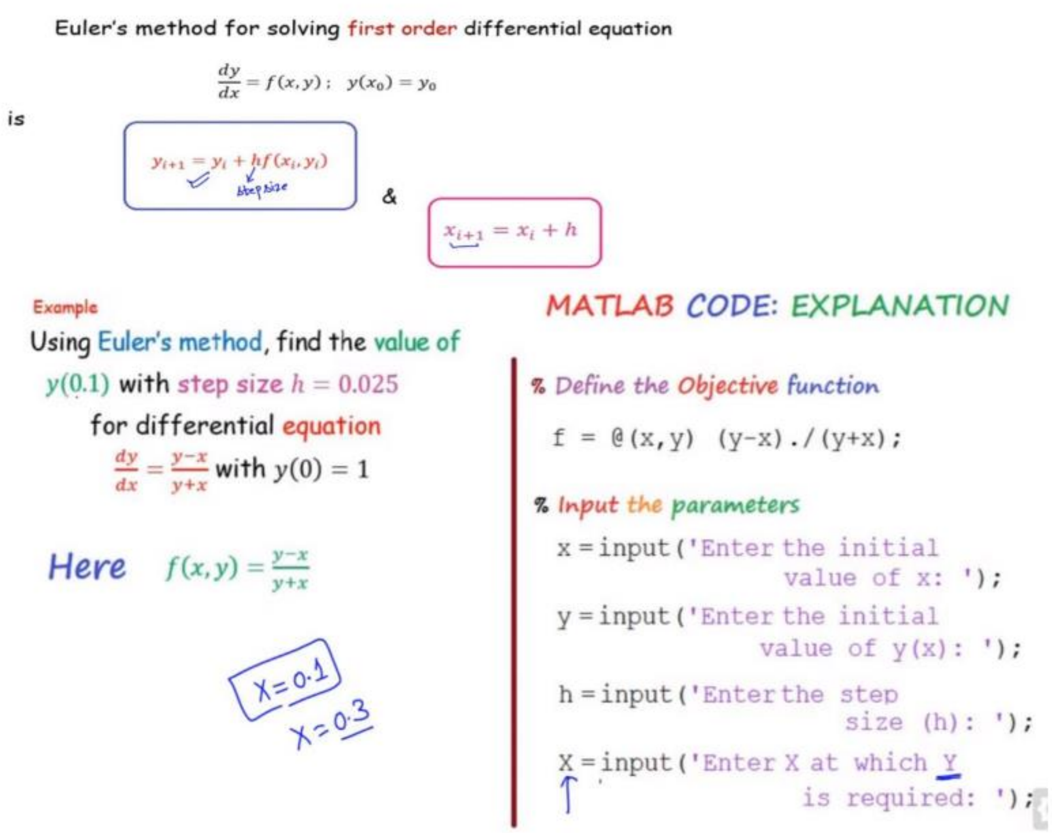
```
while abs(fa)> e
  fa = eval(subs(y,x,a));
  ga = eval(subs(g,x,a));
  if ga == 0
    disp('Division by zero.');
    break;
  end

  b = a - fa/ga;
  fprintf('step=%d\ta=%f\tf(a)=%f\n',step,a,fa);
  a = b;

  if step>N
   disp('Not convergent');
   break;
  end
  step = step + 1;
end

fprintf('Root is %f\n', a);
```

```
OUTPUT:


Enter non-linear equations:  cos(x)-x*exp(x)
Enter initial guess: 1
Tolerable error: 0.00001
Enter maximum number of steps: 20


g =


- exp(x) - sin(x) - x*exp(x)


step=1          a=1.000000  f(a)=-2.177980
step=2          a=0.653079  f(a)=-0.460642
step=3          a=0.531343  f(a)=-0.041803
step=4          a=0.517910  f(a)=-0.000464
step=5          a=0.517757  f(a)=-0.000000
Root is 0.517757
```

Lab Program 9&10

Ordinary Differential Equations: Introduction to ODE’s, Euler’s method, second order RungaKutta method, 10 MATLAB ODE45 algorithm in single variable and multivariables. Transforms: Discrete Fourier Transforms.

Lab Program 9a: Euler’ Methods



Euler's method for solving first order differential equation

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

is

$$\boxed{y_{i+1}=y_i+\underset{\text{Step size}}{h}f(x_i,y_i)}\quad \& \quad \boxed{x_{i+1}=x_i+h}$$

Example

Using Euler's method, find the value of $y(0.1)$ with step size $h=0.025$ for differential equation $\frac{dy}{dx}=\frac{y-x}{y+x}$ with $y(0)=1$

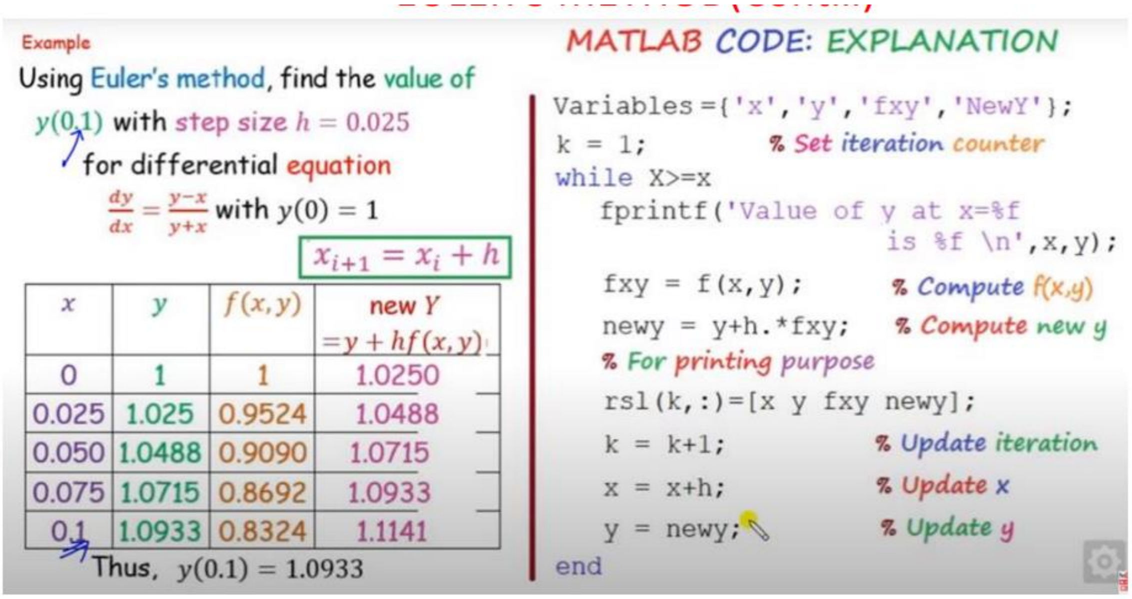
Here $f(x,y)=\frac{y-x}{y+x}$

$\boxed{X=0.1}$

$X=0.3$

MATLAB CODE: EXPLANATION

```
% Define the Objective function
 f = @(x,y) (y-x)./(y+x);

% Input the parameters
 x=input('Enter the initial
            value of x: ');
 y=input('Enter the initial
            value of y(x): ');
 h=input('Enter the step
            size (h): ');
 X=input('Enter X at which Y
            is required: ');
```



Example

Using Euler's method, find the value of $y(0.1)$ with step size $h=0.025$ for differential equation $\frac{dy}{dx}=\frac{y-x}{y+x}$ with $y(0)=1$

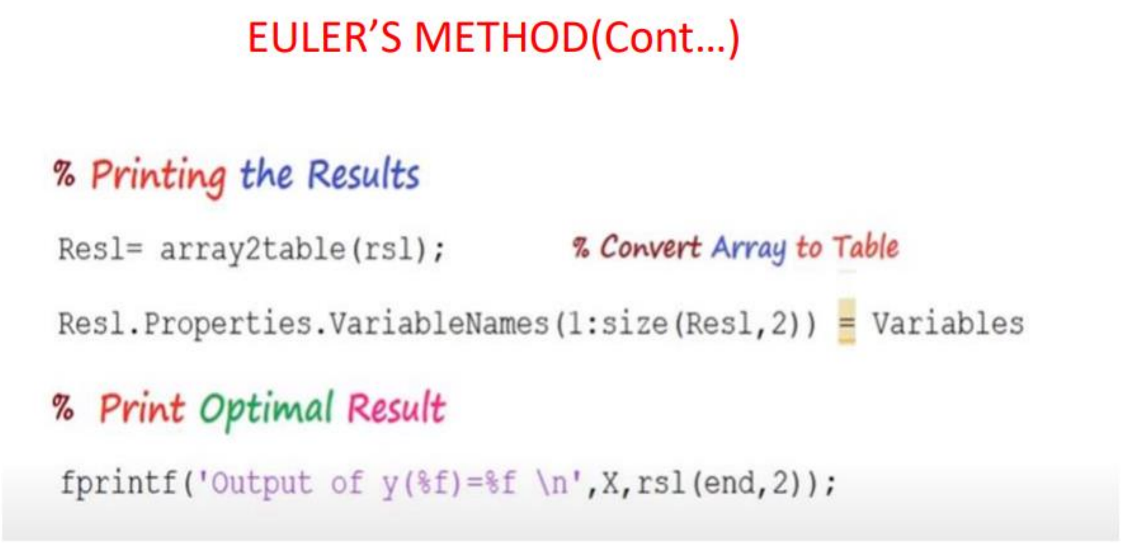
$\boxed{x_{i+1}=x_i+h}$

x	y	$f(x,y)$	new Y $=y+hf(x,y)$
0	1	1	1.0250
0.025	1.025	0.9524	1.0488
0.050	1.0488	0.9090	1.0715
0.075	1.0715	0.8692	1.0933
0.1	1.0933	0.8324	1.1141

Thus, $y(0.1)=1.0933$

MATLAB CODE: EXPLANATION

```
Variables={'x','y','fxy','NewY'};
k = 1;          % Set iteration counter
while X>=x
    fprintf('Value of y at x=%f
                is %f \n',x,y);

    fxy = f(x,y);         % Compute f(x,y)
    newy = y+h.*fxy;      % Compute new y
    % For printing purpose
    rsl(k,:)=[x y fxy newy];
    k = k+1;             % Update iteration
    x = x+h;             % Update x
    y = newy;            % Update y
end
```



EULER'S METHOD(Cont...)

```
% Printing the Results
Resl= array2table(rsl);          % Convert Array to Table
Resl.Properties.VariableNames(1:size(Resl,2)) = Variables
% Print Optimal Result
fprintf('Output of y(%f)=%f \n',X,rsl(end,2));
```

Program Code 9a: for Euler'methods:

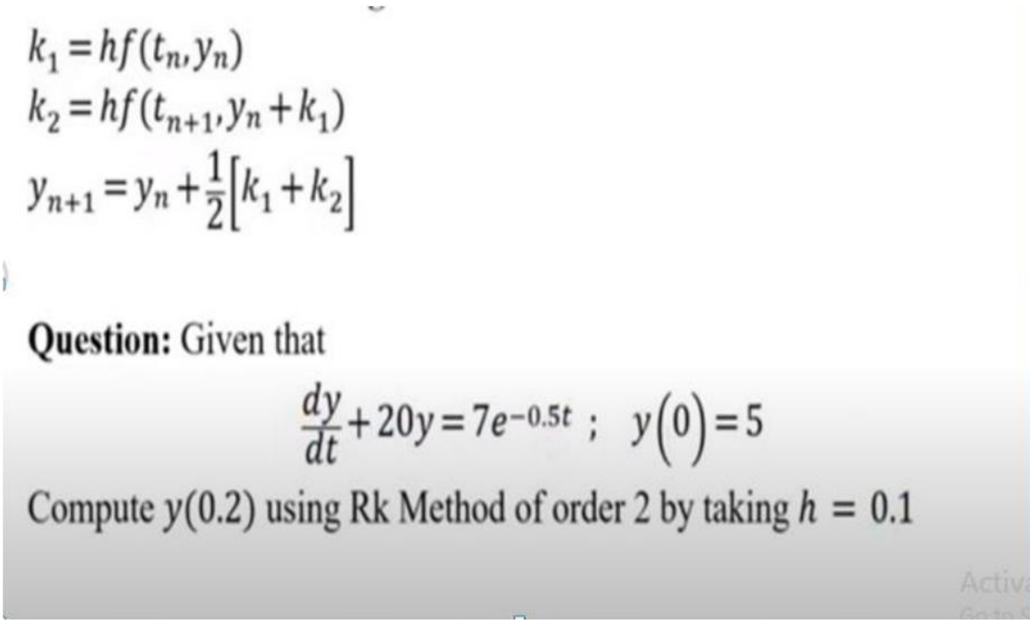
```
%Define the Objective function
f = @(x,y) (y-x)./(y+x);
%Input the parameters
x = input('Enter the initial value of x: ');
y = input('Enter the initial value of y(x): ');
h = input('Enter the step size (h): ');
X= input('Enter X at which Y  is required: ')
Variables = {'x','y', 'fxy', 'NewY'};
k = 1; % Set iteration counter
while X>=x
fprintf('Value of y at x=%f is %f \n', x,y);
fxy = f(x,y);  % Compute f(x,y)
newy = y+h.*fxy;   % Compute new y
% For printing purpose
rsl (k,:)=[x y fxy newy];
k = k+1;  % Update iteration
x = x+h;  % Update x
y = newy; % Update y
end
% Printing the Results
Resl= array2table (rsl);  % Convert Array to Table
Resl.Properties. VariableNames (1:size (Resl, 2)) =Variables
% Print Optimal Result
fprintf('Output of y(%f)=%f \n', X, rsl (end, 2));
```

OUTPUT:

```
nonlinear1
Enter the initial value of x: 0
Enter the initial value of y(x): 1
Enter the step size (h): 0.025
Enter X at which Y T is required: 0.1

X =

   0.1000

Value of y at x=0.000000 is 1.000000
Value of y at x=0.025000 is 1.025000
Value of y at x=0.050000 is 1.048810
Value of y at x=0.075000 is 1.071534
Value of y at x=0.100000 is 1.093264

Resl =
```

```
5×4 table

    x        y        fxy       NewY
  _____    ______    _______    ______

     0        1         1      1.025
  0.025    1.025    0.95238    1.0488
   0.05   1.0488    0.90899    1.0715
  0.075   1.0715    0.86917    1.0933
    0.1   1.0933    0.83239    1.1141

Output of y(0.100000)=1.093264
```

Lab Program 9b: RungaKutta Method



$$k_1 = hf(t_n, y_n)$$

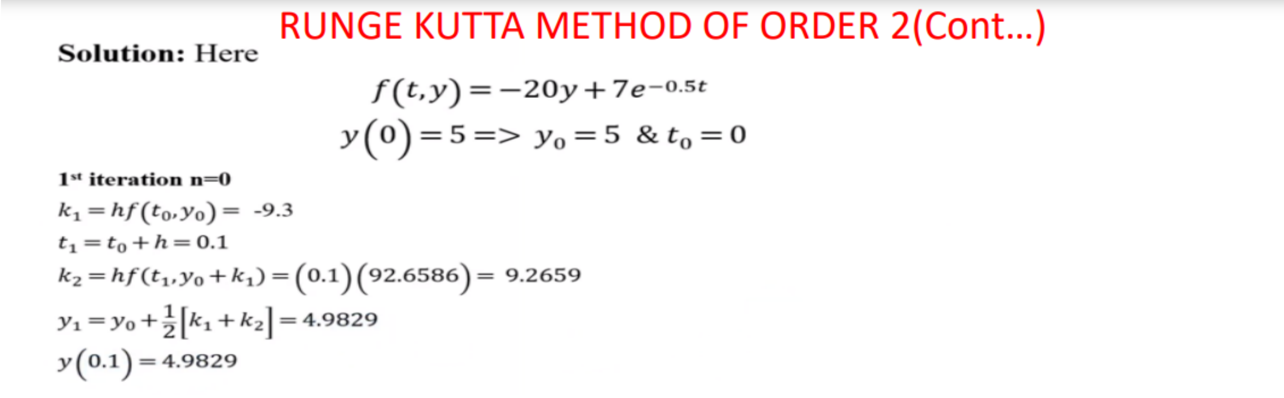
$$k_2 = hf(t_{n+1}, y_n + k_1)$$

$$y_{n+1} = y_n + \frac{1}{2}\left[k_1 + k_2\right]$$

Question: Given that

$$\frac{dy}{dt} + 20y = 7e^{-0.5t} \; ; \quad y(0) = 5$$

Compute $y(0.2)$ using Rk Method of order 2 by taking $h = 0.1$



RUNGE KUTTA METHOD OF ORDER 2(Cont…)

Solution: Here

$$f(t,y) = -20y + 7e^{-0.5t}$$

$$y(0) = 5 => y_0 = 5 \ \& \ t_0 = 0$$

1st iteration n=0

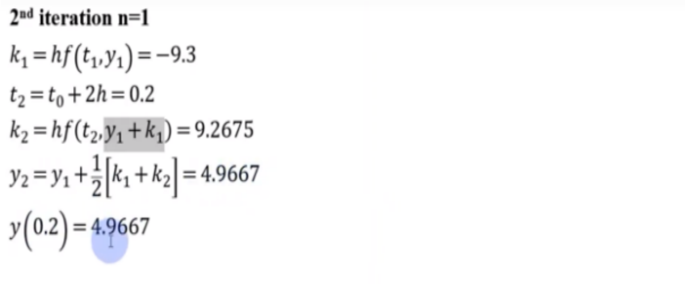
$k_1 = hf(t_0, y_0) =$ -9.3

$t_1 = t_0 + h = 0.1$

$k_2 = hf(t_1, y_0 + k_1) = (0.1)(92.6586) =$ 9.2659

$y_1 = y_0 + \frac{1}{2}[k_1 + k_2] = 4.9829$

$y(0.1) = 4.9829$



2nd iteration n=1

$k_1 = hf(t_1, y_1) = -9.3$

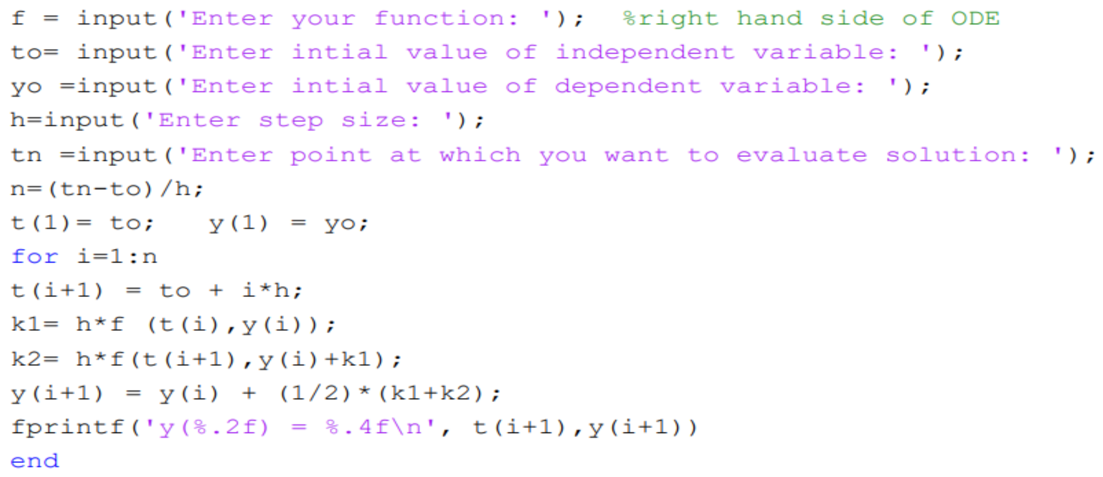
$t_2 = t_0 + 2h = 0.2$

$k_2 = hf(t_2, y_1 + k_1) = 9.2675$

$y_2 = y_1 + \frac{1}{2}[k_1 + k_2] = 4.9667$

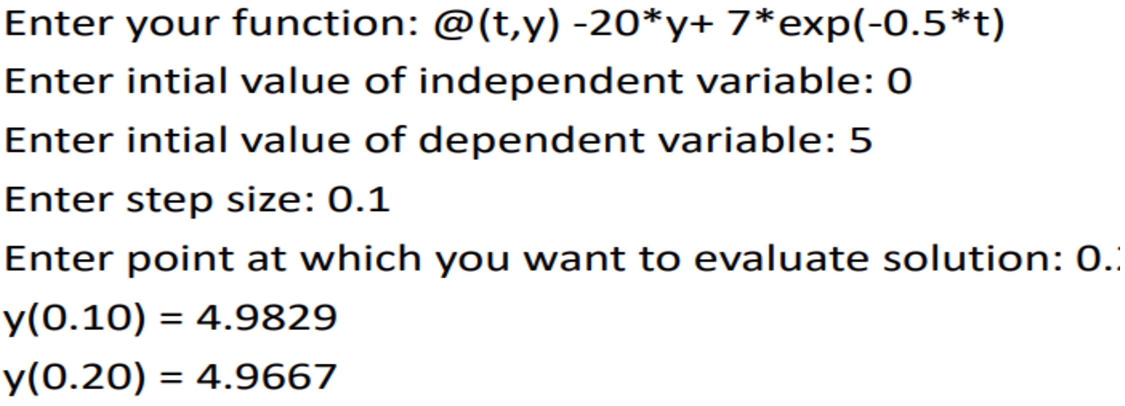
$y(0.2) = 4.9667$

Lab -Program 9B- Rungekutta method order 2.



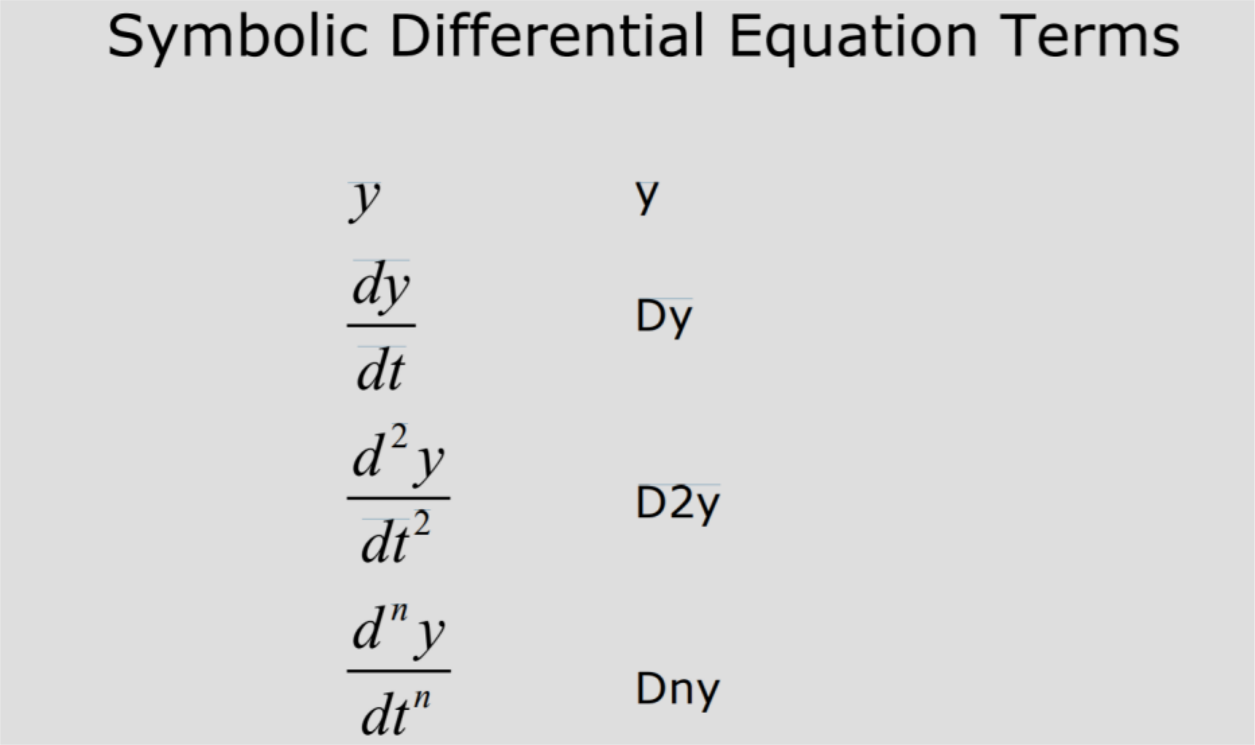
```
f = input('Enter your function: ');  %right hand side of ODE
to= input('Enter intial value of independent variable: ');
yo =input('Enter intial value of dependent variable: ');
h=input('Enter step size: ');
tn =input('Enter point at which you want to evaluate solution: ');
n=(tn-to)/h;
t(1)= to;    y(1) = yo;
for i=1:n
t(i+1) = to + i*h;
k1= h*f (t(i),y(i));
k2= h*f(t(i+1),y(i)+k1);
y(i+1) = y(i) + (1/2)*(k1+k2);
fprintf('y(%.2f) = %.4f\n', t(i+1),y(i+1))
end
```

Output:



```
Enter your function: @(t,y) -20*y+ 7*exp(-0.5*t)
Enter intial value of independent variable: 0
Enter intial value of dependent variable: 5
Enter step size: 0.1
Enter point at which you want to evaluate solution: 0.
y(0.10) = 4.9829
y(0.20) = 4.9667
```

Lab Program 9c: ODE 45



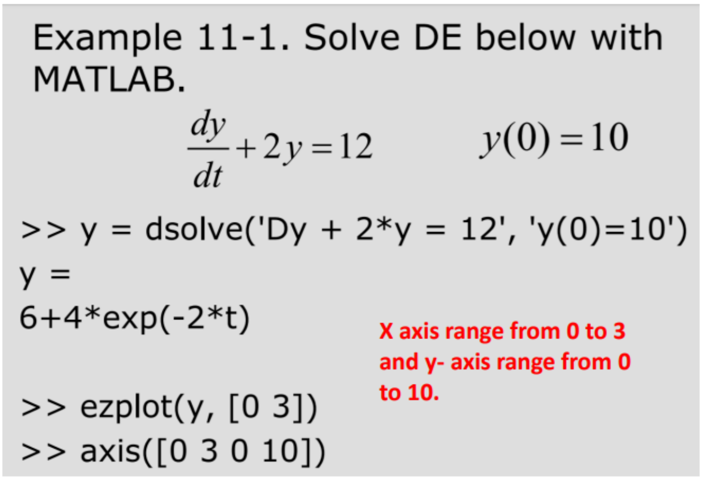
Symbolic Differential Equation Terms

y	y
$\frac{dy}{dt}$	Dy
$\frac{d^2y}{dt^2}$	D2y
$\frac{d^ny}{dt^n}$	Dny

Description

ezplot(f) - plots the expression f = f(x) over the default domain: -$2\pi < x < 2\pi$.

ezplot(f,[min,max]) - plots f = f(x) over the domain: min < x < max.



Example 11-1. Solve DE below with MATLAB.

$$\frac{dy}{dt} + 2y = 12 \qquad y(0) = 10$$

```
>> y = dsolve('Dy + 2*y = 12', 'y(0)=10')
y =
6+4*exp(-2*t)

>> ezplot(y, [0 3])
>> axis([0 3 0 10])
```

X axis range from 0 to 3 and y- axis range from 0 to 10.

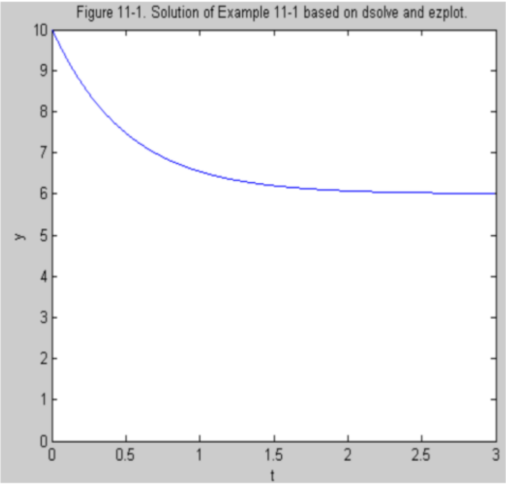
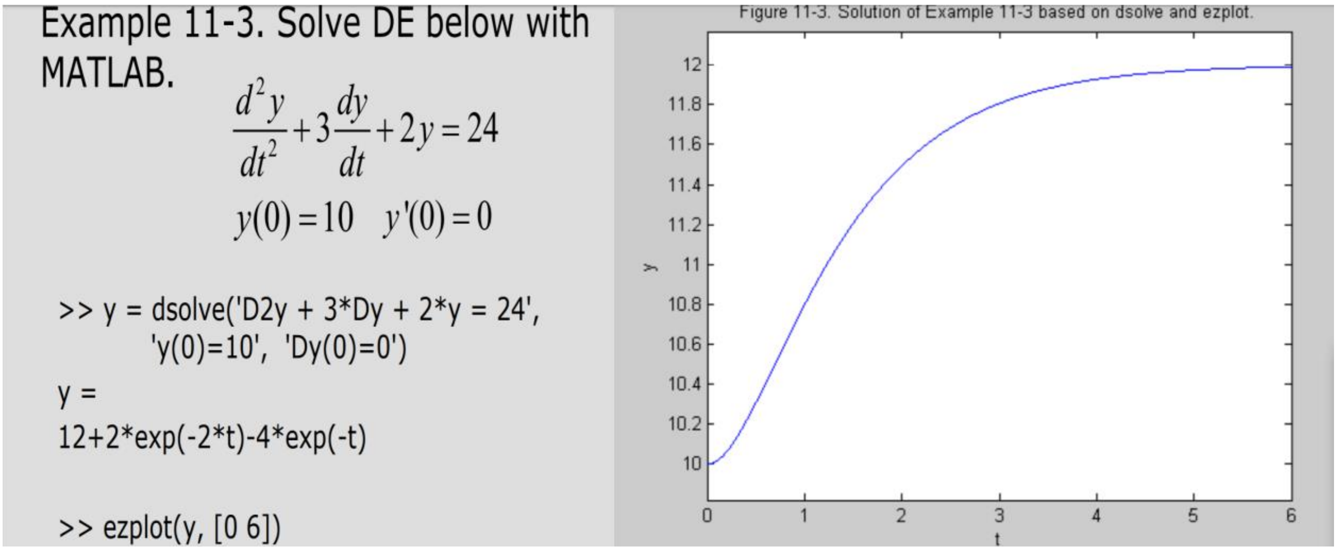


Figure 11-1. Solution of Example 11-1 based on dsolve and ezplot.



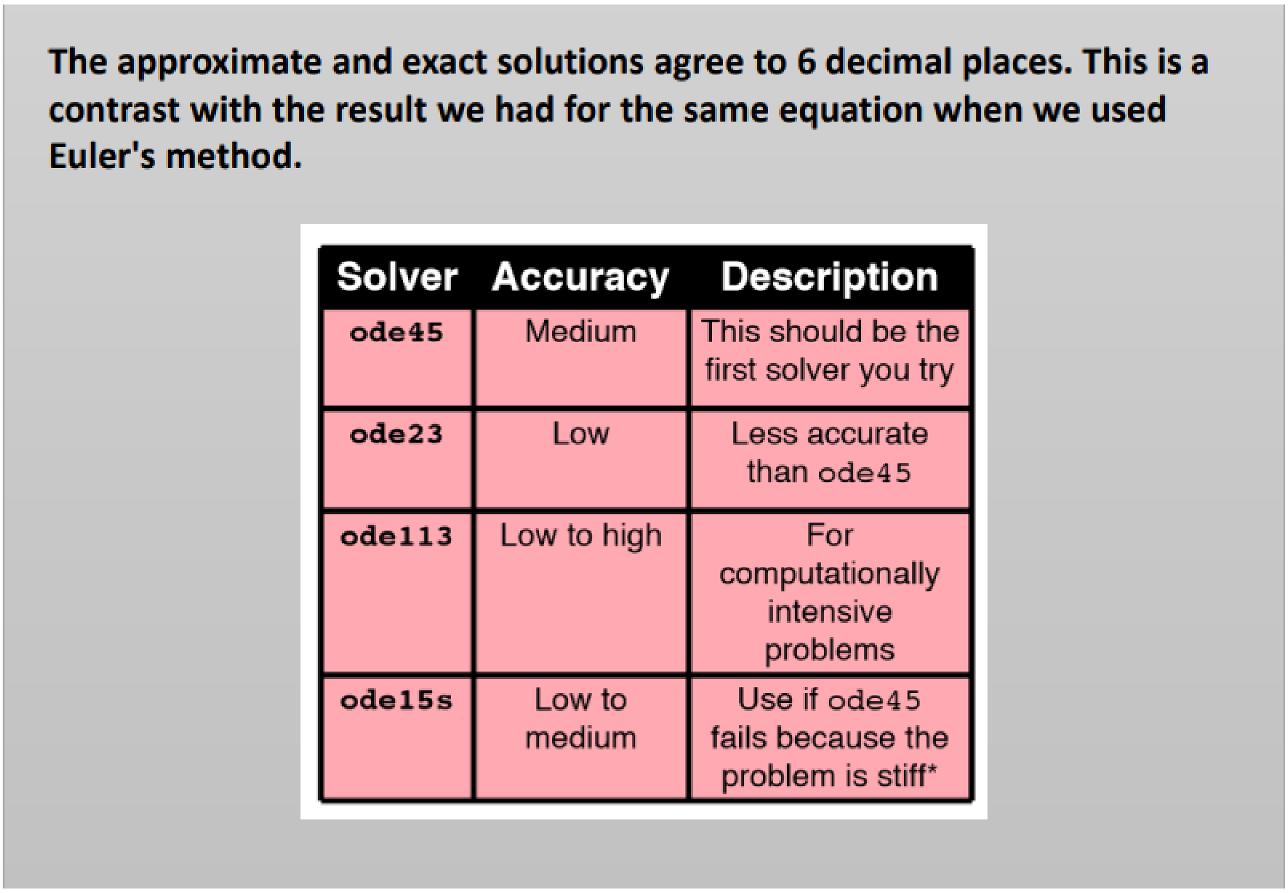
Example 11-3. Solve DE below with MATLAB.

$$\frac{d^2y}{dt^2}+3\frac{dy}{dt}+2y=24$$

$$y(0)=10 \quad y'(0)=0$$

```
>> y = dsolve('D2y + 3*Dy + 2*y = 24',
        'y(0)=10',  'Dy(0)=0')
y =
12+2*exp(-2*t)-4*exp(-t)

>> ezplot(y, [0 6])
```

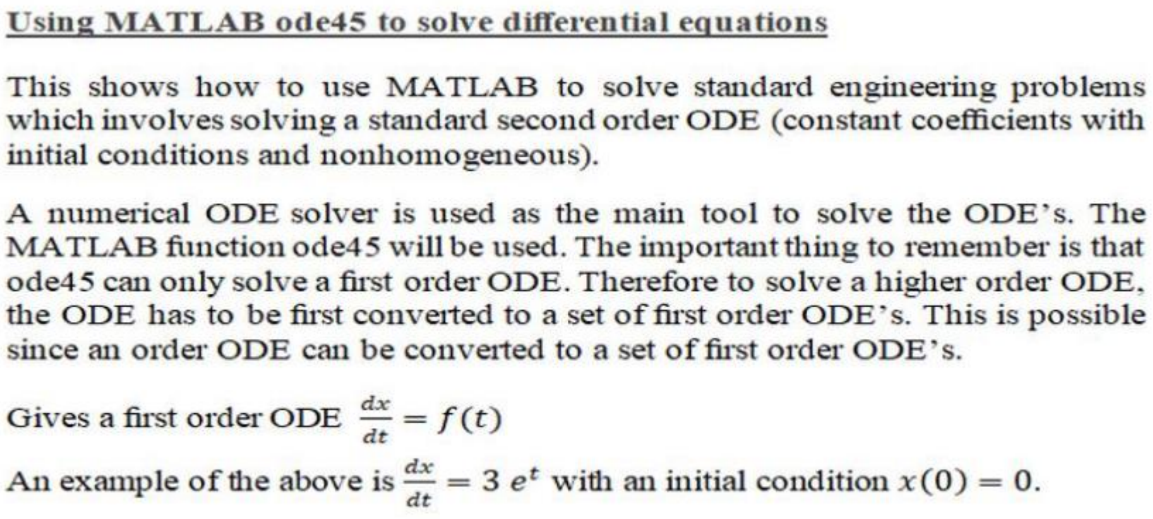
Figure 11-3. Solution of Example 11-3 based on dsolve and ezplot.



The approximate and exact solutions agree to 6 decimal places. This is a contrast with the result we had for the same equation when we used Euler's method.

Solver	Accuracy	Description
`ode45`	Medium	This should be the first solver you try
`ode23`	Low	Less accurate than `ode45`
`ode113`	Low to high	For computationally intensive problems
`ode15s`	Low to medium	Use if `ode45` fails because the problem is stiff*

2nd order ODE using ODE45



Using MATLAB ode45 to solve differential equations

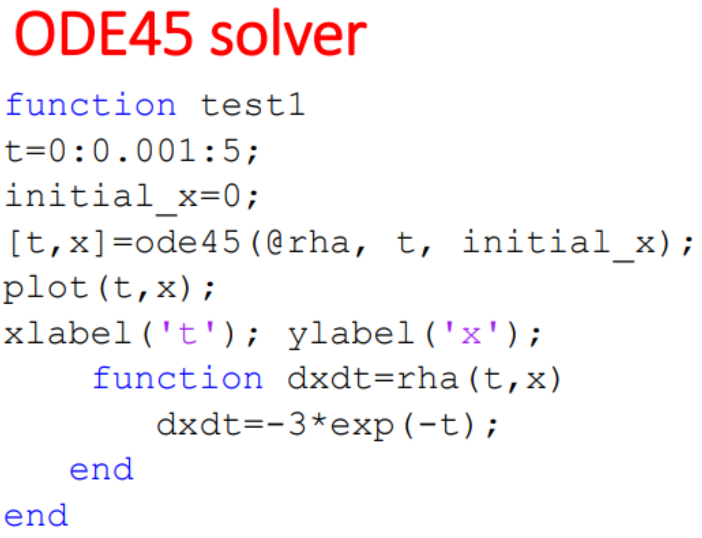
This shows how to use MATLAB to solve standard engineering problems which involves solving a standard second order ODE (constant coefficients with initial conditions and nonhomogeneous).

A numerical ODE solver is used as the main tool to solve the ODE's. The MATLAB function ode45 will be used. The important thing to remember is that ode45 can only solve a first order ODE. Therefore to solve a higher order ODE, the ODE has to be first converted to a set of first order ODE's. This is possible since an order ODE can be converted to a set of first order ODE's.

Gives a first order ODE $\frac{dx}{dt} = f(t)$

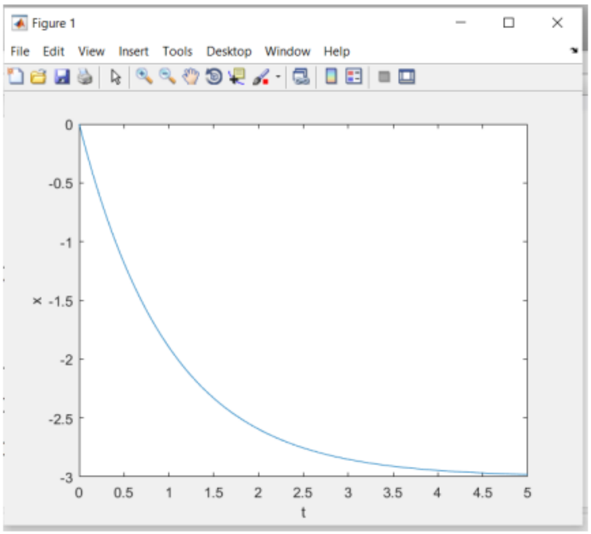
An example of the above is $\frac{dx}{dt} = 3\ e^{t}$ with an initial condition $x(0) = 0$.

Lab Program 9c: ODE 45 for dx/dt=3exp(-t)



ODE45 solver

```
function test1
t=0:0.001:5;
initial_x=0;
[t,x]=ode45(@rha, t, initial_x);
plot(t,x);
xlabel('t'); ylabel('x');
    function dxdt=rha(t,x)
        dxdt=-3*exp(-t);
   end
end
```



2nd order ODE using ODE45

To solve a second order ODE, using this as an example.

$$\frac{d^2x}{dt^2}+5\frac{dx}{dt}-4x(t)=\sin(10\,t)$$

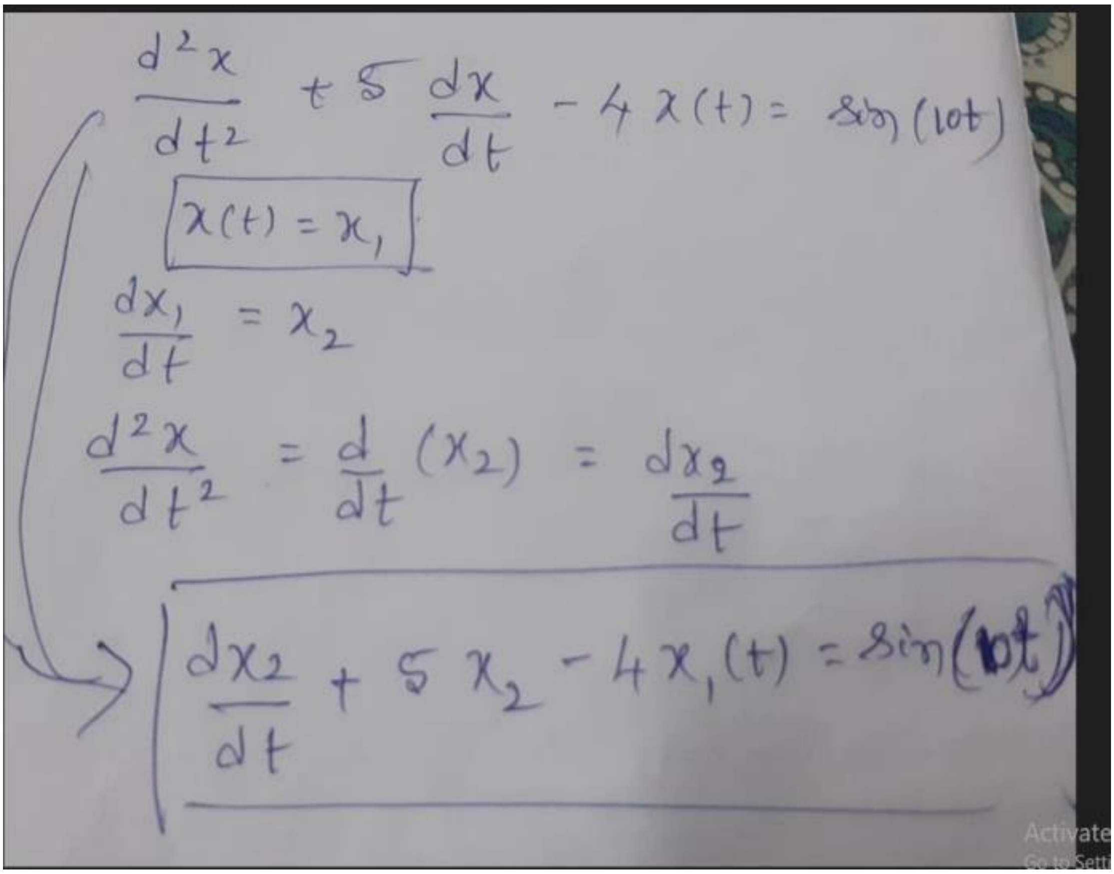
Since ode45 can only solve a first order ode, the above has to be converted to two first order ODE's as follows. Introduce 2 new state variables x_1, x_2 and carry the following derivation

$$\left.\begin{array}{l} x_1 = x \\ x_2 = x' \end{array}\right\} \xrightarrow{\text{take derivative}} \left.\begin{array}{l} x_1' = x' \\ x_2' = x'' \end{array}\right\} \xrightarrow{\text{do replacement}} \left.\begin{array}{l} x_1' = x_2 \\ x_2' = -5x' + 4x + \sin(10t) \end{array}\right\} \rightarrow \left.\begin{array}{l} x_1' = x_2 \\ x_2' = -5x_2 + 4x_1 + \sin(10t) \end{array}\right\}$$

The above gives 2 new first order ODE's. These are

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -5x_2 + 4x_1 + \sin(10t) \end{aligned}$$

Now ode45 can be used to solve the above in the same way as was done with the first example. The only difference is that now a vector is used instead of a scalar.



$$\frac{d^2x}{dt^2} + 5\frac{dx}{dt} - 4x(t) = \sin(10t)$$

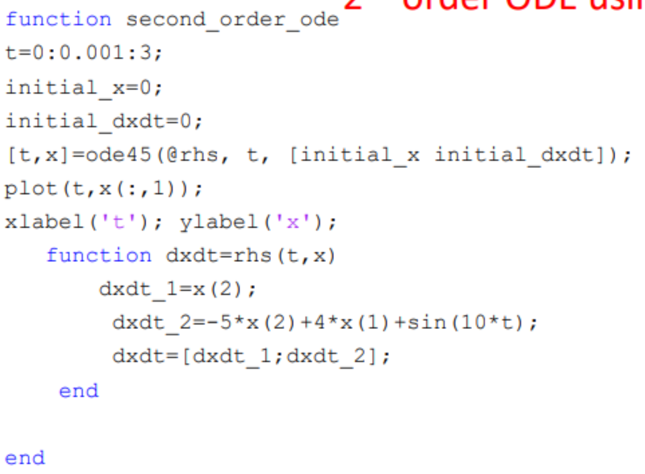
$$\boxed{x(t) = x_1}$$

$$\frac{dx_1}{dt} = x_2$$

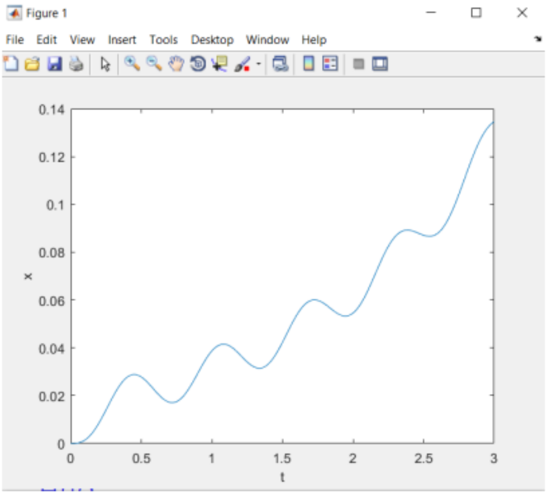
$$\frac{d^2x}{dt^2} = \frac{d}{dt}(x_2) = \frac{dx_2}{dt}$$

$$\frac{dx_2}{dt} + 5x_2 - 4x_1(t) = \sin(10t)$$

2nd order ODE using ODE45

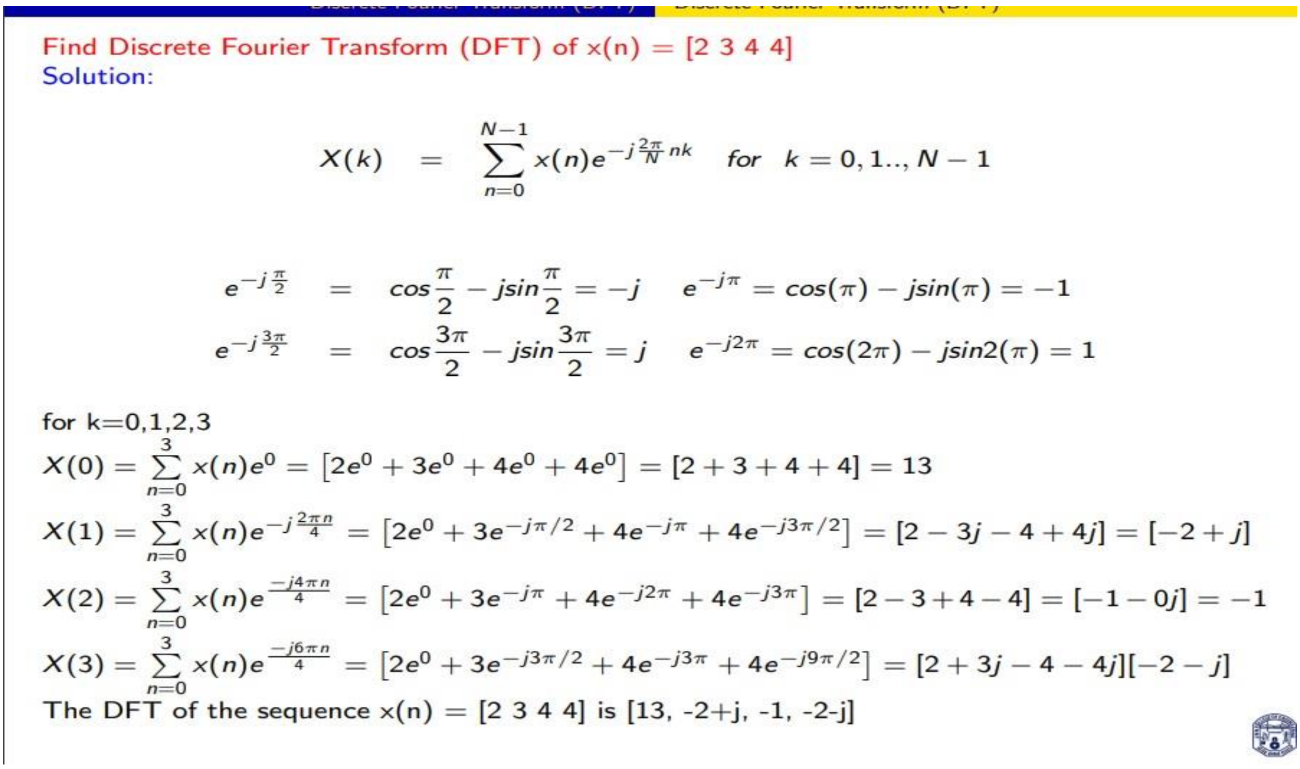


```
function second_order_ode
t=0:0.001:3;
initial_x=0;
initial_dxdt=0;
[t,x]=ode45(@rhs, t, [initial_x initial_dxdt]);
plot(t,x(:,1));
xlabel('t'); ylabel('x');
    function dxdt=rhs(t,x)
         dxdt_1=x(2);
          dxdt_2=-5*x(2)+4*x(1)+sin(10*t);
          dxdt=[dxdt_1;dxdt_2];
     end

end
```



Lab Program 10: Transforms: Discrete Fourier Transforms

Fourier Transform



Find Discrete Fourier Transform (DFT) of x(n) = [2 3 4 4]

Solution:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk} \quad for \quad k = 0, 1.., N-1$$

$$e^{-j\frac{\pi}{2}} = cos\frac{\pi}{2} - jsin\frac{\pi}{2} = -j \qquad e^{-j\pi} = cos(\pi) - jsin(\pi) = -1$$

$$e^{-j\frac{3\pi}{2}} = cos\frac{3\pi}{2} - jsin\frac{3\pi}{2} = j \qquad e^{-j2\pi} = cos(2\pi) - jsin2(\pi) = 1$$

for k=0,1,2,3

$$X(0) = \sum_{n=0}^{3} x(n)e^{0} = [2e^0 + 3e^0 + 4e^0 + 4e^0] = [2 + 3 + 4 + 4] = 13$$

$$X(1) = \sum_{n=0}^{3} x(n)e^{-j\frac{2\pi n}{4}} = [2e^0 + 3e^{-j\pi/2} + 4e^{-j\pi} + 4e^{-j3\pi/2}] = [2 - 3j - 4 + 4j] = [-2 + j]$$

$$X(2) = \sum_{n=0}^{3} x(n)e^{\frac{-j4\pi n}{4}} = [2e^0 + 3e^{-j\pi} + 4e^{-j2\pi} + 4e^{-j3\pi}] = [2 - 3 + 4 - 4] = [-1 - 0j] = -1$$

$$X(3) = \sum_{n=0}^{3} x(n)e^{\frac{-j6\pi n}{4}} = [2e^0 + 3e^{-j3\pi/2} + 4e^{-j3\pi} + 4e^{-j9\pi/2}] = [2 + 3j - 4 - 4j][-2 - j]$$

The DFT of the sequence x(n) = [2 3 4 4] is [13, -2+j, -1, -2-j]

```
xn=[2 3 4 4]
N=length(xn)
 n=0:N-1
 k=0:N-1;
WN=exp(-1j*2*pi/N);
nk=n'*k;
WNnk=WN.^nk;
Xk=xn*WNnk
```

```
Xk =
```

```
  Columns 1 through 2
  13.0000 + 0.0000i  -2.0000 + 1.0000i
  Columns 3 through 4
  -1.0000 - 0.0000i  -2.0000 - 1.0000i
```

Lab Program 11,12,13

Application of MATLAB to analyse problems in basic engineering mechanics, mechanical vibrations, control system, statistics and dynamics of different circuits. MATLAB Simulink: Introduction to MATLAB Simulink, Simulink libraries, development of basic models in Simscape Power.

Introduction to Simulink

- ❖Simulink is a software package for modeling, simulating, and analyzing dynamical systems.
- ❖It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two.
- ❖For modeling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations.
- ❖With this interface, we can draw the models just as we would with pencil and paper (or depict them as it is done in most textbooks).
- ❖Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors.
- ❖ We can also customize and create our own blocks.

2.1 Starting Simulink

To start a Simulink session, we'd need to bring up Matlab program first (Nguyen, 1995). From Matlab command window, enter:

```
>> simulink
```

Alternately, we may click on the Simulink icon located on the toolbar as shown:

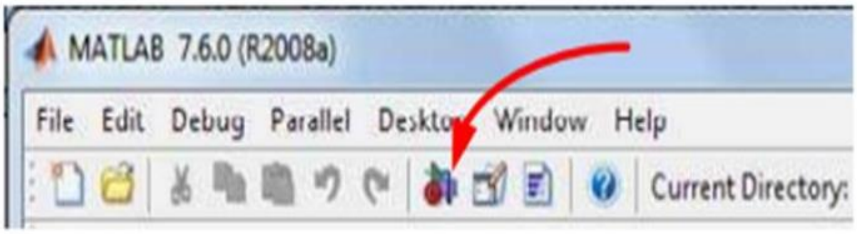


Fig. 1. Simulink icon in Matlab window

Simulink's library browser window like one shown below will pop up presenting the block set for model construction.

Simulink's library browser window like one shown below will pop up presenting the block set for model construction.

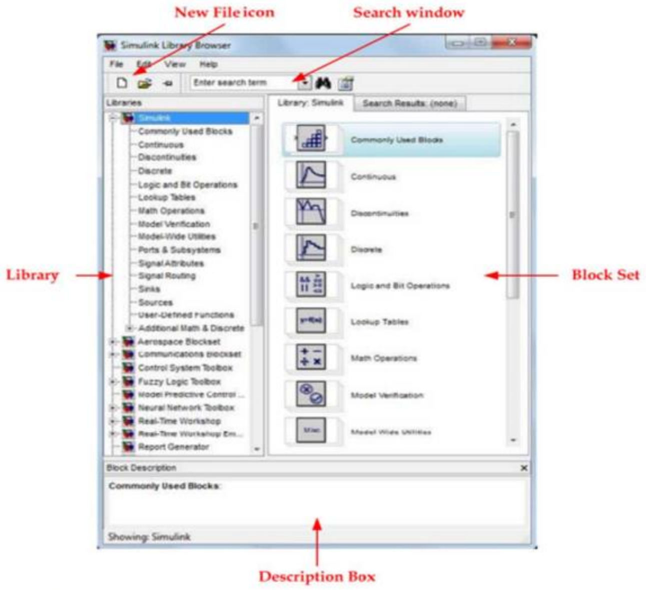


Fig. 2. Simulink’s library browser

To see the content of the blockset, click on the "+" sign at the beginning of each toolbox.
To start a model click on the NEW FILE ICON as shown in the screenshot above.
Alternately, we may use keystrokes CTRL+N.
A new window will appear on the screen. We will be constructing our model in this window. Also in this window the constructed model is simulated. A screenshot of a typical working (model) window looks like one shown below:

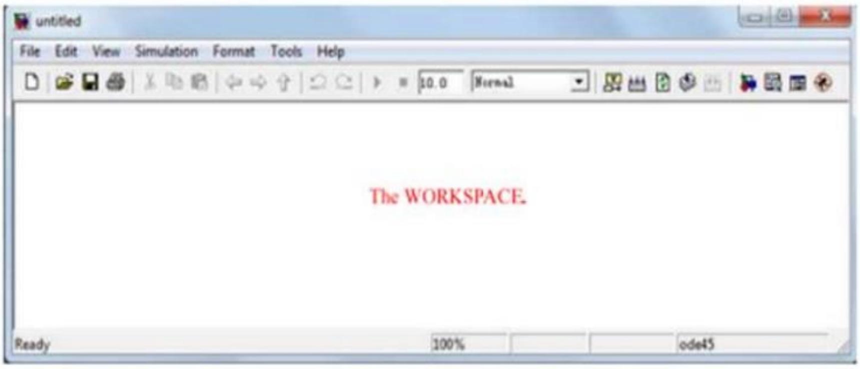


Fig. 3. Simulink workspace

Step 1. Creating Blocks.
From BLOCK SET CATEGORIES section of the SIMULINK LIBRARY BROWSER window, click on the "+" sign next to the Simulink group to expand the tree and select (click on) Sources.

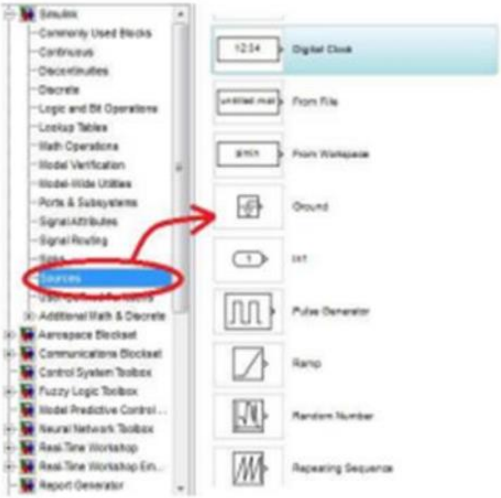
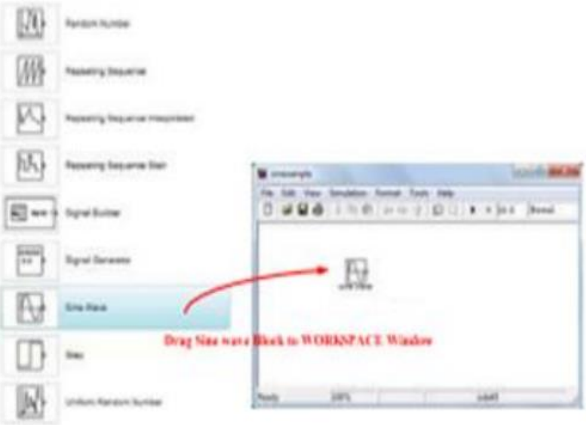


Fig. 4. Sources Block sets

A set of blocks will appear in the BLOCKSET group. Click on the Sine Wave block and drag it to the workspace window (also known as model window).



Drag Sine wave Block to WORKSPACE Window

Fig. 5. Adding Blocks to Workspace

Now we have established a source of our model.

To save a model, click on the floppy diskette icon💾 or from FILE menu, select Save or CTRL+S. All Simulink model files will have an extension ".mdl". Simulink recognizes the file with .mdl extension as a simulation model (similar to how MATLAB recognizes files with the extension .m as an MFile).

Continue to build the model by adding more components (or blocks) to the model window. We will add the Scope block from Sinks library, an Integrator block from Continuous library, and a Mux block from Signal Routing library.

NOTE: If we wish to locate a block knowing its name, we may enter the name in the SEARCH WINDOW (at Find prompt) and Simulink will bring up the specified block.

To move the blocks around, click on them and drag to a desired location.

Once all the blocks are dragged over to the work space, we may remove (delete) a block, by clicking on it once to turn on the "select mode" (with four corner boxes) and use the DEL key or keys combination CTRL-X.

Step 2. Making connections.

To establish connections between the blocks, move the cursor to the output port represented by ">" sign on the block. Once placed at a port, the cursor will turn into a cross "+" enabling us to make connection between blocks.

To make a connection: left-click while holding down the control key (on the keyboard) and drag from source port to a destination port.

The connected model is shown below.

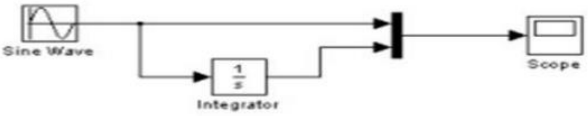


Fig. 6. Block diagram for Sine simulation

A sine signal is generated by the Sine Wave block (a source) and displayed on the scope (fig. 7). The integrated sine signal is sent towards the scope, to display it along with the original signal from the source via the Mux, whose function is to multiplex signals in form of scalar, vector, or matrix into a bus.

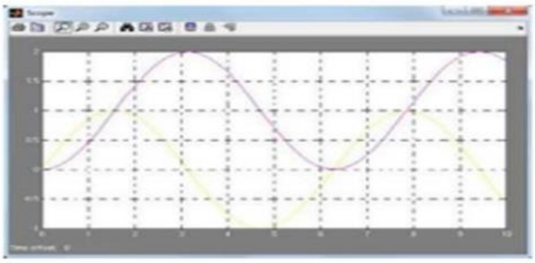


Fig. 7. Scope appearance

Step 3. Running simulation.
Now the simulation of the simple system above can be run by clicking on the play button (▶, alternatively, we may use key sequence CTRL+T, or choose Start submenu under Simulation menu).

Double click on the Scope block to display of the scope.

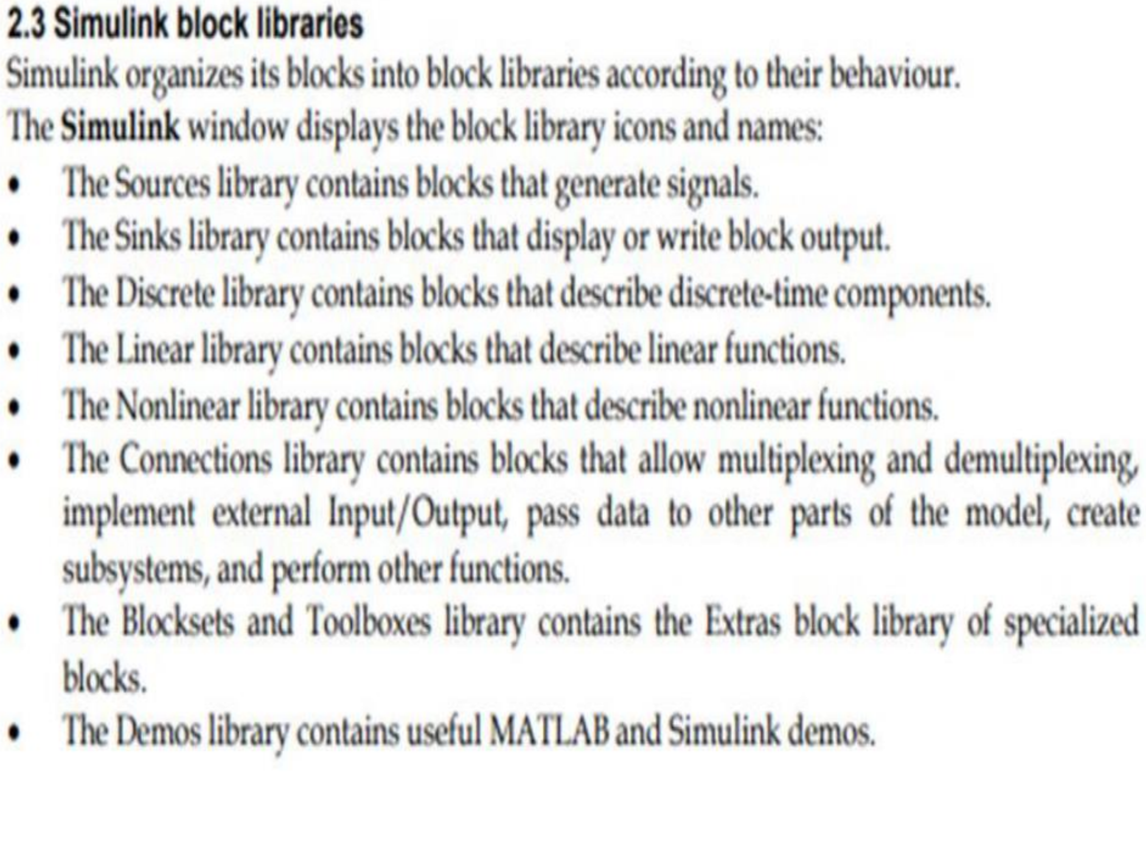
To view/edit the parameters, simply double click on the block of interest.

2.2 Handling of blocks and lines

The table below describes the actions and the corresponding keystrokes or mouse operations (Windows versions) (Nguyen, 1995).

Actions	Keystrokes or Mouse Actions
Copying a block from a library	Drag the block to the model window with the left mouse button on the OR use choose between select the COPY and PASTE from EDIT menu.
Duplicating blocks in a model	Hold down the CTRL key and select the block. Drag the block to a new location with the left mouse button.
Display block's parameters	Click doubly on the bloc.
Flip a block	CTRL-F
Rotate a block	CTRL-R
Changing blocks' names	Click on block's label and position the cursor to desired place.
Disconnecting a block	Hold down the SHIFT key and drag the block to a new location.
Drawing a diagonal line	Hold down the SHIFT key while dragging the mouse with the left button.
Dividing a line	Move the cursor to the line to where we want to create the vertex and use the left button on the mouse to drag the line while holding down the SHIFT key.

Table 1. The actions and the corresponding keystrokes or mouse operations.



2.3 Simulink block libraries

Simulink organizes its blocks into block libraries according to their behaviour.

The **Simulink** window displays the block library icons and names:

- The Sources library contains blocks that generate signals.
- The Sinks library contains blocks that display or write block output.
- The Discrete library contains blocks that describe discrete-time components.
- The Linear library contains blocks that describe linear functions.
- The Nonlinear library contains blocks that describe nonlinear functions.
- The Connections library contains blocks that allow multiplexing and demultiplexing, implement external Input/Output, pass data to other parts of the model, create subsystems, and perform other functions.
- The Blocksets and Toolboxes library contains the Extras block library of specialized blocks.
- The Demos library contains useful MATLAB and Simulink demos.

3. Viewing and analyzing simulation results

Output trajectories from Simulink can be plotted using one of three methods (The MathWorks, 1999):

- Feeding a signal into either a Scope or an XY Graph block
- Writing output to return variables and using MATLAB plotting commands
- Writing output to the workspace using To Workspace blocks and plotting the results using MATLAB plotting commands

3.1 Using the scope block

We can use display output trajectories on a Scope block during a simulation.
This simple model shows an example of the use of the Scope block:

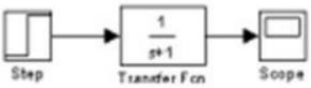


Fig. 8. Block diagram for Scope displaying

The display on the Scope shows the output trajectory. The Scope block enables to zoom in on an area of interest or save the data to the workspace.
The XY Graph block enables to plot one signal against another.

3.2 Using return variables

By returning time and output histories, we can use MATLAB plotting commands to display and annotate the output trajectories.

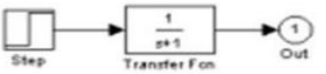


Fig. 9. Block diagram for output displaying

The block labelled **Out** is an Outport block from the Connections library. The output trajectory, yout, is returned by the integration solver. For more information, see Chapter 4. This simulation can also be run from the Simulation menu by specifying variables for the time, output, and states on the Workspace I/O page of the Simulation Parameters dialog box. then these results can be plot using:
plot (tout,yout)

3.3 Using the To Workspace block

The **To Workspace** block can be used to return output trajectories to the MATLAB workspace. The model below illustrates this use:

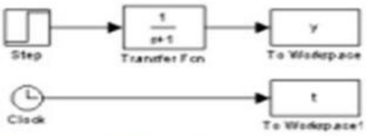


Fig. 10. Block diagram for Workspace displaying

The variables y and t appear in the workspace when the simulation is complete. The time vector is stored by feeding a Clock block into a To Workspace block. The time vector can also be acquired by entering a variable name for the time on the Workspace I/O page of the Simulation Parameters dialog box for menu-driven simulations, or by returning it using the sim command (see Chapter 4 for more information).

The **To Workspace** block can accept a vector input, with each input element's trajectory stored as a column vector in the resulting workspace variable.

4. Modeling mechanical systems with Simulink

Simulink's primary design goal is to enable the modeling, analysis, and implementation of dynamics systems so then mechanical systems. The mechanical systems consist of bodies, joints, and force elements like springs. Modeling a mechanical system need the equations of motion or the mechanical structure. Thus in general mechanical systems can be simulated by two ways:

- Using graphical representation of the mathematical model.
- Drawing directly the mechanical system using SimMechanics.

4.1 Modeling using graphical representation:

The equations of motion of mechanical systems have undergone historical development associated with such distinguished mathematicians as Newton, D'Alembert, Euler, Lagrange, Gauss, and Hamilton, among others (Wood & Kennedy, 2003). While all made significant contributions to the formulation's development of the underlying equations of motion, our interest here is on the computational aspects of mechanical simulation in an existing dynamic simulation package. Simulink is designed to model systems governed by these mathematical equations. The Simulink model is a graphical representation of mathematical operations and algorithm elements. Simulink solves the differential equation by evaluating the individual blocks according to the sorted order to compute derivatives for the states. The solver uses numeric integration to compute the evolution of states through time. Application of this method is illustrated in the first example of the section 5.

4.2 Modeling using SimMechanics

SimMechanics™ software is a block diagram modeling environment for the engineering design and simulation of rigid body machines and their motions, using the standard Newtonian dynamics of forces and torques. Instead of representing a mathematical model of the system, we develop a representation that describes the key components of the mechanical system. The base units in SimMechanics are physical elements instead of algorithm elements. To build a SimMechanics model, we must break down the mechanical system into the building blocks that describe it (Popinchalk, 2009).

After building the mechanical representation using SimMechanics, to study the system's response to and stability against external changes, we can apply small perturbations in the motion or the forces/torques to a known trajectory and force/torque set. SimMechanics software and Simulink® provide analysis modes and functions for analyzing the results of perturbing mechanical motion. To use these modes, we must first build a kinematic model of the system, one that specifies completely the positions, velocities, and accelerations of the system's bodies. We create a kinematic model by interconnecting blocks representing the bodies and joints of the system and then connecting actuators to the joints to specify the motions of the bodies. Application of this method is illustrated in the second example of the section 5.

5. Examples of modeling and control of mechanical systems

5.1 Dynamics modeling for satellite antenna dish stabilized platform

The stabilized platform is the object which can isolate motion of the vehicle, and can measure the change of platform's motion and position incessantly, exactly hold the motorial gesture benchmark, so that it can make the equipment which is fixed on the platform aim at and track object fastly and exactly. In the stabilized platform systems, the basic requirements are to maintain stable operation even when there are changes in the system dynamics and to have very good disturbance rejection capability.

The objective of this example is to develop the dynamics model simulation for satellite antenna dish stabilized platform. The dynamic model of the platform is a three degree of freedom system. It is composed of, the four bodies which are: case, outer gimbal, inner gimbal and platform as shown in fig. 11. Simulink is used to simulate the obtained dynamic model of the stabilized platform. The testing results can be used to analyze the dynamic structure of the considered system. In addition, these results can be applied to the stabilization controller design study (Leghmizi et al., 2011).

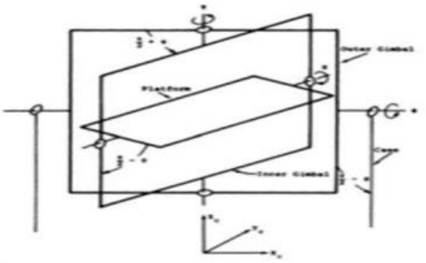


Fig. 11. The system structure

The mathematical modeling was established using Euler theory. The Euler's moment equations are

$$\overline{M} = i\overline{H} \tag{1}$$

The net torque $\overline{M}$ consists of driving torque applied by the adjacent outer member and reaction torque applied by the adjacent inner member.

$$i\overline{H} = \frac{dH}{dt} = m\overline{H} + \overline{\omega_m} \times \overline{H} \tag{2}$$

$i\overline{H}$: Inertial derivative of the vector $\overline{H}$;

$m\overline{H}$: Derivative of H calculated in a rotating frame of reference;

$\overline{\omega_m}$: Absolute rotational rate of the moving reference frame;

$\overline{H}$: Inertial angular momentum;

$\overline{M}$: External torque applied to the body.

By applying equation (2) on the different parts of the platform system , the system may be expressed as a set of second-order differential equations in the state variables. Solving this system of equations we obtain:

$$\ddot{\phi} = \frac{C_i B_o - C_o B_i}{A_i B_o - A_o B_i} \tag{3}$$

$$\ddot{\psi} = \frac{C_o A_i - C_i A_o}{A_i B_o - A_o B_i} \tag{4}$$

$$\ddot{\theta} = \frac{C_p}{B_p} - \frac{A_p}{B_p} * \frac{C_i B_o - C_o B_i}{A_i B_o - A_o B_i} \tag{5}$$

Where

$A_p = \sin\psi$

$B_p = 1$

$C_p = \frac{M^{*}_{py} - MPY}{I_{py}}$

$A_i = \cos\psi \cos\theta \sin\theta \left[\frac{I_{px} - I_{pz}}{I_{iz}} \right]$

$B_i = \left[1 + \sin^2\theta \frac{I_{px}}{I_{iz}} + \cos^2\theta \frac{I_{pz}}{I_{iz}} \right]$

$C_i = \frac{M^{*}_{miz} - MIZ}{I_{iz}}$

$A_o = 1 + \cos^2\psi \left[\frac{I_{ix} + I_{px}\cos^2\theta + I_{pz}\sin^2\theta}{I_{ox}} \right] + \sin^2\psi \left[\frac{I_{iy}}{I_{ox}} \right]$

$B_o = \cos\theta \sin\theta \cos\psi \left[\frac{I_{px} - I_{pz}}{I_{ox}} \right]$

$C_o = \frac{M^{*}_{mox} - MCX}{I_{ox}}$

Detailed equations computation is presented in the paper (Leghmizi, 2010, 2011).

Here, it suffices to note that designing a simulation for the system based on these complete nonlinear dynamics is extremely difficult. It is thus necessary to reduce the complexity of the problem by considering the linearized dynamics (Lee et al., 1996). This can be done by noting that the gimbal angles variations are effectively negligible and that the ship velocities effect is insignificant. Applying the above assumptions to the nonlinear dynamics, the following equations are obtained.

$$\ddot{\phi} = \frac{D_{ox}}{I_{px} + I_{ix} + I_{ox}}\dot{\phi} - \frac{1}{I_{px} + I_{ix} + I_{ox}}F_{ox}(\operatorname{sgn}\dot{\phi}) - \frac{I_{pz} - I_{py} + I_{pz}}{I_{px} + I_{ix} + I_{ox}}\dot{\psi}\dot{\theta} - T_{ox} \tag{6}$$

$$\ddot{\psi} = \frac{D_{iz}}{I_{pz} + I_{iz}}\dot{\psi} - \frac{1}{I_{pz} + I_{iz}}F_{iz}(\operatorname{sgn}\dot{\psi}) - \frac{I_{py} - I_{px} + I_{pz}}{I_{pz} + I_{iz}}\dot{\theta}\dot{\phi} - T_{oxx} \tag{7}$$

$$\ddot{\theta} = \frac{D_{iy}}{I_{py}}\dot{\theta} - \frac{1}{I_{py}}F_{iy}(\operatorname{sgn}\dot{\theta}) - \frac{I_{px} - I_{pz} + I_{py}}{I_{py}}\dot{\psi}\dot{\phi} - T_{a} \tag{8}$$

5.1.2 Modeling the equations of motion with Simulink

The model in fig. 12 is the graphical representation of equations (6), (7) and (8). It's obtained by using the **Simulink toolbox**.

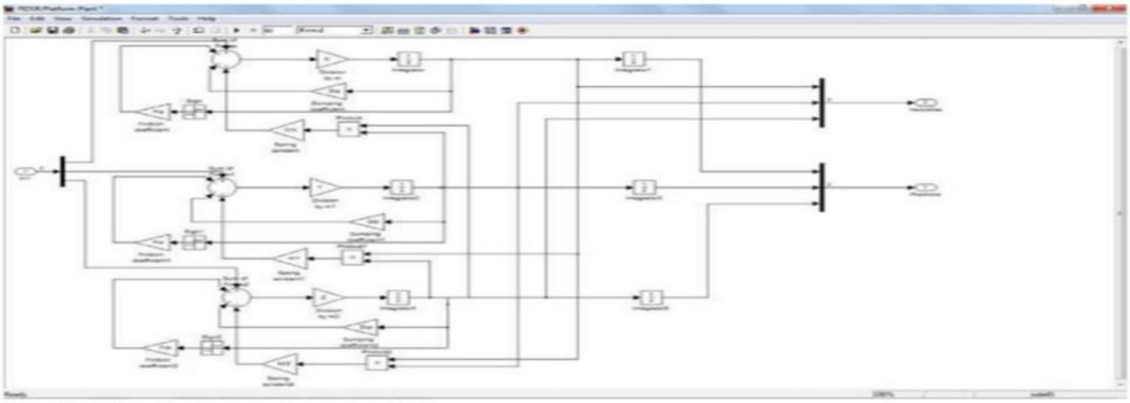


Fig. 12. The platform plant simulation

In order to enhance our understanding of the system, we performed a simulation in closed-loop mode. After that, a PID controller was applied to the closed-loop model. The PID controlled parameters was calculated using the Ziegler–Nichols method (Moradi, 2003). The obtained Simulink model is presented in the fig. 13.

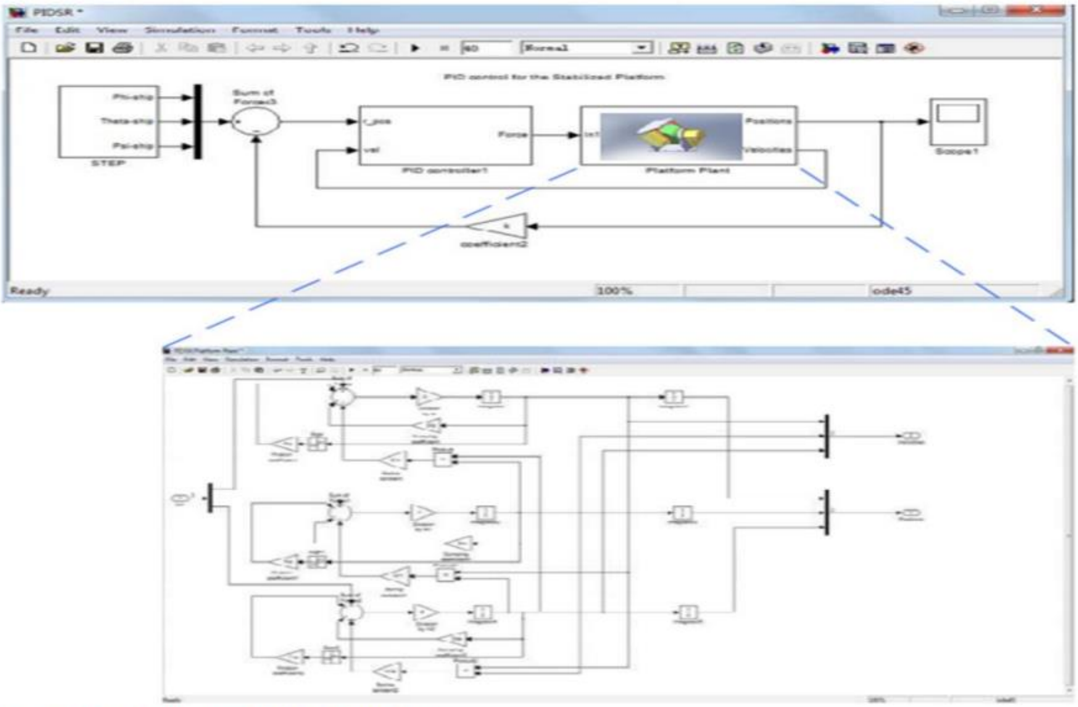


Fig. 13. Simulation model by Simulink

This simulation was particularly useful to recognize the contribution of each modelled effect to the dynamics of the system. Also, knowing the natural behavior of the system could be useful for establishing adapted control laws. Simulation results will be presented to illustrate the gimbals behaviour to different entries. They are presented in fig. 14, which contains the impulsion and step responses of the closed-loop system using the PID controller. Each graph superimposes the angular position on the X axes (blue), the Y axes (green) and the Z axes (red).

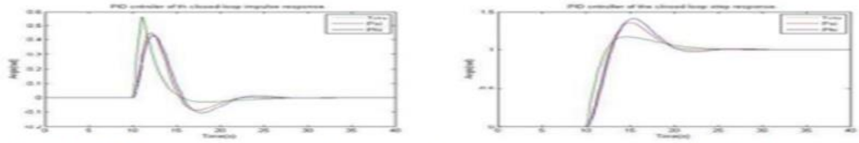


Fig. 14. The closed-loop system impulsion and step responses using the PID controller

5.2 Modeling a Stewart platform

The Stewart platform is a classic design for position and motion control, originally proposed in 1965 as a flight simulator, and still commonly used for that purpose (Stewart, 1965). Since then, a wide range of applications have benefited from the Stewart platform. A few of the industries using this design include aerospace, automotive, nautical, and machine tool technology. Among other tasks, the platform has been used, to simulate flight, model a lunar rover, build bridges, aid in vehicle maintenance, design crane and hoist mechanisms, and position satellite communication dishes and telescopes (Matlab Help).

The Stewart platform has an exceptional range of motion and can be accurately and easily positioned and oriented. The platform provides a large amount of rigidity, or stiffness, for a given structural mass, and thus provides significant positional certainty. The platform model is moderately complex, with a large number of mechanical constraints that require a robust simulation. Most Stewart platform variants have six linearly actuated legs with varying combinations of leg-platform connections. The full assembly is a parallel mechanism consisting of a rigid body top or mobile plate connected to an immobile base plate and defined by at least three stationary points on the grounded base connected to the legs.

The Stewart platform used here is connected to the base plate at six points by universal joints as shown in fig. 15. Each leg has two parts, an upper and a lower, connected by a cylindrical joint. Each upper leg is connected to the top plate by another universal joint. Thus the platform has 6*2 + 1 = 13 mobile parts and 6*3 = 18 joints connecting the parts.

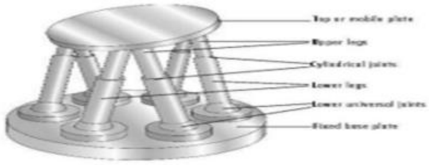


Fig. 15. Stewart platform

5.2.1 Modeling the physical Plant with SimMechanics

The Plant subsystem models the Stewart platform's moving parts, the legs and top plate. The model in the fig. 16 is obtained by using the SimMechanics toolbox. From the Matlab demos we can open this subsystem.

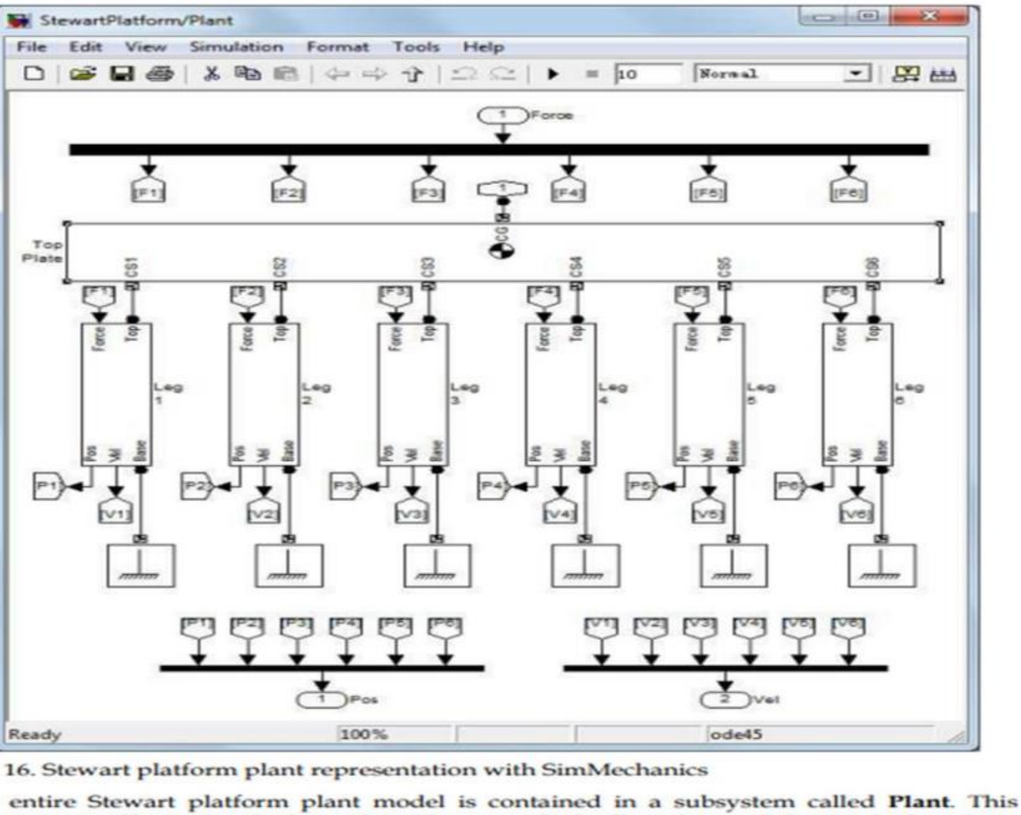


Fig. 16. Stewart platform plant representation with SimMechanics

The entire Stewart platform plant model is contained in a subsystem called **Plant**. This subsystem itself contains the base plate (the ground), the Top plate and the six platform legs. Each of the legs is a subsystem containing the individual Body and Joint blocks that make up the whole leg (see fig. 17).

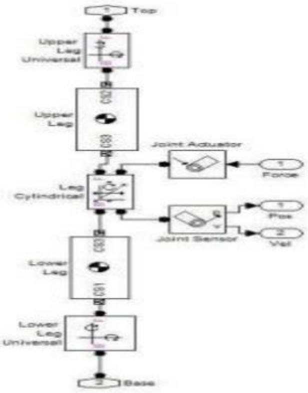


Fig. 17. Leg Subsystem content

To visualise the content of this subsystem, select one of the leg subsystems and right-click select **Look Under Mask**.

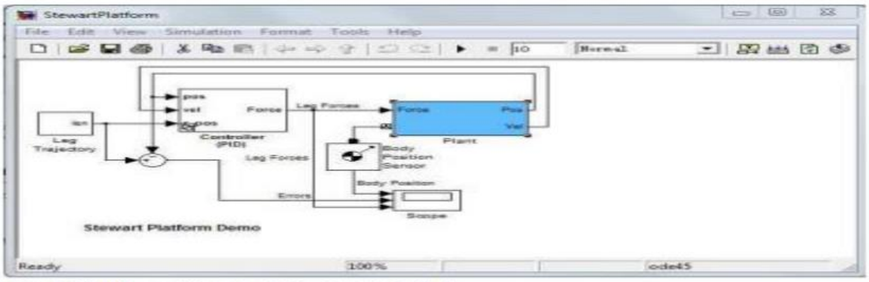


Fig. 18. Stewart Platform Control Design Model

The blue subsystem contains the Stewart platform plant presented in fig. 18. The simulation model in fig. 18 is the control of the Stewart platform's motion with the linear proportional-integral-derivative (PID) feedback system presented in fig. 19.

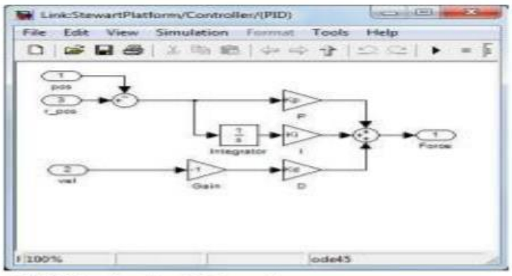

Link:StewartPlatform/Controller/(PID)
File Edit View Simulation Format Tools Help
1
pos
3
r_pos
Kp
P
1/s
Integrator
Ki
I
1
Force
2
vel
-1
Gain
Kd
D
100%
ode45

Fig. 19. Stewart Platform PID Controller Subsystem

The control transfer function of the PID linear feedback control system has the form $K_i/s + K_d.s + K_p$. The control gains Ki, Kp, and Kd in their respective blocks refer to the variables K_i, K_p, K_d defined in the workspace. Check their initialized values:
$K_i = 10000$
$K_p = 2000000$
$K_d = 45000$
To simulate the Stewart platform with the PID controller:

- Open the Scope and start the simulation.
- Observe the controlled Stewart platform motion. The Scope results given in fig. 20 show how the platform initially does not follow the reference trajectory, which starts in a position different from the platform's home configuration. The motion errors and forces on the legs are significant. Observe also that the leg forces saturate during the initial transient.

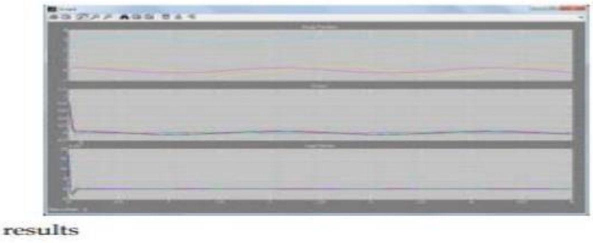


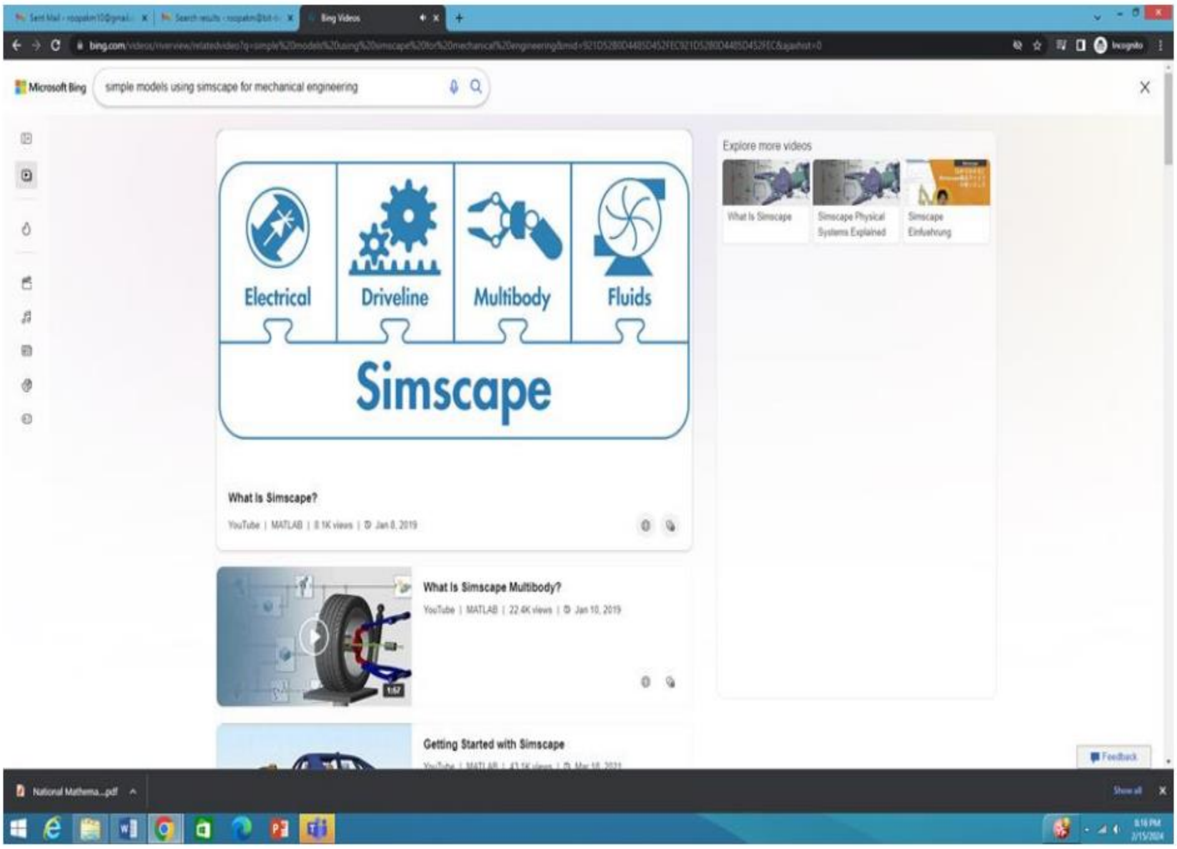
Fig. 20. Simulation results

Basics of Simscape

- ❖Simscape is a physical modeling language and environment that enables you to rapidly create models of physical systems within the Simulink environment1.
- ❖ With Simscape, you build physical component models based on physical connections that directly integrate with block diagrams and other modeling paradigms[1].
- ❖The Simscape language is a dedicated textual language for modeling physical systems and has additional constructs specific to physical modeling[2].
- ❖ The Simscape file is a dedicated file type in the MATLAB environment with the extension.ssc

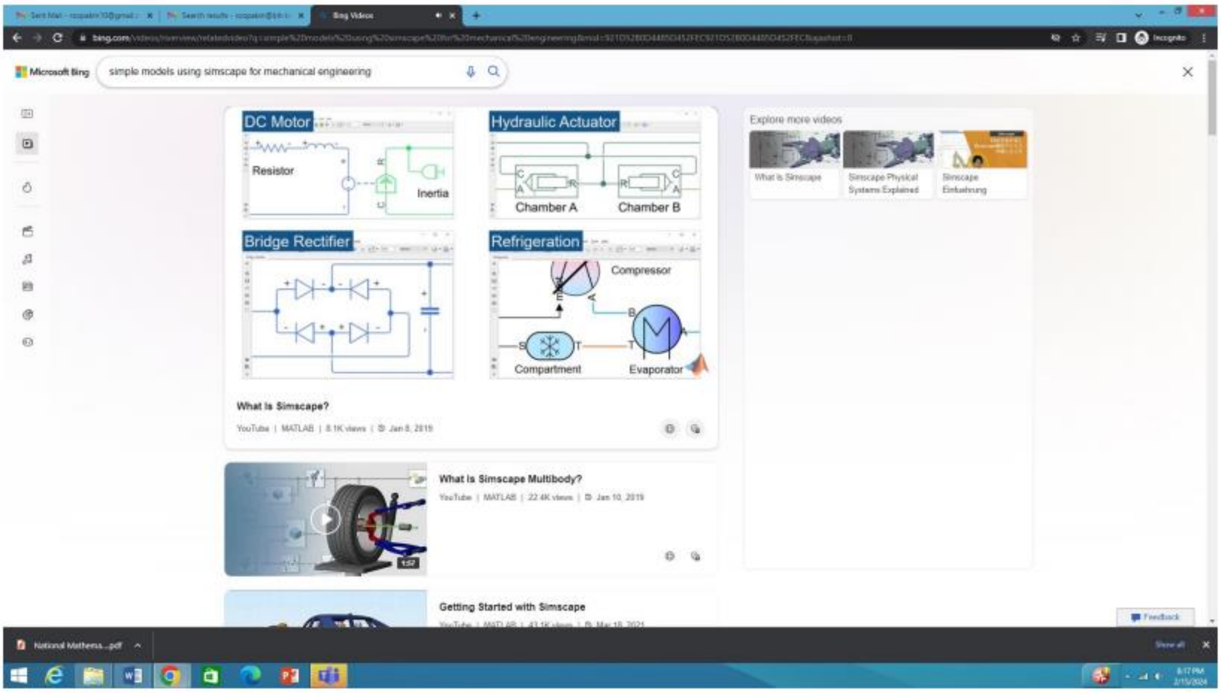
Link for study material simscap material

https://www.bing.com/videos/riverview/relatedvideo?q=what+is+simscape&mid=921D52B0D 4485D452FEC921D52B0D4485D452FEC&FORM=VIRE



simple models using simscape for mechanical engineering

Sent Mail - roopakm10@gmail...
Search results - roopakm@bit-...
Bing Videos
bing.com
Incognito
Microsoft Bing
Explore more videos
What Is Simscape
Simscape Physical Systems Explained
Simscape Einfuehrung
Electrical
Driveline
Multibody
Fluids
Simscape
What Is Simscape?
YouTube | MATLAB | 8.1K views | Jan 8, 2019
What Is Simscape Multibody?
YouTube | MATLAB | 22.4K views | Jan 10, 2019
1:57
Getting Started with Simscape
YouTube | MATLAB | 41.1K views | Mar 18, 2021
Feedback
National Mathema....pdf
Show all
8:16 PM
2/15/2024



simple models using simscape for mechanical engineering

What Is Simscape?

YouTube | MATLAB | 8.1K views | Jan 8, 2019

What Is Simscape Multibody?

YouTube | MATLAB | 22.4K views | Jan 10, 2019

Getting Started with Simscape

YouTube | MATLAB | 43.1K views | Mar 18, 2021