RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

COURSE NO: ECE - 2214

COURSE TITLE: NUMERICAL TECHNIQUES SESSIONAL

SUBMITTED BY:	SUBMITTED TO:
NAME:MD SHAHARIAR HASAN RONOK	NAME: Prof. Dr. Md. Shamim Anower
ROLL: 1710046	DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
CLASS: 2nd YEAR, EVEN SEMESTER	RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY
SESSION: 2017-2018	(RUET)
DATE OF SUBMISSION: 07.10.2020	

ACKNOWLADGEMENT

THANK YOU SO MUCH MATHWORK FOR

GIVING US THE PERMISSION TO WORK IN

YOUR WORLD KNOWN SOFTWARE "MATLAB".

WE ARE VERY MUCH GREATFUL TO YOU.

INDEX

Familiarization with MATLAB and Its build-in functions

MATLAB is a software package for high-performance mathematical computation, visualization, and programming environment. It provides an interactive environment with hundreds of built-in functions for technical computing, graphics, and animations.
MATLAB stands for Matrix Laboratory. MATLAB was written initially to implement a simple approach to matrix software developed by the LINPACK (Linear system package) and EISPACK (Eigen system package) projects.
MATLAB is a modern programming language environment, and it has refined data structures, includes built-in editing and debugging tools, and supports object-oriented programming.
MATLAB is Multi-paradigm. So, it can work with multiple types of programming approaches, such as Functional, Object-Oriented, and Visual.
Besides an environment, MATLAB is also a programming language.
As its name contains the word Matrix, MATLAB does its' all computing based on mathematical matrices and arrays. MATLAB's all types of variables hold data in the form of the array only, let it be an integer type, character type or String type variable.
MATLAB's built-in functions provide excellent tools for linear algebra computations, data analysis, signal processing, optimization, numerical solution of ordinary differential equations **(**ODEs**)**, quadrate, and many other types of scientific calculations.
Most of these functions use state-of-the-art algorithms. These are numerous functions for 2-D and 3-D graphics, as well as for animations.
MATLAB supports an external interface to run those programs from within MATLAB. The user is not limited to the built-in functions; he can write his functions in the MATLAB language.
There are also various optional "toolboxes" available from the developers of MATLAB. These toolboxes are a collection of functions written for primary applications such as symbolic computations, image processing, statistics, control system design, and neural networks.
The necessary building components of MATLAB are the matrix. The fundamental data type is the array. Vectors, scalars, real matrices, and complex matrices are all automatically handled as special cases of the primary data type. MATLAB loves matrices and matrix functions. The built-in functions are optimized for vector functions. Therefore, Vectorised commands or codes run much faster in MATLAB.

Development Environment

This is the set of tools and facilities that help you use MATLAB operations and files. Many of these tools are the graphical user interface. It involves the MATLAB desktop and command window, a command history, an editor and debugger, and browsers for considering help, the workspace, reports, and the search path.

MATLAB Mathematical Function Library

This is a vast compilation of computing design ranging from basic functions, like sum, sine, cosine, and complex mathematic, to more sophisticated features like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

MATLAB Language

This is a high level matrix/array language with control flow statement, function, data structure, input/output, and object-oriented programming characteristics. It allows both "programming in the small" to create quick and dirty throw-away programs rapidly and "programming in the large" to create large and complex application functions.

Graphics
MATLAB has extensive facilities for displaying vector and matrices as graphs, as well as annotating and printing these graphs. It contains high-level structures for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also involves low-level structures that allow us to customize the display of graphics fully as well as to build complete graphical user interfaces on our MATLAB applications.

MATLAB External Interfaces/API
This is a library that allows us to write C and FORTRAN programs that interact with MATLAB. It contains facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

MATLAB Features

As there are numerous features to describe, but here, we will focus on some of the key features:

- It is designed for numerical as well as symbolic computing.
- It's a high-level language used mainly for engineering and scientific computing.
- It works within a Desktop environment providing full features for iterative exploration, design, and problem-solving.
- Creation of custom plots for visualizing data and tools, with the help of built-in Graphics.
- Specific applications are designed to work with any particular type of problems, such as data classification, control system design and tuning, signal analysis.
- Provides several add-on toolboxes to build a wide range of engineering, scientific, and custom user interface applications.
- Provide interfaces to work with other programming languages such as C, C++, Java, .NET, Python, SQL, and Hadoop.

MATLAB Desktop

The main tools within or accessible from the MATLAB desktop are

- Command Window
- Command History Window
- Start Button
- Documents Window, containing the Editor/Debugger and the Array Editor
- Figure Windows
- Workspace Browser
- Help Browser
- Path Browser

Initializing Variables in Assignment Statement

The simplest method to initialize a variable is to assign it one or more value in an assignment statement.
An assignment statement has the standard form

```
var = expression;
```

where var is the name of the variables and expression is a scalar constant, an array, or a combination of constants, other variables, and mathematical operations (+, -, etc.). The value of the expression is computed using the standard rules of mathematics, and the resulting values are saved in the named variable. The semicolon at the last of the statement is optional. If the semicolon is absent, the values assigned to var will be echoed in the command window. If it is present, nothing will be shown in the Command Window, even though the assignment has appeared.
Examples of initializing variables with assignment statements contain

```
var = 40i;
var2 = var/5;
x = 1; y = 2;
array = [1 2 3 4];
```

The first example generates a scalar variable of type double and saves the imaginary number 40i in it.
The second example generates a scalar variable and saves the result of the expression var/5 in it.
The third example shows that multiple assignment statements can be placed on a single line, supported that they are divided by semicolons or commas.
The last example display that variables can also be initialized with arrays of data. Such arrays are build up using brackets ([]) and semicolons. All of the items of an array are listed in row order. In other words, the value in each row are recorded from left to the right, with the top-most row first, and the bottom-most row last. The single value within a row are separated by blank spaces or commas, and the rows themselves are divided by semicolons or newlines.

Initializing with Built-In Functions

Arrays can also be initialize using built-in MATLAB function. For example, the function zero can be used to generate an all-zero array of any desired size. There are a various form of the zeros function. If the function has an individual scalar argument, it will develop a square array using the single arguments as both the number of rows and the number of columns. If the function has two scalar argument, the first arguments will be the number of rows, and the second arguments will be the number of the columns. Since the size function return two values including the number of row and column in an array, it can be combined with the zero function to create an array of zeros that is the same size of another array.
Some examples using the zeros function follow:

```
a = zeros (2);
b = zeros (2, 3);
c = [1 2; 3 4];
d = zeros (size(c));
```

These statements generate the following arrays:

$$a = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

zeros(n)	Creates a n x n matrix of zeros.
zeros(m,n)	Creates a m x n matrix of zeros
zeros(size(arr))	Create a matrix of zeros of the same size as arr.
ones(n)	Creates a n x n matrix of ones.
ones(m,n)	Creates a m x n matrix of ones.
ones(size(arr))	Creates a matrix of ones of the same size as arr.
eye(n)	Creates a n x n identity matrix.
eye(m,n)	Creates an m x n identity matrix.
length(arr)	Return the length of a vector, or the longest dimension of a 2-D array.
size(arr)	Return two values specifying the number of rows and columns in arr.

Similarly, the **ones** function can be used to generate array including all ones, and the eye function can be used to generate arrays including **identity matrices**, in which all on-diagonal items are one, while all off-diagonal items are zero.

MATLAB Plotting

Creating Plotting

MATLAB makes it easy to create plots. For example in 2D, is to take a vector of **a**-coordinates, **a** = (a_1... a_n), and a vector of **b**-coordinates, **b** = (b_1...b_n), locate the points (a_i...b_i), with i = 1, 2. . . n and then connect them by straight lines.
The MATLAB commands to plot a graph is plot (a, b).
The vectors a = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10) and b = (0, 1, -1, 1, 0) produce the picture shown in figure.
>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
>> b = [0, 1, -1, 1, 0];
>> plot(a, b)

MATLAB fplot()

It is used to plot between the specific limit. The function must be of form y=f(x), where x is the vector whose specifies the limits, and y is the vector with the same size as x.
Syntax
fplot(fun, limits) //   A function fun is plotted in between the limit specified
fplot(fun, limits, linespace) // It allows plotting fun with line specification

fplot(fun, limits, tol) // It allows plotting with relative error tolerance 'tol'.If not specified de fault tolerance will be 2e-3 ie .2% accuracy.
fplot(fun, limits, tol, linespace)// It allows plotting with relative tolerance and line specific ation

Example

f(t)=t sin t, 0≤t≤10π

fplot ('x.*sin(x)',[0 10*pi])

Bar()

A bar plot is a plot in which each point is represented by a vertical bar or horizontal bar.

Syntax

bar(y) // It creates a bar graph with one bar for each element in y.
bar (x, y) // This function creates a vertical bar plot, with the values in x used to label each b ar and the values in y used to determine the height of the bar.

Example

Create Bar Graph
r^2=2 sin 5t, 0≤t≤2π
y = r sin t

```
t=linspace (0, 2*pi,200);
r=sqrt(abs(2*sin(5*t)));
y=r.*sin(t);
bar (t, y)
axis ([0 pi 0 inf]);
```

Pie()

This function creates a pie plot. This function determines the percentage of the total pie corresponding to each value of x, and plots pie slices of that size.
The optional array **explodes** controls whether or not individual pie slices are separated from the remainder of the pie.

Syntax

pie(x) // It draws a pie chart with the data in x.
pie(x, explode) // It offsets a slice from the pie.explode is a vector or matrix of zeroes and no n-zeroes corresponding to x.

Example

World population by continents.

```
cont= char('Asia','Europe','Africa',....'N.America','S.America');
pop=[3332;696;694;437;307];
pie(pop)
for i=1:5,
  gtext (cont(i,:));
end
Title ('World Population (1992)',....'fontsize', 18)
```

hist()

A histogram is a plot presenting the distribution of values within a data set. To develop a histogram, the range of values within the data set is split into evenly spaced bins, and the number of data values falling into each bin is determined.

Syntax
n=hist(y) // It bins the elements in vector y into ten equally spaced containers and returns th e number of items in each container as a row vector.
Example
Histogram of 50 randomly distributed numbers between 0 and 1.
y=randn (50, 1);
hist (y)

stem()
A two-dimensional stem plot shows data as lines extending from a baseline along the x-axis. A circle (the default) or another marker whose y-position represents the data value terminates each stem.
Syntax
stem(Y) // It plots the data sequence Y as stems that extends from equally spaced and auto matically created values along the x-
axis. When Y is a matrix, stem plot, all items in a row against the same x value.
stem(X,Y) // It plot X versus the column of Y. X and Y are vectors or matrices of a similar size. X can be the row or a column vector, and Y is a matrix with length(X) rows.
stem(...,'fill') // It specifies whether to color the circle at the end of the stem.
stem(...,LineSpec) // It specifies the line style, marker symbol, and color.
h = stem(...) // It returns a vector of Stem objects in h.
Example
f=e^-t/5 sint,0≤t≤2π
t=linspace (0, 2*pi, 200);
f=exp (-.2*t).*sin(t);
stem(t, f)

MATLAB 3D Plots

1.plot3()
The plot3 function shows a three-dimensional plot of a set of data points.
Syntax
plot3(X1,Y1,Z1,...) // where X1, Y1, Z1 are vectors or matrices, plot one or more lines in 3-D space through the points whose points are the items of X1, Y1, and Z1.
plot3(X1,Y1,Z1,LineSpec,...) // It develop and shows all the lines described by the Xn,Yn,Zn,Li neSpec quads, where LineSpec is a line specification that define line style, marker symbol, a nd color of the plotted lines.
plot3(...,'PropertyName',PropertyValue,...) // It sets properties to the specified property valu es for all Line graphics objects generated by plot3.
h = plot3(...) // It returns a column vector of handles to line graphics objects, with one handl e per line.
Example
Plot of a parametric space curve:
x(t)=t,y(t)=t^2,z(t)=t^3.
0≤t≤1
t= linspace (0, 1,100);

```
x=t; y=t.^2; z=t.^3;
plot3(x, y, z), grid
xlabel ('x(t)=t')
ylabel ('y(t)=t2')
zlabel ('z(t)=t3')
```

2.surfc()

surfc develop colored parametric surfaces specified by X, Y, and Z, with the color specified by Z or C.

Syntax

```
surf(Z)
surf(X,Y,Z)
surf(X,Y,Z,C)
surf(...,'PropertyName',PropertyValue)
surf(axes_handle,...)
surfc(...)
h = surf(...)
h = surfc(...)
hsurface = surf('v6',...), hsurface = surfc('v6',...)
```

Example

Display contour plot under surface plot.

```
[X,Y] = meshgrid(1:0.5:10,1:20);
Z = sin(X) + cos(Y);
surfc(X,Y,Z)
```

3.sphere()

The sphere function develops the x-, y-, and z-coordinates of a unit sphere for use with **surf** and **mesh**.

Syntax

sphere // It generates a sphere consisting of 20-by-20 faces.

sphere(n) // It draws a surf plot of an n-by-n sphere in the current figure.

[X,Y, Z] = sphere(...) // It returns the coordinates of a sphere in three matrices that are (n+1)-by-(n+1) in size.

Example

Generate and plot a sphere.

```
sphere(20)
axis('square')
```

or

```
[x,y,z]=sphere(20);
surf(x, y, z)
axis('square')
```

4.cylinder()

cylinder creates x, y, and z coordinates of the unit cylinder. We can draw the cylindrical object using surf or mesh, or draw it immediately by not providing output arguments.

Syntax

[X, Y, Z] = cylinder // It returns the x, y, and z coordinates of a cylinder with a radius similar to 1. The cylinder has 20 similar spaced points around its circumference.

[X, Y, Z] = cylinder(r) // It returns the x, y, and z coordinates of a cylinder using r to describe a profile curve. cylinder treats each component in r as a radius at equally spaced heights along with the unit height of the cylinder.

[X,Y,Z] = cylinder(r,n) // It returns the x, y, and z coordinates of a cylinder based on the profile curve described by vector r. The cylinder has n similar spaced points around its circumference.

cylinder(...) // with no output arguments, plot the cylinder using MATLAB surf.

Example

r=sin?(3π z)+2

0≤z≤1,0≤θ≤2θ

```
z=[0: .02:1]';
r=sin(3*pi*z)+2;
cyclinder(r), axis square
```

Multi-Dimensional Arrays in MATLAB

- Arrays with one more than two dimensions are called multi-dimensional arrays.
- Multi-dimensional arrays are created with more than two subscripts in MATLAB.
- For example:
  - Let's create a three-dimensional array using function ones (3, 8, 3).
  - This function creates a 3-by-8-by-3 array with a total of 3*8*3 = 72 elements.
  - The third subscript tells to create no. of sets of elements in rows and columns as per first & second subscripts.
- Let's have one more example:
- Here we use some more functions, and one of them is the **perms** function.
- The **perms** function returns all number of possible ways or permutations to arrange the elements of a matrix or vector in a different set of orders of a row vector.

MATLAB Symbolic Mathematics

Symbolic mathematics defines doing mathematics on symbols (not numbers!). For example, a+a is 2a. The symbolic math function is in the Symbolic Math Toolbox in MATLAB. Toolboxes include related functions and are add-ons to MATLAB.

Symbolic Variables and Expressions

MATLAB has type called **sym** for symbolic variables and expressions, and these work with string.

For example, to generate a symbolic variable **a** and perform the addition just defined, first, a symbolic variable will be created by passing the string **'a'** to the **sym** function:

```
>> a = sym('a');
>> a+a
ans =  2*a
```

Symbolic variables can also store expressions. For example, the variables b and c save symbolic expressions:

```
>> b = sym('x^2');
>> c = sym('x^4');
```

All basic numerical operations can be performed on symbolic variables and expressions (e.g., add, subtract, multiply, divide, raise to a power, etc.).
Here are some examples:

```
>> c/b
ans =  x^2
>> b^3
ans =  x^6
>> c*b
ans =  x^6
>> b + sym('4*x^2')
ans =  5*x^2
```

Simplification Functions

Several functions work with expressions and simplify the terms. It is not all expressions that can be simplified, but the simplify function does whatever it can to simplify expressions containing gathering like terms.
For example:

```
>> x = sym('x');
>> myexpr = cos(x)^2 + sin(x)^2
myexpr =  cos(x)^2 sin(x)^2
>> simplify(myexpr)
ans =  1
```

The functions **collect, expand,** and **factor** works with polynomial expressions. The collect function collects coefficients.
For example:

```
>> x = sym('x');
>> collect(x^2+4*x^3+3*x^2)
ans =  4*x^2+4*x^3
```

The **expand** functions will multiply out terms, and element will do the reverse:

```
>> expand((x+2) *(x-1))
ans =  x^2+x-2
>> factor(ans)
ans = (x+2)*(x-1)
```

The **subs** function will substitute an equation for the symbolic variable in expressions.
For example:

```
>> myexp = x^3^ +3*x^2^ -2
myexp =  x^ ^3^ +3*x^2^ -2
>> x = 3;
>> subs (myexp, x)
ans = 52
```

With symbolic mathematics, MATLAB works by default with rational numbers, defining that results are kept in fractional forms. For example, performing the addition of 1/3+1/2 would usually result in a double value:

```
>> 1/3 + 1/2
ans =  0.8333
```

However, by making the function symbolic, the result is symbolic also. Any mathematical function (e.g., double) can modify that:

```
>> sym(1/3 + 1/2)
```

ans = 5/6
>> **double**(ans)
ans =0.8333
The **numden** functions will return the numerator and denominator of the symbolic expressions separately.
>> sym(1/3+1/2)
ans = 5/6
>> [n, d] = numden(ans)
n = 5
d = 6
>> [n, d] = numden((x^3^ +x^2)/x)
n = x^2*(x+1)
d = x

MATLAB Environment Programming

Objective: To study MATLAB environment and to familiarize with command window, history, workspace, current directory, figure window, edit window, shortcuts, helplines.

MATLAB is a particular computer program optimized to perform engineering and scientific calculations.

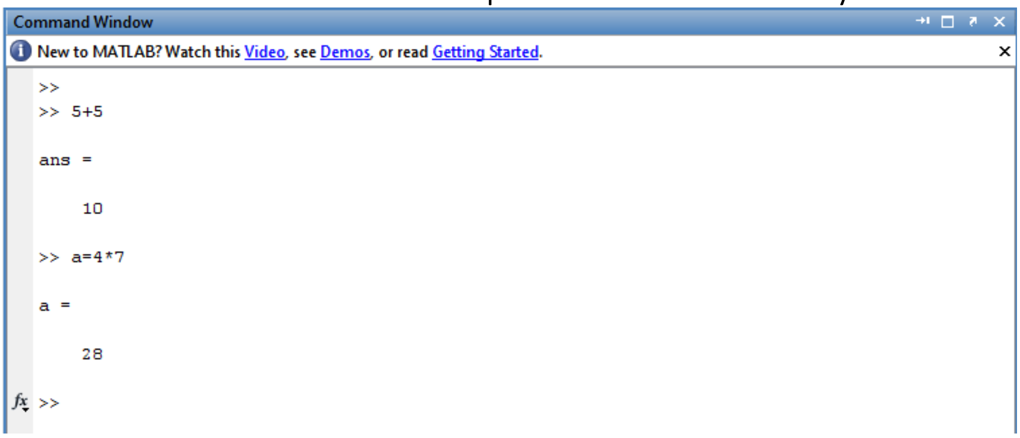
Advantages:

- Ease of use.
- Platform independence.
- Predefined functions.
- Device-independent plotting.
- GUI.
- MATLAB compiler.

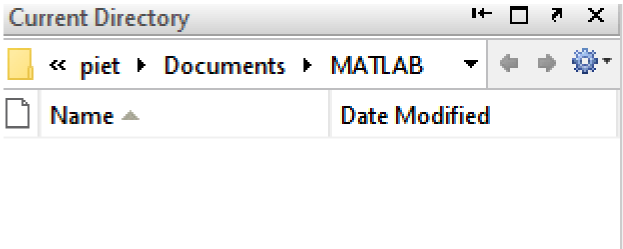
Disadvantages: Interpreted language.The cost is high.

MATLAB Environment

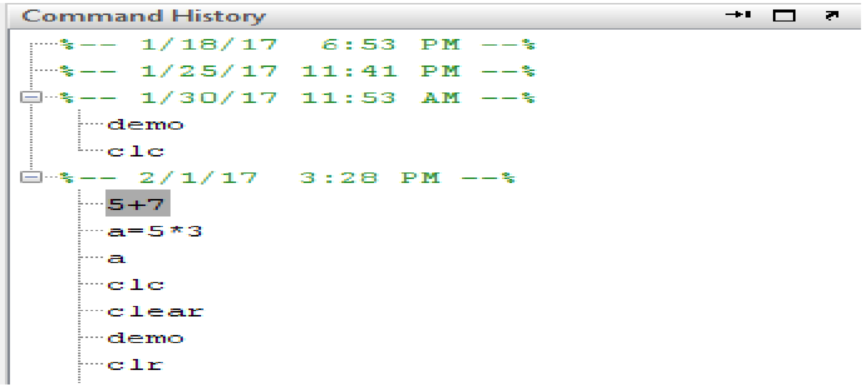
- **Command Window:** It is the space where commands may be entered.



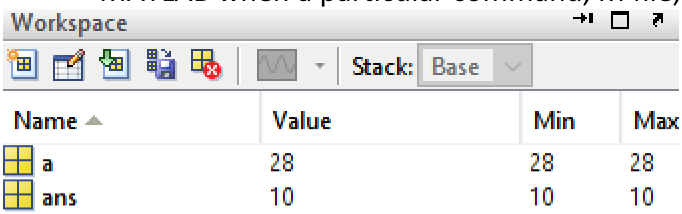
- **Figure Window:** It displays plots and graphs.
- **Edit Window:** It permits a user to create and modify MATLAB programs by creating new M files or to modify existing ones.
- **Current Directory Window:** It shows the path of the current directory.



- **Command History:** It displays a list of commands that the user has entered in the command window.



- **Workspace:** It is a collection of all the variables and array that can be used by MATLAB when a particular command, M file, or function is executed.



- **Help Files:** A user can get help from MATLAB through MATLAB documentation.

Shortcuts of MATLAB:

In MATLAB, if the statement is too long to type on a single line, it may be continued on successive lines by typing an ellipsis (...) at the end of the first line and then continuing on the next line.

Ex: a=1/2+3/2-2/3 ...+4/5-2/3;

Shortcuts:

- **clc:** clear command window.
- **clf:** clear contents of the current figure window
- **clear:** clears variables in workspace
- **abort:** (ctrl+C) For M files that appears running too long may contain an infinite loop that never terminates. To terminate, we use abort.
- **!:** It is a special character, after which any character or command will be sent to the operating system and executed as they had been types in operating system command prompt.

- **diary:** (diary filename)
  After this command, a copy of all inputs and most of the outputs typed in the command window will be echoed in the diary file.
- **diary off:** It suspends input into the diary file.
- **diary on:** It resumes input again.
- **which:** It tells which version of a file is being executed and where it is located.

MATLAB Control Statements

Objective: To study control structures (for, while, if, switch, break, continue, input/output functions, reading, and storing data).

If: If evaluates a logical expression and executes a group of statements based on the value of the expression.

Syntax of If Statement

```
if expression 1
statement1
elseif expression 2
        statement 2
        else
        statement 3
        end
```

Examples

```
>> a=7

a =  7

>> if a>0
disp('a is positive');
elseif a<0
disp('a is negative')
else
disp('a is zero')
end
```

Output:

a is positive

Switch, case, and otherwise: Switch executes certain statements based on the value of a variable or expression. Its basic form is

Syntax

```
switch  switch expression
 case    case expression
             statements
        .
        .
        otherwise
        statements
        end
```

An evaluated switch expression is a scaler or string. An evaluated case expression a scaler, a string, or a cell array of scaler or strings. The switch block tests each case is until one of the cases is true.

Examples

Conditionally display different text depending on value entered at the command line.

```
>> mynumber=input('enter a number')
enter a number -1

mynumber = -1

>> switch mynumber
case -1
disp('negative one')
case 0
disp('zero');
case 1
disp('positive one');
otherwise
disp('other value');
end
```

Output:

```
negative one
```

Example 2:

```
>> result=52;
>> switch(result)
case 52
disp('result is 52')
case {52,78}
disp('result is 52 or 78')
end
```

Output:

```
result is 52
```

Example 3:

```
>> [daynum, daystr] =weekday(date,'long','en_US')
switch(daystr)
case 'monday'
disp('start of week')
case 'tuesday'
disp('day 2')
otherwise
disp('weekend')
end
```

Output:

```
weekend
```

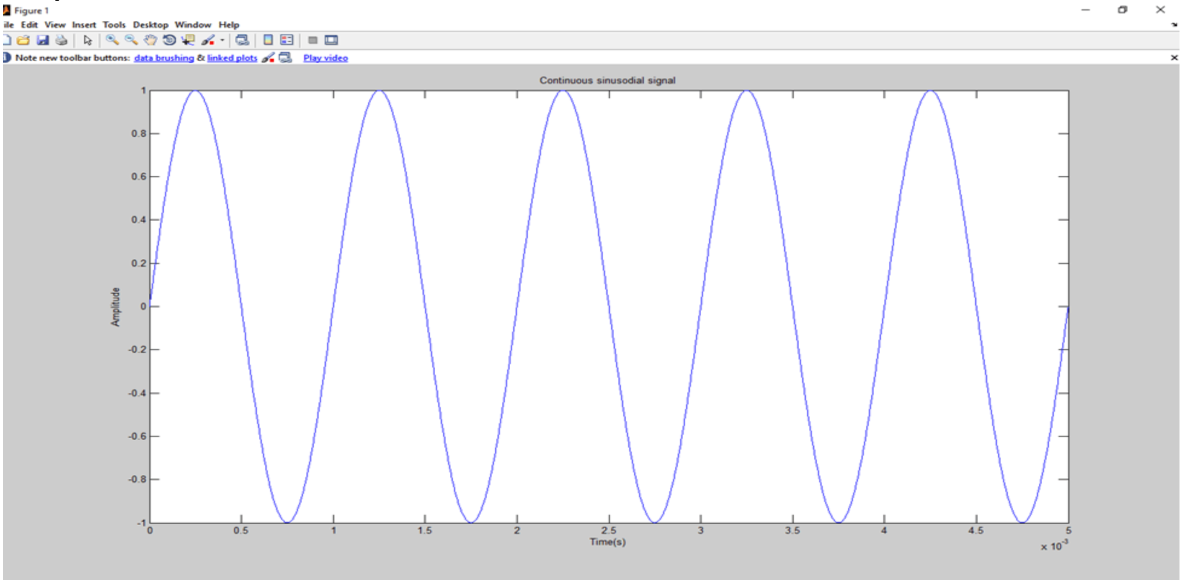
MATLAB Sine Wave Plot

Objective: To plot a sine wave of the frequency of 1KHz.

Example: Let's generate a simple continuous like sinusoidal signal with frequency FM=1KHz. In order to make it occur as a repetitive signal when plotting, a sampling rate of fs=500KHz is used.

```
fs= 500e3;
f= 1000;
nCyl=5;
t=0:1/fs:nCyl*1/f;
x=sin(2*pi*f*t);
plot(t,x)
title ('Continuous sinusoidal signal')
xlabel('Time(s)');
ylabel('Amplitude');
```

Output:



MATLAB Functions

Function	Description
dblquad	Numerically evaluate double integral
erf	Error function
feval	Execute function specified by string
fzero	Scalar nonlinear zero finding
spline	Cubic spline data interpolation
abs	Absolute value
inline	Construct INLINE object

Overview of Lab Report:

- Writing Scripts and Functions, Simple Calculations with MATLAB
- Non Linear Algebraic Equation
- Linear Algebraic Equations
- Interpolation
- Curve Fitting
- Numerical Integration
- Ordinary Differential Equations

Experiment No: 01

Name of the Experiment: Study of Bisection Method to Obtain the Roots of a Nonlinear Equation.

Objectives:

The objective of this experiment is to find the value of root of an equation by bisection method, using Matlab for a very precise value.

Theory:

For Mathematics & Numerical Methods in order to find the roots, the Bisection method is a renowned one. For a Polynomial Equation f(x) =0, its roots are founded by applying this method, provided that the roots lie within the interval [a, b] and f(x) is continuous in the interval.

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

Calculate c, the midpoint of the interval, c = (a + b)/ 2

1. Calculate the function value at the midpoint, f(c).

2. If convergence is satisfactory (i.e., c - is sufficiently small, or |f(c)| is sufficiently small), return c and stop iterating.

3. Examine the sign of f(c) and replace either (a, f (a)) or (b, f (b)) with (c, f(c)) so that there is a zero crossing within the new interval.

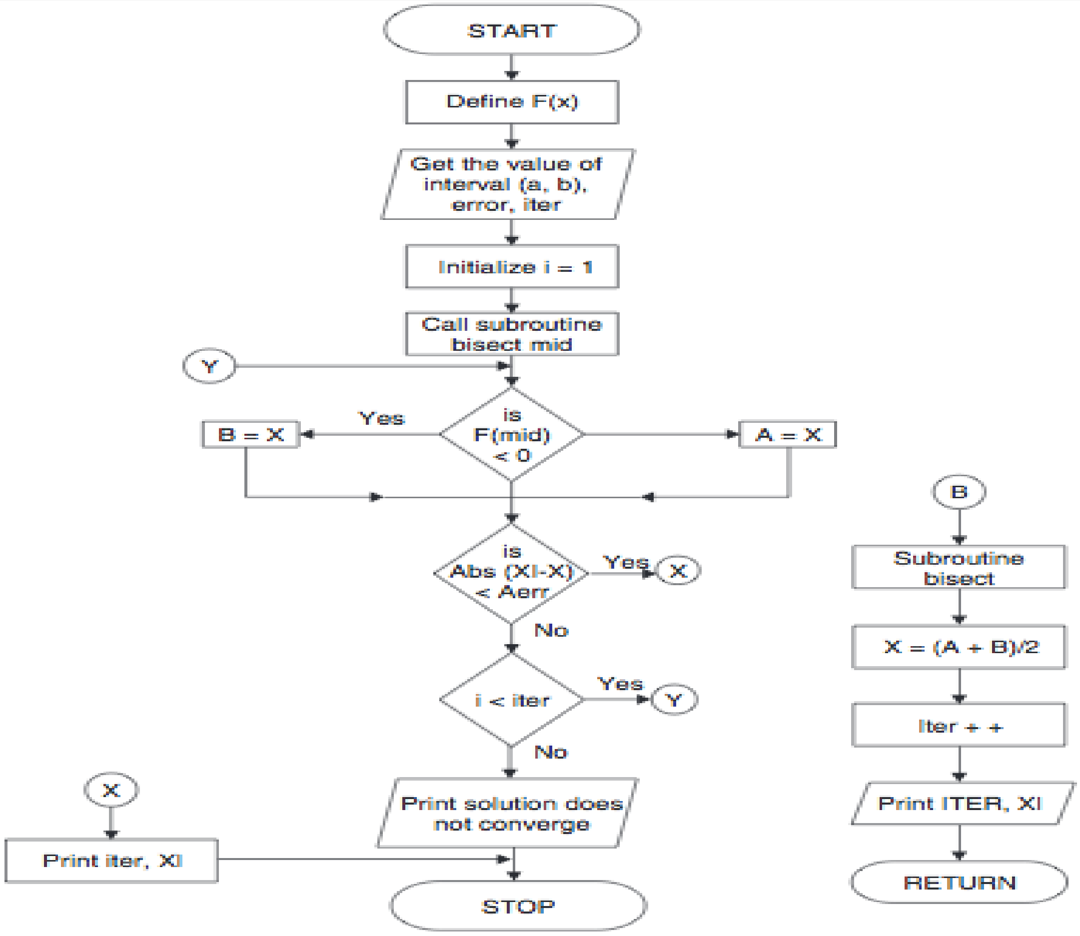
Tools:

Methodology:

Algorithm of Bisection Method:

Step 1: Choose lower *a* and upper *b* guesses for the root such that the function changes sign over the interval. This can be checked by ensuring that $f(a)f(b) < 0$.
Step2 : An estimate of the root *c* is determined by $c = (a + b)/2$
Step 3: Make the following evaluations to determine in which subinterval the root lies:
(*a*) If $f(a)f(c) < 0$, the root lies in the lower subinterval.Therefore, set $b = c$ and return step 2.
(*b*) If $f(a)f(c) > 0$, the root lies in the upper subinterval. Therefore, set $a = c$ and return to step 2.
(*c*) If $f(a)f(c) = 0$, the root equals *c*; terminate the computation.

Flowchart:



Bisection Method in MATLAB Code:

The given function is f(x) =2x^2-15x+3

```
y= @(x) 2*x^2-15*x+3 ;


while(1)
    a=input('Enter the value of 1st assumption:');
    b=input('Enter the value of 2nd assumption:');
    if y(a)*y(b)>0
    fprintf('WRONG!!\n');
    elseif y(a)*y(b)<0
        break;
    end
end
```

```
if y(a)==0
    fprintf('Root')
    return
elseif y(b)==0;
    fprintf('Root')
    return
end
display('No.    a          b          c         y')
display('-------------------------------------------')
a_arr = [];
b_arr = [];
c_arr = [];
y_arr = [];
i_arr = [];
col={'a','b','c','y'};

for i=1:1:100
    c=(a+b)/2;
     if abs(y(c))<.001
        break;
    end
    c_arr(i) = c;
    y_arr(i) = y(c);
    i_arr(i)=i;
    if y(a)*y(c)>0
        a=c;
        a_arr(i) = c;
        if i == 1
            b_arr(i) = b;
        else
            b_arr(i) = b_arr(i-1);
        end
    else
        b=c;
        b_arr(i) = c;
        if i == 1
            a_arr(i) = a;
        else
            a_arr(i) = a_arr(i-1);
        end
    end

    fprintf('%d  %f  %f  %f  %f \n',i,a,b,c,y(c));
    uitable('columnname',col,'rowname',i_arr,'data',[
a_arr',b_arr',c_arr',y_arr'],'position', [500 200 335 238] );%x,y,table
decrease from right to left,
end

datatable = table(a_arr', b_arr', c_arr',
y_arr','VariableNames',{'a','b','c','y'});
```

Output:

```
Editor - E:\4k downloader\2-2\BOOKS\RONOK\NM\c
Bisec2tab.m
            break;
        end
        c_arr(i) = c;
        y_arr(i) = y(c);
        i_arr(i)=i;
        if y(a)*y(c)>0
            a=c;
            a_arr(i) = c;
            if i == 1
                b_arr(i) = b;
            else
                b_arr(i) = b_arr(i-1);
            end
        else
            b=c;
            b_arr(i) = c;
```

```
Command Window
>> Bisec2tab
Enter the value of 1st assumption:0
Enter the value of 2nd assumption:1
No.    a          b          c          y
-------------------------------------------
1  0.000000  0.500000  0.500000  -4.000000
2  0.000000  0.250000  0.250000  -0.625000
3  0.125000  0.250000  0.125000  1.156250
4  0.187500  0.250000  0.187500  0.257813
5  0.187500  0.218750  0.218750  -0.185547
6  0.203125  0.218750  0.203125  0.035645
7  0.203125  0.210938  0.210938  -0.075073
8  0.203125  0.207031  0.207031  -0.019745
9  0.205078  0.207031  0.205078  0.007942
10  0.205078  0.206055  0.206055  -0.005903
11  0.205566  0.206055  0.205566  0.001019
12  0.205566  0.205811  0.205811  -0.002442
>>
```

	a	b	c	y
1	0	0.5000	0.5000	-4
2	0	0.2500	0.2500	-0.6250
3	0.1250	0.2500	0.1250	1.1563
4	0.1875	0.2500	0.1875	0.2578
5	0.1875	0.2188	0.2188	-0.1855
6	0.2031	0.2188	0.2031	0.0356
7	0.2031	0.2109	0.2109	-0.0751
8	0.2031	0.2070	0.2070	-0.0197
9	0.2051	0.2070	0.2051	0.0079
10	0.2051	0.2061	0.2061	-0.0059
11	0.2056	0.2061	0.2056	0.0010
12	0.2056	0.2058	0.2058	-0.0024

Fig1: Bisection method creating Table

Results & Discussion:

The resultant root of the given function is 0.2058.

Discussion:

In this experiment, we can get one root of the given nonlinear function after 12 iterations. And the value of resultant root is very close to the original root.

References:

https://www.youtube.com/watch?v=fCKUOWiM-6s

https://www.codewithc.com/bisection-method-in-matlab/

Experiment No: 02

Name of the Experiment: Study of False Position Method to Obtain the Root(s) of a Nonlinear Equation.

Objectives: The objective of this experiment is to apply false position method to find out the very precise value of the root of an equation, using MATLAB.

Theory: If the function $f(x)$ is continuous in $[a, b]$ and $f(a)f(b) < 0$ (i.e. the function f has values with different signs at a and b), then a value $c \in (a, b)$ exists such that $f(c) = 0$[1].

The false position algorithm attempts to locate the value c where the plot of f crosses over zero, by checking whether it belongs to either of the two sub-intervals [a,c],[c,b],where c is the midpoint
c=[a*f(b)- b*f(a)]/[f(b)- f(a)]

Tool: MATLAB Software

Methodology:

(I) Algorithm: Step 1: Choose lower a and upper b guesses for the root such that the function changes sign over the interval. This can be checked by ensuring that $f(a)f(b) < 0$.
Step2 : An estimate of the root c is determined by c=[a*f(b)- b*f(a)]/[f(b)- f(a)]
Step 3: Make the following evaluations to determine in which subinterval the root lies:
(*a*) If $f(a)f(c) < 0$, the root lies in the lower subinterval. Therefore, set $b = c$ and return to step 2.
(*b*) If $f(a)f(c) > 0$, the root lies in the upper subinterval. Therefore, set $a = c$ and return to step 2.
(*c*) If $f(a)f(c) = 0$, the root equals c; terminate the computation.[4 chap]

(II)Flowchart:

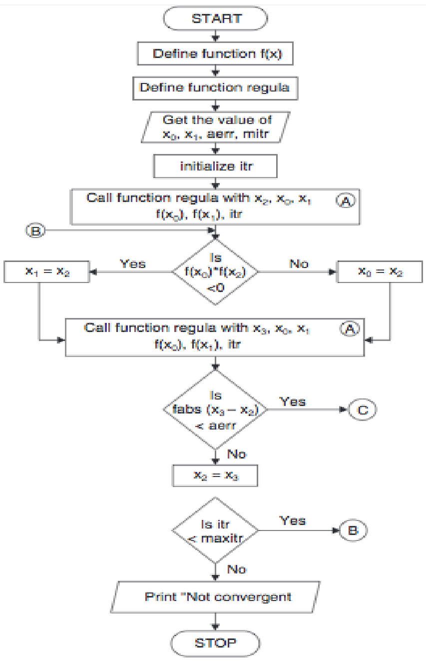
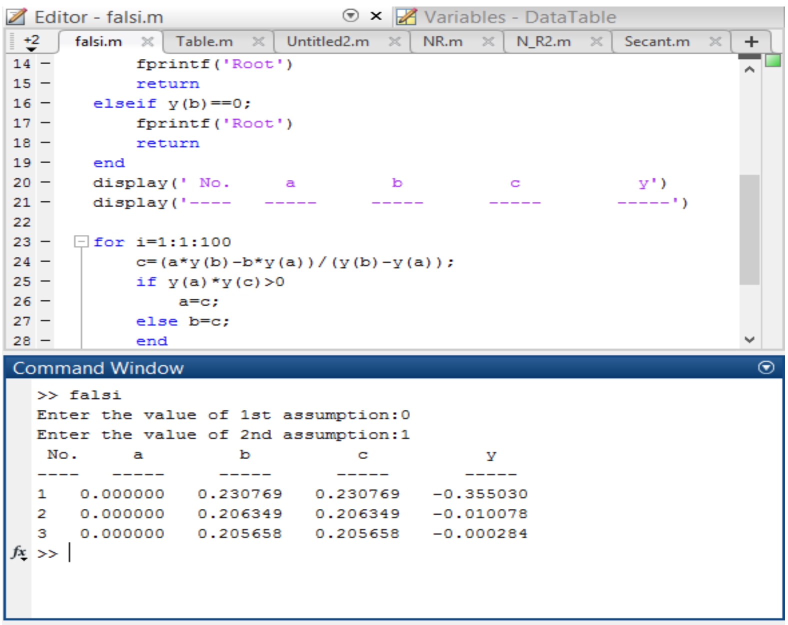


Figure 2.1 Flowchart of bisection method procedure [2]

(III) MATLAB Code: The given function is f(x) =2x^2-15x+3

```
y= @(x) 2*x^2-15*x+3 ;

while(1)
    a=input('Enter the value of 1st assumption:');
    b=input('Enter the value of 2nd assumption:');
    if y(a)*y(b)>0
       fprintf('WRONG!!\n');
    elseif y(a)*y(b)<0
        break;
    end
end
if y(a)==0
    fprintf('Root')
    return
elseif y(b)==0;
    fprintf('Root')
    return
end
display(' No.      a         b          c           y')
display('----    -----     -----      -----       -----')

for i=1:1:100
    c=(a*y(b)-b*y(a))/(y(b)-y(a));
    if y(a)*y(c)>0
        a=c;
    else b=c;
    end
    if abs(y(c))<.0001
        break;
    end
    fprintf('%d   %f   %f   %f   %f \n',i,a,b,c,y(c));
    datatables=table(a,b,c,y(c));
end
```

Output:



Result& Discussion: The roots of the given function is 0.205688.Which is nearly close to the original value (0.205638) direct calculated by calculator.
Conclusion: So from the above test we saw that nearly 3rd iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015
[2] *Regula Falsi Method Algorithm and Flowchart,*CODEWITHC, April 21, 2014.Accessed on: Jan. 23,2020[online].

Available:https://www.codewithc.com/regula-falsi-method-algorithm-flowchart/

Experiment No: 03

Name of the Experiment: Study of Newton-Raphson(NR) Iterative Method to Obtain the Root(s) of a Nonlinear Equation.

Objectives: The objective of this experiment is to apply NR iterative method to find out the very precise value of the root of an equation, using MATLAB.

Theory: The **Newton-Raphson method** (also known as Newton's method) is a way to quickly find a good approximation for the root of a real-valued function $f(x)=0$. It uses the idea that a continuous and differentiable function can be approximated by a straight line tangent to it[1].

Suppose you need to find the root of a continuous, differentiable function $f(x)$, and you know the root you are looking for is near the point $x=x0$. Then Newton's method tells us that a better approximation for the root is $x1=x0-f'(x0)/f(x0)$.

This process may be repeated as many times as necessary to get the desired accuracy. In general, for any xx-value xn, the next value is given by

$$xn+1=xn-f'(xn)/f(xn)$$

Tool: MATLAB Software

Methodology:

(I) Algorithm:

1. Start
2. Read x, e, n, d ,*x is the initial guess, e is the absolute error i.e the desired degree of accuracy,n is for operating loop,d is for checking slope*
3. Do for i =1 to n in step of 2
4. f = f(x)
5. f1 = f'(x)
6. If ([f1] < d), then display too small slope and goto 11.
   [ ] is used as modulus sign
7. x1 = x – f/f1
8. If ([(x1 – x)/x1] < e), the display the root as x1 and goto 11.
   [ ] is used as modulus sign
9. x = x1 and end loop
10. Display method does not converge due to oscillation.
11. Stop

(II)Flowchart:

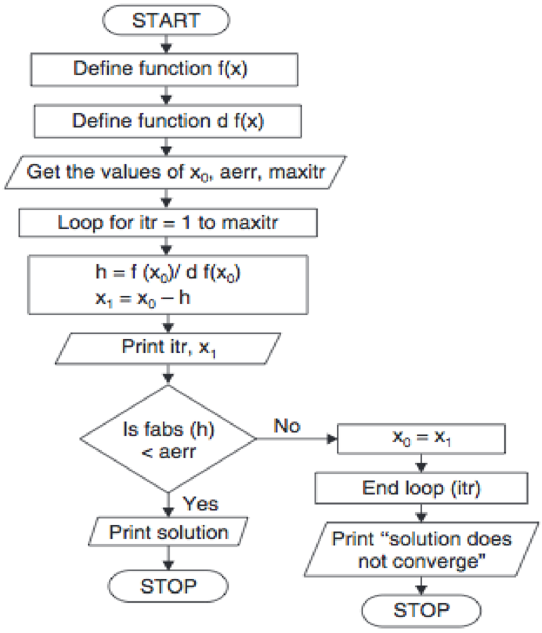


Figure 2.1 Flowchart of Newton-Raphson method procedure [2]

(III) MATLAB Code: The given function is f(x) =2x^2-15x+3

```
clear all
clc

syms x;
fun=input('Enter the fun:');
f=inline(fun);
z=diff(f(x));
f1=inline(z);
x0=input('Enter initial value:');
x=x0;
display(' No.     y        f(a)        f1(a)          x')
display('----   -----    -----      -----        -----')
for i=0:1:15
    y=x;
    x=y-(f(x)/f1(x));
    if x==y
        break
    else fprintf(' %d   %f   %f   %f   %f \n',i,y,f(x),f1(x),x);
    end
end
```

Output:

Editor - N_R2.m

Variables - DataTable

falsi.m | Table.m | Untitled2.m | NR.m | N_R2.m | Secant.m

```
f=inline(fun);
z=diff(f(x));
f1=inline(z);
x0=input('Enter initial value:');
x=x0;
display(' No.     y        f(a)        f1(a)         x')
display('----    -----     -----       -----        -----')
for i=0:1:15
    y=x;
    x=y-(f(x)/f1(x));
    if x==y
        break
    else fprintf(' %d   %f   %f   %f   %f \n',i,y,f(x),f1(x),
    end
```

Command Window

```
Enter the fun:2*x^2-15*x+3
Enter initial value:0
 No.     y         f(a)        f1(a)         x
----   -----     -----       -----        -----
 0   0.000000   0.080000   -14.200000   0.200000
 1   0.200000   0.000063   -14.177465   0.205634
 2   0.205634   0.000000   -14.177447   0.205638
 3   0.205638   0.000000   -14.177447   0.205638
>>
```

script | Ln 20 Col 4

Result& Discussion: The roots of the given function is 0.205638.Which is equal to the original value (0.205638) directly calculated by calculator.

Conclusion: So from the above test we saw that nearly 3rd iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

[2] *Newton-Raphson Method Algorithm and Flowchart,*CODEWITHC, April 21, 2014.Accessed on: Jan. 23,2020[online].

Available: https://www.codewithc.com/newton-raphson-method-algorithm-flowchart/

Experiment No: 04

Name of the Experiment: Study of Secant Method to Obtain the Root(s) of a Nonlinear Equation.

Objectives: The objective of this experiment is to apply Secant method to find out the very precise value of the root of an equation, using MATLAB.

Theory: x_0 and x_1 are two initial approximations for the root (s) of f(x) = 0 and $f(x_0)$ & $f(x_1)$ respectively, are their function values. If x_2 is the point of intersection of x-axis and the line-joining the points (x_0, $f(x_0)$) and (x_1, $f(x_1)$) then x_2 is closer to 's' than x_0 and x_1[1].

$$x_2 = x_1 - f(x_1) *[(x_1-x_0)/ f(x_1) - f(x_0)]$$

or in general the iterative process can be written as

$$x_{i+1} = x_i - f(x_i) *[(x_i - x_{i-1})/ f(x_i) - f(x_{i-1})] \quad i=1,2,3\ldots$$

Tool: MATLAB Software

Methodology:

(I) Algorithm:

1. Start
2. Get values of x0, x1 and e
   *Here x0 and x1 are the two initial guesses
   e is the stopping criteria, absolute error or the desired degree of accuracy*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0*f(x1) – x1*f(x0)] / [f(x1) – f(x0)]
5. Test for accuracy of x2
   If [(x2 – x1)/x2] > e, *Here [ ] is used as modulus sign*
   then assign x0 = x1 and x1 = x2
   goto step 4
   Else, goto step 6
6. Display the required root as x2.
7. Stop

(II)Flowchart:

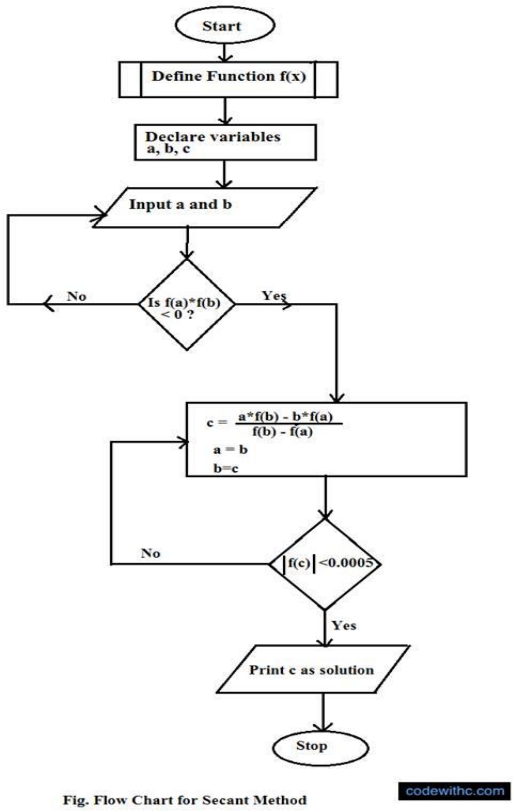


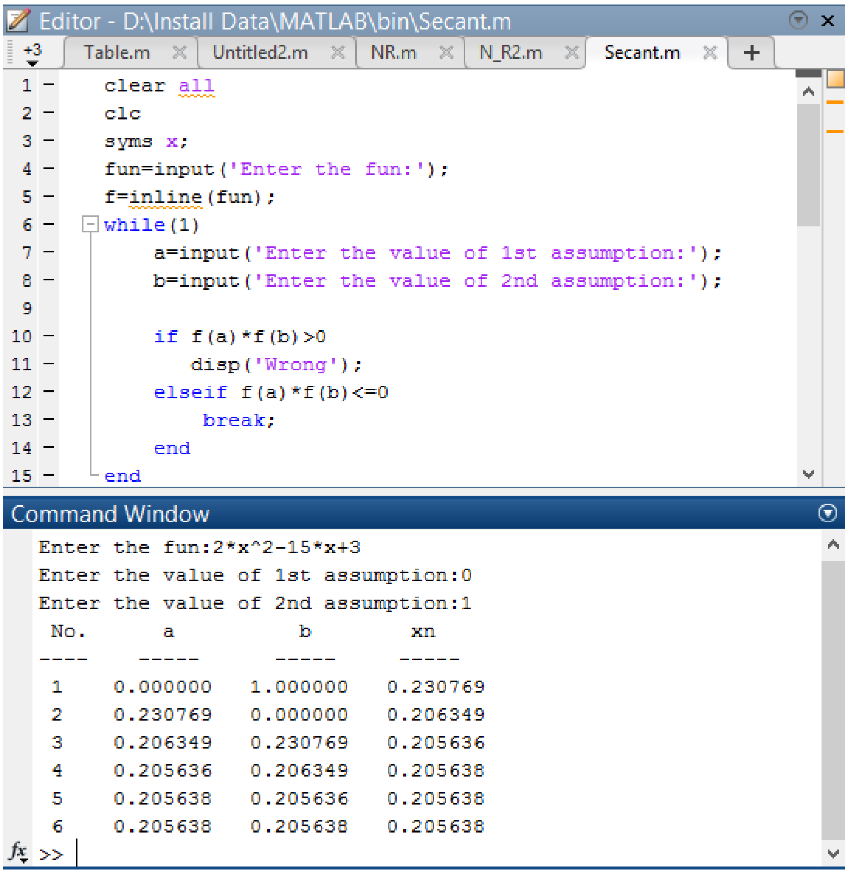
Figure 2.1 Flowchart of secant method procedure [2]

(III) MATLAB Code: The given function is f(x) =2x^2-15x+3

```
clear all
clc
syms x;
fun=input('Enter the fun:');
f=inline(fun);
while(1)
    a=input('Enter the value of 1st assumption:');
    b=input('Enter the value of 2nd assumption:');

    if f(a)*f(b)>0
       disp('Wrong');
    elseif f(a)*f(b)<=0
        break;
    end
end
if f(a)==0
    fprintf('Root')
    return
elseif f(b)==0;
    fprintf('Root')
    return
end
display(' No.      a          b          xn    ')
display('----     -----      -----      ----- ')

for i=1:1:100
    x=a-b;
```

```
    z=f(a)-f(b);
    xn=a-(x/z)*f(a);
    if xn==a
        break
    else fprintf(' %d    %f    %f    %f\n',i,a,b,xn);
    end
     b=a;
     a=xn;
end
```

Output:



Result& Discussion: The roots of the given function is 0.205638.Which is equal to the original value (0.205638) directly calculated by calculator.

Conclusion: So from the above test we saw that nearly 6th iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers"*, 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

[2] *Secant Method Algorithm and Flowchart,*CODEWITHC, April 21, 2014.Accessed on: Jan. 23,2020[online].Available: https://www.codewithc.com/secant-method-algorithm-flowchart/

Experiment No: 05

Name of the Experiment: Study of Successive Approximations (SA) Method to Obtain the Root(s) of a Nonlinear Equation.

Objectives: The objective of this experiment is to apply SA method to find out the very precise value of the root of an equation, using MATLAB.

Theory: This open method employs a formula to predict the root. Such a formula can be developed for simple fixed-point iteration (or, as it is also called, one-point iteration or successive substitution) by rearranging the function f(x) =0 so that x is on the left-hand side of the equation: x= g(x)….(1)

This transformation can be accomplished either by algebraic manipulation or by simply adding x to both sides of the original equation. For example,

x^2-2x+3=0 can be simply manipulated to yield x=(x^2+3)/2

The utility of the equation 1 is that it provides a formula to predict a new value of x as a function of an old value of x. Thus, given an initial guess at the root xi, equation 1 can be used to compute a new estimate x_{i+1} as expressed by the iterative formula
x_{i+1} = g(x_i) …….(2)

Tool: MATLAB Software

Methodology:

(I) Algorithm:

1. Start
2. Read values of x0 and e.
   *Here x0 is the initial approximation
   e is the absolute error or the desired degree of accuracy, also the stopping criteria*
3. Calculate x1 = g(x0)
4. If [x1 – x0] <= e, goto step 6.
   Here [ ] refers to the modulus sign
5. Else, assign x0 = x1 and goto step 3.
6. Display x1 as the root.
7. Stop

(II)Flowchart:

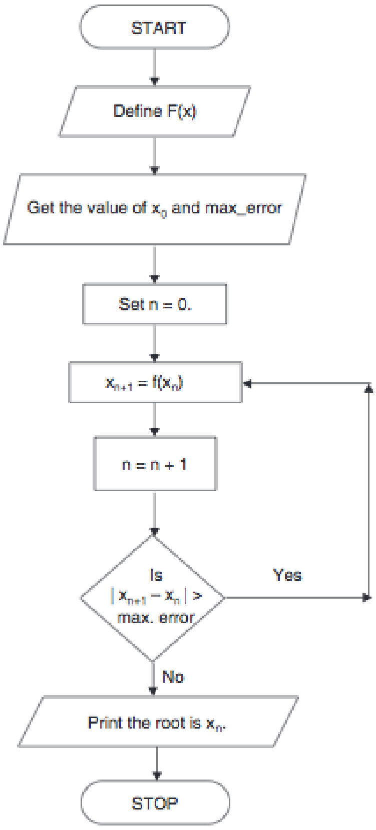
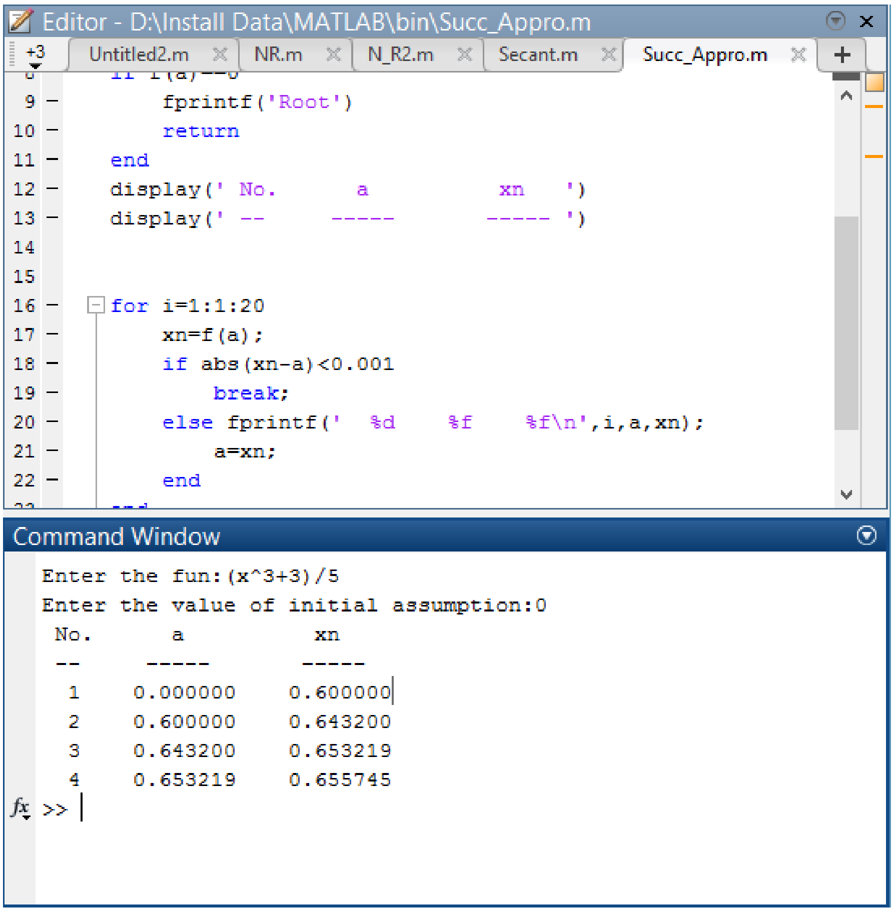


Figure 2.1 Flowchart of iteration method procedure [2]

(III) MATLAB Code: The given function is f(x) =(x^3+3)/5

```
clear all
clc
syms x;
fun=input('Enter the fun:');
f=inline(fun);
   a=input('Enter the value of initial assumption:');

if f(a)==0
    fprintf('Root')
    return
end
display(' No.      a          xn    ')
display(' --      -----       ----- ')
for i=1:1:20
    xn=f(a);
    if abs(xn-a)<0.001
        break;
    else fprintf('   %d     %f     %f\n',i,a,xn);
        a=xn;
    end
end
```

Output:



Result& Discussion: The roots of the given function is 0.655745.Which is nearly close to the original value (0.65634) direct calculated by calculator.

Conclusion: So from the above test we saw that nearly 4th iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

[2] *Iteration Method Algorithm and Flowchart,*CODEWITHC, April 21, 2014.Accessed on: Jan. 23,2020[online].

Available: https://www.codewithc.com/iteration-method-algorithm-flowchart/

Experiment No: 06

Name of the Experiment: Study of Gauss Elimination (GE) Method to Find the Solution of Simultaneous Equations.

Objectives: The objective of this experiment is to apply GE method to find out the very precise values of the equations, using MATLAB.

Theory: Solving of a system of linear algebraic equations appears frequently in many engineering problems. Most of numerical techniques which deals with partial differential equations, represent the governing equations of physical phenomena in the form of a system of linear algebraic equations. Gauss elimination technique is a well-known numerical method which is employed in many scientific problems.
Consider an arbitrary system of linear algebraic equations as follows:

$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = c_1$

$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = c_2$

....

$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = c_n$

where x_i are unknowns and a_{ij} are coefficients of unknowns and c_i are equations' constants.
This system of algebraic equation can be written in the matrix form as follows:
[A]{x}={C}
Where [A] is the matrix of coefficient and {x} is the vector of unknowns and {C} is the vector of constants. Gauss elimination method eliminate unknowns' coefficients of the equations one by one. Therefore the matrix of coefficients of the system of linear equations is transformed to an upper triangular matrix. The last transformed equation has only one unknown which can be determined easily. This evaluated unknown can be used in the upper equation for determining the next unknown and so on. Finally the system of linear equations can be solved by back substitution of evaluated unknowns[1].
Tool: MATLAB Software

Methodology:

(I)**Algorithm:**

1. Start
2. Declare the variables and read the order of the matrix n.
3. Take the coefficients of the linear equation as:
   pivot matrix 1 1 value then calculate .
4. Pivot matrix value 2 2 and make zero 3 2.

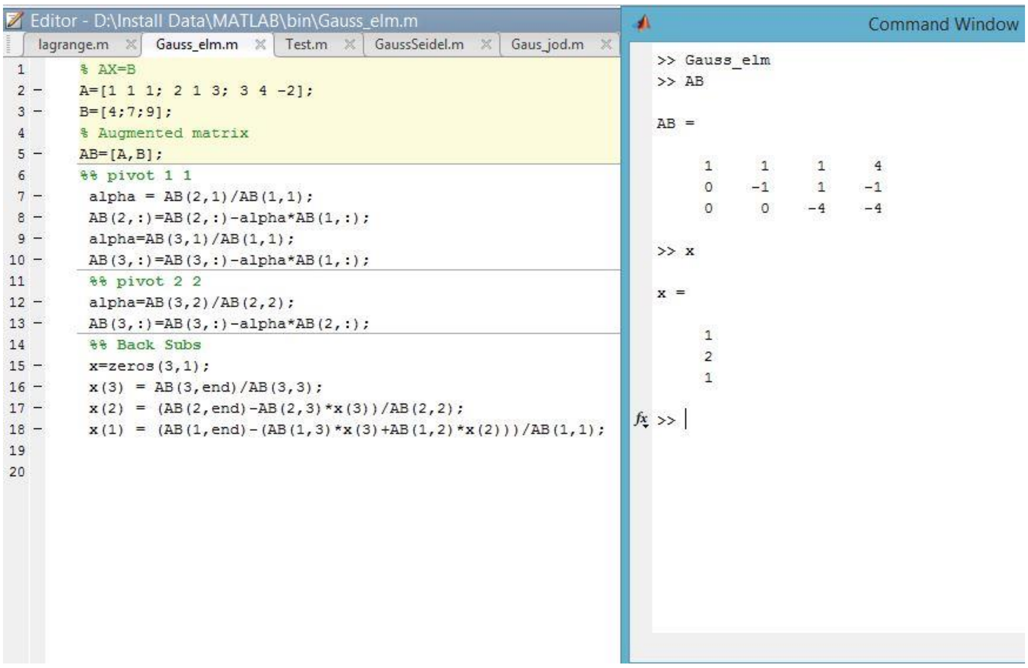
5. Stop

(II) MATLAB Code:

```
A=[1 1 1; 2 1 3; 3 4 -2];
B=[4;7;9];
% Augmented matrix
AB=[A,B];
```

```
%% pivot 1 1
alpha = AB(2,1)/AB(1,1);
AB(2,:)=AB(2,:)-alpha*AB(1,:);
alpha=AB(3,1)/AB(1,1);
AB(3,:)=AB(3,:)-alpha*AB(1,:);
%% pivot 2 2
alpha=AB(3,2)/AB(2,2);
AB(3,:)=AB(3,:)-alpha*AB(2,:);
%% Back Subs
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:



Result& Discussion: From the output the value of x matrix is 1, 2, 1.Means x=1, y=2, z=1.
Conclusion: The output is exactly the same as we learnt from the theory and it is an upper triangle.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 07

Name of the Experiment: Study of Gauss Jordan (GJ) Method to Find the Solution of Simultaneous Equations.

Objectives: The objective of this experiment is to apply GJ method to find out the very precise values of the equations, using MATLAB.

Theory: Solving of a system of linear algebraic equations appears frequently in many engineering problems. Most of numerical techniques which deals with partial differential equations, represent the governing equations of physical phenomena in the form of a system of linear algebraic equations. Gauss Jordan technique is a well-known numerical method which is employed in many scientific problems.
Consider an arbitrary system of linear algebraic equations as follows:

$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = c_1$

$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = c_2$

………………

$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = c_n$

Where x_i are unknowns and a_{ij} are coefficients of unknowns and c_i are equations' constants.
This system of algebraic equation can be written in the matrix form as follows:
[A]{x}={C}
Where [A] is the matrix of coefficient and {x} is the vector of unknowns and {C} is the vector of constants. Gauss Jordan method eliminate unknowns' coefficients of the equations one by one. Therefore the matrix of coefficients of the system of linear equations is transformed to an upper & lower triangular matrix. The last transformed equation has only one unknown which can be determined easily. This evaluated unknown can be used in the upper equation for determining the next unknown and so on. Finally the system of linear equations can be solved by back substitution of evaluated unknowns[1].
Tool: MATLAB Software

Methodology:

(I)Algorithm:

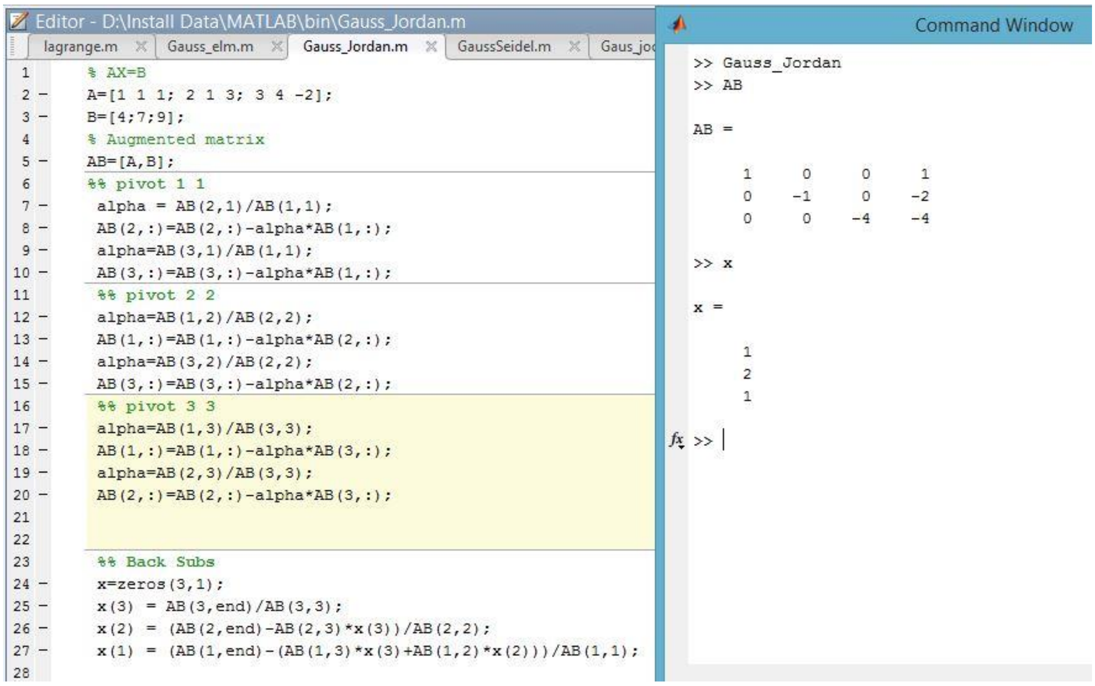
6. Start
7. Take the coefficients of the linear equation and pivot matrix 1 1 value then calculate.
8. Pivot matrix value 1 1 and make zero 2 1 and 3 1.
9. Pivot matrix value 2 2 and make zero 3 2 and 1 2.
10. Pivot matrix value 3 3 and make zero 2 3 and 1 3.
11. Stop

(II) MATLAB Code:

```
A=[1 1 1; 2 1 3; 3 4 -2];
B=[4;7;9];
% Augmented matrix
AB=[A,B];
%% pivot 1 1
 alpha = AB(2,1)/AB(1,1);
```

```
AB(2,:)=AB(2,:)-alpha*AB(1,:);
alpha=AB(3,1)/AB(1,1);
AB(3,:)=AB(3,:)-alpha*AB(1,:);
%% pivot 2 2
alpha=AB(1,2)/AB(2,2);
AB(1,:)=AB(1,:)-alpha*AB(2,:);
alpha=AB(3,2)/AB(2,2);
AB(3,:)=AB(3,:)-alpha*AB(2,:);
%% pivot 3 3
alpha=AB(1,3)/AB(3,3);
AB(1,:)=AB(1,:)-alpha*AB(3,:);
alpha=AB(2,3)/AB(3,3);
AB(2,:)=AB(2,:)-alpha*AB(3,:);
%% Back Subs
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:



Result& Discussion: From the output the value of x matrix is 1, 2, 1.Means x=1, y=2, z=1.
Conclusion: The output is exactly the same as we learnt from the theory and it is an upper triangle and also lower triangle or can be call it as diagonal matrix.

References:

[1]C. Chapra and P. Canale Raymond , “*Numerical Methods for Engineers”,* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 08

Name of the Experiment: Study of Gauss Seidel (GS) Method to Find the Solution of Simultaneous Equations.

Objectives: The objective of this experiment is to apply GS method to find out the very precise values of the equations, using MATLAB.

Theory: Although it seems that the Gauss elimination method gives an exact solution, the accuracy of this method is not very good in large systems. The main reason of inaccuracy in the gauss elimination method is round-off error because of huge mathematical operations of this technique. Furthermore, this method is very time consuming in large systems. Many of systems of linear algebraic equations which should be solved in engineering problems are large and there are lots of zeros in their coefficient matrix. To solve this kinds of problems, iterative methods often is used. Gauss-Seidel one of the iterative techniques, is very well-known because of its good performance in solving engineering problems. For a system of linear equation as follows:

$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = c_1$

$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = c_2$

………………………

$a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = c_n$

where x_i are unknowns and a_{ij} are coefficients of unknowns and c_i are equations' constants. The Gauss-Seidel method needs a starting point as the first guess. The new guess is determined by using the main equation as follows:

$$x_i = \frac{c_i - \sum_{j=1}^{n} a_{ij}x_j}{a_{ii}}, \quad i \neq j$$

Mathematically, it can be shown that if the coefficient matrix is diagonally dominant this method converges to exact solution[1].

Tool: MATLAB Software

Methodology:

(I)**Algorithm:**

12. Start
13. Take the coefficients of the linear equations.
14. Let x=0,y=0,z=0.
15. Put x,y,z in the functions and collect values in a,b,c. And again use x=a,y=b,z=c.
16. Stop

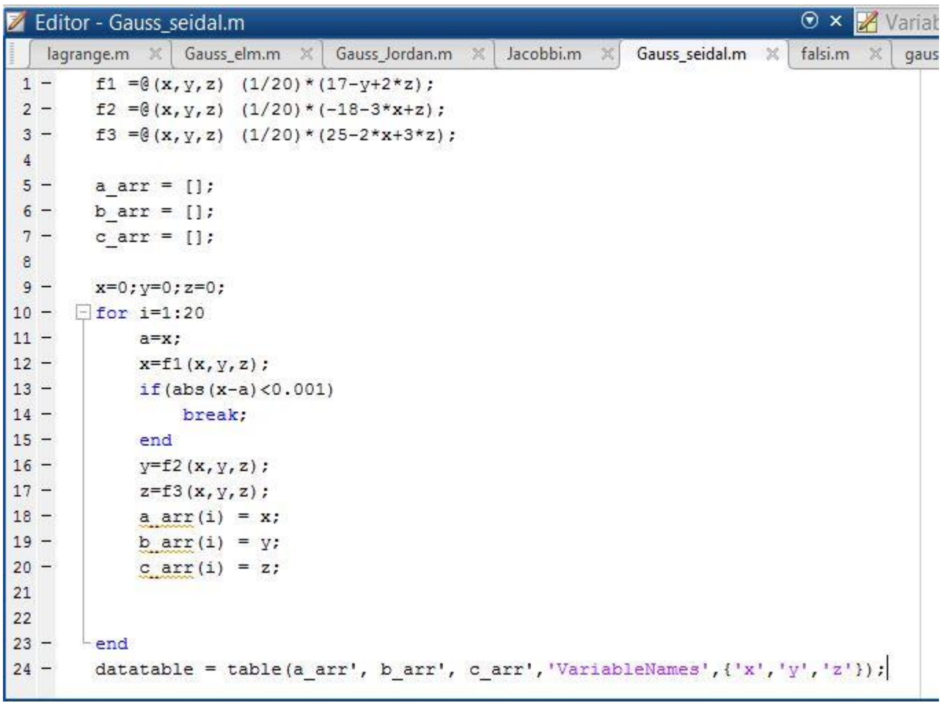
(II) MATLAB Code:

%declaring functions

```
f1 =@(x,y,z) (1/20)*(17-y+2*z);
```

```
f2 =@(x,y,z) (1/20)*(-18-3*x+z);
f3 =@(x,y,z) (1/20)*(25-2*x+3*z);

a_arr = [];
b_arr = [];
c_arr = [];

x=0;y=0;z=0;

%Iteration

for i=1:20
    a=x;
    x=f1(x,y,z);
    if(abs(x-a)<0.001)
        break;
    end
    y=f2(x,y,z);
    z=f3(x,y,z);
    a_arr(i) = x;
    b_arr(i) = y;
    c_arr(i) = z;


end
datatable = table(a_arr', b_arr', c_arr','VariableNames',{'x','y','z'});
```

Output:



MATLAB Variable: datatable
22-Feb-2020

	1 x	2 y	3 z
1	0.8500	-1.0275	1.1650
2	1.0179	-0.9944	1.3230
3	1.0320	-0.9887	1.3452
4	1.0340	-0.9878	1.3484

Figure 09: Data Table

Result& Discussion: The values of the given function is are 1.0430,-9878, 1.3484.Which is nearly close to the original values (1.043,-988, 1.34) direct calculated by calculator.
Conclusion: So from the above test we saw that nearly 3rd iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , “*Numerical Methods for Engineers”,* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 09

Name of the Experiment: Study of Jacobi Method to Find the Solution of Simultaneous Equations.

Objectives: The objective of this experiment is to apply Jacobi method to find out the very precise values of the equations, using MATLAB.

Theory: The Jacobi method is very similar to Gauss-Seidel method. The only difference is that Jacobi method doesn't use the latest evaluated x_i s in the equation. This method employs a set of old x_i s to determine a set of new x_i s. It means:

$$x_i = \frac{c_i - \sum_{j=1}^{n} a_{ij} x_j}{a_{ii}}, \qquad i \neq j$$

[1]

Tool: MATLAB Software

Methodology:

(I)Algorithm:

17. Start
18. Take the coefficients of the linear equations.
19. Let x=0,y=0,z=0.
20. Put x, y, z in the functions. And again use new x, y, z.
21. Stop

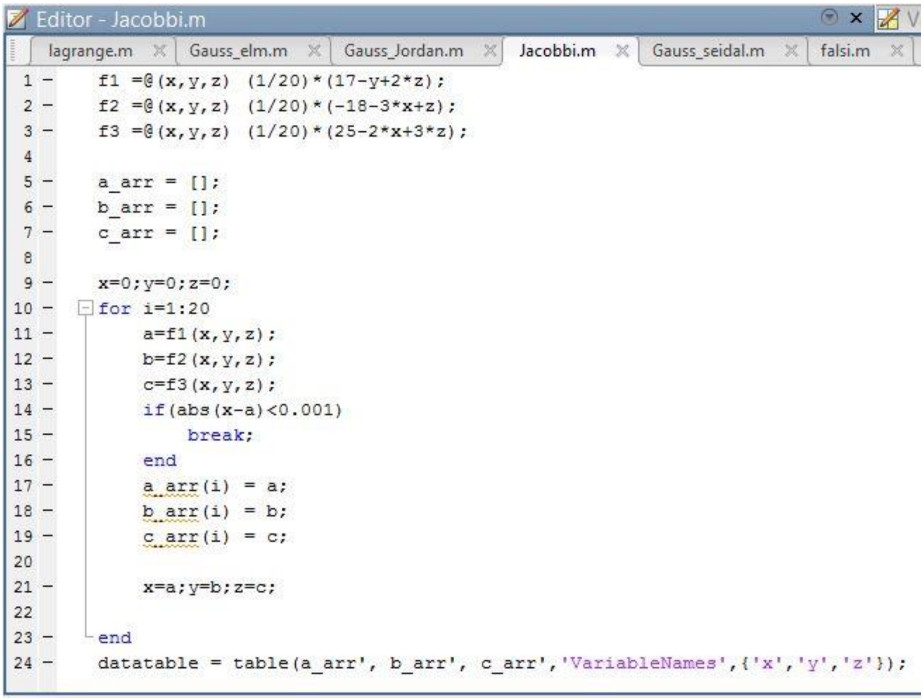
(II) MATLAB Code:

%declaring functions

```
f1 =@(x,y,z) (1/20)*(17-y+2*z);
f2 =@(x,y,z) (1/20)*(-18-3*x+z);
f3 =@(x,y,z) (1/20)*(25-2*x+3*z);

a_arr = [];
b_arr = [];
c_arr = [];

x=0;y=0;z=0;

%Iteration

for i=1:20
    a=f1(x,y,z);
    b=f2(x,y,z);
    c=f3(x,y,z);
    if(abs(x-a)<0.001)
        break;
    end
    a_arr(i) = a;
```

```
    b_arr(i) = b;
    c_arr(i) = c;

    x=a;y=b;z=c;

end
datatable = table(a_arr', b_arr', c_arr','VariableNames',{'x','y','z'});
```

Output:




```
f1 =@(x,y,z) (1/20)*(17-y+2*z);
f2 =@(x,y,z) (1/20)*(-18-3*x+z);
f3 =@(x,y,z) (1/20)*(25-2*x+3*z);

a_arr = [];
b_arr = [];
c_arr = [];

x=0;y=0;z=0;
for i=1:20
    a=f1(x,y,z);
    b=f2(x,y,z);
    c=f3(x,y,z);
    if(abs(x-a)<0.001)
        break;
    end
    a_arr(i) = a;
    b_arr(i) = b;
    c_arr(i) = c;

    x=a;y=b;z=c;

end
datatable = table(a_arr', b_arr', c_arr','VariableNames',{'x','y','z'});
```

MATLAB Variable: datatable
22-Feb-2020

	1 x	2 y	3 z
1	0.8500	-1.0275	1.1650
2	1.0179	-0.9944	1.3230
3	1.0320	-0.9887	1.3452
4	1.0340	-0.9878	1.3484

Figure 10: Data Table

Result& Discussion: The values of the given function is are 1.0430,-9878, 1.3484.Which is nearly close to the original values (1.043,-988, 1.34) direct calculated by calculator.
Conclusion: So from the above test we saw that nearly 4th iteration we get the resultant value of two roots which is very close to the original roots.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 10

Name of the Experiment: Study of Lagrange Interpolation Method to Predict the Unknown Value(s) For Any Geographic Point Data.

Objectives: The objective of this experiment is to apply Lagrange interpolation method to find out the unknown value(s) for a specific values(s) from a data table.

Theory: The Lagrange interpolating polynomial is the polynomial $P(x)$ of degree $\leq (n-1)$ that passes through the n points $(x_1, y_1 = f(x_1))$, $(x_2, y_2 = f(x_2))$, ..., $(x_n, y_n = f(x_n))$, and is given by [1]

$$P(x) = \sum_{j=1}^{n} P_j(x), \tag{1}$$

where

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}. \tag{2}$$

Written explicitly,

$$P(x) = \frac{(x-x_2)(x-x_3)\cdots(x-x_n)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_n)} y_1 + \frac{(x-x_1)(x-x_3)\cdots(x-x_n)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_n)} y_2 + \cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\cdots(x_n-x_{n-1})} y_n.$$

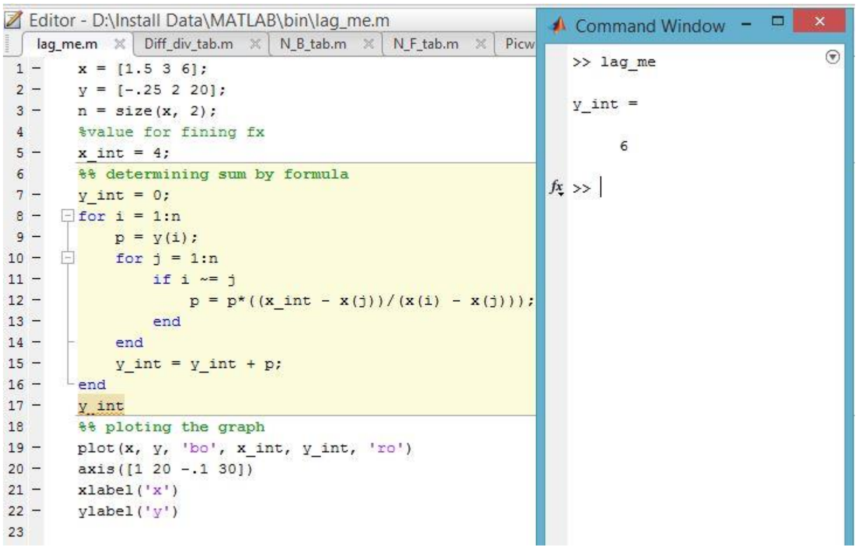
Tool: MATLAB Software
Methodology:

MATLAB Code:

```
x = [1.5 3 6];
y = [-.25 2 20];
n = size(x, 2);
%value for fining fx
x_int = 4;
%% determining sum by formula
y_int = 0;
for i = 1:n
    p = y(i);
    for j = 1:n
        if i ~= j
            p = p * ((x_int - x(j)) / (x(i) - x(j)));
        end
    end
    y_int = y_int + p;
end
y_int
%% ploting the graph
plot(x, y, 'bo', x_int, y_int, 'ro')
axis([1 20 -.1 30])
```

```
xlabel('x')
ylabel('y')
```

Output:



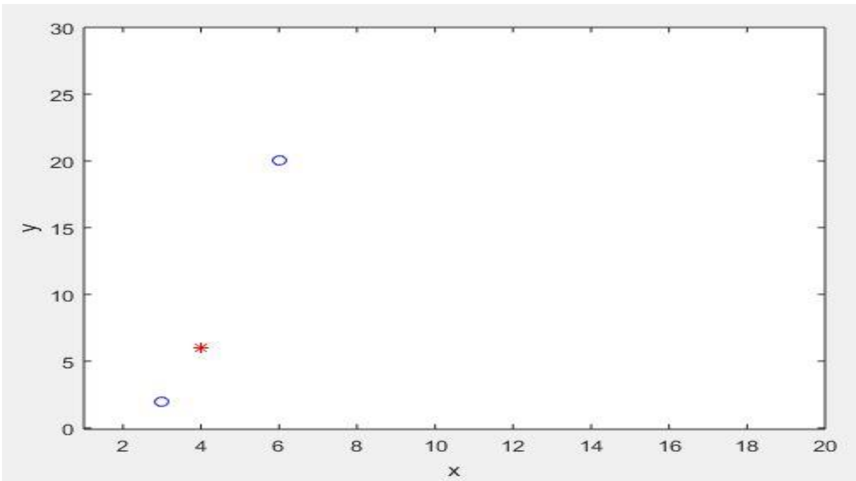


Figure 11.1: Graph Of The Function

Result(s)& Discussion: The unknown value for x = 4 is y = 6.From text book for x=4 is y=5.99 ~ 6

Conclusion: We have found the exact unknown value for 4 which is same as text book. MATLAB read 5.999 to round figure value 6.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 11

Name of the Experiment: Study of Divided Difference Method to Predict Unknown Value(s) For Any Geographic Point Data.

Objectives: The objective of this experiment is to use divided difference method to find out the very precise values of the given data point, using MATLAB.

Theory: x_i and x_j are any two tabular points, is independent of x_i and x_j. This ratio is called the first divided difference of $f(x)$ relative to x_i and x_j and is denoted by $f[x_i, x_j]$. That is

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

Since the ratio is independent of x_i and x_j we can write $f[x_0, x] = f[x_0, x_1]$

$$\frac{f(x) - f(x_0)}{(x - x_0)} = f[x_0, x_1]$$

$$f(x) = f(x_0) + (x - x_0)\, f[x_0, x_1]$$

$$= \frac{1}{x - x_0} \begin{vmatrix} f(x_0) & x_0 - x \\ f(x_1) & x_1 - x \end{vmatrix} = \frac{f_1 - f_0}{x_1 - x_0}\, x + \frac{f_0 x_1 - f_1 x_0}{x_1 - x_0}$$

So if $f(x)$ is approximated with a linear polynomial then the function value at any point x can be calculated by using $f(x) \cong P_1(x) = f(x_0) + (x - x_1)\, f[x_0, x_1]$

where $f[x_0, x_1]$ is the first divided difference of f relative to x_0 and x_1.

Similarly if $f(x)$ is a second degree polynomial then the secant slope defined above is not constant but a linear function of x. Hence we have

$$\frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

is independent of x_0, x_1 and x_2. This ratio is defined as second divided difference of f relative to x_0, x_1 and x_2. The secind divided difference are denoted as

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

Now again since $f[x_0, x_1, x_2]$ is independent of x_0, x_1 and x_2 we have

$$f[x_1, x_0, x] = f[x_0, x_1, x_2]$$

$$\frac{f[x_0, x] - f[x_1, x_0]}{x - x_1} = f[x_0, x_1, x_2]$$

$$f[x_0, x] = f[x_0, x_1] + (x - x_1)\, f[x_0, x_1, x_2]$$

$$\frac{f[x] - f[x_0]}{x - x_0} = f[x_0, x_1] + (x - x_1)\, f[x_0, x_1, x_2]$$

$$f(x) = f[x_0] + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x_0, x_1, x_2]$$

The $\mathbf{k}^{th}$ degree polynomial approximation to $\mathbf{f(x)}$ can be written as

$$\begin{aligned} f(x) = f[x_0] + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x_0, x_1, x_2] \\ + \ldots + (x - x_0)(x - x_1) \ldots (x - x_{k-1})\, f[x_0, x_1, \ldots, x_k]. \end{aligned}$$

This formula is called Newton's Divided Difference Formula.

Tool: MATLAB Software

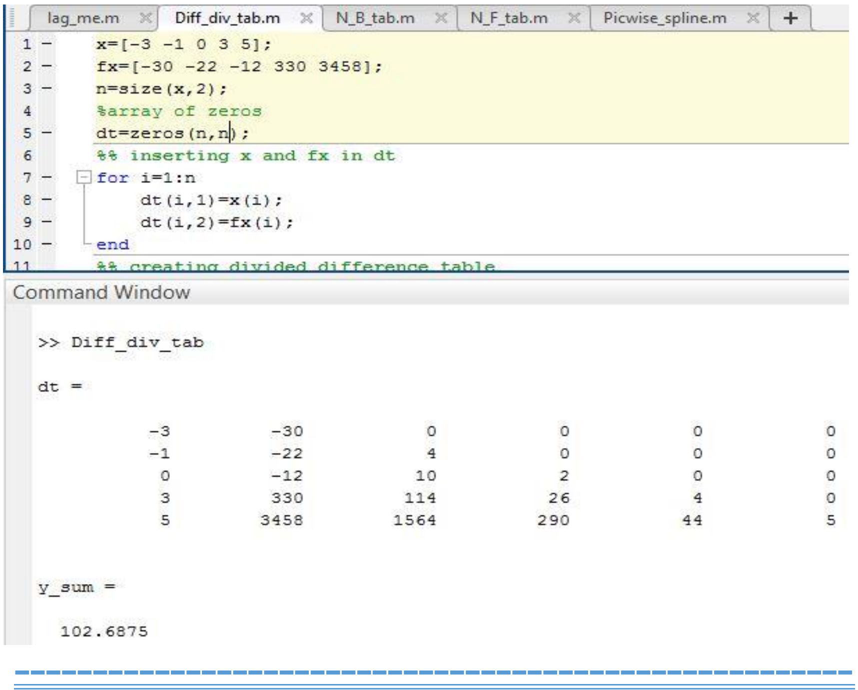
Methodology:

MATLAB CODE:

```
x=[-3 -1 0 3 5];
fx=[-30 -22 -12 330 3458];
n=size(x,2);
%array of zeros
dt=zeros(n+1,n+1);
%% inserting x and fx in dt
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
%% creating divided difference table
z=3;l=0;k=2;
for i=1:n-1
   for j=k:n
        dt(j,z)=(dt(i+1,z-1)-dt(i,z-1))/(dt(i+1,1)-dt(i-l,1));
        i=i+1;
        if(i>=n)
            break;
        end
   end
   k=k+1;l=l+1;z=z+1;
end
%value for fining fx
x_int=2.5;
%% determining sum by formula
y_sum=dt(1,2);
for i=2:n
    d=1;
    for j=1:i-1
        d=d*(x_int - x(j));
    end
    y_sum=y_sum+dt(i,i+1)*d;
end
%% result
dt
y_sum
%% ploting the graph
plot(x, fx, 'bo', x_int, y_sum, 'ro')
```

```
axis([-10 10 -1000 4000])
xlabel('x')
ylabel('y')
```

Output:



	1	2	3	4	5	6
1	-3	-30	0	0	0	0
2	-1	-22	4	0	0	0
3	0	-12	10	2	0	0
4	3	330	114	26	4	0
5	5	3458	1564	290	44	5

Figure 12.1: Table of Newton Divided Difference

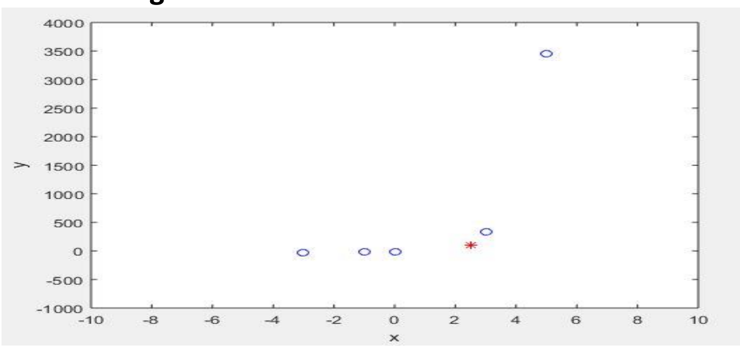


Figure 12.2: Graph Of The Function

Result(s)& Discussion: The unknown values for x = 2.5 is y = 102.6875 . From text book[1] for x=2.5 is y=102.7

Conclusion: We have found the approximate unknown value for 2.5 which is same as text book[1]. Matlab read the exact result 102.6875.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 12

Name of the Experiment: Study of Newton Forward Difference Method to Predict Unknown Value(s) for Any Geographic Point Data.

Objectives: The objective of this experiment is to use Newton Forward Difference method to find out the very precise values of the given data point, using MATLAB.

Theory:

Making use of forward difference operator and forward difference table (will be defined a little later) this scheme simplifies the calculations involved in the polynomial approximation of fuctons which are known at equally spaced data points.
Consider the equation of the linear interpolation optained in the earlier section :

$$f(x) \cong P_1(x) = a_{x-1}b = \frac{f_1 - f_0}{x_1 - x_0}x + \frac{f_0x_1 - f_1x_0}{x_1 - x_0}$$

$$= \frac{1}{(x_1 - x_0)}[(x_1 - x)f_0 + (x-x_0)f_1]$$

$$\frac{x_1 - x}{x_1 - x_0}f_0 + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0) + \frac{x - x_0}{x_1 - x_0}f_0$$

$$= f_0 + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0)$$

$$= f_0 + r\,\Delta f_0 \quad [\,r = (x - x_0) / (x_1 - x_0) \quad \Delta f_0 = f_1 - f_0\,]$$

since $x_1 - x_0$ is the step lenght **h**, **r** can be written as **$(x - x_0)/h$** and will be between **(0, 1).**

Tool: MATLAB Software

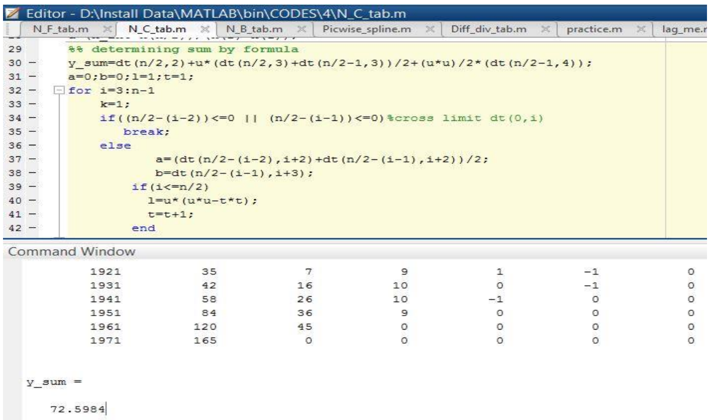
Methodology:

MATLAB Code:

```
x=[1921 1931 1941 1951 1961 1971];
fx=[35 42 58 84 120 165];
n=size(x,2);
%array of zeros
dt=zeros(n,n);
%% inserting x and fx in dt
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
dt
```

```
z=3;
for k=1:n-1
    i=1;
    for j=1:n-k
        dt(j,z)=(dt(i+1,z-1)-dt(i,z-1));
        i=i+1;
        if(j>n)
            break;
        end
    end
    z=z+1;k=k+1;
end
dt
%value for fx to find
x_int=1947;
%determining u=(x-x1)*h
u=(x_int-x(n/2))/(x(2)-x(1));
%% determining sum by formula
y_sum=dt(n/2,2)+u*(dt(n/2,3)+dt(n/2-1,3))/2+(u*u)/2*(dt(n/2-1,4));
a=0;b=0;l=1;t=1;
for i=3:n-1
    k=1;
    if((n/2-(i-2))<=0 || (n/2-(i-1))<=0)%cross limit dt(0,i)
       break;
    else
            a=(dt(n/2-(i-2),i+2)+dt(n/2-(i-1),i+2))/2;
            b=dt(n/2-(i-1),i+3);
        if(i<=n/2)
          l=u*(u*u-t*t);
          t=t+1;
        end
        for j=1:i
            k=k*j;
        end
          y_sum=y_sum+(l/k)*a+u*l*b;
    end
end
y_sum
%% ploting the graph
plot(x, fx, 'bo', x_int, y_sum, 'r*')
axis([1900 2000 0 250])
xlabel('x')
ylabel('y')
```

Output:



Editor - D:\Install Data\MATLAB\bin\CODES\4\N_C_tab.m

N_F_tab.m | N_C_tab.m | N_B_tab.m | Picwise_spline.m | Diff_div_tab.m | practice.m | lag_me.r

```
%% determining sum by formula
y_sum=dt(n/2,2)+u*(dt(n/2,3)+dt(n/2-1,3))/2+(u*u)/2*(dt(n/2-1,4));
a=0;b=0;l=1;t=1;
for i=3:n-1
    k=1;
    if((n/2-(i-2))<=0 || (n/2-(i-1))<=0)%cross limit dt(0,i)
       break;
    else
            a=(dt(n/2-(i-2),i+2)+dt(n/2-(i-1),i+2))/2;
            b=dt(n/2-(i-1),i+3);
        if(i<=n/2)
          l=u*(u*u-t*t);
          t=t+1;
        end
```

Command Window

```
        1921        35         7         9         1        -1         0
        1931        42        16        10         0        -1         0
        1941        58        26        10        -1         0         0
        1951        84        36         9         0         0         0
        1961       120        45         0         0         0         0
        1971       165         0         0         0         0         0


y_sum =

   72.5984
```

	1	2	3	4	5	6	7
1	1921	35	7	9	1	-1	0
2	1931	42	16	10	0	-1	0
3	1941	58	26	10	-1	0	0
4	1951	84	36	9	0	0	0
5	1961	120	45	0	0	0	0
6	1971	165	0	0	0	0	0

Figure 13.1: Table of Newton Forward Difference

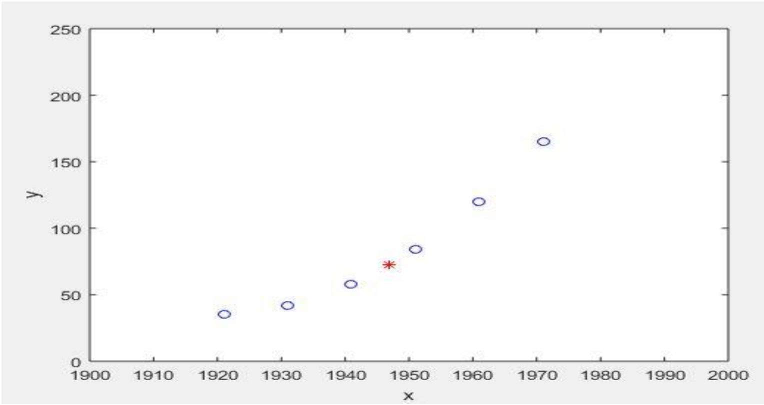


Figure 13.2: Graph Of The Function

Result(s) & Discussion: The unknown values for x = 1947 is y = 72.5984.
Conclusion: We have found the approximate unknown value for 1947 which is same as text book [1].

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 13

Name of the Experiment: Study of Newton Backward Difference Method to Predict Unknown Value(s) for Any Geographic Point Data.

Objectives: The objective of this experiment is to use Newton backward difference method to find out the very precise values of the given data point, using MATLAB.

Theory: The differences y1 – y0, y2 – y1, ……, yn – yn–1 when denoted by dy1, dy2, ……, dyn, respectively, are called first backward difference. Thus the first backward differences are:
$\nabla Y_r = Y_r - Y_{r-1}$

NEWTON'S GREGORY BACKWARD INTERPOLATION FORMULA:

$$f(a+nh+uh) = f(a+nh) + u\nabla f(a+nh) + \frac{u(u+1)}{2!}\nabla^2 f(a+nh) + \ldots + \frac{u(u+1)\ldots(u+n-1)}{n!}\nabla^n f(a+nh)$$

Tool: MATLAB Software

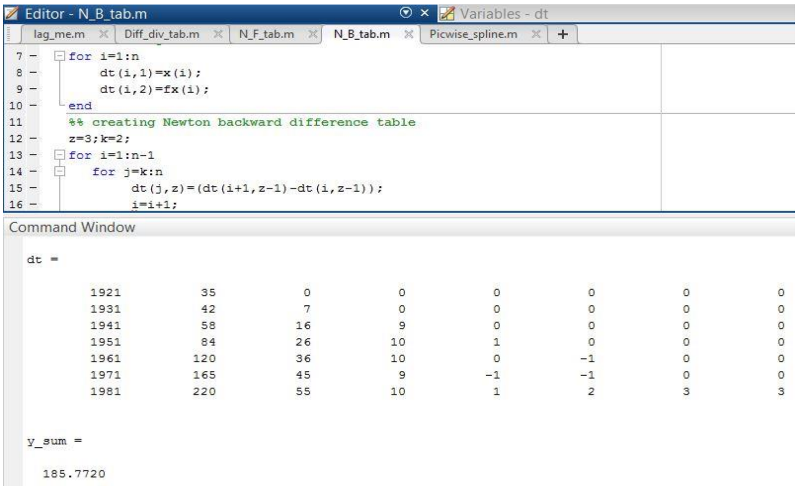
Methodology:

MATLAB Code:

```
x=[1921 1931 1941 1951 1961 1971 1981];
fx=[35 42 58 84 120 165 220];
n=size(x,2)
%array of zeros
dt=zeros(n,n)
%% inserting x and fx in dt
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
%% creating Newton backward difference table
z=3;k=2;
for i=1:n-1
   for j=k:n
        dt(j,z)=(dt(i+1,z-1)-dt(i,z-1));
        i=i+1;
        if(i>=n)
            break;
        end
   end
   k=k+1;z=z+1;
end
%value for fx to find
x_int=1975;
%determining u=(x-x1)*h
u=(x_int-x(n))/(x(2)-x(1));
%% determining sum by formula
y_sum=dt(n,2);k=1;d=1;
for i=2:4
    for j=0:i-2
        d=d*(u + j);
        k=k*(j+1);
    end
    y_sum=y_sum+(dt(n,i+1)/k)*d;
    d=1;
```

```
    dt(n,i+1);
end
%% result
dt
y_sum
%% ploting the graph
plot(x, fx, 'bo', x_int, y_sum, 'r*')
axis([1900 2000 0 250])
xlabel('x')
ylabel('y')
```

Output:



Editor - N_B_tab.m | Variables - dt

lag_me.m | Diff_div_tab.m | N_F_tab.m | N_B_tab.m | Picwise_spline.m

```
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
%% creating Newton backward difference table
z=3;k=2;
for i=1:n-1
  for j=k:n
        dt(j,z)=(dt(i+1,z-1)-dt(i,z-1));
        i=i+1;
```

Command Window

```
dt =

        1921      35       0       0       0       0       0       0
        1931      42       7       0       0       0       0       0
        1941      58      16       9       0       0       0       0
        1951      84      26      10       1       0       0       0
        1961     120      36      10       0      -1       0       0
        1971     165      45       9      -1      -1       0       0
        1981     220      55      10       1       2       3       3


y_sum =

  185.7720
```

	1	2	3	4	5	6	7	8
1	1921	35	0	0	0	0	0	0
2	1931	42	7	0	0	0	0	0
3	1941	58	16	9	0	0	0	0
4	1951	84	26	10	1	0	0	0
5	1961	120	36	10	0	-1	0	0
6	1971	165	45	9	-1	-1	0	0
7	1981	220	55	10	1	2	3	3

Figure 14.1: Table of Newton Backward Difference

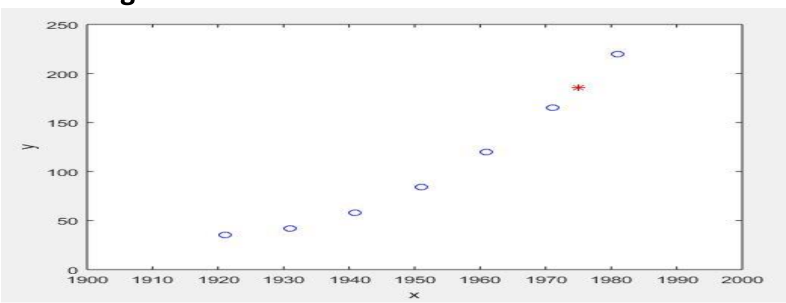


Figure 14.2: Graph Of The Function

Result(s)& Discussion: The unknown values for x = 1975 is y = 185.7720 . From text book[1] for x=1975 is y=185.8=186(round)

Conclusion: We have found the approximate unknown value for 1975 which is same as text book[1]. Matlab read the exact result 185.7720.

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 14

Name of the Experiment: Study Of Piecewise Linear Fit Interpolation Method To Predict Unknown Value(s) For Any Geographic Point Data.

Objectives: The objective of this experiment is to use piecewise linear fit interpolation method to find out the very precise values of the given data point, using MATLAB.

Theory: The interpolating polynomials which have been seen to this point have been defined on for all the *n* points (x_1, y_1), (x_2, y_2), ..., (x_n, y_n). An alternative approach is to define a different interpolating polynomial on each sub-interval under the assumption that the *x* values are given in order.

The simplest means is to take each pair of adjacent points and find an interpolating polynomial between the points which using Newton polynomials is

This can be expanded to reduce the number of required operations by reducing it to a form $ax + b$ which can be computed immediately. The reader may note that if the value $x = x_{k+1}$ is substituted into the above equation that the value is y_{k+1}.

A significant issue with piecewise linear interpolation is that the interpolant is not differentiable or *smooth*. A non-differentiable function can introduce new issues in a system almost as easily as a non-continuous function.

Given a set of *n* points (x_1, y_1), (x_2, y_2), ... (x_n, y_n) where $x_1 < x_2 < \cdots < x_n$, a piecewise linear function is defined for a point *x* such that $x_k \le x \le x_{k+1}$.

Using the Piecewise Linear Fit Interpolation Method formula we can easily calculate the aspire value for a particular point. Where x_int is the given value for which we have to find f(x).

F(x)=(y(i+1)*(x_int - x(i))-y(i)*(x_int-x(i+1)))/(x(i+1)-x(i))

Tool: MATLAB Software

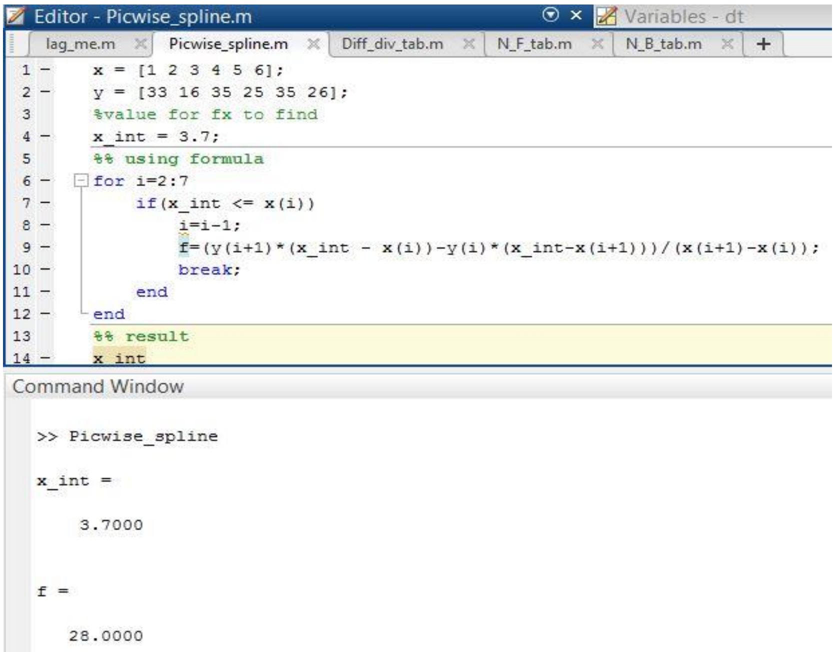
Methodology:

MATLAB Code:

```
x = [1 2 3 4 5 6];
y = [33 16 35 25 35 26];
%value for fx to find
x_int = 3.7;
%% using formula
for i=2:7
    if(x_int <= x(i))
        i=i-1;
        f=(y(i+1)*(x_int - x(i))-y(i)*(x_int-x(i+1)))/(x(i+1)-x(i));
        break;
    end
end
%% result
```

```
x_int
f
%% ploting the graph
hold on
plot(x, y, x_int, f,'ro')
axis([0 10 10 50])
xlabel('x')
ylabel('y')
```

Output:



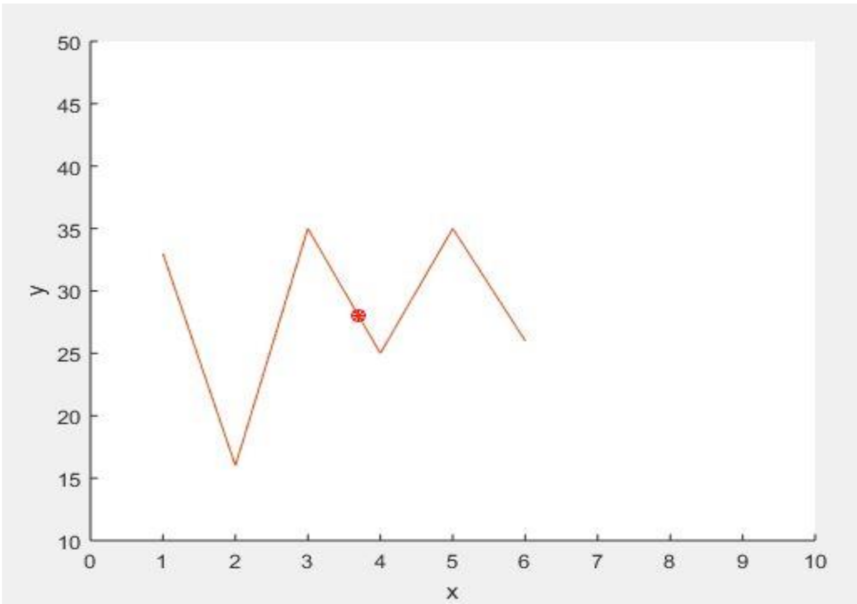


Figure 15.1: Graph of the Function

Result(s)& Discussion: The unknown values for x = 3.7 is y = 28 . From text book[1] for x=3.7 is y=28

Conclusion: We have found the exact unknown value for 3.7 which is same as text book[1].

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 15

Name of the Experiment: Study of Trapezoidal Integral Method to Calculate Integral Value of a Function with Limit.

Objectives: The objective of this experiment is to use Trapezoidal Integral Method to calculate integral value of any limited function, using MATLAB.

Theory: a The Trapezoidal Rule for approximating $_b\int_a f(x)dx$ is given by

$$_b\int_a f(x)dx \approx T_n = \Delta x2[f(x_0)+2f(x_1)+2f(x_2)+\cdots+2f(x_{n-1})+f(x_n)],$$

where $\Delta x = b-an$ and $x_i = a+i\Delta x$.

As $n\rightarrow\infty$, the right-hand side of the expression approaches the definite integral $_b\int_a f(x)dx$ [1].

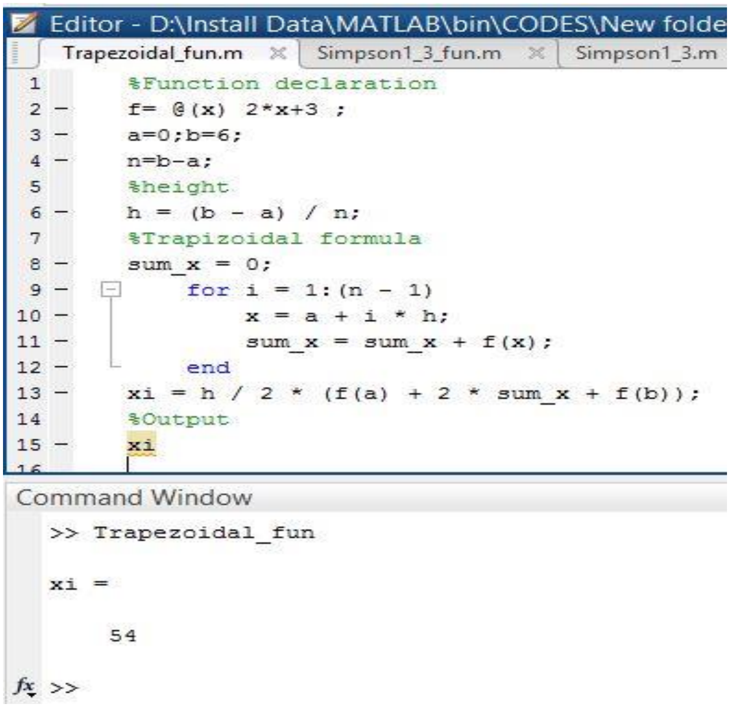
Tool: MATLAB Software

Methodology:

MATLAB Code:

```
%Function declaration
f= @(x) 2*x+3 ;
a=0;b=6;
n=b-a;
%height
h = (b - a) / n;
%Trapizoidal formula
sum_x = 0;
    for i = 1:(n - 1)
        x = a + i * h;
        sum_x = sum_x + f(x);
    end
xi = h / 2 * (f(a) + 2 * sum_x + f(b));
%Output
xi
```

Output:



Result(s)& Discussion: The integral value is 54.

Conclusion: We have found the exact integral value of function 2x+3 from limit 0 to 6 which is same as the calculated value ($\int_0^6$ **2x+3 dx**).

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 16

Name of the Experiment: Study of Simpson's 1/3 Integral Method to Calculate Integral Value of a Function with Limit.

Objectives: The objective of this experiment is to use Simpson's 1/3 Integral Method to calculate integral value of any limited function, using MATLAB.

Theory: If the interval [a,b] is split up into n subintervals, and n is an even number, the composite Simpson's rule is calculated with the following formula:

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

where $x_j = a+jh$ for $j = 0,1,\ldots,n-1,n$ with $h=(b-a)/n$; in particular, $x_0 = a$ and $x_n = b$.

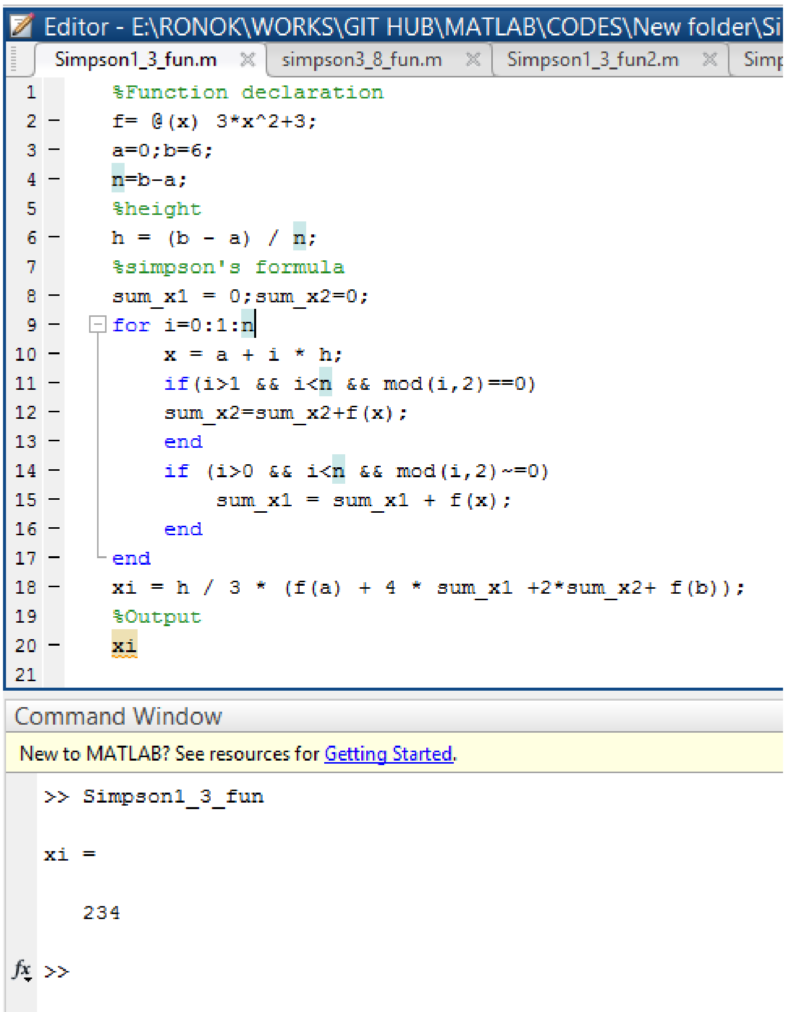
Tool: MATLAB Software

Methodology:

MATLAB Code:

```
%Function declaration

f= @(x) 3*x^2+3;

a=0;b=6;
n=b-a;
%height
h = (b - a) / n;
%simpson's formula
sum_x1 = 0;sum_x2=0;
for i=0:1:n
    x = a + i * h;
    if(i>1 && i<n && mod(i,2)==0)
    sum_x2=sum_x2+f(x);
    end
    if (i>0 && i<n && mod(i,2)~=0)
        sum_x1 = sum_x1 + f(x);
    end
end
xi = h / 3 * (f(a) + 4 * sum_x1 +2*sum_x2+ f(b));
%Output
xi
```

Output:



Result(s)& Discussion: The integral value is 234.

Conclusion: We have found the exact integral value of function $\mathbf{3x^2 +3}$ from limit 0 to 6 which is same as the calculated value ($\int_0^6 \mathbf{3x^2 +3\, dx}$).

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers*", 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 17

Name of the Experiment: Study of Simpson’s 3/8 Integral Method to Calculate Integral Value of a Function with Limit.

Objectives: The objective of this experiment is to use Simpson’s 3/8 Integral Method to calculate integral value of any limited function, using MATLAB.

Theory: If the interval [a,b] is split up into n subintervals, and n is an even number, the composite Simpson's rule is calculated with the following formula:

$$\int_a^b f(x)\,dx = 3h/8[(y_0+y_n)+3(y_1+y_2+y_4+y_5+\ldots.+y_{n-1})+2(y_3+y_6+y_9+\ldots..+y_{n-3})]$$

Where $\mathbf{x_j = a+jh}$

Tool: MATLAB Software

Methodology:

MATLAB Code:

```
%Function declaration
f= @(x) 3*x^2+3;
a=0;b=6;
n=b-a;
%height
h = (b - a) / n;
%simpson's formula
sum_x1 = 0;sum_x2=0;
for i=0:1:n
    x = a + i * h;
    if(i>1 && i<n && mod(i,3)==0)
    sum_x2=sum_x2+f(x);
    end
    if (i>0 && i<n && mod(i,3)~=0)
        sum_x1 = sum_x1 + f(x);
    end
end
xi = 3*h / 8 * (f(a) + 3 * sum_x1 +2*sum_x2+ f(b));
%Output
xi
```

Output:

Editor - D:\Install Data\MatLab\bin\simpson3_8_fun.m

simpson3_8_fun.m | Simpson1_3_fun2.m | Simpson3_8.m | Simpson1_

```
%Function declaration
f= @(x) 3*x^2+3;
a=0;b=6;
n=b-a;
%height
h = (b - a) / n;
%simpson's formula
sum_x1 = 0;sum_x2=0;
for i=0:1:n
    x = a + i * h;
    if(i>1 && i<n && mod(i,3)==0)
    sum_x2=sum_x2+f(x);
    end
    if (i>0 && i<n && mod(i,3)~=0)
        sum_x1 = sum_x1 + f(x);
    end
end
xi = 3*h / 8 * (f(a) + 3 * sum_x1 +2*sum_x2+ f(b));
%Output
xi
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> simpson3_8_fun

xi =

   234

>>
```

Result(s)& Discussion: The integration value is 234.

Conclusion: We have found the exact integral value of function **$3x^2$ +3** from limit 0 to 6 which is same as the calculated value ($\int_0^6$**$3x^2$ +3 dx**).

References:

[1]C. Chapra and P. Canale Raymond , "*Numerical Methods for Engineers",* 7th ed. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121, 2015

Experiment No: 18

Name of the Experiment: Study of Euler's Method to Solve Ordinary Differential Equation(s) (Initial Value Problem)

Objectives: The objective of this experiment is to use to Solve Ordinary Differential Equation(s) with Initial Value, using MATLAB.

Theory: This method uses the simplest extrapolation techniques for developing a solution. Equations (9.1) and (9.2) are written below for ready reference.

$$\frac{dy}{dx} = \mathrm{f}(\mathrm{x}, \mathrm{y}) \quad (1)$$

$$y = y1 \; at \; x = x1 \quad (2)$$

Given ($x1$, $y1$) the slope at this point is obtained as

$$\frac{dy}{dx}(\mathrm{x1}, \mathrm{y1}) = \mathrm{f}(\mathrm{x1}, \mathrm{y1}) \quad (3)$$

The next point $y2$ on the solution curve may be extrapolated by taking a small step in a direction given by the above slope. Thus

$$\mathrm{y}(\mathrm{x1} + \mathrm{h}) = \mathrm{y2} = \mathrm{y1} + \mathrm{hf}(\mathrm{x1}, \mathrm{y1}) \quad (4)$$

Tool: MATLAB Software

Methodology:

Problem: The temperature radiation of a ball in air at ambient temperature 300K can be describe by the differential equation

$$\frac{d\emptyset}{dx} = -2.2067 \times 10\text{-}12(\emptyset 4 - 81 \times 108)$$

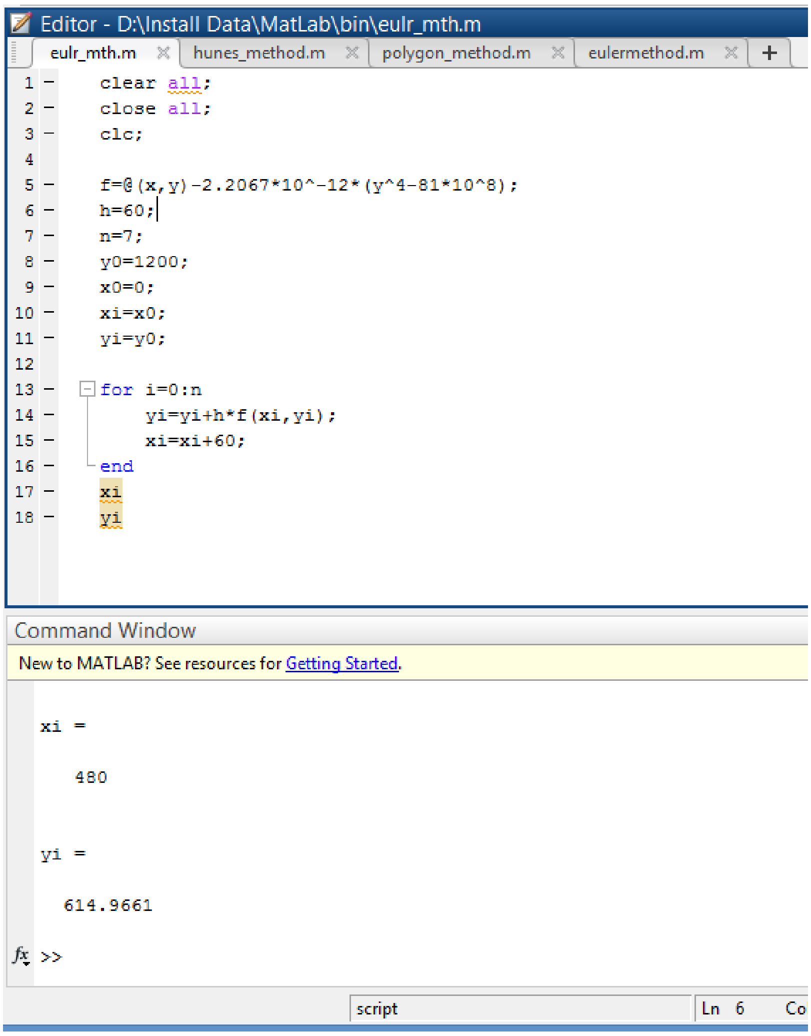
Using Euler's method find the temperature of the ball at $t = 480$ seconds where Ø is in K and t in second. It is assumed that the initial temperature of the ball is 1200K.

MATLAB Code:

```
clear all;
close all;
clc;

f=@(x,y)-2.2067*10^-12*(y^4-81*10^8);
h=60;
n=7;
y0=1200;
x0=0;
xi=x0;
yi=y0;

for i=0:n
    yi=yi+h*f(xi,yi);
```

```
    xi=xi+60;
end
xi
yi
```

Output:



Result(s)& Discussion: The result is Ø(480) = Ø8= 614.9661K

Conclusion: The result is not the exact value as we find from polygon method. There are some error.

References:

[1] PDF provided by Prof. Dr. Md. Shamim Anower

Experiment No: 19

Name of the Experiment: Study of Heun's Method to Solve Ordinary Differential Equation(s) (Initial Value Problem)

Objectives: The objective of this experiment is to use Heun's Method to Solve Ordinary Differential Equation(s) with Initial Value, using MATLAB.

Theory:

Consider the following geometric method of extrapolating the *y(x)* curve to obtain the solution to the differential equation

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_1) = y_1$$

Draw a straight line from (x_1, y_1) with a slope $s_1 = f(x_1, y_1)$. Let it cut the vertical line through $x_1 + h$ at $(x_1 + h, y2.$ [See Figure 9.2.] Determine the slope *dy/dx* of the solution curve *y(x)* at *(x1+h, y2)*. This is given by *s2 =fx2 ,y2, (x2=x1+h)*.

Now draw a straight line from (x_1, y_1) with a slope $(s_1 + s_2)/2$. The point y_2 where this straight line cuts the vertical line at $x_1 + h$ is the approximate solution of the differential equation at $x_1 + h$. Thus we have:

$$\begin{aligned} y_2 &= y_1 + h\frac{(s_1 + s_2)}{2} \\ &= y_1 + \tfrac{h}{2}[f(x_1, y_1) + f(x_1 + h, y_1 + s_1 h)] \end{aligned} \qquad (6)$$

In general the $(i + 1)$th point is obtained from the ith point using the formula

$$y_{i+1} = y_i + h\frac{(s_i + s_{i+1})}{2} \qquad (7)$$

Where $s_i = f(x_i, y_i)$ and $s_{i+1} = f(x_{i+1}, y_i + s_i h)$.

Tool: MATLAB Software

Methodology:

Problem: The temperature radiation of a ball in air at ambient temperature 300K can be describe by the differential equation

$$\frac{d\emptyset}{dx} = -2.2067 \times 10\text{-}12(\emptyset 4 - 81 \times 108)$$

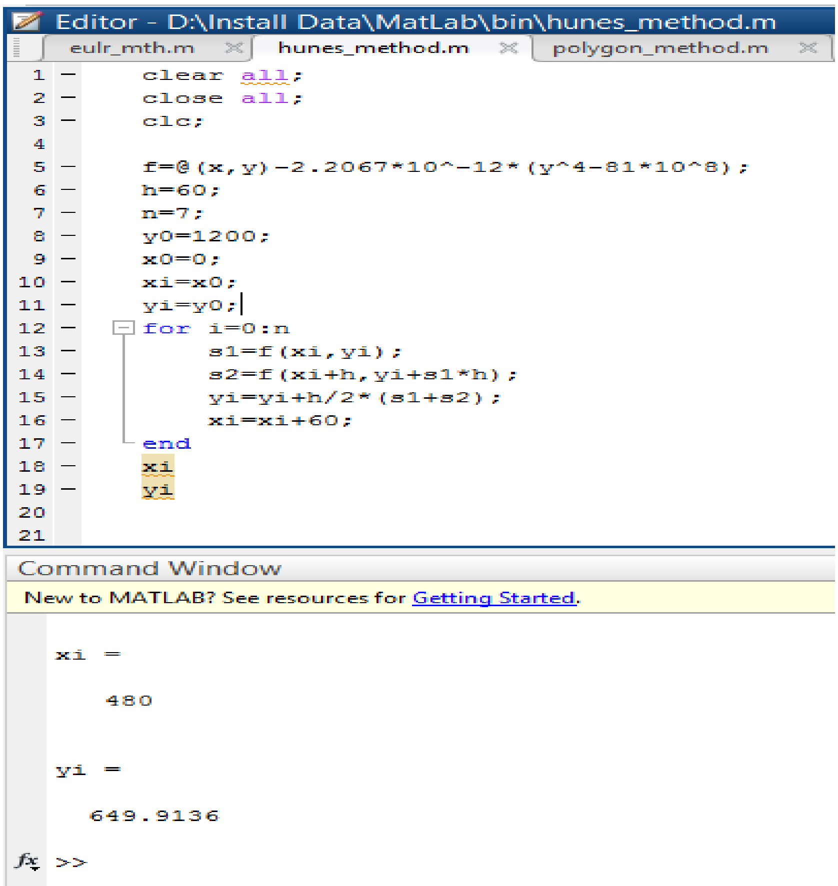
Using Euler's method find the temperature of the ball at t = **480** seconds where Ø is in K and t in second. It is assumed that the initial temperature of the ball is 1200K.

MATLAB Code:

```
clear all;
close all;
clc;
```

```
f=@(x,y)-2.2067*10^-12*(y^4-81*10^8);
h=60;
n=7;
y0=1200;
x0=0;
xi=x0;
yi=y0;
for i=0:n
    s1=f(xi,yi);
    s2=f(xi+h,yi+s1*h);
    yi=yi+h/2*(s1+s2);
    xi=xi+60;
end
xi
yi
```

Output:



Result(s)& Discussion: The result is Ø(480) = Ø8= 649.9136K

Conclusion: The result is not the exact value as we find from polygon method but near that value. There are some error.

References:

[1] PDF provided by Prof. Dr. Md. Shamim Anower

Experiment No: 20

Name of the Experiment: Study of Polygon Method to Solve Ordinary Differential Equation(s) (Initial Value Problem)

Objectives: The objective of this experiment is to use Polygon method Method to Solve Ordinary Differential Equation(s) with Initial Value, using MATLAB.

Theory:

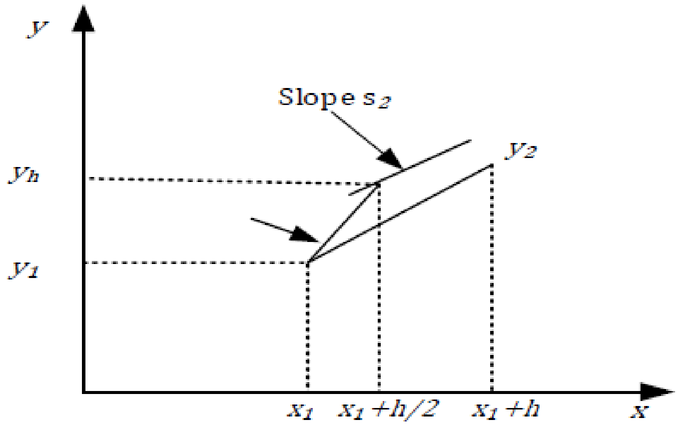


Figure 3: Illustration of Polygon method.

Starting from (x_1, y_1) draw a straight line with slope $s_1 = f(x_1, y_1)$. Find the value of y where this line cuts the vertical line erected at $x_1 + h/2$. Call it y_h. Calculate $f(x_1 + \frac{h}{2}, y_h)$ which the slope of the solution curve is at this point. Call it s_2. Go back to (x_1, y_1) and draw a straight line with slope s_2. This cuts the vertical line erected at $x_1 + h$ at y_2. This is taken as the approximate solution of the differential equation at $(x_1 + h)$. In general one would proceed from the ith point to the $(i+1)$th point in the algorithm using Equation (8)

$$y_{i+1} = y_i + hf\left(x_i + \frac{h}{2}, y_i + s_i\frac{h}{2}\right) \qquad (8)$$

Where

$$s_i = f(x_i, y_i)$$

Tool: MATLAB Software

Methodology:

Problem: The temperature radiation of a ball in air at ambient temperature 300K can be describe by the differential equation

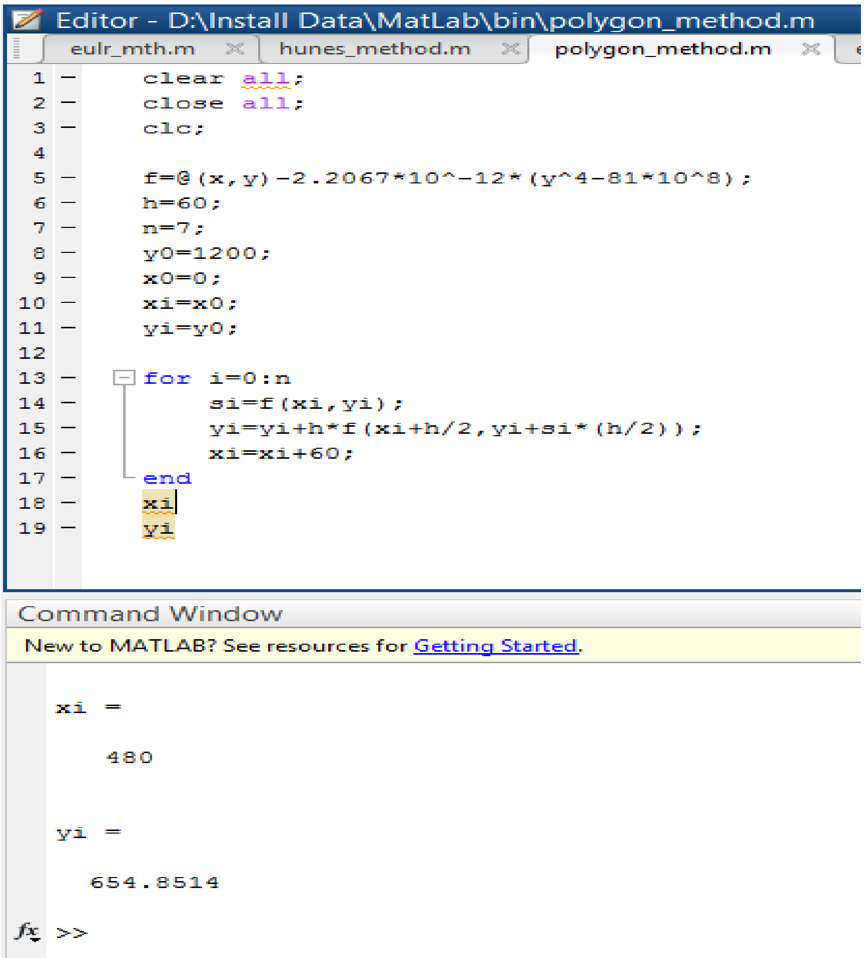
$$\frac{d\emptyset}{dx} = -\mathbf{2.2067 \times 10\text{-}12(\emptyset 4 - 81 \times 108)}$$

Using Euler's method find the temperature of the ball at $\boldsymbol{t} = \mathbf{480}$ seconds where Ø is in K and t in second. It is assumed that the initial temperature of the ball is 1200K.

MATLAB Code:

```
clear all;
close all;
clc;

f=@(x,y)-2.2067*10^-12*(y^4-81*10^8);
h=60;
n=7;
y0=1200;
x0=0;
xi=x0;
yi=y0;

for i=0:n
    si=f(xi,yi);
    yi=yi+h*f(xi+h/2,yi+si*(h/2));
    xi=xi+60;
end
xi
yi
```

Output:



Result(s) & Discussion: The result is Ø (480) = Ø8= 654.8514K

Conclusion: The result is very close to the exact value. There is very less error.

References:

[1] PDF provided by Prof. Dr. Md. Shamim Anower