

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY



DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

COURSE NO: ECE - 2214

COURSE TITLE: NUMERICAL TECHNIQUES SESSIONAL

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

Experiment No: 02

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

Objectives:

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

Theory:

The *bisection method*, which is alternatively called binary chopping, interval halving, or Bolzano's method, is one type of incremental search method in which the interval is always divided in half. If a function changes sign over an interval, the function value at the midpoint is evaluated. The location of the root is then determined as lying at the midpoint of the subinterval

within which the sign change occurs. The process is repeated to obtain refined estimates.

[1]

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 $f(c) = 0$. [2]

The bisection 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

$c = (a+b)/2$ [3]

Tool: MATLAB

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 $f(a)f(b) < 0$.

Step 2 : 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 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:

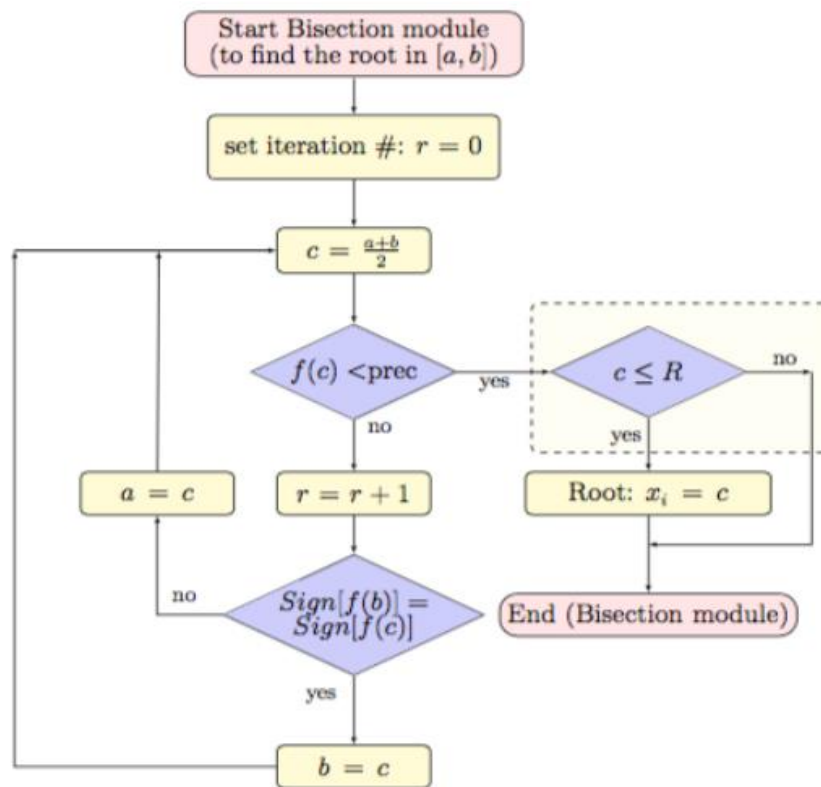


Figure 2.1 Flowchart of bisection method procedure [4]

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

```

a=input('Enter the value of 1st assumption:'); //calling value from user
b=input('Enter the value of 2nd assumption:');
y= @(x) 2*x^2-15*x+3 ; //declaring function
if y(a)*y(b)>0 //checking assumption
    fprintf('WRONG!!');
    return;
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+b)/2;
    if y(a)*y(c)>0 //checking signs
        a=c;
    else b=c;
    end
    if abs(y(a))<.001
        break;
        fprintf('%d',i);
    end
end
  
```

```

end
fprintf('%d %f %f %f %f \n',i,a,b,c,y(c)); //printing iteration
values
end

```

Output:

Editor - C:\Users\ronokarya\Documents\MATLAB\Bisection\Untitled.m

```

1 - a=input('Enter the value of 1st assumption:');
2 - b=input('Enter the value of 2nd assumption:');
3 - y= @(x) 2*x^2-15*x+3 ;
4
5 - if y(a)*y(b)>0
6 -     fprintf('WRONG!!');
7 -     return;
8 - end
9 - if y(a)==0
10 -     fprintf('Root')

```

Command Window

```

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
13 0.205566  0.205688  0.205688  -0.000712

```

Editor - C:\Users\ronokarya\Documents\MATLAB\Bisection\Untitled.m

```

1 - a=input('Enter the value of 1st assumption:');
2 - b=input('Enter the value of 2nd assumption:');
3 - y= @(x) 2*x^2-15*x+3 ;
4
5 - if y(a)*y(b)>0
6 -     fprintf('WRONG!!');
7 -     return;
8 - end

```

Command Window

```

Enter the value of 1st assumption:8
Enter the value of 2nd assumption:4
No.    a          b          c          y
-----
1  8.000000  6.000000  6.000000  -15.000000
2  8.000000  7.000000  7.000000  -4.000000
3  7.500000  7.000000  7.500000  3.000000
4  7.500000  7.250000  7.250000  -0.625000
5  7.375000  7.250000  7.375000  1.156250
6  7.312500  7.250000  7.312500  0.257813
7  7.312500  7.281250  7.281250  -0.185547
8  7.296875  7.281250  7.296875  0.035645
9  7.296875  7.289063  7.289063  -0.075073
10 7.296875  7.292969  7.292969  -0.019745
11 7.294922  7.292969  7.294922  0.007942
12 7.294922  7.293945  7.293945  -0.005903
13 7.294434  7.293945  7.294434  0.001019
14 7.294434  7.294189  7.294189  -0.002442
15 7.294434  7.294312  7.294312  -0.000712

```

Results & Discussion:

The roots of the given function is 7.294312 and 0.205688. Which is nearly close to the original value (7.294361 & 0.205638) direct calculated by calculator.

Precaution:

1. Be careful with 2x because in MATLAB the syntax is 2*x.
2. Variables are declared properly.
3. The accuracy was kept small to reduce iteration.

Conclusion:

So from the above test we saw that nearly 15th 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]http://pages.cs.wisc.edu/~sifakis/courses/cs412s13/lecture_notes/CS412_5_Feb_2013.pdf
- [3]http://pages.cs.wisc.edu/~sifakis/courses/cs412s13/lecture_notes/CS412_5_Feb_2013.pdf
- [4] C. DENİZ Adnan Menderes University, “*A FAST BISECTION BASED ANALYZER DESIGN FOR THE DETERMINATION OF MODES IN CIRCULAR WAVEGUIDES*” ,page 108, Aytepe-09010, Aydin, TURKEY, Geliş/Received: 06.04.2017; Kabul/Accepted in Revised Form: 21.07.2017

Experiment No: 03

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 $f(a)f(b) < 0$.

Step 2 : 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:

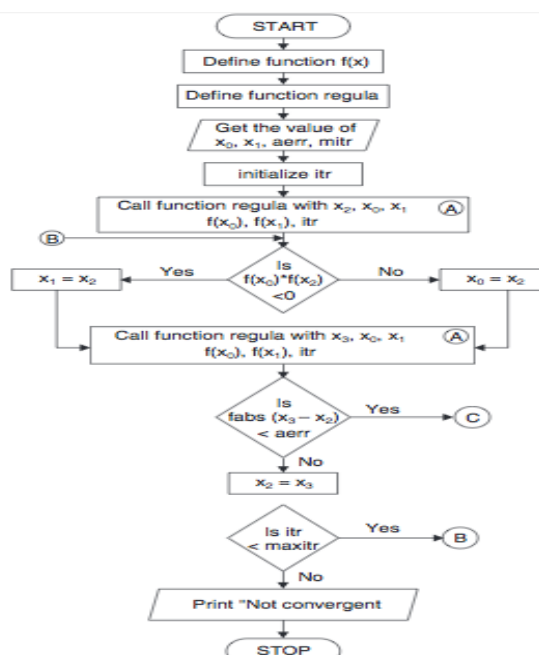


Figure 2.1 Flowchart of bisection method procedure [2]

(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:

The screenshot shows the MATLAB Editor with the file 'falsi.m' open. The code is as follows:

```

14 - fprintf('Root')
15 - return
16 - elseif y(b)==0;
17 - fprintf('Root')
18 - return
19 - end
20 - display(' No.      a      b      c      y')
21 - display('-----')
22 -
23 - for i=1:1:100
24 -     c=(a*y(b)-b*y(a))/(y(b)-y(a));
25 -     if y(a)*y(c)>0
26 -         a=c;
27 -     else b=c;
28 -     end

```

The Command Window shows the execution of the 'falsi' function. It prompts for the 1st and 2nd assumptions, which are 0 and 1 respectively. It then displays a table of results for the first three iterations:

No.	a	b	c	y
1	0.000000	0.230769	0.230769	-0.355030
2	0.000000	0.206349	0.206349	-0.010078
3	0.000000	0.205658	0.205658	-0.000284

The Command Window prompt is currently at 'fx >> |'.

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: 04

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 $x=x_0$. Then Newton's method tells us that a better approximation for the root is $x_1=x_0-f'(x_0)/f(x_0)$.

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

$$x_{n+1}=x_n-f'(x_n)/f(x_n)$$

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:

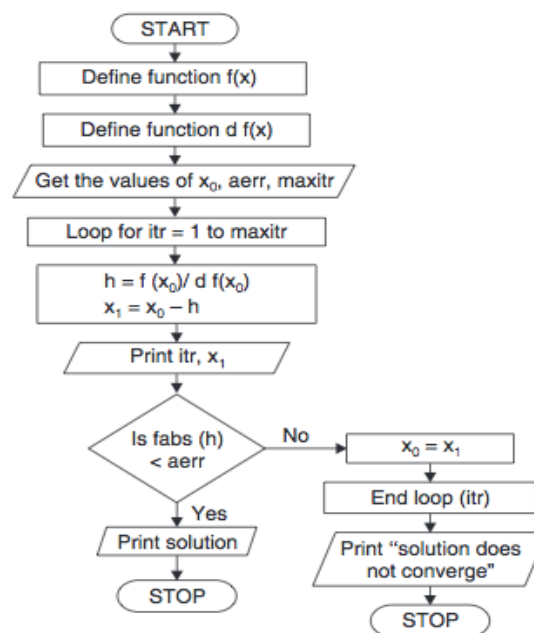


Figure 2.1 Flowchart of Newton-Raphson method procedure [2]

(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:

The image shows a MATLAB environment with an Editor window and a Command Window. The Editor window displays a script named N_R2.m. The script defines a function f, its derivative f1, and uses the Newton-Raphson method to find roots. The Command Window shows the execution of the script, including the input of the function and the initial value, and the resulting table of iterations.

```

6 - f=inline(fun);
7 - z=diff(f(x));
8 - f1=inline(z);
9 - x0=input('Enter initial value:');
10 - x=x0;
11 - display(' No.      y      f(a)      f1(a)      x')
12 - display('-----')
13 - for i=0:1:15
14 -     y=x;
15 -     x=y-(f(x)/f1(x));
16 -     if x==y
17 -         break
18 -     else fprintf(' %d      %f      %f      %f      %f \n',i,y,f(x),f1(x),
19 -         end

```

Command Window Output:

```

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
fx >>

```

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: 05

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

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

Tool: MATLAB Software

Methodology:

(I) Algorithm:

1. Start
2. Get values of x_0 , x_1 and e
*Here x_0 and x_1 are the two initial guesses
 e is the stopping criteria, absolute error or the desired degree of accuracy*
3. Compute $f(x_0)$ and $f(x_1)$
4. Compute $x_2 = [x_0 * f(x_1) - x_1 * f(x_0)] / [f(x_1) - f(x_0)]$
5. Test for accuracy of x_2
If $[(x_2 - x_1) / x_2] > e$, *Here $[\]$ is used as modulus sign*
then assign $x_0 = x_1$ and $x_1 = x_2$
goto step 4
Else, goto step 6
6. Display the required root as x_2 .
7. Stop

(II) Flowchart:

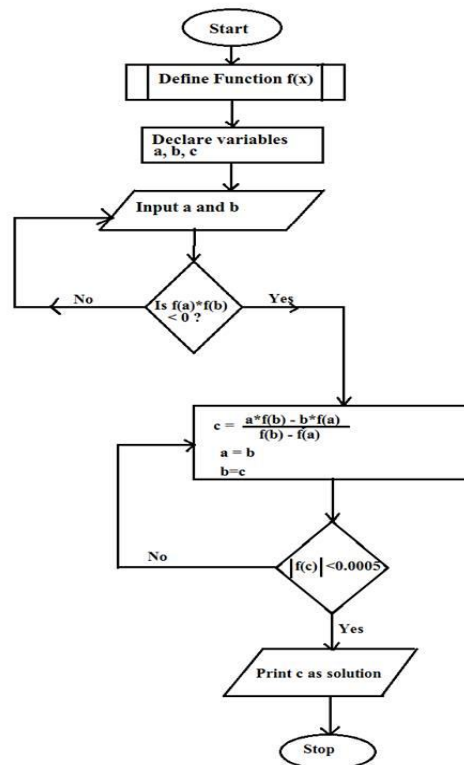


Fig. Flow Chart for Secant Method

codewithc.com

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:

The screenshot shows the MATLAB Editor with a script named 'Secant.m' and the Command Window displaying the execution results. The script implements the Secant method for finding roots of a function. The Command Window shows the user inputting the function $2x^2 - 15x + 3$ and two initial assumptions, 0 and 1. The output table shows the iterative process, with values of a , b , and xn converging to approximately 0.205638 over 6 iterations.

```

Editor - D:\Install Data\MATLAB\bin\Secant.m
+3 Table.m x Untitled2.m x NR.m x N_R2.m x Secant.m x +
1 - clear all
2 - clc
3 - syms x;
4 - fun=input('Enter the fun:');
5 - f=inline(fun);
6 - while(1)
7 -     a=input('Enter the value of 1st assumption:');
8 -     b=input('Enter the value of 2nd assumption:');
9 -
10 -     if f(a)*f(b)>0
11 -         disp('Wrong');
12 -     elseif f(a)*f(b)<=0
13 -         break;
14 -     end
15 - end

Command Window
Enter the fun:2*x^2-15*x+3
Enter the value of 1st assumption:0
Enter the value of 2nd assumption:1
No.      a      b      xn
-----
1      0.000000      1.000000      0.230769
2      0.230769      0.000000      0.206349
3      0.206349      0.230769      0.205636
4      0.205636      0.206349      0.205638
5      0.205638      0.205636      0.205638
6      0.205638      0.205638      0.205638
fx >>

```

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: 06

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:

$$x = g(x) \dots (1)$$

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

$$x^2 - 2x + 3 = 0 \text{ can be simply manipulated to yield } x = (x^2 + 3)/2$$

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 x_i , equation 1 can be used to compute a new estimate x_{i+1} as expressed by the iterative formula

$$x_{i+1} = g(x_i) \dots (2)$$

Tool: MATLAB Software

Methodology:

(I) Algorithm:

1. Start
2. Read values of x_0 and e .
*Here x_0 is the initial approximation
 e is the absolute error or the desired degree of accuracy, also the stopping criteria*
3. Calculate $x_1 = g(x_0)$
4. If $[x_1 - x_0] \leq e$, goto step 6.
Here $[]$ refers to the modulus sign
5. Else, assign $x_0 = x_1$ and goto step 3.
6. Display x_1 as the root.
7. Stop

(II) Flowchart:

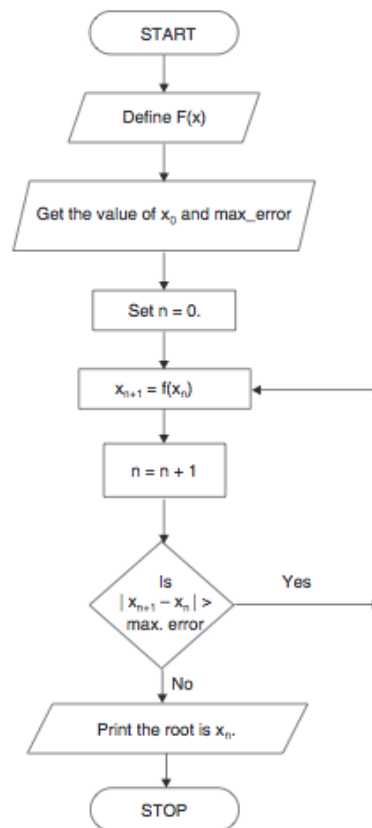


Figure 2.1 Flowchart of iteration method procedure [2]

(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
end
  
```

Output:

The image shows a MATLAB Editor window with the file 'Succ_Appro.m' open. The code implements the Successive Approximation method for finding roots of a function. It includes a function definition, a loop for iterations, and a break condition based on the absolute difference between successive values. The Command Window shows the execution of the script, displaying the function, the initial assumption, and a table of iterations.

```

Editor - D:\Install Data\MATLAB\bin\Succ_Appro.m
+3  Untitled2.m  NR.m  N_R2.m  Secant.m  Succ_Appro.m
9      fprintf('Root')
10     return
11 end
12 display(' No.      a      xn ')
13 display(' --      ----      ---- ')
14
15
16 for i=1:1:20
17     xn=f(a);
18     if abs(xn-a)<0.001
19         break;
20     else fprintf(' %d      %f      %f\n',i,a,xn);
21         a=xn;
22     end
23 end

```

Command Window

```

Enter the fun:(x^3+3)/5
Enter the value of initial assumption:0
No.      a      xn
--      ----      ----
1      0.000000      0.600000
2      0.600000      0.643200
3      0.643200      0.653219
4      0.653219      0.655745
fx >>

```

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: 07

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:

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

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

.....

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

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:

The screenshot shows the MATLAB Editor with the file 'Gauss_elm.m' open. The code is as follows:

```
1 % AX=B
2 A=[1 1 1; 2 1 3; 3 4 -2];
3 B=[4;7;9];
4 % Augmented matrix
5 AB=[A,B];
6 %% pivot 1 1
7 alpha = AB(2,1)/AB(1,1);
8 AB(2,:)=AB(2,:)-alpha*AB(1,:);
9 alpha=AB(3,1)/AB(1,1);
10 AB(3,:)=AB(3,:)-alpha*AB(1,:);
11 %% pivot 2 2
12 alpha=AB(3,2)/AB(2,2);
13 AB(3,:)=AB(3,:)-alpha*AB(2,:);
14 %% Back Subs
15 x=zeros(3,1);
16 x(3) = AB(3,end)/AB(3,3);
17 x(2) = (AB(2,end)-AB(2,3)*x(3))/AB(2,2);
18 x(1) = (AB(1,end)-(AB(1,3)*x(3)+AB(1,2)*x(2)))/AB(1,1);
19
20
```

The Command Window shows the following output:

```
>> Gauss_elm
>> AB

AB =

     1     1     1     4
     0    -1     1    -1
     0     0    -4    -4

>> x

x =

     1
     2
     1

fx >> |
```

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: 08

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:

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

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

.....

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

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:

The screenshot shows the MATLAB Editor with a script named 'Gauss_Jordan.m' and the Command Window displaying the output. The script defines matrix A, vector B, and the augmented matrix AB. It then performs row operations to reach row echelon form using pivots at (1,1), (2,2), and (3,3), followed by back substitution to find the solution vector x.

Command Window Output:

```
>> Gauss_Jordan
>> AB

AB =

     1     1     1     4
     2     1     3     7
     3     4    -2     9

>> x

x =

     1
     2
     1
```

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: 09

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:

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

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

.....

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

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);
```

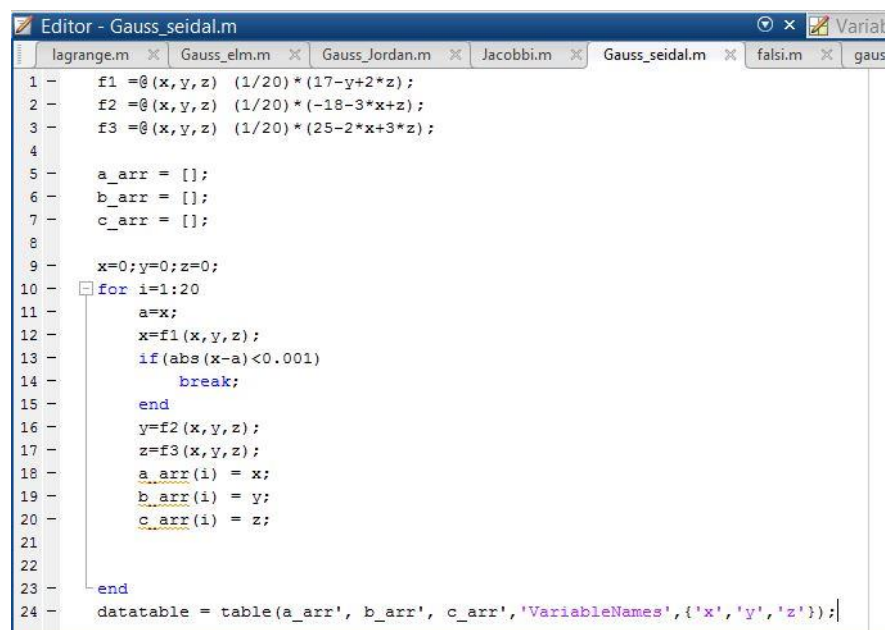
```
a_arr = [];
b_arr = [];
c_arr = [];
```

```
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:



```
Editor - Gauss_seidal.m
lagrange.m x Gauss_elm.m x Gauss_Jordan.m x Jacobbi.m x Gauss_seidal.m x falsi.m x gaus
1 f1 = @(x,y,z) (1/20)*(17-y+2*z);
2 f2 = @(x,y,z) (1/20)*(-18-3*x+z);
3 f3 = @(x,y,z) (1/20)*(25-2*x+3*z);
4
5 a_arr = [];
6 b_arr = [];
7 c_arr = [];
8
9 x=0;y=0;z=0;
10 for i=1:20
11     a=x;
12     x=f1(x,y,z);
13     if (abs(x-a)<0.001)
14         break;
15     end
16     y=f2(x,y,z);
17     z=f3(x,y,z);
18     a_arr(i) = x;
19     b_arr(i) = y;
20     c_arr(i) = z;
21
22
23 end
24 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 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: 10

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:

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

[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)
```

```

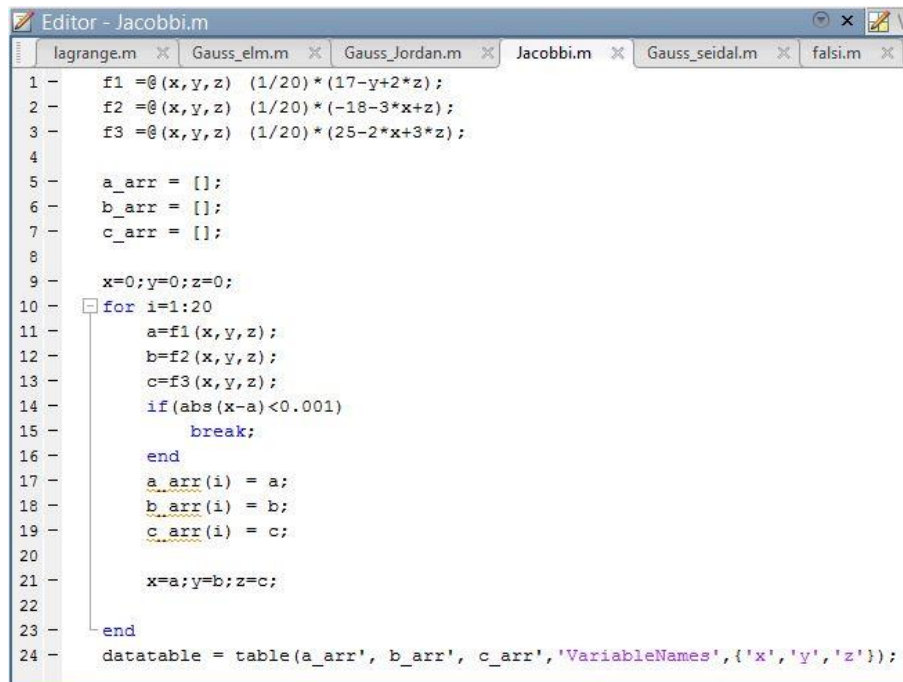
        break;
    end
    a_arr(i) = a;
    b_arr(i) = b;
    c_arr(i) = c;

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

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

```

Output:



```

Editor - Jacobbi.m
lagrange.m  Gauss_elm.m  Gauss_Jordan.m  Jacobbi.m  Gauss_seidal.m  falsi.m
1 - f1 = @(x,y,z) (1/20)*(17-y+2*z);
2 - f2 = @(x,y,z) (1/20)*(-18-3*x+z);
3 - f3 = @(x,y,z) (1/20)*(25-2*x+3*z);
4
5 - a_arr = [];
6 - b_arr = [];
7 - c_arr = [];
8
9 - x=0;y=0;z=0;
10 - for i=1:20
11 -     a=f1(x,y,z);
12 -     b=f2(x,y,z);
13 -     c=f3(x,y,z);
14 -     if(abs(x-a)<0.001)
15 -         break;
16 -     end
17 -     a_arr(i) = a;
18 -     b_arr(i) = b;
19 -     c_arr(i) = c;
20
21 -     x=a;y=b;z=c;
22
23 - end
24 - datatable = table(a_arr', b_arr', c_arr', 'VariableNames', {'x', 'y', 'z'});

```

MATLAB Variable: datatable			
22-Feb-2020			
	1	2	3
	x	y	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: 11

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]

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

where

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

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:

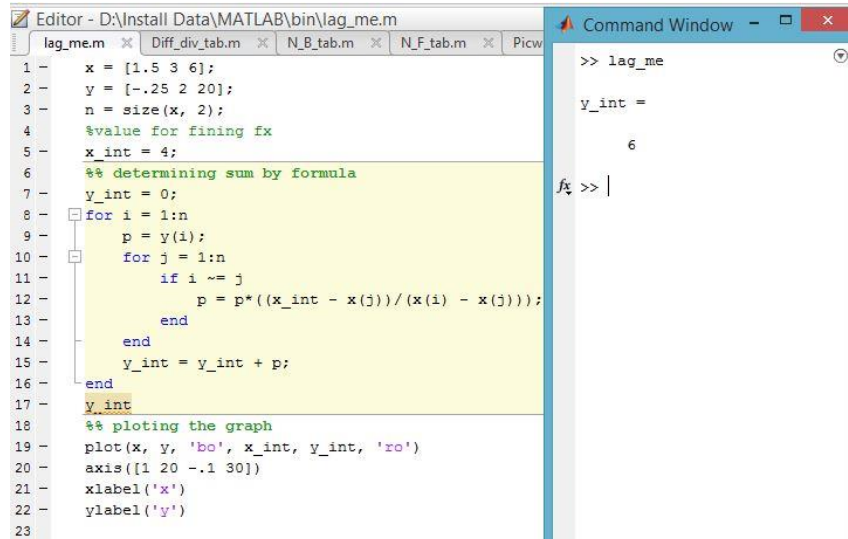
```
x = [1.5 3 6];
y = [-.25 2 20];
n = size(x, 2);
%value for fining fx
x_int = 4;
%% determining sum by formula
y_int = 0;
for i = 1:n
    p = y(i);
    for j = 1:n
        if i ~= j
            p = p * ((x_int - x(j)) / (x(i) - x(j)));
        end
    end
    y_int = y_int + p;
end
y_int
%% plotting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

```

plot(x, y, 'bo', x_int, y_int, 'ro')
axis([1 20 -1 30])
xlabel('x')
ylabel('y')

```

Output:



The screenshot shows the MATLAB Editor with the file lag_me.m open. The script contains the following code:

```

1 x = [1.5 3 6];
2 y = [-.25 2 20];
3 n = size(x, 2);
4 %value for finding fx
5 x_int = 4;
6 %% determining sum by formula
7 y_int = 0;
8 for i = 1:n
9     p = y(i);
10    for j = 1:n
11        if i ~= j
12            p = p*((x_int - x(j))/(x(i) - x(j)));
13        end
14    end
15    y_int = y_int + p;
16 end
17 y_int
18 %% plotting the graph
19 plot(x, y, 'bo', x_int, y_int, 'ro')
20 axis([1 20 -1 30])
21 xlabel('x')
22 ylabel('y')
23

```

The Command Window shows the output of the script:

```

>> lag_me

y_int =

     6
fx >> |

```

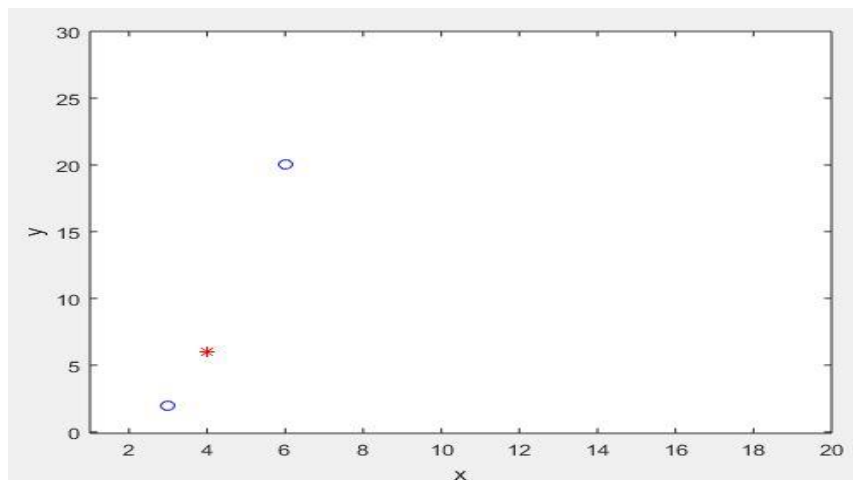


Figure 11.1: Graph Of The Function

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

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: 12

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

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

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

$$= \frac{1}{x - x_0} \left[\begin{array}{c} f(x_0) \\ f(x_1) \end{array} \right] = \frac{f_1 - f_0}{x_1 - x_0} + \frac{x}{x_1 - x_0}$$

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 second 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

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

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

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

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

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

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

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-1,1));
        i=i+1;
        if(i>=n)
            break;
        end
    end
    k=k+1;l=l+1;z=z+1;
end
%value for finding 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
%% plotting the graph
```



```

plot(x, fx, 'bo', x_int, y_sum, 'ro')
axis([-10 10 -1000 4000])
xlabel('x')
ylabel('y')

```

Output:

```

lag_me.m  Diff_div_tab.m  N_B_tab.m  N_F_tab.m  Picwise_spline.m  +
1  x=[-3 -1 0 3 5];
2  fx=[-30 -22 -12 330 3458];
3  n=size(x,2);
4  %array of zeros
5  dt=zeros(n,n);
6  %% inserting x and fx in dt
7  for i=1:n
8      dt(i,1)=x(i);
9      dt(i,2)=fx(i);
10 end
11 %% creating divided difference table

Command Window

>> Diff_div_tab

dt =

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

y_sum =

    102.6875

```

	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

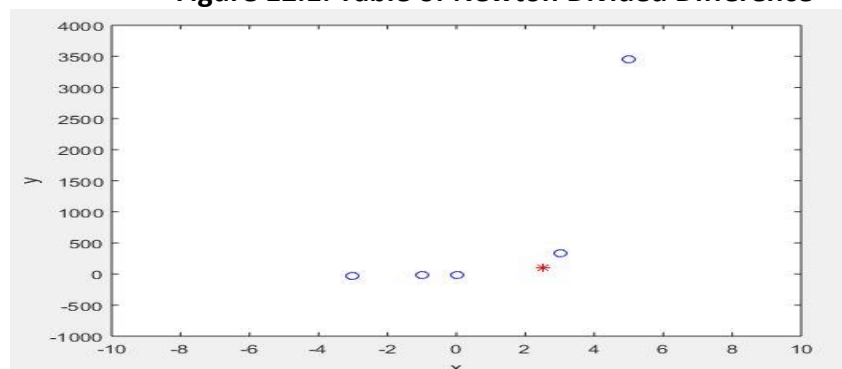


Figure 12.2: Graph Of The Function

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: 13

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 fuctions which are known at equally spaced data points.

Consider the equation of the linear interpolation obtained in the earlier section :

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

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

$$\begin{aligned} & \frac{x_1 - x}{x_1 - x_0}f_0 + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0) + \frac{x - x_0}{x_1 - x_0}f_0 \\ &= f_0 + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0) \end{aligned}$$

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

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:

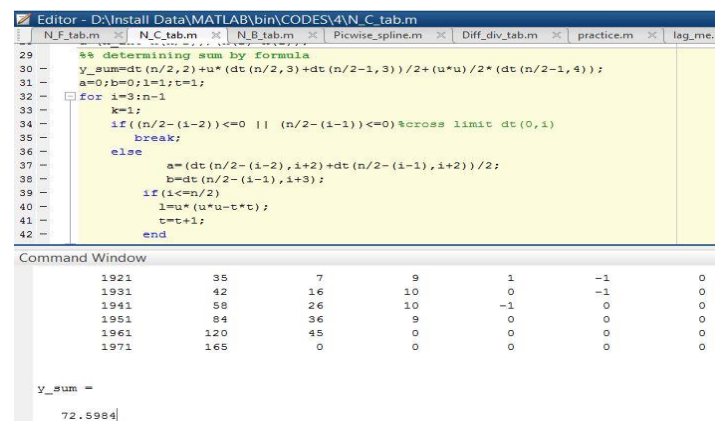
```
x=[1921 1931 1941 1951 1961 1971];
fx=[35 42 58 84 120 165];
n=size(x,2);
%array of zeros
dt=zeros(n,n);
%% inserting x and fx in dt
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
```

```

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
    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+(1/k)*a+u*l*b;
    end
end
y_sum
%% plotting the graph
plot(x, fx, 'bo', x_int, y_sum, 'r*')
axis([1900 2000 0 250])
xlabel('x')
ylabel('y')

```

Output:



	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

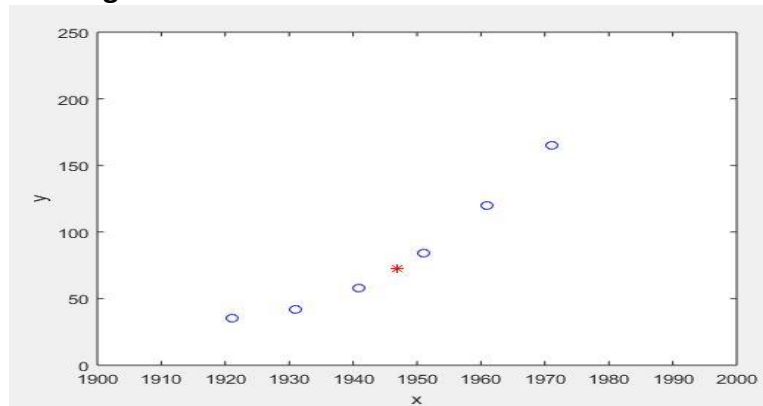


Figure 13.2: Graph Of The Function

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: 14

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 $y_1 - y_0, y_2 - y_1, \dots, y_n - y_{n-1}$ when denoted by $\Delta y_1, \Delta y_2, \dots, \Delta y_n$, 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:

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

Tool: MATLAB Software

Methodology:

MATLAB Code:

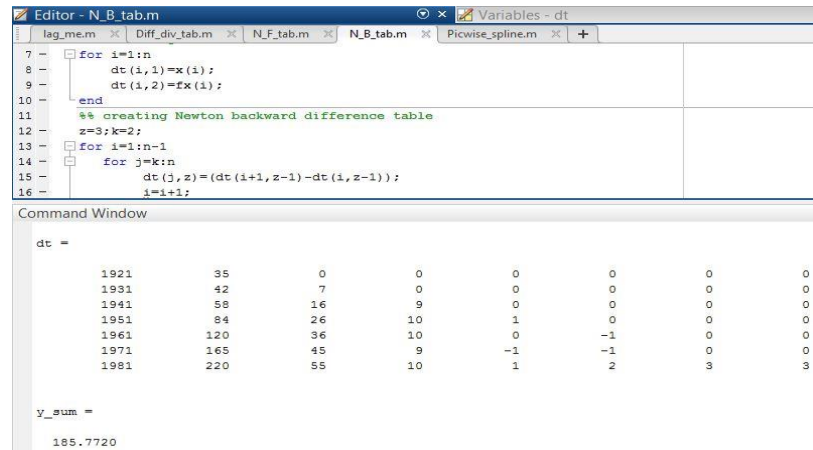
```
x=[1921 1931 1941 1951 1961 1971 1981];
fx=[35 42 58 84 120 165 220];
n=size(x,2)
%array of zeros
dt=zeros(n,n)
%% inserting x and fx in dt
for i=1:n
    dt(i,1)=x(i);
    dt(i,2)=fx(i);
end
%% creating Newton backward difference table
z=3;k=2;
for i=1:n-1
    for j=k:n
        dt(j,z)=(dt(i+1,z-1)-dt(i,z-1));
        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
    %% plotting the graph
    plot(x, fx, 'bo', x_int, y_sum, 'r*')
    axis([1900 2000 0 250])
    xlabel('x')
    ylabel('y')

```

Output:



	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

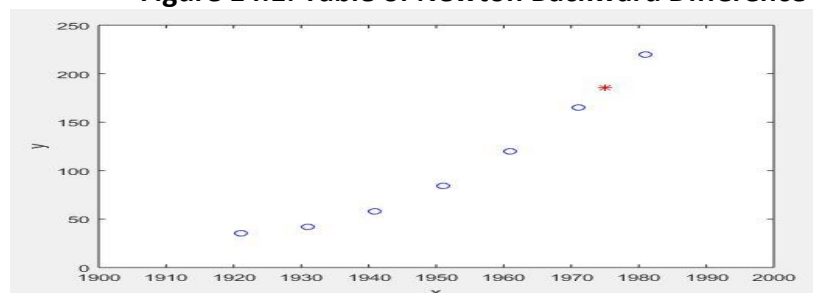


Figure 14.2: Graph Of The Function

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: 15

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 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. 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 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ where $x_1 < x_2 < \dots < x_n$, a piecewise linear function is defined for a point x such that $x_k \leq x \leq x_{k+1}$.

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:

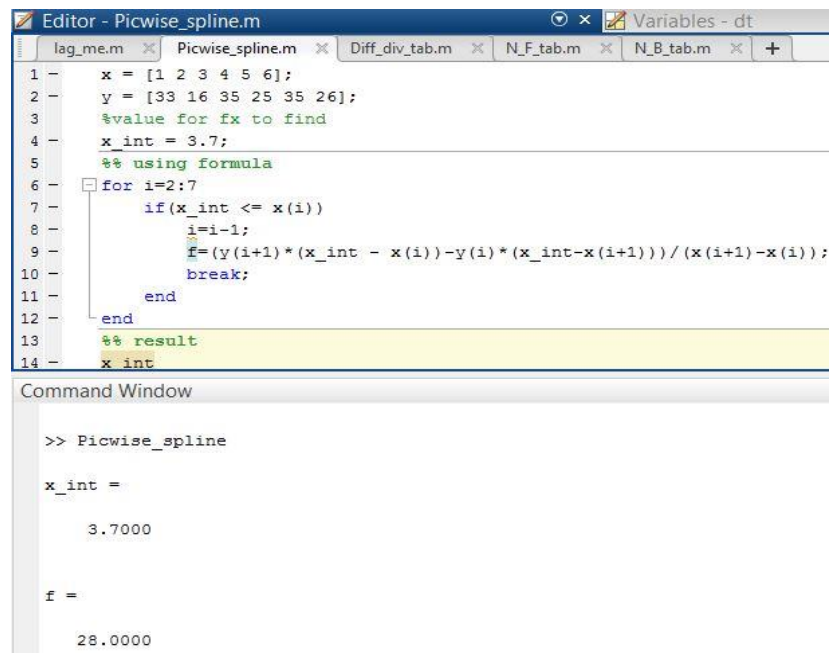
```
x = [1 2 3 4 5 6];
y = [33 16 35 25 35 26];
%value for fx to find
x_int = 3.7;
%% using formula
for i=2:7
    if(x_int <= x(i))
        i=i-1;
        f=(y(i+1)*(x_int - x(i))-y(i)*(x_int-x(i+1)))/(x(i+1)-x(i));
        break;
    end
end
```

```

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

```

Output:



The screenshot shows the MATLAB Editor with the file `Picwise_spline.m` open. The script defines a piecewise linear function and finds the value of `f` at `x_int = 3.7` using linear interpolation. The Command Window shows the execution results:

```

>> Picwise_spline

x_int =

    3.7000

f =

    28.0000

```

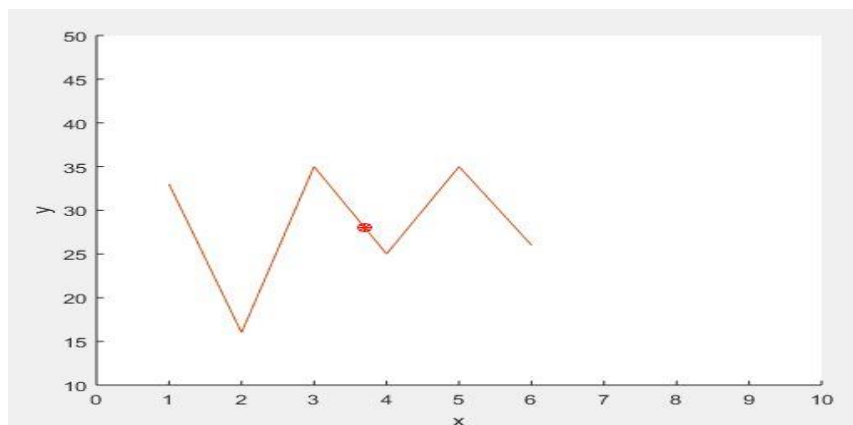


Figure 15.1: Graph of the Function

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: 16

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 $\int_a^b f(x)dx$ is given by

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

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

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

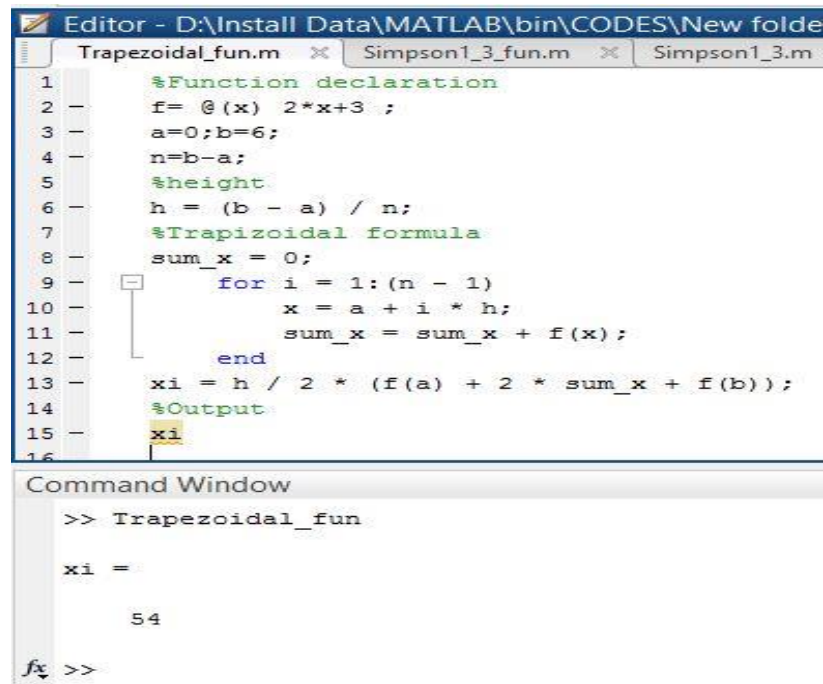
Tool: MATLAB Software

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:



The image shows a MATLAB Editor window with three tabs: 'Trapezoidal_fun.m', 'Simpson1_3_fun.m', and 'Simpson1_3.m'. The 'Trapezoidal_fun.m' tab is active, displaying the following code:

```
1 %Function declaration
2 f = @(x) 2*x+3;
3 a=0;b=6;
4 n=b-a;
5 %height
6 h = (b - a) / n;
7 %Trapezoidal formula
8 sum_x = 0;
9 for i = 1:(n - 1)
10     x = a + i * h;
11     sum_x = sum_x + f(x);
12 end
13 xi = h / 2 * (f(a) + 2 * sum_x + f(b));
14 %Output
15 xi
16
```

Below the editor is the Command Window, which shows the execution of the script:

```
>> Trapezoidal_fun
xi =
    54
fx >>
```

Result(s)& Discussion: The integral value of a function with limit 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