

THE NEOTIA UNIVERSITY

LAB MANUAL

**NUMERICAL METHODS AND
OPTIMIZATION TECHNIQUES**

ES-MA/P/401

CONTENT

SL NO.	NAME OF THE EXPERIMENT	EXPERIMENT NUMBER
1	Implementation of Newton Gregory forward interpolation method	ES-MA/P/401/01
2	Implementation of derivative by divided difference	ES-MA/P/401/02
3	Implementation of derivative using difference table	ES-MA/P/401/03
4	Implementation of Lagrange interpolation method	ES-MA/P/401/04
5	Implementation of Neville's interpolation method	ES-MA/P/401/05
6	Implementation of divided difference interpolation method	ES-MA/P/401/06
7	Implementation of successive bisection method to find the root of the transcendental equations	ES-MA/P/401/07
8	Implementation of secant method to find the root of the transcendental equations	ES-MA/P/401/08
9	Implementation of method of false position to find the root of the transcendental equations	ES-MA/P/401/09

10	Implementation of Newton-Raphson iterative method to find the root of the transcendental equations	ES-MA/P/401/10
11	Implementation of Gauss elimination method to find the root of a system of linear equations	ES-MA/P/401/11
12	Implementation of Gauss Jordan elimination method to find the root of a system of linear equations	ES-MA/P/401/12
13	Implementation of Gauss Siedel iterative method to find the root of a system of linear equations	ES-MA/P/401/13
14	Implementation of Trapezoidal rule for numerical integration	ES-MA/P/401/14
15	Implementation of Simpson's $1/3^{\text{rd}}$ rule for numerical integration	ES-MA/P/401/15

NEWTON GREGORY FORWARD POLYNOMIAL METHOD OF INTERPOLATION

Problem: Find the value of a polynomial by the Newton Gregory method of interpolation in between the following points, where the function values at respective points are given.

x	f(x)
0.0	0
0.2	0.203
0.4	0.423
0.6	0.684
0.8	1.030
1.0	1.557

PROCEDURE:

If at the points $x_0, x_1, x_2, \dots, x_n$ the corresponding function values are $f_0, f_1, f_2, \dots, f_n$, then a polynomial can be fitted with those given points and value of the polynomial at another point can be calculated by using this method, with only restriction that here x_i 's have to be equally spaced.

Corresponding 'Difference Table' can be obtained as,

x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
x_0	f_0	$\Delta f_0 = f_1 - f_0$			
x_1	f_1	Δf_1	$\Delta^2 f_0$		
x_2	f_2	Δf_2	$\Delta^2 f_1$	$\Delta^3 f_0$	$\Delta^4 f_0$
x_3	f_3	Δf_3	$\Delta^2 f_2$	$\Delta^3 f_1$	
x_4	f_4	Δf_4			
x_5	f_5				

From the difference table,

For 1st order:

$$\Delta f_0 = f_1 - f_0$$

$$\Delta f_1 = f_2 - f_1$$

So generalizing, $\Delta f_i = f_i - f_{i-1}$

For 2nd order:

$$\Delta^2 f_0 = \Delta(\Delta f_0) = f_2 + f_0 - 2f_1$$

$$\Delta^2 f_1 = f_3 + f_1 - 2f_2$$

So generalizing, $\Delta^2 f_i = f_{i+2} + f_i - 2f_{i+1}$

Similarly, $\Delta^3 f_i = f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i$

In this way generalizing for nth order it can be obtained as,

$$\Delta^n f_i = f_{i+n} - \binom{n}{1} f_{i+n-1} + \binom{n}{2} f_{i+n-2} - \binom{n}{3} f_{i+n-3} + \dots$$

If s can be calculated as, $s = \frac{x_s - x_i}{h}$ where $h = x_{i+1} - x_i$ and x_s is the value of x at which the value of the polynomial is desired to find out.

Then the value of the polynomial at x_s can be calculated as,

$$P_n(x_s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \dots$$

ALGORITHM:

1. Initialize no of iterations as 'n', which is equal to the no of x_i 's.
2. Initialize the point $m=0.73$ around which solution of the polynomial is desired to calculate.
3. Initialize a row vector x to take the x_i 's as inputs.
4. for j loop in steps of 1
5. Coefficients i.e. a_i 's are calculated.
6. end for
7. Initialize a matrix delta to take different orders of calculated delta's as inputs.
8. Calculate $s = (m-x(1))/(x(2)-x(1))$
9. $sum=0$;
10. $product=s$;
11. for j loop 1 to $n-2$ in steps of 1
12. $product=(product*(s-j))/(j+1)$
13. $pdt = product*delta(j)$
14. $sum=sum + pdt$
15. end for
16. $P=f(1)+(s* delta(1))+sum$
17. Display P as a solution.

SIMULATION CODE:

```
% Newton Gregory Method%
clear all;
x0=0.73;
x=[0.0 0.2 0.4 0.6 0.8 1.0]';
h=x(2)-x(1);
s=(x0-x(1))/h;
a=zeros(6,5);
v=[0 0.203 0.423 0.684 1.030 1.557]';
A=[v a];
n=6;

for (j=2:n)

    for(i=1:n-1)

        A(i,j)=A(i+1,j-1)-A(i,j-1);

    end
    n=n-1;
end

i=1;
```

```

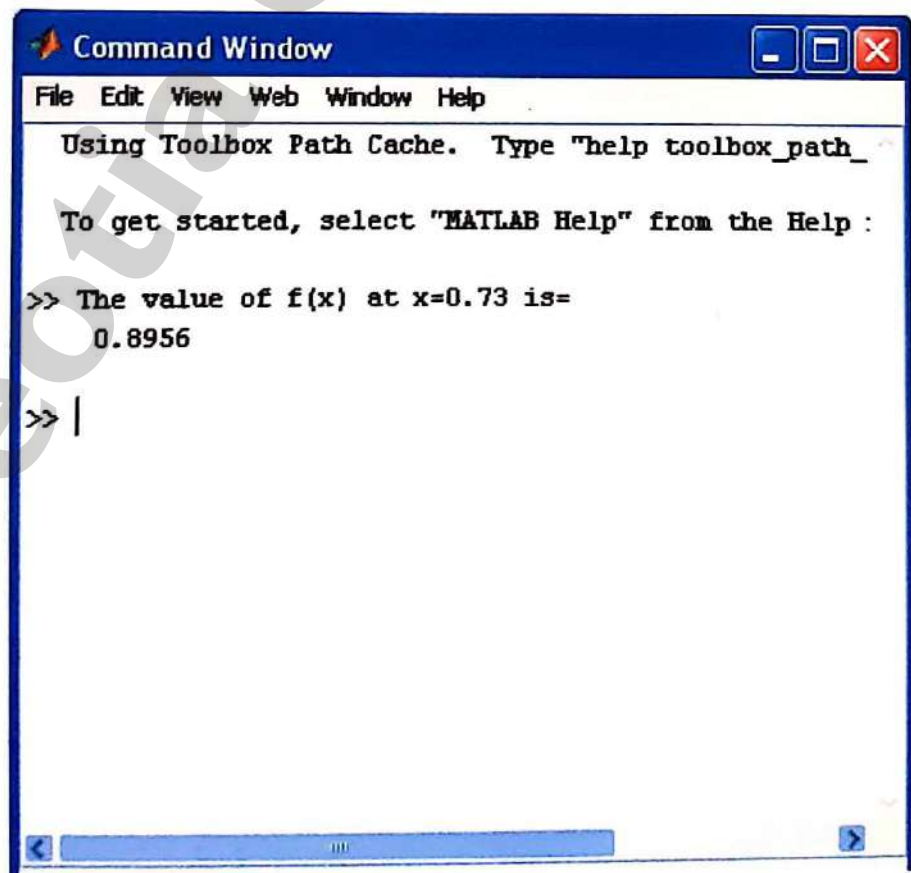
P=0;
for(j=2:5)
    f=1;
    for(k=0:j-2)
        f=(f*(s-k));

    end
    P=P+(f*A(i,j))/factorial(j-1);
end

PN=A(1,1)+P;
disp('The value of f(x) at x=0.73 is=');
disp(PN);

```

RESULT AFTER SIMULATION:



DERIVATIVE BY DIVIDED DIFFERENCE

Problem: Find the derivative of the fitted polynomial through the following points where the corresponding functional values are also given.

x	$f(x)$
0	1
2	3
3	7
5	21
6	31

PROCEDURE:

If at the points $x_0, x_1, x_2, \dots, x_n$ the corresponding function values are $f_0, f_1, f_2, \dots, f_n$
Then the fitted polynomial of x through these x_i 's is given by

$$P_n(x) = f[x_0] + (x - x_0) * f[x_0, x_1] + (x - x_0)(x - x_1) * f[x_0, x_1, x_2] + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1}) * f[x_0, x_1, \dots, x_n] \quad (1)$$

Thus the polynomial for the divided difference becomes, $f(x) = P_n(x) + e$ (2)
Let f_i be the value of the polynomial at x_i . Then,

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} = f_0^{[1]}$$

$$f[x_1, x_2] = \frac{f_2 - f_1}{x_2 - x_1} = f_1^{[1]}$$

Thus, $f[x_i, x_j] = f[x_j, x_i]$, where '[1]' denotes the first order of the divided difference.

Similarly, for second order of the divided difference

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f_0^{[2]}$$

$$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} = f_1^{[2]}$$

Now it can be written as,

$$f[x_0, x_1, x_2, \dots, x_n](x - x_0)(x - x_1) \dots (x - x_{n-1}) = f[x_0, x_1, \dots, x_n] \prod_{i=0}^{n-1} (x - x_i) \quad (3)$$

Thus neglecting the error e , from (1), (2) and (3) in generally it can be written as,

$$P_n(x) = f_0^{[0]} + (x - x_0)f_0^{[1]} + (x - x_0)(x - x_1)f_0^{[2]} + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1})f_0^{[n]} \quad (4)$$

Now from equation (3),

$$\frac{d}{dx} \left\{ \prod_{i=0}^{n-1} (x - x_i) \right\} = \sum_{i=0}^{n-1} \frac{(x - x_0)(x - x_1) \dots (x - x_{n-1})}{(x - x_i)} \quad (5)$$

$$= \sum_{i=0}^{n-1} \prod_{j=0, j \neq i}^{n-1} (x - x_j), \forall j \neq i$$

Finally the derivative of the polynomial $P_n(x)$ can be obtained as,

$$P'_n(x) = 0 + f[x_0, x_1] + f[x_0, x_1, x_2] \{ (x - x_1) + (x - x_0) \} + \dots + f[x_0, x_1, \dots, x_{n-1}] \sum_{i=0}^{n-1} \prod_{j=0, j \neq i}^{n-1} (x - x_j) \quad (6)$$

The difference table becomes,

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4}]$
x_0	f_0	$f[x_0, x_1] = a_1$	$f[x_0, x_1, x_2] = a_2$	$f[x_0, x_1, x_2, x_3] = a_3$	$f[x_0, x_1, x_2, x_3, x_4] = a_4$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	
x_2	f_2	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$		
x_3	f_3	$f[x_3, x_4]$			
x_4	f_4				
x_5	f_5				

ALGORITHM:

1. Initialize no of iterations as 'n', which is equal to the no of x_i 's.
2. Initialize the point $m=3.0$ around which derivative of the polynomial is desired to calculate.
3. Initialize a row vector x to take the x_i 's as inputs.
4. Initialize a column vector v to take the f_i 's as inputs.
5. Initialize a 5×4 matrix 'a' with zeros such as it's first column becomes v .
6. for j loop 2 to n in steps of 1
7. for i loop 1 to n-1 in steps of 1
8. Coefficients i.e. a_i 's are calculated as $A(i, j) = \frac{A(i+1, j-1) - A(i, j-1)}{x(i+j-1) - x(i)}$
9. end for
10. $n=n-1$.
11. end for
12. Display the difference table A.

13. Initialize another column vector M to take only the different elements of the first row of the matrix A.
14. sum=0;
15. product=1;
16. for j loop 1 to n in steps of 1
17. for j loop 1 to n in steps of 1
18. product=product*(m-x(i))
19. P=product*M(j);
20. sum=sum+P;
21. end for
22. end for
23. sum=sum+A(1,1)
24. sum is stored at matrix A.
25. Initialize i=1 and S=0
26. for j loop 5 to 3 in 1 steps of decreasing
27. Initialize P=0
28. for z in loop 1 to j-1
29. Initialize P1=1
30. for k in loop 1 to j-1
31. if z ≠ k
32. P1=P1*(x0-x(k))
33. end if
34. end for
35. P=P+P1
36. end for
37. S=A(i,j)*P
38. end for
39. L=S+A(1,2)
40. Display L as the solution i.e. derivative of the polynomial.

SIMULATION CODE:

% Derivative using Divided Difference%

```
clear all;
x0=4.1;
x=[0 2 3 5 6]';
a=zeros(5,4);
v=[1 3 7 21 31]';
A=[v a];
n=5;

for j=2:n
    for i=1:n-1
        A(i,j)=(A(i+1,j-1)-A(i,j-1))/(x(i+j-1)-x(i));
```

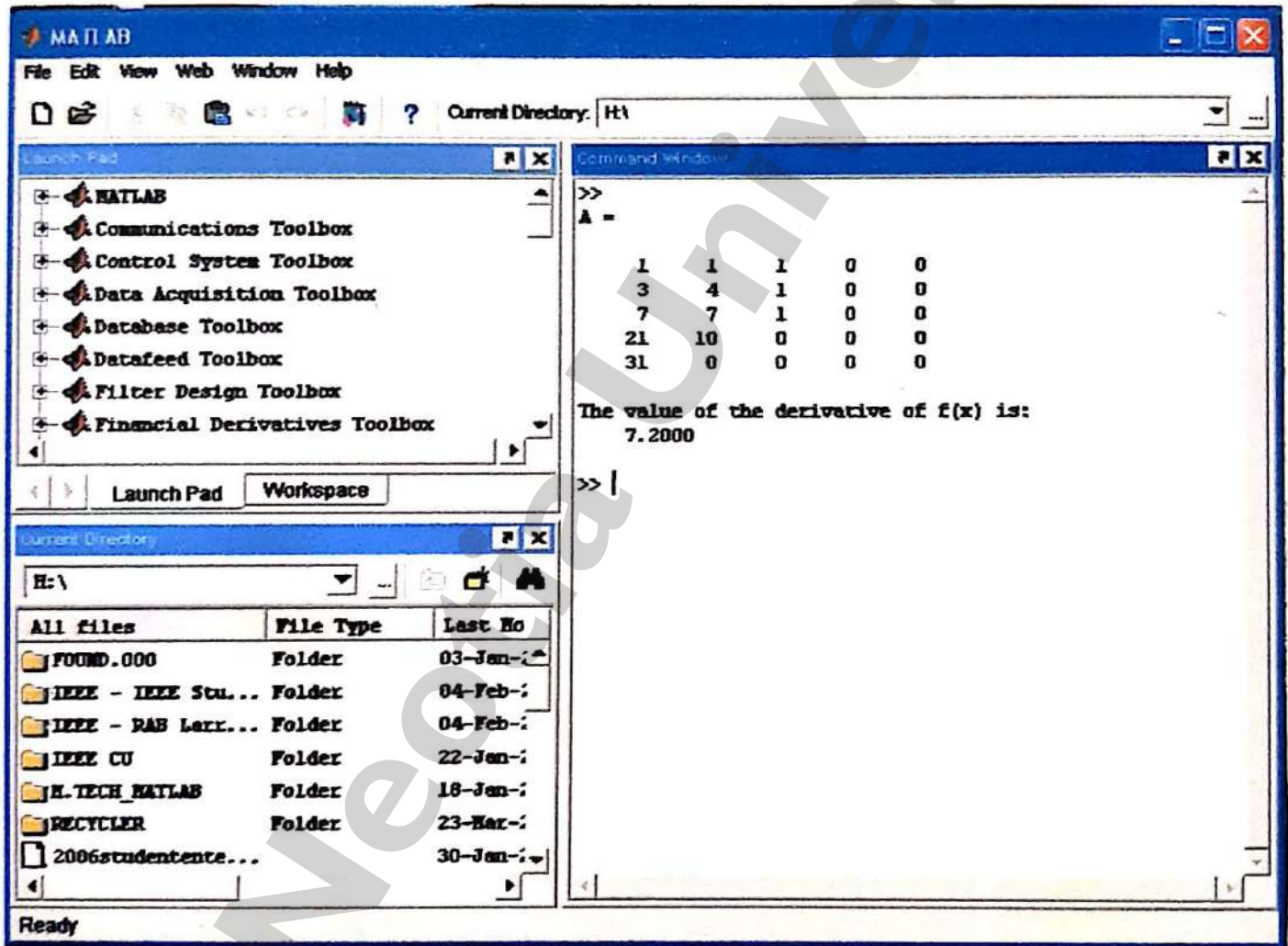
```

    end;
    n=n-1;
end;

disp(A);
i=1;
S=0;
for j=5:-1:3
    P=0;
    for z=1:j-1
        P1=1;
        for k=1:j-1
            if(z~=k)
                P1=P1*(x0-x(k));
            end;
        end;
        P=P+P1;
    end;
    S=A(i,j)*P;
end;

L=S+A(1,2);
disp('The value of the derivative of f(x) is:');
disp(L);

```



DIFFERENTIATION USING DIFFERENCE TABLE

Problem: Find the value of the derivative of a polynomial in fitted through the following points, where the function values at respective points are given.

x	$f(x)$
1.3	3.669
1.9	6.686
2.5	12.182
3.1	22.198
3.7	40.447
4.3	73.700
4.9	134.290

PROCEDURE:

If at the points $x_0, x_1, x_2, \dots, x_n$ the corresponding function values are $f_0, f_1, f_2, \dots, f_n$, then a polynomial can be fitted with those given points and value of the polynomial at another point can be calculated by using this method, with only restriction that here x_i 's have to be equally spaced.

Corresponding 'Difference Table' can be obtained as,

x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
x_0	f_0	$\Delta f_0 = f_1 - f_0$			
x_1	f_1	Δf_1	$\Delta^2 f_0$		
x_2	f_2	Δf_2	$\Delta^2 f_1$	$\Delta^3 f_0$	$\Delta^4 f_0$
x_3	f_3	Δf_3	$\Delta^2 f_2$	$\Delta^3 f_1$	
x_4	f_4	Δf_4			
x_5	f_5				

From the difference table,

For 1st order:

$$\Delta f_0 = f_1 - f_0$$

$$\Delta f_1 = f_2 - f_1$$

So generalizing, $\Delta f_i = f_i - f_{i-1}$

For 2nd order:

$$\Delta^2 f_0 = \Delta(\Delta f_0) = f_2 + f_0 - 2f_1$$

$$\Delta^2 f_1 = f_3 + f_1 - 2f_2$$

So generalizing, $\Delta^2 f_i = f_{i+2} + f_i - 2f_{i+1}$

$$\text{Similarly, } \Delta^3 f_i = f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i$$

In this way generalizing for nth order it can be obtained as,

$$\Delta^n f_i = f_{i+n} - n f_{i+n-1} + \frac{n(n-1)}{2!} f_{i+n-2} - \frac{n(n-1)(n-2)}{3!} f_{i+n-3} + \dots$$

If s can be calculated as, $s = \frac{x_s - x_i}{h}$ where $h = x_{i+1} - x_i$ and x_s is the value of x at which the value of the polynomial is desired to find out.

Then the value of the polynomial at x_s can be calculated as,

$$P_n(x_s) = f_i + s \Delta f_i + \frac{s(s-1)}{2!} \Delta^2 f_i + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_i + \dots + \prod_{j=0}^{n-1} (s-j) \frac{\Delta^n f_i}{n!}$$

Since $\frac{ds}{dx} = \frac{1}{h}$ so $P'_n(S) = f'(x)$

Thus the derivative of the polynomial becomes,

$$\frac{d}{dx}[P_n(S)] = \frac{d}{ds}[P_n(S)] \cdot \frac{ds}{dx} = \frac{1}{h} [\Delta f_i + \sum_{j=2}^n \{ \sum_{k=0}^{j-1} \prod_{l=0}^{j-1} (s-l) \} \frac{\Delta^j f_i}{j!}]$$

ALGORITHM:

19. Initialize no of iterations as 'n', which is equal to the no of x_i 's.
20. Initialize the point $m=3.3$ around which derivative of the polynomial is desired to calculate.
21. Initialize a row vector x to take the x_i 's as inputs.
22. Initialize a column vector v to take the f_i 's as inputs.
23. Initialize a 5×4 matrix 'delta' with zeros such as it's first column becomes v .
24. for j loop 2 to n in steps of 1
25. for i loop 1 to $n-1$ in steps of 1
26. Coefficients i.e. a_i 's are calculated as

$$\text{delta}(i, j) = \text{delta}(i+1, j-1) - \text{delta}(i, j-1)$$
27. end for
28. $n=n-1$.
29. end for
30. Display the difference table delta.
31. Initialize $L=0$, $S=0$.
32. Initialize $h=x(2)-x(1)$ and $s=m-x(1)/h$
33. for k loop 3 to n in steps of 1
34. Initialize $\text{sum}=0$
35. for i loop 1 to $k-1$ in steps of 1
36. if $j \neq i$
37. $\text{product} = \text{product} * (s-(j-1))$
38. end if
39. end for
40. Compute $\text{sum} = \text{sum} + \text{product}$
41. end for
24. Compute $M = (\text{sum} * \text{delta}(1, k)) / \text{factorial}(k-1)$
25. Compute $S = S + M$
26. end for
27. Compute $L = (S + (\text{delta}(1, 2) * s) + f(1)) / h$
28. Display L as final result.

SIMULATION CODE:

```
% Derivative by Difference Table%
clear all;
x=[1.3 1.9 2.5 3.1 3.7 4.3 4.9];
f=[3.669 6.686 12.182 22.198 40.447 73.700 134.290];

n=input('Enter the value of n:');
m=input('Enter the value of m:');
a=zeros(7,6);
delta=[f a];

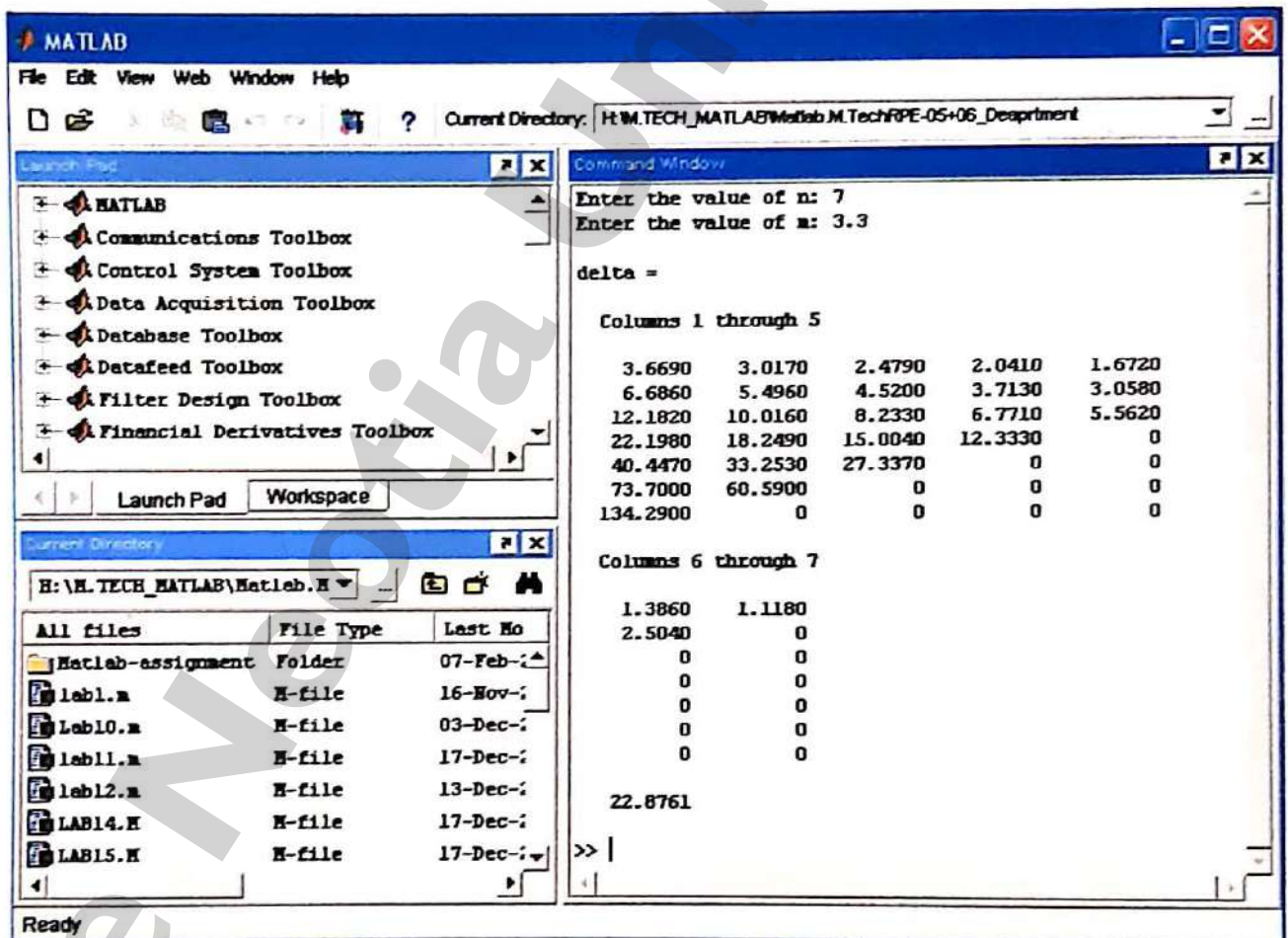
for(j=2:1:n)
    for(i=1:1:n-1)
        delta(i,j)=delta(i+1,j-1)-delta(i,j-1);
    end;
    n=n-1;
end;

delta
L=0;
S=0;
h=x(2)-x(1);
s=(m-x(1))/h;
for(k=3:1:n)
    sum=0;

    for(i=1:1:k-1)

        product=1;
        for(j=1:1:k-1)
            if(j~=i)
                product=product*(s-(j-1));
            end;
        end;
        sum=sum+product;
    end;
    M=(sum*delta(1,k))/factorial(k-1);
    S=S+M;
end;
L=(S+(delta(1,2)*s)+f(1))/h;
disp(L);
```

RESULT AFTER SIMULATION:



LAGRANGE INTERPOLATION METHOD

Problem: Find the value of a polynomial of a variable at a particular value of the variable where the value of the polynomial at different values of the variable is given.

x	$f(x)$
3.2	22.0
2.7	17.8
1.0	14.2
4.8	38.3

PROCEDURE:

The universal technique for interpolating is to fit a polynomial through the points surrounding the point 'y' where the value of the function is to be found.

Let, a polynomial is assumed in the following form:

$$P(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} f_0 + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} f_1 + \\ \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} f_2 + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} f_3$$

Where, f_1, f_2, f_3, f_4 are the values of the polynomial at the points x_1, x_2, x_3, x_4 .

In a more concise notation:

$$P(x) = \sum_{i=1}^n f(x_i) \prod_{\substack{j=1 \\ j \neq i}}^n \frac{(x-x_j)}{(x_i-x_j)}$$

Increasing the order of the polynomial does not necessarily yield a better accuracy of interpolation.

ALGORITHM:

1. Read x, n
2. for $i = 1$ to $(n+1)$ in steps of 1 do Read x_i, f_i ,
end for

Remarks: The above statement reads x_i s and the corresponding values of f_i 's.

3. sum $\leftarrow 0$
4. for $i = 1$ to $(n+1)$ in steps of 1 do
5. prodfunc $\leftarrow 1$
6. for $j = 1$ to $(n+1)$ in steps of 1 do
7. if $j \neq i$ then
$$\text{prodfunc} \leftarrow \text{prodfunc} * \frac{(x-x_j)}{(x_i-x_j)}$$

end for

8. sum $\leftarrow \text{sum} + f_i * \text{prodfunc}$

Remarks: sum is the value of f at x

end for

9. Write x , sum
10. Stop

SIMULATION CODE:

%Lagrange polynomial%

clear all;

X=1;

Y=1;

P=0;

n= input('enter the value of n');

m= input('enter the value of m');

x=[3.2 2.7 1.0 4.8 5.6];

fx=[22.0 17.8 14.2 38.3 51.7];

for(i=1:1:n)

 X=1;

 Y=1;

 for(j=1:1:n)

 if (i~=j)

 X=(X*(m-x(j)));

 Y=(Y*(x(i)-x(j)));

 end;

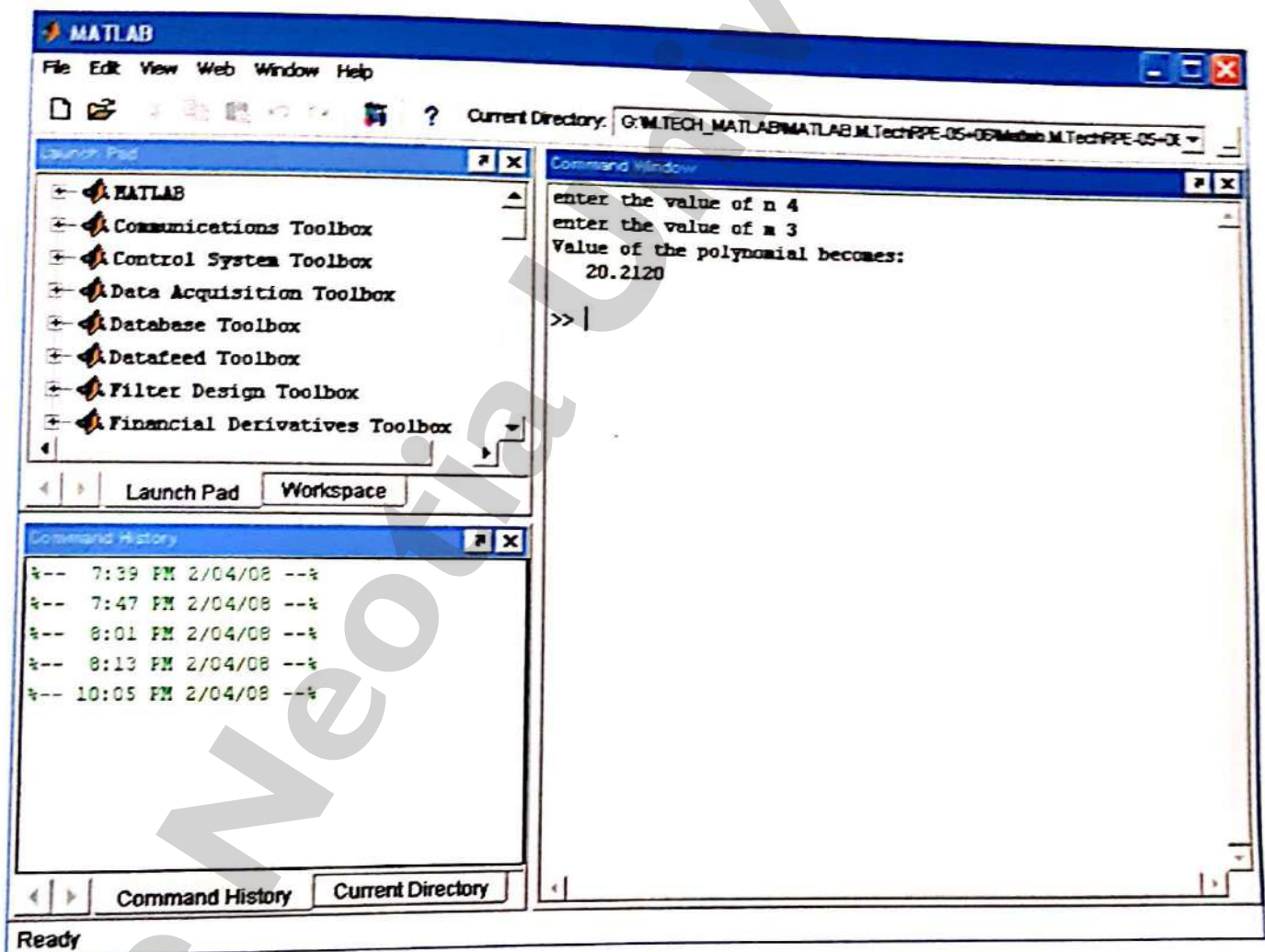
 end;

 P=(P+(X*fx(i))/Y);

end;

disp(P);

RESULT AFTER SIMULATION:



NEVILLE'S METHOD

Problem: Find the value of a polynomial by the Neville's method of interpolation in between the following points, where the function values at respective points are given.

x	f(x)
32.0	0.52992
22.2	0.37784
41.6	0.66393
10.1	0.17537
50.5	0.63608

PROCEDURE:

The universal technique for interpolation is to fit a polynomial through the points surrounding the point 'x' where the value of the function is to be found.

Let in between the points (x_1, f_1) and (x_2, f_2) the fitted polynomial can be written as

$$\begin{aligned} f(x) = P(x) &= \frac{(x-x_2)}{(x_1-x_2)} f_1 + \frac{(x-x_1)}{(x_2-x_1)} f_2 \\ &= \frac{(x-x_2).f_1 + (x_1-x)f_2}{(x_1-x_2)} \end{aligned}$$

Where, f_1 and f_2 are the function values at the points x_1 and x_2 , respectively. x is the point around which the polynomial has to be calculated.

In general way, for any number of points the solution for the fitted polynomial can be written as,

$$P_{i,j} = \frac{(x-x_i).P_{i+1,j-1} + (x_{i+j}-x).P_{i,j-1}}{x_{i+j}-x_i}$$

Procedure is continued with number of successive approximations, and stops when the difference between two successive P_i 's becomes within a pre-defined error limit.

ALGORITHM:

1. Initialize no of iterations as 'n', which is equal to the no of x_i 's.
2. Initialize the point $xx=27.5$ around which solution of the polynomial is desired to calculate.
3. Initialize a column vector x to take the x_i 's as inputs.
4. Initialize a 5x5 matrix p .
5. Enter the function values i.e. $f(x_i)$'s in the first column of matrix p .
6. for $j = 2$ to 5 in steps of 1 loop
7. for $i = 1$ to $n-1$ in steps of 1 loop
8. Calculate $P_{i,j} = \frac{(x-x_i).P_{i+1,j-1} + (x_{i+j}-x).P_{i,j-1}}{x_{i+j}-x_i}$
9. end for.
10. Decrease n in one step i.e. $n=n-1$.
11. end for.
12. Display the solution as $p(1,n)$.

SIMULATION CODE:

```
% Neville's Method%

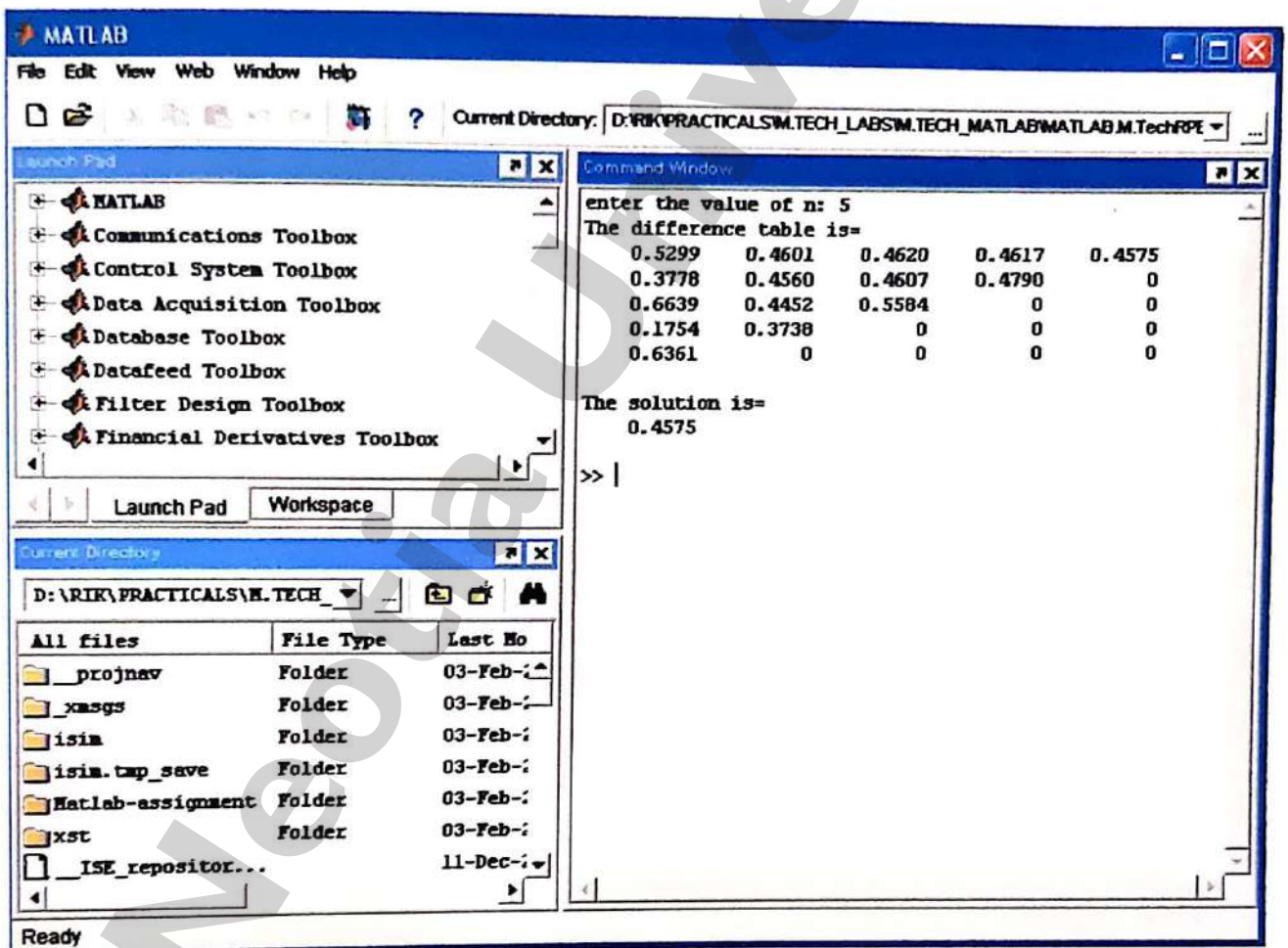
clear all;
n=input('enter the value of n:');
nn=n;
xx=27.5;
x=[32 22.2 41.6 10.1 50.5]';
p=zeros(5,5);
p(1:5,1)=[0.52992 0.37784 0.66393 0.17537 0.63608]';
for(j=2:5)
    for(i=1:nn-1)
        
$$p(i,j)=((xx-x(i))*p(i+1,j-1)+(x(i+j-1)-xx)*p(i,j-1))/(x(i+j-1)-x(i));$$

    end;
    nn=nn-1;
end;

disp('The difference table is=');
disp(p);

disp('The solution is=');
disp(p(1,n));
```

RESULT AFTER SIMULATION:



ALGORITHM:

1. Initialize no of iterations as 'n', which is equal to the no of x_i 's.
2. Initialize the point $m=3.0$ around which solution of the polynomial is desired to calculate.
3. Initialize a row vector x to take the x_i 's as inputs.
4. Initialize a column vector v to take the f_i 's as inputs.
5. Initialize a 5×4 matrix 'a' with zeros such as it's first column becomes v .
6. for j loop 2 to n in steps of 1
7. for i loop 1 to n-1 in steps of 1
8. Coefficients i.e. a_i 's are calculated as $A(i, j) = \frac{A(i+1, j-1) - A(i, j-1)}{x(i+j-1) - x(i)}$
9. end for
10. $n=n-1$.
11. end for
12. Display the difference table A.
13. Initialize another column vector M to take only the different elements of the first row of the matrix A.
14. sum=0;
15. product=1;
16. for j loop 1 to n in steps of 1
17. for j loop 1 to n in steps of 1
18. product=product*($m-x(i)$)
19. $P=\text{product} * M(j)$;
20. sum=sum+P;
21. end for
22. end for
23. sum=sum+A(1,1)
24. Display sum as the final solution.

SIMULATION CODE:

```
% Divided Difference%
clear all;
m=3;
n=5;
x=[3.2 2.7 1.0 4.8 5.6];
a=zeros(5,4);
v=[22 17.8 14.2 38.3 51.7]';
A=[v a];
```

DIVIDED DIFFERENCE INTERPOLATION

Problem: Find the value of a polynomial by the Divided Difference method of interpolation in between the following points, where the function values at respective points are given.

x	f(x)
3.2	22.0
2.7	17.8
1.0	14.2
4.8	38.3
5.6	51.7

PROCEDURE:

If at the points $x_0, x_1, x_2, \dots, x_n$ the corresponding function values are $f_0, f_1, f_2, \dots, f_n$. Then the fitted polynomial of x through these x_i 's is given by

$$P_n(x) = a_0 + (x - x_0) * a_1 + (x - x_0)(x - x_1) * a_2 + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1}) * a_n$$

Let f_i be the value of the polynomial at x_i . Then,

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} = f_0^{[1]}$$

$$f[x_1, x_2] = \frac{f_2 - f_1}{x_2 - x_1} = f_1^{[1]}$$

$$\text{In general way, } f[x_s, x_t] = \frac{f_t - f_s}{x_t - x_s} = f_s^{[1]}$$

$$\text{Similarly, } f[x_t, x_s] = \frac{f_s - f_t}{x_s - x_t} = f_t^{[1]}$$

Thus, $f[x_s, x_t] = f[x_t, x_s]$, where '[1]' denotes the first order of the divided difference.

Similarly, for second order of the divided difference

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f_0^{[2]}$$

$$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} = f_1^{[2]}$$

Generalizing in this way for n^{th} order the polynomial becomes,

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0}$$

Now it is desired to find the values of the coefficients a_i 's i.e. a_0, a_1, \dots, a_n .

Since the polynomial will satisfy the points x_i 's, thus,

At, $x = x_0$

$$P_n(x_0) = a_0 = f_0$$

At, $x = x_1$

$$P_n(x_1) = a_0 + (x_1 - x_0)a_1$$

$$\text{or, } a_1 = \frac{f_1 - a_0}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0} = f_0^{[1]}$$

At, $x = x_2$

$$P_n(x_2) = a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 = f_2$$

After doing some manipulation,

$$\begin{aligned} a_2 &= \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} \\ &= \frac{f_1^{[1]} - f_0^{[1]}}{x_2 - x_0} \\ &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= f[x_0, x_1, x_2] = f_0^{[2]} \end{aligned}$$

Similarly, $a_3 = f_0^{[3]}$

$$\text{Similarly, } a_n = \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0} = f_0^{[n]}$$

Thus the process continues with finding all these coefficients and then computes the functional value at given point x , from the fitted polynomial curve. The difference table becomes,

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4}]$
x_0	f_0	$f[x_0, x_1] = a_1$	$f[x_0, x_1, x_2] = a_2$	$f[x_0, x_1, x_2, x_3] = a_3$	$f[x_0, x_1, x_2, x_3, x_4] = a_4$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	
x_2	f_2	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$		
x_3	f_3	$f[x_3, x_4]$			
x_4	f_4				
x_5	f_5				

```

for j=2:n
    for i=1:n-1
         $A(i,j) = (A(i+1,j-1) - A(i,j-1)) / (x(i+j-1) - x(i));$ 
    end
    n=n-1;
end;

disp('The Difference Table becomes=');
disp(A);
M=[ A(1,2) A(1,3) A(1,4) A(1,5)];
sum=0;
product=1;

    for(i=1:1:n)

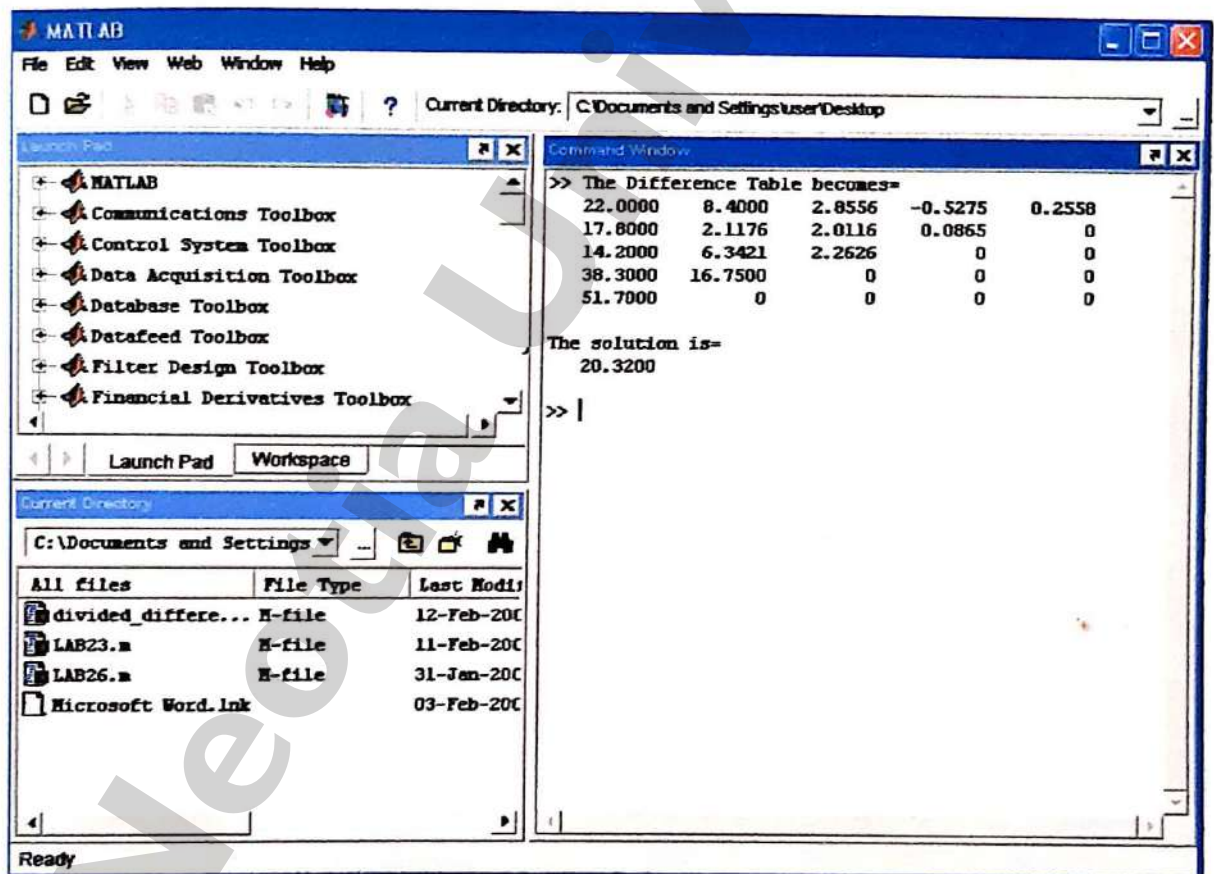
        for(j=1:1:n)

            product=product*(m-x(i));
            P=product*M(j);
            sum=sum+P;
        end;
    end;

sum=sum+A(1,1);
disp('The solution is=');
disp(sum);

```

RESULT AFTER SIMULATION:



THE METHOD OF SUCCESSIVE BISECTION

Problem: To find the root of the equation $f(x) = x^3 + x^2 - 3x - 3 = 0$ using Method of Successive Bisection

PROCEDURE:

This method begins with an iterative cycle by picking two trial points which enclose the root. Two points x_0 and x_1 enclose a root if $f(x_0)$ and $f(x_1)$ are of opposite signs. The method then

bisects the interval $(x_0 \text{ and } x_1)$ and denotes the midpoint x_2 as, $x_2 = \frac{x_0 + x_1}{2}$. If $f(x_2) = 0$ then we

have a root. However, if $f(x_2) > 0$ (from fig 1a) then the root is between x_0 and x_2 . Now the method replaces x_1 by x_2 and searches for the root in this new interval which is half of the previous interval. This again calculates $f(x_2)$ at the midpoint of this new interval (from fig 1b). If this time $f(x_2) < 0$, then the root lies between x_2 and x_1 . Then again x_0 is replaced by x_2 and again the iterative procedure repeats. By repeating in this way the search procedure always encloses the root in the search interval and the search interval is halved in each iteration. The method stops the iterative cycle as soon as the search interval becomes smaller than the pre-assumed precision.

ILLUSTRATING FIGURES:

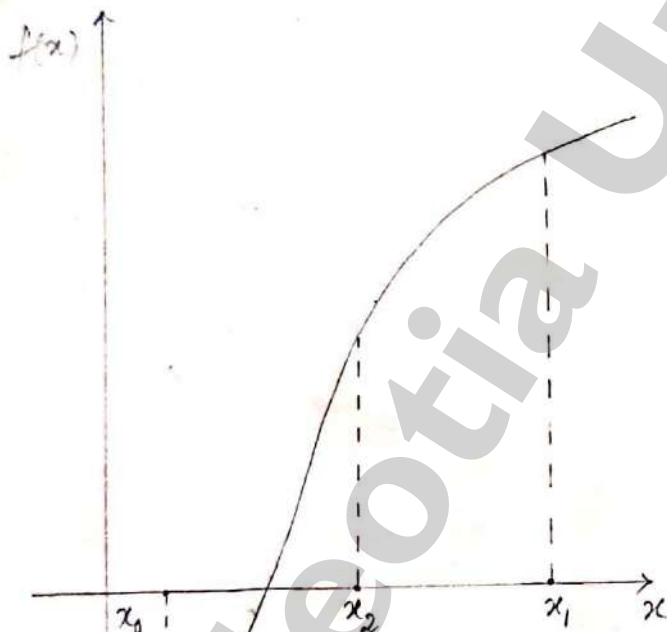


fig - 1a

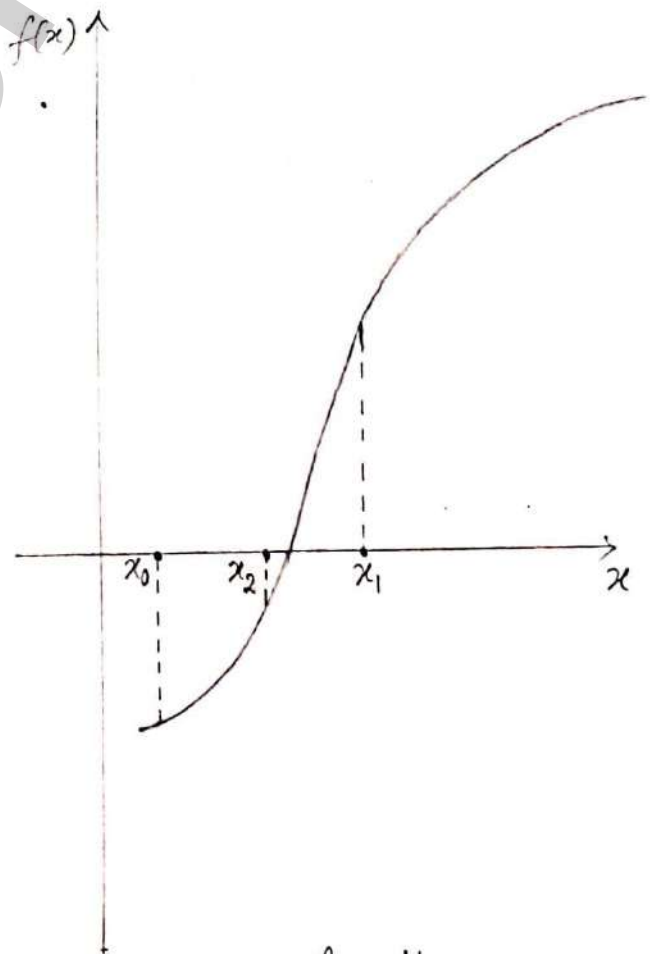


fig - 1b

ALGORITHM:

1. Read x_0, x_1, e

Remarks: x_0 and x_1 are two starting values which enclose the desired root, e is the error allowed in the answer. Steps 2, 3 and 4 are called initialization steps which prepare the ground for the succeeding steps in the algorithm.

2. $y_0 \leftarrow f(x_0)$
3. $y_1 \leftarrow f(x_1)$
4. $i \leftarrow 0$
5. if (sign (y_0) = sign (y_1)) then

Begin Write 'Starting Values unsuitable'.

Write x_0, x_1, y_0, y_1

Stop end

6. While $\left| \frac{(x_1 - x_0)}{x_1} \right| > e$ do

Begin

7. $x_2 \leftarrow \frac{(x_0 + x_1)}{2}$
8. $y_2 \leftarrow f(x_2)$
- 9.
10. $i \leftarrow i + 1$
11. if (sign (y_0) = sign (y_2)) then
12. $x_0 \leftarrow x_2$
13. else
14. $x_1 \leftarrow x_2$
15. end
16. Write 'Solution Converges to a root'
17. Write 'Number of Iterations =', i
18. Write x_2, y_2
19. Stop

SIMULATION CODE:

```
% Bisection Theorem%
```

```
x0= input ('Enter the value of x0');
```

```

x1= input ('Enter the value of x1');
e= input ('Enter the value of e');
y0= x0^3+x0^2-3*x0-3;
y1= x1^3+x1^2-3*x1-3;
i=0;
if ((y0*y1) <= 0)

    while abs((x1-x0)/x1) > e

        x2 = ((x0+x1)/2);
        y2 = x2^3+x2^2-3*x2-3;
        i = i+1;

        if (y0*y2)<0

            x0 = x0;
            x1= x2;
            end;

        if (y1*y2)<0

            x0 = x2;
            x1 = x1;
            end;

        end;
        disp(x2);
    else
        disp('wrong input');
    end;
end;

```

RESULT AFTER SIMULATION:

SET I	SET II	SET II
Enter the value of x0 0 Enter the value of x1 2 Enter the value of e 0.0001	Enter the value of x0 -1.2 Enter the value of x1 0.8 Enter the value of e 0.0001	Enter the value of x0 2 Enter the value of x1 3 Enter the value of e 0.0001
Output: 1.7321	Output: -1.0000	Output: wrong input

SECANT METHOD

Problem: To find the root of the equation $f(x) = x^3 + x^2 - 3x - 3 = 0$ using SECANT Method

PROCEDURE:

In this method $f'(x)$ is approximated by the expression:

$$f'(x) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \quad (1)$$

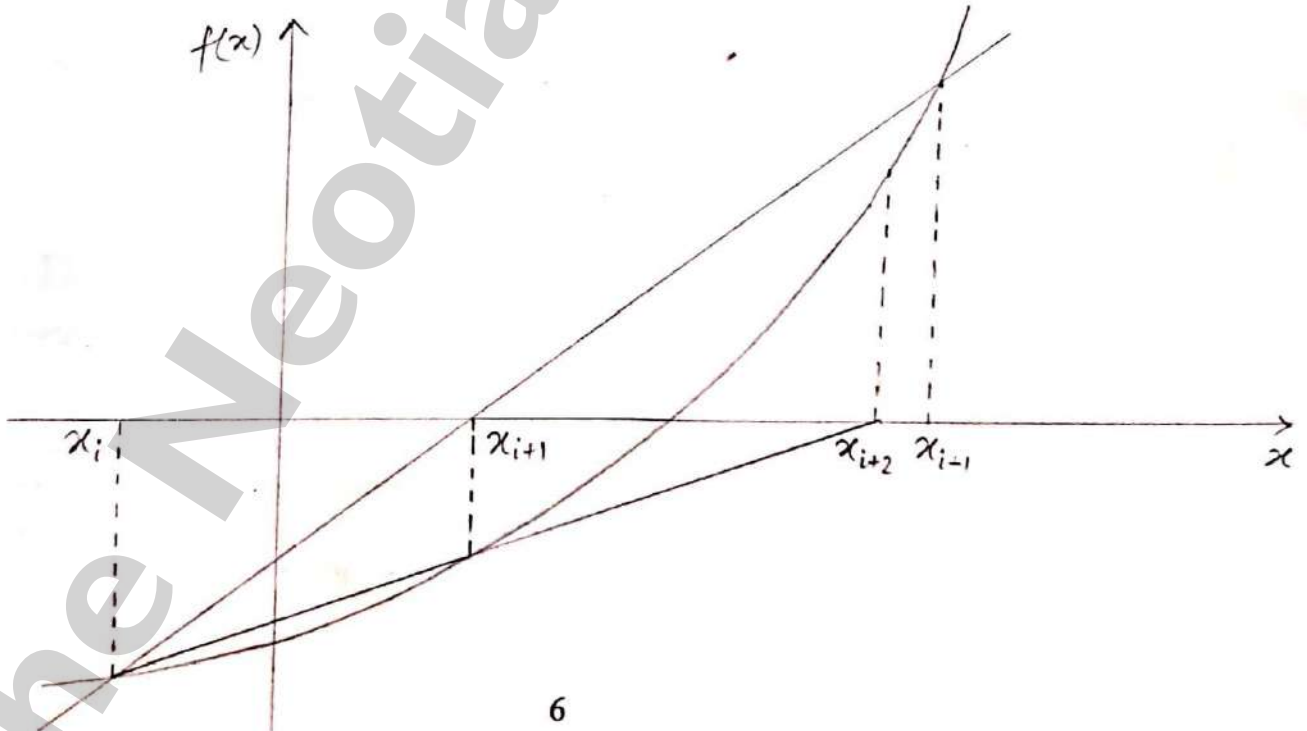
Where x_i and x_{i-1} are two approximations to the root. The iterative formula to go from the i th to the $(i+1)$ th approximation is given by,

$$x_{i+1} = x_i - \frac{f(x_i)}{(f(x_i) - f(x_{i-1})) / (x_i - x_{i-1}))} \quad (2)$$

A geometrical interpretation of the method is explained by the figure below. First a secant is drawn connecting $f(x_{i-1})$ and $f(x_i)$. The point which it cuts the x -axis is x_{i+1} . Another secant is drawn through $f(x_i)$ and $f(x_{i+1})$ to get x_{i+2} . This is repeated till $f(x) \approx 0$. For computation the suitable iterative formula becomes

$$x_{i+1} = \frac{x_{i-1}f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} \quad (3)$$

ILLUSTRATING FIGURE:



6

fig - 2

ALGORITHM:

1. Read $x_0, x_1, e, \text{delta}, n$

Remarks: x_0, x_1 , are two initial guesses to the root. e is the prescribed precision, delta the minimum allowed value of slope and n the maximum number of iterations to be permitted.

2. $f_0 \leftarrow f(x_0)$

3. $f_1 \leftarrow f(x_1)$

Remarks: statements 5-12 are repeated until the procedure converges to a root or iterations exceed n .

4. for $i = 1$ to n in steps of 1 do

5. if $|f_1 - f_0| < \text{delta}$ then GOTO step 15

6. $x_2 \leftarrow (x_0 \cdot f_1 - x_1 \cdot f_0) / (f_1 - f_0)$

7. $f_2 \leftarrow f(x_2)$

8. if $|f_2| < e$ then GOTO step 17

9. $f_0 \leftarrow f_1$

10. $f_1 \leftarrow f_2$

11. $x_0 \leftarrow x_1$

12. $x_1 \leftarrow x_2$

endfor

13. Write 'Does not Converge', x_0, x_1, f_0, f_1

14. Stop

15. Write 'Slope too small', i, f_0, f_1, x_0, x_1

- 16.

17. Stop

18. Write 'Convergent Solution', i, x_2, f_2

19. Stop

SIMULATION CODE:

% Secant Method%

$x_0 = \text{input}(\text{'Enter the value of } x_0\text{'})$;

$x_1 = \text{input}(\text{'Enter the value of } x_1\text{'})$;

$e = \text{input}(\text{'Enter the value of precision=e'})$;

$\text{delta} = \text{input}(\text{'Enter the value of minimum slope=delta'})$;

$n = \text{input}(\text{'Enter the value of maximum iterations=n'})$;

$f_0 = x_0^3 - 3 \cdot x_0^2 + x_0 + 1$;

```

f1=x1^3-3*x1^2+x1+1;
for(i=1:1:n)
    if abs(f1-f0)< delta
        disp('Slope is too small');
    else
        x2=((x0*f1-x1*f0)/(f1-f0));
        f2=x2^3-3*x2^2+x2+1;
        if abs(f2)< e
            disp('Convergent Solution');
            disp(x2);
        else
            f0=f1;
            f1=f2;
            x0=x1;
            x1=x2;
        end;
    end;
end;

```

RESULT AFTER SIMULATION:

SET I:	SET II:
Enter the value of x0 0 Enter the value of x1 2 Enter the value of precision=e 0.001 Enter the value of minimum slope=delta 0.0001 Enter the value of maximum iterations=n 10	Enter the value of x0 -2.5 Enter the value of x1 -1.3 Enter the value of precision=e 0.001 Enter the value of minimum slope=delta 0.0001 Enter the value of maximum iterations=n 15
Output: Convergent Solution 1.0	Output: Convergent Solution -0.4142

METHOD OF FALSE POSITION

Problem: To find the root of the equation $f(x) = x^2 - 49 = 0$ using Method of False Position.

PROCEDURE:

The procedure of the method of false position or Regula Falsi starts by locating two points x_0 and x_1 where the functions have opposite signs. Now two points are connected by a straight line and then the point where it cuts the x-axis has been located. Let the intersection point be x_2 . $f(x_2)$ is computed then. If $f(x_2)$ and $f(x_0)$ are of opposite signs then x_1 is replaced by x_2 and a straight line is drawn connecting $f(x_2)$ to $f(x_0)$ to find the new intersection point. If $f(x_2)$ and $f(x_1)$ are of same sign then x_0 is replaced by x_2 and the procedure proceeds as before. In both cases the new interval of search is smaller than the initial interval and ultimately it is guaranteed to converge to the root.

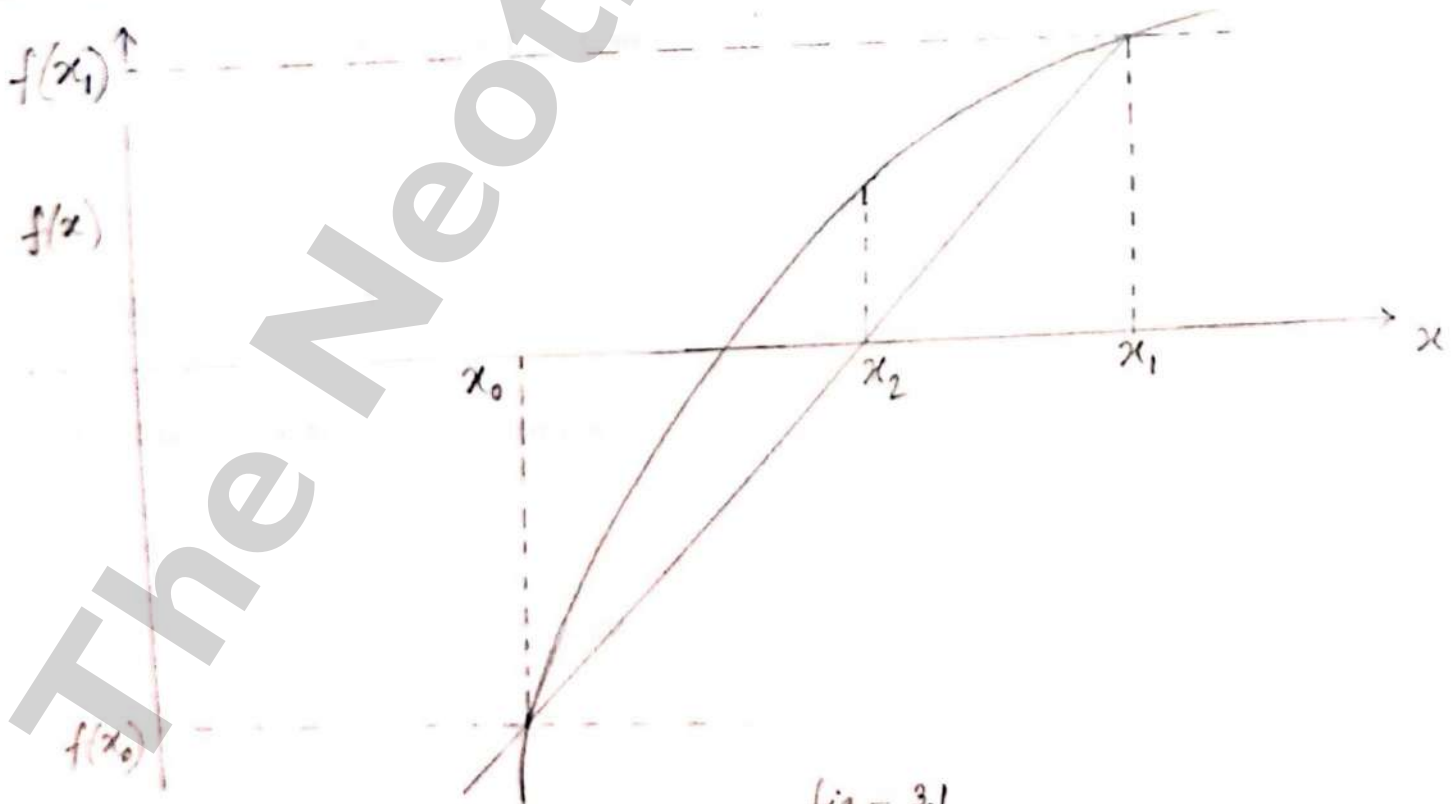
The equation to find successive approximations from the figure (3.1) becomes,

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \tan \theta = \frac{f(x_1)}{x_1 - x_2} \quad (1)$$

$$\text{Or, } x_1 - x_2 = \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

$$\text{Or, } x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)} \quad (2)$$

ILLUSTRATING FIGURE:



ALGORITHM:

1. Read x_0, x_1, e, n

Remarks: x_0 and x_1 are two initial guesses to the root such that $\text{sign}(f(x_0)) \neq \text{sign}(f(x_1))$. The prescribed precision is e and n is the maximum number of iterations. Step 2 and step 3 are initialization steps.

2. $f_0 \leftarrow f(x_0)$
3. $f_1 \leftarrow f(x_1)$
4. for $i = 1$ to n in steps of 1 do
5. $x_2 \leftarrow \frac{(x_0 f_1 - x_1 f_0)}{(f_1 - f_0)}$
6. $f_2 \leftarrow f(x_2)$
7. if $|f_2| \leq e$ then
8. begin Write 'Convergent Solution', x_2, f_2
9. Stop end
10. if $\text{sign}(f_2) \neq \text{sign}(f_0)$
11. then begin $x_1 \leftarrow x_2$
12. $f_1 \leftarrow f_2$ end
13. else begin $x_0 \leftarrow x_2$
14. $f_0 \leftarrow f_2$ end
- endfor
15. Write 'Does not Converge in n iterations'
16. Write x_2, f_2 .
17. Stop

SIMULATION CODE:

% Regula Falsi or Method of False Position%

```
x0= input('Enter the value of x0');
x1= input('Enter the value of x1');
e= input('Enter the value of precision=e');
n= input('Enter the value of maximum iterations=n');
f0=x0^2-53.29;
f1=x1^2-53.29;

for(i=1:1:n)
```

```

x2=((x0*f1-x1*f0)/(f1-f0));
f2=x2^2-53.29;

if abs(f2)< e
    disp('Convergent Solution');
    disp(x2);
end;

if (abs(f2)== abs(f0))
    x0=x2;
    f0=f2;
else
    x1=x2;
    f1=f2;
end;

end;

```

RESULT AFTER SIMULATION:

SET I:	SET II:
Enter the value of x0 5 Enter the value of x1 9 Enter the value of precision=e 0.001 Enter the value of maximum iterations=n 10	Enter the value of x0 -9 Enter the value of x1 -5 Enter the value of precision=e 0.001 Enter the value of maximum iterations=n 10
Output: Convergent Solution 7.00	Output: Convergent Solution -7.00

NEWTON-RAPHSON ITERATIVE METHOD

**Problem: To find the root of the equation $f(x) = x^3 + x^2 - 3x - 3 = 0$ using
NEWTON-RAPHSON Iterative Method**

PROCEDURE:

Considering a curve $f(x)$ given in the following figure, starting from an arbitrary point x_0 the root can be reached by using the following procedure.

First the slope of $f(x)$ is determined at $x = x_0$ i.e. $f'(x_0)$. Then the next approximation x_1 is picked up to the root by using the equation

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

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

In general from i th to $(i+1)$ th approximation

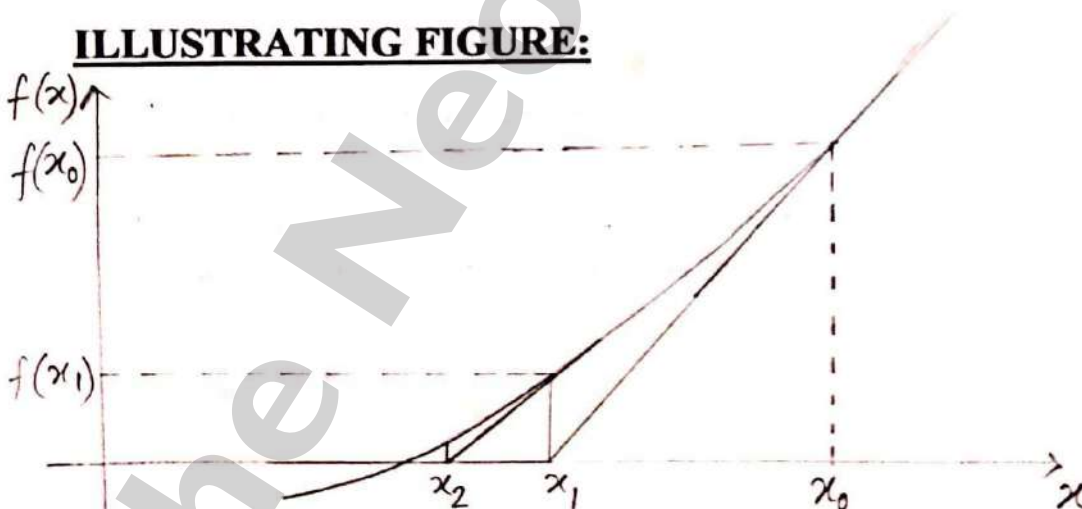
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Assuming that $f'(x_i)$ is never zero for any x_i .

Iterative cycle is stopped when two successive values x_i of are nearly equal within a prescribed tolerance.

First a tangent to the curve $f(x)$ is drawn at $x = x_0$ and the point where it cuts the x -axis is taken as the next approximation. The same procedure is repeated till $f(x_i)$ becomes nearly zero which in turn implies that x_{i+1} and x_i are almost equal.

ILLUSTRATING FIGURE:



19

fig - 5

ALGORITHM:

9. Read x_0 , epsilon, delta, n.

Remarks: x_0 is the initial guess, epsilon is the prescribed relative error, delta is the prescribed lower bound for f' and n the maximum number of iterations to be allowed. Statement 3 to 8 is repeated until the procedure converges to a root or iterations equal n.

10. For $i=1$ to n in steps of 1 do

11. $f_0 \leftarrow f(x_0)$

12. $f'_0 \leftarrow f'(x_0)$

13. if $f'_0 < \text{delta}$ then go to step 11

14. $x_1 \leftarrow x_0 - \frac{f_0}{f'_0}$

15. if $\left| \frac{x_1 - x_0}{x_1} \right| < \text{epsilon}$ then go to step 13

16. $x_0 \leftarrow x_1$
end for

17. Write 'does not converge in n iterations', f_0, f'_0, x_0, x_1

18. Stop

19. Write 'Slope too small', f_0, f'_0, x_0, i

20. Stop

21. Write 'Convergent solution', $x_1, f(x_1), i$

22. Stop

SIMULATION CODE:

% Newton-Raphson Method%

x_0 = input ('Enter the value of x_0 =');

epsilon = input ('Enter the value of epsilon=');

delta = input ('Enter the value of delta=');

n = input ('Enter the value of n=');

for (i=1:n)

```

f0= x0^2-25;
fdiff= 2*x0;
if abs(fdiff)<= delta
    disp ('Slope too small');
else
    x1=x0-(f0/abs(fdiff));
    if abs((x1-x0)/x1)< epsilon
        disp('Convergent solution');
        disp(x1);
    end;
    x0=x1;
end;
end;
end;

```

RESULT AFTER SIMULATION:

SET I	SET II	SET III
Enter the value of x0=6.8 Enter the value of epsilon=0.0001 Enter the value of delta=0.001 Enter the value of n=10	Enter the value of x0=-5 Enter the value of epsilon=0.0001 Enter the value of delta=0.001 Enter the value of n=10	Enter the value of x0=3.4 Enter the value of epsilon=0.0001 Enter the value of delta=0.001 Enter the value of n=10
Output: 5.0000	Output: -5.0000	Output: 5.0000

GAUSS ELIMINATION METHOD

Problem: Find the solution of the set of equations such as:

$$\begin{aligned}4x_{11} - 2x_{12} + x_{13} &= 15 \\-3x_{21} - x_{22} + 4x_{23} &= 8 \\x_{31} - x_{32} + 3x_{33} &= 13\end{aligned}$$

PROCEDURE:

This method is a direct method which consists of transforming the given system of simultaneous equations to an equivalent upper triangular system. From this transformed system the required solution can be obtained.

Consider the system of n equations in n unknowns given by $Ax = B$

$$\text{Or, } \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (1)$$

To transform the system to an equivalent upper triangular system the following operations are considered:

1. The row operation $R_i \rightarrow R_i - \frac{a_{i1}}{a_{11}} R_1, i = 2, 3, \dots, n$. Making all the entries in the first column zero the first equation obtained is called pivotal equation. $a_{11} \neq 0$, is called pivot and $\frac{-a_{i1}}{a_{11}}$ for $i = 2, 3, \dots, n$ are called the multipliers for the first elimination. If $a_{11} = 0$, then the first has to be interchanged with another suitable row so as to have $a_{11} \neq 0$.
2. Next is $R_i \rightarrow R_i - \frac{a_{i2}}{a_{22}} R_2, i = 3, 4, \dots, n$ which makes all entries $a_{32}, a_{42}, \dots, a_{n2}$ in the second column zero.
3. In general the row operation becomes $R_i \rightarrow R_i - \frac{a_{ik}}{a_{kk}} R_k, i = k+1, k+2, \dots, n$ which make all the entries $a_{k+1}, a_{k+2}, \dots, a_{nk}$ in the k^{th} column zero.
4. Hence the given system of equations is reduced to the form $DX = U$ where U is the upper triangular matrix. The required solution x then can be obtained by $X = U \setminus D$

ALGORITHM:

1. Input the A matrix and B matrix together in a C matrix with n no of rows and m no. of columns.
2. for k from 1 to (n-1) in steps of 1
3. for i from (k+1) to n in steps of 1
4. $u = C(i,k)/C(k,k)$
5. for j from k to (n+1) in steps of 1
6. $C(i,j) = C(i,j) - (u * C(k,j))$
7. end for
8. end for
9. end for.
10. upper triangular matrix is $U = C(1:n, 1:m-1)$
11. diagonal matrix is $D = C(:, m)$
12. solution x is obtained by $x = U \backslash D$

SIMULATION CODE:

```
clear all;

n=input('Enter the no. of rows of the matrix:');
m=input('Enter the no. of columns of the matrix:');

C=[4 -2 1 15;-3 -1 4 8;1 -1 3 13];

[n,m]=size(C);
disp('The matrices are');
A=C(1:n,1:m-1)
B=C(:,m)

for(k=1:(n-1))
    for(i=(k+1):1:n)
        u=C(i,k)/C(k,k);
        for(j=k:1:(n+1))
            C(i,j)=C(i,j)-(u*C(k,j));
        end;
    end;
end;

disp('The upper triangular matrix is-');
U=C(1:n,1:m-1)
D=C(:,m);
disp('The solutions of the given equations are-');
x=U \ D
```

RESULT AFTER SIMULATION:

```
Command Window
File Edit View Web Window Help

A =
     4     -2      1
    -3     -1      4
     1     -1      3

B =
    15
     8
    13

The upper triangular matrix is-

U =
     4.0000    -2.0000     1.0000
         0    -2.5000     4.7500
         0         0     1.8000

The solutions of the given equations are-

x =
     2.0000
    -2.0000
     3.0000

>>

Ready
```

GAUSS-JORDAN ELIMINATION METHOD

Problem: Find the solution of the set of equations such as:

$$6x_1 - 2x_2 + x_3 = 11$$

$$x_1 + 2x_2 - 5x_3 = -1$$

$$-2x_1 + 7x_2 + 2x_3 = 5$$

PROCEDURE:

Let us consider the system of equations $Ax = B$

If A is a diagonal matrix then the given system reduces to

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & & 0 \\ & & \ddots & \\ 0 & & & a_{nn} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

This system reduces to following n equations with their solutions directly.

$$a_{11}x_1 = b_1$$

$$\text{or, } x_1 = \frac{b_1}{a_{11}}$$

$$a_{nn}x_n = b_n$$

$$\text{or, } x_n = \frac{b_n}{a_{nn}}$$

The method of obtaining the solution of the system of equations by reducing the matrix A to a diagonal matrix is known as Gauss-Jordan elimination method.

1. Firstly the system is rearranged in such a fashion that the diagonal elements of A magnitude that are greater than the sum of the remaining elements in the corresponding

$$\text{row i.e. } |a_{ij}| = \sum_{\substack{i=1 \\ j \neq i}}^n |a_{ij}|, i = 1, 2, \dots, n$$

2. Then the lower and upper triangular forms of the matrix A is obtained by LU Decomposition method and stored in different matrices.
3. Remaining diagonal elements are arranged in another matrix setting rest of the elements to be zero, which will form the diagonal matrix.
4. From that diagonal matrix the solutions for the system of equations can be obtained.

SIMULATION CODE:

```
clear all;
A=[6 -2 1 11;1 2 -5 -1;-2 7 2 5];
C=A(:,1:3);
b=A(:,4);

for(i=1:3)
    x0(i)=0;
end;

for(i=1:2)
    sum=0;
    for(j=1:3)
        if(i~=j)
            sum=sum+abs(C(i,j));
        end;
    end;
    if(abs(C(i,i)<sum))
        for(k=i+1:3)
            if(abs(C(i,i)<abs(C(k,i))))
                for(j=1:4)
                    D=A(i,j);
                    A(i,j)=A(k,j);
                    A(k,j)=D;
                end;
            end;
        end;
    end;
end;

C=A(:,1:3)
b=A(:,4)

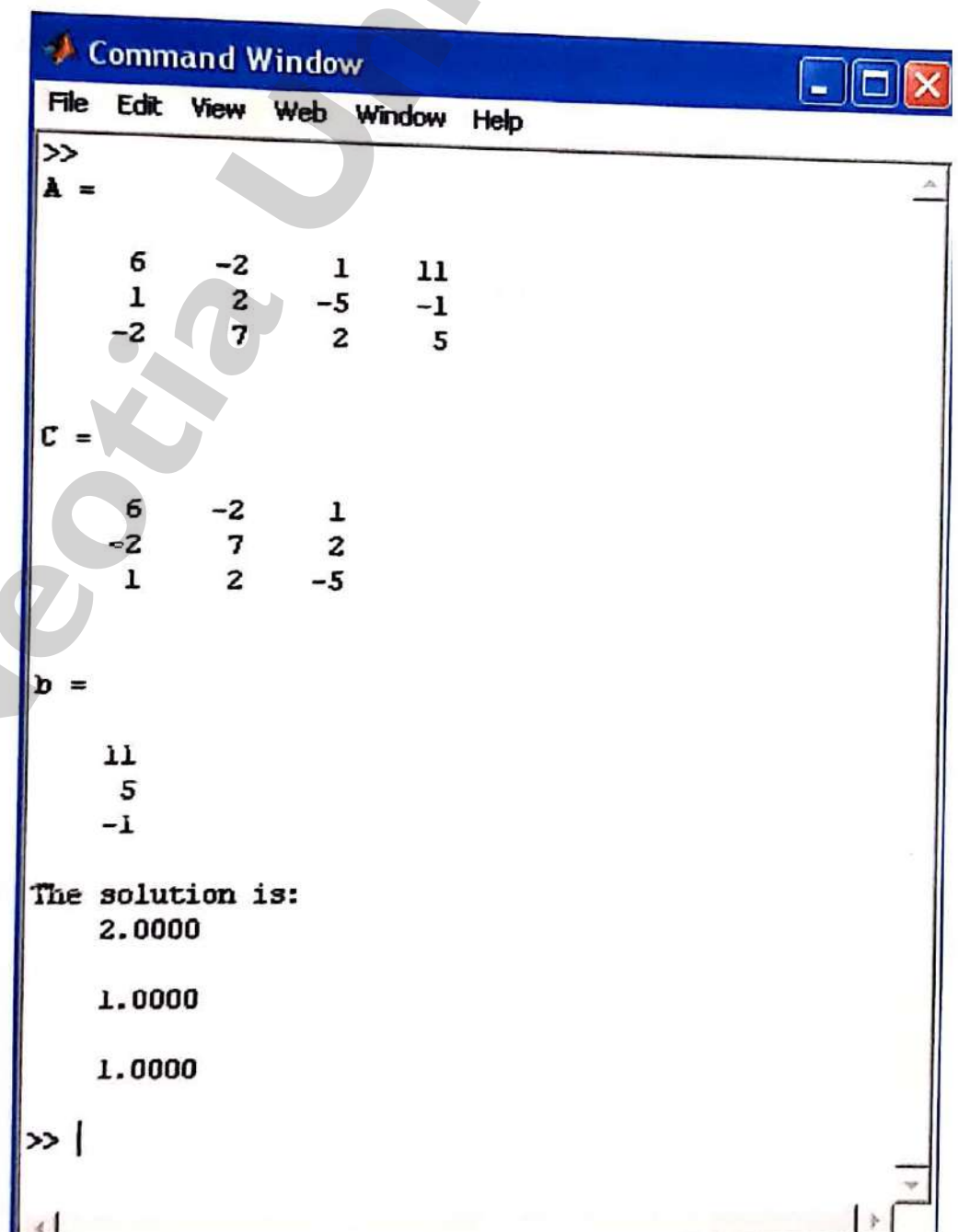
for(k=1:100)
    for(i=1:3)
        E(i)=0;
        for(j=1:3)
            if(i~=j)
                E(i)=E(i)-(C(i,j)*x0(j));
            end;
        end;
        x(i)=(b(i)+E(i))/C(i,i);
    end;
    for(i=1:3)
        if(abs(x(i)-x0(i))<0.00001)
            break;
        end;
    end;
end;
```

```

end;
for(i=1:3)
    x0(i)=x(i);
end;
end;
end;
disp('The solution is:');
for(i=1:3)
    disp( x(i));
end;

```

RESULT AFTER SIMULATION:



The screenshot shows the MATLAB Command Window with the following content:

```

>>
A =
     6     -2     1    11
     1      2    -5     -1
    -2      7      2      5

C =
     6     -2      1
    -2      7      2
     1      2     -5

b =
    11
     5
    -1

The solution is:
    2.0000

    1.0000

    1.0000

>> |

```

GAUSS-SEIDEL ITERATIVE METHOD

Problem: Find the solution of the set of equations such as:

$$6x_1 - 2x_2 + x_3 = 11$$

$$x_1 + 2x_2 - 5x_3 = -1$$

$$-2x_1 + 7x_2 + 2x_3 = 5$$

PROCEDURE:

Let us consider the set of n simultaneous equations in n unknowns

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = a_{1n+1} \quad (1)$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = a_{2n+1} \quad (2)$$

$$a_{31}x_1 + a_{32}x_2 + \dots + a_{3n}x_n = a_{3n+1} \quad (3)$$

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = a_{nn+1} \quad (4)$$

In the Gauss-Seidel method x_2^0, \dots, x_n^0 are set equal to zero and x_1^1 is calculated from equation (1). This value of x_1 and zeros are substituted for x_3, x_4, \dots, x_n in equation (2). The values x_1^1 and x_2^1 along with zeros for x_4, x_5, \dots, x_n are used in equation (3) to compute x_3^1 . Finally, $x_1^1, x_2^1, x_3^1, \dots, x_{n-1}^1$ are used in equation (4) to compute x_n^1 . With this the first iteration ends. In the second iteration $x_2^1, x_3^1, x_4^1, \dots, x_n^1$ are used to compute x_1^2 . $x_1^2, x_3^1, \dots, x_n^1$ are used to get x_2^2 and so on. In this method only the latest approximations for the values of variables are used in iteration.

ALGORITHM:

1. for $i = 1$ to n in steps of 1 and $j = 1$ to $n+1$

2. in steps of 1 do Read a_{ij} end for

3. Read e, m

Remarks: e is the allowed iterative error and m is the maximum number of iterations allowed for the solution to converge.

4. for $i = 1$ to n in steps of 1 do $x_i \leftarrow 0$ end for

Remarks: all x_i s are initialized to zero.

5. for iteration = 1 to m do

6. big $\leftarrow 0$

7. for $i = 1$ to n in steps of 1 do

8. sum $\leftarrow 0$

9. for $j = 1$ to n in steps of 1 do

10. if ($j \neq i$) then sum \leftarrow sum + $a_{ij}x_j$ end for

11. temp $\leftarrow (a_{in+1} - \text{sum}) / a_{ii}$

12. reerror $\leftarrow |(x_i - \text{temp}) / \text{temp}|$

13. if reerror $>$ big then big \leftarrow reerror

14. $x_i \leftarrow \text{temp}$
15. end for
16. if ($\text{big} \leq e$) then
17. begin Write 'Converges to a solution'
18. for $i = 1$ to n in steps of 1 do Write x_i end for.
19. end for
20. Write 'Does not converge in m iterations'.
21. for $i = 1$ to n in steps of 1 do Write x_i end for.
22. stop.

SIMULATION CODE:

```

clear all;

C=[6 -2 1 11; 1 2 -5 -1; -2 7 2 5];

A=C(:,1:3);

for(i=1:2)
    sum=0;
    for(j=1:3)
        if(i~=j)
            sum=sum+abs(A(i,j));
        end
    end
    if(abs(A(i,i)<sum))
        for(k=i+1:3)
            if(abs(A(i,i))<abs(A(k,i)))
                for(j=1:4)
                    D=C(i,j);
                    C(i,j)=C(k,j);
                    C(k,j)=D;
                end
            end
        end
    end
end
end
end
end

A=C(:,1:3)
b=C(:,4)

L=zeros(3,3);

```

```

U=zeros(3,3);
D=zeros(3,3);

d=diag(A);
D=diag(d);
for(i=1:3)
    for(j=i+1:3)
        U(i,j)=A(i,j);
    end
end

for(i=2:3)
    for(j=1:i-1)
        L(i,j)=A(i,j);
    end
end

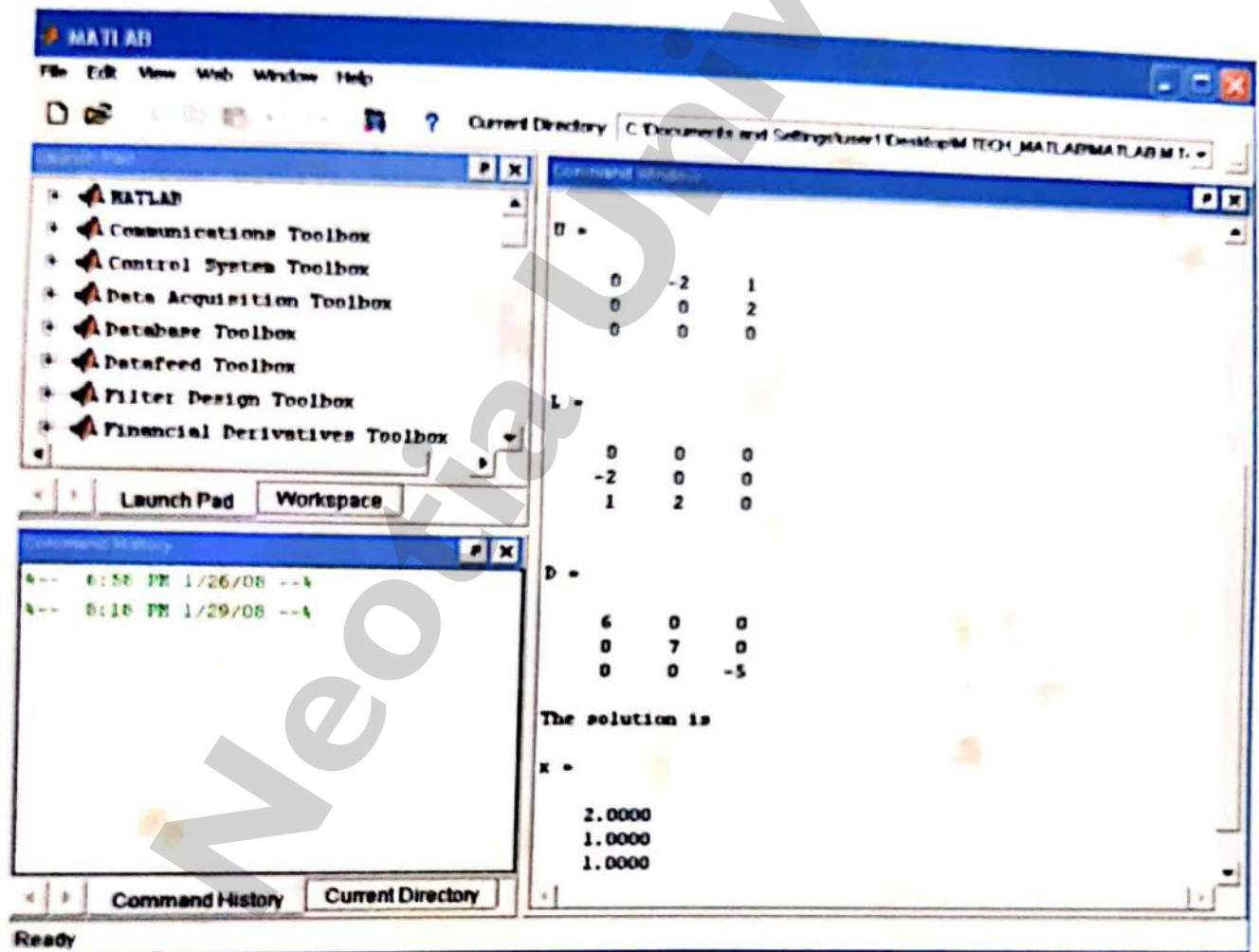
for(i=1:3)
    inverseD(i,i)=1/(D(i,i));
end

x=[0 0 0]';
for(i=1:20)
    x1=(inverseD*b)-(inverseD*(L+U)*x);
    if((x1-x)<0.00001)
        break
    end
    x=x1;
end

U
L
D
disp('The solution is');
x=A\b

```

RESULT AFTER SIMULATION:



TRAPEZOIDAL RULE FOR NUMERICAL INTEGRATION

Problem: Find the value of a given function by the Trapezoidal Rule of integration in between the points $x = 1.8$ and $x = 3.4$ where the value of the functions at different points are as follows:

x	f(x)
1.6	4.953
1.8	6.050
2.0	7.389
2.2	9.025
2.4	11.023
2.6	13.464
2.8	16.445
3.0	20.086
3.2	24.533
3.4	29.964
3.6	36.598
3.8	44.701

Newton-Kotes Formula:

In a region if x has equidistant values then the integration of a polynomial in the range a to b is given as,

$\int_a^b P_n(x_s) \cdot ds$, where the polynomial is given as,

$$P_n(x) = f_0 + s \cdot \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_0 + \dots$$

$$\text{Here, } s = \frac{(x - x_0)}{h}$$

$$\text{Or, } h \cdot ds = dx$$

Now, $P_n(x_s)$ is converted from x to s .

Let, $n=1$, then $i=0, 1$: it becomes a linear polynomial. Let the limits of integration are x_0 and x_1 ,

$$\begin{aligned} \int_{x_0}^{x_1} f(x) \cdot dx &= \int_{x_0}^{x_1} [f_0 + s \cdot \Delta f_0] \cdot dx \\ &= h \int_0^1 [f_0 + s \Delta f_0] \cdot ds \\ &= h f_0 [s]_0^1 + h \Delta f_0 \left[\frac{s^2}{2} \right]_0^1 \\ &= h \cdot f_0 + \frac{h \cdot \Delta f_0}{2} \\ &= \frac{h}{2} [2f_0 + f_1 - f_0] \\ &= \frac{h}{2} [f_0 + f_1] \end{aligned}$$

Let, $n=2$, then $i=0, 1, 2$: it becomes a quadratic polynomial. Let the limits of integration are x_0, x_1, x_2 .

$$\begin{aligned} \int_{x_0}^{x_2} f(x) \cdot dx &= \int_{x_0}^{x_2} \left[f_0 + s \cdot \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 \right] \cdot dx \\ &= h \int_0^2 \left[f_0 + s \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 \right] \cdot ds \\ &= \frac{h}{3} [6f_0 + 6(f_1 - f_0) + (f_2 - 2f_1 + f_0)] \end{aligned}$$

$$= \frac{h}{3}[f_0 + 4f_1 + f_2]$$

Let, $n=3$, then $i=0, 1, 2, 3$: it becomes a cubical polynomial.

$$\int_{x_0}^{x_3} f(x).dx = \frac{3h}{8}[f_0 + 3f_1 + 3f_2 + f_3]$$

PROCEDURE:

Let there are n intervals. At each interval given function is approximated linearly i.e. x_1 and x_2 are connected through a straight line so that Newton-Kotes formula for $n=1$ can be applicable.

Thus in general,

$$\begin{aligned} \int_a^b f(x).dx &= \frac{h}{2} \sum_{i=1}^n (f_i + f_{i+1}) \\ &= \frac{h}{2} [f_1 + 2f_2 + 2f_3 + \dots + 2f_n + f_{n+1}] \end{aligned}$$

ALGORITHM:

1. Enter the values of x as inputs.
2. Enter the functional values $f(x)$ as inputs.
3. Initialize no. of iterations to be performed by the method.
4. Compute $h = x_2 - x_1$
5. Define a variable $P=0$
6. for i loop 2 to n in steps of 1
7. $P = P + 2f(x_i)$
8. end
9. Compute $S = (h/2) * (f(x_1) + P + f(x_{n+1}))$;
10. Display the result of integration stored in S .

SIMULATION CODE:

```
% Trapezoidal Formula%
clear all;
x=[ 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 ];
fx=[ 6.050 7.389 9.025 11.023 13.464 16.445 20.086 24.533 29.964 ];

n=8;
h=(x(2)-x(1));
P=0;
for(i=2:1:n)

    P=P+2*fx(i);

end;

S=(h/2)*(fx(1)+P+fx(n+1));
disp('The Result of Integration is=');
disp(S);
```

RESULT AFTER SIMULATION:

The image shows the MATLAB software interface. The top menu bar includes File, Edit, View, Web, Window, and Help. The current directory is set to D:\RIK\PRACTICALS\SM.TECH_LAB\SM.TECH_MATLAB\Matlab. The Launch Pad window displays a list of toolboxes: MATLAB, Communications Toolbox, Control System Toolbox, Data Acquisition Toolbox, Database Toolbox, Datafeed Toolbox, Filter Design Toolbox, and Financial Derivatives Toolbox. The Command Window shows the output of a simulation: >> The Result of Integration is= 1.3439. The Current Directory window shows a list of files and folders in the directory D:\RIK\PRACTICALS\SM.TECH_LAB\SM.TECH_MATLAB\Matlab. The status bar at the bottom indicates 'Ready'.

Launch Pad

- MATLAB
- Communications Toolbox
- Control System Toolbox
- Data Acquisition Toolbox
- Database Toolbox
- Datafeed Toolbox
- Filter Design Toolbox
- Financial Derivatives Toolbox

Command Window

```
>> The Result of Integration is=
1.3439

>> |
```

Current Directory

D:\RIK\PRACTICALS\SM.TECH_LAB\SM.TECH_MATLAB\Matlab

All files	File Type	Last Mo
Matlab-assignment	Folder	07-Feb-17
Divided_Differe...	M-file	12-Feb-17
lab1.m	M-file	16-Nov-16
Lab10.m	M-file	03-Dec-16
lab11.m	M-file	17-Dec-16
lab12.m	M-file	13-Dec-16
LAB14.M	M-file	17-Dec-16

Ready

SIMPSON'S ONE-THIRD RULE FOR NUMERICAL INTEGRATION

Problem: Find the value of a given function $f(x) = e^{-x^2}$ by the Simpson's one-third Rule of integration in between the points $x = 0.2$ and $x = 1.6$ where the value of the functions at different points are as follows:

x	f(x)
1.6	4.953
1.8	6.050
2.0	7.389
2.2	9.025
2.4	11.023
2.6	13.464
2.8	16.445
3.0	20.086
3.2	24.533
3.4	29.964
3.6	36.598
3.8	44.701

Newton-Kotes Formula:

In a region if x has equidistant values then the integration of a polynomial in the range a to b is given as,

$\int_a^b P_n(x) . ds$, where the polynomial is given as,

$$P_n(x) = f_0 + s.\Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_0 + \dots$$

$$\text{Here, } s = \frac{(x - x_0)}{h}$$

$$\text{Or, } h.ds = dx$$

Now, $P_n(x)$ is converted from x to s .

Let, $n=1$, then $i=0, 1$: it becomes a linear polynomial. Let the limits of integration are x_0 and x_1 .

$$\begin{aligned} \int_{x_0}^{x_1} f(x) . dx &= \int_{x_0}^{x_1} [f_0 + s.\Delta f_0] . dx \\ &= h \int_0^1 [f_0 + s\Delta f_0] . ds \\ &= hf_0[s]_0^1 + h\Delta f_0 \left[\frac{s^2}{2} \right]_0^1 \\ &= h.f_0 + \frac{h.\Delta f_0}{2} \\ &= \frac{h}{2} [2f_0 + f_1 - f_0] \\ &= \frac{h}{2} [f_0 + f_1] \end{aligned}$$

Let, $n=2$, then $i=0, 1, 2$: it becomes a quadratic polynomial. Let the limits of integration are x_0, x_1, x_2 .

$$\begin{aligned} \int_{x_0}^{x_2} f(x) . dx &= \int_{x_0}^{x_2} \left[f_0 + s.\Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 \right] . dx \\ &= h \int_0^2 \left[f_0 + s.\Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 \right] . ds \\ &= \frac{h}{3} [6f_0 + 6(f_1 - f_0) + (f_2 - 2f_1 + f_0)] \end{aligned}$$

$$= \frac{h}{3}[f_0 + 4f_1 + f_2]$$

Let, $n=3$, then $i=0, 1, 2, 3$: it becomes a cubical polynomial.

$$\int_{x_0}^{x_3} f(x) dx = \frac{3h}{8}[f_0 + 3f_1 + 3f_2 + f_3]$$

PROCEDURE:

Let there are 3 points which can be approximated in a quadratic polynomial. For $n=2$, from Newton-Kote's formula,
The function is,

$$f(x) = \frac{h}{3}[f_0 + 4f_1 + f_2]$$

Thus in general,

$$\int_a^b f(x) dx = \frac{h}{3}[f(a) + 4f_1 + 2f_2 + 4f_3 + 2f_4 + 4f_5, \dots + 4f_{n-1} + f(b)] \text{ for } n=\text{even.}$$

ALGORITHM:

1. Enter the values of x as inputs.
2. Initialize no. of iterations n .
3. for i loop 1 to 14 in steps of 1
4. Functional values i.e. $f(x)$ are calculated as $f(x) = e^{-x^2}$ and stored in a vector F .
5. end for.
6. Initialize $S1=S2=0$
7. Compute $h=x_2-x_1$
8. for i loop 2 to $(n-1)$ in steps of 1
9. if modulo division of $n=0$ i.e. n is even
10. Compute $S1=S1+(4*(F(i)))$
11. else i.e. n is odd
12. $S2=S2+(2*(F(i)))$
13. end if
14. end for
15. Compute $P=(h/3)*(S1+S2+F(1)+F(n))$ as final result
16. Display the result.

SIMULATION CODE:

```
%Simpson's 1/3 rule%
clear all;
x=[0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5];
n=14;
for(i=1:1:14)

    F(i)=exp(-x(i)*x(i));

end;
disp(F);
S1=0;
S2=0;
h=x(2)-x(1);
for(i=2:1:n-1)

    if(mod(i,2)==0)

        S1=S1+(4*(F(i)));
    else
        S2=S2+(2*(F(i)));
    end;
end;
P=(h/3)*(S1+S2+F(1)+F(n));
disp('The Result of Integration is=');
disp(P);
```

RESULT AFTER SIMULATION:

