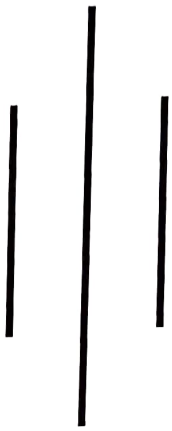
Lab Report

Of

Numerical Method

Subject Code: CSC207



Submitted To

SOCH COLLEGE OF IT

(AFFILATED TO TRIBHUVAN UNIVERSITY)

Ranipauwa, Pokhara-11

Submitted By

Ankit Pangeni

University Registration Number: 5-2-1179-4-2019

College Roll Number: 04

Program: Bachelor of Science in Computer Science and Information Technology (B.Sc.CSIT)

Semester: Third (3rd)

Batch : 2076

List of Exercises

S/N	Title of Experiment	Date	Remarks	Page number
1.	Bisection Method			
2.	Newton Raphson Method			
3.	False position (Regula Falsi) Method			
4.	Secant Method			
5.	Fixed point Iteration Method			
6.	Lagrange Interpolation			
7.	Newton's divided difference interpolation.			
8.				
9.				
10.				
11.				
12.				
13.				
14.				
15.				
16.				
17.				
18.				
19.				
20.				
21.				

Faculty Name:

Faculty Signature:

Lab Administrator Name:

Lab Administrator Signature:

External Examiner Name:

External Examiner Signature

B.Sc. CSIT 3rd Semester
Numerical Methods
Lab-

- Ankit Pandey

Title: Evaluation of Bisection Method

Objective: The objective of this experiment is to evaluate bisection method in C program.

Basic Theory:

Bisection method is one of the iterative methods of solving the sytem of non-linear equations. It is also known as half Interval method. one of the first numerical methods developed to find the root of non linear equation.

This method is based on the fouwing theorem:
An equation $f(x)=0$ where $f(x)$ is a real continuous function, has at least one root between x_l and x_u if $f(x_l) \cdot f(x_u) < 0$.

Note that:- If $f(x_l) \cdot f(x_u) > 0$, there may or may not be any root between x_l and x_u. If $f(x_l) \cdot f(x_u) < 0$ then there may be more than one root between x_l and x_u. So the theorem only guarantees one root between x_l and x_u. Since the method is based on finding the root between two points, the method falls under the category of bracketing methods.

General rule:
- If the $f(x_l)$ and $f(x_u)$ have the same sign (i.e. $f(x_l)$ and $f(x_u) > 0$)
  - There is no root between x_l & x_u. / even no. of roots.
- If the $f(x_l)$ and $f(x_u)$ have opposite signs.
  - There is an odd no. of roots.

Algorithm:

Step-1: Start

Step-2: Define function $f(x)$ and error (E)

Step-3: Take two initial value for root as x_1 and x_2.

Step-4: Compute $f(x_1)$ and $f(x_2)$

Step-5: If $f(x_1) \times f(x_2) > 0$, then x_1 and x_2 do not coverage and root does not lie in between x_1 and x_2 and go to step (10); otherwise continue.

Step-6: ~~Else~~ If $f(x_1) < 0$ then set $a = x_1$ and $b = x_2$ else,
set $a = x_2$ and $b = x_1$

Step-7: Calculate root, $x_n = \frac{a+b}{2}$ and also calculate $f(x_n)$.

Step-8: If $f(x_n) > 0$ then, set $b = x_n$
else
set $a = x_n$

Step-9: Repeat till step (8), until absolute value of $\frac{b-a}{b}$ is less than E, then display root.

Root: $\left(\frac{a+b}{2}\right)$.

Go to step (10) else to step (7)

Step 10: Stop

Executable Code :-

```
#include <stdio.h>
#include <math.h>
#define f(x) pow(x,2) - 4*x - 10
#define e 0.0001
Void main()
  {
 float x,x1,x2,x0;
  do
  {
  printf(" Enter the value of x1 and x2 \n");
   Scanf("%of %of", &x1, &x2);
   }
  while (f(x1)*f(x2)>0);
    do
    {
    x0= (x1+x2)/2;
    if ((f(x1)*f(x2)) <0)
    {
    x2=x0;
    }
    else x1=x0;
    x = (x1+x2)/2;
    }
    while (fabs(x0-x)>e);
    printf("root is =", x0);
    getch();
     }
```

Output:

Enter the vaue of x_1 and x_2

1

-3

Root is:- 0.999878.

B.Sc. CSIT 3rd Semester.
Numerical Methods - Ankit Pangeni
Lab-

Title: Evaluation of Newton Raphson method.

Objective: The objective of this experiment is to evaluate the Newton Raphson method in C program.

Basic Theory:-

Newton's Raphson method is another iterative method of solving system of non linear equations. It is based on the principle that if the initial guess of the root of $f(x)=0$ is at x_i, and if one draws the tangent to the curve at $f(x_i)$, the point x_{i+1} where the tangent crosses the x-axis is an improved estimate of the root.

It is an open method and starts with one initial guess for finding real root of non-linear equations.

In Newton Raphson method if x_0 is the initial guess then the next approximated root x_1 is obtained by the following formula.

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

If we suppose $f'(x_0)$ as $g(x_0)$ then, $x_1 = x_0 - \frac{f(x_0)}{g(x_0)}$

ie. The next approximated root x_{n+1} with initial guess x_n is

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

Algorithm:

1. Start
2. Define function as $f(x)$
3. Define first derivative of $f(x)$ as $g(x)$
4. Input initial guess (x_0), tolerable error (e) and maximum iteration (N).
5. Initialize iteration counter $i=1$.
6. If $g(x_0)=0$ then print "Mathematical Error" and goto Step(12) otherwise goto step(7).
7. Calculate $x_1 = x_0 - f(x_0)/g(x_0)$
8. Increment iteration counter $i = i+1$
9. If $i>=N$ then print "Not Convergent" and goto step(12) otherwise goto Step(10)
10. If $|f(x_1)|>e$ then set $x_0 = x_1$ and goto Step(6) otherwise goto (11)
11. Print root as x_1
12. Stop.

Executable Code:-

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#define f(x)  3*x - cos(x) - 1
#define g(x)  3 + sin(x)
void main()
  {
  float x0, x1, f0, f1, g0, e;
  int step=1, N;
  clrscr();
printf("\n Enter initial guess: \n");
 scanf("%f", &x0);
printf(" Enter tolerable error: \n");
scanf("%f", &e);
printf(" Enter maximum iteration: \n");
scanf("%d", &N);
printf("\n Step \t\t x0 \t\t f(x0) \t\t x1 \t\t f(x1) \n");
do
 {
g0 = g(x0);
f0 = f(x0);
if (g0 == 0.0)
   {
   printf(" Mathematical Error.");
   exit(0);
     }
x1 = x0 - f0/g0;
printf(" %d \t\t %f \t\t %f \t\t %f \t\t %f \n",
          step, x0, f0, x1, f1);
        x0 = x1;
```

```
step= step+1;
if (step>N)
  {
  printf("Not Convergent.");
   exit(0);
    }
  f1= f(x1);
   }
   while (fabs(f1)>e);
  printf("\n Root is: %of", x1);
   getch();
  }
```

Output:

```
Enter initial guess:
   1
Enter tolerable error:
   0.00001
Enter maximum iteration:
   10

Step        x0
1           1.000000
2           0.620016
3           0.607121

   Root is: 0.607102
```

B.Sc. CSIT 3rd Semester
Numerical Methods
Lab

- Ankit Pangeni

Title: Evaluation of False position (Regula Fasi) Method

Objective: The objective of this experiment is to evaluate the Regula Fasi (False position) method in C Program.

Basic Theory:

Regula Fasi is one of the bracketing and convergence guarenteed method for finding real root of non-linear equation. It starts with two initial guesses say x_0 and x_1 such that x_0 and x_1 brackets the root ie $f(x_0) \cdot f(x_1) < 0$

It is based on the fact that if $f(x)$ is real and continuous function, and for two initial guesses x_0 and x_1 brackets the root such that $f(x_0) \cdot f(x_1) < 0$ then there exists at least one root between x_0 and x_1.

If x_0 and x_1 are two guesses then we compute the new approximated root as:

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} \times f(x_0)$$

Now, we have the following three different case:

(1) If $f(x_2) = 0$ then root is x_2
(2) If $f(x_0) \cdot f(x_2) < 0$ then root lies between x_0 & x_2
(3) If $f(x_0) \cdot f(x_2) > 0$, then root lies betn x_1 & x_2

And then process is repeated until we have desired answer.

Algorithm

Step-1: Start

Step-2: Define function $f(x)$

Step-3: Choose initial guesses x_0 and x_1 such that $f(x_0) \cdot f(x_1) < 0$ ($f(x_0)$ (−), $f(x_1)$ (+))

Step-4: Choose pre-specified tolerable error e.

Step-5: Calculated new approximated root as

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

Step-6: Calculate $f(x_0) * f(x_2)$

a) If $f(x_0) * f(x_2) < 0$ then $x_0 = x_0$ and $x_1 = x_2$

b) If $f(x_0) * f(x_2) > 0$ then $x_0 = x_2$ and $x_1 = x_1$

c) if $f(x_0) * f(x_2) = 0$ then goto Step(8).

Step-7: If $|f(x_2)| > e$ then goto Step(5) otherwise goto Step(8)

Step-8: Display x_2 as root.

Step-9: Stop

Executable code.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define f(x) x*log10(x)-1.2
int main()
  {
  float x0,x1,x2,f0,f1,f2,e;
   int step=1;
   clrscr();
   up:
   printf("\n Enter two initial guesses:\n");
   scanf("%f %f", &x0, &x1);
   printf("Enter tolerable error:\n");
   scanf("%f", &e);
    f0=f(x0);
    f1=f(x1);
   if (f0*f1>0.0)
      {
       printf("Incorrect Initial Guesses.\n");
         goto up;
      }
printf("\n step \t\t x0 \t\t x1 \t\t x2 \t\t f(x2)\n");
  do
   {
    x2=x0-(x1-x0)*f0/(f1-f0);
    f2=f(x2);
   printf("%d \t\t %f \t %f \t %f \t %f\n", step,
        x0,x1,x2,f2);
    if (f0*f2<0)
        {x1=x2; f1=f2; }
```

```
else
  {
   x0=x2;
    f0=f2;
   }
  step=step+1;
   }
   while (fabs(f2)>e);
    printf ("\n Root is: %of", x2);
      getch();
       return 0;
       }
```

Output

```
Enter two initial guesses:
2
3
Enter tolerable error:
0.000001
```

Step	x0	x1	x2	f(x2)
1	2.000000	3.000000	2.721014	-0.017091
2	2.721014	3.000000	2.740206	-0.000384
3	2.740206	3.000000	2.740636	-0.000009
4	2.740636	3.000000	2.740646	-0.000000

Root is: 2.740646

B.Sc. CSIT 3rd Semester.
Numerical Methods.
Lab.

- Ankit Pangeni

Title: Evaluation of Secant method.

Objective: The objective of this experiment is to evaluate Secant method in c program.

Basic Theory:

Secant method is an open method and starts ~~two~~ with initial guesses for finding real root of non-linear equations. This method requires two initial guesses, but unlike the bisection and regula falsi method, the two initial guesses do not need to bracket the root of the equation.

Suppose x_0 and x_1 are initial guesses. Then, the condition that $f(x_0) \cdot f(x_1) < 0$ is not necessarily required. The Secant method may or may not converge, but when it converges, it converges faster than the bisection method. However, since the derivative is approximated, it converges slower than Newton-Raphson method.

In Secant method, if x_0 and x_1 are initial guesses, then next approximated root x_2 is obtained by:-

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

And an algorithm for Secant method involves repitition of above process, ie. we use x_1 & x_2 to find x_3 & so on until we find root within desired accuracy.

Algorithm:

Step-1: Start

Step-2: Define function as $f(x)$

Step-3: Input initial guesses x_0 and x_1, tolerable error (e) and maximum iteration (N)

Step-4: Initialize iteration counter $i=1$

Step-5: If $f(x_0) = f(x_1)$ then print "Mathematical Error" and goto step(11) otherwise goto step(6)

Step-6: Calculate $x_2 = x_1 - (x_1 - x_0) * f(x_1) / (f(x_1) - f(x_0))$

Step-7: Increment iteration counter $i = i+1$

Step-8: If $i >= N$ then print "Not Convergent" and goto step(11) otherwise goto step(9)

Step-9: If $|f(x_2)| > e$ then set $x_0 = x_1$, $x_1 = x_2$ and goto step(5) otherwise goto step(10)

Step-10: Print root as x_2

Step-11: Stop

Executable code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#define f(x)  x*x*x - 2*x - 5

void main()
{
float x0, x1, x2, f0, f1, f2, e;
int step=1, N;
clrscr();
printf("\n Enter initial guesses :\n");
scanf("%f %f", &x0, &x1);
printf(" Enter tolerable error:\n");
scanf("%f", &e);
printf(" Enter maximum iteration:\n");
scanf("%d", &N);

printf("\n Step \t\t x0 \t\t x1 \t\t x2 \t\t f(x2)\n");
do {
   f0 = f(x0);  f1 = f(x1);
   if(f0==f1)
   {
     printf("Mathematical Error.");
     exit(0);
   }
   x2 = x1 - (x1-x0) * f1 / (f1-f0);
   f2 = f(x2);
   printf("%d \t\t %f \t\t %f \t\t %f \t\t
          %f\n", step, x0, x1, x2, f2);
```

```
x0=x1; f0=f1; x1=x2; f1=f2;
step=step+1;
if (step >N)
    {
    printf("Not Convergent.");
      exit(0);
    }
while (fabs(f2)>e);
printf("\n Root is: %of", x2);
  getch();
    }
```

Output:

Enter initial guesses:

1

2

Enter tolerable error:

0.00001

Enter maximum Iteration:

10

Step	x_0	x_1	x_2	$f(x_2)$
1	1.000000	2.000000	2.200000	1.248001
2	2.000000	2.200000	2.088968	-0.062124
3	2.200000	2.088968	2.094233	-0.003554
4	2.088968	2.094233	2.094553	0.000012
5	2.094233	2.094533	2.094552	0.000001

Root is: 2.094552.

B.Sc. CSIT 3rd Semester -Ankit Pangeni
Numerical Methods
Lab-

Title: Evaluation of Fixed Point Iteration Method

Objective: The objective of this experiment is to evaluate ~~bisect~~ fixed point iteration method in C program.

Basic Theory:

Fixed point iteration method is open and simple method for finding real root of non-linear equation by successive approximation. It requires only one initial guess to start. Since it is open method its convergence is not guaranteed. This method is also known as iterative method.

To find the root of non linear equation $f(x)=0$ by fixed point iteration method, we write given equation $f(x)=0$ in the form of $x=g(x)$

If x_0 is initial guess then next approximated root in this method is obtained by [general form $x_{n+1}=g(x_n)$]

$x_1=g(x_0)$

And similarly, $x_2=g(x_1)$ and so on. The process is repeated until we get root with desired accuracy.

Note: While expressing $f(x)=0$ to $x=g(x)$, we can have many different forms. For convergence following criteria must be satisfied

$|g'(x)|<1$

Algorithm

Step-1: Start

Step-2: Define function $f(x)$

Step-3: Define function $g(x)$ which is obtained from $f(x)=0$ such that $x=g(x)$ and $|g'(x)<1|$

Step-4: Choose initial guess x_0, Tolerable error 'e' and Maximum Iteration 'N'.

Step-5: Initialize iteration counter: step=1

Step-6: Calculate $x_1=g(x_0)$

Step-7: Increment iteration counter: step=step+1

Step-8: If step >N then print "Not Convergent" and goto Step(12) otherwise goto Step(10)

Step-9: Set $x_0=x_1$ for next Iteration.

Step-10: If $|f(x_1)|>e$ then goto Step(6) otherwise goto Step(11)

Step-11: Display x_1 as root

Step-12: Stop.

Executable code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define f(x)  cos(x)-3*x+1
#define g(x)  (1+cos(x))/3

int main()
  {
  int step=1, N;
  float x0,x1,e;
  clrscr();
  printf("Enter initial guess:");
  scanf("%f", &x0);
  printf("Enter tolerable error:");
  scanf("%f", &e);
  printf("Enter maximum iteration:");
  scanf("%d", &N);
  printf("\n Step \t x0 \t\t f(x0) \t\t x1 \t\t f(x1) \n");
   do
   {
  x1=g(x0);
  printf("%d \t %f \t %f \t %f \t %f\n", step,
     x0, f(x0), x1, f(x1));

   step=step+1;
   if(step>N)
     {
     printf("Not Convergent.");
      exit(0);
      }
     x0=x1;
   }
  while(fabs(f(x1))>e);
     printf("\n Root is %f", x1);
      getch();
       return 0;
          }
```

Output:

Enter initial guess: 1

Enter tolerable error: 0.000001

Enter maximum iteration: 10

Step	x_0	$f(x_0)$	x_1	$f(x_1)$
1	1.000000	-1.459698	0.513434	0.330761
2	0.513434	0.330761	0.623688	-0.059333
3	0.623688	-0.059333	0.603910	0.011391
4	0.603910	0.011391	0.607707	-0.002162
5	0.607707	-0.002162	0.606986	0.000411
6	0.606986	0.000411	0.607124	-0.000078
7	0.607124	-0.000078	0.607098	0.000015
8	0.607098	0.000015	0.607102	-0.000003
9	0.607102	-0.000003	0.607102	+0.00001

Root is 0.0607102.

B.Sc. CSIT 3rd Semester. -Ankit Pangeni
Numerical Methods.
Lab -

Title: Evaluation of Lagrange Interpolation.

Objective: The objective of this experiment is to evaluate the Lagrange Interpolation in C program.

Basic Theory:

Suppose we have following sets of data tabulated for x (independent variable) and y (dependant variable)

x	x_0	x_1	x_2	----	x_n
y	y_0	y_1	y_2	----	y_n

Then the method of finding the value of $y=f(x)$ corresponding to any value $x=x_i$ within x_0 and x_n is called interpolation. Thus interporation is the process of finding the value of function for any intermediate value of independent variable. If we need to estimate the value of function $f(x)$ outside the tabular values, then the process is called extrapolation. However, in general extrapolation is also included in interpolation.

There are different methods of interpolation ex: Lagrange Interporation, newtons divided difference interpolation, newtons forward difference interpolation, newtons backward difference interporation, ex.

Here, we discuss about Lagrange interporation.
Lagranges Interporation formula is given by

$$P_n(x) = \sum_{i=0}^{n} f_i l_i(x) \quad \text{where, } l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}$$

It's general formula is given by:

⇒ If $y = f(x)$ takes the value of $y_0, y_1, y_2, y_3 \ldots y_n$ corresponding to $x_0, x_1, x_2, x_3, \ldots, x_n$ then

$$y = f(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)} y_0 + \frac{(x-x_0)(x-x_2)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)} y_1 + \ldots + \frac{(x-x_0)(x-x_1)\ldots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\ldots(x_n-x_{n-1})} y_n$$

is known as Lagrange Interpolation formula for unequal intervals & is very simple to implement on computer.

Algorithm

Step-1: Start

Step-2: Read number of data (n)

Step-3: Read data x_i and y_i for $i=1$ to n.

Step-4: Read value of independent variables say x_p. whose corresponding value of dependent say y_p is to be determined.

Step-5: Initialize $y_p = 0$

Step-6: For $i=1$ to n
  Set $p=1$
  For $j=1$ to n
    If $i \neq j$ then
      Calculate $p = p * (x_p - x_j) / (x_i - x_j)$
    End if
  Next j
  Calculate $y_p = y_p + p * y_i$
Next i

Step-7: Display value of y_p as interpolated value.

Step-8: Stop.

Executable code:

```
#include <stdio.h>
#include <conio.h>
void main()
  {
  float x[100], y[100], xp, yp=0, p;
  int i, j, n;
  clrscr();
  printf("Enter number of data:");
   scanf("%d", &n);
   printf("Enter data:\n");
    for(i=0; i<n; i++)
        {
      printf("x[%d] = ", i);
       scanf("%f", &x[i]);
       printf("y[%d] = ", i);
        scanf("%f", &y[i]);
       }
 printf("Enter interpolation point:");
  scanf("%f", &xp);
   for(i=1; i<=n; i++)
         {
          p=1;
         for(j=1; j<=n; j++)
           {
           if(i!=j)
           {
           p=p* (xp-x[j])/(x[i]-x[j]);
           }
         }
        yp=yp+ p* y[i];
         }
```

```
printf("Interpolated value at %0.3f is %0.3f.", xp, yp);
getch();
}
```

Output:

```
Enter number of data: 5
Enter data:
x[1] = 5
y[1] = 150
x[2] = 7
y[2] = 392
x[3] = 11
y[3] = 1452
x[4] = 13
y[4] = 2366
x[5] = 17
y[5] = 5202

Enter interpolation point: 9
Interpolated value at 9.000 is 810.000
```

B.Sc. CSIT 3rd semester. - Ankit Pangeni
Numerical Methods.
Lab - 07

Title: Evaluation of Newton's Divided Difference Interpolation.

Objective: The objective of this experiment is to evaluate the Newton's divided difference interpolation in a C program.

Basic Theory:

Newton's divided difference Interpolation is one of the interpolation technique used when the interval difference is not same for all sequence of values.

Let us consider a polynomial of degree n of the form

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

To construct the polynomial we need to find coefficients $a_0, a_1, a_2, \ldots, a_n$. Suppose $x_0, x_1, x_2, \ldots, x_n$ be given interpolating points & $f(x_0), f(x_1), f(x_2), \ldots, f(x_n)$ be the $(n+1)$ values of function $y = f(x)$ corresponding to $x_0, x_1, x_2, \ldots, x_n$ where interval difference are not same.

Then first divided difference is given by

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

Second divided difference is given by

$$f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$ and so on.

Which gives the Newton's Dividend Difference formula:

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

Algorithm.

Step-1: Start

Step-2: Read the number of points, say n.

Step-3: Read the value at which interpolated value is needed, say x

Step-4: Read given data points.

Step-5: Calculate first divided difference as

```
For i:=0 to n-1
    dd[i]= fx[i]
End For
```

Step-6: Calculate second to nth divided differences as

```
For i=0 to n-1
    for j= n-1 to i+1
        dd[j] = (dd[j]- dd[j-1]) / (x[j] - x[j-1-i])
End For.
End for.
```

Step-7: Set v:=0 and p:=1

Step-8: Calculated interpolated value as

```
For i:=0 to n-1
    For j:=0 to i-1
        p=p*(x-xj)
    End For
    v:=v+dd[i]*p
    Reset p:=1
End For
```

Step-9: Print the interpolated value v

Step-10: Terminate.

Executable code.

```
#include <stdio.h>
#include <conio.h>
int main()
  {
  int n,i,j;
 float v=0, p,xv, x[10], fx[10], a[10];
  printf("Enter the number of points \n");
   scanf("%d", &n);
   printf("Enter the value of x \n");
    scanf("%f", &xv);
    for(i=0;i<n;i++)
       {
       printf("Enter the value of x and fx at i=%d \n",i);
       scanf("%f %f", &x[i], &fx[i]);
       }
     for(i=0; i<n; i++)
            a[i]= fx[i];
      for(p=0; i<n;i++)
         {
          for(j=n-1; j>i;j--)
              a[j] = (a[j]-a[j-1])/(x[j]-x[j-1-i]);
         }
       v=0;
      for(p=0; p<n;i++)
          {
           p=1;
           for(j=0; j<=i-1;j++)
              p=p* (xv-x[j]);
              v=v+ a[i]*p;
            }
          printf("Interpolation value=%f", v);
              getch();
                return 0;  }
```

Output

```
Enter the number of points
4
Enter the vawe of x
1.3
Enter the vawe of x and fx at i=0
0.5    0.4794
Enter the vawe of x and fx at i=1
1      0.8415
Enter the vawe of x and fx at i=2
1.5    0.9975
Enter the vawe of x and fx at i=3
2.0    0.9093

Interporation vawe : 0.962270
```