Mechi Multiple Campus

(Tribhuvan University)

Bhadrapur, Jhapa

Lab Report of Numerical Methods (CACS-252)

Faculty of Humanities & Social Sciences

Tribhuvan University

Kritipur, Nepal

Submitted By

Name: Santosh Bhandari

Roll No: 58

Submitted To

Mechi Multiple Campus

Department of Bachelor in Computer

Bhadrapur, Jhapa, Nepal

Certificate from the supervisor

This is to certify that the Lab Report entitled **"Numerical Methods"** is an academic work done by **"Santosh Bhandari"** submitted in the partial fulfillment of the requirements for the degree of **Bachelor of Computer Application** at Faculty of Humanities and Social Science, Tribhuvan University under my guidance and supervision. To the best of my knowledge, the worked performed by him in the Lab report is his own creation.

Signature of the Supervisor:-
Name:-
Designation:-
Date:-

Acknowledgement

I would like to express my special thanks of gratitude to my Numerical Methods teacher "Mr. Narayan Dhamala" and to the whole faculty member of humanities and social science who has provided me this opportunity. I am really grateful to sir Narayan Dhamala for this kind of support in this report.

We are making this report as it compulsory required by Mechi Multiple Campus. It is compulsory report that should be submitted to the college in order to get practical marks. For this report our Database Management System teacher has motivated us. We are also guided by him to score good marks in practical. He has suggested how to make the report. He was the main source for us to make the report ready for Mechi Multiple Campus. In this report, we the student of BCA were engaged and I would like to express my deep thankful particularly to all of them. We are doing this report to be very helpful for coming days.

Thank You.

Table of Content

1. Implementation of Bisection Method

Definition

Bisection method is an iterative bracketing method used for finding the root of the non-linear equation which uses two initial guess value.

Algorithm

① Input a function f(x), initial guess value x_1 and x_2 and stopping criteria E.
② Compute $f_1 = f(x_1)$ and $f_2 = f(x_2)$
③ If $f_1 * f_2 >= 0$ then x_1 and x_2 doesnot bracket the root and go to step 2 and chose another guess value otherwise continue.
④ Compute $x_0 = \frac{x_1 + x_2}{2}$, and $f_0 = f(x_0)$
⑤ If $f_0 * f_1 < 0$ then root lies in (x_0, x_1) and set $x_1 = x_1$ and $x_2 = x_0$
⑥ If $f_0 * f_2 < 0$ then root lies in (x_0, x_2) and set $x_1 = x_0$ and $x_2 = x_2$.
⑦ If $\left|\frac{x_2 - x_1}{x_2}\right| \leq E$ then root $= x_0$ otherwise goto step 4.

Some code

```
#include<stdio.h>
#include<math.h>
#define f(x) (x*x*x - 4*x - 9)
#define E 0.0001
void main(){
    float a, b, c, f0, f1, f2;
    lbl:
    printf("Enter Two initial Guess value:");
    scanf("%f%f", &a, &b);
    f1 = f(a)
    f2 = f(b)
    if(f1 * f2 >= 0){
        printf("The initial Guess Value is not correct. Please chose
                another Guess value\n");
        goto lbl;
```

```
else{
    top:
    c=(a+b)/2;
    fc=f(c);
    if(fc==c)
        printf("The root is : %f\n",c);
    if(f1*fc<0){
        b=a;
        f2=fc;
    }else{
        a=c;
        f2=fc;
    }
    if(fabs((b-a)/b)<=E){
        printf("The root is : %f\n",c);
    }else{
        goto top;
    }
}
}
```

Output

```
Enter Two initial Guess Value: 2
3
The root is : 2.750000
```

---

2. Implementation of False Position Method

Defination:

A false position method is a bracketing iterative method for finding the root of the non-linear equation which uses two initial guess values.

Algorithm

① Input a function $f(x)$, Initial guess value x_1 and x_2 and stopping criteria ϵ.
② Compute $f(1) = f(x_1)$ and $f(2) = f(x_2)$
③ If $f_1 * f_2 >= 0$ then x_1 and x_2 doesnot bracket the root and go to step 1 and choose another guess value otherwise continue.
④ Compute $x_0 = x - f(x_1) \frac{(x_2 - x_1)}{f(x_2) - f(x_1)}$, and $f_0 = f(x_0)$
⑤ If $f_0 * f_1 < 0$ then root lies in (x_0, x_1) and set $x_1 = x_1$ and $x_2 = x_0$
⑥ If $f_0 * f_2 < 0$ then root lies in $(x_0 x_2)$ and set $x_1 = x_0$ and $x_2 = x_2$.
⑦ If $\left| \frac{x_2 - x_1}{x_2} \right| \leq \epsilon$ then root $= x_0$ otherwise goto step 4.

Source code

```
#include <stdio.h>
#include <math.h>
#define f(x) (x*x-x-2)
#define E 0.0001
void main(){
    float x0,x1,x2,f0,f1,f2;
    lbl:
    printf("Enter Two Initial Guess Value:");
    scanf("%f%f",&x1,&x2);
    f1=f(x1);
    f2=f(x2);
    if(f2*f1>=0){
        printf("The Initial Guess value is not correct. Please choose
        another Guess value\n");
        goto lbl;
    }else{
        top:
        x0=(x1-((f1*(x2-x1))/(f2-f1)));
        f0=f(x0);
        if(f0==0){
            printf("The Root is: %f\n",x0);
    }
```

```
            if (f1*fo<0){
                x2=xo;
            }else{
                x1=xo;
            }
            if(fabs((x2-x1)/x2)<=E){
                printf("The Root is : %f\n", xo);
            }else{
                goto top;
            }
        }
    }
```

Output

```
Enter Your Initial Guess Value: 1
3
The Root is: 2.000064
```

3. Implementation of Newton Raphson Method

Definition

A Newton Raphson Method is a non-bracketing iterative method for finding the root of the non-linear equation which uses only one initial guess value.

Algorithm

① Input a function f(x), Initial Guess Value (x_0) and Error Tolerance (E).
② Compute $f(x_0)$ and $f'(x_0)$
③ Compute value of x_1 using

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

④ If $\left|\frac{x_1 - x_0}{x_1}\right| \leq E$, then Stop and print root $= x_1$,
ELSE
Set $x_0 = x_1$, and goto Step 2.

Source code

```
#include<stdio.h>
#include<math.h>
#define f(x) (x*x-3*x+2)
#define g(x) (2x-3)
#define E 0.0001
void main(){
    float x0,g0,f0,x1;
    printf("Enter Initial Guess Value:");
    scanf("%f",&x0);
    top:
    f0=f(x0);
    g0=g(x0);
    x1=x0-(f0/g0);
    if(fabs((x1-x0)/x1)>=E){
        printf("The Root is:%f",x1);
    }else{
        x0=x1;
        goto top;
    }
}
```

Output

```
Enter Initial Guess Value: 0
The Root is: 1.0000
```

4. Implementation of Fixed point Iteration Method.

Definition

A fixed point iteration method is a non-Bracketing iterative method for finding the root of non-linear equation which uses only one initial guess value.

Algorithm

① Input a function f(x) and Re-Arrange f(x0) as x=g(x)
② Provide an initial Guess value x0 and Error Tolerance E.
③ Compute $x_1 = g(x_0)$

④ If $\left|\frac{x_1 - x_0}{x_1}\right| \leq \epsilon$, then algorithm stops and print root = x_1,

Else

$x_0 = x_1$, and goto Step3.

Source code

```
#include<stdio.h>
#include<math.h>
#define f(x) (x*x+x-2)
#define g(x) (2-x*x)
#define E 0.000)
Void main(){
    float x0,g0,f0,x1;
    printf("Enter Initial Guess value:");
    scanf("%f",&x0);
    top:
     x1=g(x0);
    if(fabs((x1-x0)/x1)<=E){
        printf("The Root is: %f", x1);
    }else{
        x0=x1;
        goto top;
    }
}
```

Output

Enter Initial Guess Value: 0

The Root is: -2.0000

5. Implementation of Numerical Differentiation of Continuous Function

Definition

The process of finding the derivative of a function is known as numerical differentiation.

Numerical Differentiation for continuous function:-

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Algorithm

① Input a function $f(x)$, Value of x and h.

② Compute first Derivative as:-

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

③ Calculate Second Derivative as:-

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

④ Print the result of first and Second Derivative.

Source code

```
#include<stdio.h>
#define f(x) (x*x)
void main(){
    float x,h,k,l;
    printf("Enter the value of x and h:\n");
    scanf("%f%f",&x,&h);
    k=(f(x+h)-f(x-h))/2*h;
    l=(f(x+h)-2*f(x)+f(x-h))/h*h;
    printf("First Derivative: %f\nSecond Derivative: %f",
    k,l);
```

Output

Enter the Value of x and h:
2
0.25
First Derivative : 0.187500
Second Derivative: -4.000000

6. Implementation of Integration Using Trapezoidal Rule

Definition

Trapezoidal is a technique for evaluating definite integrals. This method is based on Newton's Code Quadrature Formula.

Algorithm

① Input a function f(x), Value of lower limit, Upper limit and number of sub interval (n).
② Calculate Step Size $(h) = \frac{(b-a)}{n}$
③ Calculate integration = f(a) + f(b)
④ Set i = 1.
⑤ Loop while i<=n
    x = a + i*h
    integration = integration + 2*f(x)
    i = i+1
⑥ Integration = integration * h/2
⑦ Print result Integration.

Source code

```
#include <stdio.h>
#define f(x) (x*x)
void main(){
    float a,b,h,i,l,n;
    printf("Enter the Value of Lower Limit, Upper Limit and
    No. of Sub Interval:\n");
```

```
scanf("%f%f%f", &a, &b, &n);
h=(b-a)/n;
for(i=a+h; i<=b-h; i=i+h)
    l=l+(2*f(i));
printf("Result : %f", ((f(a)+f(b))+l));
```

Output

Enter the value of Lower Limit, Upper Limit and No. of Sub Interval:
0
2
4
Result: 11.000000

7. Implementation of Integration Using Simpson 1/3 Rule

Defination

Simpson's 1/3 rule is an extension of the trapezoidal rule in which the integrand is approximated by a second-order polynomial. Simpson rule can be derived from the various way using Newton's divided difference polynomial, Lagrange polynomial and the method of coefficients.

Algorithm

① Input function f(x), lower limit (a), upper limit (b) and No. of Sub Interval(n).

② Calculate:- Step Size (h) = (b-a)/n

③ Calculate: Integration = f(a)+f(b)

④ Set i=1 and loop while i<=n

```
    K=a+i*h;
    If i%2=0
        Integration = Integration + 2*f(K)
    else
        Integration = Integration + 4*f(K)
    i=i+1
```

⑤ Integration = Integration * step_size(h)/3

⑥ Print the Result = Integration

Source code

```
#include<stdio.h>
#define f(x) (x*x)
void main(){
        float a,b,n,h,i,m,o,res;
        int j;
        printf("Enter the value of lower limit, Upper limit and No.of
        Sub Interval:\n");
        scanf("%f%f%f",&a,&b,&n);
        h=(b-a)/n;
        for(i=1;i<=n-1;j++){
                i=a+j*h;
                if(j%2==0)
                    m=m+2*f(i);
                else
                    o=o+4*f(i);
        }
        res=(f(a)+f(b)+m+o)*h/3;
        printf("Result: %f",res);
}
```

Output

```
Enter the Value of Lower Limit, Upper Limit and No. of Sub Interval:
0
2
4
Result: 2.666667
```

11. Implementation of Integration Using Simpson 3/8 Rule

Definition

Simpson rule is one of the numerical methods which is used to evaluate the definite integral. Simpson 3/8 is completely based on the cubic interpolation rather than the quadratic interpolation.

Algorithm

① Input a function f(x), Lower Limit (a), Upper Limit (b) and No. of Sub Interval (n).
② Calculate :- Step Size (h) = (b-a)/n
③ Calculate :- Integration = f(a) + f(b)
④ Set i=1 and loop while i<=n

```
K= a+i*h;
If i%3=0
    Integration = Integration + 2*f(K)
Else
    Integration = Integration + 3*f(K)
i=i+)
```

⑤ Integration = Integration *3h/8
⑥ Print the Result = Integration

Source Code

```
#include<Stdio.h>
#define f(x) (x*x)
void main(){
    float a,b,n,h,l,m,o,res;
    printf("Enter the Value of Lower Limit, Upper Limit and No. of
    Sub Interval:\n");
    scanf("%f%f%f",&a,&b,&n);
    h=(b-a)/n;
    for(i=1;i<=n-1;i++){
        l=a+i*h;
```

```
if(i%3==0)
    m=m+2*f(i);
else
    o=o+3*f(i);
}
res=(f(a)+f(b)+m+o)*(3*h/8);
printf("Result: %f", res);
```

3

Output

Enter the Value of Lower Limit, Upper Limit and No. of Sub Interval :

0

2

4

Result : 2.296875

9. Implementation of Eulers Method

Definition

Eulers method is a technique for approximating solutions of first-order differential equations. It is relies on the fact that the equation $y' = f(x,y)$ can be used to calculate the derivative of the solution at any point.

Algorithm

① Input a function f(x), value of x_0 and y_0, Step size h and Calculation point (n).

② Set i = 0 and loop while i <= n

$y_1 = y_0 + h * f(x_0, y_0)$;
$x_0 = h + x_0$;
$y_0 = y_1$;
$i = i + h$;

③ Result : y_1

Source Code

```
#include<stdio.h>
#define f(x,y) (3*x*x+1)
void main(){
    float h,x0,y0,y1,n,i;
    printf("Enter the Value of x0 and y0\n");
    scanf("%f%f",&x0,&y0);
    printf("Enter step size h:\n");
    scanf("%f",&h);
    printf("Enter the point value to be Calculated:\n");
    scanf("%f",&n);
    for(i=x0+h;i<n;i=i+h){
        y1=y0+h*f(x0,y0);
        x0=h+x0;
        y0=y1;
    }
    printf("The Result is %.4f",y1);
}
```

Output

```
Enter the Value of no and yo
1
2
Enter Step Size H:
0.5
Enter the point value to be Calculated:
2
The Result is 7.8750
```

10. Implementation of Heuns Method

Definition

Heun's Method is a second order Rungee-Kutta method that is also known as the improved Euler Method. Heun's method is the method to solve ordinary differential Equation.

Algorithm

① Input a function f(x), Value of no and yo, Step Size (H) and Calculation point (n).

② Set i = no+h and loop while i<=n

```
m1 = f(no, yo);
K = no + h;
m2 = f(K, yo + h*f(no, yo));
y1 = yo + ((m1+m2)/2) * h;
no = K;
yo = y1;
```

③ Display Result as y1.

Soure Code

```
#include<stdio.h>
#define f(x,y) (2*y/x)
void main(){
    float h, no, yo, K, y1, n, m1, m2, x1, i;
    printf("Enter the Value of no and yo :\n");
    scanf("%f%f", &no, &yo);
    printf("Enter Step Size H:\n");
    scanf("%f", &h);
    printf("Enter the Point Value to be Calculated :");
```

```
scanf("%f", &n);
for(i=no+h; i<=n; q=q+h){
    m1=f(no,yo);
    k=no+h;
    m2=f(k, yo+h*f(no,yo));
    y1=yo+((m1+m2)/2)*h;
    no=k;
    yo=y1;
}
printf("The Result is %.4f", y1);
}
```

Output

```
Enter the Value of no and Yo:
1
2
Enter Step Size H:
0.5
Enter the point Value to be Calculated: 2
The Result is 8.3056
```

11. Implementation of RK Method.

Definition

Runge-Kutta Method is an effective and widely used method for solving the initial-value problems of differential equations. It can be used to construct higher order accurate numerical method by function's self without needing the high order derivatives functions.

Algorithm

(1) Input a function f(x), Value of no and Yo, Step Size (H) and Calculation Point (n).

(2) Set q = no+h and loop while q<=n

$m_1 = f(n_0, y_0)$

$m_2 = f(n_0 + h/2, y_0 + m_1 * h/2)$

$m_3 = f(n_0 + h/2, y_0 + m_2 * h/2)$

$m_4 = f(n_0 + 1, y_0 + 2 * m_3 * h)$

$y_1 = y_0 + ((m_1 + m_2 + m_3 + m_4)/6) * h$;
$x_0 = x_0 + h$;
$y_0 = y_1$;

③ Display the Result as y_1.

Source Code

```
#include<stdio.h>
#define f(x,y) (x+y)
void main(){
    float h, x0,y0, k,y1,n,m1,m2,m3,m4,x1,s;
    printf("Enter the Value of x0 and y0 :\n");
    scanf("%f%f", &x0, &y0);
    printf("Enter Step Size of H: ");
    scanf("%f", &h);
    printf("Enter the point Value to be Calculated : ");
    scanf("%f", &n);
    for(s=x0+h; s<=n; s=s+h){
        m1=f(x0,y0);
        k=h/2;
        m2=f(x0+k, y0+m1*k);
        m3=f(x0+k, y0+m2*k);
        m4=f(x0+1, y0+m3*h);
        y1=y0+((m1+m2+m3+m4)/6)*h;
        x0=x0+h;
        y0=y1;
    }
    printf("The Result is %f.", y1);
}
```

Output

```
Enter the Value of x0 and y0:
0
1
Enter Step Size of H: 0.5
Enter the point Value to be Calculated : 1
The Result is 1.528435
```

12. Implementation of Jacobi Iteration

Definition

In Numerical analysis, Jacobi method is iterative approach for finding the numerical solution of diagonally dominant System for linear equations.

Algorithm

① Input a function f1(x) and f2(x), Initial guess value and Error tolerance value.

② Loop while e1>E and e2>E

x1new = f1(x1,x2)
x2new = f2(x1,x2)
e1 = fabs(x1new - x1)
e2 = fabs(x2new - x2)
x1 = x1new;
x2 = x2new

③ Display the result as x1 : x1 and x2 : x2

Source Code

```
#include<stdio.h>
#include<math.h>
#define f1(x1,x2) ((5-x2)/3)
#define f2(x2,x2) ((5-x1)/-3)
void main(){
    float x1,x2, E, x1new, x2new, e1,e2;
    printf("Enter initial Guess: \n");
    Scanf("%f%f", &x1, &x2);
    printf("Enter Error Tolerance: ");
    Scanf("%f", & E);
    do{
        x1new = f1(x1,x2);
        x2new = f2(x1,x2);
        e1 = fabs(x1new - x1);
```

```
        e2 = fabs(x2new - x2);
        x1 = x1new;
        x2 = x2new;
    }while(e1 > E && e2 > E);
    printf("The result of x1 is %.4f and x2 is %.4f.",
        x1, x2);
}
```

Output

```
Enter Initial Guess
0
0
Enter Error Tolerance: 0.0001
The Result of x1 is 2.0000 and x2 is -1.0000.
```

13. Implementation of Gauss Seidal.

Definition

Gauss Seidal method is iterative approach for solving system of linear equation. In this method, first given system of linear equations are arranged in diagonally dominant form.

Algorithm

① Input a function f1(x) and f2(x), Initial Guess Value and Error Tolerance Value.

② Loop While e1 > E and e2 > E

```
x1new = f1(x1, x2);
e1 = fabs(x1new - x1);
x1 = x1new;
x2new = f2(x1, x2);
e2 = fabs(x2new - x2);
x2 = x2new;
```

③ Display Result x1: x1 and x2: x2.

Source Code

```
#include<stdio.h>
#include<math.h>
Void main
#define f1(x1,x2) ((5-x2)/3)
#define f2(x1,x2) ((5-x1)/-3)
void main(){
    float x1,x2,E,x1new,x2new,e1,e2;
    printf("Enter Initial Guess :\n");
    scanf("%f%f",&x1,&x2);
    printf("Enter error Tolerance : ");
    scanf("%f",&E);
    do{
        x1new=f1(x1,x2);
        e1=fabs(x1new-x1);
        x1=x1new;
        x2new=f2(x1,x2);
        e2=fabs(x2new-x2);
        x2=x2new;
    }while(e1>E && e2>E)
    printf("The Result of x1 is %.4f and x2 is %.4f.",x1,x2);
}
```

Output

```
Enter Initial Guess :
0
0
Enter Error Tolerance : 0.0001
The Result is x1 is 2.0000 and x2 is -1.0000
```

14. Implementation of Gauss Elimination Method.

Definition

In linear algebra, Gauss Elimination Method is a procedure for solving systems of linear equation. It is also known as Row Reduction Technique.

Algorithm

① Read Augmented Matrix(A) of n by n+1 Size.
② Transform Augmented Matrix to Upper Triangular Matrix by Row Operations.
③ Obtain Solution by Backward Substitution.
④ Display the result.

Source code

```
#include<stdio.h>
#include<stdlib.h>
void main(){
    float a[5][5],x[5],ratio;
    int i,j,k;
    printf("Solution of 3x3 Matrix Linear Equation.\n Enter the
        Value:\n");
    for(i=0;i<3;i++){
        for(j=0;j<=3+1;j++){
            printf("a[%d][%d] = ",i,j);
            scanf("%f",&a[i][j]);
        }
    }
    for(i=1;i<=3-1;i++){
        if(a[i][i]==0){
            printf("Mathematical Error!");
            exit;
        }
        for(j=i+1;j<=3;j++){
            ratio=a[j][i]/a[i][i];
            for(k=1;k<=3+1;k++)
                a[j][k]=a[j][k]-ratio*a[i][k];
        }
    }
    x[3]=a[3][3+1]/a[3][3];
    for(i=3-1;i>=1;i--){
        x[i]=a[i][3+1];
        for(j=i+1;j<=3;j++)
```

```
            x[i]=x[i] - a[i][j] * x[j];
        x[i]=x[i] /a[i][i];
    }
    printf("\nResult:\n");
    for(i=1; i<=3; i++)
        printf("x %d = %.04f \n", i, x[i]);
}
```

Output

Solution of 3x3 Linear Equation.
Enter the Value:
a[1][1]=2
a[1][2]=2
a[1][3]=1
a[1][4]=6
a[2][1]=4
a[2][2]=2
a[2][3]=3
a[2][4]=4
a[3][1]=1
a[3][2]=1
a[3][3]=1
a[3][4]=0

Result:
x1=5.0000
x2=1.0000
x3=-6.0000

15. Implementation of Gauss Jordan Method.

Definition

In linear algebra, Gauss Jordan Method is a procedure for Solving Systems of linear equations. In this method, the problem of systems of linear equation having n unknown variables, matrix having rows n and columns n+1 is formed.

Algorithm

① Read Argmented Matrix (A) of n by n+1 size.
② Transform A to Diagonal matrix by Row Operation.
③ Obtain the Solution by Making All Dragonal Elements to 1.
④ Display the result.

Soure Code

```
#include<stdio.h>
#include<stdlib.h>
void main(){
    float a[5][5],x[5],ratio;
    int i,j,k,n=3;
    printf("Solution of 3x3 Linear Equation.\n Enter the Value:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n+1;j++){
            printf("a[%d][%d] = ",i,j);
            scanf("%f",&a[i][j]);
        }
    for(i=1;i<=n;i++){
        if(a[i][i]==0){
            printf("Mathematical Error.");
            exit;
        }
        for(j=1;j<=n;j++)
            if(i!=j){
                ratio = a[j][i]/a[i][i];
                for(k=1;k=n+1;k++)
                    a[j][k]=a[j][k]-ratio*a[i][k];
            }
    }
    for(i=1;i<=n;i++)
        x[i]=a[i][n+1]/a[i][i];
    printf("\n Result:\n");
    for(i=1;i<=n;i++)
        printf("x%d = %0.4f\n",i,x[i]);
}
```

Output

Solution of 3x3 linear Equation
Enter the Value:
a[1][1] = 2
a[1][2] = 4
a[1][3] = -6
a[1][4] = -8
a[2][1] = 1
a[2][2] = 3
a[2][3] = 1
a[2][4] = 10
a[3][1] = 2
a[3][2] = -4
a[3][3] = -2
a[3][4] = -12

Result:
x1 = 1.0000
x2 = 2.0000
x3 = 3.0000

16. Implementation of Linear Interpolation Method

Definition

Linear interpolation is a form of interpolation, which involves the generation of new values based on an existing set of values. It is achieved by geometrically rendering a straight line between two adjacent point on a graph or plane.

Algorithm

① Read data points of x1 and x2.
② Read corresponding function value f(x1) and f(x2).
③ Enter the point to calculate Interpolation(x)
④ Calculate $f = f(x_1) + (x - x_1) \frac{f(x_2) - f(x_1)}{x_2 - x_1}$
⑤ Display the result as f.

Source code

```
#include <stdio.h>
void main(){
        float f,f1,f2,x,x1,x2;
        printf("Enter the Value of x1 and x2 :\n");
        scanf("%f%f",&x1,&x2);
        printf("Enter x2 and x2 corresponding function value :\n");
        scanf("%f%f",&f1,&f2);
        printf("Enter the point for which you want to calculate Interpolation:");
        scanf("%f",&x);
        f=f1+((x-x1)*(f2-f1)/(x2-x1));
        printf("The Required Result is : %.4f",f);
}
```

Output

```
Enter the Value of x1 and x2 :
2
3
Enter x1 and x2 corresponding function Value :
1.4142
1.73921
Enter the point for which you want to Calculate Interpolation : 2.5
The Required Result is : 1.5731
```