Numerical Method Lab coding for chapter 1(Finding the Root of a Nonlinear Function):

By Mausham Adhikari

Features of Bisection Method

Type – closed bracket

No. of initial guesses – 2

Convergence – linear

Rate of convergence – slow but steady

Accuracy – good

Programming effort – easy

Approach – middle point

Below is a source code in C program for bisection method to find a root of the nonlinear function x^3 – 4*x – 9. The initial guesses taken are a and b. The calculation is done until the following condition is satisfied:

|a-b| < 0.0005 OR If (a+b)/2 < 0.0005 (or both equal to zero)

where, (a+b)/2 is the middle point value.

Variables:

itr – a counter variable which keeps track of the no. of iterations performed

maxmitr – maximum number of iterations to be performed

x – the value of root at the nth iteration

a, b – the limits within which the root lies

allerr – allowed error

x1 – the value of root at (n+1)th iteration

f(x) = x^3 – 4*x – 9

Source Code for Bisection Method in C:

```
#include<stdio.h>

#include<math.h>

float fun (float x)

{

    return (x*x*x - 4*x - 9);

}

void bisection (float *x, float a, float b, int *itr)

/* this function performs and prints the result of one iteration */

{

    *x=(a+b)/2;

    ++(*itr);

    printf("Iteration no. %3d X = %7.5f\n", *itr, *x);

}

void main ()

{

    int itr = 0, maxmitr;

    float x, a, b, allerr, x1;

    printf("\nEnter the values of a, b, allowed error and maximum iterations:\n");

    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);

    bisection (&x, a, b, &itr);

    do

    {

        if (fun(a)*fun(x) < 0)

            b=x;
        else
            a=x;
        bisection (&x1, a, b, &itr);
        if (fabs(x1-x) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x1);
            return 0;
        }
        x=x1;
    }
    while (itr < maxmitr);
    printf("The solution does not converge or iterations are not sufficient");
    return 1;
}
```

For Input, enter the value of a,b, allowed error and maximum iteration:

```
3
2
0.0005
20
```

For output, run the code with these input data.

Features of Newton Raphson Method:

Type – open bracket

No. of initial guesses – 1

Convergence – quadratic

Rate of convergence – faster

Accuracy – good

Programming effort – easy

Approach – Taylor’s series

Below is a very short and simple source code in C program for Newton’s method to find the root of x*log10(x) – 1.2.

Variables:

itr – a counter which keeps track of the no. of iterations performed

maxmitr – maximum number of iterations to be performed

df(x) – the derivative of f(x) with respect to x

x0 – the value of root at nth iteration

x1 – the value of root at (n+1)th iteration

allerr – allowed error

f(x) = x*log10(x) – 1.2

Source Code for Newton Raphson Method in C:

```
#include<stdio.h>

#include<math.h>

float f(float x)

{

   return x*log10(x) - 1.2;

}

float df (float x)
```

```
{
    return log10(x) + 0.43429;
}
void main()
{
    int itr, maxmitr;
    float h, x0, x1, allerr;
    printf("\nEnter x0, allowed error and maximum iterations\n");
    scanf("%f %f %d", &x0, &allerr, &maxmitr);
    for (itr=1; itr<=maxmitr; itr++)
    {
        h=f(x0)/df(x0);
        x1=x0-h;
        printf(" At Iteration no. %3d, x = %9.6f\n", itr, x1);
        if (fabs(h) < allerr)
        {
            printf("After %3d iterations, root = %8.6f\n", itr, x1);
            return 0;
        }
        x0=x1;
    }
    printf(" The required solution does not converge or iterations are insufficient\n");
    return 1;
}
```

For input, enter x0 allowed error and maximum iterations

2

0.0001

10

For output, run the code with above data.

Features of Regula Falsi Method:

Type – closed bracket

No. of initial guesses – 2

Convergence – linear

Rate of convergence – slow

Accuracy – good

Programming effort – easy

Approach – interpolation

Below is a short and simple source code in C program for regula falsi method to find the root of cos(x) – x*e^x. Here, x0 and x1 are the initial guesses taken.

Variables:

itr – a counter which keeps track of the no. of iterations performed

maxmitr – maximum number of iterations to be performed

x0, x1 – the limits within which the root lies

x2 – the value of root at nth iteration

x3 – the value of root at (n+1)th iteration

allerr – allowed error

x – value of root at nth iteration in the regula function f(x0), f(x1) – the values of f(x) at x0 and x1 respectively

f(x) = cos(x) – x*e^x

Source Code for Regula Falsi Method in C:

```
#include<stdio.h>

#include<math.h>

float f(float x)

{

    return cos(x) - x*exp(x);

}

void regula (float *x, float x0, float x1, float fx0, float fx1, int *itr)

{

    *x = x0 - ((x1 - x0) / (fx1 - fx0))*fx0;

    ++(*itr);

    printf("Iteration no. %3d X = %7.5f \n", *itr, *x);

}

void main ()

{

    int itr = 0, maxmitr;

    float x0,x1,x2,x3,allerr;

    printf("\nEnter the values of x0, x1, allowed error and maximum iterations:\n");

    scanf("%f %f %f %d", &x0, &x1, &allerr, &maxmitr);

    regula (&x2, x0, x1, f(x0), f(x1), &itr);

    do

    {
        if (f(x0)*f(x2) < 0)
            x1=x2;
        else
            x0=x2;
        regula (&x3, x0, x1, f(x0), f(x1), &itr);
        if (fabs(x3-x2) < allerr)
        {
            printf("After %d iterations, root = %6.4f\n", itr, x3);
            return 0;
        }
        x2=x3;
    }
    while (itr<maxmitr);
    printf("Solution does not converge or iterations not sufficient:\n");
    return 1;
}
```

For input, enter the value of x0, x1, allowed error and maximum iterations:

```
0
1
0.0005
20
```

For output, use above data and run the code.

Features of Fixed Point Iteration Method:

Type – open bracket

No. of initial guesses – 1

Convergence – linear

Rate of convergence – fast

Accuracy – good

Programming effort – easy

Approach – modification

Below is a source code in C program for iteration method to find the root of (cosx+2)/3. The desired degree of accuracy in the program can be achieved by continuing the iteration i.e. by increasing the maximum number of iterations.

f(x) = (cos(x) +2)/3

Source Code for Iteration Method in C:

```
#include<stdio.h>

#include<math.h>

float raj(float);

main()

{

   float a[100],b[100],c=100.0;

   int i=1,j=0;

   b[0]=(cos(0)-3*0+1);

   printf("\nEnter initial guess:\n");

   scanf("%f",&a[0]);

   printf("\n\n The values of iterations are:\n\n ");

   while(c>0.00001)

   {

      a[j+1]=raj(a[j]);
```

```
        c=a[j+1]-a[j];

        c=fabs(c);

        printf("%d\t%f\n",j,a[j]);

        j++;


    }

    printf("\n The root of the given function is %f",a[j]);

}

float raj(float x)

{

    float y;

    y=(cos(x)+2)/3;

    return y;

}
```

Input, Enter the initial guess

1

Output, run the code.

Features of Secant Method:

Type – open bracket

No. of initial guesses – 2

Convergence – super linear

Rate of convergence – faster

Accuracy – good

Programming effort – tedious

Approach – interpolation

Below is a short and simple C programming source code for Secant method to find the root of x^3 – 4 or x*x*x – 4.

f(x) = x*x*x – 4

Source Code for Secant Method in C:

```
#include<stdio.h>

float f(float x)

{

    return(x*x*x-4); // f(x)= x^3-4

}

float main()

{

    float a,b,c,d,e;

    int count=1,n;

    printf("\n\nEnter the values of a and b:\n"); //(a,b) must contain the solution.

    scanf("%f%f",&a,&b);

    printf("Enter the values of allowed error and maximun number of iterations:\n");

    scanf("%f %d",&e,&n);

    do

    {

        if(f(a)==f(b))

        {

            printf("\nSolution cannot be found as the values of a and b are same.\n");

        return;

        }
        c=(a*f(b)-b*f(a))/(f(b)-f(a));

        a=b;

        b=c;

        printf("Iteration No-%d    x=%f\n",count,c);

        count++;

        if(count==n)

        {

        break;

        }

    } while(fabs(f(c))>e);

    printf("\n The required solution is %f\n",c);


}
```

For input, enter the value of a and b:

2

3

For output, use above data and run the code.

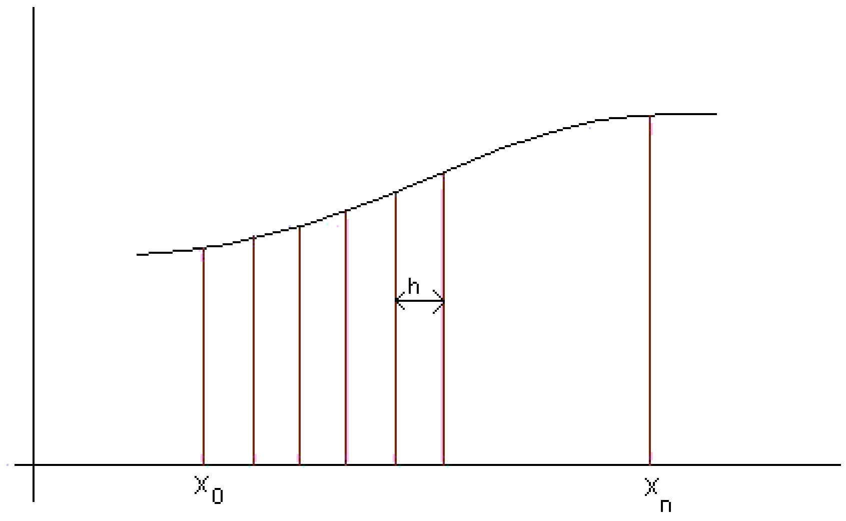
Chapter-4

Numerical Integration

1. Trapezoidal Method:

The basic working principle of the trapezoidal method c program is that trapezoidal method splits the area under the curve into a number of trapeziums. Then, area of each trapezium is calculated, and all the individual areas are summed up to give the total area under the curve which is the value of definite integral.

C program for Trapezoidal method - integral graph



In the above figure, area under the curve between the points x0 and xn is to be determined. In order to determine the area, we divide the total interval (xn– x0) into ‘n’ small interval each of length ‘h’:

h=(xn– x0)/n

After that, the C source code for trapezoidal method uses the following formula to calculate the value of definite integral:

$$\int_{x_0}^{x_n} f(x)\,dx = \frac{1}{2}h[(y_0 + y_n) + 2(y_1 + y_2 + \dots + y_{n-1})]$$

C program for Trapezoidal method - integration formula

```
#include<stdio.h>

#include<conio.h>
```

```
#include<math.h>

float f(float x)

{

   return(1/(1+pow(x,2)));

}

void main()

{

   int i,n;

   float x0,xn,h,y[20],so,se,ans,x[20];

   printf("\n Enter values of x0,xn,h:\n");

   scanf("%f%f%f",&x0,&xn,&h);

   n=(xn-x0)/h;

   if(n%2==1)

   {

      n=n+1;

   }

   h=(xn-x0)/n;

   printf("\nrefined value of n and h are:%d  %f\n",n,h);

   printf("\n Y values \n");

   for(i=0; i<=n; i++)

   {

      x[i]=x0+i*h;

      y[i]=f(x[i]);

      printf("\n%f\n",y[i]);

   }
   so=0;

   se=0;

   for(i=1; i<n; i++)

   {

      if(i%2==1)

      {

         so=so+y[i];

      }

      else

      {

         se=se+y[i];

      }

   }

   ans=h/3*(y[0]+y[n]+4*so+2*se);

   printf("\nfinal integration is %f",ans);

   getch();

}
```

It uses a user defined function to calculate the value of function i.e f(x) =1 /(1 + x2). The variable data type used in the program are integer and float types. There are only three variables to be input to the program – x0 (initial boundary value), xn (final boundary value) and h (strip length).

Input

Enter the value of x_0, x_n and h as 0, 3, 0.5.

Output

Run above code with given input.

2. Simpson's 1/3 :

In the source code below, a function f(x) = 1/(1+x) has been defined. The calculation using Simpson 1/3 rule in C is based on the fact that the small portion between any two points is a parabola. The program follows the following steps for calculation of the integral.

As the program gets executed, first of all it asks for the value of lower boundary value of x i.e. x0, upper boundary value of x i.e. xn and width of the strip, h.

Then the program finds the value of number of strip as n=(xn – x0)/h and checks whether it is even or odd. If the value of 'n' is odd, the program refines the value of 'h' so that the value of 'n' comes to be even.

After that, this C program calculates value of f(x) i.e 'y' at different intermediate values of 'x' and displays values of all intermediate values of 'y'.

After the calculation of values of 'c', the program uses the following formula to calculate the value of integral in loop.

Integral = *((y0 + yn) +4(y1 + y3 + ……….+ yn-1) + 2(y2 + y4 +……….+ yn-2))

Finally, it prints the values of integral which is stored as 'ans' in the program.

If f(x) represents the length, the value of integral will be area, and if f(x) is area, the output of Simpson 1/3 rule C program will be volume. Hence, numerical integration can be carried out using the program below; it is very easy to use, simple to understand, and gives reliable and accurate results.

f(x) = 1/(1+x)

Source Code for Simpson 1/3 Rule in C:

```
#include<stdio.h>

#include<conio.h>

float f(float x)

{

   return(1/(1+x));

}

void main()

{
```

```
int i,n;

float x0,xn,h,y[20],so,se,ans,x[20];

printf("\n Enter values of x0,xn,h: ");

scanf("%f%f%f",&x0,&xn,&h);

n=(xn-x0)/h;

if(n%2==1)

{

    n=n+1;

}

h=(xn-x0)/n;

printf("\n Refined value of n and h are:%d %f\n",n,h);

printf("\n Y values: \n");

for(i=0; i<=n; i++)

{

    x[i]=x0+i*h;

    y[i]=f(x[i]);

    printf("\n %f\n",y[i]);

}

so=0;

se=0;

for(i=1; i<n; i++)

{

    if(i%2==1)

    {

        so=so+y[i];
        }
        else
        {
            se=se+y[i];
        }
    }
    ans=h/3*(y[0]+y[n]+4*so+2*se);
    printf("\n Final integration is %f",ans);
    getch();
}
```

Input

Enter the value of x_0, x_n and h as 2, 4 and 0.5.

Output

Run the above code with given input.

Chapter-5

Ordinary Differential Equations

1. Euler's Method:

Mathematically, here, the curve of solution is approximated by a sequence of short lines i.e. by the tangent line in each interval. (Derivation) Using these information, the value of the value of ‘yn’ corresponding to the value of ‘xn‘ is to determined by dividing the length (xn – x) into n strips.

Therefore, strip width= (xn – x)/n and xn=x0+ nh.

Again, if m be the slope of the curve at point, y1= y0 + m(x0 , yo)h.

Similarly, values of all the intermediate y can be found out.

Below is a source code for Euler’s method in C to solve the ordinary differential equation dy/dx = x+y. It asks for the value of of x0 , y0 ,xn   and h. The value of slope at different points is calculated using the function ‘fun’.

The values of y are calculated in while loop which runs till the initial value of x is not equal to the final value. All the values of ‘y’ at corresponding ‘x’ are shown in the output screen.

dy/dx = x+y

Source Code for Euler’s Method in C:

```
#include<stdio.h>

float fun(float x,float y)

{

   float f;

   f=x+y;

   return f;

}

main()

{

   float a,b,x,y,h,t,k;

   printf("\nEnter x0,y0,h,xn: ");

   scanf("%f%f%f%f",&a,&b,&h,&t);

   x=a;

   y=b;

   printf("\n  x\t  y\n");
```

```
    while(x<=t)

    {

        k=h*fun(x,y);

        y=y+k;

        x=x+h;

        printf("%0.3f\t%0.3f\n",x,y);

    }

}
```

Input

give $x_{0,}$ $y_{0,}$ h and x_n as 0 , 1, 0.1, 1

Output

Run above code with given Input.

2. Modified Euler's Method:

The input data for Modified Euler's Method in C given below are initial and final values of x i.e. x0 and xn, initial value of y i.e y0 and the value of increment i.e h.

Here's how the C source code for Modified Euler's method works:

When the program is executed, it checks the error and if the program is found to be error free, the program runs. Firstly, it asks the value of initial condition i.e. initial value of x and y, value of increment h, and final value of x.

After getting the initial condition, this C program calls the defined function float fun(float,float) for the calculation of slope. The function for calculation of slope is:-

y' = x2 + y

Then, value of x is increased by h i.e. x1 = x0 + h and so on. Using the new value of x and y, new slope is calculated. Mean slope of this newly calculated and initial slope is calculated.

The new value of y is : y= y0 + h * slope. The value of x is not increased till the any two consecutive value of y are found to be same or with the given limit of tolerance.

The process is repeated till the initial value of x is not equal to the final value of x.

Finally, the program prints the intermediate as well as final values of x and corresponding y. These values can be transferred to Curve Software to obtain curve and its equations.

Source Code for Modified Euler's Method in C:

```
#include<stdio.h>

#include<math.h>

#include<string.h>

float fun(float,float);

main()

  {

     int i,j,c;

     float x[100],y[100],h,m[100],m1,m2,a,s[100],w;

     printf("\n     C program for Modified Euler Method \n\n");

     printf("  Enter the initial value of x:");

     scanf("%f",&x[0]);

     printf("\n  Enter the value of increment h:");

     scanf("%f",&h);

     printf("\n  Enter the final value of x:");

     scanf("%f",&a);

     printf("\n  Enter the initial value of the variable y :");

     scanf("%f",&y[0]);

     s[0]=y[0];

     for(i=1;x[i-1]<a;i++)
```

```
        {
            w=100.0;
           x[i]= x[i-1]+h;
           m[i]=fun(x[i-1],y[i-1]);
           c=0;
           while(w>0.0001)
            {
                m1=fun(x[i],s[c]);
                m2=(m[i]+m1)/2;
                s[c+1]=y[i-1]+m2*h;
                w=s[c]-s[c+1];
                w=fabs(w);
                c=c+1;
            }
          y[i]=s[c];
        }
      printf("\n\n The respective values of x and y are\n     x  \t     y\n\n");
      for(j=0;j<i;j++)
        {
            printf("  %f\t%f",x[j],y[j]);
            printf("\n");
        }
   }
float fun(float a,float b)
   {
      float c;

      c=a*a+b;

      return(c);

   }
```

Input

Enter the initial value of x as 0

Enter the value of increment h as 0.05

Enter the final value of x as 0.1

Enter the initial value of y as 1

Output

run above code with given inputs.

3. Runge-Kutta order 4 Method:

The source code below to solve ordinary differential equations of first order by RK4 method first asks for the values of initial condition i.e. user needs to input x0 and y0 . Then, the user has to define increment 'h' and the final value of x at which y is to be determined.

In this program for Runge Kutta method in C, a function f(x,y) is defined to calculate slope whenever it is called.

f(x,y) = (x-y)/(x+y)

Source Code for Runge Kutta Method in C:

```
#include<stdio.h>

#include<math.h>

float f(float x,float y);

int main()
```

```
{
    float x0,y0,m1,m2,m3,m4,m,y,x,h,xn;
    printf("Enter x0,y0,xn,h:");
    scanf("%f %f %f %f",&x0,&y0,&xn,&h);
    x=x0;
    y=y0;
    printf("\n\nX\t\tY\n");
    while(x<xn)
    {
        m1=f(x0,y0);
        m2=f((x0+h/2.0),(y0+m1*h/2.0));
        m3=f((x0+h/2.0),(y0+m2*h/2.0));
        m4=f((x0+h),(y0+m3*h));
        m=((m1+2*m2+2*m3+m4)/6);
        y=y+m*h;
        x=x+h;
        printf("%f\t%f\n",x,y);
    }
}
float f(float x,float y)
{
    float m;
    m=(x-y)/(x+y);
    return m;
}
```

Input

Enter X_0, $y_{0,}$ x_n and h as 0, 2, 2 and 0.5

Output

run above code with given input.