Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Lab Report

Topic : Numerical Solution of Ordinary Differential Equation

Course Title : Sessional Based on CSE 2103
Course Code : CSE 2104

Submitted By :
Md. Tanzid Hasan
Section : A
Roll No : 1603054

Submitted To :

Shyla Afroge

Assistant Professor

Dept. of Computer Science & Engineering

Date : 26-06-2018

Experiment No : 01

Experiment Name : Modified Euler Method

Theory : Considering the differential equation

$$y' = f(x, y) \qquad \text{.....(1a)}$$

with the initial condition

$$y(x_0) = x_0 \qquad \text{.....(1b)}$$

Suppose that we wish to solve the equation (1) for values of y at $x = x_r = x_0 + rh$ (r = 1, 2, …..). Integrating Eq. (1), we obtain

$$y_1 = y_0 + \int_{x_0}^{x_1} f(x, y)dx \qquad \text{.....(2)}$$

Assuming that $f(x, y) = f(x_0, y_0)$ in $x_0 \leq x \leq x_1$, this gives Euler's formula

$$y_1 = y_0 + hf(x_0, y_0) \qquad \text{.....(3a)}$$

Similarly for the range $x_1 \leq x \leq x_2$, we have

$$y_2 = y_1 + \int_{x_0}^{x_1} f(x, y)dx$$

Substituting $f(x_1, y_1)$ for $f(x, y)$ in $x_1 \leq x \leq x_2$ we obtain

$$y_2 = y_1 + hf(x_1, y_1) \qquad \text{.....(3b)}$$

Proceeding in this, we obtain the general formula

$$y_{n+1} = y_n + hf(x_n, y_n) \qquad \text{.....(4)}$$

The process is very slow and to obtain reasonable accuracy with Euler's method, we need to take a smaller value for h.

Instead of approximating $f(x, y)$ by $f(x_0, y_0)$ in (4), we now approximate the integral in (4) by means of trapezoidal rule to obtain

$$y_1 = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1)] \qquad \text{.....(5)}$$

We thus obtain the iteration formula

$$y_1^{(n+1)} = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{n})], \quad n = 0, 1, 2, \ldots \qquad \text{.....(6)}$$

where y_1^{n} is the n^{th} approximation to y_1. The iteration formula (6) can be started by choosing y_1^{0} from Euler's formula :

$$y_1^{(0)} = y_0 + h[f(x_0, y_0)]$$

Code :

```
#include <bits/stdc++.h>
using namespace std;
double df(double f, double g)
{
   double r=(3*f)+(g/2);
   return r;
}
int main()
{
   double h,y0,y1,x0,x,k1,k2;
   cout<<"Enter x0 : ";
   cin>>x0;
   cout<<"Enter y0 : ";
   cin>>y0;
   cout<<"Enter x : ";
   cin>>x;
   for(;;)
   {
      cout<<endl<<"Enter h (Enter 0 to break) : ";
      cin>>h;
      if(h==0)
         break;
      int a=(x-x0)/h;
      double c=0;
      for(int i=0;i<a;i++)
      {
         k1=df(x0+c,y0);
         y1=y0+(h*df(x0+c,y0));
```

```
            k2=df(x0+c+h,y1);
            y0=y0+(h/2)*(k1+k2);
            c=c+h;
            cout<<"y"<<i+1<<" : "<<y0<<endl;
            cout<<endl;
        }
        cout<<"y("<<x<<") : "<<y0<<endl;
    }
    return 0;
}
```

Input :

Enter x0 : 0

Enter y0 : 1

Enter x : 0.2

Enter h (Enter 0 to break) : 0.05

Output :

y(0.2) : 1.16708

Discussion : In this program, initial value of x and y were taken as input. Then, for which x the output was to be shown, that desired x was taken as input also. With x , h was also taken in input as it is the difference between each x. After that, the solution of Modified Euler's Method was algorithmize. The process is faster than Euler's formula.

Experiment No : 02

Experiment Name : 2nd Order Runge-Kutta Method

Theory : Euler's method is less efficient in practical problems since it requires h to be small for obtaining reasonable accuracy. The Runge-Kutta methods are designed to give greater accuracy and they possess the advantage of requiring only the function values at some selected points on the subinterval.
If we substitute $y_1 = y_0 + hf(x_0, y_0)$ on the right side of $y_1 = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1)]$, we obtain

$$y_1 = y_0 + \frac{h}{2}[f_0 + f(x_0 + h, y_0 + hf_0)]$$

where $f_0 = f(x_0, y_0)$. If we now set

$$k_1 = hf_0 \text{ and } k_2 = hf(x_0 + h, y_0 + k_1)$$

then the above equation becomes

$$y_1 = y_0 + \frac{1}{2}(k_1 + k_2), \qquad \ldots\ldots\ldots\ldots(1)$$

where is the second order Runge-Kutta formula.

Code :

```
#include <bits/stdc++.h>
using namespace std;
double df(double f,double g)
{
   double r=(3*f)+(g/2);
   return r;
}
int main()
{
   double h,y0,x0,x,k1,k2;
   cout<<"Enter x0 : ";
   cin>>x0;
   cout<<"Enter y0 : ";
   cin>>y0;
```

```
    cout<<"Enter x : ";
    cin>>x;
    for(;;)
    {
        cout<<endl<<"Enter h (Enter 0 to break) : ";
        cin>>h;
        if(h==0)
            break;
        int a=(x-x0)/h;
        double c=0;
        for(int i=0;i<a;i++)
        {
            k1=h*df(x0+c,y0);
            cout<<"k1 : "<<k1<<endl;
            k2=h*df(x0+h+c,y0+k1);
            cout<<"k2 : "<<k2<<endl;
            y0=y0+(0.5)*(k1+k2);
            cout<<"y0 : "<<y0<<endl;
            c=c+h;
            cout<<endl;
        }
        cout<<"y("<<x<<") : "<<y0<<endl;
    }
    return 0;
}
```

Input :

Enter x0 : 0

Enter y0 : 1

Enter x : 0.2

Enter h (Enter 0 to break) : 0.05

Output :

y(0.2) : 1.16708

Discussion : In this program, initial value of x and y were taken as input. Then, for which x the output was to be shown, that desired x was taken as input also. With x , h was also taken in input as it is the difference between each x. After that, the solution of 2^{nd} Order Runge-Kutta Method was algorithmize. The process is faster than Modified Euler's formula and to obtain reasonable accuracy with 2^{nd} Order Runge-Kutta Method, we need to take a smaller value of h.

Experiment No : 03

Experiment Name : 4th Order Runge-Kutta Method

Theory : The error in this formula can be shown to be of order h^3 by expanding both sides, by Taylor's series. Thus, the left side, gives

$$y_0 + hy_0' + \frac{h^2}{2}y_0'' + \frac{h^3}{6}y_0''' + \cdots$$

and on the right side

$$k_2 = hf(x_0 + h, y_0 + hf_0) = h[f_0 + h\frac{\partial f}{\partial x_0} + hf_0\frac{\partial f}{\partial y_0} + O(h^2)]$$

Since

$$\frac{df(x,y)}{dx} = \frac{\partial f}{\partial x} + f\frac{\partial f}{\partial y}$$

we obtain

$$k_2 = h[f_0 + hf_0' + O(h^2)] = hf_0 + h^2f_0' + O(h^3)$$

so that the right side of 2nd Order Runge-Kutta equation gives

$$y_0 + \frac{1}{2}[hf_0 + hf_0 + h^2f_0' + O(h^3)] = y_0 + hf_0 + \frac{h^2}{2}f_0' + O(h^3)$$
$$= y_0 + hy_0' + \frac{h^2}{2}y_0'' + O(h^3)$$

It therefore follows that the Taylor series expansions of both sides of 2nd Order Runge-Kutta equation agree up to terms of order h^3 which means that the error in this formula is of order h^3.

More generally, if we set

$$y_1 = y_0 + W_1k_1 + W_2k_2 \qquad \text{.....(1a)}$$

where

$$\left.\begin{aligned} k_1 &= hf_0 \\ k_2 &= hf(x_0 + \alpha_0 h, y_0 + \beta_0 k_1) \end{aligned}\right\} \qquad \text{.....(1b)}$$

Then the Taylor series expansions of both sides of the last equation in Eqs. (1a) gives the identity

$$y_0 + hf_0 + \frac{h^2}{2}\left(\frac{\partial f}{\partial x} + f_0\frac{\partial f}{\partial y}\right) + O(h^3)$$
$$= y_0 + (W_1 + W_2)hf_0 + W_2h^2\left(\alpha_0\frac{\partial f}{\partial x} + \beta_0 f_0\frac{\partial f}{\partial y}\right) + O(h^3)$$

Equating the coefficient of $f(x, y)$ and its derivatives on both sides, we obtain the relations

$$W_1 + W_2 = 1, \qquad W_1\alpha_0 = \frac{1}{2}, \qquad W_2\beta_0 = \frac{1}{2} \qquad \ldots\ldots\ldots(2)$$

Clearly, $\alpha_0 = \beta_0$ and if α_0 is assigned any value arbitrarily, then the remaining parameters can be determined uniquely. If we set, for example, $\alpha_0 = \beta_0 = 1$, then we immediately obtain $W_1 = W_2 = \frac{1}{2}$, which gives formula 2nd Order Runge-Kutta equation.

It follows, therefore, that thee are several Second Order Runge-Kutta formula and that formula (1) and (2) constitute just one of several such formula.

Higher-order Runge-Kutta formula exist, of which we mention only the the 4th Order formula defined by

$$y_1 = y_0 + W_1k_1 + W_2k_2 + W_3k_3 + W_4k_4 \qquad \ldots\ldots\ldots\ldots..(3a)$$

where

$$\left.\begin{aligned} k_1 &= hf(x_0, y_0) \\ k_2 &= hf(x_0 + \alpha_0 h, y_0 + \beta_0 k_1) \\ k_3 &= hf(x_0 + \alpha_1 h, y_0 + \beta_1 k_1 + v_1 k_2) \\ k_4 &= hf(x_0 + \alpha_2 h, y_0 + \beta_2 k_1 + v_2 k_2 + \delta_1 k_3) \end{aligned}\right\} \qquad \ldots\ldots\ldots\ldots....(3b)$$

where the parameters have to be determined by expanding both sides of the first equation of (3a) by Taylor's series and securing agreement of terms up to and including those containing h^4.

Code :

```
#include <bits/stdc++.h>
using namespace std;
double df(double f,double g)     {
   double r=(3*f)+(g/2);
   return r;
}
int main()   {
   double h,y0,x0,x,k1,k2,k3,k4;
   cout<<"Enter x0 : ";
   cin>>x0;
   cout<<"Enter y0 : ";
```

```
cin>>y0;
cout<<"Enter x : ";
cin>>x;
for(;;)
{
   cout<<endl<<"Enter h (Enter 0 to break) : ";
   cin>>h;
   if(h==0)
      break;
   int a=(x-x0)/h;
   double c=0;
   for(int i=0;i<a;i++)
   {
      k1=h*df(x0+c,y0);
      cout<<"k1 : "<<k1<<endl;
      k2=h*df(x0+(h/2)+c,y0+(k1/2));
      cout<<"k2 : "<<k2<<endl;
      k3=h*df(x0+(h/2)+c,y0+(k2/2));
      cout<<"k3 : "<<k3<<endl;
      k4=h*df(x0+h+c,y0+k3);
      cout<<"k4 : "<<k4<<endl;
      y0=y0+(1.0/6)*(k1+(2*k2)+(2*k3)+k4);
      cout<<"y0 : "<<y0<<endl;
      c=c+h;
      cout<<endl;
   }
   cout<<"y("<<x<<") : "<<y0<<endl;
}
return 0;   }
```

Input :

Enter x0 : 0

Enter y0 : 1

Enter x : 0.2

Enter h (Enter 0 to break) : 0.05

Output :

y(0.2) : 1.16722

Discussion : In this program, initial value of x and y were taken as input. Then, for which x the output was to be shown, that desired x was taken as input also. With x , h was also taken in input as it is the difference between each x. After that, the solution of 4^{th} Order Runge-Kutta Method was algorithmize. The process is faster than 2^{nd} Order Runge-Kutta Method.