Date : Page No :

Assignment Number

Problem Statement

Program in C to approximate the solution of a given differential equation using 4th order Runge Kutta method.

Theory

The 4th-order Runge Kutta method is similar to Simpson's rule: a sample of the slope is made at the mid-point on the interval, as well as the end points, and a weighted average is taken, placing more weight on the slope at the mid point.

It should be noted that Runge-Kutta refers to an entire class of IVP solvers, which includes Euler's method and Heun's method. We are looking at one particularly effective, yet simple, case.

Given the IVP

$$y^{(1)}(t) = f(\, t, y(t)\,)$$
$$y(t_0) = y_0$$

if we want to estimate $y(t_1)$, we set $h = t_1 - t_0$. Remember that $f(\, t, y\,)$ gives the slope at the point (t, y). Thus, we can find the slope at (t_0, y_0):

$$K_0 = f(\, t_0, y_0\,)$$

Next, we use this slope to estimate y(t_0 + h/2) ≈ y_0 + ½ hK_0 and sample the slope at this intermediate point:

$$K_1 = f(\ t_0 + ½h, y_0 + ½h\ K_0\ )$$

Using this new slope, we estimate y(t_0 + h/2) ≈ y_0 + ½ hK_1 and sample the slope at this new point:

$$K_2 = f(\ t_0 + ½h, y_0 + ½h\ K_1\ )$$

Finally, we use this last approximation of the slope to estimate y(t_1) = y(t_0 + h) ≈ y_0 + hK_2 and sample the slope at this point:

$$K_3 = f(\ t_0 + h, y_0 + h\ K_2\ )$$

All four of these slopes, K_0, K_1, K_2, and K_3 approximate the slope of the solution on the interval [t_0, t_1], and therefore we take the following weighted average:

$$(k_0 + 2*k_1 + 2*k_2 + k_3) / 6$$

Therefore, we approximate y(t_1) by

$$y(t_1) = y(t_0) + (k_0 + 2*k_1 + 2*k_2 + k_3) / 6$$

Note how the two values K_1 and K_2 both approximate the slope at the midpoint. Thus, the averaging is very similar to Simpson's rule which also places a weight of 4 on the midpoint.

Algorithm

Input :

1. A differential equation, say f(x, y)
2. Initial value of x, say x0
3. Initial value of y, say y0
4. Final value of x to find the value of y for, say xn
5. Width of each step, say h

Output :

Steps :

```
Step 1 : At first,  f(x, y) = dy/dx is to be defined
Step 2 : Input x0
Step 3 : Input y0
Step 4 : Input steps
Step 5 : Input xn
Step 6 : h = (xn-x0)/h
Step 7 : x = x0
Step 8 : y = y0
Step 9 : k1 = h * f(x ,y)
Step 10 :      k2 = h * f(x+h/2, y+k1/2)
Step 11 :      k3 = h * f(x+h/2, y+k2/2)
Step 12 :      k4 = h * f(x+h/2, y+k3/2)
Step 13 :      y = y + (k1+2k2+2k3+k4)/6
Step 14 :      x = x + h
Step 15 :      step = step - 1
Step 16 :      If(step != 0) Then
    1. goto step 9
Step 17 :      Print “y at x = ” x “ : ” y
```

Source Code

```
#include <stdio.h>

double f(double x, double y){
   return (x – y)/2; // the differential equation
}

int main(){
   double x0, y0, xn;
   int steps;
   printf("\nEnter x0 : ");
   scanf("%lf", &x0);
   printf("\nEnter value of y(%g) : ", x0);
   scanf("%lf", &y0);
   printf("\nEnter xn : ");
   scanf("%lf", &xn);
   printf("\nEnter number of steps : ");
   scanf("%d", &steps);

   double h = (xn - x0)/steps, x = x0, y = y0;
   while(steps--){
      double k1 = h*f(x, y);
      double k2 = h*f(x + h/2, y + k1 / 2);
      double k3 = h*f(x + h/2, y + k2 / 2);
      double k4 = h*f(x + h, y + k3);
      y += (k1 + 2*k2 + 2*k3 + k4)/6;
      x += h;
   }
   printf("\ny(%g) : %g", x, y);

   return 0;
}
```

Input and Output

Set 1:

Differential Equation : 2*x*y

```
Enter x0 : 1

Enter value of y(1) : 1

Enter xn : 1.1

Enter number of steps : 10

y(1.1) : 1.23368
```

Set 2:

Differential Equation : 1 + y/x

```
Enter x0 : 1

Enter value of y(1) : 1

Enter xn : 3

Enter number of steps : 10

y(3) : 6.2958
```

Discussion

1. In summary, then, each of the k_i gives us an estimate of the size of the *y*-jump made by the actual solution across the whole width of the interval. The first one uses Euler's Method, the next two use estimates of the slope of the solution at the midpoint, and the last one uses an estimate of the slope at the right end-point. Each k_i uses the earlier k_i as a basis for its prediction of the *y*-jump.
2. This means that the Runge-Kutta formula for y_{n+1}, namely:

   $$y_{n+1}=y_n+(1/6)(k_1+2k_2+2k_3+k_4)$$

   is simply the *y*-value of the current point plus a weighted average of four different *y*-jump estimates for the interval, with the estimates based on the slope at the midpoint being weighted twice as heavily as the those using the slope at the end-points.
3. As we have just seen, the Runge-Kutta algorithm is a little hard to follow even when one only considers it from a geometric point of view. In reality the formula was not originally derived in this fashion, but with a purely analytical approach. After all, among other things, our geometric "explanation" doesn't even account for the weights that were used. If you're feeling ambitious, a little research through a decent mathematics library should yield a detailed analysis of the derivation of the method.