2nd Order Runge-Kutta Method and its C-programming

Course: MPHYCC-05 Modeling and Simulation, MPHYCC-09 Lab-II
(M.Sc. Sem-II)

By

Dr. Sanjay Kumar

Assistant Professor
Department of Physics

Patna University

Contact Details: Email- sainisanjay35@gmail.com;
Contact No.- 9413674416

Runge-Kutta Methods

Consider the typical differential equation:

$$\frac{dy}{dx}=f(x,y) \quad \text{with the initial condition} \quad y(x=x_0)=y_0 \ .$$

Our aim is to find out y(x) on a descretized space.

From the Euler's method (discussed in the previous lecture), the solution is given as:

$$y_{i+1}=y_i+h\ f(x_i\ ,y_i) \qquad \text{(here we use the notations: } x(i)=x_i,\ y(i)=y_i)$$

where $h=x_{i+1}-x_i$

$$y_{i+1}=y_i+h\ y'_i \qquad \text{with } y'_i=f(x_i, y_i)$$

We can derive the above formula from the Taylor's series expansion;

$$y_{i+1} = y(x_{i+1} + h) = y_i + h\ y_i' + (h^2/2)(y_i'')+...$$

Only considering first order term:

$$y_{i+1} = y(x_{i+1} + h) = y_i + h\ y_i' = y_i+h\ f(x_i\ ,y_i)$$

Thus the Euler's method is first order accurate because we neglect the higher order term of the Taylor's series which leads to the trucation error ~ $O(h^2)$. As a result, sometime the Euler's method is known as first order Runge-Kutta Method.

2^{nd} Order Runge-Kutta Method: Geometric Interpretation

To understand the 2nd order Runge Kutta method, we once again consider the typical first order differential equation:

$$\frac{dy}{dx}=f(x,y) \quad \text{with the initial condition} \quad y(x=x_1)=y_1$$

Now let's assume h to be the equidistance value of x, i.e.,

$x_2 = x_1+h$; $x_3 = x_2+h$; ..... ; $x_{i+1} = x_i+h$

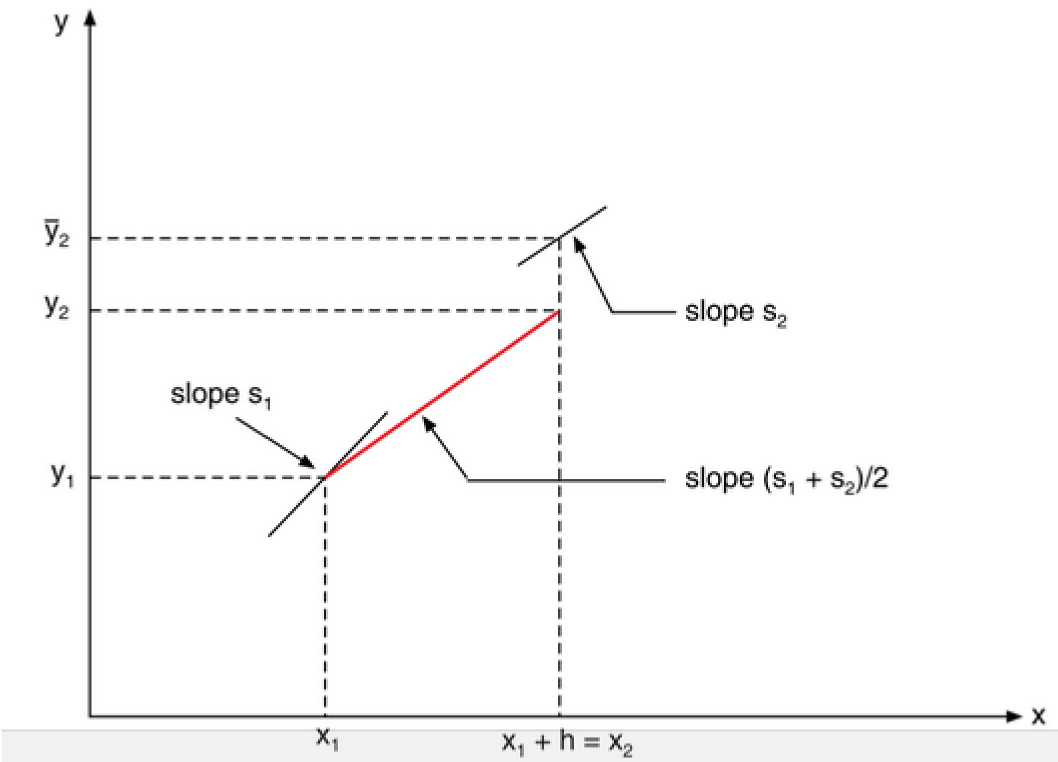

Figure: The figure geometrically illustrates 2nd order Runge-Kutta method.

To understand the 2nd order Runge-Kutta method refers to the above figure. First of all, we calculate the slope $s_1=f(x_1,y_1)$ of the solution curve y(x) at point (x_1,y_1). Then, lets draw a straight line from the initial point (x_1,y_1) with the slope s_1. Lets assume that the straight line cuts the vertical line through x_1+h at $(x_1+h,\ \hat{y}_2)$. Note that by definition $s_1=(\hat{y}_2 - y_1)/(x_2-x_1)=(\hat{y}_2 - y_1)/h$. This implies that $\hat{y}_2 = y_1 + s_1 h$.

Now, determine the slope of the solution curve y(x) at the point $(x_1+h,\ \hat{y}_2)$. This is given by $s_2=f(x_1+h,\ \hat{y}_2)= f(x_2,\ y_1 + s_1 h)$. Now, go back to the point (x_1,y_1) and draw a straight line with a slope $s=(s_1+s_2)/2$. In the 2nd order Runge-Kutta method, the point y_2, where this straight line cuts the vertical line x_1+h, is the approximate solution of the considered differential equation at the point x_1+h. By definition of the slope:

$$s=(y_2-y_1)/(x_2-x_1)=(y_2-y_1)/h$$

$$y_2 = y_1 + h\, s$$

$$y_2 = y_1 + h\, (s_1+s_2)/2$$

where $s_1=f(x_1, y_1)$ and $s_2= f(x_2, y_1 + s_1 h)$.

In general, the $(i+1)^{th}$ is obtained from the i^{th} point using the formula:

$$y_{i+1} = y_i + h\, (s_i+s_{i+1})/2$$

where $s_i=f(x_i, y_i)$ and $s_2= f(x_{i+1}, y_i + s_i h)$.

This method is also known as Heun's method. Thus the Runge-Kutta 2nd order method is second-order accurate i.e., from the Taylor's series expansion we can show that the trucation error ~ $O(h^3)$.

Assignment

Solve the differential equation using Runge-Kutta 2nd order method

$$\frac{dy}{dx}=-y$$

find y for x ε [0, 2] with the initial condition $y(x=0)=y_0=1$.

Algorithm to Write a Program of the Runge-Kutta 2nd order method

Problem: $\frac{dy}{dx}=f(x,y)$ *with the initial condition* $y(x=x_0)=y_0$ *find y(x) for $x_0<x<L$*

1. Input x_0, L, y_0, n

2. $h=(x_n-x_0)/n$

3. Do iteraction (i=1,n)
{$s_1=h*f(x_0, y_0)$
$x_1=x_0+h$;
$s_2=h*f(x_1,y_0+s_1)$
$y_1=y_0+0.5*(s_1+s_2)$
write x_1, y_1
$y_0=y_1$
$x_0=x_1$ }

4. end

C- Program of the Runge-Kutta 2nd order method

Problem: $\frac{dy}{dx}=-y$ *with the initial condition* $y(x=0)=1$ *find y(x) for $0<x<2$*

```
#include <stdio.h>
#include <math.h>
int main()
{float k1, k2, x0, l, y0, h, x1, y1;
int n,i;
printf("enter the value of n \n");
scanf("%d",&n);
printf("enter the initial point x0, last point L and initial condition y0:\n");
```

```
scanf("%f %f %f",&x0,&l,&y0);
h=(l-x0)/n;

for(i=1;i<=n;i++)
{x1=x0+h;
k1=-h*y0;
k2=-h*(y0+k1);
y1=y0+0.5*(k1+k2);
printf("x[%d] and y[%d]:%f\t\t%f \n",i,i,x1,y1);
x0=x1;
y0=y1;}

return 0;}
```

Output of the program:

```
enter the value of n
5
enter the initial point x0, last point L and initial condition y0:
0 2 1
x[1] and y[1]:0.400000          0.680000
x[2] and y[2]:0.800000          0.462400
x[3] and y[3]:1.200000          0.314432
x[4] and y[4]:1.600000          0.213814
x[5] and y[5]:2.000000          0.145393
```

\

C- Program of the Runge-Kutta 2nd order method using 1D Array

Problem: $\frac{dy}{dx}=-y$ *with the initial condition* $y(x=0)=1$ *find y(x) for 0 <x<2*

```
#include <stdio.h>
#include <math.h>

int main()
{float k1[100],k2[100],x[100],y[100],y1[100],h;
```

```
int n,i;

printf("enter the value of n and h \n");
scanf("%d%f",&n,&h);
printf("enter the value of x[1] and y[1]:\n");
scanf("%f%f",&x[1],&y[1]);

for(i=1;i<=n+1;i++)
{k1[i]=-h*y[i];
x[i+1]=x[i]+h;
y[i+1]=y[i]+k1[i];
k2[i]=-h*y[i+1];
y[i+1]=y[i]+0.5*(k1[i]+k2[i]);}
for(i=2;i<=n+1;i++)
printf("x[%d] and y[%d]:%f  %f \n",i,i,x[i],y[i]);
return 0;}
```

Output of the program:

```
enter the value of n and h
5   0.4
enter the value of x[1] and y[1]:
0    1
x[2] and y[2]:0.400000  0.680000
x[3] and y[3]:0.800000  0.462400
x[4] and y[4]:1.200000  0.314432
x[5] and y[5]:1.600000  0.213814
x[6] and y[6]:2.000000  0.145393
```