Runge-Kutta Method

1 Introduction

The Runge-Kutta Method is a numerical method used to approximate the solution to ordinary differential equations. The essence of the Runge-Kutta method lies in its ability to estimate the solution at a given point by considering not just the slope at the initial point (as in Euler's method) but also by incorporating information about the slope at several intermediate points within each step. This approach allows the method to approximate the true solution curve better, resulting in more accurate and stable results.
Among the Runge-Kutta methods of different orders, the one discussed in this documentation is the Runge-Kutta method of fourth order (often referred to simply as "RK4"). RK4 evaluates the slope four times within each step and uses a weighted average of these slopes to compute the next value of the solution. This method ensures the accuracy and efficiency of the calculation.

2 Runge-Kutta Method

Let us have the following equation and eventually, we want to find y(t),

$$\frac{d^2y}{dt^2} + A\frac{dy}{dt} + By = 0 \tag{1}$$

In **Runge-Kutta method**, numerical integration of ordinary differential equations is achieved by using trial steps that correspond to intervals designed to cancel out lower-order error terms. Starting with the first-order approximation,

$$\frac{dy}{dt} = \frac{y(t+\Delta t) - y(t)}{\Delta t} = f(y,t) \tag{2}$$

$$y(t+\Delta t) = \Delta t f(y,t) + y(t) \tag{3}$$

$$y(t+1) = y(t) + \Delta t f(y,t) \tag{4}$$

2.1 Geometrical Interpretation

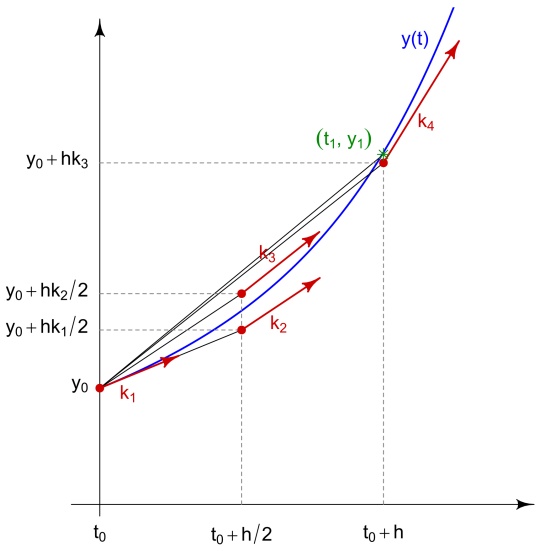

Figure 1: Approximation via Runge-Kutta Method Method

The time interval is of size h = Δt and it is further divided into sub-intervals to ensure that the numerically computed curve is as close as possible to the actual curve.
First of all, the slope at y_0 is computed, called k_1, then using k_1 the slope at the midpoint of the interval is computed, called k_2. Further using k_2, the slope at the same point is computed to get closer to the actual curve called k_3. At last, using k_3, the slope at the endpoint of the interval i.e., k_4 is computed. From the figure above, it is clear that the curves having slopes k_1, k_2, k_3, and k_4(the red ones) are a good approximation to the actual blue curve. Finally, the weighted average of all these slopes is taken to approximate the curve at each time step of the loop.

2.2 Formulation of the Method

For a step size Δ t

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5}$$

$$t_{n+1} = t_n + h \tag{6}$$

where

$$k_1 = f(y_n, t_n) \tag{7}$$

$$k_2 = f(y_n + \frac{\Delta t}{2}k_1, t_n + \frac{\Delta t}{2}) \tag{8}$$

$$k_3 = f(y_n + \frac{\Delta t}{2}k_2, t_n + \frac{\Delta t}{2}) \tag{9}$$

$$k_4 = f(y_n + \Delta t k_3, t_n + \Delta t) \tag{10}$$

3 Error Analysis

3.1 Local Truncation Error

The local truncation error for RK4 is of order $O(h^5)$, where h is the step size. This means that the error introduced in a single step varies with the fifth power of the step size.

3.2 Global Truncation Error

The global truncation error, which accumulates over all the steps, is of order $O(h^4)$. This is because the local errors accumulate over approximately $1/h$ steps.

3.3 Error Term Derivation

To understand how these error terms arise, consider the Taylor series expansion of the solution:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + \frac{h^4}{4!}y^{(4)}(t) + O(h^5) \tag{11}$$

4 Error Terms

To see how the error terms and finally the truncation errors are approximated for the Runge-Kutta method, we have taken a simpler case of a first-order differential equation, which can be extended to higher orders in the same manner but will be complex. The Governing Equation is:

$$\frac{df}{dt} = \frac{f(t+\Delta t) - f(t)}{\Delta t} \tag{12}$$

If we write the Taylor expansion of the RHS, we get

$$\frac{df}{dt} = (\frac{f(t)}{\Delta t} + \frac{df(t)}{dt} + \frac{\Delta t}{2!}\frac{d^2f(t)}{dt^2} + \frac{(\Delta t)^2}{3!}\frac{d^3f(t)}{dt^3} + ...) - \frac{f(t)}{\Delta t} \tag{13}$$

Hence, we get

$$\frac{df}{dt} = \frac{df}{dt} + \frac{\Delta t}{2!}\frac{d^2f(t)}{dt^2} + \frac{(\Delta t)^2}{3!}\frac{d^3f(t)}{dt^3} + ... \tag{14}$$

Only the first term of the RHS matches the LHS, and the rest are error terms. Finally, we can write

$$\frac{df}{dt} = \frac{df}{dt} + error(O(\Delta t)) \tag{15}$$

Hence, the local error is of order Δ t for the first-order differential part approximated via numerical methods.
In the same way, if we do it for higher order differential terms, then, finally, we will get the following values for Runge-Kutta Method: **Local error** in the Runge-Kutta Method of 4th order represents the error about one step and it is $O(\Delta t)^5$
Global error in the Runge-Kutta Method of 4th order represents the error across the whole trajectory and it is $O(\Delta t)^4$

5 Implementing Runge-Kutta Method in C++

The following is how you can implement the Runge-Kutta method as a numerical technique in a C++ program:
Let's define a **for loop** that will perform the same function as discussed above in Section 2.2:

```
// Part of the code for implementing the Runge-Kutta method

for(double t=0 ;t<100 ;t+=h) // h is the step size
{ // f() is a function that will perform the necessary operation where t,y, and v are all
    initial values of the parameters involved
   double k1 = h * f(t,y,v);
   double k2 = h * f(t + 0.5 * h, y + 0.5 * k1, v + 0.5 * k1);
   double k3 = h * f(t + 0.5 * h, y + 0.5 * k2, v + 0.5 * k2);
   double k4 = h * f(t + h, y + k3, v + k3);
   y += (1.0 / 6.0 * k1 + 1.0 / 3.0 * k2 + 1.0 / 3.0 * k3 + 1.0 / 6.0 * k4);
}
```

The Runge-Kutta method of order 4 is better than the Euler Method in terms of efficiency and accuracy. As with the Euler Method, reducing the step size too much can lead to numerical instability and increased computational cost, despite lower truncation errors.