Xingcheng Zhang

CSc-301- P

Prof. Erik Grimmelmann

12/20/2018

Runge Kutta method to solve differential equation

In numerical analysis, the Runge Kutta method is a family of implicit and explicit iterative methods, including the well-known program called the Euler method, used for time discretization and approximate solutions of ordinary differential equations. These methods were developed around 1900 by German mathematicians Carl Runge and Martin Kutta.

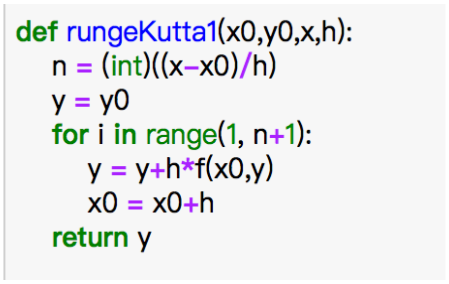
The Runge Kutta method is a high-precision single-step algorithm that is widely used in engineering. Due to the high precision of this algorithm, measures are taken to suppress the error, so the implementation principle is also complicated. The algorithm is built on top of mathematical support.

In my program, I used four Runge Kutta's methods to compare the difference among these methods with different steps. Of course, there are still different methods. For example, the midpoint method is also widely used in math field. Most efforts to increase the order of Runge Kutta method, have been accomplished by increasing the number of Taylor's series terms used and thus the number of function evaluations.

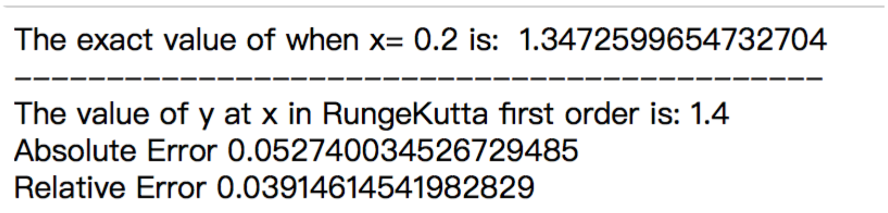
However, the simplest Runge–Kutta method is the (forward) Euler method, the original formula is yi+1 = yi + h*ki, where h is the step size, the expression of yi+1 is exactly the same as the first two terms of the Taylor expansion of y(xi+1), that is, the local truncation error is O(h2). When the arithmetic mean of the slope approximation k1 at point xi and the slope k2 at the right endpoint xi+1 is taken as an approximation of the average slope k*, then an improved Euler's formula of second-order accuracy is obtained: yi+1 = yi + h(k1 + k2), where k1=f(xi,yi), k2=f(xi+h,yi+hk1) and so on, if the slope values k1,k2 of several stores are estimated in the interval [xi,xi+1], ...,km, and using their weighted average as an approximation of the average slope k*, it is clear that a high-order counting formula with very high precision can be constructed.

The above two sets of formulas have the same in common in the form: they all use f(x, y) to be linearly combined at some points to obtain the approximate value yi+1 of y(xi+1), and increase the number of calculations. To increase the order of truncation errors, their error estimates can be estimated using the Taylor expansion at xi at f(x, y). Therefore, we can consider constructing the Rayleigh formula with the linear combination of the function values of the function f(x, y) at several points. The construction requires the Taylor expansion and the solution y(x) of the approximate formula at f(xi, yi). The first few terms of the Taylor expansion at xi coincide, so that the Rayleigh formula reaches the required order. It avoids both the high-order derivative and the order of the accuracy of the calculation method.

The following method is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations. It is the simplest Runge–Kutta method as well. In my attached programming file, the simplest Runge-Kutta method is mostly like an approximate value. As an example, I used a sample differential equation dy / dx = -2*y + x + 4" when h = 0.2, y (0) = 1, calculate y when x = 0.2



```python
def rungeKutta1(x0,y0,x,h):
    n = (int)((x-x0)/h)
    y = y0
    for i in range(1, n+1):
        y = y+h*f(x0,y)
        x0 = x0+h
    return y
```



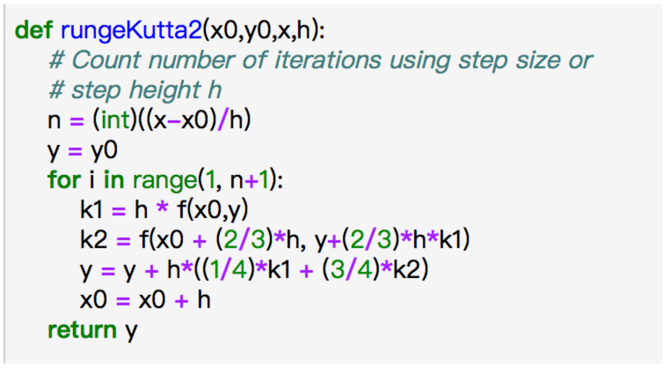
```
The exact value of when x= 0.2 is:  1.3472599654732704
---------------------------------------------
The value of y at x in RungeKutta first order is: 1.4
Absolute Error 0.052740034526729485
Relative Error 0.03914614541982829
```

This method has a larger absolute error and relative error when compare with the exact value. Consider calculating the shape of such an unknown curve: it has a given starting point and satisfies a given differential equation. Here, the "differential equation" can be regarded as a formula that can calculate the tangent slope of this point by the position of an arbitrary point on the curve. The idea is that you only know the starting point of the curve at first P0, the rest of the curve is unknown, but by differential equations we can get the tangent line of that point, then take a small step forward along the tangent to get P1, same steps to get P2, P3 and so on. In general, this line is not far from the original unknown curve, and any small error can be obtained by reducing the step size. The smaller step size, the more accurate result will show.

The following method is the second order Runge Kutta. We used two steps to implement the equation. It is much better than that of the first order algorithm.

$$\begin{aligned}
k_1 &= f(t_n,\ y_n),\\
k_2 &= f(t_n + \tfrac{2}{3}h,\ y_n + \tfrac{2}{3}hk_1),\\
y_{n+1} &= y_n + h\left(\tfrac{1}{4}k_1 + \tfrac{3}{4}k_2\right).
\end{aligned}$$

I decided to choose this formula to implement the solution because compare its solution to other methods, such as endpoint method, it seems more accurate in my example question. The output is 1.324



```
def rungeKutta2(x0,y0,x,h):
    # Count number of iterations using step size or
    # step height h
    n = (int)((x-x0)/h)
    y = y0
    for i in range(1, n+1):
        k1 = h * f(x0,y)
        k2 = f(x0 + (2/3)*h, y+(2/3)*h*k1)
        y = y + h*((1/4)*k1 + (3/4)*k2)
        x0 = x0 + h
    return y
```

```
----------------------------------------------
The value of y at x in RungeKutta second order is: 1.324
Absolute Error -0.02325996547327036
Relative Error -0.017264645331533705
```

We can see that the second order method is more accurate than the first order method. The absolute error and relative error are also smaller than the previous one. The reason we can get more accurate number is because we add first order method and one more term by doing Taylor series expansion. If we add more term, it will be more accurate.

Then, we have the third order Runge Kutta method which is more accurate to prove my statement. The Taylor series expansion is $y(x+h) = y(x) + \frac{k1}{6} + \frac{2*k2}{3} + \frac{k3}{6} + O(h^4)$

This method is a third order Runge-Kutta method for approximating the solution of the initial value problem $y'(x) = f(x,y)$; $y(x_0) = y_0$ which evaluates the integrand, $f(x,y)$, three times per step. For step $n+1$,

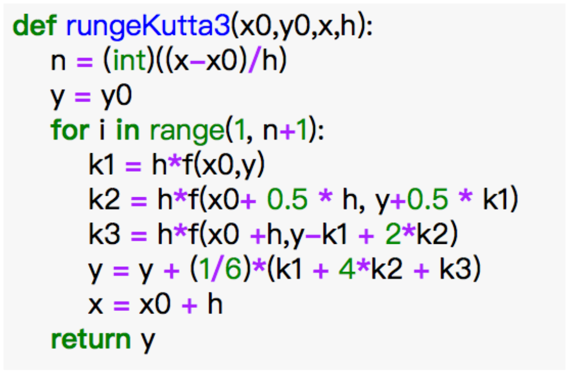
$$y_{n+1} = y_n + \frac{1}{6}(k1 + 4 * k2 + k3)$$

$$k1 = h * f(x_n, y_n)$$

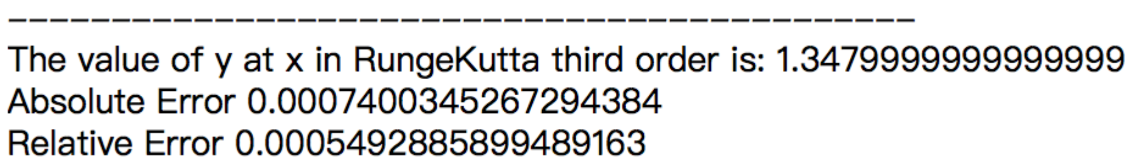
$$k2 = h * f\left(x_n + \frac{h}{2}, y_n + \frac{k1}{2}\right)$$

$$k3 = h * f(x_n + h, y_n - k1 + 2 * k2)$$

This method is a third order procedure for which Richardson extrapolation can be used. The following program will implement the method above.



```
def rungeKutta3(x0,y0,x,h):
    n = (int)((x-x0)/h)
    y = y0
    for i in range(1, n+1):
        k1 = h*f(x0,y)
        k2 = h*f(x0+ 0.5 * h, y+0.5 * k1)
        k3 = h*f(x0 +h,y-k1 + 2*k2)
        y = y + (1/6)*(k1 + 4*k2 + k3)
        x = x0 + h
    return y
```



```
-------------------------------------------------
The value of y at x in RungeKutta third order is: 1.3479999999999999
Absolute Error 0.0007400345267294384
Relative Error 0.0005492885899489163
```

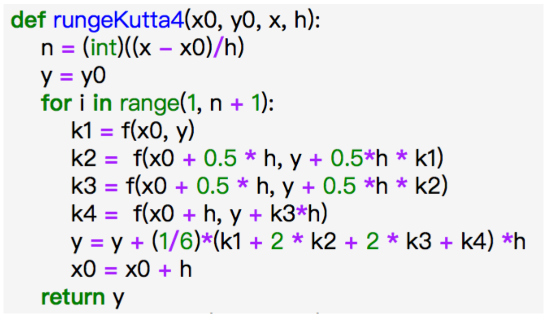
We can see that the output is almost more approaching the exact value. Both absolute and relative error are smaller than second order method. Third order improved Runge-Kutta methods, have been developed for numerical integration of first order ordinary differential equations. They are almost two-step in nature and they are computationally more efficient and produced smaller errors compared to the existing Runge-Kutta methods of the first two orders.

The Fourth Order Runge-Kutta method is the better method than above methods. I will talk more details about this method. The development of the Fourth Order Runge-Kutta method closely follows those for the Second Order. As with the second order technique there are many variations of the fourth order method, and they all use four approximations to the slope. The slope weighted average at four points on [xi,xi+1] is used as an approximation of the average slope k* to form a series of Fourth Order Runge-Kutta formulas. It has Fourth Order precision, that is, the local truncation error is O(h5). The most commonly used Fourth Order Runge Kutta method is described below.

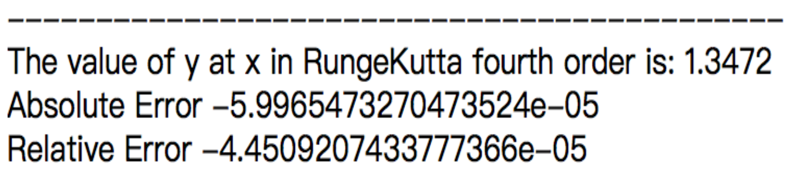
$$k1 = f(x_n, y_n)$$
$$k2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} * k1\right)$$
$$k3 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} * k2\right)$$
$$k4 = f(x_n + h, y_n + h * k3)$$
$$y_{n+1} = y_n + \frac{h}{6} * (k1 + 2 * k2 + 2 * k3 + k4)$$

From our lecture note and online resource, we can see the formulas are straight forward. k1 is the slope at the beginning of the time step (the same k1 as before). If we use the slope k1 to step in the middle of the time step, then k2 is an estimate of the slope at the midpoint. For the new approximation of y(t), the slope is proved to be more accurate than k1. If we use the slope k2 to step in the middle of the time step, then k3 is another estimate of the slope at the midpoint. Finally, we use the slope k3 to step through the entire time step, and k4 is an estimate of the endpoint slope. The Fourth Order Runge-Kutta method is more complicated than others.

Let us see how the output looks like when compare it with exact value in the program.



```
def rungeKutta4(x0, y0, x, h):
    n = (int)((x - x0)/h)
    y = y0
    for i in range(1, n + 1):
        k1 = f(x0, y)
        k2 =  f(x0 + 0.5 * h, y + 0.5*h * k1)
        k3 = f(x0 + 0.5 * h, y + 0.5 *h * k2)
        k4 =  f(x0 + h, y + k3*h)
        y = y + (1/6)*(k1 + 2 * k2 + 2 * k3 + k4) *h
        x0 = x0 + h
    return y
```



```
------------------------------------------
The value of y at x in RungeKutta fourth order is: 1.3472
Absolute Error -5.996547327047352e-05
Relative Error -4.4509207433777366e-05
```

The result shows that the output by using Fourth Order Runge Kutta method is pretty closed to the exact value. Even though there are still some absolute error, it is better than all the methods we have discussed above.

In conclusion, the derivation of the Runge Kutta method is based on the Taylor expansion method, so it requires that the solution obtained has better smoothness. If the smoothness of the solution is poor, then the numerical solution obtained using the fourth-order Runge Kutta method may not be as accurate as the improved Euler method. In the actual calculation, the appropriate algorithm should be selected for the specific characteristics of the problem. For problems that are not very smooth, it is best to use a low-order algorithm to take the step h small.

Reference

Runge–Kutta methods. (2018, Dec 09).Retrieved from https://en.wikipedia.org/wiki/Runge–Kutta_methods

Euler method. (2018, November 30). Retrieved from https://en.wikipedia.org/wiki/Euler_method

Cheever, E., & Swarthmore College. (n.d.). Retrieved from http://lpsa.swarthmore.edu/NumInt/NumIntSecond.html

Cheever, E., & Swarthmore College. (n.d.). Retrieved from http://lpsa.swarthmore.edu/NumInt/NumIntFourth.html

Teknomo, K. (n.d.). Third Order Runge-Kutta Method (RK3) to Solve ODE. Retrieved from https://people.revoledu.com/kardi/tutorial/ODE/Runge Kutta 3.htm

Professor Erik K. Grimmelmann, Ph.D. Lecture note. Initial Value Problems for Ordinary Differential Equations