Runge-Kutta Method With Adaptive Stepsize

Daksh Kadian
(1911060)

Third Year Integrated M.Sc.
School of Physical Sciences
NISER, Bhubaneshwar

1 Introduction

We have learnt about Runge-Kutta method of solving Ordinary Differential Equations (ODEs) already. It is a method where we know the intital condiditions of the equation and then use a small 'step' value to calculate the next value of the function. Now we can calculate the values on and on in a similar way. The most common method of the Runge-Kutta class of methods is the RK4 method. For this method we require the intital conditons (provided to us already) and a stepsize which will tell us how close the values of the function are plotted with respect to the independent variable. If we require an accurate description of the function then the stepsize needs to be small. For a rough estimate, the error in the function at any point is given as:

$$error \sim (stepsize)^4$$

As the stepsize decreases, the number of steps required to estimate the function in its entire domain will increase too. These extra steps will consume precious resources of the machine and it will take a longer time. If we have decided to use RK4 method already then the obvious and intuitive solution is to have an adaptive stepsize which changes its value based on the maximum permissible error (decided by us) such that the performance impact on the machine is minimum. This can be achieved by fixing an initial stepsize and then changing it based on the requirements of the region where we are working and this is done by reducing the stepsize when the function is in a region of high volatility (i.e. it is changing relatively quickly in a short interval) so that high accuracy is maintained and by increasing the stepsize when the function is in a region of low volatility (i.e. the value of the function does not change significantly in a large interval).

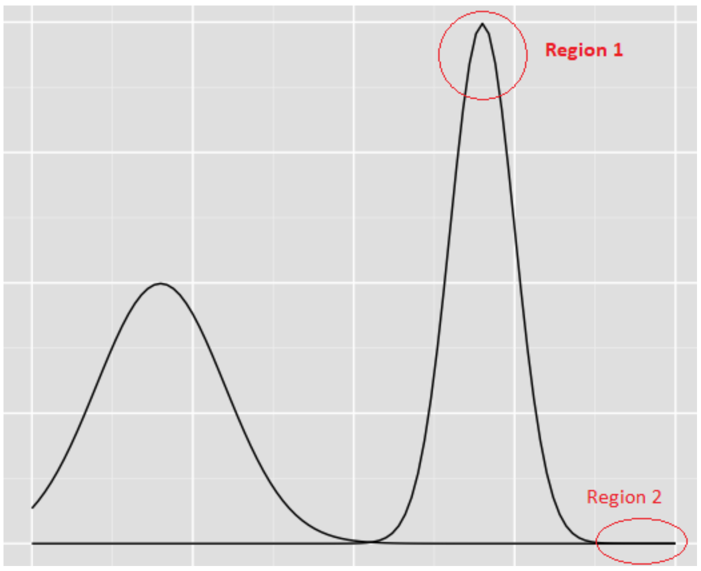

Figure 1: Example Function

In the function plotted in Figure 1, the stepsize should be decccreased in Region 1 and should be increased in Region 2. This will make our code more efficient than having a constant stpepize and will even increase the accuracy in certain cases where the nature of the function could not be predicted.

2 Working

The simplest method for adaptive stepsize is step doubling. In this method, every step is is taken twice, once as a full step and then as two half steps. It might seem like we are actually doing more work in a single calculation but we are actually calculating the next value of the function with half step accuracy and this is the ereason why it is more efficient than having an initial stepsize which is equal to our current half stepsize. To show this quantitaively, take one full step to be $2h$ and a half step will then be h. The true solution $y(x + 2h)$ and the twon approximations $y_1 andy_2$ are related as:

$$y(x + 2h) = y_1 + (2h)^5\phi + O(h^6) + ... \tag{1}$$

$$y(x + 2h) = y_2 + 2(h)^5\phi + O(h^6) + ... \tag{2}$$

Where ϕ is a part of the Taylor Expansion and is of the order $y^5/5!$. The expression in (1) involves $(2h)^5$ since the stepsize is 2h, while the expression in (2) involves $2(h^5)$ since the error on each step is $h^5\phi$. The difference between the two numerical estimates is a convenient indicator of truncation error

$$\Delta \equiv y_2 - y_1 \tag{3}$$

On solving (1), (2) and (3) and ignoring terms of orders highers than h^6, we get

$$y(x+2h) = y_2 + \frac{\Delta}{15} + O(h^6) \tag{4}$$

As is obvious, (4) is of oreder 5 and it is one order higher than normal RK4. This extra accuracy is helpful if we keep our Δ vaue in check as an accuracy parameter.

3 Implementation

For my implementation of adaptive stepsize RK4 solver, I used a step doubling and halving adaptive method. It will determine if it is better to increase or decrease the stepsize from the initial stepsize. This will make our program even more efficient and accurate. In essence, we will do the Runge-Kutta calculations 3 times for each step. First we will calculate using the current stepsize and then we willcalculate using half stepsize. If the difference in the value of the funation is large enough, we will make current stepsize half. Then we compare the original stepsize to double stepsize. If the difference between original and double stepsize is small enough, we double the origianl stepsize. A fixed stepsize is also enforced to make sure that division by zero is not a problem is the stepsize becomes too small. This algorithm can also be implemented within these steps to make the overall code even more accurate and efficient.
To show this implementation, we solve the following ODE using doubling adaptive RK4 method

$$\frac{dx}{dt} = e^x - xe^t \tag{5}$$

4 Running the Code

Both fixed stepsize and adaptive stepsize RK4 was run for this ODE and a screenshot of the CPU usage was taken from the task manager. Though it is not the most accurate description (as this task is not very resource intensive), a vague idea can be gathered from the stats. The peak for fixed stepsize run shows a higher CPU usage than for the adaptive stepsize run.

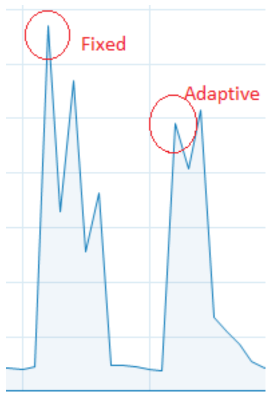

Figure 2: CPU Usage

When both the plots are plotted on the same graph, it becomes evident that this low resource usage does not come at the cost of accuracy of the solution.

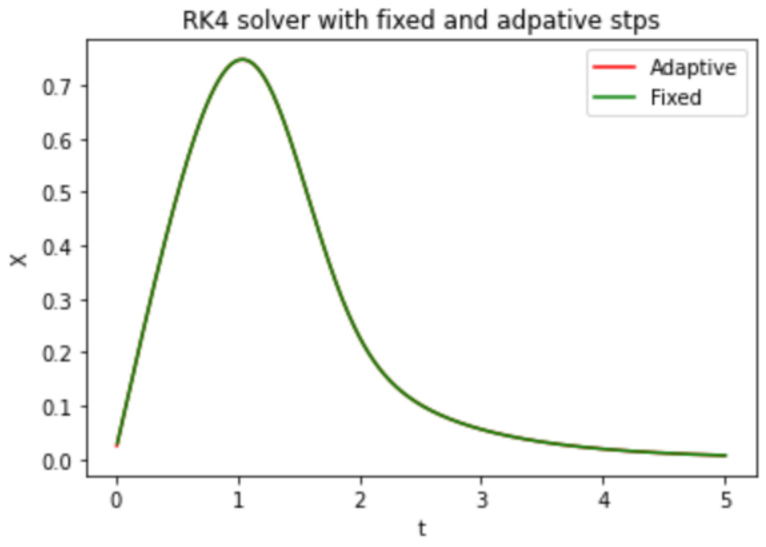

Figure 3: Both Plots