Adaptive step-size Runge-Kutta Methods

Maitrey Sharma

November 22, 2021

1 Introduction

While working with algorithms to solve ODEs, the choice of the step-size h is important as if it is too large, the truncation error may be unacceptable and if it is too small, we are expending unnecessary computer memory. Moreover, a constant step-size may not be appropriate for the entire range of integration. For example, if the solution curve starts off with rapid changes before becoming smooth we should use a small h at the beginning and increase it as we reach the smooth region. Therefore we need to develop adaptive methods which estimate the truncation error at each integration step and automatically adjust the step size to keep the error within prescribed limits.

2 Formulation

The adaptive Runge-Kutta methods use so-called embedded integration formulas. These formulas come in pairs: One formula has the integration order m, and the other one is of order $m + 1$. The idea is to use both formulas to advance the solution from x to $x + h$. Denoting the results by $y_m(x + h)$ and $y_{m+1}(x + h)$, an estimate of the truncation error in the formula of order m is obtained from

$$E(h) = y_{m+1}(x + h) - y_m(x + h) \tag{1}$$

Here are Runge-Kutta formulae of order five:

$$\begin{aligned} K_0 &= hF(x, y) \\ K_i &= hF\left(x + A_i h, y + \sum_{j=0}^{i-1} B_{ij} K_j\right), i = 1, 2, \ldots, 6 \end{aligned} \tag{2}$$

$$y_5(x + h) = y(x) + \sum_{i=0}^{6} C_i K_i \tag{3}$$

The embedded fourth-order formula is

$$y_4(x+h) = y(x) + \sum_{i=0}^{6} D_i K_i \tag{4}$$

The coefficients appearing in these formulas proposed by Dormand and Prince are given in table (1). Using (1), we have

$$E(h) = y_5(x+h) - y_4(x+h) = \sum_{i=0}^{6} (C_i - D_i) K_i \tag{5}$$

To control the error measure $e(h)$ in the $E(h)$, out of the many approaches is to equate to the root-mean-square error in $E(h)$, so

$$e(h) = \bar{E}(h) = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} E_i^2(h)} \tag{6}$$

Error control is achieved by adjusting the increment h so that the per-step error $e(h)$ is approximately equal to a prescribed tolerance ε. Noting that the truncation error in the fourth-order formula is $\mathcal{O}(h^5)$, we conclude that

$$\frac{e(h_1)}{e(h_2)} \approx \left(\frac{h_1}{h_2}\right) \tag{7}$$

Let us suppose that we performed an integration step with h_1 that resulted in the error $e(h_1)$. The step size h_2 that we should have used can now be obtained from (7) by setting $e(h_2) = \varepsilon$:

$$h_2 = h_1 \left[\frac{\varepsilon}{e(h_1)}\right]^{1/5} \tag{8}$$

If $h_2 \geq h_1$ we could repeat the integration step with h_2, but as the error was below the tolerance, we can accept current step and try h_2 in the next step. However, if $h_2 > h_1$, we must scrap the current step and repeat it with h_2.

The equation (8) can be modified slightly just to be safe and not overestimate the step-sizes as

$$h_2 = 0.9 h_1 \left[\frac{\varepsilon}{e(h_1)}\right]^{1/5} \tag{9}$$

Also to prohibit large fluctuations in values of h, following constrains are applied

$$0.1 \leq \frac{h_2}{h_1} \leq 10 \tag{10}$$

$e(h)$ applies to a single integration step, that is, it is a measure of the local truncation error. The global truncation error is caused by the accumulation of the local errors. Because $e(h)$

is a conservative estimate of the actual error, setting $\varepsilon = \varepsilon_{global}$ is usually adequate. If the number of integration steps is very large, it is advisable to decrease ε accordingly.

Now finally writing the algorithm, consider h as the trial value of the increment for the first integration step and K_0 being computed from scratch only there. We use

$$(K_0)_{m+1} = \frac{h_{m+1}}{h_m}(K_6)_m \tag{11}$$

If mth step was accepted, and

$$(K_0)_{m+1} = \frac{h_{m+1}}{h_m}(K_0)_m \tag{12}$$

To prove (11), we let $i = 6$ in (2), obtaining

$$(K_6)_m = h_m F\left[x_m + A_6 h_m, y_m + \sum_{i=0}^{5} B_{6i}(K_i)_m\right] \tag{13}$$

From table (1) we have last row of B-coefficients is identical to the C-coefficients (i.e., $B_{6i} - C_i$). Also as $A_6 = 1$. Therefore,

$$(K_6)_m = h_m F\left[x_m + h_m, y_m + \sum_{i=0}^{5} C_i(K_i)_m\right] \tag{14}$$

But according to (3), the fifth-order formula is

$$y_{m+1} = y_m + m_{i=0}^{6} C_i(K_i)_m \tag{15}$$

Since $C_6 = 0$ from table (1), we can reduce the upper limit of the summation from 6 to 5. Therefore, (14) becomes

$$(K_6)_m = h_m F(x_{m+1}, y_{m+1}) = \frac{h_m}{h_{m+1}}(K_0)_{m+1} \tag{16}$$

which completes the proof. The validity of (12) is rather obvious by inspection of the first equation of (2). Because step $m + 1$ repeats step m with a different value of h, we have

$$(K_0)_m = h_m F(x_m, y_m) \qquad (K_0)_{m+1} = h_{m+1} F(x_m, y_m) \tag{17}$$

which leads directly to (12).

This algorithm is coded in accompanying Python file.

3 Problem Statement

Now that we have formulated and coded the algorithm we can consider an application in a real-world physical problem. Consider the aerodynamic drag force acting on a certain object in free fall

$$F_D = av^2 e^{-by} \tag{18}$$

Table 1: Dormand-Prince coefficients.

i	A_i	B_{ij}						C_i	D_i
0	–	–	–	–	–	–	–	$\frac{35}{384}$	$\frac{5179}{57600}$
1	$\frac{1}{5}$	$\frac{1}{5}$	–	–	–	–	–	0	0
2	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	–	–	–	–	$\frac{500}{1113}$	$\frac{7571}{16695}$
3	$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	–	–	–	$\frac{125}{192}$	$\frac{393}{640}$
4	$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	–	–	$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
5	1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	–	$\frac{11}{84}$	$\frac{187}{2100}$
6	1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$

where v = velocity of object in m/s; y = elevation of the object in metres; $a = 7.45\,\mathrm{kg\,m^{-1}}$; $b = 10.53 \times 10^{-5}\,\mathrm{m^{-1}}$. The exponential term accounts for the change of air density with elevation. The differential equation describing the fall is

$$m\ddot{y} = -mg + F_D \tag{19}$$

where $g = 9.806\,65\,\mathrm{m\,s^{-2}}$ and $m = 114\,\mathrm{kg}$ is the mass of the object. Taking the initial condition that the object was released from elevation of 9 km, let us use our algorithm to find elevation and speed after a 10 second fall.

The differential equation and the initial conditions are

$$\begin{aligned} \ddot{y} &= -g + \frac{a}{m}\dot{y}^2 exp(-by) \\ &= -9.80665 + \frac{7.45}{114}\dot{y}^2 exp(-10.53 \times 10^{-5} y) \end{aligned} \tag{20}$$

with $y(0) = 9000\,\mathrm{m}$ and $\dot{y}(0) = 0$ Let us define $\dot{y}$ as a vector:

$$\dot{y} = \begin{bmatrix} \dot{y}_0 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ -9.80665 + (65.351 \times 10^{-3}) y_1^2 exp(-10.53 \times 10^{-5} y_0) \end{bmatrix} \tag{21}$$

and

$$y(0) = \begin{bmatrix} 9000m \\ 0 \end{bmatrix} \tag{22}$$

The driver program for this problem is written in the accompanying Jupyter notebook. We specified a per-step error tolerance of 10^{-2} in `integrate`. Considering the magnitude of y, this should be enough for five decimal point accuracy in the solution.

The first step was carried out with the prescribed trial value $h = 0.5\,\mathrm{s}$. Apparently the error was well within the tolerance, so that the step was accepted. Subsequent step sizes, determined from (9), were considerably larger. Inspecting the output, we see that at $t = 10\,\mathrm{s}$ the object is moving with the speed $v = -\dot{y} = 19.52\,\mathrm{s}$ at an elevation of $y = 8831\,\mathrm{m}$.