

Adaptive Time-stepping with Multistep and Multistage Methods

Varun Shankar

January 27, 2016

1 Introduction

Often, it is beneficial to choose a certain value of Δt so as to meet an error tolerance; further, it is useful to have the ability to change Δt on the fly. In this document, we will discuss adaptive time-stepping with both LMMs and RK methods.

2 Adaptive Linear Multistep Methods

We will demonstrate adaptive time-stepping using an AB-AM predictor-corrector pair. This is the most common approach for adaptive time-stepping in an LMM. Let us use AB2 and AM2.

Let $y(t_{n+1})$ be the true solution to the IVP at time t_{n+1} , and $\tilde{y}_{AB}^{n+1}$ be the numerical solution generated by the AB2 method, and y_{AM}^{n+1} be the numerical solution generated by the AM method. We can write out the LTEs for each of these methods:

$$y(t_{n+1}) - \tilde{y}_{AB}^{n+1} = \frac{5}{12}(\Delta t)^3 y'''(\xi_{AB}), \quad (1)$$

$$y(t_{n+1}) - y_{AM}^{n+1} = -\frac{1}{12}(\Delta t)^3 y'''(\xi_{AM}). \quad (2)$$

If we assume that y''' is constant over the interval, then it will have the same value regardless of whether it is evaluated at ξ_{AB} or at ξ_{AM} . We can thus safely subtract the above two equations from each other, giving us

$$y_{AM}^{n+1} - \tilde{y}_{AB}^{n+1} = \frac{1}{2}(\Delta t)^3 C, \quad (3)$$

where $C = y'''(xi)$ is the constant value taken by y''' over the interval. The approximate error at the time-step is therefore

$$|y(t_{n+1}) - y_{AM}^{n+1}| = \frac{1}{12}(\Delta t)^3 y'''(xi), \quad (4)$$

$$\implies |y(t_{n+1}) - y_{AM}^{n+1}| = \frac{1}{12}(\Delta t)^3 \frac{2}{\Delta t}, \quad (5)$$

$$\implies |y(t_{n+1}) - y_{AM}^{n+1}| = \frac{1}{12}(\Delta t)^3 \frac{2|y_{AM}^{n+1} - \tilde{y}_{AB}^{n+1}|}{(\Delta t)^3}, \quad (6)$$

$$\implies |y(t_{n+1}) - y_{AM}^{n+1}| = \frac{1}{6}|y_{AM}^{n+1} - \tilde{y}_{AB}^{n+1}|. \quad (7)$$

Then, adaptive time-stepping would proceed as follows:

1. Predict with AB.
2. Correct with AM.
3. Compute the error term corresponding to the AB-AM pair.
4. If the error term is “large”, then reduce Δt , interpolate f and y to the appropriate time-levels.
5. If the error term is “small”, increase Δt , interpolate f and y to the appropriate time levels.

3 Adaptive Runge-Kutta Methods

While one could use the above approach for an RK method, it is unnecessary. Recall that explicit RK methods of higher order are *more* stable than the lower-order ones. Thus, instead of using an explicit method and an implicit method to estimate the error, we can use an explicit method of a “lower” order, and another explicit method of a higher order to estimate the error at any given time level.

While there are other possible ways to control the step-size with RK methods, we will focus on a variant of this approach. RK methods of this type are called *Embedded Runge-Kutta Methods*. We will discuss this with an order 4-5 pair. The idea is as follows: use a fifth-order RK method with as few stages as possible (this turns out to be 6); using those 6 stages in some other combination, also obtain a fourth-order approximation to the solution. A difference between the fourth-order and fifth-order solution then gives an error estimate for the current time-step in terms of the coefficients and stages!

Let us formalize this a bit. Let the general form of the fifth-order RK method

be

$$y^{n+1} = y^n + \Delta t \sum_{i=1}^6 b_i k_i + O(\Delta t^6), \quad (8)$$

$$k_1 = f(t_n, y^n), \quad (9)$$

$$k_2 = f(t_n + c_2 \Delta t, y^n + \Delta t a_{21} k_1), \quad (10)$$

$$\dots \quad (11)$$

$$k_6 = f(t_n + c_6 \Delta t, y^n + \Delta t (a_{61} k_1 + \dots + a_{65} k_5)). \quad (12)$$

The *embedded* fourth-order formula is

$$\tilde{y}^{n+1} = y^n + \Delta t \sum_{i=1}^6 b_i^* k_i + O(\Delta t^5). \quad (13)$$

Consequently, the error in the time-step can automatically be estimated as

$$y^{n+1} - \tilde{y}^{n+1} = \sum_{i=1}^6 (b_i - b_i^*) k_i. \quad (14)$$

In other words, we get the error estimate automatically! There is no need to work out the truncation error for the specific pair of RK methods like we did in the multistep methods.

Matlab's ODE45 time integrator uses an RK45 method called Dormand-Prince method, with seven stages. However, the 7th stage of a step is the 1st stage of the next step, thus leading to only 6 function evaluations. Unlike other embedded RK methods which continue the 4th order solution, the Dormand-Prince method minimizes and continue with the 5th order solution! Thus, we are performing a local extrapolation in each time-step.