The RKM ODE Solver

M. McNelis

Contents

1. Overview

The RKM algorithm is a Runge–Kutta scheme that solves ordinary differential equations (ODE) or time-dependent partial differential equations (PDE) with an adaptive time step. I originally derived it in one of my papers to solve fluid dynamic equations. The algorithm presented here is mostly the same, though it has some minor changes. Here I wanted to increase its applicability and implement it in a Python library to make it more accessible.

1.1. How the RKM scheme works

We are mainly interested in numerically solving the ODE

$$\frac{dy}{dt} = f(t, y)\,, \tag{1}$$

where t is the time, $y(t)$ is the solution and $f(t, y)$ is the source function. We will study one-dimensional solutions for simplicity, but the algorithm can be easily generalized for a multi-dimensional vector $Y(t)$ (and also time-dependent PDEs). However, we use an explicit Runge–Kutta method, so we are limited to non-stiff systems.

Suppose at step n (or $t = t_n$) we have a numerical solution y_n, which we had computed using a standard Runge–Kutta (RK) scheme of order p (i.e. the global error $E \sim (\Delta t)^p$)

$$y_n = y_{n-1} + \sum_{i=1}^{s} b_i\, \Delta y_{n-1}^{(i)}\,, \tag{2}$$

where y_{n-1} is the previous solution. The sum runs over s stages, where the intermediate Euler steps are

$$\Delta y_{n-1}^{(1)} = \Delta t_{n-1} f(t_{n-1}, y_{n-1})\,, \tag{3a}$$

$$\Delta y_{n-1}^{(2)} = \Delta t_{n-1} f(t_{n-1} + c_2 \Delta t_{n-1}, y_{n-1} + a_{21} \Delta y_{n-1}^{(1)})\,, \tag{3b}$$

$$\ldots \tag{3c}$$

$$\Delta y_{n-1}^{(s)} = \Delta t_{n-1} f(t_{n-1} + c_s \Delta t_{n-1}, y_{n-1} + a_{sj} \Delta y_{n-1}^{(j)})\,, \tag{3d}$$

and Δt_{n-1} is the step size that we used to evaluate y_n and the index $j \in (1, s-1)$. The coefficients (b_i, c_i, a_{ij}) make up the Butcher tableau of your choice.

Now we want to update the solution y_{n+1} with a new step size Δt_n. First, we compute the adaptive step size by estimating the local truncation error of the intermediate Euler step

$$\Delta y_n^{(1)} = \Delta t_n f(t_n, y_n)\,, \tag{4}$$

which is

$$E_n = \frac{1}{2} C (\Delta t_n)^2\,. \tag{5}$$

(review) The mean-value theorem tells us that somewhere on the interval $t \in [t_n, t_n + \Delta t_n]$, the coefficient C is the second time derivative $y''(t)$; it is also approximately constant. To obtain an expression for C, we compute the intermediate Euler step (4) using the old step size (denoted by $\star$):

$$\Delta y_n^{(1\star)} = \Delta t_{n-1} f(t_n, y_n)\,. \tag{6}$$

Then we approximate the corresponding coefficient $C^\star$ using central differences:[1]

$$C^\star = \frac{2\big(y_{n+1}^{(1\star)} - 2y_n + y_{n-1}\big)}{(\Delta t_{n-1})^2} + O(\Delta t_{n-1})\,, \tag{8}$$

where $y_{n+1}^{(1\star)} = y_n + \Delta y_n^{(1\star)}$. Equation (8) is only first-order accurate due to the truncation error present in $y_{n+1}^{(1\star)}$. The numerical solutions y_n and y_{n-1} are assumed to have smaller errors if they are computed with a RK scheme of order $p \geq 3$.

It is safe to assume that $C \approx C^\star$ on the interval $t \in [t_n, t_n + \Delta t_n]$ as long as Δt_n is not that much larger than Δt_{n-1}. Now we can finally solve for Δt_n by setting Eq. (5) to the desired error tolerance:

$$\frac{1}{2}|C|(\Delta t_n)^2 = \max(\epsilon_0 |y_n|,\, \epsilon_0 \Delta t_n |f(t_n, y_n)|)\,, \tag{9}$$

where ϵ_0 is the tolerance parameter. Here we take either the relative tolerance $\epsilon_0 |y_n|$ or the incremental tolerance $\epsilon_0 \Delta t_n |f(t_n, y_n)|$, whichever one is larger.[2]

---

[1] One can also estimate $C^\star$ using the traditional step-doubling technique, where we subtract one Euler step with Δt_{n-1} from two consecutive Euler steps with $\frac{1}{2}\Delta t_{n-1}$:

$$C^\star = \frac{2\big(f(t_n + \frac{1}{2}\Delta t_{n-1}, y_n + \frac{1}{2}\Delta t_{n-1} f(t_n, y_n)) - f(t_n, y_n)\big)}{\Delta t_{n-1}} + O(\Delta t_{n-1})\,. \tag{7}$$

However, it has the same numerical accuracy as Eq. (8) because of the truncation error in the second Euler half-step (and it costs an extra function evaluation).

[2] For multi-dimensional vectors, we replace the absolute value $|...|$ by the ℓ_p–norm $||...||_p$.

Therefore, our formula for the adaptive step size is

$$\Delta t_n = \begin{cases} \sqrt{\dfrac{2\epsilon_0|y_n|}{|C|}}\,, & \text{if } |y_n| \geq \dfrac{2\epsilon_0|f_n|^2}{|C|} \\ \\ \dfrac{2\epsilon_0|f_n|}{|C|}\,, & \text{if } |y_n| < \dfrac{2\epsilon_0|f_n|^2}{|C|}\,. \end{cases} \tag{10}$$

This step size can then be used for the next RK iteration:

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i \Delta y_n^{(i)}\,. \tag{11}$$

Notice that we can recycle the source function in Eq. (6) to evaluate the first intermediate Euler step $\Delta y_n^{(1)}$ in the usual RK scheme.

The RKM scheme is highly versatile since we can turn any standard RK method that uses a fixed step size into an adaptive one. In particular, we can easily integrate our algorithm with a high-order scheme to solve ODEs. One may also combine it with low-order schemes that are commonly used in time-dependent PDEs to evolve the system in three spatial dimensions (e.g. the heat equation).[3]

The algorithm is also able to advance in time while avoiding step rejections. Other adaptive schemes evaluate two different RK methods to estimate the local truncation error. If the truncation error is above the tolerance, the step is rejected and needs to be repeated with a smaller step size. In the RKM scheme, we can adjust the step size before completing the full step. This property is useful in situations where the adaptive step size rapidly decreases or oscillates.

1.2. Step size controls

In adaptive schemes, we usually place control measures to prevent the step size from changing too rapidly after each step:

$$\Delta t_n \leftarrow \max\left(\alpha_{\text{low}}\Delta t_{n-1}, \min(\Delta t_n, \alpha_{\text{high}}^{1/(1+p)}\Delta t_{n-1})\right)\,, \tag{12}$$

[3]The numerical accuracy of time-dependent PDEs is often limited by the finite spatial resolution of the lattice. For this reason, low to mid-order Runge–Kutta schemes are generally preferred over high-order ones.

where we set the control factors to $\alpha_{\text{low}} = 0.2$ and $\alpha_{\text{high}} = 1.4$, respectively. In addition, we impose lower and upper limits on Δt_n:

$$\Delta t_n \leftarrow \max\left(\Delta t_{\min}, \min(\Delta t_n, \Delta t_{\max})\right) , \tag{13}$$

where $\Delta t_{\min} = 10^{-7}$ and $\Delta t_{\max} = 1$.

We note that the parameter α_{high} is suppressed for higher-order RK schemes. This is because Eq. (10) is only optimized for the first intermediate Euler step; the time dependence of Δt_n is not sensitive to the order if $p \geq 3$. Increasing the step size to $\alpha_{\text{high}} \Delta\tau_{n-1}$ may be acceptable for low-order methods, but it would be more difficult to control the local truncation error of high-order methods. Thus, we want to take a more conservative approach and increase the step size more gradually if p is large. This is also why we usually use a small value for α_{high} (in embedded schemes, we use $\alpha_{\text{high}} = 5$).

2. Other adaptive schemes

2.1. Step doubling

The is the step doubling method (SDRK).

However, we need a total of $3s-1$ function evaluations to advance the SDRK step (if accepted on the first try), while the RKM scheme only uses s function evaluations per step.

2.2. Embedded schemes

A potential disadvantage our approach is that Eq. (10) is only optimized for the first intermediate Euler step. It turns out that the time dependence of Δt_n is not sensitive to the order of the RK scheme (if $p \geq 3$). Therefore, Eq. (10) may not be the optimal choice for controlling the local truncation error of higher-order methods (even though the global error $E \sim [\epsilon_0^p, \epsilon_0^{p/2}]$).