Numerical Analysis – Computational Session

Adaptive Step-size Runge–Kutta Methods

November 26, 2025

1 Introduction

Applying a Runge–Kutta method with a constant step size can be very inefficient. But how could we choose a different subdivision? A viable strategy is to select the step size h such that the local error is everywhere equal to a certain tolerance “Tol” specified by the user.

Let y_1 be the numerical solution of the Runge–Kutta method. We construct a second Runge–Kutta method with $\widehat{y}_1$ being the numerical approximation. We estimate the local error by using the difference $\widehat{y}_1 - y_1$.

2 Embedded method

Let an s-stage method of order p be given, with coefficients c_i, a_{ij}, and b_j. We look for a an approximation $\widehat{y}_1$ of order $\widehat{p} < p$ that uses the same function evaluations, i.e.,

$$\widehat{y}_1 = y_0 + h(\widehat{b}_1 k_1 + \ldots + \widehat{b}_s k_s),$$

where the k_i are given by a method from the family of Runge–Kutta methods. The k_1 use the same evaluations of f, but we change the weights b_i to $\widehat{b}_i$. To have more freedom, one often adds a term with $f(x_1, y_1)$, that we need to compute for the next time step in any case, and look for $\widehat{y}_1$ in the form

$$\widehat{y}_1 = y_0 + h\left(\widehat{b}_1 k_1 + \ldots + \widehat{b}_s k_s + \widehat{b}_s f(x_1, y_1)\right).$$

Example. For the Runge method, whose Butcher tableau is

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

we can take the explicit Euler method as the embedded method. Indeed, we can use the same function evaluation $k_1 = f(t_0, y_0)$ from the Euler method into the Runge method. Then

$$\text{err} = y_1 - \widehat{y}_1 = y_0 + h\,f\big(t_0 + \tfrac{h}{2}, y_0 + \tfrac{h}{2}k_1\big) - (y_0 + h\,f(t_0, y_0)) = h(k_2 - k_1).$$

This expression is then an approximation of the local error for the explicit Euler method.

For a general method, one must perform the Taylor expansion of k_i and $f(x_1, y_1)$ and compare the result with the exact solution. Since the c_i and the a_{ij} are already known, one obtains a linear system for the $\widehat{b}_i$.

3 Calculation of the optimal step size h

If one applies the method with a given value h for the step size, the error estimate satisfies ($\widehat{p} < p$)

$$y_1 - \widehat{y}_1 = (y_1 - y(t_0 + h)) + (y(t_0 + h) - \widehat{y}_1) = \mathcal{O}(h^{p+1}) + \mathcal{O}(h^{\widehat{p}+1}) \approx C \cdot h^{\widehat{p}+1}. \tag{1}$$

The optimal h, called h_{opt}, is the one for which this estimate $y_1 - \widehat{y}_1 \approx C \cdot h^{\widehat{p}+1}$ is close to Tol, i.e.,

$$\text{Tol} \approx C \cdot h_{\text{opt}}^{\widehat{p}+1}. \tag{2}$$

Eliminating C from (1) and (2), we obtain

$$h_{\text{opt}} = 0.9 \cdot h \cdot \sqrt[\widehat{p}+1]{\frac{\text{Tol}}{\|y_1 - \widehat{y}_1\|}}. \tag{3}$$

The factor 0.9 has been added to make the final algorithm "safer". In general, we also assume an additional restriction of the type $0.2h \leqslant h_{\text{opt}} \leqslant 5h$ to avoid big changes in h.

For the norm in (3), one typically uses

$$\|y_1 - \widehat{y}_1\| = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_{i1} - \widehat{y}_{i1}}{sc_i}\right)^2}, \quad \text{where} \quad sc_i = 1 + \max(|y_{i0}|, |y_{i1}|),$$

and y_{i0}, y_{i1}, and $\widehat{y}_{i1}$ is the ith component of y_0, y_1, and $\widehat{y}_1$, respectively. This represents a mix between relative error and absolute error.

4 Algorithm for the automatic selection of the step size

The algorithm for the automatic selection of the step size is formalized in Algorithm 1.

Algorithm 1: Algorithm for the automatic selection of the step size.

```
1  Input: the right-hand side $f(t, y)$, initial values $t_0$, $y_0$, $h$, and the tolerance Tol.
2  while $t_0 < t_{\text{end}}$ do
3      Using the step size $h$, compute $y_1$, $\widehat{y}_1$, err $= \|y_1 - \widehat{y}_1\|$, and $h_{\text{opt}}$ as in (3).
4      if err $\leqslant$ Tol then
5          $t_0 \leftarrow t_0 + h$
6          $y_0 \leftarrow y_1$
7          $h \leftarrow \min(h_{\text{opt}}, t_{\text{end}} - t_0)$
8      else
9          $h \leftarrow h_{\text{opt}}$
10     end if
11 end while
```

5 A concrete example: the Dormand–Prince method

The Dormand–Prince method (1980) is an adaptive step-size Runge–Kutta method which estimates the error by calculating two approximations at each step: one with 4th-order and one with 5th-order accuracy, and the local error is estimated by taking the difference between the 4th-order and the 5th-order solution. The step size is then adjusted to minimize the error, making the method efficient and reliable.

The Dormand–Prince method is important because it is the default method behind the `ode45` solver for MATLAB[1]. The Butcher tableau for the Dormand–Prince method is the following:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

The second-to-last row provides the 5th-order solution coefficients, and the last row presents the 4th-order coefficients for error estimation. Below, you have a friendlier text version that you can directly copy-paste into your MATLAB implementation:

```
A = [ 0, 0, 0, 0, 0, 0;
      1/5, 0, 0, 0, 0, 0;
      3/40, 9/40, 0, 0, 0, 0;
      44/45, -56/15, 32/9, 0, 0, 0;
      19372/6561, -25360/2187, 64448/6561, -212/729, 0, 0;
      9017/3168, -355/33, 46732/5247, 49/176, -5103/18656, 0;
      35/384, 0, 500/1113, 125/192, -2187/6784, 11/84 ];
c = [ 1/5, 3/10, 4/5, 8/9, 1, 1 ]';
w1 = [ 35/384, 0, 500/1113, 125/192, -2187/6784, 11/84 ];
w2 = [ 5179/57600, 0, 7571/16695, 393/640, -92097/339200, 187/2100, 1/40 ];
```

6 Your task

Implement the Dormand–Prince method with adaptive step-size control and apply it to numerically solve the problem (called "Brusselator", describing a chemical reaction)

$$
\begin{cases} y_1' = 1 + y_1^2 y_2 - 4y_1 \\ y_2' = 3y_1 - y_1^2 y_2 \end{cases} \quad \text{with initial conditions} \quad \begin{cases} y_1(0) = 1.5 \\ y_2(0) = 3 \end{cases} \tag{4}
$$

on the interval $[0, 20]$. Use a tolerance $\mathrm{Tol} = 10^{-6}$ (in MATLAB, write `Tol = 1e-6`).

[1]You may want to have a look at the MATLAB documentation here: https://www.mathworks.com/help/matlab/ref/ode45.html

<u>Hints</u>:

Your code should be composed of three scripts:

1. One MATLAB function (you may call it `RK45.m`) implementing the Dormand–Prince algorithm with the automatic step-size selection. In particular:
   - For the Dormand–Prince method, use the formulas of the Runge–Kutta family, specialized with the coefficients provided in the Butcher tableau above.
   - For the automatic selection of the step size, keep as a reference Algorithm 1 above.
2. Another MATLAB function that implements the "Brusselator" right-hand side from (4).
3. One script that will serve as the main driver of your code, with the parameters h, t_0, t_{end}, Tol, the right-hand side $f(t, y)$ defined as a MATLAB function handle, and the initial conditions provided in (4). In the main script, you should calculate the numerical solutions (i.e., the two trajectories y_1 and y_2) and plot them. Your plot should look similar to the one in Figure 1 below.

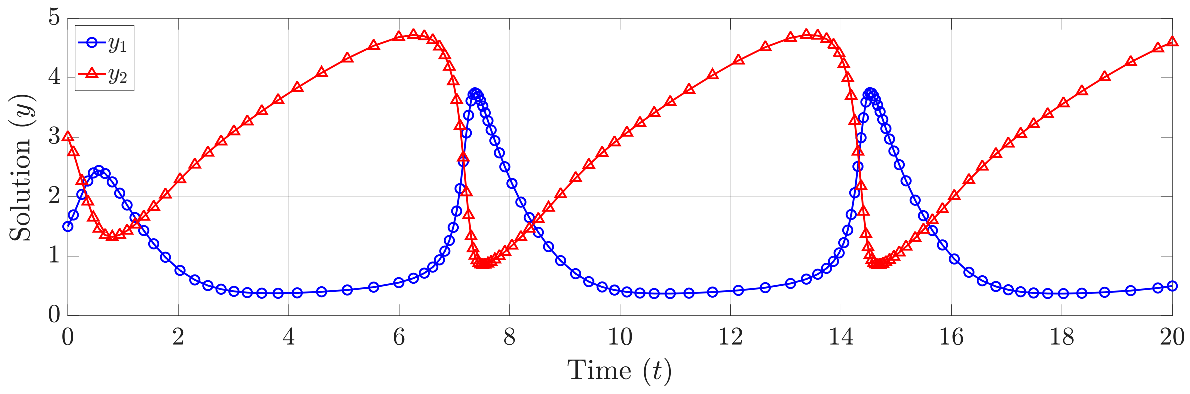

Figure 1: Numerical solution of the Brusselator ODE.