Error Control and the Runge-Kutta-Fehlberg Method

So far, we have assumed that the time-stepping methods that we have been using for solving \( y' = f(t, y) \) on the interval \( t_0 < t < T \) compute the solution at times \( t_1, t_2, \ldots \) that are equally spaced. That is, we define \( t_{n+1} - t_n = h \) for some value of \( h \) that is fixed over the entire interval \([t_0, T]\) on which the problem is being solved. However, in practice, this is ill-advised because

- the chosen time step may be too small to resolve the solution with sufficient accuracy, especially if it is highly oscillatory, or
- the chosen time step may be too large when the solution is particularly smooth, thus wasting computational effort required for evaluations of \( f \).

This is reminiscent of the problem of choosing appropriate subintervals when applying composite quadrature rules to approximate definite integrals. In that case, adaptive quadrature rules were designed to get around this problem. These methods used estimates of the error in order to determine whether certain subintervals should be divided.

The counterpart to this approach for initial values problems would involve estimating the global error, perhaps measured by \( \max_{0 \leq n \leq T/h} |y(t_n) - y_n| \), and then, if it is too large, repeating the time-stepping process with a smaller value of \( h \). However, this is impractical, because it is difficult to obtain a sharp estimate of global error, and much of the work involved would be wasted due to overwriting of solution values, unlike with adaptive quadrature, where each subinterval can be integrated independently.

Instead, we propose to estimate the local truncation error at each time step, and use that estimate to determine whether \( h \) should be varied for the next time step. This approach minimizes the amount of extra work that is required to implement this kind of adaptive time-stepping, and it relies on an error estimate that is easy to compute.

The general approach is as follows: suppose that we have computed \( y_n \), an approximation of \( y(t_n) \). We then compute \( y_{n+1} \), an approximation of \( y(t_{n+1}) \), using two time-stepping methods of different orders of accuracy. Specifically, one method is \( p \)th-order accurate, and the other is \((p+1)\)st-order accurate. That is, the local truncation error in these methods is \( O(h^p) \) and \( O(h^{p+1}) \), respectively.

Let \( y_{n+1} \) and \( \tilde{y}_{n+1} \) be the approximations of \( y(t_{n+1}) \) computed using the methods of order \( p \) and \((p+1)\), respectively. Then, if we assume that \( y_n = y(t_n) \) exactly, the local truncation error in
these two methods is given by
\[ \tau_{n+1}(h) = \frac{y(t_{n+1}) - y_{n+1}}{h}, \quad \tilde{\tau}_{n+1}(h) = \frac{y(t_{n+1}) - \tilde{y}_{n+1}}{h}, \]
since under this assumption, local truncation error reduces to the exact error committed during a single time step, rather than an estimate.

It follows from subtracting the two truncation errors that
\[ \tau_{n+1}(h) = \tilde{\tau}_{n+1}(h) + \frac{1}{h}(\tilde{y}_{n+1} - y_{n+1}). \]
However, \( \tau_{n+1}(h) = O(h^p) \), while \( \tilde{\tau}_{n+1}(h) = O(h^{p+1}) \). Therefore, we can neglect \( \tilde{\tau}_{n+1}(h) \) and obtain the simple error estimate
\[ \tau_{n+1}(h) = \frac{1}{h}(\tilde{y}_{n+1} - y_{n+1}). \]

Our goal is to determine how to modify \( h \) so that the local truncation error is approximately equal to a prescribed tolerance \( \varepsilon \), and therefore is not too large nor too small. Because \( \tau_{n+1}(h) \) is the local truncation error of a method that is \( p \)-th order accurate, it follows that if we replace \( h \) by \( qh \) for some scaling factor \( q \), the error is multiplied by \( q^p \). Therefore, we relate the error obtained with step size \( qh \) to our tolerance, and obtain
\[ |\tau_{n+1}(qh)| \approx \left| \frac{q^p}{h}(\tilde{y}_{n+1} - y_{n+1}) \right| \leq \varepsilon. \]
Solving for \( q \) yields
\[ q \leq \left( \frac{\varepsilon h}{|\tilde{y}_{n+1} - y_{n+1}|} \right)^{1/p}. \]
In practice, though, the step size is kept bounded by chosen values \( h_{\text{min}} \) and \( h_{\text{max}} \) in order to avoid missing sensitive regions of the solution by using excessively large time steps, as well as expending too much computational effort on regions where \( y(t) \) is oscillatory by using step sizes that are too small.

The Runge-Kutta-Fehlberg method is an example of an adaptive time-stepping method. It uses a fourth-order and fifth-order Runge-Kutta method that share some evaluations of \( f(t, y) \), in order to reduce the number of evaluations of \( f \) per time step to six, rather than the ten that would normally be required from a pairing of fourth- and fifth-order methods.