Euler’s Method, cont’d

We conclude our discussion of Euler’s method with an example of how the previous convergence analyses can be used to select a suitable time step $h$.

**Example** Consider the IVP

$$y' = -y, \quad 0 < t < 10, \quad y(0) = 1.$$ 

We know that the exact solution is $y(t) = e^{-t}$. Euler’s method applied to this problem yields the difference equation

$$y_{n+1} = y_n - hy_n = (1 - h)y_n, \quad y_0 = 1.$$ 

We wish to select $h$ so that the error at time $T = 10$ is less than 0.001. To that end, we use the error bound

$$|y(t_n) - y_n| \leq \frac{hM}{2L} [e^{L(t_n-t_0)} - 1],$$

with $M = 1$, since $y''(t) = e^{-t}$, which satisfies $0 < y''(t) < 1$ on $[0, 10]$, and $L = 1$, since $f(t, y) = -y$ satisfies $|\frac{\partial f}{\partial y}| = | -1| \equiv 1$. Substituting $t_n = 10$ and $t_0 = 0$ yields

$$|y(10) - y_n| \leq \frac{h}{2} [e^{10} - 1] \approx 1101.27h.$$ 

Ensuring that the error at this time is less than $10^{-3}$ requires choosing $h < 9.08 \times 10^{-8}$. However, the bound on the error at $t = 10$ is quite crude. Applying Euler’s method with this time step yields a solution whose error at $t = 10$ is $2 \times 10^{-11}$.

Now, suppose that we include roundoff error in our error analysis. The optimal time step is

$$h = \sqrt{\frac{2\delta}{M}},$$

where $\delta$ is a bound on the roundoff error during any time step. We use $\delta = 2u$, where $u$ is the unit roundoff, because each time step performs only two floating-point operations. Even if $1 - h$ is computed once, in advance, its error still propagates to the multiplication with $y_n$. In a typical
double-precision floating-point number system, \( u \approx 1.1 \times 10^{-16} \). It follows that the optimal time step is
\[
h = \sqrt{\frac{2\delta}{M}} = \sqrt{\frac{2(1.1 \times 10^{-16})}{1}} \approx 1.5 \times 10^{-8}.
\]
With this value of \( h \), we find that the error at \( t = 10 \) is approximately \( 3.55 \times 10^{-12} \). This is even more accurate than with the previous choice of time step, which makes sense, because the new value of \( h \) is smaller. \( \square \)

Runge-Kutta Methods

We have seen that Euler’s method is first-order accurate. We would like to use Taylor series to design methods that have a higher order of accuracy. First, however, we must get around the fact that an analysis of the global error, as was carried out for Euler’s method, is quite cumbersome. Instead, we will design new methods based on the criteria that their local error, the error committed during a single time step, is higher-order in \( h \). We now define the concept of local error precisely.

**Definition** The local truncation error of the difference equation
\[
y_{n+1} = y_n + h\varphi(t_n, y_n)
\]
is
\[
\tau_{n+1}(h) = \frac{y(t_{n+1}) - y(t_n)}{h} - \varphi(t_n, y(t_n)).
\]
That is, the local truncation error is the residual due to substituting the exact solution \( y(t) \) into the numerical method described by the difference equation.

Note that due to the division by \( h \), which also occurred in the global error analysis of Euler’s method, the local truncation error is of the same order as the global error.

Using higher-order Taylor series directly to approximate \( y(t_{n+1}) \) is cumbersome, because it requires evaluating derivatives of \( f \). Therefore, our approach will be to use evaluations of \( f \) at carefully chosen values of its arguments, \( t \) and \( y \), in order to create an approximation that is just as accurate as a higher-order Taylor series expansion of \( y(t+h) \).

To find the right values of \( t \) and \( y \) at which to evaluate \( f \), we need to take a Taylor expansion of \( f \) evaluated at these (unknown) values, and then match the resulting numerical scheme to a Taylor series expansion of \( y(t+h) \) around \( t \). To that end, we state a generalization of Taylor’s theorem to functions of two variables.

**Theorem** Let \( f(t, y) \) be \((n+1)\) times continuously differentiable on a convex set \( D \), and let \((t_0, y_0) \in D\). Then, for every \((t, y) \in D\), there exists \( \xi \) between \( t_0 \) and \( t \), and \( \mu \) between \( y_0 \) and \( y \), such that
\[
f(t, y) = P_n(t, y) + R_n(t, y),
\]
where
\[
P_n(t, y) = \sum_{k=0}^{n} \frac{1}{k!} \left( \frac{\partial^n f}{\partial y^k} \right)(t_0, y_0) (y-y_0)^k + \sum_{k=0}^{n-1} \frac{k+1}{(k+1)!} \left( \frac{\partial^{k+1} f}{\partial y^{k+1}} \right)(t_0, y_0) (y-y_0)^{k+1}.
\]
where \( P_n(t, y) \) is the \( n \)th Taylor polynomial of \( f \) about \((t_0, y_0)\),
\[
P_n(t, y) = f(t_0, y_0) + \left[ (t - t_0) \frac{\partial f}{\partial t}(t_0, y_0) + (y - y_0) \frac{\partial f}{\partial y}(t_0, y_0) \right] + \]
\[
\left[ \frac{(t - t_0)^2}{2} \frac{\partial^2 f}{\partial t^2}(t_0, y_0) + (t - t_0)(y - y_0) \frac{\partial^2 f}{\partial t \partial y}(t_0, y_0) + \frac{(y - y_0)^2}{2} \frac{\partial^2 f}{\partial y^2}(t_0, y_0) \right] + \]
\[
\cdots + \left[ \frac{1}{n!} \sum_{j=0}^{n} \left( \begin{array}{c} n \\ j \end{array} \right) (t - t_0)^{n-j}(y - y_0)^j \frac{\partial^n f}{\partial t^{n-j} \partial y^j}(t_0, y_0) \right],
\]
and \( R_n(t, y) \) is the remainder term associated with \( P_n(t, y) \),
\[
R_n(t, y) = \frac{1}{(n + 1)!} \sum_{j=0}^{n+1} \left( \begin{array}{c} n + 1 \\ j \end{array} \right) (t - t_0)^{n+1-j}(y - y_0)^j \frac{\partial^{n+1} f}{\partial t^{n+1-j} \partial y^j}(\xi, \mu).
\]

We now illustrate our proposed approach in order to obtain a method that is second-order accurate; that is, its local truncation error is \( O(h^2) \). This involves matching
\[
y + hf(t, y) + \frac{h^2}{2} \frac{d}{dt}[f(t, y)] + \frac{h^3}{6} \frac{d^2}{dt^2}[f(\xi, y)]
\]
to
\[
y + ha_1 f(t + \alpha_1, y + \beta_1),
\]
where \( t \leq \xi \leq t + h \) and the parameters \( a_1, \alpha_1 \) and \( \beta_1 \) are to be determined. After simplifying by removing terms or factors that already match, we see that we only need to match
\[
f(t, y) + \frac{h}{2} \frac{d}{dt}[f(t, y)] + \frac{h^2}{6} \frac{d^2}{dt^2}[f(t, y(t))]
\]
with
\[
a_1 f(t + \alpha_1, y + \beta_1),
\]
at least up to terms of \( O(h) \), so that the local truncation error will be \( O(h^2) \).

Applying the multivariable version of Taylor’s theorem to \( f \), we obtain
\[
a_1 f(t + \alpha_1, y + \beta_1) = a_1 f(t, y) + a_1\alpha_1 \frac{\partial f}{\partial t}(t, y) + a_1\beta_1 \frac{\partial f}{\partial y}(t, y) + \]
\[
\frac{\alpha_1^2}{2} \frac{\partial^2 f}{\partial t^2}(\xi, \mu) + \alpha_1\beta_1 \frac{\partial^2 f}{\partial t \partial y}(\xi, \mu) + \frac{\beta_1^2}{2} \frac{\partial^2 f}{\partial y^2}(\xi, \mu),
\]
where \( \xi \) is between \( t \) and \( t + \alpha_1 \) and \( \mu \) is between \( y \) and \( y + \beta_1 \). Meanwhile, computing the full derivatives with respect to \( t \) in the Taylor expansion of the solution yields
\[
f(t, y) + \frac{h}{2} \frac{\partial f}{\partial t}(t, y) + \frac{h}{2} \frac{\partial f}{\partial y}(t, y)f(t, y) + O(h^2).
Comparing terms yields the equations

\[ a_1 = 1, \quad a_1 \alpha_1 = \frac{h}{2}, \quad a_1 \beta_1 = \frac{h}{2} f(t, y), \]

which has the solutions

\[ a_1 = 1, \quad \alpha_1 = \frac{h}{2}, \quad \beta_1 = \frac{h}{2} f(t, y). \]

The resulting numerical scheme is

\[ y_{n+1} = y_n + f \left( t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n) \right). \]

This scheme is known as the midpoint method, or the explicit midpoint method. Note that it evaluates \( f \) at the midpoint of the intervals \([t_n, t_{n+1}]\) and \([y_n, y_{n+1}]\), where the midpoint in \( y \) is approximated using Euler’s method with time step \( h/2 \).

The midpoint method is the simplest example of a Runge-Kutta method, which is the name given to any of a class of time-stepping schemes that are derived by matching multivariable Taylor series expansions of \( f(t, y) \) with terms in a Taylor series expansion of \( y(t+h) \). Another often-used Runge-Kutta method is the modified Euler method

\[ y_{n+1} = y_n + \frac{h}{2} \left[ f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n)) \right], \]

which resembles the Trapezoidal Rule from numerical integration, and is also second-order accurate.

However, the best-known Runge-Kutta method is the fourth-order Runge-Kutta method, which uses four evaluations of \( f \) during each time step. The method proceeds as follows:

\[
\begin{align*}
k_1 &= hf(t_n, y_n), \\
k_2 &= hf \left( t_n + \frac{h}{2}, y_n + \frac{1}{2} k_1 \right), \\
k_3 &= hf \left( t_n + \frac{h}{2}, y_n + \frac{1}{2} k_2 \right), \\
k_4 &= hf \left( t_{n+1}, y_n + k_3 \right), \\
y_{n+1} &= y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4).
\end{align*}
\]

In a sense, this method is similar to Simpson’s Rule from numerical integration, which is also fourth-order accurate, as values of \( f \) at the midpoint in time are given four times as much weight as values at the endpoints \( t_n \) and \( t_{n+1} \).

**Example** We compare Euler’s method with the fourth-order Runge-Kutta scheme on the initial value problem

\[ y' = -2ty, \quad 0 < t \leq 1, \quad y(0) = 1, \]
Figure 1: Solutions of $y' = -2ty$, $y(0) = 1$ on $[0,1]$, computed using Euler’s method and the fourth-order Runge-Kutta method which has the exact solution $y(t) = e^{-t^2}$. We use a time step of $h = 0.1$ for both methods. The computed solutions, and the exact solution, are shown in Figure 1.

It can be seen that the fourth-order Runge-Kutta method is far more accurate than Euler’s method, which is first-order accurate. In fact, the solution computed using the fourth-order Runge-Kutta method is visually indistinguishable from the exact solution. At the final time $T = 1$, the relative error in the solution computed using Euler’s method is $0.038$, while the relative error in the solution computing using the fourth-order Runge-Kutta method is $4.4 \times 10^{-6}$. □