Chapter 8

Ordinary differential equations

Consider first a scalar, first-order ordinary differential equation (ODE) of the form

$$\frac{dy}{dt} = f(y,t) \qquad \text{with} \qquad y(t_0) = y_0. \tag{8.1}$$

The problem we address now is the advancement of such a system in time by integration of this differential equation. As the quantity being integrated, f, is itself a function of the result of the integration, y, the problem of integration of an ODE is fundamentally different than the problem of numerical quadrature discussed in §7, in which the function being integrated was given. Note that ODEs with higher-order derivatives and systems of ODEs present a straightforward generalization of the present discussion, as will be shown in due course. Note also that we refer to the independent variable in this chapter as time, t, but this is done without loss of generality and other interpretations of the independent variable are also possible.

The ODE given above may be "solved" numerically by marching it forward in time, step by step. In other words, we seek to approximate the solution y to (8.1) at timestep $t_{n+1} = t_n + h_n$ given the solution at the initial time t_0 and the solution at the previously-computed timesteps t_1 to t_n. For simplicity of notation, we will focus our discussion initially on the case with constant stepsize h; generalization to the case with nonconstant h_n is straightforward.

8.1 Taylor-series methods

One of the simplest approaches to march the ODE (8.1) forward in time is to appeal to a Taylor series expansion in time, such as

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \dots . \tag{8.2}$$

From our ODE, we have:

$$y' = \frac{dy}{dt} = f$$

$$y'' = \frac{dy'}{dt} = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt} = f_t + ff_y$$

$$y''' = \frac{dy''}{dt} = \frac{d}{dt}(f_t + ff_y) = \frac{\partial}{\partial t}(f_t + ff_y) + \frac{\partial}{\partial y}(f_t + ff_y)\frac{dy}{dt} = f_{tt} + f_t f_y + 2ff_{yt} + f_y^2 f + f^2 f_{yy},$$

etc. Denoting the numerical approximation of $y(t_n)$ as y_n, the time integration method based on the first two terms of (8.2) is given by

$$\boxed{y_{n+1} = y_n + hf(y_n, t_n).} \tag{8.3}$$

This is referred to as the **explicit Euler** method, and is the simplest of all time integration schemes. Note that this method neglects terms which are proportional to h^2, and thus is "second-order" accurate over a single time step. As with the problem of numerical quadrature, however, a more relevant measure is the accuracy achieved when marching the ODE over a given time interval $(t_0, t_0 + T)$ as the timesteps h are made smaller. In such a setting, we lose one in the order of accuracy (as in the quadrature problem discussed in §7) and thus, over a specified time interval $(t_0, t_0 + T)$, **explicit Euler is first-order accurate**.

We can also base a time integration scheme on the first three terms of (8.2):

$$y_{n+1} = y_n + hf(y_n, t_n) + \frac{h^2}{2}[f_t(y_n, t_n) + f(y_n, t_n)f_y(y_n, t_n)].$$

Even higher-order Taylor series methods are also possible. We do not pursue such high-order Taylor series approaches in the present text, however, as their computational expense is relatively high (due to all of the cross derivatives required) and their stability and accuracy is not as good as some of the other methods which we will develop.

Note that a Taylor series expansion in time may also be written around t_{n+1}:

$$y(t_n) = y(t_{n+1}) - hy'(t_{n+1}) + \frac{h^2}{2}y''(t_{n+1}) - \frac{h^3}{6}y'''(t_{n+1}) + \dots$$

The time integration method based on the first two terms of this Taylor series is given by

$$\boxed{y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}).} \tag{8.4}$$

This is referred to as the **implicit Euler** method. It also neglects terms which are proportional to h^2, and thus is "second-order" accurate over a single time step. As with explicit Euler, over a specified time interval $(t_0, t_0 + T)$, **implicit Euler is first-order accurate**.

If f is nonlinear in y, implicit methods such as the implicit Euler method given above are difficult to use, because knowledge of y_{n+1} is needed (before it is computed!) to compute f in order to advance from y_n to y_{n+1}. Typically, such problems are approximated by some type of linearization or iteration, as will be discussed further in class. On the other hand, if f is linear in y, implicit strategies such as (8.4) are easily solved for y_{n+1}.

8.2 The trapezoidal method

The formal solution of the ODE (8.1) over the interval $[t_n, t_{n+1}]$ is given by

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(y, t)dt.$$

Approximating this integral with the trapezoidal rule from §7.1.1 gives

$$\boxed{y_{n+1} = y_n + \frac{h}{2}[f(y_n, t_n) + f(y_{n+1}, t_{n+1})].} \tag{8.5}$$

This is referred to as the **trapezoidal** or **Crank-Nicholson** method. We defer discussion of the accuracy of this method to §8.4, after we discuss first an illustrative model problem.

8.3 A model problem

A scalar model problem which is very useful for characterizing various time integration strategies is

$$y' = \lambda y \quad \text{with} \quad y(t_0) = y_0, \tag{8.6}$$

where λ is, in general, allowed to be complex. The exact solution of this problem is $y = y_0 e^{\lambda(t-t_0)}$. The utility of this model problem is that the exact solution is available, so we can compare the numerical approximation using a particular numerical method to the exact solution in order to quantify the pros and cons of the numerical method. The insight we gain by studying the application of the numerical method we choose to this simple model problem allows us to predict how this method will work on more difficult problems for which the exact solution is not available.

Note that, for $\Re(\lambda) > 0$, the magnitude of the exact solution grows without bound. We thus refer to the exact solution as being **unstable** if $\Re(\lambda) > 0$ and **stable** if $\Re(\lambda) \leq 0$. Graphically, we denote the region of stability of the exact solution in the complex plane λ by the shaded region shown in Figure 8.1.

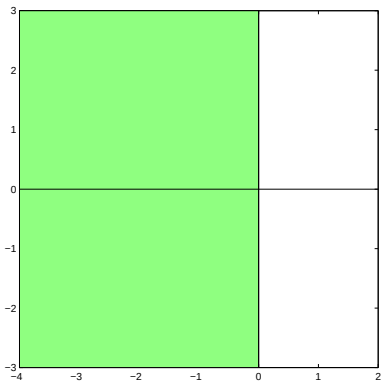

Figure 8.1: Stability of the exact solution to the model problem $y' = \lambda y$ in the complex plane λ.

8.3.1 Simulation of an exponentially-decaying system

Consider now the model problem (8.6) with $\lambda = -1$. The exact solution of this system is simply a decaying exponential. In Figure 8.2, we show the application of the explicit Euler method, the implicit Euler method, and the trapezoidal method to this problem. Note that the explicit Euler method appear to be unstable for the large values of h. Note also that all three methods are more accurate as h is refined, with the trapezoidal method appearing to be the most accurate.

8.3.2 Simulation of an undamped oscillating system

Consider now the second-order ODE for a simple mass/spring system given by

$$y'' = -\omega^2 y \quad \text{with} \quad y(t_0) = y_0, \quad y'(t_0) = 0, \tag{8.7}$$

where $\omega = 1$. The exact solution is $y = y_0 \cos[\omega(t - t_0)] = (y_0/2)[e^{i\omega(t-t_0)} + e^{-i\omega(t-t_0)}]$.

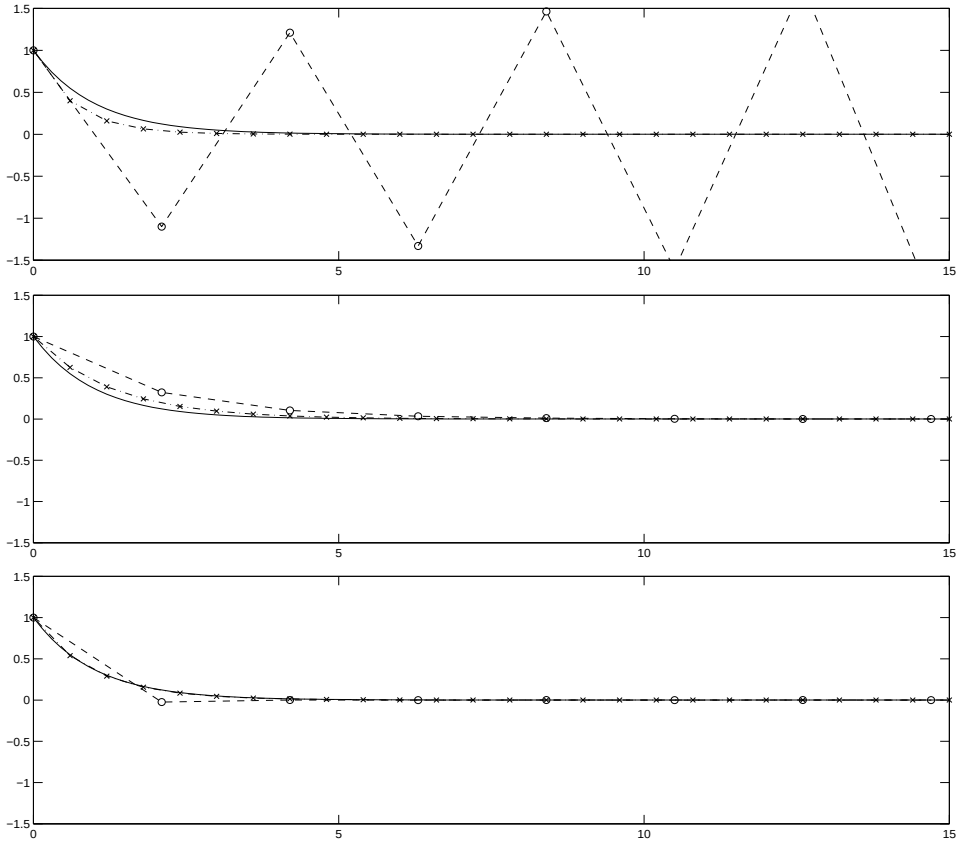

Figure 8.2: Simulation of the model problem $y' = \lambda y$ with $\lambda = -1$ using the explicit Euler method (top), the implicit Euler method (middle), and the trapezoidal method (bottom). Symbols denote: $\circ$, $h = 2.1$; $\times$, $h = 0.6$; ——, exact solution.

We may easily write this second-order ODE as a first-order system of ODEs by defining $y_1 = y$ and $y_2 = y'$ and writing:

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}'}_{\mathbf{y}'} = \underbrace{\begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}}_{\mathbf{y}}. \tag{8.8}$$

The eigenvalues of A are $\pm i\omega$. Note that the eigenvalues are imaginary; if we has started with the equation for a damped oscillator, the eigenvalues would have a negative real part as well. Note also that A may be diagonalized by its matrix of eigenvectors:

$$A = S\Lambda S^{-1} \quad \text{where} \quad \Lambda = \begin{pmatrix} i\omega & 0 \\ 0 & -i\omega \end{pmatrix}.$$

Thus, we have

$$\mathbf{y}' = S\Lambda S^{-1}\mathbf{y} \quad \Rightarrow \quad S^{-1}\mathbf{y}' = \Lambda S^{-1}\mathbf{y} \quad \Rightarrow \quad \mathbf{z}' = \Lambda \mathbf{z},$$

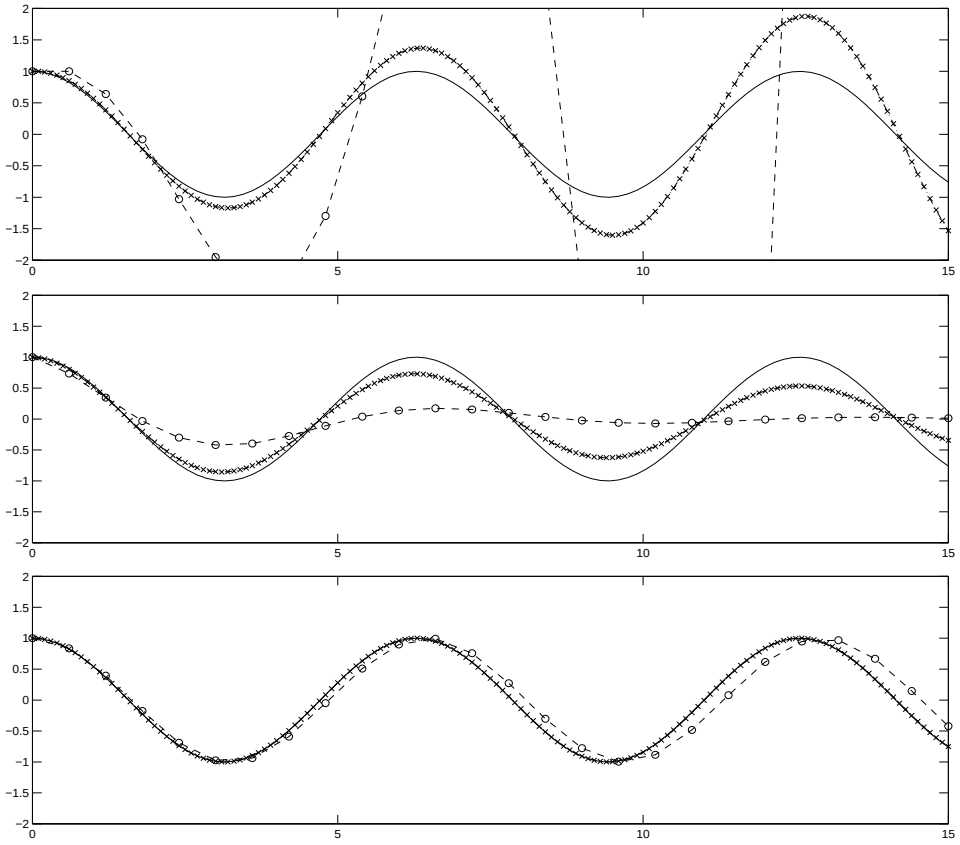

Figure 8.3: Simulation of the oscillatory system $y'' = -\omega^2 y$ with $\omega = 1$ using the explicit Euler method (top), the implicit Euler method (middle), and the trapezoidal method (bottom). Symbols denote: $\circ$, $h = 0.6$; $\times$, $h = 0.1$; ——, exact solution.

where we have defined $\mathbf{z} = S^{-1}\mathbf{y}$. In terms of the components of $\mathbf{z}$, we have decoupled the dynamics of the system:

$$z_1' = i\omega z_1$$
$$z_2' = -i\omega z_2.$$

Each of these systems is exactly the same form as our scalar model problem (8.6) with complex (in this case, pure imaginary) values for λ. Thus, eigenmode decompositions of physical systems (like mass/spring systems) motivate us to look at the scalar model problem (8.6) over the complex plane λ. In fact, our original second-order system (8.7), as re-expressed in (8.8), will be stable iff there are no eigenvalues of A with $\Re(\lambda) > 0$.

In Figure 8.3, we show the application of the explicit Euler method, the implicit Euler method, and the trapezoidal method to the first-order system of equations (8.8). Note that the explicit Euler method appears to be unstable for both large and small values of h. Note also that all three methods are more accurate as h is refined, with the trapezoidal method appearing to be the most accurate.

We see that some numerical methods for time integration of ODEs are more accurate than others, and some numerical techniques are sometimes unstable, even for ODEs with stable exact solutions.

In the next two sections, we develop techniques to quantify both the stability and the accuracy of numerical methods for time integration of ODEs by application of these numerical methods to the model problem (8.6).

8.4 Stability

For stability of a numerical method for time integration of an ODE, we want to insure that, if the exact solution is bounded, the numerical solution is also bounded. We often need to restrict the timestep h in order to insure this. To make this discussion concrete, consider a system whose exact solution is bounded and define:
1) a stable numerical scheme: one which does not blow up for any h,
2) an unstable numerical scheme: one which blows up for any h, and
3) a conditionally stable numerical scheme: one which blows up for some h.

8.4.1 Stability of the explicit Euler method

Applying the explicit Euler method (8.3) to the model problem (8.6), we see that

$$y_{n+1} = y_n + \lambda h y_n = (1 + \lambda h) y_n.$$

Thus, assuming constant h, the solution at time step n is:

$$y_n = (1 + \lambda h)^n y_0 \triangleq \sigma^n y_0 \qquad \Rightarrow \qquad \sigma = 1 + \lambda h.$$

For large n, the numerical solution remains stable iff

$$|\sigma| \leq 1 \qquad \Rightarrow \qquad (1 + \lambda_R h)^2 + (\lambda_I h)^2 \leq 1.$$

The region of the complex plane which satisfies this stability constraint is shown in Figure 8.4. Note that this region of stability in the complex plane λh is consistent with the numerical simulations shown in Figure 8.2a and 8.3a: for real, negative λ, this numerical method is conditionally stable (*i.e.*, it is stable for sufficiently small h), whereas for pure imaginary λ, this numerical method is unstable for any h, though the instability is mild for small h.

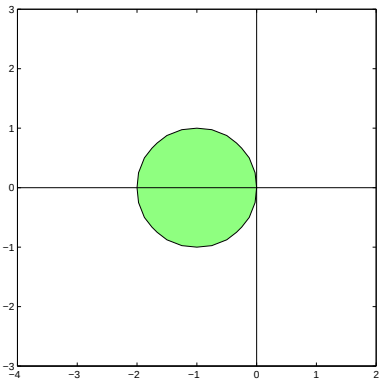

Figure 8.4: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using the explict Euler method.

8.4.2 Stability of the implicit Euler method

Applying the implicit Euler method (8.4) to the model problem (8.6), we see that

$$y_{n+1} = y_n + \lambda h y_{n+1} \qquad \Rightarrow \qquad y_{n+1} = (1 - \lambda h)^{-1} y_n.$$

Thus, assuming constant h, the solution at time step n is:

$$y_n = \left(\frac{1}{1 - \lambda h} \right)^n y_0 \triangleq \sigma^n y_0 \qquad \Rightarrow \qquad \sigma = \frac{1}{1 - \lambda h}.$$

For large n, the numerical solution remains stable iff

$$|\sigma| \leq 1 \qquad \Rightarrow \qquad (1 - \lambda_R h)^2 + (\lambda_I h)^2 \geq 1.$$

The region of the complex plane which satisfies this stability constraint is shown in Figure 8.5. Note that this region of stability in the complex plane λh is consistent with the numerical simulations shown in Figure 8.2b and 8.3b: this method is stable for any stable ODE for any h, and is even stable for some cases in which the ODE itself is unstable.

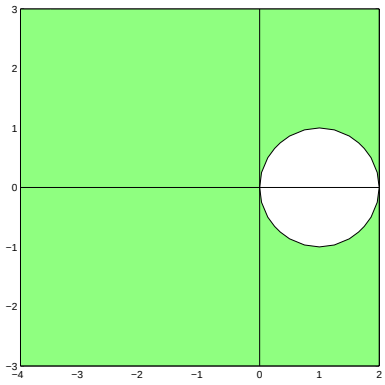

Figure 8.5: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using the implicit Euler method.

8.4.3 Stability of the trapezoidal method

Applying the trapezoidal method (8.5) to the model problem (8.6), we see that

$$y_{n+1} = y_n + \frac{\lambda h}{2}(y_n + y_{n+1}) \qquad \Rightarrow \qquad y_{n+1} = \left(\frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}} \right) y_n.$$

Thus, assuming constant h, the solution at time step n is:

$$y_n = \left(\frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}} \right)^n y_0 \triangleq \sigma^n y_0 \qquad \Rightarrow \qquad \sigma = \frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}}.$$

For large n, the numerical solution remains stable iff

$$|\sigma| \leq 1 \qquad \Rightarrow \quad \ldots \quad \Rightarrow \qquad \Re(\lambda h) \leq 0.$$

The region of the complex plane which satisfies this stability constraint coincides exactly with the region of stability of the exact solution, as shown in Figure 8.6. Note that this region of stability in the complex plane λh is consistent with the numerical simulations shown in Figure 8.2c and 8.3c, which are stable for systems with $\Re(\lambda) < 0$ and marginally stable for systems with $\Re(\lambda) = 0$.

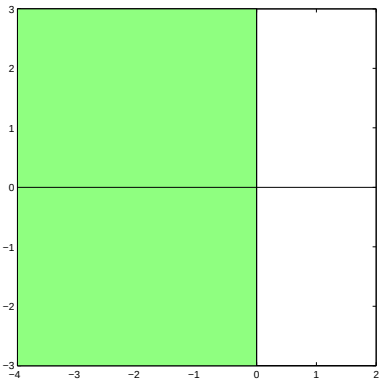

Figure 8.6: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using the trapezoidal method.

8.5 Accuracy

Revisiting the model problem $y' = \lambda y$, the exact solution (assuming $t_0 = 0$ and $h =$ constant) is

$$y(t_n) = e^{\lambda t_n} y_0 = (e^{\lambda h})^n y_0 = \left(1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \dots\right)^n y_0.$$

On the other hand, solving the model problem with explicit Euler led to

$$y_n = (1 + \lambda h)^n y_0 \triangleq \sigma^n y_0,$$

solving the model problem with implicit Euler led to

$$y_n = \left(\frac{1}{1 - \lambda h}\right)^n y_0 = \left(1 + \lambda h + \lambda^2 h^2 + \lambda^3 h^3 + \dots\right)^n y_0 \triangleq \sigma^n y_0,$$

and solving the model problem with trapezoidal led to

$$y_n = \left(\frac{1 + \frac{\lambda h}{2}}{1 - \frac{\lambda h}{2}}\right)^n y_0 = \left(1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{4} + \dots\right)^n y_0 \triangleq \sigma^n y_0.$$

To quantify the accuracy of these three methods, we can compare the amplification factor σ in each of the numerical approximations to the exact value $e^{\lambda h}$. The leading order error of the explicit Euler and implicit Euler methods are seen to be proportional to h^2, as noted in §8.1, and the leading order error of the trapezoidal method is proportional to h^3. Thus, over a specified time interval $(t_0, t_0 + T)$, **explicit Euler and implicit Euler are first-order accurate** and **trapezoidal is second-order accurate**. The higher order of accuracy of the trapezoidal method implies an improved rate of convergence of this scheme to the exact solution as the timestep h is refined, as observed in Figures 8.2 and 8.3.

8.6 Runge-Kutta methods

An important class of explicit methods, called Runge-Kutta methods, is given by the general form:

$$\begin{aligned} k_1 &= f\Big(y_n, t_n\Big) \\ k_2 &= f\Big(y_n + \beta_1\, h\, k_1, t_n + \alpha_1\, h\Big) \\ k_3 &= f\Big(y_n + \beta_2\, h\, k_1 + \beta_3\, h\, k_2, t_n + \alpha_2\, h\Big) \\ &\vdots \\ y_{n+1} &= y_n + \gamma_1\, h\, k_1 + \gamma_2\, h\, k_2 + \gamma_3\, h\, k_3 + \dots\,, \end{aligned} \tag{8.9}$$

where the constants α_i, β_i, and γ_i are selected to match as many terms as possible of the exact solution:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \dots$$

where

$$\begin{aligned} y' &= f \\ y'' &= f_t + ff_y \\ y''' &= f_{tt} + f_t f_y + 2ff_{yt} + f_y^2 f + f^2 f_{yy}, \end{aligned}$$

etc. Runge-Kutta methods are explicit and "self starting", as they don't require any information about the numerical approximation of the solution before time t_n; this typically makes them quite easy to use. As the number of intermediate steps k_i in the Runge-Kutta method is increased, the order of accuracy of the method can also be increased. The stability properties of higher-order Runge-Kutta methods are also generally quite favorable, as will be shown.

8.6.1 The class of second-order Runge-Kutta methods (RK2)

Consider first the family of two-step schemes of the form (8.9):

$$\begin{aligned} k_1 &= f(y_n, t_n), \\ k_2 &= f(y_n + \beta_1\, h\, k_1, t_n + \alpha_1\, h) \\ &\approx f(y_n, t_n) + f_y(y_n, t_n)\,\Big(\beta_1\, h\, f(y_n, t_n)\Big) + f_t(y_n, t_n)\,\Big(\alpha_1\, h\Big), \\ y_{n+1} &= y_n + \gamma_1\, h\, k_1 + \gamma_2\, h\, k_2 \\ &\approx y_n + \gamma_1\, h\, f(y_n, t_n) + \gamma_2\, h\,\Big(f(y_n, t_n) + \beta_1\, h\, f_y(y_n, t_n)\, f(y_n, t_n) + \alpha_1\, h\, f_t(y_n, t_n)\Big) \\ &\approx y_n + (\gamma_1 + \gamma_2)\, h\, f(y_n, t_n) + \gamma_2\, h^2\, \beta_1\, f_y(y_n, t_n)\, f(y_n, t_n) + \gamma_2\, h^2\, \alpha_1\, f_t(y_n, t_n). \end{aligned}$$

Note that the approximations given above are exact if f is linear in y and t, as it is in our model problem. The exact solution we seek to match with this scheme is given by

$$y(t_{n+1}) = y(t_n) + hf(y_n, t_n) + \frac{h^2}{2}\Big(f_t(y_n, t_n) + f(y_n, t_n) f_y(y_n, t_n)\Big) + \dots$$

Matching coefficients to as high an order as possible, we require that

$$\left.\begin{aligned} \gamma_1 + \gamma_2 &= 1 \\ \gamma_2\, h^2\, \beta_1 &= \frac{h^2}{2} \\ \gamma_2\, h^2\, \alpha_1 &= \frac{h^2}{2} \end{aligned}\right\} \quad \Rightarrow \qquad \alpha_1 = \beta_1, \quad \gamma_2 = \frac{1}{2\alpha_1}, \quad \gamma_1 = 1 - \frac{1}{2\alpha_1}.$$

Thus, the general form of the two-step second-order Runge-Kutta method (RK2) is

$$\begin{aligned} k_1 &= f\Big(y_n, t_n\Big) \\ k_2 &= f\Big(y_n + \alpha\, h\, k_1, t_n + \alpha\, h\Big) \\ y_{n+1} &= y_n + \Big(1 - \frac{1}{2\alpha}\Big) h\, k_1 + \Big(\frac{1}{2\alpha}\Big) h\, k_2, \end{aligned} \tag{8.10}$$

where α is a free parameter. A popular choice is $\alpha = 1/2$, which is known as the midpoint method and has a clear geometric interpretation of approximating a central difference formula in the integration of the ODE from t_n to t_{n+1}. Another popular choice is $\alpha = 1$, which is equivalent to perhaps the most common so-called "predictor-corrector" scheme, and may be computed in the following order:

$$\begin{aligned} \text{predictor}: \quad & y^*_{n+1} = y_n + h f(y_n, t_n) \\ \text{corrector}: \quad & y_{n+1} = y_n + \frac{h}{2}\Big[f(y_n, t_n) + f(y^*_{n+1}, t_{n+1})\Big]. \end{aligned}$$

The "predictor" (which is simply an explicit Euler estimate of y_{n+1}) is only "stepwise 2nd-order accurate". However, as we shown below, calculation of the "corrector" (which looks roughly like a recalculation of y_{n+1} with a trapezoidal rule) results in a value for y_{n+1} which is "stepwise 3rd-order accurate" (and thus the scheme is globally 2nd-order accurate).

Applying an RK2 method (for some value of the free parameter α) to the model problem $y' = \lambda y$ yields

$$\begin{aligned} y_{n+1} &= y_n + \Big(1 - \frac{1}{2\alpha}\Big) h\, \lambda y_n + \Big(\frac{1}{2\alpha}\Big) h\, \lambda (1 + \alpha\, h\, \lambda) y_n \\ &= \Big(1 + \lambda h + \frac{\lambda^2\, h^2}{2}\Big) y_n \triangleq \sigma\, y_n \qquad \Rightarrow \qquad \sigma = 1 + \lambda h + \frac{\lambda^2\, h^2}{2}. \end{aligned}$$

The amplification factor σ is seen to be a truncation of the Taylor series of the exact value $e^{\lambda h} = 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \ldots$ We thus see that the leading order error of this method (for any value of α) is proportional to h^3 and, over a specified time interval $(t_0, t_0 + T)$, **an RK2 method is second-order accurate**. Over a large number of timesteps, the method is stable iff $|\sigma| \leq 1$; the domain of stability of this method is illustrated in Figure 8.7.

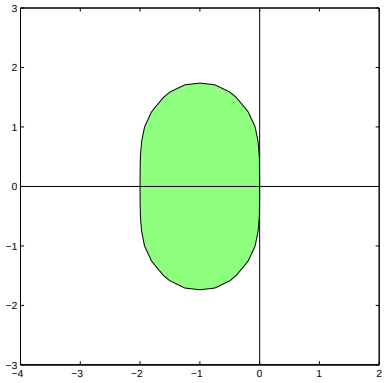

Figure 8.7: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using RK2.

8.6.2 A popular fourth-order Runge-Kutta method (RK4)

The most popular fourth-order Runge-Kutta method is

$$\begin{aligned} k_1 &= f\Big(y_n, t_n\Big) \\ k_2 &= f\Big(y_n + \frac{h}{2}\,k_1, t_{n+1/2}\Big) \\ k_3 &= f\Big(y_n + \frac{h}{2}\,k_2, t_{n+1/2}\Big) \\ k_4 &= f\Big(y_n + h\,k_3, t_{n+1}\Big) \\ y_{n+1} &= y_n + \frac{h}{6}k_1 + \frac{h}{3}\Big(k_2 + k_3\Big) + \frac{h}{6}k_4 \end{aligned} \tag{8.11}$$

This scheme usually performs very well, and is the workhorse of many ODE solvers. This particular RK4 scheme also has a reasonably-clear geometric interpretation, as discussed further in class.

A derivation similar to that in the previous section confirms that the constants chosen in (8.11) indeed provide fourth-order accuracy, with the $\lambda - \sigma$ relationship again given by a truncated Taylor series of the exact value:

$$\sigma = 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24}.$$

Over a large number of timesteps, the method is stable iff $|\sigma| \leq 1$; the domain of stability of this method is illustrated in Figure 8.8.

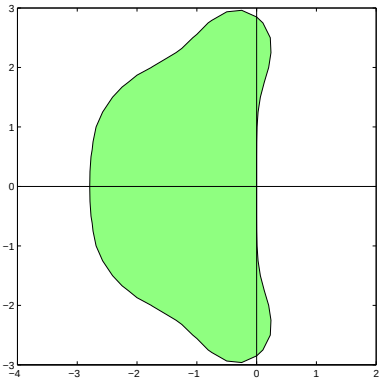

Figure 8.8: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using RK4.

8.6.3 An adaptive Runge-Kutta method (RKM4)

Another popular fourth-order scheme, known as the Runge-Kutta-Merson method, is

$$\begin{aligned} k_1 &= f\Big(y_n, t_n\Big) \\ k_2 &= f\Big(y_n + \frac{h}{3}\,k_1, t_{n+1/3}\Big) \\ k_3 &= f\Big(y_n + \frac{h}{6}(k_1 + k_2), t_{n+1/3}\Big) \\ k_4 &= f\Big(y_n + \frac{h}{8}(k_1 + 3k_3), t_{n+1/2}\Big) \\ y^*_{n+1} &= y_n + \frac{h}{2}k_1 - \frac{3h}{2}k_3 + 2hk_4 \\ k_5 &= f\Big(y^*_{n+1}, t_{n+1}\Big) \\ y_{n+1} &= y_n + \frac{h}{6}k_1 + \frac{2h}{3}k_4 + \frac{h}{6}k_5. \end{aligned} \tag{8.12}$$

Note that one extra computation of f is required in this method as compared with the method given in (8.11). With the same sort of analysis as we did for RK2, it may be shown that both y^*_{n+1} and y_{n+1} are "stepwise 5th-order accurate", meaning that using either to advance in time over a given interval gives global 4th-order accuracy. In fact, if $\tilde{y}(t)$ is the exact solution to an ODE and y_n takes this exact value of $\tilde{y}(t_n)$ at $t = t_n$, then it follows after a bit of analysis that the errors in y^*_{n+1} and y_{n+1} are

$$y^*_{n+1} - \tilde{y}(t_{n+1}) = -\frac{h^5}{120}\tilde{y}^{(v)} + O(h^6) \tag{8.13}$$

$$y_{n+1} - \tilde{y}(t_{n+1}) = -\frac{h^5}{720}\tilde{y}^{(v)} + O(h^6). \tag{8.14}$$

Subtracting (8.13) from (8.14) gives

$$y_{n+1} - y^*_{n+1} = \frac{h^5}{144}\tilde{y}^{(v)} + O(h^6),$$

which may be substituted on the RHS of (8.14) to give

$$y_{n+1} - \tilde{y}(t_{n+1}) = -\frac{1}{5}(y_{n+1} - y^*_{n+1}) + O(h^6). \tag{8.15}$$

The quantity on the LHS of (8.15) is the error of our current "best guess" for y_{n+1}. The first term on the RHS is something we can compute, even if we don't know the exact solution $\tilde{y}(t)$. Thus, even if the exact solution $\tilde{y}(t)$ is unknown, we can still estimate the error of our best guess of y_{n+1} with quantities which we have computed. We may use this estimate to decide whether or not to refine or coarsen the stepsize h to attain a desired degree of accuracy on the entire interval. As with the procedure of adaptive quadrature, it is straightforward to determine whether or not the error on any particular step is small enough such that, when the entire (global) error is added up, it will be within a predefined acceptable level of tolerance.

8.6.4 A low-storage Runge-Kutta method (RKW3)

Amongst people doing very large simulations with specialized solvers, a third-order scheme which is rapidly gaining popularity, known as the Runge-Kutta-Wray method, is

$$\begin{aligned} k_1 &= f\Big(y_n, t_n\Big) \\ k_2 &= f\Big(y_n + \beta_1\, h\, k_1, t_n + \alpha_1\, h\Big) \\ k_3 &= f\Big(y_n + \beta_2\, h\, k_1 + \beta_3\, h\, k_2, t_n + \alpha_2\, h\Big) \\ y_{n+1} &= y_n + \gamma_1\, h\, k_1 + \gamma_2\, h\, k_2 + \gamma_3\, h\, k_3, \end{aligned} \tag{8.16}$$

where

$$\begin{aligned} &\beta_1 = 8/15, \quad &&\beta_2 = 1/4, \quad &&\beta_3 = 5/12, \\ &\alpha_1 = 8/15, \quad &&\alpha_2 = 2/3, \quad && \\ &\gamma_1 = 1/4, \quad &&\gamma_2 = 0, \quad &&\gamma_3 = 3/4. \end{aligned}$$

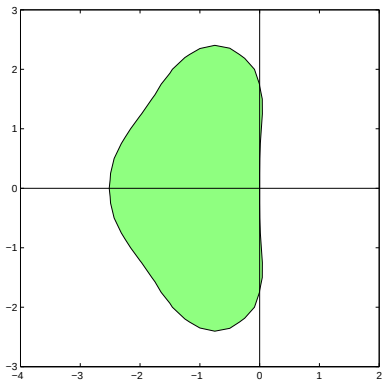

Figure 8.9: Stability of the numerical solution to $y' = \lambda y$ in the complex plane λh using third-order Runge-Kutta.