Chapter 7

Ordinary Differential Equations

7.1 Euler's method

Consider the general first-order ordinary differential equation (ODE),

$$y' = f(x, y) \tag{7.1}$$

(where y' denotes the derivative dy/dx), subject to the general initial-value boundary condition

$$y(x_0) = y_0 \ . \tag{7.2}$$

If we can find a method for numerically solving this problem, then we should have little difficulty generalizing it to deal with a system of n simultaneous first-order ODEs.

It is important to appreciate that the numerical solution to a differential equation is only an approximation to the actual solution. The actual solution, $y(x)$, to Eq. (7.1) is (presumably) a continuous function of a continuous variable, x. However, when we solve this equation numerically, the best we can do is to evaluate approximations to the function $y(x)$ at a series of discrete grid-points x_n, where $n = 0, 1, 2, \ldots$ and $x_0 < x_1 < x_2 < \ldots$. For the moment, we shall restrict our discussion to equally spaced grid-points, where

$$x_n = x_0 + n\,h. \tag{7.3}$$

Here, the quantity h is referred to as the step-length. Let y_n be our approximation to $y(x)$ at the grid-point x_n. A numerical integration scheme is essentially a method which somehow employs the information contained in the original ODE (7.1) to construct a series of rules interrelating the various y_n.

Suppose that we have evaluated an approximation y_n to the solution $y(x)$ of Eq. (7.1) at the grid-point x_n. The approximate gradient of $y(x)$ at this point is therefore given by

$$\tilde{y}'_n = f(x_n, \tilde{y}_n). \tag{7.4}$$

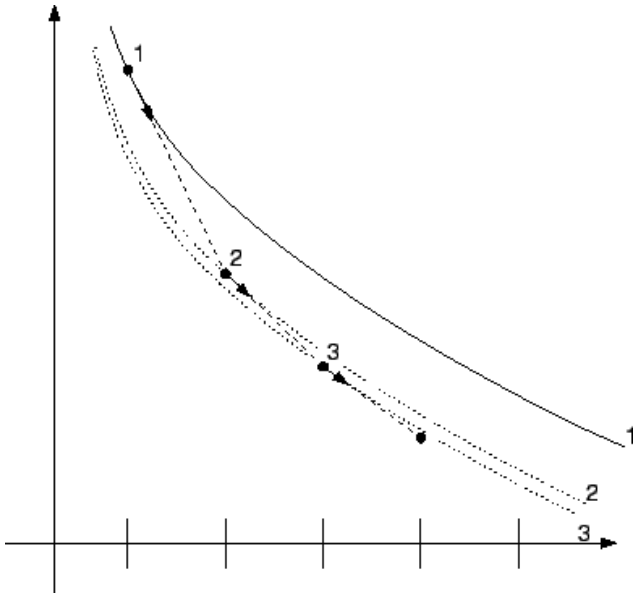

Figure 7.1: Graphical illustration of the forward-Euler method for an exponential-like curve. Starting at point 1, the tangent of the curve is taken and linearly extrapolated to obtain point 2. There again the same procedure is used to obtain point 3. Note that point 2 lies on curve 2 and point three lies on curve 3, both of which are offset against the original curve.

Let us approximate the curve $y(x)$ as a straight-line between the neighbouring grid-points x_n and x_{n+1}. It follows that

$$\tilde{y}_{n+1} = \tilde{y}_n + \tilde{y}'_n\, h, \tag{7.5}$$

or

$$\tilde{y}_{n+1} = \tilde{y}_n + f(x_n, \tilde{y}_n)\, h. \tag{7.6}$$

The above formula is the essence of Euler's method. It enables us to calculate all of the y_n, given the initial value y_0 at the first grid-point x_0. Euler's method is illustrated in Fig. 1.

7.1.1 Numerical errors

There are two major sources of error associated with a numerical integration scheme for ODEs: namely, truncation error and round-off error. Truncation error arises in Euler's method because the curve $y(x)$ is not generally a straight-line between the neighbouring grid-points x_n and x_{n+1}, as assumed above. The error associated with this approximation can easily be assessed by Taylor-expanding $y(x)$ about $x = x_n$:

$$\begin{aligned} y(x_n{+}h) &= y(x_n) + h\, y'(x_n) + \frac{h^2}{2}\, y''(x_n) + \ldots \\ &= y_n + h\, f(x_n, y_n) + \frac{h^2}{2}\, y''(x_n) + \ldots\ . \end{aligned} \tag{7.7}$$

A comparison of Eqs. (7.4) and (7.5) yields

$$y_{n+1} = y_n + h\, f(x_n, y_n) + O(h^2). \tag{7.8}$$

In other words, every time we take a step using Euler's method we get a truncation error of $O(h^2)$, where h is the step-length. Suppose we use Euler's method to integrate our ODE over the interval $[a, b]$. This requires $(b-a)/h = O(h^{-1})$ steps. If each step involves an error of $O(h^2)$, and the errors are simply cumulative (a somewhat conservative assumption), then the net truncation error is $O(h)$. In other words, the error associated with integrating an ODE over a finite interval using Euler's method is directly proportional to the step-length. Thus, if we want to keep the relative error in the integration below about 10^{-6} then we would need to take about one million steps per unit interval in x. Euler's method is termed a first-order integration method because the truncation error associated with integrating over a finite interval scales like h^1. More generally, an integration method is conventionally called n-th order accurate if its truncation error per step is $O(h^{n+1})$, i.e. if the error for integrating over a fixed interval is $O(h^n)$.

Note that truncation error would arise even if computers performed floating-point arithmetic operations with infinite accuracy. Real computers are only capable of storing a floating-point number to a fixed number of decimal places. For every type of computer, there is a characteristic positive number η, which is defined as the smallest number which when added to a number of order unity gives rise to a new number: i.e., a number which when taken away from the original number yields a non-zero result.[1] Every floating-point operation involves a round-off error of $O(\eta)$ which arises from the finite accuracy to which floating-point numbers are stored by the computer. Suppose again that we use Euler's method to integrate our ODE over an interval $[a, b]$. This entails $O(h^{-1})$ integration steps, and, therefore, $O(h^{-1})$ floating-point operations. If each floating-point operation incurs an error of $O(\eta)$, and the errors are simply cumulative, then the net round-off error is $O(\eta/h)$.

The total error ϵ associated with integrating our ODE over an x-interval of order unity, $b-a \simeq 1$, is (approximately) the sum of the truncation and round-off errors. Thus, for Euler's method,

$$\epsilon \simeq \frac{\eta}{h} + h \; . \tag{7.9}$$

Clearly, at large step-size the error is dominated by truncation error, whereas round-off error dominates at small step-size. The net error attains its minimum value, $\epsilon_0 \sim \eta^{1/2}$, when $h = h_0 \sim \eta^{1/2}$. There is clearly no point in making the step-length h any smaller than h_0, since this increases the number of floating-point operations but does not lead to an increase in the overall accuracy. It is also clear that the ultimate accuracy of Euler's method (or any other integration method) is determined by the accuracy η at which floating-point numbers are stored on the computer performing the calculation.

7.1.2 Numerical instability

Consider the following example. Suppose that our ODE is

$$y' = -\alpha \, y \; , \tag{7.10}$$

[1] In F90, this number can be obtained as '`epsilon(1.)`'

where $\alpha > 0$, subject to the boundary condition

$$y(0) = 1 \; . \tag{7.11}$$

Of course, we can solve this problem analytically to give

$$y(x) = \exp(-\alpha\, x) \; . \tag{7.12}$$

Note that the solution is a monotonically decreasing function of x. We can also solve this problem numerically using Euler's method. Appropriate grid-points are

$$x_n = n\, h \; , \tag{7.13}$$

where $n = 0, 1, 2, \ldots$. Euler's method yields

$$y_{n+1} = (1-\alpha\, h)\, y_n = (1-\alpha h)^2 y_{n-1} = (1-\alpha h)^3 y_{n-2} \; . \tag{7.14}$$

Note one curious fact. If $h > 2/\alpha$ then $1-\alpha h < -1$, and thus $|y_{n+1}| > |y_n|$. In other words, if the step-length is made too large then the numerical solution becomes an oscillatory function of x of monotonically increasing amplitude: i.e., the numerical solution diverges from the actual solution. This type of catastrophic failure of a numerical integration scheme is called a numerical instability. All simple integration schemes become unstable if the step-length is made sufficiently large.

7.1.3 Chaos from Euler solution of ODEs

According to the Poincaré–Bendixson theorem, chaos cannot occur in 2-dimensional systems of autonomous ordinary differential equations (ODEs) such as

$$\begin{aligned} \frac{dx}{dt} &= f(x, y) \; , \\ \frac{dy}{dt} &= g(x, y) \; . \end{aligned} \tag{7.15}$$

However, when such equations are solved numerically, the continuous flows represented by the ODEs are approximated by discrete-time maps. One conceptually simple method for implementing such a solution is the Euler method,

$$\begin{aligned} x_{n+1} &= x_n + h f(x_n, y_n) \; , \\ y_{n+1} &= y_n + h g(x_n, y_n) \; , \end{aligned} \tag{7.16}$$

where h is a small time step. If h is sufficiently small, the solution of the map approximates the solution of the flow with arbitrary accuracy, provided cumulative round-off error is kept small by calculating with adequate floating point number precision. However, if h is too large, the numerical solution will deviate significantly from the real solution and will

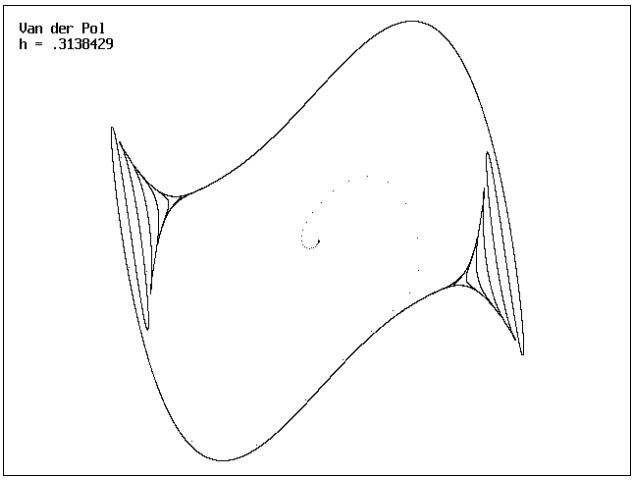

Figure 7.2: Chaotic solutions from using Euler's method to solve the Van der Pol equations (7.17), (7.18).

eventually approach a periodic cycle, or a strange attractor, or it will escape to infinity, often with all three behaviours appearing in sequence as h is increased.

As an example, consider the Van der Pol oscillator, given by

$$\frac{dx}{dt} = y\,, \tag{7.17}$$

$$\frac{dy}{dt} = b(1-x^2)y - x\,, \tag{7.18}$$

whose solution is a limit cycle. It has been used to model electrical oscillators, heartbeats, and pulsating stars called Cepheids. Setting $b = 1$ and using Euler's method to solve the equations, we find that for $h \lesssim 0.1$ the solution is reasonably accurate. For larger h, however, the typical periodic cycle to chaos and then to unbounded solutions is observed. An example of the solution in the chaotic regime for an initial condition of $x_0 = y_0 = 0.01$ is shown in Fig. 7.2.

A similar system with a circular limit-cycle ($x^2 + y^2 = 1$) attractor is given by the equations

$$\frac{dx}{dt} = y\,, \tag{7.19}$$

$$\frac{dy}{dt} = (1-x^2-y^2)y - x\,. \tag{7.20}$$

Its behaviour is similar to the Van der Pol equation and with a sufficiently large h produces the strange attractor shown in Figure 7.3.

A third example is the undamped harmonic oscillator

$$\frac{dx}{dt} = y\,, \tag{7.21}$$

$$\frac{dy}{dt} = -x\,, \tag{7.22}$$

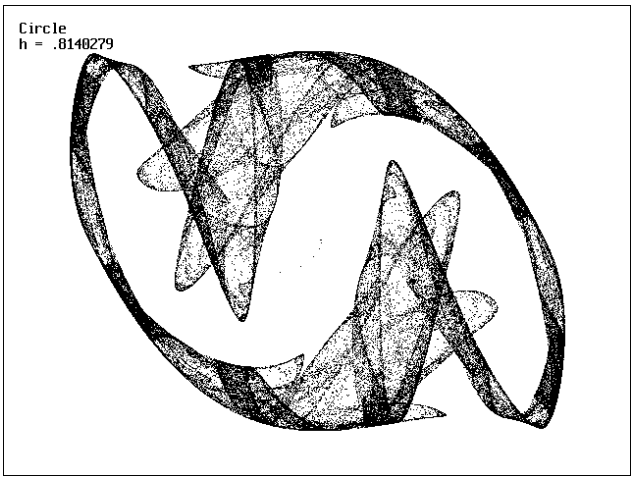

Figure 7.3: Chaotic solution from using Euler's method to solve the ODEs (7.19), (7.20).

whose phase-space solutions consist in a circular trajectories centred at $x = y = 0$. The trajectory should exactly close on itself after one period, independent of the initial condition. When solved with the Euler method, the trajectory does not close on itself for any h but spirals outward. This is because each iteration advances the trajectory along a tangent to the circle at the position of the previous iteration. A typical trajectory is shown in Fig. 7.4.

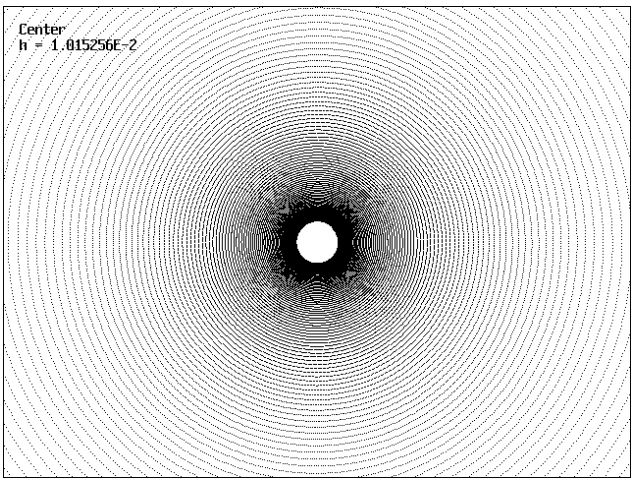

Figure 7.4: Unbounded solution obtained using Euler's method to solve the harmonic-oscillator equations (7.21), (7.22).

Although these examples suggest that the Euler method is seriously flawed, most other numerical methods also exhibit the same phenomena if a sufficiently large step size is taken.

7.2 Runge–Kutta methods

As shown in previous sections, there are two main reasons why Euler's method is not generally used in scientific computing. Firstly, the truncation error per step associated with this method is far larger than those associated with other, more advanced, methods (for a given value of h). Secondly, Euler's method is too prone to numerical instabilities.

The main reason that Euler's method has such a large truncation error per step is that when evolving the solution from x_n to x_{n+1}, the method only evaluates derivatives at the beginning of the interval: i.e., at x_n. The method is, therefore, very asymmetric with respect to the beginning and the end of the interval.

We can construct a more symmetric integration method by making an Euler-like trial step to the midpoint of the interval, and then using the values of both x and y at the midpoint to make the real step across the interval. To be more exact,

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= h\, f\left(x_n+\frac{h}{2}, y_n+\frac{k_1}{2}\right), \\ y_{n+1} &= y_n + k_2 + O(h^3). \end{aligned} \tag{7.23}$$

As indicated in the error term, this symmetrization cancels out the first-order error, making the method second-order. In fact, the above method is generally known as a second-order Runge–Kutta method. Euler's method can be thought of as a first-order Runge–Kutta method.

Of course, there is no need to stop at second order. By using two trial steps per interval, it is possible to cancel out both the first- and second-order error terms and thereby construct a third-order Runge–Kutta method. Likewise, three trial steps per interval yield a fourth-order method.

The general expression for the total error, ϵ, associated with integrating our ODE over an x-interval of order unity using an n-th order Runge–Kutta method is approximately

$$\epsilon \sim \frac{\eta}{h} + h^n. \tag{7.24}$$

As before, the first term corresponds to round-off error, whereas the second term represents truncation error. The minimum practical step-length, h_0, and the minimum error, ϵ_0, take the values

$$h_0 \sim \eta^{1/(n+1)}, \tag{7.25}$$

$$\epsilon_0 \sim \eta^{n/(n+1)}, \tag{7.26}$$

respectively. In Table 7.1, these values are tabulated against n using $\eta = 2.22 \times 10^{-16}$ (the value appropriate to double precision). It can be seen that h_0 increases and ϵ_0 decreases as

Table 7.1: The minimum practical step-length h_0 and minimum error ϵ_0 for an n-th order Runge–Kutta method using double precision arithmetic.

n	h_0	ϵ_0
1	1.5×10^{-8}	1.5×10^{-8}
2	6.1×10^{-6}	3.7×10^{-11}
3	1.2×10^{-4}	1.8×10^{-12}
4	7.4×10^{-4}	3.0×10^{-13}
5	2.4×10^{-3}	9.0×10^{-14}

n gets larger. However, the relative change in these quantities becomes progressively less dramatic as n increases.

In the majority of cases, the limiting factor when numerically integrating an ODE is not round-off error, but rather the computational effort involved in calculating the function $f(x, y)$.

Note that it is not normally beneficial to use Runge–Kutta schemes of *very* high order, because Eq. (7.26) shows that the achievable precision ϵ_0 does not increase much beyond, say, $n = 4$, while the memory-requirements of the program and the importance of round-off errors increases with increasing order n.

Although there is no hard and fast general rule, in most problems encountered in computational physics, a good choice is $n = 4$. In other words, in most situations of interest a fourth-order Runge Kutta integration method is an excellent choice.

The standard fourth-order Runge–Kutta method takes the form:

$$\begin{aligned} k_1 &= h\, f(x_n, y_n), \\ k_2 &= h\, f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right), \\ k_3 &= h\, f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right), \\ k_4 &= h\, f(x_n + h, y_n + k_3), \\ y_{n+1} &= y_n + \frac{k_1 + 2\,k_2 + 2\,k_3 + k_4}{6} + O(h^5). \end{aligned} \tag{7.27}$$

Let us briefly discuss the components in the algorithm above.

- k_1 is Euler's prediction for what we've previously called the vertical jump from the current point to the next Euler-predicted point along the numerical solution.
- k_2 we have never seen before. Notice the x value x_n+$h/2$ at which it is evaluating the function f. It lies halfway across the prediction interval. What about the y-value

that is coupled with it namely, $y_n+k_1/2$? It is the current y-value plus half of the Euler-predicted jump that we just discussed as being the meaning of k_1. So this too is a halfway value, this time vertically halfway up from the current point to the Euler-predicted next point. To summarize, then, the function f is being evaluated at a point that lies halfway between the current point and the Euler-predicted next point. Recalling that the function f gives us the slope of the solution curve, we can see that evaluating it at the halfway point just described, i.e. $f(x_n+h/2, y_n+k_1/2)$, gives us an estimate of the slope of the solution curve at this halfway point. Multiplying this slope by h, just as with the Euler method before, produces a prediction of the y-jump made by the actual solution across the whole width of the interval, only this time the predicted jump is not based on the slope of the solution at the left end of the interval, but on the estimated slope halfway to the Euler-predicted next point.

- k_3 has a formula which is quite similar to that of k_2, except that where k_1 used to be is now a k_2. Essentially the f-value here is yet another estimate of the slope of the solution at the "midpoint" of the prediction interval. This time, however, the y-value of the midpoint is not based on Euler's prediction, but on the y-jump predicted already with k_2. Once again, this slope-estimate is multiplied by h, giving us yet another estimate of the y-jump made by the actual solution across the whole width of the interval.

- k_4 evaluates f at x_n+h , which is the end-point of the interval. The corresponding y-value, y_n+k_3, is an estimate of the y-value at the right end of the interval, based on the y-jump just predicted by k_3. The f-value thus found is once again multiplied by h, just as with the three previous k_i, giving us a final estimate of the y-jump made by the actual solution across the whole width of the interval.

In summary, each of the k_i gives us an estimate of the size of the y-jump made by the actual solution across the whole width of the interval. The first one uses Euler's method, the next two use estimates of the slope of the solution at the midpoint, and the last one uses an estimate of the slope at the right end-point. Each k_i uses the earlier k_i as a basis for its prediction of the y-jump.

The fourth-order Runge–Kutta scheme (7.27) is probably the most common method to integrate first-order ODEs The generalization of this method to deal with systems of coupled first-order ODEs is almost completely straight-forward and we will apply it in Section 8.4.

7.2.1 Appendix

ODEs in Fortran 95

You will find a lot of information about ODEs in Fortran 95 in [NR90] (`http://www.nrbook.com/a/bookf90pdf.php` – scroll down to section B16).

ODEs in Mathematica

You will find a lot of information about ODEs in Mathematica at `http://mathworld.wolfram.com/OrdinaryDifferentialEquation.html`.

Lab exercise: Euler vs. Runge–Kutta

You will write a Fortran code to compare Euler's and the 4th-order Runge–Kutta method and generate Figures 7.5 and 7.6 in order to asses the performance of these two methods when applied to the following system of ODEs:

$$\frac{dx}{dt} = v \,, \tag{7.28}$$

$$\frac{dv}{dt} = -\omega^2 x \,, \tag{7.29}$$

subject to the initial conditions $x(0) = 0$ and $v(0) = \omega$ at $t = 0$. In fact, this system can be solved analytically to give

$$x = \sin \omega t \,. \tag{7.30}$$

Single precision

All calculations are first performed to single precision: i.e., by using float, rather than double, variables.

(a) Solve the ODEs Eqs. (7.28), (7.29) using Euler's and 4th-order Runge-Kutta method. Compare your results by plotting in the same graph the analytical solution given in Eq. (7.30) and your solutions.

(b) Reproduce Figure 7.5.

Let us compare the above solution with that obtained numerically using either Euler's method or the standard fourth-order Runge–Kutta method. Figure 7.5 shows the integration errors associated with these two methods (calculated by integrating the above system, with $k = 1$, from $t = 0$ to $t = 10$, and then taking the difference between the numerical and analytic solutions) plotted against the inverse step-size h^{-1} in a log-log graph.

It can be seen that at large values of h, the error associated with Euler's method becomes much greater than unity (i.e., the magnitude of the numerical solution greatly exceeds that of the analytic solution), indicating the presence of numerical instability. There are no similar signs of instability associated with the Runge–Kutta method for $h \leq 1$. At intermediate h, the error associated with Euler's method decreases smoothly like h^{-1}: in this regime, the dominant error is truncation error, which is expected to scale like h^{-1} for a first-order method. The error associated with the Runge–Kutta method similarly scales like h^{-4} – as expected for a fourth-order scheme – in the truncation error dominated regime. Note that, as h is decreased, the error associated with both methods eventually starts to rise in a jagged curve that scales roughly like h^1. This is a manifestation of round-off error. The minimum error associated with both methods corresponds to the boundary between

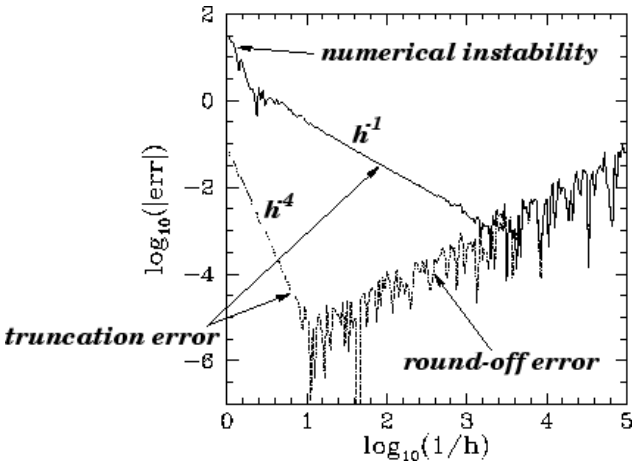

Figure 7.5: Global integration errors associated with Euler's method (solid curve) and fourth-order Runge–Kutta method (dotted curve) plotted against the step-size h. Single-precision calculation.

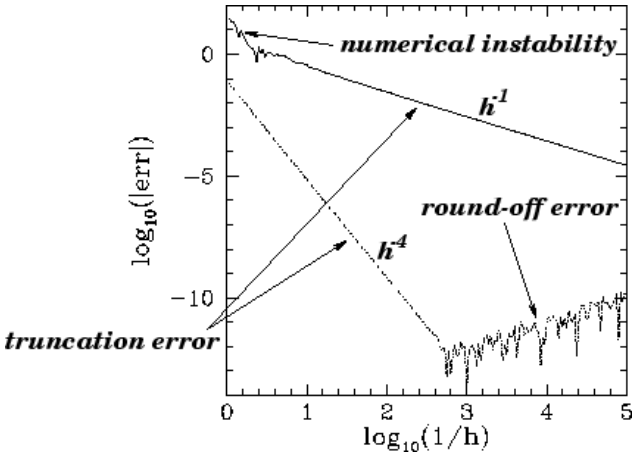

Figure 7.6: Global integration errors associated with Euler's method (solid curve) and fourth-order Runge–Kutta method (dotted curve) plotted against the step-size h. Double-precision calculation.

the truncation error and round-off error dominated regimes. Thus, for Euler's method the minimum error is about 10^{-3} at $h \sim 10^{-3}$, whereas for the Runge–Kutta method the minimum error is about 10^{-5} at $h \sim 10^{-1}$. Clearly, the performance of the Runge–Kutta method is vastly superior to that of Euler's method, since the former method is capable of attaining much greater accuracy than the latter using a far smaller number of steps (i.e., a far larger h).

Double precision

Redo your calculation using double-precision and generate a figure that is similar to Fig. 7.6.

Figure 7.6 displays similar data to that shown in Fig. 7.5, except that now all of the calculations are performed to double precision. The figure exhibits the same broad features as those apparent in Fig. 7.5. The major difference is that the round-off error has been

reduced by about nine orders of magnitude, allowing the Runge–Kutta method to attain a minimum error of about 10^{-12} (see Table 7.1) – a remarkably performance!

Figures 7.5 and 7.6 illustrate why scientists rarely use Euler's method, or single precision numerics, to integrate systems of ODEs.