Chapter 2

Ordinary Differential Equations

Many of the laws of physics are most conveniently formulated in terms of differential equations. It is therefore not surprising that the numerical solution of differential equations is one of the most common tasks in modeling physical systems. The most general form of an ordinary differential equation is a set of M coupled first-order equations

$$\frac{d\mathbf{y}}{dx} = \mathbf{f}(x,\mathbf{y}) \,, \tag{2.1}$$

where x is the independent variable and $\mathbf{y}$ is a set of M dependent variables ($\mathbf{f}$ is thus an M-component vector). Differential equations of higher order can be written in this first-order form by introducing auxiliary functions. For example, the one-dimensional motion of a particle of mass m under a force field $F(z)$ is described by the second-order equation

$$m\frac{d^2z}{dt^2} = F(z) \,. \tag{2.2}$$

If we define the momentum

$$p(t) = m\frac{dz}{dt} \,,$$

then (2.2) becomes the two coupled first-order (Hamilton's) equations

$$\frac{dz}{dt} = \frac{p}{m} \;\; ; \;\; \frac{dp}{dt} = F(z) \,, \tag{2.3}$$

which are in the form of (2.1). It is therefore sufficient to consider in detail only methods for first-order equations. Since the matrix structure

of coupled differential equations is of the most natural form, our discussion of the case where there is only one independent variable can be generalized readily. Thus, we need be concerned only with solving

$$\frac{dy}{dx} = f(x, y) \tag{2.4}$$

for a single dependent variable $y(x)$.

In this chapter, we will discuss several methods for solving ordinary differential equations, with emphasis on the initial value problem. That is, find $y(x)$ given the value of y at some initial point, say $y(x = 0) = y_0$. This kind of problem occurs, for example, when we are given the initial position and momentum of a particle and we wish to find its subsequent motion using Eqs. (2.3). In Chapter 3, we will discuss the equally important boundary value and eigenvalue problems.

2.1 Simple methods

To repeat the basic problem, we are interested in the solution of the differential equation (2.4) with the initial condition $y(x = 0) = y_0$. More specifically, we are usually interested in the value of y at a particular value of x, say $x = 1$. The general strategy is to divide the interval $[0, 1]$ into a large number, N, of equally spaced subintervals of length $h = 1/N$ and then to develop a recursion formula relating y_n to $\{y_{n-1}, y_{n-2}, \ldots\}$, where y_n is our approximation to $y(x_n = nh)$. Such a recursion relation will then allow a step-by-step integration of the differential equation from $x = 0$ to $x = 1$.

One of the simplest algorithms is Euler's method, in which we consider Eq. (2.4) at the point x_n and replace the derivative on the left-hand side by its forward difference approximation (1.4a). Thus,

$$\frac{y_{n+1} - y_n}{h} + \mathcal{O}(h) = f(x_n, y_n)\,, \tag{2.5}$$

so that the recursion relation expressing y_{n+1} in terms of y_n is

$$y_{n+1} = y_n + hf(x_n, y_n) + \mathcal{O}(h^2)\,. \tag{2.6}$$

This formula has a local error (that made in taking the single step from y_n to y_{n+1}) that is $\mathcal{O}(h^2)$ since the error in (1.4a) is $\mathcal{O}(h)$. The "global" error made in finding $y(1)$ by taking N such steps in integrating from $x = 0$ to $x = 1$ is then $N\mathcal{O}(h^2) \approx \mathcal{O}(h)$. This error decreases only

linearly with decreasing step size so that half as large an h (and thus twice as many steps) is required to halve the inaccuracy in the final answer. The numerical work for each of these steps is essentially a single evaluation of f.

As an example, consider the differential equation and boundary condition

$$\frac{dy}{dx} = -xy\ ;\ y(0) = 1\ , \tag{2.7}$$

whose solution is

$$y = e^{-x^2/2}\ .$$

The following FORTRAN program integrates forward from $x = 0$ to $x = 3$ using Eq. (2.6) with the step size input, printing the result and its error as it goes along.

```
C chap2a.for
      FUNC(X,Y)=-X*Y                !dy/dx
20    PRINT *, ' Enter step size ( .le. 0 to stop)'
      READ *, H
      IF (H .LE. 0.) STOP
      NSTEP=3./H                    !number of steps to reach X=3
      Y=1.                          !y(0)=1
      DO 10 IX=0,NSTEP-1            !loop over steps
         X=IX*H                     !last X value
         Y=Y+H*FUNC(X,Y)            !new Y value from Eq 2.6
         DIFF=EXP(-0.5*(X+H)**2)-Y !compare with exact value
         PRINT *, IX,X+H,Y,DIFF
10    CONTINUE
      GOTO 20                       !start again with new value of H
      END
```

Errors in the results obtained for

$$y(1) = e^{-1/2} = 0.606531\ ,\ \ y(3) = e^{-9/2} = 0.011109$$

with various step sizes are shown in the first two columns of Table 2.1. As expected from (2.6), the errors decrease linearly with smaller h. However, the fractional error (error divided by y) increases with x as more steps are taken in the integration and y becomes smaller.

Table 2.1 Error in integrating $dy/dx = -xy$ with $y(0) = 1$

	Euler's method Eq. (2.6)		Taylor series Eq. (2.10)		Implicit method Eq. (2.18)	
h	$y(1)$	$y(3)$	$y(1)$	$y(3)$	$y(1)$	$y(3)$
0.500	−.143469	.011109	.032312	−.006660	−.015691	.001785
0.200	−.046330	.006519	.005126	−.000712	−.002525	.000255
0.100	−.021625	.003318	.001273	−.000149	−.000631	.000063
0.050	−.010453	.001665	.000317	−.000034	−.000157	.000016
0.020	−.004098	.000666	.000051	−.000005	−.000025	.000003
0.010	−.002035	.000333	.000013	−.000001	−.000006	.000001
0.005	−.001014	.000167	.000003	.000000	−.000001	.000000
0.002	−.000405	.000067	.000001	.000000	.000000	.000000
0.001	−.000203	.000033	.000000	.000000	.000000	.000000

■ **Exercise 2.1** A simple and often stringent test of an accurate numerical integration is to use the final value of y obtained as the initial condition to integrate backward from the final value of x to the starting point. The extent to which the resulting value of y differs from the original initial condition is then a measure of the inaccuracy. Apply this test to the example above.

Although Euler's method seems to work quite well, it is generally unsatisfactory because of its low-order accuracy. This prevents us from reducing the numerical work by using a larger value of h and so taking a smaller number of steps. This deficiency becomes apparent in the example above as we attempt to integrate to larger values of x, where some thought shows that we obtain the absurd result that $y = 0$ (exactly) for $x > h^{-1}$. One simple solution is to change the step size as we go along, making h smaller as x becomes larger, but this soon becomes quite inefficient.

Integration methods with a higher-order accuracy are usually preferable to Euler's method. They offer a much more rapid increase of accuracy with decreasing step size and hence greater accuracy for a fixed amount of numerical effort. One class of simple higher order methods can be derived from a Taylor series expansion for y_{n+1} about y_n:

$$y_{n+1} = y(x_n + h) = y_n + hy'_n + \frac{1}{2}h^2 y''_n + \mathcal{O}(h^3)\,. \tag{2.8}$$

From (2.4), we have

$$y'_n = f(x_n, y_n)\,, \tag{2.9a}$$

and

$$y_n'' = \frac{df}{dx}(x_n, y_n) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} f \,, \tag{2.9b}$$

which, when substituted into (2.8), results in

$$y_{n+1} = y_n + hf + \frac{1}{2}h^2\left[\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial y}\right] + \mathcal{O}(h^3) \,, \tag{2.10}$$

where f and its derivatives are to be evaluated at (x_n, y_n). This recursion relation has a local error $\mathcal{O}(h^3)$ and hence a global error $\mathcal{O}(h^2)$, one order more accurate than Euler's method (2.6). It is most useful when f is known analytically and is simple enough to differentiate. If we apply Eq. (2.10) to the example (2.7), we obtain the results shown in the middle two columns of Table 2.1; the improvement over Euler's method is clear. Algorithms with an even greater accuracy can be obtained by retaining more terms in the Taylor expansion (2.8), but the algebra soon becomes prohibitive in all but the simplest cases.

2.2 Multistep and implicit methods

Another way of achieving higher accuracy is to use recursion relations that relate y_{n+1} not just to y_n, but also to points further "in the past," say y_{n-1}, y_{n-2}, ... To derive such formulas, we can integrate one step of the differential equation (2.4) *exactly* to obtain

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y)\, dx \,. \tag{2.11}$$

The problem, of course, is that we don't know f over the interval of integration. However, we can use the values of y at x_n and x_{n-1} to provide a linear extrapolation of f over the required interval:

$$f \approx \frac{(x - x_{n-1})}{h} f_n - \frac{(x - x_n)}{h} f_{n-1} + \mathcal{O}(h^2) \,, \tag{2.12}$$

where $f_i \equiv f(x_i, y_i)$. Inserting this into (2.11) and doing the x integral then results in the Adams-Bashforth two-step method,

$$y_{n+1} = y_n + h\left(\frac{3}{2} f_n - \frac{1}{2} f_{n-1}\right) + \mathcal{O}(h^3) \,. \tag{2.13}$$

Related higher-order methods can be derived by extrapolating with higher-degree polynomials. For example, if f is extrapolated by a cubic polynomial fitted to f_n, f_{n-1}, f_{n-2}, and f_{n-3}, the Adams-Bashforth four-step method results:

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) + \mathcal{O}(h^5)\,. \qquad (2.14)$$

Note that because the recursion relations (2.13) and (2.14) involve several previous steps, the value of y_0 alone is not sufficient information to get them started, and so the values of y at the first few lattice points must be obtained from some other procedure, such as the Taylor series (2.8) or the Runge-Kutta methods discussed below.

■ **Exercise 2.2** Apply the Adams-Bashforth two- and four-step algorithms to the example defined by Eq. (2.7) using Euler's method (2.6) to generate the values of y needed to start the recursion relation. Investigate the accuracy of $y(x)$ for various values of h by comparing with the analytical results and by applying the reversibility test described in Exercise 2.1.

The methods we have discussed so far are all "explicit" in that the y_{n+1} is given directly in terms of the already known value of y_n. "Implicit" methods, in which an equation must be solved to determine y_{n+1}, offer yet another means of achieving higher accuracy. Suppose we consider Eq. (2.4) at a point $x_{n+1/2} \equiv (n + \frac{1}{2})h$ mid-way between two lattice points:

$$\left.\frac{dy}{dx}\right|_{x_{n+1/2}} = f(x_{n+1/2}\,,\, y_{n+1/2})\,. \qquad (2.15)$$

If we then use the symmetric difference approximation for the derivative (the analog of (1.3b) with $h \to \frac{1}{2}h$) and replace $f_{n+1/2}$ by the average of its values at the two adjacent lattice points [the error in this replacement is $\mathcal{O}(h^2)$], we can write

$$\frac{y_{n+1} - y_n}{h} + \mathcal{O}(h^2) = \frac{1}{2}[f_n + f_{n+1}] + \mathcal{O}(h^2)\,, \qquad (2.16)$$

which corresponds to the recursion relation

$$y_{n+1} = y_n + \frac{1}{2}h[f(x_n, y_n) + f(x_{n+1}, y_{n+1})] + \mathcal{O}(h^3)\,. \qquad (2.17)$$

This is all well and good, but the appearance of y_{n+1} on both sides of this equation (an implicit equation) means that, in general, we must solve a non-trivial equation (for example, by the Newton-Raphson method discussed in Section 1.3) at each integration step; this can be very time consuming. A particular simplification occurs if f is linear in y, say $f(x,y) = g(x)y$, in which case (2.17) can be solved to give

$$y_{n+1} = \left[\frac{1+\frac{1}{2}g(x_n)h}{1-\frac{1}{2}g(x_{n+1})h}\right] y_n \,. \tag{2.18}$$

When applied to the problem (2.7), where $g(x) = -x$, this method gives the results shown in the last two columns of Table 2.1; the quadratic behavior of the error with h is clear.

■ **Exercise 2.3** Apply the Taylor series method (2.10) and the implicit method (2.18) to the example of Eq. (2.7) and obtain the results shown in Table 2.1. Investigate the accuracy of integration to larger values of x.

The Adams-Moulton methods are both multistep and implicit. For example, the Adams-Moulton two-step method can be derived from Eq. (2.11) by using a quadratic polynomial passing through f_{n-1}, f_n, and f_{n+1},

$$f \approx \frac{(x-x_n)(x-x_{n-1})}{2h^2} f_{n+1} - \frac{(x-x_{n+1})(x-x_{n-1})}{h^2} f_n + \frac{(x-x_{n+1})(x-x_n)}{2h^2} f_{n-1} + \mathcal{O}(h^3) \,,$$

to interpolate f over the region from x_n to x_{n+1}. The implicit recursion relation that results is

$$y_{n+1} = y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}) + \mathcal{O}(h^4) \,. \tag{2.19}$$

The corresponding three-step formula, obtained with a cubic polynomial interpolation, is

$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}) + \mathcal{O}(h^5) \,. \tag{2.20}$$

Implicit methods are rarely used by solving the implicit equation to take a step. Rather, they serve as bases for "predictor-corrector" algorithms, in which a "prediction" for y_{n+1} based only on an explicit method

is then "corrected" to give a better value by using this prediction in an implicit method. Such algorithms have the advantage of allowing a continuous monitoring of the accuracy of the integration, for example by making sure that the correction is small. A commonly used predictor-corrector algorithm with local error $\mathcal{O}(h^5)$ is obtained by using the explicit Adams-Bashforth four-step method (2.14) to make the prediction, and then calculating the correction with the Adams-Moulton three-step method (2.20), using the predicted value of y_{n+1} to evaluate f_{n+1} on the right-hand side.

2.3 Runge-Kutta methods

As you might gather from the preceding section, there is quite a bit of freedom in writing down algorithms for integrating differential equations and, in fact, a large number of them exist, each having it own peculiarities and advantages. One very convenient and widely used class of methods are the Runge-Kutta algorithms, which come in varying orders of accuracy. We derive here a second-order version to give the spirit of the approach and then simply state the equations for the third- and commonly used fourth-order methods.

To derive a second-order Runge-Kutta algorithm (there are actually a whole family of them characterized by a continuous parameter), we approximate f in the integral of (2.11) by its Taylor series expansion about the mid-point of the integration interval. Thus,

$$y_{n+1} = y_n + hf(x_{n+1/2}\ ,\ y_{n+1/2}) + \mathcal{O}(h^3)\ , \tag{2.21}$$

where the error arises from the quadratic term in the Taylor series, as the linear term integrates to zero. Although it seems as if we need to know the value of $y_{n+1/2}$ appearing in f in the right-hand side of this equation for it to be of any use, this is not quite true. Since the error term is already $\mathcal{O}(h^3)$, an approximation to y_{n+1} whose error is $\mathcal{O}(h^2)$ is good enough. This is just what is provided by the simple Euler's method, Eq. (2.6). Thus, if we define k to be an intermediate approximation to twice the difference between $y_{n+1/2}$ and y_n, the following two-step procedure gives y_{n+1} in terms of y_n:

$$k = hf(x_n, y_n)\ ; \tag{2.22a}$$

$$y_{n+1} = y_n + hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k) + \mathcal{O}(h^3)\ . \tag{2.22b}$$

This is a second-order Runge-Kutta algorithm. It embodies the general idea of substituting approximations for the values of y into the right-hand

side of implicit expressions involving f. It is as accurate as the Taylor series or implicit methods (2.10) or (2.17), respectively, but places no special constraints on f, such as easy differentiability or linearity in y. It also uses the value of y at only one previous point, in contrast to the multipoint methods discussed above. However, (2.22) does require the evaluation of f twice for each step along the lattice.

Runge-Kutta schemes of higher-order can be derived in a relatively straightforward way. Any of the quadrature formulas discussed in Chapter 1 can be used to approximate the integral (2.11) by a finite sum of f values. For example, Simpson's rule yields

$$y_{n+1} = y_n + \frac{h}{6}[f(x_n, y_n) + 4f(x_{n+1/2}\ ,\ y_{n+1/2}) + f(x_{n+1}\ ,\ y_{n+1})] + \mathcal{O}(h^5)\ . \tag{2.23}$$

Schemes for generating successive approximations to the y's appearing in the right-hand side of a commensurate accuracy then complete the algorithms. A third-order algorithm with a local error $\mathcal{O}(h^4)$ is

$$\begin{aligned} k_1 &= hf(x_n, y_n)\ ; \\ k_2 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1)\ ; \\ k_3 &= hf(x_n + h, y_n - k_1 + 2k_2)\ ; \\ y_{n+1} &= y_n + \tfrac{1}{6}(k_1 + 4k_2 + k_3) + \mathcal{O}(h^4)\ . \end{aligned} \tag{2.24}$$

It is based on (2.23) and requires three evaluations of f per step. A fourth-order algorithm, which requires f to be evaluated four times for each integration step and has a local accuracy of $\mathcal{O}(h^5)$, has been found by experience to give the best balance between accuracy and computational effort. It can be written as follows, with the k_i as intermediate variables:

$$\begin{aligned} k_1 &= hf(x_n, y_n)\ ; \\ k_2 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1)\ ; \\ k_3 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2)\ ; \\ k_4 &= hf(x_n + h, y_n + k_3)\ ; \\ y_{n+1} &= y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5)\ . \end{aligned} \tag{2.25}$$

■ **Exercise 2.4** Try out the second-, third-, and fourth-order Runge-Kutta methods discussed above on the problem defined by Eq. (2.7). Compare the computational effort for a given accuracy with that of other methods.

■ **Exercise 2.5** The two coupled first-order equations

$$\frac{dy}{dt} = p\,; \quad \frac{dp}{dt} = -4\pi^2 y \tag{2.26}$$

define simple harmonic motion with period 1. By generalizing one of the single-variable formulas given above to this two-variable case, integrate these equations with any particular initial conditions you choose and investigate the accuracy with which the system returns to its initial state at integral values of t.

2.4 Stability

A major consideration in integrating differential equations is the numerical stability of the algorithm used; i.e., the extent to which round-off or other errors in the numerical computation can be amplified, in many cases enough for this "noise" to dominate the results. To illustrate the problem, let us attempt to improve the accuracy of Euler's method and approximate the derivative in (2.4) directly by the symmetric difference approximation (1.3b). We thereby obtain the three-term recursion relation

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n) + \mathcal{O}(h^3)\,, \tag{2.27}$$

which superficially looks about as useful as either of the third-order formulas (2.10) or (2.18). However, consider what happens when this method is applied to the problem

$$\frac{dy}{dx} = -y\,; \quad y(x=0) = 1\,, \tag{2.28}$$

whose solution is $y = e^{-x}$. To start the recursion relation (2.27), we need the value of y_1 as well as $y_0 = 1$. This can be obtained by using (2.10) to get

$$y_1 = 1 - h + \frac{1}{2}h^2 + \mathcal{O}(h^3)\,.$$

(This is just the Taylor series for e^{-h}.) The following FORTRAN program then uses the method (2.27) to find y for values of x up to 6 using the value of h input:

Table 2.2 Integration of $dy/dx = -y$ with $y(0) = 1$

x	Exact	Error	x	Exact	Error	x	Exact	Error
0.2	.818731	−.000269	3.3	.036883	−.000369	5.5	.004087	−.001533
0.3	.740818	−.000382	3.4	.033373	−.000005	5.6	.003698	.001618
0.4	.670320	−.000440	3.5	.030197	−.000380	5.7	.003346	−.001858
0.5	.606531	−.000517	3.6	.027324	.000061	5.8	.003028	.001989
0.6	.548812	−.000538	3.7	.024724	−.000400	5.9	.002739	−.002257
			3.8	.022371	.000133	6.0	.002479	.002439

```
C chap2b.for
10      PRINT *,' Enter value of step size ( .le. 0 to stop)'
        READ *, H
        IF (H .LE. 0) STOP
        YMINUS=1                          !Y(0)
        YZERO=1.-H+H**2/2                 !Y(H)
        NSTEP=6./H
        DO 20 IX=2,NSTEP                  !loop over X values
           X=IX*H                         !X at this step
           YPLUS=YMINUS-2*H*YZERO         !Y from Eq. 2.27
           YMINUS=YZERO                   !roll values
           YZERO=YPLUS
           EXACT=EXP(-X)                  !analytic value at this point
           PRINT *, X,EXACT,EXACT-YZERO
20      CONTINUE
        GOTO 10                           !get another H value
        END
```

Note how the three-term recursion is implemented by keeping track of only three local variables, YPLUS, YZERO, and YMINUS.

A portion of the output of this code run for $h = 0.1$ is shown in Table 2.2. For small values of x, the numerical solution is only slightly larger than the exact value, the error being consistent with the $\mathcal{O}(h^3)$ estimate. Then, near $x = 3.5$, an oscillation begins to develop in the numerical solution, which becomes alternately higher and lower than the exact values lattice point by lattice point. This oscillation grows larger as the equation is integrated further (see values near $x = 6$), eventually overwhelming the exponentially decreasing behavior expected.

The phenomenon observed above is a symptom of an instability in the algorithm (2.27). It can be understood as follows. For the problem

(2.28), the recursion relation (2.27) reads

$$y_{n+1} = y_{n-1} - 2hy_n \,. \tag{2.29}$$

We can solve this equation by assuming an exponential solution of the form $y_n = Ar^n$ where A and r are constants. Substituting into (2.29) then results in an equation for r,

$$r^2 + 2hr - 1 = 0 \,,$$

the constant A being unimportant since the recursion relation is linear. The solutions of this equation are

$$r_+ = (1+h^2)^{1/2} - h \approx 1 - h \,;\; r_- = -(1+h^2)^{1/2} - h \approx -(1+h) \,,$$

where we have indicated approximations valid for $h \ll 1$. The positive root is slightly less than one and corresponds to the exponentially decreasing solution we are after. However, the negative root is slightly less than -1, and so corresponds to a spurious solution

$$y_n \sim (-)^n(1+h)^n \,,$$

whose magnitude increases with n and which oscillates from lattice point to lattice point.

The general solution to the linear difference equation (2.27) is a linear combination of these two exponential solutions. Even though we might carefully arrange the initial values y_0 and y_1 so that only the decreasing solution is present for small x, numerical round-off during the recursion relation [Eq. (2.29) shows that two positive quantities are subtracted to obtain a smaller one] will introduce a small admixture of the "bad" solution that will eventually grow to dominate the results. This instability is clearly associated with the three-term nature of the recursion relation (2.29). A good rule of thumb is that instabilities and round-off problems should be watched for whenever integrating a solution that decreases strongly as the iteration proceeds; such a situation should therefore be avoided, if possible. We will see the same sort of instability phenomenon again in our discussion of second-order differential equations in Chapter 3.

■ **Exercise 2.6** Investigate the stability of several other integration methods discussed in this chapter by applying them to the problem (2.28). Can you give analytical arguments to explain the results you obtain?

(2.28), the recursion relation (2.27) reads

$$y_{n+1} = y_{n-1} - 2hy_n \,. \tag{2.29}$$

We can solve this equation by assuming an exponential solution of the form $y_n = Ar^n$ where A and r are constants. Substituting into (2.29) then results in an equation for r,

$$r^2 + 2hr - 1 = 0 \,,$$

the constant A being unimportant since the recursion relation is linear. The solutions of this equation are

$$r_+ = (1+h^2)^{1/2} - h \approx 1 - h \,;\; r_- = -(1+h^2)^{1/2} - h \approx -(1+h) \,,$$

where we have indicated approximations valid for $h \ll 1$. The positive root is slightly less than one and corresponds to the exponentially decreasing solution we are after. However, the negative root is slightly less than -1, and so corresponds to a spurious solution

$$y_n \sim (-)^n (1+h)^n \,,$$

whose magnitude increases with n and which oscillates from lattice point to lattice point.

The general solution to the linear difference equation (2.27) is a linear combination of these two exponential solutions. Even though we might carefully arrange the initial values y_0 and y_1 so that only the decreasing solution is present for small x, numerical round-off during the recursion relation [Eq. (2.29) shows that two positive quantities are subtracted to obtain a smaller one] will introduce a small admixture of the "bad" solution that will eventually grow to dominate the results. This instability is clearly associated with the three-term nature of the recursion relation (2.29). A good rule of thumb is that instabilities and round-off problems should be watched for whenever integrating a solution that decreases strongly as the iteration proceeds; such a situation should therefore be avoided, if possible. We will see the same sort of instability phenomenon again in our discussion of second-order differential equations in Chapter 3.

■ **Exercise 2.6** Investigate the stability of several other integration methods discussed in this chapter by applying them to the problem (2.28). Can you give analytical arguments to explain the results you obtain?