Advanced Engineering Mathematics

SECOND EDITION

Michael D. Greenberg
Department of Mechanical Engineering
University of Delaware, Newark, Delaware

PEARSON

ISBN 978-81-7758-546-9

10 9 8 7

This edition is manufactured in India and is authorized for sale only in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives. Circulation of this edition outside of these territories is UNAUTHORIZED.

Published by Dorling Kindersley (India) Pvt. Ltd., licensees of Pearson Education in South Asia.

Head Office: 7th Floor, Knowledge Boulevard, A-8(A) Sector-62, Noida (U.P) 201309, India.
Registered Office: 11 Community Centre, Panchsheel Park, New Delhi 110 017, India.

Printed in India by Saurabh Printers Pvt. Ltd.

Chapter 6

Quantitative Methods: Numerical Solution of Differential Equations

6.1 Introduction

Following the introduction in Chapter 1, Chapters 2–5 cover both the underlying theory of differential equations and analytical solution techniques as well. That is, the objective thus far has been to find an analytical solution – in closed form if possible or as an infinite series if necessary. Unfortunately, a great many differential equations encountered in applications, and most nonlinear equations in particular, are simply too difficult for us to find analytical solutions.

Thus, in Chapters 6 and 7 our approach is fundamentally different, and complements the analytical approach adopted in Chapters 2–5: in Chapter 6 we develop *quantitative* methods, and in Chapter 7 our view is essentially *qualitative*. More specifically, in this chapter we "discretize" the problem and seek, instead of an analytical solution, the numerical values of the dependent variable at a discrete set of values of the independent variable so that the result is a table or graph, with those values determined approximately (but accurately), rather than exactly.

Perhaps the greatest drawback to numerical simulation is that whereas an analytical solution explicitly displays the dependence of the dependent variable(s) on the various physical parameters (such as spring stiffnesses, driving frequencies, electrical resistances, inductances, and so on), one can carry out a numerical solution only for a specific set of values of the system parameters. Thus, parametric studies (i.e., studies of the qualitative and quantitative effects of the various parameters upon the solution) can be tedious and unwieldy, and it is useful to reduce the number of parameters as much as possible (by nondimensionalization, as discussed in Section 2.4.4) before embarking upon a numerical study.

The numerical solution of differential equations covers considerable territory so the present chapter is hardly complete. Rather, we aim at introducing the funda-

mental ideas, concepts, and potential difficulties, as well as specific methods that are accurate and readily implemented. We do mention computer software that carries out these computations automatically, but our present aim is to provide enough information so that you will be able to select a specific method and program it. In contrast, in Chapter 7, where we look more at qualitative issues, we rely heavily upon available software.

6.2 Euler's Method

In this section and the two that follow, we study the numerical solution of the first-order initial-value problem

$$y' = f(x, y); \quad y(a) = b \tag{1}$$

on $y(x)$.

To motivate the first and simplest of these methods, Euler's method, consider the problem

$$y' = y + 2x - x^2; \quad y(0) = 1 \quad (0 \le x < \infty) \tag{2}$$

with the exact solution (Exercise 1)

$$y(x) = x^2 + e^x. \tag{3}$$

Of course, in practice one wouldn't solve (2) numerically because we can solve it analytically and obtain the solution (3), but we will use (2) as an illustration.

In Fig. 1 we display the direction field defined by $f(x, y) = y + 2x - x^2$, as well as the exact solution (3). In graphical terms, Euler's method amounts to using the direction field as a road map in developing an approximate solution to (2). Beginning at the initial point P, namely $(0, 1)$, we move in the direction dictated by the lineal element at that point. As seen from the figure, the farther we move along that line, the more we expect our path to deviate from the exact solution. Thus, the idea is not to move very far. Stopping at $x = 0.5$, say, for the sake of illustration, we revise our direction according to the slope of the lineal element at that point Q. Moving in that new direction until $x = 1$, we revise our direction at R, and so on, moving in x increments of 0.5. We call the x increment the **step size** and denote it as h. In Fig. 1, h is 0.5.

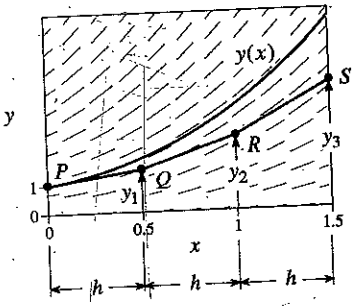


Figure 1. Direction field motivation of Euler's method, for the initial-value problem (2).

Let us denote the y values at $Q, R, \ldots$ as $y_1, y_2, \ldots$. They are computed as $y_1 = y_0 + f(x_0, y_0)h$, $y_2 = y_1 + f(x_1, y_1)h, \ldots$, where (x_0, y_0) is the initial point P. Expressed as a numerical algorithm, the **Euler method** is therefore as follows:

$$\boxed{y_{n+1} = y_n + f(x_n, y_n)h, \qquad (n = 0, 1, 2, \ldots)} \tag{4}$$

where f is the function on the right side of the given differential equation (1), $x_0 = a$, $y_0 = b$, h is the chosen step size, and $x_n = x_0 + nh$.

Euler's method is also known as the **tangent-line method** because the first straight-line segment of the approximate solution is tangent to the exact solution

$y(x)$ at P, and each subsequent segment emanating from (x_n, y_n) is tangent to the solution curve through that point.

Apparently, the greater the step size the less accurate the results, in general. For instance, the first point Q deviates more and more from the exact solution as the step size is increased – that is, as the segment PQ is extended. Conversely, we expect the approximate solution to approach the exact solution curve as h is reduced. This expectation is supported by the results shown in Table 1 for the initial-

Table 1. Comparison of numerical solution of (2) using Euler's method, with the exact solution.

x	$h = 0.5$	$h = 0.1$	$h = 0.02$	$y(x)$
0.5	1.5000	1.7995	1.8778	1.8987
1.0	2.6250	3.4344	3.6578	3.7183
1.5	4.4375	6.1095	6.5975	6.7317

value problem (2), obtained by Euler's method with step sizes of $h = 0.5, 0.1$, and 0.02; we have included the exact solution $y(x)$, given by (3), for comparison. With $h = 0.5$, for instance,

$$y_1 = y_0 + \left(y_0 + 2x_0 - x_0^2\right) h = 1 + (1 + 0 - 0)\,(0.5) = 1.5,$$
$$y_2 = y_1 + \left(y_1 + 2x_1 - x_1^2\right) h = 1.5 + (1.5 + 1 - 0.25)\,(0.5) = 2.625,$$
$$y_3 = y_2 + \left(y_2 + 2x_2 - x_2^2\right) h = 2.625 + (2.625 + 2 - 1)\,(0.5) = 4.4375.$$

With $h = 0.1$, the values tabulated at $x = 0.5, 1.0, 1.5$ are y_5, y_{10}, y_{15}, with the intermediate computed y values omitted for brevity.

Scanning each row of the tabulation, we can see that the approximate solution appears to be converging to the exact solution as $h \to 0$ (though we cannot be certain from such results no matter how small we make h), and that the convergence is not very rapid, for even with $h = 0.02$ the computed value at $x = 1.5$ is in error by 2%.

As strictly computational as this sounds, two important theoretical questions present themselves: Does the method converge to the exact solution as $h \to 0$ and, if so, how fast? By a method being **convergent** we mean that for any fixed x value in the x interval of interest the sequence of y values, obtained using smaller and smaller step size h, tends to the exact solution $y(x)$ as $h \to 0$.

Let us see whether the Euler method is convergent. Observe that there are two sources of error in the numerical solution. One is the tangent-line approximation upon which the method is based, and the other is the accumulation of numerical roundoff errors within the computing machine since a machine can carry only a finite number of significant figures, after which it rounds off (or chops off, depending upon the machine). In discussing convergence, one ignores the presence of such roundoff error and considers it separately. Thus, in this discussion we imagine our computer to be perfect, carrying an infinite number of significant figures.

Local truncation error. Although we are interested in the accumulation of error after a great many steps have been carried out, to reach any given x, it seems best to begin by investigating the error incurred in a single step, from x_{n-1} to x_n (or from x_n to x_{n+1}, it doesn't matter). We need to distinguish between the exact and approximate solutions so let us denote the exact solution at x_n as $y(x_n)$ and the approximate numerical solution at x_n as y_n. These are given by the Taylor series

$$\begin{aligned} y(x_n) &= y(x_{n-1}) + y'(x_{n-1})\,(x_n - x_{n-1}) + \frac{y''(x_{n-1})}{2!}(x_n - x_{n-1})^2 + \cdots \\ &= y(x_{n-1}) + y'(x_{n-1})h + \frac{y''(x_{n-1})}{2!}h^2 + \cdots \end{aligned} \tag{5}$$

and the Euler algorithm

$$y_n = y_{n-1} + f\,(x_{n-1}, y_{n-1})\,h, \tag{6}$$

respectively. It is important to understand that the Euler method (6) amounts to retaining only the first two terms of the Taylor series in (5). Thus, it replaces the actual function by its tangent-line approximation.

We suppose that $y(x_{n-1})$ and y_{n-1} are identical, and we ask how large the error $e_n \equiv y(x_n) - y_n$ is after making that single step, from x_{n-1} to x_n. We can get an expression for e_n by subtracting (6) from (5), but the right side will be an infinite series. Thus, it is more convenient to use, in place of the (infinite) Taylor series (5), the (finite) Taylor's formula with remainder,

$$y(x_n) = y(x_{n-1}) + y'(x_{n-1})h + \frac{y''(\xi)}{2!}h^2, \tag{7}$$

where ξ is some point in the interval $[x_{n-1}, x_n]$. Now, subtracting (6) from (7), and noting that $y'(x_{n-1}) = f\,[x_{n-1}, y(x_{n-1})] = f(x_{n-1}, y_{n-1})$ because of our supposition that $y(x_{n-1}) = y_{n-1}$, gives

$$e_n = \frac{y''(\xi)}{2}h^2. \tag{8}$$

The latter expression for e_n is awkward to apply since we don't know ξ, except that $x_{n-1} \le \xi \le x_n$.* However, (8) is of more interest in that it shows how the single-step error e_n varies with h. Specifically, since $x_{n-1} \le \xi \le x_{n-1} + h$, we see that as $h \to 0$ we have $\xi \to x_{n-1}$, so (8) gives $e_n \sim \dfrac{y''(x_{n-1})}{2}h^2 = Ch^2$ as $h \to 0$, where C is a constant. Accordingly, we say that e_n is of order h^2 and write

$$\boxed{e_n = O(h^2)} \tag{9}$$

*It also appears that we do not know the y'' function, but it follows from (2) that $y'' = y' + 2 - 2x = (y + 2x - x^2) + 2 - 2x = y + 2 - x^2$.

as $h \to 0$. [The big oh notation is defined in Section 4.5, and (9) simply means that $e_n \sim Ch^2$ as $h \to 0$ for some nonzero constant C.]

Since the error e_n is due to truncation of the Taylor series it is called the truncation error—more specifically, the **local truncation error** because it is the truncation error incurred in a single step.

Accumulated truncation error and convergence. Of ultimate interest, however, is the truncation error that has accumulated over *all* of the preceding steps since that error is the difference between the exact solution and the computed solution at any given x_n. We denote it as $E_n \equiv y(x_n) - y_n$ and call it the **accumulated truncation error**. If it seems strange that we have defined both the local and accumulated truncation errors as $y(x_n) - y_n$, it must be remembered that the former is that which results from a single step (from x_{n-1} to x_n) whereas the latter is that which results from the entire sequence of steps (from x_0 to x_n).

We can estimate E_n at a fixed x location (at least insofar as its order of magnitude) as the local truncation error e_n times the number of steps n. Since $e_n = O(h^2)$, this idea gives

$$E_n = O(h^2) \cdot n = O(h^2)\frac{n\,h}{h} = O(h^2)\frac{x_n - x_0}{h} = O(h) \cdot (x_n - x_0) = O(h). \quad (10)$$

The last step followed because the selected x_n location was held fixed as $h \to 0$. The big oh notation is insensitive to scale factors, so we absorbed the $x_n - x_0$ factor into the $O(h)$. Thus,

$$\boxed{E_n = O(h),} \quad (11)$$

which result tells us how fast the numerical solution converges to the exact solution (at any fixed x location) as $h \to 0$. Namely, $E_n \sim Ch$ for some nonzero constant C. To illustrate, consider the results shown in Table 1, and consider $x = 1.5$, say, in particular. According to $E_n \sim Ch$, if we reduce h by a factor of five, from 0.1 to 0.02, then likewise we should reduce the error by a factor of five. We find that $(6.7317 - 6.1095)/(6.7317 - 6.5975) \approx 4.6$, which is indeed close to five. We can't expect it to be exactly five for two reasons: First, (11) holds only as $h \to 0$, whereas we have used $h = 0.1$ and 0.02. Second, we obtained the values in Table 1 using a computer, and a computer introduces an additional error, due to roundoff, which has not been accounted for in our derivation of (11). Probably it is negligible in this example.

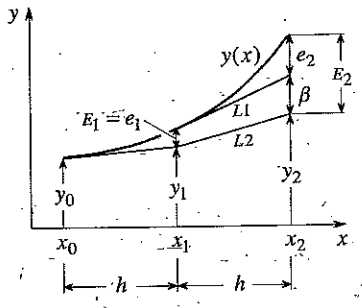


Figure 2. The global truncation error.

While (11) can indeed be proved rigorously, be aware that our reasoning in (10) was only heuristic. To understand the shortcoming of our logic, consider the diagram in Fig. 2, where we show only two steps, for simplicity.

Our reasoning, in writing $E_n = O(h^2) \cdot n$ in (10), is that the accumulated truncation error E_n is (at least insofar as order of magnitude) the sum of the n single-step errors. However, that is not quite true. We see from Fig. 2 that E_2 is $e_2 + \beta$, not the sum of the single-step errors $e_2 + e_1$, and β is not identical to e_1. The difference between β and e_1 is the result of the *slightly* different slopes of $L1$ and $L2$ acting over the *short* distance h, and that difference can be shown to be a higher-order effect that does not invalidate the final result that $E_n = O(h)$, provided that

$L2$ acting over the *short* distance h, and that difference can be shown to be a higher-order effect that does not invalidate the final result that $E_n = O(h)$, provided that f is well enough behaved (for example, if f, f_x, and f_y are all continuous on the x, y region of interest).

In summary, (11) shows that the Euler method (4) is convergent because the accumulated truncation error tends to zero as $h \to 0$. More generally if, for a given method, $E_n = O(h^p)$ as $h \to 0$, then the method is convergent if $p > 0$, and we say that it is **of order p**. Thus, *the Euler method is a first-order method.*

Although convergent and easy to implement, Euler's method is usually too inaccurate for serious computation because it is only a first-order method. That is, since the accumulated truncation error is proportional to h to the first power, we need to make h extremely small if the error is to be extremely small. Why can't we do that? Why can't we merely let $h = 10^{-8}$, say? There are two important reasons. One is that with $h = 10^{-8}$, it would take 10^8 steps to generate the Euler solution over a unit x interval. That number of steps might simply be impractically large in terms of computation time and expense.

Second, besides the truncation error that we have discussed there is also machine roundoff error, and that error can be expected to grow with the number of calculations. Thus, as we diminish the step size h and increase the number of steps, to reduce the truncation error, we inflict a roundoff error penalty that diminishes the intended increase in accuracy. In fact, we can anticipate the existence of an optimal h value so that to decrease h below that value is counterproductive. Said differently, a given level of accuracy may prove unobtainable because of the growth in the roundoff error as h is reduced.

Finally, there is an important practical question: How do we know how small to choose h? We will have more to say about this later, but for now let us give a simple procedure, namely, reducing h until the results settle down to the desired accuracy. For instance, suppose we solve (2) by Euler's method using $h = 0.5$ first. Pick any fixed point x in the interval of interest, such as $x = 1.5$. The computed solution there is 4.4375. Now reduce h, say to 0.1, and run the program again. The result this time, at $x = 1.5$, is 6.1095. Since those results differ considerably, reduce h again, say to 0.02, and run the program again. Simpy repeat that procedure until the solution at $x = 1.5$ settles down to the desired number of significant figures. Accept the results of the final run, and discard the others. (Of course, one will not have an exact solution to compare with as we did in Table 1.)

The foregoing idea is merely a rule of thumb, and is the same idea that we use in computing an infinite series: keep adding more and more terms until successive partial sums agree with the desired number of significant figures.

Closure. The Euler method is embodied in (4). It is easy to implement, either using a hand-held calculator or programming it to be run on a computer. The method is convergent but only of first order and hence is not very accurate. Thus, it is important to develop more accurate methods, and we do that in the next section.

We also use our discussion of the Euler method to introduce the concept of the local and accumulated truncation errors e_n and E_n, respectively, which are

due to the approximate discretization of the problem and which have nothing to do with additional errors that enter due to machine roundoff. The former is the error incurred in a single step, and the latter is the accumulated error over the entire caclulation. Finally, we define the method to be convergent if the accumulated truncation error E_n tends to zero at any given fixed point x, as the step size h tends to zero, and of order p if $E_n = O(h^p)$ as $h \to 0$. The Euler method is convergent and of order one.

EXERCISES 6.2

1. Derive the particular solution (3) of the initial-value problem (2).

2. Use the Euler method to compute, by hand, y_1, y_2, and y_3 for the specified initial-value problem using $h = 0.2$.

(a) $y' = -y; \quad y(0) = 1$
(b) $y' = 2xy; \quad y(0) = 0$
(c) $y' = 3x^2y^2; \quad y(0) = 0$
(d) $y' = 1 + 2xy^2; \quad y(1) = -2$
(e) $y' = 2xe^{-y}; \quad y(1) = -1$
(f) $y' = x^2 - y^2; \quad y(3) = 5$
(g) $y' = x \sin y; \quad y(0) = 0$
(h) $y' = \tan(x + y); \quad y(1) = 2$
(i) $y' = 5x - 2\sqrt{y}; \quad y(0) = 4$
(j) $y' = \sqrt{x + y}; \quad y(0) = 3$

3. Program and run Euler's method for the initial-value problem $y' = f(x, y)$, with $y(0) = 1$ and $h = 0.1$, through y_{10}. Print $y_1, \ldots, y_{10}$ and the exact solution $y(x_1), \ldots, y(x_{10})$ as well. (Six significant figures will suffice.) Evaluate E_{10}. Use the $f(x, y)$ specified below.

(a) $2x$	(b) $-6y^2$	(c) $x + y$
(d) $y \sin x$	(e) $(y^2 + 1)/2$	(f) $4xe^{-y}$
(g) $1 + x^2 + y$	(h) $-y \tan x$	(i) e^{x+y}

4. (a)–(h) Program and run Euler's method for the initial-value problem $y' = f(x, y)$ (with f given in the corresponding part of Exercise 3), and print out the result at $x = 0.5$. Use $h = 0.1$, then 0.05, then 0.01, then 0.005, then 0.001, and compute the accumulated truncation error at $x = 0.5$ for each case. Is the rate of decrease of the accumulated truncation error, as h decreases, consistent with the fact that Euler's method is a first-order method? Explain.

5. Thus far we have taken the step h to be positive, and therefore developed a solution to the right of the initial point. Is Euler's method valid if we use a negative step, $h < 0$, and hence develop a solution to the left? Explain.

6. We have seen that by discretizing the problem, we can approximate the solution $y(x)$ of a differential equation $y' = f(x, y)$ by a discrete variable y_n by solving

$$y_{n+1} = y_n + f(x_n, y_n)h \tag{6.1}$$

sequentially, for $n = 0, 1, 2, \ldots$. Besides being a numerical algorithm for the calculation of the y_n's, (6.1) is an example of a **difference equation** governing the sequence of y_n's, just as $y' = f(x, y)$ is a differential equation governing $y(x)$. If f is simple enough it may be possible to solve (6.1) for y_n analytically, and that idea is the focus of this exercise. Specifically, consider $y' = Ay$, where A is a given constant. Then (6.1) becomes

$$y_{n+1} = (1 + Ah)y_n. \tag{6.2}$$

(a) Derive the solution

$$y_n = C(1 + Ah)^n \tag{6.3}$$

of (6.2), where C is the initial value y_0, if one is specified.
(b) Show that as $h \to 0$ (6.3) does converge to the solution Ce^{Ax} of the original equation $y' = Ay$. HINT: Begin by expressing $(1 + Ah)^n$ as $e^{\ln(1+Ah)^n}$. NOTE: Thus, for the simple differential equation $y' = Ay$ we have been able to prove the convergence of the Euler method by actually solving (6.2) for y_n, in closed form, then taking the limit of that result as $h \to 0$.
(c) Use computer software to obtain the solution (6.3) of the difference equation (6.2). On *Maple*, for instance, use the rsolve command.

7. In this section we have taken the step size h to be a constant from one step to the next. Is there any reason why we could not vary h from one step to the next? Explain.

6.3 Improvements: Midpoint Rule and Runge–Kutta

Our objective in this section is to develop more accurate methods than the first-order Euler method – namely, higher-order methods. In particular, we are aiming at the widely used fourth-order Runge–Kutta method, which is an excellent general-purpose differential equation solver. To bridge the gap between these two methods, we begin with some general discussion about how to develop higher-order methods.

6.3.1. Midpoint rule. To derive more accurate differential equation solvers, Taylor series (better yet, Taylor's formula with remainder) offers a useful line of approach. To illustrate, consider the Taylor's formula with remainder,

$$y(x) = y(a) + y'(a)(x-a) + \frac{y''(\xi)}{2!}(x-a)^2, \tag{1}$$

where ξ is some point in $[a, x]$. If we let $x = x_{n+1}$, $a = x_n$, and $x - a = x_{n+1} - x_n = h$, then (1) becomes

$$y(x_{n+1}) = y(x_n) + y'(x_n)h + \frac{y''(\xi)}{2!}h^2. \tag{2}$$

Since $y' = f(x, y)$, we can replace the $y'(x_n)$ in (2) by $f(x_n, y(x_n))$. Also, the last term in (2) can be expressed more simply as $O(h^2)$ so we have

$$y(x_{n+1}) = y(x_n) + f(x_n, y(x_n))h + O(h^2). \tag{3}$$

If we neglect the $O(h^2)$ term and call attention to the approximation thereby incurred by replacing the exact values $y(x_{n+1})$ and $y(x_n)$ by the approximate values y_{n+1} and y_n, respectively, then we have the Euler method

$$y_{n+1} = y_n + f(x_n, y_n)h. \tag{4}$$

Since the term that we dropped in passing from (3) to (4) was $O(h^2)$, the local truncation error is $O(h^2)$, and the accumulated truncation error is $O(h)$.

One way to obtain a higher-order method is to retain more terms in the Taylor's formula. For instance, begin with

$$y(x_{n+1}) = y(x_n) + y'(x_n)h + \frac{y''(x_n)}{2}h^2 + \frac{y'''(\eta)}{6}h^3 \tag{5}$$

in place of (2) or, since $y'' = \dfrac{d}{dx}f(x, y(x)) = f_x + f_y y' = f_x + f_y f$,

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + f(x_n, y(x_n))h \\ &\quad + \frac{1}{2}\left[f_x(x_n, y(x_n)) + f_y(x_n, y(x_n))f(x_n, y(x_n))\right]h^2 + O(h^3). \end{aligned} \tag{6}$$

If we truncate (6) by dropping the $O(h^3)$ term, and change $y(x_{n+1})$ and $y(x_n)$ to y_{n+1} and y_n, respectively, then we have the method

$$y_{n+1} = y_n + f(x_n, y_n)h + \frac{1}{2}\left[f_x(x_n, y_n) + f_y(x_n, y_n)f(x_n, y_n)\right]h^2 \qquad (7)$$

with a local truncation error that is $O(h^3)$ and an accumulated truncation error that is $O(h^2)$; that is, we now have a *second*-order method.

Why do we say that (7) is a second-order method? Following the same heuristic reasoning as in Section 6.2, the accumulated truncation error E_n is of the order of the local truncation error times the number of steps so

$$E_n = O(h^3) \cdot n = O(h^3)\frac{n\,h}{h} = O(h^3)\frac{x_n}{h} = O(h^2) \cdot x_n = O(h^2),$$

as claimed. In fact, as a simple rule of thumb it can be said that if the local truncation error is $O(h^p)$, with $p > 1$, then the accumulated truncation error is $O(h^{p-1})$, and one has a $(p-1)$th-order method.

Although the second-order convergence of (7) is an improvement over the first-order convergence of Euler's method, the attractiveness of (7) is diminished by an approximately threefold increase in the computing time per step since it requires three function evaluations (f, f_x, f_y) per step whereas Euler's method requires only one (f). It's true that to carry out one step of Euler's method we need to evaluate f, multiply that by h, and add the result to y_n, but we can neglect the multiplication by h and addition of y_n on the grounds that a typical $f(x, y)$ involves many more arithmetic steps than that. Thus, as a rule of thumb, one compares the computation time per step of two methods by comparing only the number of function evaluations per step.

Not satisfied with (7) because it requires three function evaluations, let us return to Taylor's formula (5). If we replace h by $-h$, that amounts to making a backward step so the term on the left will be $y(x_{n-1})$ instead of $y(x_{n+1})$. Making those changes, and also copying (5), for comparison, we have

$$y(x_{n-1}) = y(x_n) - y'(x_n)h + \frac{y''(x_n)}{2}h^2 - \frac{y'''(\zeta)}{6}h^3, \qquad (8a)$$

$$y(x_{n+1}) = y(x_n) + y'(x_n)h + \frac{y''(x_n)}{2}h^2 + \frac{y'''(\eta)}{6}h^3, \qquad (8b)$$

respectively, where ζ is some point in $[x_{n-1}, x_n]$ and η is some point in $[x_n, x_{n+1}]$. Now we can eliminate the bothersome y'' terms by subtracting (8a) from (8b). Doing so gives

$$y(x_{n+1}) - y(x_{n-1}) = 2y'(x_n)h + \frac{y'''(\eta) + y'''(\zeta)}{6}h^3$$

or

$$y(x_{n+1}) = y(x_{n-1}) + 2f\left(x_n, y(x_n)\right)h + O\left(h^3\right).$$

Finally, if we drop the $O(h^3)$ term and change $y(x_{n+1})$, $y(x_{n-1})$, $y(x_n)$ to y_{n+1}, y_{n-1}, y_n, respectively, we have

$$\boxed{y_{n+1} = y_{n-1} + f(x_n, y_n)(2h),} \tag{9}$$

which method is known as the **midpoint rule**. Like (7), the midpoint rule is a second-order method but, unlike (7), it requires only one function evaluation per step. It is an example of a **multi-step method** because it uses information from more than one of the preceding points – namely, from two: the nth and $(n-1)$th. Thus, it is a two-step method whereas Euler's method and the Taylor series method given by (7) are single-step methods.

A disadvantage of the midpoint rule (and other multi-step methods) is that it is not self-starting. That is, the first step gives y_1 in terms of x_0, y_0, y_{-1}, but y_{-1} is not defined. Thus, (9) applies only for $n \geq 1$, and to get the method started we need to compute y_1 by a different method. For instance, we could use Euler's method to obtain y_1 and then switch over to the midpoint rule (9). Of course, if we do that we should do it not in a single Euler step but in many so as not to degrade the subsequent second-order accuracy.

EXAMPLE 1. Consider the same "test problem" as in Section 6.2,

$$y' = y + 2x - x^2; \qquad y(0) = 1, \qquad (0 \leq x < \infty) \tag{10}$$

with the exact solution $y(x) = x^2 + e^x$. Let us use the midpoint rule with $h = 0.1$. To get it started, carry out ten steps of Euler's method with $h = 0.01$. The result of those steps is the approximate solution 1.11358 at $x = 0.1$, which we now take as y_1. Then proceeding with the midpoint rule we obtain from (9)

$$\begin{aligned} y_2 &= y_0 + 2\left(y_1 + 2x_1 - x_1^2\right) h \\ &= 1 + 2\left(1.11358 + 0.2 - 0.01\right)(0.1) = 1.26072 \\ y_3 &= y_1 + 2\left(y_2 + 2x_2 - x_2^2\right) h \\ &= 1.11358 + 2\left(1.26072 + 0.4 - 0.04\right)(0.1) = 1.43772, \end{aligned}$$

and so on. The results are shown in Table 1 and contrasted with the less accurate Euler results using the same step size, $h = 0.1$. ∎

Before leaving the midpoint rule, it is interesting to interpret the improvement in accuracy, from Euler to midpoint, graphically. If we solve

$$y(x_{n+1}) \approx y(x_n) + y'(x_n)h \qquad \text{(Euler)} \tag{11}$$

and

$$y(x_{n+1}) \approx y(x_{n-1}) + 2y'(x_n)h \qquad \text{(midpoint)} \tag{12}$$

for $y'(x_n)$, we have

$$y'(x_n) \approx \frac{y(x_{n+1}) - y(x_n)}{h} \qquad \text{(Euler)} \tag{13}$$

Table 1. Comparison of Euler, midpoint rule, and exact solutions of the initial-value problem (10), with $h = 0.1$.

x	Euler	Midpoint	Exact
0.10	1.10000	1.11358	1.11517
0.20	1.22900	1.26072	1.26140
0.30	1.38790	1.43772	1.43986
0.40	1.57769	1.65026	1.65182
0.50	1.79946	1.89577	1.89872

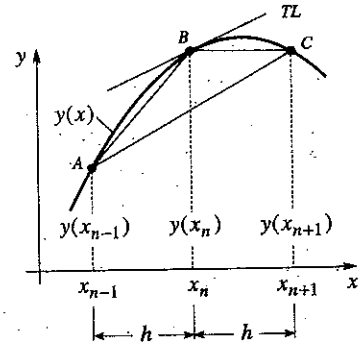


Figure 1. Graphical interpretation of midpoint rule versus Euler.

and

$$y'(x_n) \approx \frac{y(x_{n+1}) - y(x_{n-1})}{2h}, \qquad \text{(midpoint)} \tag{14}$$

which are difference quotient approximations of the derivative $y'(x_n)$. In Fig. 1, we can interpret (14) and (13) as approximating $y'(x_n)$ by the slopes of the chords AC and BC, respectively, while the exact $y'(x_n)$ is the slope of the tangent line TL at x_n. We can see from the figure that AC gives a more accurate approximation than BC.

6.3.2. Second-order Runge–Kutta. The Runge–Kutta methods are developed somewhat differently. Observe that the low-order Euler method $y_{n+1} = y_n + f(x_n, y_n)h$ amounts to an extrapolation away from the initial point (x_n, y_n) using the slope $f(x_n, y_n)$ at that point. Expecting an average slope to give greater accuracy, one might try the algorithm

$$y_{n+1} = y_n + \frac{1}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right] h, \tag{15}$$

which uses an average of the slopes at the initial and final points. Unfortunately, the formula (15) does not give y_{n+1} explicitly since y_{n+1} appears not only on the left-hand side but also in the argument of $f(x_{n+1}, y_{n+1})$. Intuition tells us that we should still do well if we replace that y_{n+1} by an estimated value, say, the Euler estimate $y_{n+1} = y_n + f(x_n, y_n)h$. Then the revised version of (15) is

$$y_{n+1} = y_n + \frac{1}{2}\left\{f(x_n, y_n) + f\left[x_{n+1}, y_n + f(x_n, y_n)h\right]\right\} h. \tag{16}$$

Thus, guided initially by intuition we can put the idea on a rigorous basis by considering a method of the form

$$y_{n+1} = y_n + \left\{af(x_n, y_n) + bf\left[x_n + \alpha h, y_n + \beta f(x_n, y_n)h\right]\right\} h \tag{17}$$

and choosing the adjustable parameters a, b, α, β so as to make the order of the method (17) as high as possible; α, β determine the second slope location, and a, b

determine the "weights" of the two slopes. That is, we seek a, b, α, β so that the left- and right-hand sides of

$$y(x_{n+1}) \approx y(x_n) + \{af[x_n, y(x_n)] + bf[x_n + \alpha h, y(x_n) + \beta f[x_n, y(x_n)]\, h]\}\, h \tag{18}$$

agree to as high a degree in h as possible. Thus, expand the left-hand side (LHS) and right-hand side (RHS) of (18) in Taylor series in h:

$$\begin{aligned} \text{LHS} &= y(x_n) + y'(x_n)h + \frac{y''(x_n)}{2}h^2 + \cdots \\ &= y + fh + \frac{1}{2}(f_x + f_y f)\, h^2 + \cdots \end{aligned} \tag{19}$$

where y means $y(x_n)$ and the arguments of f, f_x, f_y are $x_n, y(x_n)$. Similarly (Exercise 9),

$$\text{RHS} = y + (a+b)fh + (\alpha f_x + \beta f f_y)\, bh^2 + \cdots. \tag{20}$$

Matching the h terms requires that

$$a + b = 1. \tag{21a}$$

Matching the h^2 terms, for any function f requires that

$$\alpha b = \frac{1}{2} \quad \text{and} \quad \beta b = \frac{1}{2}. \tag{21b}$$

The outcome then is that any method (17), with a, b, α, β chosen so as to satisfy (21), has a local truncation error that is $O(h^3)$ and is therefore a second-order method [subject to mild conditions on f such as the continuity of f and its first- and second-order partial derivatives so that we can justify the steps in deriving (19) and (20)]. These are the **Runge–Kutta methods of second order.**†

For instance, with $a = b = 1/2$ and $\alpha = \beta = 1$ we have

$$y_{n+1} = y_n + \frac{1}{2}\{f(x_n, y_n) + f[x_{n+1}, y_n + f(x_n, y_n)h]\}\, h, \tag{22}$$

which is usually expressed in the computationally convenient form

$$\boxed{\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{2}(k_1 + k_2), \\ k_1 &= hf(x_n, y_n), \qquad k_2 = hf(x_{n+1}, y_n + k_1). \end{aligned}} \tag{23}$$

†The Runge–Kutta method was originated by *Carl D. Runge* (1856–1927), a German physicist and mathematician, and extended by the German aerodynamicist and mathematician *M. Wilhelm Kutta* (1867–1944). Kutta is well known for the Kutta–Joukowski formula for the lift on an airfoil, and for the "Kutta condition" of classical airfoil theory.

To understand this result, note that Euler's method would give $y_{n+1} = y_n + f(x_n, y_n)\,h$. If we denote that Euler estimate as y_{n+1}^{Euler}, then (22) can be expressed as

$$y_{n+1} = y_n + \frac{f(x_n, y_n) + f\left(x_{n+1}, y_{n+1}^{\text{Euler}}\right)}{2}\,h.$$

That is, we take a tentative step using Euler's method, then we average the slopes at the initial point x_n, y_n and at the Euler estimate of the final point $x_{n+1}, y_{n+1}^{\text{Euler}}$, and then make another Euler step, this time using the improved (average) slope. For this reason (23) is also known as the **improved Euler method**.

A different choice, $a = 0, b = 1, \alpha = \beta = 1/2$, gives what is known as the **modified Euler method**.

EXAMPLE 2. Let us proceed through the first two steps of the improved Euler method (23) for the same test problem as was used in Example 1,

$$y' = y + 2x - x^2; \qquad y(0) = 1, \qquad (0 \le x < \infty) \tag{24}$$

with $h = 0.1$; a more detailed illustration is given in Section 6.3.3 below. Here, $f(x, y) = y + 2x - x^2$.

$n = 0$:

$$k_1 = hf(x_0, y_0) = 0.1\left[1 + 0 - (0)^2\right] = 0.1,$$
$$k_2 = hf(x_1, y_0 + k_1) = 0.1\left[(1 + 0.1) + 2(0.1) - (0.1)^2\right] = 0.129,$$
$$y_1 = y_0 + \tfrac{1}{2}(k_1 + k_2) = 1 + 0.5\,(0.1 + 0.129) = 1.1145;$$

$n = 1$:

$$k_1 = hf(x_1, y_1) = 0.1\left[1.1145 + 2(0.1) - (0.1)^2\right] = 0.13045,$$
$$k_2 = hf(x_2, y_1 + k_1)$$
$$= 0.1\left[(1.1145 + 0.13045) + 2(0.2) - (0.2)^2\right] = 0.160495,$$
$$y_2 = y_1 + \tfrac{1}{2}(k_1 + k_2) = 1.1145 + 0.5\,(0.13045 + 0.160495) = 1.2600,$$

compared with the values $y(x_1) = y(0.1) = 1.1152$ and $y(x_2) = y(0.2) = 1.2614$ obtained from the known exact solution $y(x) = x^2 + e^x$. ∎

6.3.3. Fourth-order Runge–Kutta. Using this idea of a weighted average of slopes at various points in the x, y plane, with the weights and locations determined so as to maximize the order of the method, one can derive higher-order Runge–Kutta methods as well, although the derivations are quite tedious. One of the most

commonly used is the **fourth-order Runge–Kutta method**:

$$\boxed{\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 &= hf\left(x_n, y_n\right), \qquad k_2 = hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_1\right), \\ k_3 &= hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_2\right), \quad k_4 = hf\left(x_{n+1}, y_n + k_3\right), \end{aligned}} \tag{25}$$

which we give without derivation. Here the effective slope used is a weighted average of the slopes at the four points (x_n, y_n), $(x_n + h/2, y_n + k_1/2)$, $(x_n + h/2, y_n + k_2/2)$ and $(x_{n+1}, y_n + k_3)$ in the x, y plane, an average because the sum of the coefficients $1/6$, $2/6$, $2/6$, $1/6$ that multiply the k's is 1. Similarly, the sum of the coefficients $1/2, 1/2$ in the second-order version (23) is 1 as well.

EXAMPLE 3. As a summarizing illustration, we solve another "test problem,"

$$y' = -y; \qquad y(0) = 1 \tag{26}$$

by each of the methods considered, using a step size of $h = 0.05$ and single precision arithmetic (on the computer used that amounts to carrying eight significant figures; double precision would carry 16). The results are given in Table 2, together with the exact solution $y(x) = e^{-x}$ for comparison; 0.529E + 2, for instance, means 0.529×10^2. The value of y_1 for the midpoint rule was obtained by Euler's method with a reduced step size of 0.0025.

To illustrate the fourth-order Runge–Kutta calculation, let us proceed through the first step:

$$\begin{aligned} n = 0: \qquad k_1 &= hf\left(x_0, y_0\right) = -hy_0 = -0.05(1) = -0.05, \\ k_2 &= hf\left(x_0 + \tfrac{h}{2}, y_0 + \tfrac{1}{2}k_1\right) = -h\left(y_0 + \tfrac{1}{2}k_1\right) \\ &= -0.05(1 - 0.025) = -0.04875, \\ k_3 &= hf\left(x_0 + \tfrac{h}{2}, y_0 + \tfrac{1}{2}k_2\right) = -h\left(y_0 + \tfrac{1}{2}k_2\right) \\ &= -0.05(1 - 0.024375) = -0.04878125, \\ k_4 &= hf\left(x_1, y_0 + k_3\right) = -h\left(y_0 + k_3\right) \\ &= -0.05(1 - 0.04878125) = -0.047560938, \\ y_1 &= y_0 + \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 0.95122943, \end{aligned}$$

which final result does agree with the corresponding entry in Table 2. Actually, there is a discrepancy of 2 in the last digit, but an error of that size is not unreasonable in view of the fact that the machine used carried only eight significant figures.

Most striking is the excellent accuracy of the fourth-order Runge–Kutta method, with six significant figure accuracy over the entire calculation.

COMMENT. We see that the midpoint rule and the second-order Runge–Kutta method yield comparable results initially, but the midpoint rule eventually develops an error that

Table 2. Comparison of the Euler, midpoint rule, second-order Runge–Kutta, fourth-order Runge–Kutta, and exact solutions of the initial-value problem (26), with $h = 0.05$.

x	Euler	Midpoint	2nd-order Runge–Kutta	4th-order Runge–Kutta	Exact $= e^{-x}$
0.00	1.00000000 E+0	1.00000000 E+0	1.00000000 E+0	1.00000000 E+0	1.00000000 E+0
0.05	0.94999999 E+0	0.95116991 E+0	0.95125002 E+0	0.95122945 E+0	0.95122945 E+0
0.10	0.90249997 E+0	0.90488303 E+0	0.90487659 E+0	0.90483743 E+0	0.90483743 E+0
0.15	0.85737497 E+0	0.86068159 E+0	0.86076385 E+0	0.86070800 E+0	0.86070800 E+0
0.20	0.81450623 E+0	0.81881487 E+0	0.81880164 E+0	0.81873077 E+0	0.81873077 E+0
0.25	0.77378094 E+0	0.77880013 E+0	0.77888507 E+0	0.77880079 E+0	0.77880079 E+0
0.30	0.73509192 E+0	0.74093485 E+0	0.74091440 E+0	0.74081820 E+0	0.74081820 E+0
⋮					
2.00	0.12851217 E+0	0.13573508 E+0	0.13545239 E+0	0.13533530 E+0	0.13533528 E+0
2.05	0.12208656 E+0	0.12853225 E+0	0.12884909 E+0	0.12873492 E+0	0.12873492 E+0
2.10	0.11598223 E+0	0.12288185 E+0	0.12256770 E+0	0.12245644 E+0	0.12245644 E+0
2.15	0.11018312 E+0	0.11624406 E+0	0.11659253 E+0	0.11648417 E+0	0.11648415 E+0
2.20	0.10467397 E+0	0.11125745 E+0	0.11090864 E+0	0.11080316 E+0	0.11080315 E+0
2.25	0.99440269 E−1	0.10511832 E+0	0.10550185 E+0	0.10539923 E+0	0.10539922 E+0
2.30	0.94468258 E−1	0.10074562 E+0	0.10035863 E+0	0.10025885 E+0	0.10025885 E+0
⋮					
5.00	0.59205294 E−2	0.12618494 E−1	0.67525362 E−2	0.67379479 E−2	0.67379437 E−2
5.05	0.56245029 E−2	0.25511871 E−3	0.64233500 E−2	0.64093345 E−2	0.64093322 E−2
5.10	0.53432779 E−2	0.12592983 E−1	0.61102118 E−2	0.60967477 E−2	0.60967444 E−2
5.15	0.50761141 E−2	-0.10041796 E−2	0.58123390 E−2	0.57994057 E−2	0.57994043 E−2
5.20	0.48223082 E−2	0.12693400 E−1	0.55289874 E−2	0.55165654 E−2	0.55165626 E−2
5.25	0.45811930 E−2	-0.22735195 E−2	0.52594491 E−2	0.52475194 E−2	0.52475161 E−2
5.30	0.43521333 E−2	0.12920752 E−1	0.50030509 E−2	0.49915947 E−2	0.49915928 E−2
⋮					
9.70	0.47684727 E−4	0.64383668 E+0	0.61541170 E−4	0.61283507 E−4	0.61283448 E−4
9.75	0.45300490 E−4	-0.67670959 E+0	0.58541038 E−4	0.58294674 E−4	0.58294663 E−4
9.80	0.43035467 E−4	0.71150762 E+0	0.55687164 E−4	0.55451608 E−4	0.55451590 E−4
9.85	0.40883693 E−4	-0.74786037 E+0	0.52972413 E−4	0.52747200 E−4	0.52747171 E−4
9.90	0.38839509 E−4	0.78629363 E+0	0.50390008 E−4	0.50174691 E−4	0.50174654 E−4
9.95	0.36897534 E−4	-0.82648975 E+0	0.47933496 E−4	0.47727641 E−4	0.47727597 E−4
10.00	0.35052657 E−4	0.86894262 E+0	0.45596738 E−4	0.45399935 E−4	0.45399931 E−4

oscillates in sign, from step to step, and grows in magnitude. The reason for this strange (and incorrect) behavior will be studied in Section 6.5. ▮

Of course, in real applications we do not have the exact solution to compare with the numerical results. In that case, how do we know whether or not our results are sufficiently accurate? A useful rule of thumb, mentioned in Section 6.2, is to redo the entire calculation, each time with a smaller step size, until the results "settle down" to the desired number of significant digits.

Thus far we have taken h to be a constant, for simplicity, but there is no reason why it cannot be varied from one step to the next. In fact, there may be a compelling reason to do so. For instance, consider the equation $y' + y = \tanh 20x$ on $-10 \leq x \leq 10$. The function $\tanh 20x$ is almost a constant, except near the origin, where it varies dramatically approximately from -1 to $+1$. Thus, we need a very fine step size h near the origin for good accuracy, but to use that h over the entire x interval would be wasteful in terms of computer time and expense.

One can come up with a rational scheme for varying the step size to maintain a consistent level of accuracy, but such refinements are already available within existing software. For example, the default numerical differential equation solver in *Maple* is a "fourth-fifth order Runge–Kutta–Fehlberg method" denoted as RKF45 in the literature. According to RKF45, a tentative step is made, first using a fourth-order Runge–Kutta method, and then again using a fifth-order Runge–Kutta method. If the two results agree to a prespecified number of significant digits, then the fifth-order result is accepted. If they agree to more than that number of significant digits, then h is increased and the next step is made. If they agree to less than that number of significant digits, then h is decreased and the step is repeated.

6.3.4. Empirical estimate of the order. (Optional) The relative accuracies achieved by the different methods, as seen from the results in Table 2, strikingly reveal the importance of the order of the method. Thus, it is important to know how to verify the order of whatever method we use, if only as a partial check on the programming.

Recall that by a method being of order p we mean that at any chosen x the error behaves as Ch^p for some constant C:

$$y_{\text{exact}} - y_{\text{comp}} \sim Ch^p \tag{27}$$

as $h \to 0$. Suppose we wish to check the order of a given method. Select a test problem such as the one in Example 3, and use the method to compute y at any x point such as $x = 1$, for two different h's say h_1 and h_2. Letting $y_{\text{comp}}^{(1)}$ and $y_{\text{comp}}^{(2)}$ denote the y's computed at $x = 1$ using step sizes of h_1 and h_2, respectively, we have

$$y_{\text{exact}} - y_{\text{comp}}^{(1)} \approx Ch_1^p,$$
$$y_{\text{exact}} - y_{\text{comp}}^{(2)} \approx Ch_2^p.$$

Dividing one equation by the other, to cancel the unknown C, and solving for p,

gives

$$p \approx \frac{\ln\left[\frac{y_{\text{exact}}-y_{\text{comp}}^{(1)}}{y_{\text{exact}}-y_{\text{comp}}^{(2)}}\right]}{\ln\left[\frac{h_1}{h_2}\right]}. \tag{28}$$

To illustrate, let us run the test problem (26) with Euler's method, with $h_1 = 0.1$ and $h_2 = 0.05$. The results at $x = 1$ are

$$\begin{aligned} h_1 &= 0.1, \qquad y_{\text{comp}}^{(1)} = 0.348678440100, \\ h_2 &= 0.05, \qquad y_{\text{comp}}^{(2)} = 0.358485922409, \end{aligned}$$

and since $y_{\text{exact}}(1) = 0.367879441171$, (28) gives $p \approx 1.03$, which is respectably close to 1. We should be able to obtain a more accurate estimate of p by using smaller h's since (27) holds only as $h \to 0$. In fact, using $h_1 = 0.05$ and $h_2 = 0.02$ gives $p \approx 1.01$. Using those same step sizes, we also obtain $p \approx 2.05, 2.02$, and 4.03 for the midpoint rule, second-order Runge–Kutta, and fourth-order Runge–Kutta methods, respectively.

Why not use even smaller h's to determine p more accurately? Two difficulties arise. One is that as the h's are decreased the computed solutions become more and more accurate, and the $y_{\text{exact}} - y_{\text{comp}}^{(1)}$ and $y_{\text{exact}} - y_{\text{comp}}^{(2)}$ differences in (28) are known to fewer and fewer significant figures, due to cancelation. This is especially true for a high-order method. The other difficulty is that (27) applies to the truncation error alone so, implicit in our use of (27) is the assumption that roundoff errors are negligible. If we make h too small, that assumption may become invalid. For both of these reasons it is important to use extended precision for such calculations, as we have for the preceding calculations.

6.3.5. Multi-step and predictor-corrector methods. (Optional) We've already called attention to the multi-step nature of the midpoint rule. Our purpose in this optional section is to give a brief overview of a class of multi-step methods known as **Adams–Bashforth methods**, obtained by integrating $y' = f(x, y)$ from x_n to x_{n+1}:

$$\int_{x_n}^{x_{n+1}} y'\,dx = \int_{x_n}^{x_{n+1}} f\left(x, y(x)\right) dx \tag{29}$$

or

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f\left(x, y(x)\right) dx. \tag{30}$$

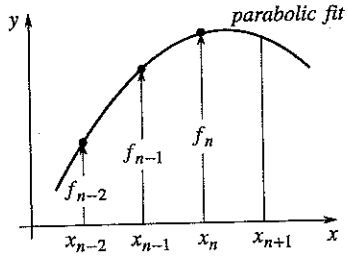


Figure 2. Adams–Bashforth interpolation of f for the case $m = 2$.

To evaluate the integral, we fit $f\left(x, y(x)\right)$ with a polynomial of degree m, which is readily integrated. The polynomial interpolates (i.e., takes on the same values as) $f\left(x, y(x)\right)$ at the $m + 1$ points $x_{n-m}, \ldots, x_{n-1}, x_n$ as illustrated in Fig. 2 for the case $m = 2$. As the simplest case, let $m = 0$. Then the zeroth degree polynomial approximation of $f\left(x, y(x)\right)$ on $[x_n, x_{n+1}]$ is $f\left(x, y(x)\right) \approx f_n$, where f_n denotes $f(x_n, y_n)$, and (30) gives the familiar Euler method $y_{n+1} = y_n + f_n h$. Omitting

the steps in this overview we state that with $m = 3$ one obtains the fourth-order **Adams–Bashforth method**

$$y_{n+1} = y_n + (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})\frac{h}{24} \tag{31a}$$

with a local truncation error of

$$(e_n)_{AB} = \frac{251}{720}y^{(v)}(\xi)h^5 \tag{31b}$$

for some ξ in $[x_{n-3}, x_n]$. We can see that (31a) is not self-starting; it applies only for $n = 3, 4, \ldots$, so the first three steps (for $n = 0, 1, 2$) need to be carried out by some other method.

Suppose that instead of interpolating f at $x_{n-m}, \ldots, x_{n-1}, x_n$ we interpolate at $x_{n-m+1}, \ldots, x_n, x_{n+1}$. With $m = 3$, again, one obtains the fourth-order **Adams–Moulton method**

$$y_{n+1} = y_n + (9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})\frac{h}{24} \tag{32a}$$

with a local truncation error of

$$(e_n)_{AM} = -\frac{19}{720}y^{(v)}(\xi)h^5, \tag{32b}$$

where the ξ's in (31b) and (32b) are different, in general.

Although both methods are fourth order, the Adams–Moulton method is more accurate because the constant factor in $(e_n)_{AM}$ is roughly thirteen times smaller than the constant in $(e_n)_{AB}$. This increase in accuracy occurs because the points $x_{n-m+1}, \ldots, x_n, x_{n+1}$ are more centered on the interval of integration (from x_n to x_{n+1}) than the points $x_{n-m}, \ldots, x_{n-1}, x_n$. On the other hand, the term $f_{n+1} = f(x_{n+1}, y_{n+1})$ in (32a) is awkward because the argument y_{n+1} is not yet known! [If f is linear in y, then (32a) can be solved for y_{n+1} by simple algebra, and the awkwardness disappears.] Thus, the method (32a) is said to be of **closed** type, whereas (31a) and all of our preceding methods have been of **open** type.

To handle this difficulty, it is standard practice to solve closed formulas by iteration. Using superscripts to indicate the iterate, (32a) becomes

$$y_{n+1}^{(k+1)} = y_n + \left[9f\left(x_{n+1}, y_{n+1}^{(k)}\right) + 19f_n - 5f_{n-1} + f_{n-2}\right]\frac{h}{24}. \tag{33}$$

To start the iteration, we compute $y_{n+1}^{(0)}$ from a **predictor formula**, with subsequent corrections made by the **corrector formula** (33). It is recommended that the predictor and corrector formulas be of the same order (certainly, the corrector should never be of lower order than the predictor) with the corrector applied only once. Thus, the Adams–Bashforth and Adams–Moulton methods constitute a natural **predictor-corrector** pair with "AB" as the predictor and "AM" as the corrector. Why might we choose the fourth-order AB–AM predictor-corrector over

the Runge–Kutta method of the same order or vice versa? On the negative side, AB–AM is not self-starting, it requires the storage of f_{n-3}, f_{n-2}, and f_{n-1}, and is more tedious to program. On the other hand, it involves only two function evaluations per step (namely, f_n and f_{n+1}) if the corrector is applied only once, whereas Runge–Kutta involves four. Thus, if $f(x, y)$ is reasonably complicated then we can expect AB–AM to be almost twice as fast. In large-scale computing, the savings can be significant.

Closure. Motivated to seek higher-order methods than the first-order Euler method, we use a Taylor series approach to obtain the second-order midpoint rule. Though more accurate, a disadvantage of the midpoint rule is that it is not self-starting. Pursuing a different approach, we look at the possibility of using a weighted average of slopes at various points in the x, y plane, with the weights and locations determined so as to maximize the order of the method. We thereby derive the second-order Runge–Kutta method and present, without derivation, the fourth-order Runge–Kutta method. The latter is widely used because it is accurate and self-starting.

Because of the importance of the order of a given method, we suggest that the order be checked empirically using a test problem with a known exact solution. The resulting approximate expression for the order is given by (28).

In the final section we return to the idea of multistep methods and present a brief overview of the Adams–Bashforth methods, derived most naturally from an approximate integral approach. Though not self-starting, the fourth-order Adams-Bashforth method (31a) is faster than the Runge–Kutta method of the same order because it requires only one function evaluation per step (namely, f_n; the f_{n-1}, f_{n-2}, and f_{n-3} terms are stored from previous steps). A further refinement consists of predictor-corrector variations of the Adams–Bashforth methods. However, we stress that such refinements become worthwhile only if the scope of the computational effort becomes large enough to justify the additional inconvenience caused by such features as the absence of self-starting and predictor-corrector iteration. Otherwise, one might as well stick to a simple and accurate method such as fourth-order Runge–Kutta.

Computer software. Computer-software systems such as *Maple* include numerical differential equation solvers. In *Maple* one can use the dsolve command together with a numeric option. The default numerical solution method is the RKF45 method mentioned above. Note that with the numeric option of dsolve one does not specify a step size h since that choice is controlled within the program and, in general, is varied from step to step to maintain a certain level of accuracy. To specify the absolute error tolerance one can use an additional option called abserr, which is formatted as abserr = Float(1,2-digits) and which means 1 times 10 to the one- or two-digit exponent. For instance, to solve

$$y' = -y; \quad y(0) = 1$$

for $y(x)$ with an absolute error tolerance of 1×10^{-5}, and to print the results at $x = 2, 10$, enter

with(DEtools):

and return. Then enter

dsolve({diff($y(x), x$) $= -y(x)$, $y(0) = 1$}, $y(x)$, type = numeric,
value = array([2, 10]), abserr = Float(1, −5));

and return. The printed result is

$$\left[\begin{array}{c} [x, y(x)] \\ \left[\begin{array}{ll} 2. & .1353337989380555 \\ 10. & .00004501989255717160 \end{array}\right] \end{array}\right]$$

For comparison, the exact solution is $y(2) = \exp(-2) = 0.1353352832$ and $y(10) = \exp(-10) = 0.0000453999$, respectively.

EXERCISES 6.3

1. Evaluate y_1 and y_2 by hand, by the second-order and fourth-order Runge–Kutta methods, with $h = 0.02$. Obtain the exact values $y(x_1)$ and $y(x_2)$ as well.

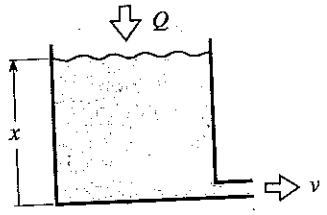
(a) $y' = 3000xy^{-2}$; $y(0) = 2$
(b) $y' = 40xe^{-y}$; $y(0) = 3$
(c) $y' = x + y$; $y(-1) = 5$
(d) $y' = -y\tan x$; $y(1) = -8$
(e) $y' = (y^2 + 1)/4$; $y(0) = 0$
(f) $y' = -2y\sin x$; $y(2) = 5$

2. (a)–(f) Program the second- and fourth-order Runge–Kutta methods and use them to solve the initial-value problem given in the corresponding part of Exercise 1 but with the initial condition $y(0) = 1$. Use $h = 0.05$. Print out all computed values of y, up to $x = 0.5$, as well as the exact solution.

3. Using the test problem (10), do an empirical evaluation of the order of the given method. Use (28), with $h = 0.1$ and 0.05, say. Do the evaluation at two different locations, such as $x = 1$ and $x = 2$. (The order should not depend upon x so your results at the two points should be almost identical.)

(a) Euler's method
(b) Second-order Runge–Kutta method
(c) Fourth-order Runge–Kutta method

4. (*Liquid level*) Liquid is pumped into a tank of horizontal cross-sectional area A (m^2) at a rate Q (liters/sec), and is drained by a valve at its base as sketched in the figure.



According to Bernoulli's principle, the efflux velocity $v(t)$ is approximately $\sqrt{2gx(t)}$, where g is the acceleration of gravity. Thus, a mass balance gives

$$\begin{aligned} Ax'(t) &= Q(t) - Bv(t) \\ &= Q(t) - B\sqrt{2gx(t)}, \end{aligned} \tag{4.1}$$

where B is the cross-sectional area of the efflux pipe. For definiteness, suppose that $A = 1$ and $B\sqrt{2g} = 0.01$ so

$$x' = Q(t) - 0.01\sqrt{x}. \tag{4.2}$$

We wish to know the depth $x(t)$ at the end of 10 minutes ($t = 600$ sec), 20 minutes, ..., up to one hour. Program the computer solution of (4.2) by the second-order Runge–Kutta method for the following cases, and use it to solve for those x values: $x(600), x(1200), \ldots, x(3600)$. (Using the rule of thumb given below Example 3, reduce h until those results settle down to four significant digits.)

(a) $Q(t) = 0.02$; $x(0) = 0$
(b) $Q(t) = 0.02$; $x(0) = 2$
(c) $Q(t) = 0.02$; $x(0) = 4$
(d) $Q(t) = 0.02$; $x(0) = 6$

(e) $Q(t) = 0.02\left(1 - e^{-0.004t}\right)$; $x(0) = 0$
(f) $Q(t) = 0.02\left(1 - e^{-0.004t}\right)$; $x(0) = 8$
(g) $Q(t) = 0.02t$; $x(0) = 0$
(h) $Q(t) = 0.02(1 + \sin 0.1t)$; $x(0) = 0$
NOTE: Surely, we will need h to be small compared to the period 20π of $Q(t)$ in part (h).

5. (a)–(h) (*Liquid level*) Same as Exercise 4, but use fourth-order Runge–Kutta instead of second order.

6. (a)–(h) (*Liquid level*) Same as Exercise 4, but use computer software to do the numerical solution of the differential equation. In *Maple*, for instance, the dsolve command uses the fourth-fifth order RKF45 method.

7. (*Liquid level*) (a) For the case where $Q(t)$ is a constant, derive the general solution of (4.2) in Exercise 4 as

$$Q - 0.01\sqrt{x} - Q\ln\left(Q - 0.01\sqrt{x}\right) = 0.00005t + C, \quad (7.1)$$

where C is the constant of integration.
(b) Evaluate C in (7.1) if $Q = 0.02$ and $x(0) = 0$. Then, solve (7.1) for $x(t)$ at $t = 600, 1200, \ldots, 3600$. NOTE: Unfortunately, (7.1) is in implicit rather than explicit form, but you can use computer software to solve. In *Maple*, for instance, the relevant command is fsolve.

8. Suppose that we have a convergent method, with $E_n \sim Ch^p$ as $h \to 0$. Someone offers to improve the method by either halving C or by doubling p. Which would you choose? Explain.

9. Expand the right-hand side of (18) in a Taylor series in h and show that the result is as given in (20). HINT: To expand the $f(x_n + \alpha h, y + \beta f h)$ term you need to use chain differentiation.

10. (a) Program the fourth-order Runge–Kutta method (25) and use it to run the test problem (10) and to compute y at $x = 1$ using $h = 0.05$ and then $h = 0.02$. From those values and the known exact solution, empirically verify that the method is fourth order.
(b) To see what harm a programming error can cause, change the $x_n + h/2$ in the formula for k_2 to x_n, repeat the two evaluations of y at $x = 1$ using $h = 0.05$ and $h = 0.02$, and empirically determine the order of the method. Is it still a fourth-order method?
(c) Instead of introducing the programming error suggested in part (b), suppose we change the coefficient of k_2 in $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ from 2 to 3. Do you think the method will still be convergent? Explain.

11. (*Rectangular, trapezoidal, and Simpson's rule*) Consider the special case where f in $y' = f$ is a function of x only. Integrating $y' = f(x)$ from x_n to x_{n+1}, we have

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x)\,dx \quad (11.1)$$

If we fit $f(x)$, over $[x_n, x_{n+1}]$, with a *zeroth*-degree polynomial (i.e., a constant) that interpolates f at x_n, then we have $f(x) \approx f(x_n)$, and (11.1) gives $y(x_{n+1}) \approx y(x_n) + f(x_n)h$ and hence the Euler method $y_{n+1} = y_n + f(x_n)h$.

(a) Show that if we fit $f(x)$, over $[x_n, x_{n+1}]$, with a *first*-degree polynomial (a straight line) that interpolates f at x_n and x_{n+1}, then $f(x) \approx f(x_n) + [f(x_{n+1}) - f(x_n)]\,(x - x_n)/h$. Putting that approximation into (11.1), derive the approximation

$$y(x_{n+1}) = y(x_n) + \frac{1}{2}\left[f(x_n) + f(x_{n+1})\right]h, \quad (11.2)$$

and show that (for the special case where f is a function of x only) (11.2) is identical to the second-order Runge–Kutta method (23).
(b) Show that if we fit $f(x)$, over $[x_n, x_{n+1}]$, with a second-degree polynomial (a parabola) that interpolates f at x_n, $x_n + h/2$, and $x_{n+1} = x_n + h$, and put that approximation into (11.1), then one obtains

$$y(x_{n+1}) = y(x_n) + \frac{1}{6}\left[f(x_n) + 4f(x_n + h/2) + f(x_{n+1})\right]h, \quad (11.3)$$

and show that (for the case where f is a function of x only) (11.3) is identical to the fourth-order Runge–Kutta method (25). NOTE: These three results amount to the well-known **rectangular**, **trapezoidal**, and **Simpson's** rules of numerical integration for a single interval of size h. If we sum over all of the intervals, they take the forms

$$\int_a^b f(x)\,dx = \begin{cases} \left[f(a) + f(a+h) + \cdots + f(b)\right]h, \\ \left[f(a) + 2f(a+h) + 2f(a+2h)\right. \\ \qquad \left. + \cdots + 2f(b-h) + f(b)\right]\frac{h}{2}, \\ \left[f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h)\right. \\ \qquad \left. + \cdots + 4f(b-h) + f(b)\right]\frac{h}{6}, \end{cases} \quad (11.4)$$

respectively. In passing from (11.3) to the last line of (11.4) we have replaced $h/2$ by h everywhere in

(11.3), so that the revised (11.3) reads $y_{n+1} = y_n + [f(x_n) + 4f(x_{n+1}) + f(x_{n+2})]\,h/3$, where $x_n = a+nh$. For the rectangular and trapezoidal rules the number of subdivisions, $(b-a)/h$, can be even or odd, but for Simpson's rule it must be even. The order of the error for these integration methods is $O(h)$, $O(h^2)$, and $O(h^4)$, respectively.

12. (a) Using $m = 1$, derive from (30) the Adams–Bashforth method

$$y_{n+1} = y_n + (3f_n - f_{n-1})\,\frac{h}{2}. \tag{12.1}$$

(b) Determine the order of the method (12.1) empirically by using it to solve the test problem (10), at $x = 1$, with two different step sizes, and then using (28).

13. This exercise is to take you through the fourth-order AB–AM predictor-corrector scheme.
(a) For the problem $y' = 2xy^2, y(0) = 1$, compute y_1, y_2, y_3 from the exact solution, with $h = 0.1$, and use those as starting values to determine y_4, by hand, by means of the fourth-order AB–AM predictor-corrector scheme given by (31a) and (33). Apply the corrector three times.
(b) Continuing in the same way, determine y_5.

6.4 Application to Systems and Boundary-Value Problems

The methods developed in the preceding sections are for an initial-value problem with a single first-order differential equation, but what if we have a system of differential equations, a higher-order equation, or a boundary-value problem? In this section we extend the Euler and fourth-order Runge–Kutta methods to these cases.

6.4.1. Systems and higher-order equations. Consider the system of initial-value problems

$$u'(x) = f(x,u,v); \qquad u(a) = u_0 \tag{1a}$$

$$v'(x) = g(x,u,v); \qquad v(a) = v_0 \tag{1b}$$

on $u(x)$ and $v(x)$. To extend Euler's method to such a system, we merely apply it to each of the problems (1a) and (1b) as follows:

$$\begin{aligned} u_{n+1} &= u_n + f(x_n,u_n,v_n)h, \\ v_{n+1} &= v_n + g(x_n,u_n,v_n)h \end{aligned} \tag{2}$$

for $n = 0, 1, 2, \ldots$. Equations (2) are coupled (since each involves both u and v), as were equations (1), but that coupling causes no complication because it occurs in the right-hand sides of the equations, and the values u_n, v_n are already known from the preceding step.

EXAMPLE 1. Consider the system

$$\begin{aligned} u' &= x + v; \qquad u(0) = 0 \\ v' &= uv^2; \qquad v(0) = 1. \end{aligned} \tag{3}$$

The latter looks fairly simple, but it is not. It is nonlinear because of the uv^2 term. Turning to numerical solution using the Euler method (2), let $h = 0.1$, say, and let us go through the first couple of steps. First, $u_0 = 0$ and $v_0 = 1$ from the initial conditions. Then,

$$
\begin{aligned}
n = 0: \\
&u_1 = u_0 + (x_0 + v_0)h = 0 + (0 + 1)(0.1) = \underline{0.1}, \\
&v_1 = v_0 + u_0 v_0^2 h = 1 + (0)(1)^2(0.1) = \underline{1}. \\
n = 1: \\
&u_2 = u_1 + (x_1 + v_1)h = 0.1 + (0.1 + 1)(0.1) = \underline{0.21}, \\
&v_2 = v_1 + u_1 v_1^2 h = 1 + (0.1)(1)^2(0.1) = \underline{1.01},
\end{aligned}
$$

and so on. ∎

Similarly, if the system contains more than two equations.

Next, we show how to adapt the fourth-order Runge–Kutta method to the system (1). Recall that for the single equation

$$
y' = f(x, y); \qquad y(a) = y_0 \tag{4}
$$

the algorithm is

$$
\begin{aligned}
&y_{n+1} = y_n + \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \\
&k_1 = hf(x_n, y_n), \qquad k_2 = hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_1\right), \\
&k_3 = hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_2\right), \quad k_4 = hf\left(x_{n+1}, y_n + k_3\right).
\end{aligned} \tag{5}
$$

For the system (1) it becomes

$$
\begin{aligned}
u_{n+1} &= u_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \\
v_{n+1} &= v_n + \frac{1}{6}\left(l_1 + 2l_2 + 2l_3 + l_4\right),
\end{aligned}
$$

$$
\begin{aligned}
k_1 &= hf(x_n, u_n, v_n), \\
l_1 &= hg(x_n, u_n, v_n), \\
k_2 &= hf\left(x_n + \frac{h}{2}, u_n + \frac{1}{2}k_1, v_n + \frac{1}{2}l_1\right), \\
l_2 &= hg\left(x_n + \frac{h}{2}, u_n + \frac{1}{2}k_1, v_n + \frac{1}{2}l_1\right), \\
k_3 &= hf\left(x_n + \frac{h}{2}, u_n + \frac{1}{2}k_2, v_n + \frac{1}{2}l_2\right),
\end{aligned} \tag{6}
$$

$$l_3 = hg\left(x_n + \frac{h}{2}, u_n + \frac{1}{2}k_2, v_n + \frac{1}{2}l_2\right),$$
$$k_4 = hf\left(x_{n+1}, u_n + k_3, v_n + l_3\right),$$
$$l_4 = hg\left(x_{n+1}, u_n + k_3, v_n + l_3\right),$$

and similarly for systems containing more than two equations.

EXAMPLE 2. Let us illustrate (6) using the same system as in Example 1,

$$\begin{aligned} u' &= x + v; \qquad u(0) = 0 \\ v' &= uv^2; \qquad v(0) = 1. \end{aligned} \tag{7}$$

With $h = 0.1$, say, (6) gives

$n = 0$:

$$k_1 = h(x_0 + v_0) = (0.1)(0 + 1) = 0.1,$$
$$l_1 = hu_0v_0^2 = (0.1)(0)(1)^2 = 0,$$
$$k_2 = h\left[\left(x_0 + \frac{h}{2}\right) + \left(v_0 + \frac{l_1}{2}\right)\right] = (0.1)\left[(0 + 0.05) + (1 + 0)\right] = 0.105,$$
$$l_2 = h\left(u_0 + \frac{k_1}{2}\right)\left(v_0 + \frac{l_1}{2}\right)^2 = (0.1)\,(0 + 0.05)\,(1 + 0)^2 = 0.005,$$
$$k_3 = h\left[\left(x_0 + \frac{h}{2}\right) + \left(v_0 + \frac{l_2}{2}\right)\right]$$
$$= (0.1)\left[(0 + 0.05) + (1 + 0.0025)\right] = 0.10525,$$
$$l_3 = h\left(u_0 + \frac{k_2}{2}\right)\left(v_0 + \frac{l_2}{2}\right)^2$$
$$= (0.1)\,(0 + 0.0525)\,(1 + 0.0025)^2 = 0.005276,$$
$$k_4 = h\left[x_1 + (v_0 + l_3)\right]$$
$$= (0.1)\left[0.1 + (1 + 0.005276)\right] = 0.110528,$$
$$l_4 = h\,(u_0 + k_3)\,(v_0 + l_3)^2$$
$$= (0.1)\,(0 + 0.10525)\,(1 + 0.005276)^2 = 0.010636,$$
$$u_1 = u_0 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$
$$= 0 + \frac{1}{6}\left(0.1 + 0.21 + 0.2105 + 0.110528\right) = \underline{0.105171},$$
$$v_1 = v_0 + \frac{1}{6}\left(l_1 + 2l_2 + 2l_3 + l_4\right)$$
$$= 1 + \frac{1}{6}\left(0 + 0.01 + 0.010552 + 0.010636\right) = \underline{1.005198}.$$

$$
\begin{aligned}
n = 1: \\
&k_1 = 0.110520, \quad l_1 = 0.010627, \\
&k_2 = 0.116051, \quad l_2 = 0.016382, \\
&k_3 = 0.116339, \quad l_3 = 0.016760, \\
&k_4 = 0.122196, \quad l_4 = 0.023134, \\
&u_2 = \underline{0.221420}, \quad v_2 = \underline{1.021872},
\end{aligned}
$$

and so on for $n = 2, 3, \ldots$. We suggest that you fill in the details for the calculation of the $k_1, \ldots, v_2$ values shown above for $n = 1$. ∎

Of course, the idea is to carry out such calculations on a computer, not by hand. The calculations shown in Examples 1 and 2 are merely intended to clarify the methods.

What about higher-order equations? The key is to re-express an nth-order equation as an equivalent system of n first-order equations.

EXAMPLE 3. The problem

$$
y''' - xy'' + y' - 2y^3 = \sin x; \qquad y(1) = 2,\ y'(1) = 0,\ y''(1) = -3 \tag{8}
$$

can be converted to an equivalent system of three first-order equations as follows. Define $y' = u$ and $y'' = v$ (hence $u' = v$). Then (8) can be re-expressed in the form

$$
\begin{aligned}
y' &= u; & y(1) &= 2 \\
u' &= v; & u(1) &= 0 \\
v' &= \sin x + 2y^3 - u + xv; & v(1) &= -3.
\end{aligned} \tag{9a,b,c}
$$

Of the three differential equations in (9), the first two merely serve to introduce the auxiliary dependent variables u and v, and since v' is y''' the third one is a restated version of the given equation $y''' - xy'' + y' - 2y^3 = \sin x$. Equation (9a) is the y equation, so the initial condition is on $y(1)$, namely, $y(1) = 2$, as given in (8). Equation (9b) is the u equation, so the initial condition is on $u(1)$, and we have $u(1) = y'(1) = 0$, from (8). Similarly, for equation (9c).

The system (9) can now be solved by the Euler or fourth-order Runge–Kutta methods or any other such algorithm. To illustrate, let us carry out the first two steps using Euler's method, taking $h = 0.2$, say.

$$
\begin{aligned}
n = 0: \\
&y_1 = y_0 + u_0 h = 2 + (0)(0.2) = \underline{2}, \\
&u_1 = u_0 + v_0 h = 0 + (-3)(0.2) = -0.6, \\
&v_1 = v_0 + \left(\sin x_0 + 2y_0^3 - u_0 + x_0 v_0\right) h \\
&\quad = -3 + \left[\sin 1 + 2(2)^3 - 0 + (1)(-3)\right](0.2) = -0.231706. \\
n = 1: \\
&y_2 = y_1 + u_1 h = 2 + (-0.6)(0.2) = \underline{1.88},
\end{aligned}
$$

$$
\begin{aligned}
u_2 &= u_1 + v_1 h = -0.6 + (-0.231706)(0.2) = -0.646341, \\
v_2 &= v_1 + \left(\sin x_1 + 2y_1^3 - u_1 + x_1 v_1\right) h \\
&= -0.231706 + \left[\sin 1.2 + 2(2)^3 - (-0.6) + (1.2)(-0.231706)\right] (0.2) \\
&= 3.219092,
\end{aligned}
$$

and so on for $n = 2, 3, \ldots$.

COMMENT. Observe that at each step we compute y, u, and v, yet we are not really interested in the auxiliary variables u and v. Perhaps we could just compute $y_1, y_2, \ldots$ and not the u, v values? No; equations (9) are coupled so we need to bring all three variables along together. Of course, we don't need to print or plot u and v, but we do need to compute them. ∎

EXAMPLE 4. Examples 1 and 2 involve a system of first-order equations, and Example 3 involve a single higher-order equation. As a final example, consider a combination of the two such as the initial-value problem

$$
\begin{aligned}
&u'' - 3xuv = \sin x; \quad u(0) = 4, \quad u'(0) = -1 \\
&v'' + 2u - v = 5x; \quad v(0) = 7, \quad v'(0) = 0.
\end{aligned}
\tag{10}
$$

The idea is exactly the same as before. We need to recast (10) as a system of first-order initial value problems. We can do so by introducing auxiliary dependent variables w and z according to $u' = w$ and $v' = z$. Then (10) becomes

$$
\begin{aligned}
u' &= w; & u(0) &= 4 \\
w' &= \sin x + 3xuv; & w(0) &= -1 \\
v' &= z; & v(0) &= 7 \\
z' &= 5x - 2u + v; & z(0) &= 0
\end{aligned}
\tag{11}
$$

which system can now be solved by Euler's method or any other such numerical differential equation solver. ∎

6.4.2. Linear boundary-value problems. Our discussion is based mostly upon the following example.

EXAMPLE 5. Consider the third-order boundary-value problem

$$
y''' - x^2 y = -x^4; \qquad y(0) = 0, \; y'(0) = 0, \; y(2) = 4. \tag{12}
$$

To solve numerically, we begin by recasting (12) as the first-order system:

$$
\begin{aligned}
y' &= u; & y(0) &= 0, \quad y(2) = 4 \\
u' &= v; & u(0) &= 0 \\
v' &= x^2 y - x^4.
\end{aligned}
\tag{13a,b,c}
$$

However, we cannot apply the numerical integration techniques that we have discussed because the problem (13c) does not have an initial condition so we cannot get the solution started. Whereas (13c) is missing an initial condition on v, (13a) has an extra condition – the right end condition $y(2) = 4$, but that condition is of no help in developing a numerical integration scheme that develops a solution beginning at $x = 0$.

Nevertheless, the linearity of (12) saves the day and permits us to work with an initial-value version instead. Specifically, suppose that we solve (numerically) the four initial-value problems

$$\begin{aligned}
&L[Y_1] = 0, \quad && Y_1(0) = 1, \quad Y_1'(0) = 0, \quad Y_1''(0) = 0,\\
&L[Y_2] = 0, \quad && Y_2(0) = 0, \quad Y_2'(0) = 1, \quad Y_2''(0) = 0,\\
&L[Y_3] = 0, \quad && Y_3(0) = 0, \quad Y_3'(0) = 0, \quad Y_3''(0) = 1,\\
&L[Y_p] = -x^4, \quad && Y_p(0) = 0, \quad Y_p'(0) = 0, \quad Y_p''(0) = 0,
\end{aligned} \tag{14}$$

where $L = d^3/dx^3 - x^2$ is the differential operator in (12). The nine initial conditions in the first three of these problems were chosen so as to have a nonzero determinant so that Y_1, Y_2, Y_3 comprise a fundamental set of solutions (i.e., a linearly independent set of solutions) of the homogeneous equation $L[Y] = 0$. The three initial conditions on the particular solution Y_p were chosen as zero for simplicity; any values will do since any particular solution will do. Suppose we imagine that the four initial-value problems in (14) have now been solved by the methods discussed above. Then Y_1, Y_2, Y_3, Y_p are known functions of x over the interval of interest $[0, 2]$, and we have the general solution

$$y(x) = C_1Y_1(x) + C_2Y_2(x) + C_3Y_3(x) + Y_p(x) \tag{15}$$

of $L[y] = -x^4$. Finally, we evaluate the integration constants C_1, C_2, C_3 by imposing the boundary conditions given in (12):

$$\begin{aligned}
y(0) &= 0 = C_1 + 0 + 0 + 0,\\
y'(0) &= 0 = 0 + C_2 + 0 + 0,\\
y(2) &= 4 = C_1Y_1(2) + C_2Y_2(2) + C_3Y_3(2) + Y_p(2).
\end{aligned} \tag{16}$$

Solving (16) gives $C_1 = C_2 = 0$ and $C_3 = [4 - Y_p(2)]/Y_3(2)$, so we have the desired solution of (12) as

$$y(x) = \frac{4 - Y_p(2)}{Y_3(2)}\, Y_3(x) + Y_p(x). \tag{17}$$

In fact, since $C_1 = C_2 = 0$ the functions $Y_1(x)$ and $Y_2(x)$, have dropped out so we don't need to calculate them. All we need are $Y_3(x)$ and $Y_p(x)$, and these are found by the numerical integration of the initial-value problems

$$\begin{aligned}
Y_3' &= U_3, \quad && Y_3(0) = 0,\\
U_3' &= V_3, \quad && U_3(0) = 0,\\
V_3' &= x^2Y_3, \quad && V_3(0) = 1,
\end{aligned} \tag{18}$$

and

$$\begin{aligned} Y_p' &= U_p, & Y_p(0) &= 0, \\ U_p' &= V_p, & U_p(0) &= 0, \\ V_p' &= x^2 Y_p - x^4, & V_p(0) &= 0, \end{aligned} \tag{19}$$

respectively.

COMMENT. Remember that whereas initial-value problems have unique solutions (if the functions involved are sufficiently well behaved), boundary-value problems can have no solution, a unique solution, or even an infinite number of solutions. How do these possibilities work out in this example? The clue is that (17) fails if $Y_3(2)$ turns out to be zero. The situation is seen more clearly from (16), where all of the possibilities come into view. Specifically, if $Y_3(2) \neq 0$, then we can solve uniquely for C_3, and we have a unique solution, given by (17). If $Y_3(2)$ does vanish, then there are two possiblities as seen from (16): if $Y_p(2) \neq 4$, then there is no solution, and if $Y_p(2) = 4$ then there are an infinite number of solutions of (12), namely,

$$y(x) = C_3 Y_3(x) + Y_p(x), \tag{20}$$

where C_3 remains arbitrary. ∎

We see that boundary-value problems are more difficult than initial-value problems. From Example 5 we see that a nonhomogeneous nth-order linear boundary-value problem generally involves the solution of $n + 1$ initial-value problems, although in Example 5 (in which $n = 3$) we were lucky and did not need to solve for two of the four unknowns, Y_1 and Y_2.

Nonlinear boundary-value problems are more difficult still, because we cannot use the idea of finding a fundamental set of solutions plus a particular solution and thus forming a general solution, as we did in Example 5, and which idea is based upon linearity. One viable line of approach comes under the heading of **shooting methods**. For instance, to solve the nonlinear boundary-value problem

$$y'' + \sin y = 3x; \qquad y(0) = 0,\ y(5) = 2 \tag{21}$$

we can solve the initial-value problem

$$\begin{aligned} y' &= u, & y(0) &= 0 \\ u' &= 3x - \sin y, & u(0) &= u_0 \end{aligned} \tag{22}$$

iteratively. That is, we can guess at the initial condition u_0 [which is the initial slope $y'(0)$] and solve (22) for y(x) and u(x). Next, we compare the computed value of $y(5)$ with the boundary condition $y(5) = 2$ (which we have not yet used). If the computed value is too high, then we return to (22), reduce the value of u_0, and solve again. Comparing the new computed value of $y(5)$ with the prescribed value $y(5) = 2$, we again revise our value of u_0. If these revisions are done in

a rational way, one can imagine obtaining a convergent scheme. Such a scheme is called a *shooting method* because of the obvious analogy with the shooting of a projectile, with the intention of having the projectile strike the ground at some distant prescribed point.

Thus, we can see the increase in difficulty as we move away from linear initial-value problems. For a linear boundary-value problem of order n we need to solve not one problem but $n+1$ of them. For a nonlinear boundary-value problem we need to solve an infinite sequence of them, in principle; in practice, we need to carry out only enough iterations to produce the desired accuracy.

Closure. In Section 6.4.1 we extend the Euler and fourth-order Runge–Kutta solution methods to cover systems of equations and higher-order equations. In that discussion it is more convenient to use n-dimensional vector notation because of its compactness, but that notation is not be introduced until Chapters 9 and 10. Nonetheless, let us indicate the result, if only for the Euler method, for future reference. The idea is that we can express the system

$$\begin{aligned} y_1'(x) &= f_1\left(x, y_1(x), \ldots, y_n(x)\right); \quad y_1(a) = y_{10}, \\ &\vdots \qquad\qquad\qquad\qquad\quad \vdots \\ y_n'(x) &= f_n\left(x, y_1(x), \ldots, y_n(x)\right); \quad y_n(a) = y_{n0}, \end{aligned} \tag{23}$$

in the vector form

$$\mathbf{y}'(x) = \mathbf{f}\left(x, \mathbf{y}(x)\right); \qquad \mathbf{y}(a) = \mathbf{y_0}, \tag{24}$$

where the boldface letters denote "n-dimensional column vectors:"

$$\mathbf{y}(x) = \begin{bmatrix} y_1(x) \\ \vdots \\ y_n(x) \end{bmatrix}; \quad \mathbf{y}'(x) = \begin{bmatrix} y_1'(x) \\ \vdots \\ y_n'(x) \end{bmatrix}, \quad \mathbf{f}(x, \mathbf{y}(x)) = \begin{bmatrix} f_1(x, \mathbf{y}(x)) \\ \vdots \\ f_n(x, \mathbf{y}(x)) \end{bmatrix}, \tag{25}$$

and where $f_j(x, \mathbf{y}(x))$ is simply a shorthand notation for $f_j(x, y_1(x), \ldots, y_n(x))$. Then the Euler algorithm corresponding to (24) is

$$\mathbf{y_{n+1}} = \mathbf{y_n} + \mathbf{f}\left(x_n, \mathbf{y_n}\right) h. \tag{26}$$

In Section 6.4.2 we turn to boundary-value problems, but only linear ones. Given an nth-order linear boundary-value problem $L[y] = f(x)$ on an interval $[a, b]$ plus n boundary conditions the idea is to solve the problems

$$\begin{aligned} &L\left[Y_1\right] = 0; \quad Y_1(a) = 1, \; Y_1'(a) = \cdots = Y_1^{(n-1)}(a) = 0, \\ &\qquad\vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ &L\left[Y_n\right] = 0; \quad Y_n(a) = Y_n'(a) = \cdots = Y_n^{(n-2)}(a) = 0, \; Y_n^{(n-1)}(a) = 1, \\ &L\left[Y_p\right] = f; \quad Y_p(a) = \cdots = Y_p^{(n-1)}(a) = 0 \end{aligned} \tag{27}$$

for $Y_1(x), \ldots, Y_n(x), Y_p(x)$ and to form the general solution as

$$y(x) = C_1 Y_1(x) + \cdots + C_n Y_n(x) + Y_p(x). \tag{28}$$

Finally, application of the original boundary conditions to (28) yields n linear algebraic equations for $C_1, \ldots, C_n$, which equations will have a unique solution, no solution, or an infinity of solutions.

Computer software. No new software is needed for the methods described in this section. For instance, we can use the *Maple* command dsolve, with the numeric option, to solve the problem

$$u' = x + v; \quad u(0) = 0$$
$$v' = -5uv; \quad v(0) = 1$$

and to print the results at $x = 1, 2$, and 3. First, enter

with(DEtools):

and return. Then enter

dsolve({diff$(u(x), x) = x + v(x)$, diff$(v(x), x) = -5 * u(x) * v(x)$,
$u(0) = 0$, $v(0) = 1$}, $\{u(x), v(x)\}$, type = numeric,
value = array ([1, 2, 3]));

and return. The printed result is

$$\left[\left[\begin{array}{cc} & [x, u(x), v(x)] \\ 1. & 1.032499017614234 \quad .07285274036469075 \\ 2. & 2.544584704578166 \quad .000014134883458 36790 \\ 3. & 5.044585755162072 \quad -.3131443346304622 \times 10^{-9} \end{array}\right]\right]$$

The only differences between the command above and the one given at the end of Section 6.3 is that here we have entered two differential equations, two initial conditions, and two dependent variables, and we have omitted the abserr option.

Observe that to solve a differential equation, or system of differential equations, numerically, we must first express the equations as a system of first-order equations, as illustrated in Example 4. However, to use the *Maple* dsolve command we can leave the original higher-order equations intact.

EXERCISES 6.4

1. In Example 2 we gave $k_1, l_1, k_2, l_2, k_3, l_3, k_4, l_4$ for $n = 1$, and the resulting values of u_2 and v_2, but did not show the calculations. Provide those calculations, as we did for the step $n = 0$.

2. As we did in Example 1, work out y_1, z_1, by hand. Use three methods: Euler, second-order Runge–Kutta, and fourth-order Runge–Kutta, and take $h = 0.2$. These problems are rigged so as to have simple closed-form solutions, some of which are given in brackets. Compare your results with the exact solution.

(a) $y' = z; \quad y(0) = 1$
$z' = -y; \quad z(0) = 0$

(b) $y' = 4z; \quad y(2) = 5$
$z' = -y; \quad z(2) = 0$

(c) $y' = -z^2/y; \quad y(0) = 1 \quad [y(x) = e^{-x}]$
$z' = -y; \quad z(0) = 1 \quad [z(x) = e^{-x}]$

(d) $y' = 2xz^2/y; \quad y(1) = 1 \quad [y(x) = x^2]$
$z' = y/z^2; \quad z(1) = 1 \quad [z(x) = x]$

(e) $y' = (x + y)z - 1; \quad y(1) = 1 \quad [y(x) = x]$
$z' = -yz^3; \quad z(1) = 1 \quad [z(x) = 1/x]$

3. (a)–(e) First, read Exercise 2. Use computer software to solve the initial-value problem given in the corresponding part of Exercise 2, for $y(x)$ and $z(x)$ at $x = 3, 5$, and 10, and compare those results with the exact solution at those points.

4. (a) Just as (2) and (6) give the Euler and fourth-order Runge–Kutta algorithms for the second-order system (1), write down the analogous Euler, second-order Runge–Kutta, and fourth-order Runge–Kutta algorithms for the third-order system

$$\begin{aligned} x'(t) &= F(t, x, y, z), \quad x(a) = x_0 \\ y'(t) &= G(t, x, y, z), \quad y(a) = y_0 \\ z'(t) &= H(t, x, y, z), \quad z(a) = z_0. \end{aligned} \tag{4.1}$$

Use the Euler and second-order Runge–Kutta algorithms to work out x_1, y_1, z_1 and x_2, y_2, z_2, by hand, for the case where F, G, H are $y - 1, z, t + x + 3(z - y + 1)$, respectively, with the initial conditions $x(0) = -3$, $y(0) = 0$, $z(0) = 2$ using $h = 0.3$.

(b) Same as (a), but with $x(0) = y(0) = z(0) = 0$.

(c) Same as (a), but with $x(0) = 1, y(0) = 0, z(0) = 0$

(d) Same as (a), but with $x(0) = y(0) = 0, z(0) = 10$.

5. We re-expressed (8) and (10) as the equivalent systems of first-order initial-value problems (9) and (11), respectively. Do the same for the problem given. You need go no further.

(a) $mx'' + cx' + kx = f(t); \quad x(0) = x_0, \quad x'(0) = x_0'$

(b) $Li'' + Ri' + (1/C)i = E'(t); \quad i(0) = i_0, \quad i'(0) = i_0'$

(c) $y'' - xyy' = \sin x; \quad y(1) = 5, \quad y'(1) = -1$

(d) $y'' + y' - 4y = 3x; \quad y(0) = 2, \quad y'(0) = 1$

(e) $y''' - 2\sin y' = 3x; \quad y(-2) = 7, \quad y'(-2) = 4$
$y''(-2) = 0$

(f) $y''' + xy = \cos 2x; \quad y(1) = 3, \quad y'(1) = 2, \quad y''(1) = 0$

(g) $x'' + 2x - 3y = 10\cos 3t; \quad x(0) = 2, \quad x'(0) = -1$
$y'' - x + 5y = 0; \quad y(0) = 4, y'(0) = 3$

(h) $y''' + xy'z = f(x); \quad y(3) = 2, \quad y'(3) = -1$
$y''(3) = 6$
$z'' + y - z = g(x); \quad z(3) = 0, z'(3) = 8$

(i) $x''' - 3xz = \sin t; \quad x(1) = x'(1) = 0, \quad x''(1) = 3$
$y' + 2x + y - 5z = 0; \quad y(1) = 6$
$z' - 4xy = e^{2t}; \quad z(1) = 2$

(j) $y''' = yz; \quad y(0) = 1, \quad y'(0) = y''(0) = 0$
$z'' = -xy + z; \quad z(0) = 5, z'(0) = 1$

6. Use computer software to solve the given system numerically, and print out the solution for $y(x)$ and $z(x)$ at $x = 1, 2$.

(a) $y'' - 2xz' = 5x; \quad y(0) = 2, \quad y'(0) = -1$
$z' + yz^3 = -3; \quad z(0) = 1$

(b) $y' + 3xz = x^2; \quad y(0) = 1$
$z''' - y^2z' + z = 0; \quad z(0) = z'(0) = 2, \quad z''(0) = -1$

(c) $y' = z' + x; \quad y(0) = 1$
$z'' + y^2z^2 = 3; \quad z(0) = -1, \quad z'(0) = 0$

(d) $y' - z'' = x; \quad y(0) = -1$
$z''' - y^2z = 3; \quad z(0) = 1, \quad z'(0) = z''(0) = 0$

7. Complete the solution of Example 5 by using computer software to solve (18) for $Y_3(x)$ and (19) for $Y_p(x)$, at $x = 0.5$ and 1, and then using (17) to determine $y(x)$ at those points.

8. Use the method explained in Example 5 to reduce the given linear boundary-value problem to a system of linear initial-value problems. Then complete the solution and solve for the specified quantity, either using computer software or by programming any of the numerical solution methods that we have studied. Obtain accuracy to five significant figures, and indicate why you believe that you have achieved that accuracy.

If you believe that there is no solution or that it exists but is nonunique, then give your reasoning. HINT: You can specify homogeneous initial conditions for the Y_p problem, as we did in Example 5, but be aware that you do not *have* to use homogeneous conditions, and that you may be able to reduce your labor by a more optimal choice of those conditions.

(a) $y'' - 2xy' + y = 3\sin x; \quad y(0) = 1, \quad y(2) = 3.$
Determine $y(1)$.

(b) $y'' + (\cos x)y = 0; \quad y(0) = 1, \quad y(10) = 2.$
Determine $y(2)$.

(c) $y'' - [\ln(x+1)]y' - y = 2\sin 3x + 1;$
$y(0) = 3, \quad y(2) = -1.$
Determine $y(x)$ at $x = 0.5, 1.0, 1.5$.

(d) $y'' + y' - xy = x^3; \quad y(0) = 1, \quad y(5) = 2.$
Determine $y(x)$ at $x = 1, 2, 3, 4$.

(e) $y''' + xy = 2x^3; \quad y(1) = y'(1) = 0, \quad y''(2) = -3.$
Determine $y(2)$.

(f) $y''' + xy' + y = x; \quad y(1) = 2, \quad y'(1) = 0.4, \quad y'(5) = 3.$
Determine $y(x)$ at $x = 4, 5$.

6.5 Stability and Difference Equations

6.5.1. Introduction. In progressing from the simple Euler method to the more sophisticated higher-order methods our aim was improvement in accuracy. However, there are cases where the results obtained not only fail to be sufficiently accurate but are grossly incorrect, as illustrated in the two examples to follow. The second one introduces the idea of stability, and in Section 6.5.2 we concentrate on that topic.

EXAMPLE 1. The initial-value problem

$$y' - 2y = -6e^{-4x}; \quad y(0) = 1 \tag{1}$$

has the exact solution $y(x) = \exp(-4x)$. If we solve it by the fourth-order Runge–Kutta method for the step sizes $h = 0.1, 0.05$, and 0.01, we obtain in Table 1 the results shown, at

Table 1. Runge–Kutta solution of (1).

x	$h = 0.1$	$h = 0.05$	$h = 0.01$	Exact
1	0.179006 E−1	0.182893 E−1	0.183153 E−1	0.183156 E−1
4	−0.167842 E+0	−0.106538 E−1	−0.146405 E−3	0.112535 E−6
8	−0.500286 E+3	−0.317586 E+2	−0.436763 E+0	0.126642 E−13
12	−0.149120 E+7	−0.946704 E+5	−0.130197 E+4	0.142516 E−20

the representative points $x = 1, 4, 8$, and 12. Since the Runge–Kutta method is convergent, the results should converge to the exact solution at any given x as h tends to zero, but that convergence is hard to see in the tabulated results except for $x = 1$. In fact, it is doubtful that we could ever come close to the exact values at $x = 8$ or 12 since we might need to make h so small that roundoff errors might come to dominate before the accumulated truncation error is sufficiently reduced.

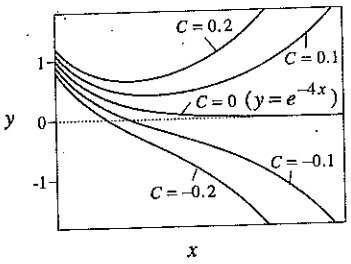


Figure 1. Solution curves for the equation $y' - 2y = -6e^{-4x}$.

More central to the purpose of this example is to see that with h fixed the results diverge dramatically from the exact solution as x increases so as to become grossly incorrect. We cannot blame this strange and unexpected result on complications due to nonlinearity because (1) is linear.

To understand the source of the difficulty, note that the general solution of the differential equation is $y(x) = \exp(-4x) + C\exp(2x)$, where C is an arbitrary constant. The initial condition implies that $C = 0$, leaving the particular solution $y(x) = \exp(-4x)$. In Fig. 1 we show several solution curves for values of C close to and equal to zero, and we can see the rapid divergence of neighboring curves from the solution $y(x) = \exp(-4x)$. Thus, the explanation of the difficulties found in the tabulated numerical results is that even a very small numerical error shifts us from the exact solution curve to a neighboring curve, which then diverges from the true solution. ∎

EXAMPLE 2. In Example 3 of Section 6.3 we solved the equation $y' = -y$, with initial condition $y(0) = 1$, by several methods – from the simple Euler method to the more accurate and sophisticated fourth-order Runge–Kutta method, and we gave the results in Table 2. Since the midpoint rule and the second-order Runge–Kutta methods are both of second order we expected their accuracy to be comparable. Indeed they were initially, but the midpoint rule eventually developed an error that oscillated in sign from step to step and grew in magnitude (see Table 2 in Section 6.3). Let us solve the similar problem

$$y' = -2y; \quad y(0) = 1 \tag{2}$$

by the midpoint rule, with $h = 0.05$. Since the midpoint rule is not self-starting, we use ten Euler steps from $x = 0$ to $x = 0.05$ before switching over to the midpoint rule. We have plotted the results in Fig. 2, along with the exact solution, $y(x) = \exp(-2x)$. Once again, we see that the midpoint rule results follow the exact solution initially, but they develop an error that oscillates in sign and grows such that the results are soon hopelessly incorrect.

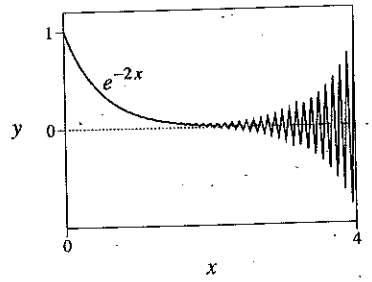


Figure 2. Illustration of numerical instability associated with the midpoint rule, for the initial-value problem (2).

This numerical difficulty is different from the one found above in Example 1, for rather than being due to an extreme sensitivity to initial conditions, it is associated with machine roundoff error and is an example of numerical instability. ∎

6.5.2. Stability. Let us analyze the phenomenon of numerical instability that we encountered in Example 2. Recall that we denote the exact solution of a given initial-value problem as $y(x_n)$ and the numerical solution as y_n. Actually, the latter is not quite the same as the computer printout because of the inevitable presence of machine roundoff errors. Thus, let us distinguish further between the numerical solution y_n that would be generated on a perfect computer, and the solution y_n^* that is generated on a real machine and which includes the effects of numerical roundoff – that is, the truncation of numbers after a certain number of significant figures.

It is useful to decompose the total error, at any nth step, as

$$\begin{aligned} \text{Total error} &= y(x_n) - y_n^* = [y(x_n) - y_n] + [y_n - y_n^*] \\ &= [\text{accum. truncation error}] + [\text{accum. roundoff error}]. \end{aligned} \tag{3}$$

We ask two things of a method: first, that the accumulated truncation error tend to zero at any fixed x as the step size h tends to zero and, second, that the accumulated roundoff error remain small compared to the exact solution. The first is the issue of **convergence**, discussed earlier in this chapter, and the second is the issue of **stability**, our present concern.

We have already noted that the midpoint rule can produce the strange behavior shown in Fig. 2, so let us study the application of that method to the standard "test problem,"

$$y' = Ay; \quad y(0) = 1, \tag{4}$$

where it is useful to include the constant A as a parameter. The midpoint rule generates y_n according to the algorithm

$$\begin{aligned} y_{n+1} &= y_{n-1} + 2hf\,(x_n, y_n) \\ &= y_{n-1} + 2Ay_n; \qquad y_0 = 1 \end{aligned} \tag{5}$$

for $n = 0, 1, 2, \ldots$.

To determine whether a solution algorithm, in this case (5), is stable, it is customary to "inject" a roundoff error at any step in the solution, say at $n = 0$, and to see how much the perturbed solution differs from the exact solution as n increases, assuming that no further roundoff errors occur. Thus, in place of (5), consider the perturbed problem

$$y^*_{n+1} = y^*_{n-1} + 2Ahy^*_n; \quad y^*_0 = 1 - \epsilon, \tag{6}$$

say, where ϵ is the (positive or negative) roundoff error in the initial condition. Defining the error $e_n \equiv y_n - y^*_n$ and subtracting (6) from (5), gives

$$e_{n+1} = e_{n-1} + 2Ahe_n, \tag{7}$$

with the initial condition $e_0 = \epsilon$, as governing the evolution of e_n. We call (7) a **difference equation**. Just as certain differential equations can be solved by seeking solutions in the form $y(x) = e^{\lambda x}$, the appropriate form for the difference equation (7) is

$$e_n = \rho^n, \tag{8}$$

where ρ is to be determined. Putting this expression into (7) gives

$$\rho^{n+1} - 2Ah\rho^n - \rho^{n-1} = 0 \tag{9}$$

or

$$\left(\rho^2 - 2Ah\rho - 1\right)\frac{1}{\rho^n} = 0. \tag{10}$$

Since $1/\rho^n$ is not zero, it follows from (10) that we must have $\rho^2 - 2Ah\rho - 1 = 0$, so we have the two roots

$$\rho = Ah + \sqrt{1 + A^2h^2} \quad \text{and} \quad \rho = Ah - \sqrt{1 + A^2h^2}. \tag{11}$$

By considerations analogous to those for differential equations, we have

$$e_n = C_1\left(Ah + \sqrt{1 + A^2h^2}\right)^n + C_2\left(Ah - \sqrt{1 + A^2h^2}\right)^n \tag{12}$$

as the general solution of (7).

If we let $h \to 0$, then

$$\left(Ah + \sqrt{1 + A^2h^2}\right)^n \sim (Ah + 1)^n = e^{n \ln (1+Ah)} \sim e^{nAh} = e^{Ax_n}, \tag{13}$$

where we have used the identity $a = e^{\ln a}$, the Taylor expansion $\ln (1 + x) = x - x^2/2 + \cdots \sim x$, and the fact that $x_n = nh$. Similarly,

$$\begin{aligned}\left(Ah - \sqrt{1 + A^2h^2}\right)^n &\sim (Ah - 1)^n \\ &= (-1)^n(1 - Ah)^n = (-1)^n e^{n \ln (1-Ah)} \\ &\sim (-1)^n e^{-nAh} = (-1)^n e^{-Ax_n},\end{aligned} \tag{14}$$

so (12) becomes

$$e_n = C_1 e^{Ax_n} + C_2(-1)^n e^{-Ax_n} \tag{15}$$

as $h \to 0$. Since there are two arbitrary constants, C_1 and C_2, two initial conditions are appropriate, whereas we have attached only the single condition $e_0 = \epsilon$ in (7). With no great loss of generality let us specify as a second initial condition $e_1 = 0$. Imposing these conditions on (15), we have

$$\begin{aligned}e_0 &= \epsilon = C_1 + C_2, \\ e_1 &= 0 = C_1 e^{4x_1} - C_2 e^{-4x_1}.\end{aligned}$$

Finally, solving for C_1 and C_2 and inserting these values into (15), gives

$$e_n = \frac{\epsilon}{2 \cosh Ax_1}\left[e^{A(x_n - x_1)} + (-1)^n e^{-A(x_n - x_1)}\right]. \tag{16}$$

To infer from (16) whether the method is stable or not, we consider the cases $A > 0$ and $A < 0$ separately. If $A > 0$, then the second term in (16) decays to zero, and even though the first term grows exponentially, it remains small compared to the exact solution $y(x_n) = \exp (Ax_n)$ as n increases because ϵ is very small (for example, on the order of 10^{-10}). We conclude, formally, that if $A > 0$ then the midpoint rule is stable.

On the other hand, if $A < 0$, then the second term starts out quite small, due to the ϵ factor, but grows exponentially with x_n and oscillates due to the $(-1)^n$, whereas the exact solution is $\exp (-x)$. This is precisely the sort of behavior that was observed in Example 2 (where A was -2), and we conclude that if $A < 0$, then the midpoint rule is unstable.

Since the stability of the midpoint rule depends upon the sign of A in the test equation $y' = Ay$ (stability for $A > 0$ and instability for $A < 0$), we say that the

midpoint rule is only **weakly stable**. If, instead, a method is stable independent of the sign of A, then we classify it as **strongly stable**.

Having found that the midpoint rule when applied to the equation $y' = Ay$ is stable for $A > 0$ and unstable for $A < 0$, what about the stability of the midpoint rule when it is applied to an equation $y' = f(x, y)$ that is more complicated? Observing that A is the partial derivative of Ay (i.e., the right-hand side of $y' = Ay$) with respect to y, we expect, as a rule of thumb, the midpoint rule to be stable if $\partial f/\partial y > 0$ and unstable if $\partial f/\partial y < 0$ over the x, y domain of interest. For instance, if $y' = e^{xy}$ on $x > 0$, then we can expect the midpoint rule to be stable because $\partial(e^{xy})/\partial y = xe^{xy} > 0$ on $x > 0$, but if $y' = e^{-xy}$ on $x > 0$, then we can expect the midpoint rule to be unstable on $x > 0$ because $\partial(e^{-xy})/\partial y = -xe^{-xy} < 0$ on $x > 0$.

Besides arriving at the above-stated conclusions as to the stability of the midpoint rule for the test equation $y' = Ay$, we can now understand the origin of the instability, for notice that the difference equations $y_{n+1} - 2Ahy_n - y_{n-1} = 0$ and $e_{n+1} - 2Ahe_n - e_{n-1} = 0$, governing y_n and e_n, are identical. Thus, analogous to (15) we must have

$$y_n = B_1 e^{Ax_n} + B_2(-1)^n e^{-Ax_n} \tag{17}$$

for arbitrary constants B_1, B_2, as h tends to zero. The first of these terms coincides with the exact solution of the original equation $y' = Ay$, and the second term (which gives rise to the instability if $A < 0$) is an extraneous solution that enters because we have replaced the original first-order differential equation by a second-order difference equation (second-order because the difference between the subscripts $n+1$ and $n-1$ is 2). Single-step methods (e.g., Euler and Runge–Kutta) are strongly stable (i.e., independent of the sign of A) because the resulting difference equation is only first order so there are no extraneous solutions. Thus, we can finally see why, in Example 3 of Section 6.3, the midpoint rule proved unstable but the other methods were stable.

Understand that these stability claims are based upon analyses in which we let h tend to zero, whereas in practice h is, of course, finite. To illustrate what can happen as h is varied, let us solve

$$y' = -1000(y - x^3) + 3x^2; \quad y(0) = 0 \tag{18}$$

by Euler's method. The exact solution is simply $y(x) = x^3$ so that at $x = 1$, for instance, we have $y(1) = 1$. By comparison, the values computed by Euler's method are as given in Table 2.

Even from this limited data we can see that we do have the stability claimed above for the single-step Euler method, but only when h is made sufficiently small. To understand this behavior, consider the relevant test equation $y' = -1000y$, namely, $y' = Ay$, where $A = \partial[-1000(y - x^3) + 3x^2]/\partial y = -1000$. Then Euler's method for that test equation is $y_{n+1} = y_n - 1000hy_n$. Similarly, $y^*_{n+1} = y^*_n - 1000hy^*_n$. Subtracting these two equations, we find that the roundoff error $e_n = y_n - y^*_n$ satisfies the simple difference equation

$$e_{n+1} = (1 - 1000h)e_n. \tag{19}$$

Table 2. Finite-h stability.

h	Computed $y(1)$
0.2500	2.3737566×10^{5}
0.1000	8.7725049×10^{14}
0.0100	Exponential overflow
0.0010	0.99999726
0.0001	0.99999970

Letting $n = 0, 1, 2, \ldots$ in (19) reveals that the solution of (19) is

$$e_n = (1 - 1000h)^n e_0, \tag{20}$$

where e_0 is the initial roundoff error. If we take the limit as $h \to 0$, then

$$e_n = (1 - 1000h)^n e_0 = e_0 e^{n \ln(1-1000h)} \sim e_0 e^{-1000nh} = e_0 e^{-1000x_n}, \tag{21}$$

which is small compared to the exact solution $y_n = e^{-1000x_n}$ because of the e_0 factor, so the method is stable. This result is in agreement with the numerical results given in Table 2: as $h \to 0$ the scheme is stable. However, in a real calculation h is, of course, finite and it appears, from the tabulation that there is some critical value, say h_{cr}, such that the guaranteed stability is realized only if $h < h_{cr}$. To see this, let us retain (20) rather than let $h \to 0$. It is seen from (20) that if $|1 - 1000h| < 1$, then $e_n \to 0$ as $n \to \infty$, and if $|1 - 1000h| > 1$, then $e_n \to \infty$ as $n \to \infty$. Thus, for stability we need $|1 - 1000h| < 1$ or $-1 < 1 - 1000h < 1$. The right-hand inequality imposes no restriction on h because it is true for all h's (provided that h is positive, as is normally the case), and the left-hand inequality is true only for $h < 0.002$. Hence $h_{cr} = 0.002$ in this example, and this result is consistent with the tabulated results, which show instability for the h's greater than that value, and stability for the h's smaller than that value. Thus, when we say that the Euler method is strongly stable, what we should really say is that it is strongly stable for *sufficiently small* h. Likewise for the Runge–Kutta and other single-step methods.

6.5.3. Difference equations. (Optional) Difference equations are important in their own right, and the purpose of this Section 6.5.3 is not only to clarify some of the steps in Section 6.5.2, but also to take this opportunity to present some basics regarding the theory and solution of such equations.

To begin, we define a **difference equation of order** N as a relation involving $y_n, y_{n+1}, \ldots, y_{n+N}$. As we have seen, one way in which difference equations arise is in the numerical solution of differential equations. For instance, if we discretize the differential equation $y' = -y$ and solve by Euler's method or the midpoint rule, then in place of the differential equation we have the first- and second-order difference equations $y_{n+1} = y_n - hy_n = (1 - h)y_n$ and $y_{n+1} = y_{n-1} - 2hy_n$, or

$$y_{n+1} - (1 - h)y_n = 0 \tag{22}$$

and

$$y_{n+1} + 2hy_n - y_{n-1} = 0, \tag{23}$$

respectively. In case it is not clear that (23) is of second order, we could let $n-1 = m$ and obtain $y_{m+2} + 2hy_{m+1} - y_m = 0$ instead, which equation is more clearly of second order. That is, the order is always the difference between the largest and smallest subscripted indices.

Analogous to differential equation terminology, we say that (22) and (23) are **linear** because they are of the form

$$a_0(n)y_{n+N} + a_1(n)y_{n+N-1} + \cdots + a_N(n)y_n = f(n), \tag{24}$$

homogeneous because $f(n)$ is zero in each case, and of **constant-coefficient** type because their a_j's are constants rather than functions of n. By a solution of (24) is meant any sequence y_n that reduces (24) to a numerical identity for each n under consideration, such as $n = 0, 1, 2, \ldots$.

The theory of difference equations is analogous to that of differential equations. For instance, just as one seeks solutions to a linear homogeneous differential equation with constant coefficients in the form $y(x) = e^{\lambda x}$, one seeks solutions to a linear homogeneous difference equation with constant coefficients in the form

$$y_n = \rho^n \tag{25}$$

as we did in Section 6.5.2. [In case these forms don't seem analogous, observe that $e^{\lambda x} = (e^{\lambda})^x$ is a constant to the power x, just as ρ^n is a constant to the power n.] Putting (25) into such an Nth-order difference equation gives an Nth-degree polynomial equation on ρ, the characteristic equation corresponding to the given difference equation, and if the N roots $(\rho_1, \ldots, \rho_N)$ are distinct, then

$$y_n = C_1\rho_1^n + \cdots + C_N\rho_2^N, \tag{26}$$

where the C_j's are arbitrary constants, can be shown to be a **general solution** of the difference equation in the sense that every solution is of the form (26) for some specific choice of the C_j's. For an Nth-order linear differential equation, N initial conditions (y and its first $N-1$ derivatives at the initial point) are appropriate for narrowing a general solution down to a particular solution. Likewise for a linear difference equation N initial conditions are appropriate – namely, the first N values $y_0, y_1, \ldots, y_{N-1}$.

EXAMPLE 3. Solve the difference equation

$$y_{n+1} - 4y_n = 0. \tag{27}$$

Since (27) is linear, homogeneous and of constant-coefficient type, seek solutions in the form (25). Putting that form into (27) gives

$$\rho^{n+1} - 4\rho^n = (\rho - 4)\rho^n = 0 \tag{28}$$

so that if $\rho \neq 0$ then $\rho - 4 = 0$, $\rho = 4$, and the general solution of (27) is

$$y_n = C(4)^n. \tag{29}$$

For example, if an initial condition $y_0 = 3$ is specified, then $y_0 = 3 = C(4)^0 = C$ gives $C = 3$ and hence the particular solution $y_n = 3(4)^n$.

Actually, (29) is simple enough to solve more directly since (for $n = 0, 1, \ldots$) it gives $y_1 = 4y_0$, $y_2 = 4y_1 = 4^2 y_0$, $y_3 = 4y_2 = 4^3 y_0$, and so on, so one can see that $y_n = y_0(4)^n$ or, if y_0 is not specified, $y_n = C(4)^n$, which is the same as (29). ∎

EXAMPLE 4. Solve the difference equation

$$y_{n+2} - y_{n+1} - 6y_n = 0. \tag{30}$$

Seeking solutions in the form (25) gives the characteristic equation $\rho^2 - \rho - 6 = 0$ with roots -2 and 3 so the general solution of (30) is

$$y_n = C_1(-2)^n + C_2(3)^n. \tag{31}$$

If initial conditions are prescribed, say $y_0 = 4$ and $y_1 = -13$, then

$$\begin{aligned} y_0 &= 4 = C_1 + C_2, \\ y_1 &= -13 = -2C_1 + 3C_2 \end{aligned}$$

give $C_1 = 5$ and $C_2 = -1$. ∎

If the characteristic polynomial has a pair of complex conjugate roots, say $\rho_1 = \alpha + i\beta$ and $\rho_2 = \alpha - i\beta$, then the solution can still be expressed in real form, for if we express ρ_1 and ρ_2 in polar form (as is explained in Section 22.2 on complex numbers) as

$$\rho_1 = r\, e^{i\theta} \quad \text{and} \quad \rho_2 = r\, e^{-i\theta}, \tag{32}$$

where $r = \sqrt{\alpha^2 + \beta^2}$ and $\theta = \tan^{-1}(\beta/\alpha)$, then

$$\begin{aligned} C_1\rho_1^n + C_2\rho_2^n &= C_1 r^n e^{in\theta} + C_2 r^n e^{-in\theta} \\ &= r^n \left(C_1 e^{in\theta} + C_2 e^{-in\theta} \right) \\ &= r^n \left[C_1 (\cos n\theta + i \sin n\theta) + C_2 (\cos n\theta - i \sin n\theta) \right] \\ &= r^n (C_3 \cos n\theta + C_4 \sin n\theta), \end{aligned} \tag{33}$$

where $C_3 = C_1 + C_2$ and $C_4 = i(C_1 - C_2)$ are arbitrary constants.

EXAMPLE 5. Solve the difference equation

$$y_{n+2} + 4y_n = 0. \tag{34}$$

The characteristic roots are $\pm 2i$ so (32) becomes $\rho_1 = 2e^{i\pi/2}$ and $\rho_2 = 2e^{-i\pi/2}$, and (33) gives the general solution

$$y_n = 2^n \left(A \cos \frac{n\pi}{2} + B \sin \frac{n\pi}{2} \right), \tag{35}$$

where A, B are arbitrary constants. ∎

As we have seen, one way in which difference equations arise is in the numerical solution of differential equations, wherein the continuous process (described by the differential equation) is approximated by a discrete one (described by the difference equation). However, they also arise directly in modeling discrete processes. To illustrate, let p_n be the principal in a savings account at the end of the nth year. We say that the process is discrete in that p is a function of the integer variable n rather than a continuous time variable t. If the account earns an annual interest of I percent, then the growth of principal from year to year is governed by the difference equation

$$p_{n+1} = \left(1 + \frac{I}{100} \right) p_n, \tag{36}$$

which is of the same form as (22).

In fact, discrete processes governed by nonlinear difference equations are part of the modern theory of *dynamical systems*, in which theory the phenomenon of chaos plays a prominent role. Let us close with a brief mention of one such discrete process that is familiar to those who study dynamical systems and chaos. Let x_n, y_n be a point in a Cartesian x, y plane, and let its polar coordinates be r and θ_n. Consider a simple process, or **mapping**, which sends that point into a point x_{n+1}, y_{n+1} at the same radius but at an incremented angle $\theta_n + \alpha$. Then, recalling the identity $\cos(A + B) = \cos A \cos B - \sin A \sin B$, we can express $x_{n+1} = r \cos(\theta_n + \alpha) = r \cos \theta_n \cos \alpha - r \sin \theta_n \sin \alpha = x_n \cos \alpha - y_n \sin \alpha$ and, recalling the identity $\sin(A + B) = \sin A \cos B + \sin B \cos A$, we can express $y_{n+1} = r \sin(\theta_n + \alpha) = r \sin \theta_n \cos \alpha + r \cos \theta_n \sin \alpha = y_n \cos \alpha + x_n \sin \alpha$. Thus, the process is described by the system of linear difference equations

$$\begin{aligned} x_{n+1} &= (\cos \alpha) x_n - (\sin \alpha) y_n, \\ y_{n+1} &= (\sin \alpha) x_n + (\cos \alpha) y_n. \end{aligned} \tag{37}$$

Surely, if one plots such a sequence of points it will fall on the circle of radius r centered at the origin. Suppose that we now modify the process by including two quadratic x_n^2 terms, so that we have the *non*linear system

$$\begin{aligned} x_{n+1} &= (\cos \alpha) x_n - (\sin \alpha) \left(y_n - x_n^2 \right), \\ y_{n+1} &= (\sin \alpha) x_n + (\cos \alpha) \left(y_n - x_n^2 \right). \end{aligned} \tag{38}$$

This system, studied initially by *M. Henòn*, turns out to be remarkably complex and interesting by virtue of the nonlinearity. For a discussion of the main results, we

highly recommend the little book *Mathematics and the Unexpected*, by Ivar Ekeland (Chicago: University of Chicago Press, 1988).

Closure. This section is primarily about the concept of stability in the numerical solution of differential equations. A scheme is stable if the roundoff error remains small compared to the exact solution. Normally, one establishes the stability or instability of a method with respect to the simple test equation $y' = Ay$. Assuming that roundoff enters in the initial condition and that the computer is perfect thereafter, one can derive a difference equation governing the roundoff error e_n, and solve it analytically to see if e_n remains small. Doing so, we show that the midpoint rule is only weakly stable: stable if $A > 0$ and unstable if $A < 0$. As a rule of thumb, we suggest that for a given differential equation $y' = f(x, y)$ we can expect the midpoint rule to be stable if $\partial f/\partial y > 0$ and unstable if $\partial f/\partial y < 0$ over the x, y region of interest.

To explain the source of the instability in the midpoint rule, we observe that the exact solution (17) of the midpoint rule difference equation corresponding to the test equation $y' = Ay$ contains two terms, one that corresponds to the exact solution of $y' = Ay$ and the other extraneous. The latter enters because the midpoint rule difference equation is of second order, whereas the differential equation is only of first order, and it is that extraneous term that leads to the instability. Single-step methods such as the Euler and Runge–Kutta methods, however, are strongly stable, provided that h is sufficiently small.

Observe that the only multi-step method that we examine is the midpoint rule; we neither show nor claim that all multi-step methods exhibit such instability. For instance, it is left for the exercises to show that the multi-step fourth-order Adams–Moulton method is strongly stable (for sufficiently small h). Thus, the idea is that the extraneous terms in the solution, that arise because the difference equation is of a higher order than the differential equation, can, but need not, cause trouble.

We close the section with a brief study of difference equations, independent of any connection with differential equations and stability since they are important in their own right in the modeling of discrete systems. We stress how analogous are the theories governing differential and difference equations that are linear, homogeneous, and with constant coefficients.

Computer software. Just as many differential equations can be solved analytically by computer-algebra systems, so can many difference equations. Using *Maple*, for instance, the relevant command is **rsolve**. For instance, to solve the difference equation $y_{n+2} - y_{n+1} - 6y_n = 0$ (from Example 4), enter

$$\text{rsolve}(y(n+2) - y(n+1) - 6 * y(n) = 0,\ y(n));$$

and return. The result is

$$\left(\frac{3}{5}y(0) - \frac{1}{5}y(1)\right)(-2)^n - \left(-\frac{2}{5}y(0) - \frac{1}{5}y(1)\right)(3)^n$$

the correct solution for any initial values $y(0)$ and $y(1)$. Of course, we could re-

express the latter as

$$C_1(-2)^n + C_2(3)^n$$

if we wish. If we had initial conditions $y_0 = 4$ and $y_1 = -7$, say, then we would have entered

rsolve($\{y(n+2) - y(n+1) - 6 * y(n) = 0,\ y(0) = 4,\ y(1) = -7\}$, $y(n)$);

and would have obtained

$$\frac{19}{5}(-2)^n + \frac{1}{5}(3)^n$$

as the desired particular solution.

EXERCISES 6.5

1. If the given initial-value problem were to be solved by the fourth-order Runge–Kutta method (and we are not asking you to do that), do you think accurate results could be obtained? Explain. The x domain is $0 \le x < \infty$.

(a) $y' = 2y - 8x + 4; \quad y(0) = 0$
(b) $y' = y - 2e^{-x}; \quad y(0) = 1$
(c) $y' = y + 5e^{-4x}; \quad y(0) = -1$
(d) $y' = 1 + 3(y - x); \quad y(0) = 0$

2. It is natural to wonder how well we would fare trying to solve (1) using computer software. Using the *Maple* dsolve command with the abserr option, see if you can obtain accurate results at the points $x = 1, 4, 8, 12$ listed in Table 1.

3. One can see if a computed solution exhibits instability, as did the solution obtained by the midpoint rule and plotted in Fig. 2, when we have the exact solution to compare it with. In practice, of course, we don't have the exact solution to compare with; if we did, then we would not be solving numerically in the first place. Thus, when a computed solution exhibits an oscillatory behavior how do we know that it is incorrect; perhaps the exact solution has exactly that oscillatory behavior? One way to find out is to rerun the solution with h halved. If the oscillatory behavior is part of the exact solution, then the new results will oscillate every two steps rather than every step. Using this idea, run the case shown in Fig. 2 twice, for $h = 0.05$ and $h = 0.025$, and comment on whether the results indicate a true instability or not.

4. We derived the solution (12) of the difference equation (7) in the text. Verify, by direct substitution, that (12) does satisfy (7) for any choice of the arbitrary constants C_1 and C_2.

5. In (13) we showed that $\left(Ah + \sqrt{1 + A^2h^2}\right)^n \sim e^{Ax_n}$ as $h \to 0$, yet it would appear that $\left(Ah + \sqrt{1 + A^2h^2}\right)^n \sim \left(\sqrt{1}\right)^n = 1$. Explain the apparent contradiction.

6. The purpose of this exercise is to explore the validity of the rule of thumb that we gave regarding the solution of the equation $y' = f(x, y)$ by the midpoint rule – namely, that the method should be stable if $\partial f/\partial y > 0$ and unstable if $\partial f/\partial y < 0$ over the region of interest. Specifically, in each case apply the rule of thumb and draw what conclusions you can about the stability of the midpoint rule solution of the given problem. Then, program and run the midpoint rule with $h = 0.05$, say, over the given x interval. Discuss the numerical results and whether the rule of thumb was correct. (Since the midpoint rule is not self-starting, use ten Euler steps from $x = 0$ to $x = 0.05$ to get the method started.)

(a) $y' = e^{x+y}; \quad y(0) = 1, 0 \le x \le 4$
(b) $y' = e^{x-y}; \quad y(0) = 1, 0 \le x \le 4$
(c) $y' = (4 - x)y; \quad y(0) = 1, 0 \le x \le 10$
(d) $y' = (x - 1)y; \quad y(0) = 1, 0 \le x \le 5$
(e) $y' = (x + y)/(1 + x); \quad y(0) = 2, 0 \le x \le 4$

7. We stated in the text that the results in Table 2 are consistent with a critical h value of 0.002 because the calculations change from unstable to stable as h decreases from 0.01 to 0.001. Program and carry out the Euler calculation of the solution to the initial-value problem (18) using $h = 0.0021$ and 0.0019, out to around $x = 1$, and see if these h values continue to bracket the change from unstable to stable. (You may try to bracket h_{cr} even more tightly if you wish.)

8. (*Stability of second-order Runge–Kutta methods*) In Sec-

tion 6.3 we derived the general second-order Runge–Kutta method, which includes these as special cases: the *improved Euler method*,

$$y_{n+1} = y_n + \frac{h}{2}\left\{f(x_n, y_n) + f\left[x_{n+1}, y_n + hf(x_n, y_n)\right]\right\}, \tag{8.1}$$

and the *modified Euler method*,

$$y_{n+1} = y_n + hf[x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n)]. \tag{8.2}$$

(a) For the test equation $y' = Ay$, show that the improved Euler method is strongly stable for sufficiently small h (i.e., as $h \to 0$). For the case where $A < 0$, show that that stability is achieved only if $h < h_{cr} = 2/|A|$.
(b) For the test equation $y' = Ay$, show that the modified Euler method is strongly stable for sufficiently small h (i.e., as $h \to 0$). For the case where $A < 0$, show that that stability is achieved only if $h < h_{cr} = 2/|A|$.

9. (*Strong stability of the multi-step Adams–Bashforth method*) Recall, from Section 6.3, the fourth-order Adams–Bashforth method

$$y_{n+1} = y_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right), \tag{9.1}$$

where $f_n = f(x_n, y_n)$. This exercise is to show that the "AB" method is strongly stable for sufficiently small h (i.e., as $h \to 0$) even though it is a multi-step method.

(a) Consider the test equation $y' = Ay$, where the constant A can be positive or negative; that is, let $f(x, y) = Ay$ be a solution of the fourth-order difference equation (9.1) in the form $y_n = \rho^n$, show that ρ must satisfy the fourth-degree characteristic equation

$$\rho^4 - (1 + 55\alpha)\rho^3 + 59\alpha\rho^2 - 37\alpha\rho + 9\alpha = 0, \tag{9.2}$$

where $\alpha = Ah/24$.
(b) Notice that as h tends to zero so does α, and (9.2) reduces to $\rho^4 - \rho^3 = 0$, with the roots $= 0, 0, 0, 1$. Thus, if we denote the roots of (9.2) as $\rho_1, \ldots, \rho_4$, then we see that the first three of these tend to zero and the last to unity as $h \to 0$, and the general solution for y_n behaves as

$$y_n = C_1\rho_1^n + C_2\rho_2^n + C_3\rho_3^n + C_4\rho_4^n \sim C_4 1^n \tag{9.3}$$

as $h \to 0$. Since p^n tends to zero, unity, or infinity, depending upon whether p is smaller than, equal to, or greater than unity, we need to examine the 1^n term in (9.3) more closely. Specifically, seeking ρ_4 in the power series form $\rho_4 = 1 + a\alpha + \cdots$, put that form into (9.2). Equating coefficients of α on both sides of that equation through first-order terms, show that $a = 24$. Thus, in place of (9.3) we have the more informative statement

$$y_n \sim C_4(1 + 24\alpha)^n = C_4(1 + Ah)^n \sim C_4 e^{Ax_n} \tag{9.4}$$

as $n \to \infty$. Show why the final step in (9.4) is true. Since the right-hand side of (9.4) is identical to the exact solution $y(x)$ of the given differential equation, whether A is positive or negative, we conclude that the AB method is strongly stable for sufficiently small h.

10. (*Strong stability of the multi-step Adams–Moulton method*) Recall, from Section 6.3, the fourth-order Adams–Moulton method,

$$y_{n+1} = y_n + \frac{h}{24}\left(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right). \tag{10.1}$$

Proceeding along the same lines as outlined in Exercise 9, show that the AM method is stongly stable for sufficiently small h.

11. Derive the general solution of the given difference equation. If initial conditions are specified, then find the corresponding particular solution. In each case $n = 0, 1, 2, \ldots$.

(a) $y_{n+1} - 4y_n = 0; \quad y_0 = 5$
(b) $y_{n+2} - y_n = 0; \quad y_0 = 1,\ y_1 = 3$
(c) $y_{n+2} + y_{n+1} - 6y_n = 0; \quad y_0 = 9,\ y_1 = -2$
(d) $y_{n+2} - 4y_{n+1} + 3y_n = 0; \quad y_0 = 3,\ y_1 = 1$
(e) $y_{n+2} + 3y_{n+1} + 2y_n = 0; \quad y_0 = 5,\ y_1 = 7$
(f) $y_{n+3} - y_{n+2} - 4y_{n+1} + 4y_n = 0; \quad y_0 = 3,\ y_1 = 5,\ y_2 = 9$
(g) $y_{n+4} - 5y_{n+2} + 4y_n = 0$
(h) $y_{n+4} - 6y_{n+2} + 8y_n = 0$

12. (a)–(h) Use computer software to obtain the general solution of the corresponding problem in Exercise 11, and the particular solution as well, if initial conditions are given.

13. (*Repeated roots*) Recall from the theory of linear homogeneous differential equations with constant coefficients that if, when we seek $y(x) = e^{\lambda x}$, λ is a root of multiplicity k of the characteristic equation, then it gives rise to the solutions $y(x) = \left(C_1 + C_2 x + \cdots + C_k x^{k-1}\right) e^{\lambda x}$, where $C_1, \ldots, C_k$ are arbitrary constants. An analogous result holds for difference equations. Specifically, verify that the characteristic equation of $y_{n+2} - 2by_{n+1} + b^2 y_n = 0$ has the root b with multiplicity 2, and that $y_n = (C_1 + C_2 n)b^n$.

14. Show that if $y_n^{(1)}$ and $y_n^{(2)}$ are solutions of the second-order linear homogeneous difference equation $a_0(n)y_{n+2} + a_1(n)y_{n+1} + a_2(n)y_n = 0$, and Y_n is any particular solution of the nonhomogeneous equation $a_0(n)y_{n+2} + a_1(n)y_{n+1} + a_2(n)y_n = f_n$, then $y_n = C_1y_n^{(1)} + C_2y_n^{(2)} + Y_n$ is a solution of the nonhomogeneous equation.

15. (*Nonhomogeneous difference equations*) For the given equation, first find the general solution of the homogeneous equation. Then adapt the method of undetermined coefficients from the theory of differential equations and find a particular solution. Finally, give the general solution of the given nonhomogeneous difference equation. (First, read Exercise 14.)

(a) $y_{n+1} - 3y_n = n$
(b) $y_{n+1} - 2y_n = 3\sin n$
(c) $y_{n+1} - y_n = 2 + \cos n$
(d) $y_{n+2} - 5y_{n+1} + 6y_n = 2n^2 - 6n - 1$
(e) $y_{n+2} + y_{n+1} - 2y_n = n^2$
(f) $y_{n+2} - 4y_n = 6n^2 - 1$
(g) $y_{n+2} - y_n = e^n$
(h) $y_{n+4} - 7y_{n+2} + 12y_n = n + 6$

16. (a)–(h) Use computer software to obtain the general solution of the corresponding problem in Exercise 15.

Chapter 6 Review

To solve a differential equation $y' = f(x, y)$ numerically, one begins by discretizing the problem and working with the discrete variables x_n, y_n in place of the continuous variables $x, y(x)$. The solution is accomplished using a numerical algorithm that gives y_{n+1} in terms of y_n in the case of a single-step method, or in terms of y_n and one or more of the preceding values $y_{n-1}, y_{n-2}, \ldots$ in the case of a multi-step method. If the algorithm gives y_{n+1} explicitly, then it is said to be of **open** type; if not it is of **closed** type. All methods considered in this chapter are of open type, except for the Adams–Moulton method (Section 6.3.5).

Decomposing the error as

$$\text{Total error } = y(x_n) - y_n^* = [y(x_n) - y_n] + [y_n - y_n^*],$$

where $y(x_n)$ is the exact solution of the differential equation, y_n is the exact solution of the numerical algorithm (i.e., carried out on a perfect machine, having no roundoff error), and y_n^* is the actual solution of the algorithm (carried out on a real machine), we call $y(x_n) - y_n$ the **accumulated truncation error** and $y_n - y_n^*$ the **accumulated roundoff error**. If the former is of order $O(h^p)$ at a fixed x point as $h \to 0$, then the method is said to be of **order** p. If $p > 0$, then the accumulated truncation error tends to zero and the method is described as **convergent**. The greater p is, the more accurate the method is for a given step size h. Besides requiring of a method that it be convergent, we also require that it be **stable**; that is, we require that the accumulated roundoff error (which is inevitable in a real machine) remain small compared to the exact solution of the differential equation.

We begin with the **Euler method** $y_{n+1} = y_n + f(x_n, y_n)h$, which is simple but not very accurate because it is only a first-order method. Desiring greater accuracy, we introduce the second-order **midpoint rule** and second- and fourth-order **Runge–Kutta methods**, the latter providing us with an accurate general-purpose differential equation solver.

One's interest in higher-order methods is not just a matter of accuracy because, in principle, one could rely exclusively on the simple and easily programmed Euler method, and make h small enough to achieve any desired accuracy. There are two problems with that idea. First, as h is decreased the number of steps increases and one can expect the numerical roundoff error to grow, so that it may not be possible to achieve the desired accuracy. Second, there is the question of economy. For instance, while the fourth-order Runge–Kutta method (for example) is about four times as slow as the Euler method (because it requires four function evaluations per step compared to one for the Euler method), the gain in accuracy that it affords is so great that we can use a step size much more than four times that needed by the Euler method for the same accuracy, thereby resulting in greater economy.

Naturally, higher-order methods are more complex and hence more tedious to program. Thus, we strongly urge (in Section 6.3.4) the empirical estimation of the order, if only as a check on the programming and implementation of the method.

In Section 6.4 we showed that the methods developed for the single equation $y' = f(x, y)$ can be used to solve systems of equations and higher-order equations as well. There we also study boundary-value problems, and find them to be significantly more difficult than initial-value problems. However, we show how to use the principle of superposition to convert a boundary-value problem to one or more problems of initial-value type, provided that the problem is linear.

Finally, in Section 6.5 we look at "what can go wrong," mostly insofar as numerical instability due to the growth of roundoff error, and an analytical approach is put forward for predicting whether a given method is stable. Actually, stability depends not only on the solution algorithm but also on the differential equation, and our analyses are for the simple test equation $y' = Ay$ rather than for the general case $y' = f(x, y)$. We find that whereas the differential equation $y' = Ay$ is of first order, the difference equation expressed by the algorithm is of higher order if the method is of multi-step type. Thus, it has among its solutions the exact solution (as $h \to 0$) and one or more extraneous solutions as well. It is those extraneous solutions that can cause instability. For instance, the midpoint rule is found to be stable if $A > 0$ and unstable if $A < 0$; we classify it as weakly stable because its stability depends upon the sign of A. However, the fourth-order Adams–Bashforth and Adams–Moulton methods are stable, even though they are multistep methods because the extraneous solutions do not grow. Single-step methods such as Euler and those of Runge–Kutta type do not give rise to extraneous solutions and are stable.

Finally, we stress that even if a method is stable as $h \to 0$, h needs to be reduced below some critical value for that stability to be manifested.