3.5 EXERCISES

1. An RL circuit with a 5-Ω resistor and a 0.05-H inductor carries a current of 1 A at $t = 0$, at which time a voltage source $E(t) = 5 \cos 120t$ V is added. Determine the subsequent inductor current and voltage.

2. An RC circuit with a 1-Ω resistor and a 0.000001-F capacitor is driven by a voltage $E(t) = \sin 100t$ V. If the initial capacitor voltage is zero, determine the subsequent resistor and capacitor voltages and the current.

3. The pathway for a binary electrical signal between gates in an integrated circuit can be modeled as an RC circuit, as in Figure 3.13(b); the voltage source models the transmitting gate, and the capacitor models the receiving gate. Typically, the resistance is 100 Ω, and the capacitance is very small, say, $10^{-12}$ F (1 picofarad, pF). If the capacitor is initially uncharged and the transmitting gate changes instantaneously from 0 to 5 V, how long will it take for the voltage at the receiving gate to reach (say) 3 V? (This is the time it takes to transmit a logical “1.”)

4. If the resistance in the RL circuit of Figure 3.13(a) is zero, show that the current $I(t)$ is directly proportional to the integral of the applied voltage $E(t)$. Similarly show that if the resistance in the RC circuit of Figure 3.13(b) is zero, the current is directly proportional to the derivative of the applied voltage. (In engineering applications, it is often necessary to generate a voltage, rather than a current, which is the integral or derivative of another voltage. Group Project E shows how this is accomplished using an operational amplifier.)

5. The power generated or dissipated by a circuit element equals the voltage across the element times the current through the element. Show that the power dissipated by a resistor equals $I^2 R$, the power associated with an inductor equals the derivative of $(1/2) L I^2$, and the power associated with a capacitor equals the derivative of $(1/2) C E^2$.

6. Derive a power balance equation for the RL and RC circuits. (See Problem 5.) Discuss the significance of the signs of the three power terms.

7. An industrial electromagnet can be modeled as an RL circuit, while it is being energized with a voltage source. If the inductance is 10 H and the wire windings contain 3 Ω of resistance, how long does it take a constant applied voltage to energize the electromagnet to within 90% of its final value (that is, the current equals 90% of its asymptotic value)?

8. A $10^{-8}$-F capacitor (10 nanofarads) is charged to 50 V and then disconnected. One can model the charge leakage of the capacitor with a RC circuit with no voltage source and the resistance of the air between the capacitor plates. On a cold dry day, the resistance of the air gap is $5 \times 10^{13}$ Ω; on a humid day, the resistance is $7 \times 10^6$ Ω. How long will it take the capacitor voltage to dissipate to half its original value on each day?

3.6 IMPROVED EULER’S METHOD

Although the analytical techniques presented in Chapter 2 were useful for the variety of mathematical models presented earlier in this chapter, the majority of the differential equations encountered in applications cannot be solved either implicitly or explicitly. This is especially true of higher-order equations and systems of equations, which we study in later chapters. In this section and the next, we discuss methods for obtaining a numerical approximation of the solution to an initial value problem for a first-order differential equation. Our goal is to develop algorithms that you can use with a calculator or computer. These algorithms also extend naturally to higher-order equations (see Section 5.3). We describe the rationale behind each method but leave the more detailed discussion to texts on numerical analysis.¹

¹An applet, maintained on the web at http://alamos.math.arizona.edu/~rchlik/JODE/index.html, automates most of the differential equation algorithms discussed in this book.

Consider the initial value problem

\[(1) \quad y' = f(x, y), \quad y(x_0) = y_0.\]

To guarantee that (1) has a unique solution, we assume that \(f\) and \(\partial f/\partial y\) are continuous in a rectangle \(R := \{(x, y): a < x < b, c < y < d\}\) containing \((x_0, y_0)\). It follows from Theorem 1 in Chapter 1 that the initial value problem (1) has a unique solution \(\phi(x)\) in some interval \(x_0 - \delta < x < x_0 + \delta\), where \(\delta\) is a positive number. Because \(\delta\) is not known a priori, there is no assurance that the solution will exist at a particular point \(x(\neq x_0)\), even if \(x\) is in the interval \((a, b)\). However, if \(\partial f/\partial y\) is continuous and bounded\(^1\) on the vertical strip

\[S := \{(x, y): a < x < b, -\infty < y < \infty\},\]

then it turns out that (1) has a unique solution on the whole interval \((a, b)\). In describing numerical methods, we assume that this last condition is satisfied and that \(f\) possesses as many continuous partial derivatives as needed.

In Section 1.4 we used the concept of direction fields to motivate a scheme for approximating the solution to the initial value problem (1). This scheme, called Euler's method, is one of the most basic, so it is worthwhile to discuss its advantages, disadvantages, and possible improvements. We begin with a derivation of Euler's method that is somewhat different from that presented in Section 1.4.

Let \(h > 0\) be fixed \((h\) is called the step size\) and consider the equally spaced points

\[(2) \quad x_n := x_0 + nh, \quad n = 0, 1, 2, \ldots.\]

Our goal is to obtain an approximation to \(\phi(x)\) of the initial value problem (1) at those points \(x_n\) that lie in the interval \((a, b)\). Namely, we will describe a method that generates values \(y_0, y_1, y_2, \ldots\) that approximate \(\phi(x)\) at the respective points \(x_0, x_1, x_2, \ldots\); that is,

\[y_n = \phi(x_n), \quad n = 0, 1, 2, \ldots.\]

Of course, the first "approximant" \(y_0\) is exact, since \(y_0 = \phi(x_0)\) is given. Thus, we must describe how to compute \(y_1, y_2, \ldots\).

For Euler's method we begin by integrating both sides of equation (1) from \(x_n\) to \(x_{n+1}\) to obtain

\[\phi(x_{n+1}) - \phi(x_n) = \int_{x_n}^{x_{n+1}} f(t, \phi(t)) dt = \int_{x_n}^{x_{n+1}} f(t, \phi(t)) dt,\]

where we have substituted \(\phi(x)\) for \(y\). Solving for \(\phi(x_{n+1})\), we have

\[(3) \quad \phi(x_{n+1}) = \phi(x_n) + \int_{x_n}^{x_{n+1}} f(t, \phi(t)) dt.\]

Without knowing \(\phi(t)\), we cannot integrate \(f(t, \phi(t))\). Hence, we must approximate the integral in (3). Assuming we have already found \(y_n = \phi(x_n)\), the simplest approach is to approximate the area under the function \(f(t, \phi(t))\) by the rectangle with base \([x_n, x_{n+1}]\) and height \(f(x_n, \phi(x_n))\) (see Figure 3.14). This gives

\[\phi(x_{n+1}) \approx \phi(x_n) + (x_{n+1} - x_n)f(x_n, \phi(x_n)).\]

Substituting \(h\) for \(x_{n+1} - x_n\) and the approximation \(y_n\) for \(\phi(x_n)\), we arrive at the numerical scheme

\[(4) \quad y_{n+1} = y_n + hf(x_n, y_n), \quad n = 0, 1, 2, \ldots,\]

which is Euler's method.

---

\(^1\)A function \(g(x, y)\) is bounded on \(S\) if there exists a number \(M\) such that \(|g(x, y)| \leq M\) for all \((x, y)\) in \(S\).
Starting with the given value \( y_0 \), we use (4) to compute \( y_1 = y_0 + hf(x_0, y_0) \) and then use \( y_1 \) to compute \( y_2 = y_1 + hf(x_1, y_1) \), and so on. Several examples of Euler’s method can be found in Section 1.4.

As discussed in Section 1.4, if we wish to use Euler’s method to approximate the solution to the initial value problem (1) at a particular value of \( x \), say, \( x = c \), then we must first determine a suitable step size \( h \) so that \( x_0 + Nh = c \) for some integer \( N \). For example, we can take \( N = 1 \) and \( h = c - x_0 \) in order to arrive at the approximation after just one step:

\[
\phi(c) = \phi(x_0 + h) = y_1.
\]

If, instead, we wish to take 10 steps in Euler’s method, we choose \( h = (c - x_0)/10 \) and ultimately obtain

\[
\phi(c) = \phi(x_0 + 10h) = \phi(x_{10}) \approx y_{10}.
\]

In general, depending on the size of \( h \), we will get different approximations to \( \phi(c) \). It is reasonable to expect that as \( h \) gets smaller (or, equivalently, as \( N \) gets larger), the Euler approximations approach the exact value \( \phi(c) \). On the other hand, as \( h \) gets smaller, the number (and cost) of computations increases and hence so do machine errors that arise from round-off. Thus, it is important to analyze how the error in the approximation scheme varies with \( h \).

If Euler’s method is used to approximate the solution \( \phi(x) = e^x \) to the problem

\[(5) \quad y' = y, \quad y(0) = 1, \]

at \( x = 1 \), then we obtain approximations to the constant \( e = \phi(1) \). It turns out that these approximations take a particularly simple form that enables us to compare the error in the approximation with the step size \( h \). Indeed, setting \( f(x, y) = y \) in (4) yields

\[
y_{n+1} = y_n + hy_n = (1 + h)y_n, \quad n = 0, 1, 2, \ldots.
\]

Since \( y_0 = 1 \), we get

\[
y_1 = (1 + h)y_0 = 1 + h, \\
y_2 = (1 + h)y_1 = (1 + h)(1 + h) = (1 + h)^2, \\
y_3 = (1 + h)y_2 = (1 + h)(1 + h)^2 = (1 + h)^3,
\]
and, in general,

\[ y_n = (1 + h)^n, \quad n = 0, 1, 2, \ldots. \]

For the problem in (5) we have \( x_0 = 0 \), so to obtain approximations at \( x = 1 \), we must set \( nh = 1 \). That is, \( h \) must be the reciprocal of an integer \( (h = 1/n) \). Replacing \( n \) by \( 1/h \) in (6), we see that Euler's method gives the (familiar) approximation \( (1 + h)^{1/h} \) to the constant \( e \). In Table 3.4, we list this approximation for \( h = 1, 10^{-1}, 10^{-2}, 10^{-3}, \text{and} 10^{-4} \), along with the corresponding errors

\[ e - (1 + h)^{1/h}. \]

From the second and third columns in Table 3.4, we see that the approximation gains roughly one decimal place in accuracy as \( h \) decreases by a factor of 10; that is, the error is roughly proportional to \( h \). This observation is further confirmed by the entries in the last column of Table 3.4. In fact, using methods of calculus (see Exercises 1.4, Problem 13), it can be shown that

\[ \lim_{h \to 0} \frac{\text{error}}{h} = \lim_{h \to 0} \frac{e - (1 + h)^{1/h}}{h} = \frac{e}{2} \approx 1.35914. \]

The general situation is similar: When Euler's method is used to approximate the solution to the initial value problem (1), the error in the approximation is at worst a constant times the step size \( h \). Moreover, in view of (7), this is the best one can say.

Numerical analysts have a convenient notation for describing the convergence behavior of a numerical scheme. For fixed \( x \) we denote by \( y(x; h) \) the approximation to the solution \( \phi(x) \) of (1) obtained via the scheme when using a step size of \( h \). We say that the numerical scheme converges at \( x \) if

\[ \lim_{h \to 0} y(x; h) = \phi(x). \]

In other words, as the step size \( h \) decreases to zero, the approximations for a convergent scheme approach the exact value \( \phi(x) \). The rate at which \( y(x; h) \) tends to \( \phi(x) \) is often expressed in terms of a suitable power of \( h \). If the error \( \phi(x) - y(x; h) \) tends to zero like a constant times \( h^p \), we write

\[ \phi(x) - y(x; h) = O(h^p) \]

and say that the method is of order \( p \). Of course, the higher the power \( p \), the faster is the rate of convergence as \( h \to 0 \).
As seen from our earlier discussion, the rate of convergence of Euler’s method is $O(h)$; that is, Euler’s method is of order $p = 1$. In fact, the limit in (7) shows that for equation (5), the error is roughly $1.36h$ for small $h$. This means that to have an error less than 0.01 requires $h < 0.01/1.36$, or $n = 1/h > 136$ computation steps. Thus Euler’s method converges too slowly to be of practical use.

How can we improve Euler’s method? To answer this, let’s return to the derivation expressed in formulas (3) and (4) and analyze the “errors” that were introduced to get the approximation. The crucial step in the process was to approximate the integral

$$
\int_{x_n}^{x_{n+1}} f(t, \phi(t)) \, dt
$$

by using a rectangle (recall Figure 3.14). This step gives rise to what is called the local truncation error in the method. From calculus we know that a better (more accurate) approach to approximating the integral is to use a trapezoid—that is, to apply the trapezoidal rule (see Figure 3.15). This gives

$$
\int_{x_n}^{x_{n+1}} f(t, \phi(t)) \, dt \approx \frac{h}{2} \left[ f(x_n, \phi(x_n)) + f(x_{n+1}, \phi(x_{n+1})) \right],
$$

which leads to the numerical scheme

$$
y_{n+1} = y_n + \frac{h}{2} \left[ f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right], \quad n = 0, 1, 2, \ldots
$$

We call equation (8) the trapezoid scheme. It is an example of an implicit method; that is, unlike Euler’s method, equation (8) gives only an implicit formula for $y_{n+1}$, since $y_{n+1}$ appears as an argument of $f$. Assuming we have already computed $y_n$, some root-finding technique such as Newton’s method (see Appendix B) might be needed to compute $y_{n+1}$. Despite the inconvenience of working with an implicit method, the trapezoid scheme has two advantages over Euler’s method. First, it is a method of order $p = 2$; that is, it converges at a rate that is proportional to $h^2$ and hence is faster than Euler’s method. Second, as described in Group Project H, the trapezoid scheme has the desirable feature of being stable.

Can we somehow modify the trapezoid scheme in order to obtain an explicit method? One idea is first to get an estimate, say, $y_{n+1}^*$, of the value $y_{n+1}$ using Euler’s method and then use formula (8) with $y_{n+1}$ replaced by $y_{n+1}^*$ on the right-hand side. This two-step process is an example of a predictor-corrector method. That is, we predict $y_{n+1}$ using Euler’s method and
then use that value in (8) to obtain a "more correct" approximation. Setting $y_{n+1} = y_n + hf(x_n, y_n)$ in the right-hand side of (8), we obtain

$$(9) \quad y_{n+1} = y_n + \frac{h}{2} \left[ f(x_n, y_n) + f(x_n + h, y_n + hf(x_n, y_n)) \right], \quad n = 0, 1, \ldots,$$

where $x_{n+1} = x_n + h$. This explicit scheme is known as the **improved Euler's method**.

**Example 1**

Compute the improved Euler's method approximation to the solution $\phi(x) = e^x$ of

$$y' = y, \quad y(0) = 1$$

at $x = 1$ using step sizes of $h = 1, 10^{-1}, 10^{-2}, 10^{-3}$, and $10^{-4}$.

**Solution**

The starting values are $x_0 = 0$ and $y_0 = 1$. Since $f(x, y) = y$, formula (9) becomes

$$y_{n+1} = y_n + \frac{h}{2} \left[ y_n + (y_n + hy_n) \right] = y_n + hy_n + \frac{h^2}{2} y_n;$$

that is,

$$y_{n+1} = \left( 1 + h + \frac{h^2}{2} \right) y_n.$$

Since $y_0 = 1$, we see inductively that

$$y_n = \left( 1 + h + \frac{h^2}{2} \right)^n, \quad n = 0, 1, 2, \ldots$$

To obtain approximations at $x = 1$, we must have $1 = x_0 + nh = nh$, and so $n = 1/h$. Hence, the improved Euler's approximations to $e = \phi(1)$ are just

$$y_n = \left( 1 + h + \frac{h^2}{2} \right)^{1/h}.$$

In Table 3.5 we have computed this approximation for the specified values of $h$, along with the corresponding errors

$$e - \left( 1 + h + \frac{h^2}{2} \right)^{1/h}.$$ 

Comparing the entries of this table with those of Table 3.4, we observe that the improved Euler's method converges much more rapidly than the original Euler's method. In fact, from the first few entries in the second and third columns of Table 3.5, it appears that the approximation gains two decimal places in accuracy each time $h$ is decreased by a factor of 10. In other words, the error is roughly proportional to $h^2$ (see the last column of the table and also Problem 4). The entries in the last two rows of the table must be regarded with caution. Indeed, when $h = 10^{-3}$ or $h = 10^{-4}$, the true error is so small that our calculator rounded it to zero, to five decimal places. The entries in color in the last column may be inaccurate due to the loss of significant figures in the calculator arithmetic. •
TABLE 3.5 Improved Euler’s Approximation to \( e = 2.71828 \ldots \)

<table>
<thead>
<tr>
<th>( h )</th>
<th>( (1 + h + h^2)^{1/h} )</th>
<th>Error</th>
<th>Error ( /h^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.50000</td>
<td>0.21828</td>
<td>0.21828</td>
</tr>
<tr>
<td>( 10^{-1} )</td>
<td>2.71408</td>
<td>0.00420</td>
<td>0.42010</td>
</tr>
<tr>
<td>( 10^{-2} )</td>
<td>2.71824</td>
<td>0.00004</td>
<td>0.44966</td>
</tr>
<tr>
<td>( 10^{-3} )</td>
<td>2.71828</td>
<td>0.00000</td>
<td>0.45271</td>
</tr>
</tbody>
</table>

As Example 1 suggests, the improved Euler’s method converges at the rate \( O(h^2) \), and indeed it can be proved that in general this method is of order \( p = 2 \).

A step-by-step outline for a subroutine that implements the improved Euler’s method over a given interval \([x_0, c]\) is described below. For programming purposes it is usually more convenient to input the number of steps \( N \) in the interval rather than the step size \( h \) itself. For an interval starting at \( x = x_0 \) and ending at \( x = c \), the relation between \( h \) and \( N \) is

\[
Nh = c - x_0 .
\]

(Note that the subroutine includes an option for printing \( x \) and \( y \).) Of course, the implementation of this algorithm with \( N \) steps on the interval \([x_0, c]\) only produces approximations to the actual solution at \( N + 1 \) equally spaced points. If we wish to use these points to help graph an approximate solution over the whole interval \([x_0, c]\), then we must somehow “fill in” the gaps between these points. A crude method is to simply join the points by straight-line segments producing a polygonal line approximation to \( \phi(x) \). More sophisticated techniques for prescribing the intermediate points are used in professional codes.

**IMPROVED EULER’S METHOD SUBROUTINE**

**Purpose**
To approximate the solution \( \phi(x) \) to the initial value problem

\[
y' = f(x, y), \quad y(x_0) = y_0 ,
\]

for \( x_0 \leq x \leq c \).

**INPUT**
- \( x_0, y_0, c, N \) (number of steps), PRNTR (=1 to print a table)

**Step 1**
Set step size \( h = (c - x_0)/N \), \( x = x_0, y = y_0 \)

**Step 2**
For \( i = 1 \) to \( N \), do Steps 3–5

**Step 3**
Set

\[
F = f(x, y) \\
G = f(x + h, y + hf)
\]

**Step 4**
Set

\[
x = x + h \\
y = y + h(F + G)/2
\]

**Step 5**
If PRNTR = 1, print \( x, y \)
Now we want to devise a program that will compute \( \phi(c) \) to a desired accuracy. As we have seen, the accuracy of the approximation depends on the step size \( h \). Our strategy, then, will be to estimate \( \phi(c) \) for a given step size and then halve the step size and recompute the estimate, halve again, and so on. When two consecutive estimates of \( \phi(c) \) differ by less than some prescribed tolerance \( \varepsilon \), we take the final estimate as our approximation to \( \phi(c) \). Admittedly, this does not guarantee that \( \phi(c) \) is known to within \( \varepsilon \), but it provides a reasonable stopping procedure in practice.\(^1\) The following procedure also contains a safeguard to stop if the desired tolerance is not reached after \( M \) halvings of \( h \).

### IMPROVED EULER’S METHOD WITH TOLERANCE

**Purpose** To approximate the solution to the initial value problem

\[
y' = f(x, y) , \quad y(x_0) = y_0 ,
\]

at \( x = c \), with tolerance \( \varepsilon \)

**INPUT**

- \( x_0, y_0, c, \varepsilon \)
- \( M \) (maximum number of halvings of step size)

**Step 1** Set \( z = y_0 \), PRNTR = 0

**Step 2** For \( m = 0 \) to \( M \), do Steps 3–7\(^\dagger\)

**Step 3** Set \( N = 2^m \)

**Step 4** Call IMPROVED EULER’S METHOD SUBROUTINE

**Step 5** Print \( h, y \)

**Step 6** If \( |y - z| < \varepsilon \), go to Step 10

**Step 7** Set \( z = y \)

**Step 8** Print “\( \phi(c) \) is approximately”; \( y \); “but may not be within the tolerance”; \( \varepsilon \)

**Step 9** Go to Step 11

**Step 10** Print “\( \phi(c) \) is approximately”; \( y \); “with tolerance”; \( \varepsilon \)

**Step 11** STOP

**OUTPUT** Approximations of the solution to the initial value problem at \( x = c \) using \( 2^m \) steps

If one desires a stopping procedure that simulates the relative error

\[
\left| \frac{\text{approximation} - \text{true value}}{\text{true value}} \right|
\]

then replace Step 6 by

**Step 6'** If \( \left| \frac{z - y}{y} \right| < \varepsilon \), go to Step 10.

### Example 2

Use the improved Euler's method with tolerance to approximate the solution to the initial value problem

\[(12) \quad y' = x + 2y , \quad y(0) = 0.25 ,\]

at \( x = 2 \). For a tolerance of \( \varepsilon = 0.001 \), use a stopping procedure based on the absolute error.

\(^1\) Professional codes monitor accuracy much more carefully and vary step size in an adaptive fashion for this purpose.

\(^\dagger\) To save time, one can start with \( m = K < M \) rather than with \( m = 0 \).
Solution The starting values are $x_0 = 0, y_0 = 0.25$. Because we are computing the approximations for $c = 2$, the initial value for $h$ is

$$h = (2 - 0)2^{-0} = 2 .$$

For equation (12), we have $f(x, y) = x + 2y$, so the numbers $F$ and $G$ in the subroutine are

$$F = x + 2y ,$$

$$G = (x + h) + 2(y + hF) = x + 2y + h(1 + 2x + 4y) ,$$

and we find

$$x = x + h ,$$

$$y = y + \frac{h}{2}(F + G) = y + \frac{h}{2}(2x + 4y) + \frac{h^2}{2}(1 + 2x + 4y) .$$

Thus, with $x_0 = 0, y_0 = 0.25$, and $h = 2$, we get for the first approximation

$$y = 0.25 + (0 + 0) + 2(0 + 1) = 5.25 .$$

To describe the further outputs of the algorithm, we use the notation $y(2; h)$ for the approximation obtained with step size $h$. Thus, $y(2; 2) = 5.25$, and we find from the algorithm

$$y(2; 1) = 11.25000 \quad y(2; 2^{-5}) = 25.98132$$

$$y(2; 2^{-1}) = 18.28125 \quad y(2; 2^{-6}) = 26.03172$$

$$y(2; 2^{-2}) = 23.06067 \quad y(2; 2^{-7}) = 26.04468$$

$$y(2; 2^{-3}) = 25.12012 \quad y(2; 2^{-8}) = 26.04797$$

$$y(2; 2^{-4}) = 25.79127 \quad y(2; 2^{-9}) = 26.04880 .$$

Since $|y(2; 2^{-9}) - y(2; 2^{-8})| = 0.00083$, which is less than $e = 0.001$, we stop.

The exact solution of (12) is $\phi(x) = \frac{1}{2}(e^{2x} - x - \frac{1}{2})$, so we have determined that

$$\phi(2) = \frac{1}{2}(e^{4} - \frac{5}{2}) = 26.04880 .$$

In the next section, we discuss methods with higher rates of convergence than either Euler's or the improved Euler's methods.

### 3.6 Exercises

1. Show that when Euler's method is used to approximate the solution of the initial value problem

$$y' = 5y , \quad y(0) = 1 ,$$

at $x = 1$, then the approximation with step size $h$ is $(1 + 5h)^{1/h}$.

2. Show that when Euler's method is used to approximate the solution of the initial value problem

$$y' = -\frac{1}{2}y , \quad y(0) = 3 ,$$

at $x = 2$, then the approximation with step size $h$ is

$$3 \left(1 - \frac{h}{2}\right)^{3h} .$$

---

3.7 \textbf{HIGHER-ORDER NUMERICAL METHODS: TAYLOR AND RUNGE-KUTTA}

In Sections 1.4 and 3.6, we discussed a simple numerical procedure, Euler’s method, for obtaining a numerical approximation of the solution \( \phi(x) \) to the initial value problem

\[
(1) \quad y' = f(x, y), \quad y(x_0) = y_0.
\]

Euler’s method is easy to implement because it involves only linear approximations to the solution \( \phi(x) \). But it suffers from slow convergence, being a method of order 1; that is, the error is \( O(h) \). Even the improved Euler’s method discussed in Section 3.6 has order of only 2. In this section we present numerical methods that have faster rates of convergence. These include \textbf{Taylor methods}, which are natural extensions of the Euler procedure, and \textbf{Runge-Kutta methods}, which are the more popular schemes for solving initial value problems because they have fast rates of convergence and are easy to program.

As in the previous section, we assume that \( f \) and \( \frac{\partial f}{\partial y} \) are continuous and bounded on the vertical strip \( \{ (x, y) : a < x < b, -\infty < y < \infty \} \) and that \( f \) possesses as many continuous partial derivatives as needed.

To derive the Taylor methods, let \( \phi_n(x) \) be the exact solution of the related initial value problem

\[
(2) \quad \phi_n = f(x, \phi_n), \quad \phi_n(x_0) = y_n.
\]

The Taylor series for \( \phi_n(x) \) about the point \( x_n \) is

\[
\phi_n(x) = \phi_n(x_n) + h\phi_n'(x_n) + \frac{h^2}{2!}\phi_n''(x_n) + \cdots,
\]

where \( h = x - x_n \). Since \( \phi_n \) satisfies (2), we can write this series in the form

\[
(3) \quad \phi_n(x) = y_n + hf(x_n, y_n) + \frac{h^2}{2!}\phi''(x_n) + \cdots.
\]

Observe that the recursive formula for \( y_{n+1} \) in Euler’s method is obtained by truncating the Taylor series after the linear term. For a better approximation, we will use more terms in the Taylor series. This requires that we express the higher-order derivatives of the solution in terms of the function \( f(x, y) \).

If \( y \) satisfies \( y' = f(x, y) \), we can compute \( y'' \) by using the chain rule:

\[
(4) \quad y'' = \frac{\partial f}{\partial x}(x, y) y' + \frac{\partial f}{\partial y}(x, y) y'' \\
= \frac{\partial f}{\partial x}(x, y) + \frac{\partial f}{\partial y}(x, y)f(x, y) \\
=: f_2(x, y).
\]

In a similar fashion, define \( f_3, f_4, \ldots \), that correspond to the expressions \( y'''(x), y^{(4)}(x) \), etc. If we truncate the expansion in (3) after the \( h^p \) term, then, with the above notation, the recursive formulas for the \textbf{Taylor method of order} \( p \) are

\[
(5) \quad x_{n+1} = x_n + h,
\]

\[
(6) \quad y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2!}f_2(x_n, y_n) + \cdots + \frac{h^p}{p!}f_p(x_n, y_n).
\]
As before, \( y_n = \phi(x_n) \), where \( \phi(x) \) is the solution to the initial value problem (1). It can be shown\(^1\) that the Taylor method of order \( p \) has the rate of convergence \( O(h^p) \).

**Example 1**

Determine the recursive formulas for the Taylor method of order 2 for the initial value problem

\[
y' = \sin(xy), \quad y(0) = \pi.
\]

**Solution**

We must compute \( f_2(x, y) \) as defined in (4). Since \( f(x, y) = \sin(xy) \),

\[
\frac{\partial f}{\partial x}(x, y) = y \cos(xy), \quad \frac{\partial f}{\partial y}(x, y) = x \cos(xy).
\]

Substituting into (4), we have

\[
f_2(x, y) = \frac{\partial f}{\partial x}(x, y) + \frac{\partial f}{\partial y}(x, y)f(x, y)
= y \cos(xy) + x \cos(xy) \sin(xy)
= y \cos(xy) + \frac{x}{2} \sin(2xy),
\]

and the recursive formulas (5) and (6) become

\[
x_{n+1} = x_n + h,
\]

\[
y_{n+1} = y_n + h \sin(x_n y_n) + \frac{h^2}{2} \left[ y_n \cos(x_n y_n) + \frac{x_n}{2} \sin(2x_n y_n) \right],
\]

where \( x_0 = 0, y_0 = \pi \) are the starting values. \( \diamond \)

The convergence rate, \( O(h^p) \), of the \( p \)th-order Taylor method raises an interesting question: If we could somehow let \( p \) go to infinity, would we obtain exact solutions for the interval \([x_0, x_0 + h]\)? This possibility is explored in depth in Chapter 8. Of course, a practical difficulty in employing high-order Taylor methods is the tedious computation of the partial derivatives needed to determine \( f_p \) (typically these computations grow exponentially with \( p \)). One way to circumvent this difficulty is to use one of the Runge–Kutta methods.\(^{11}\)

Observe that the general Taylor method has the form

\[
y_{n+1} = y_n + hF(x_n, y_n; h),
\]

where the choice of \( F \) depends on \( p \). In particular [compare (6)], for

\[
p = 1, \quad F = T_1(x, y; h) = f(x, y),
\]

\[
p = 2, \quad F = T_2(x, y; h) = f(x, y) + \frac{h}{2} \left[ \frac{\partial f}{\partial x}(x, y) + \frac{\partial f}{\partial y}(x, y)f(x, y) \right].
\]

The idea behind the Runge–Kutta method of order 2 is to choose \( F \) in (8) of the form

\[
F = K_2(x, y; h) = f(x + \alpha h, y + \beta hf(x, y)),
\]

\(^1\)See *Introduction to Numerical Analysis* by J. Stoer and R. Bulirsch (Springer-Verlag, New York, 2002).

\(^{11}\) *Historical Footnote*: These methods were developed by C. Runge in 1895 and W. Kutta in 1901.
where the constants $\alpha, \beta$ are to be selected so that (8) has the rate of convergence $O(h^2)$. The advantage here is that $K_2$ is computed by two evaluations of the original function $f(x, y)$ and does not involve the derivatives of $f(x, y)$.

To ensure $O(h^2)$ convergence, we compare this new scheme with the Taylor method of order 2 and require

$$T_2(x, y; h) - K_2(x, y; h) = O(h^2), \quad \text{as} \quad h \to 0.$$ 

That is, we choose $\alpha, \beta$ so that the Taylor expansions for $T_2$ and $K_2$ agree through terms of order $h$. For $(x, y)$ fixed, when we expand $K_2 = K_2(h)$ as given in (10) about $h = 0$, we find

$$K_2(h) = K_2(0) + \frac{dK_2}{dh}(0)h + O(h^2)$$

$$= f(x, y) + \left[ \alpha \frac{\partial f}{\partial x}(x, y) + \beta \frac{\partial f}{\partial y}(x, y)f(x, y) \right] h + O(h^2),$$

where the expression in brackets for $dK_2/dh$, evaluated at $h = 0$, follows from the chain rule. Comparing (11) with (9), we see that for $T_2$ and $K_2$ to agree through terms of order $h$, we must have $\alpha = \beta = 1/2$. Thus,

$$K_2(x, y; h) = f \left( x + \frac{h}{2}, y + \frac{h}{2} f(x, y) \right).$$

The Runge–Kutta method we have derived is called the **midpoint method** and it has the recursive formulas

$$x_{n+1} = x_n + h,$$  

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

By construction, the midpoint method has the same rate of convergence as the Taylor method of order 2; that is, $O(h^2)$. This is the same rate as the improved Euler’s method.

In a similar fashion, one can work with the Taylor method of order 4 and, after some elaborate calculations, obtain the classical **fourth-order Runge–Kutta method**. The recursive formulas for this method are

$$x_{n+1} = x_n + h,$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where

$$k_1 = hf(x_n, y_n),$$

$$k_2 = hf \left( x_n + \frac{h}{2}, y_n + \frac{k_1}{2} \right),$$

$$k_3 = hf \left( x_n + \frac{h}{2}, y_n + \frac{k_2}{2} \right),$$

$$k_4 = hf(x_n + h, y_n + k_3).$$
The classical fourth-order Runge–Kutta method is one of the more popular methods because its rate of convergence is $O(h^4)$ and it is easy to program. Typically, it produces very accurate approximations even when the number of iterations is reasonably small. However, as the number of iterations becomes large, other types of errors may creep in.

Program outlines for the fourth-order Runge–Kutta method are given below. Just as with the algorithms for the improved Euler’s method, the first program (the Runge–Kutta subroutine) is useful for approximating the solution over an interval $[x_0, c]$ and takes the number of steps in the interval as input. As in Section 3.6, the number of steps $N$ is related to the step size $h$ and the interval $[x_0, c]$ by

$$Nh = c - x_0.$$ 

The subroutine has the option to print out a table of values of $x$ and $y$. The second algorithm (Runge–Kutta with tolerance) on page 136 is used to approximate, for a given tolerance, the solution at an inputted value $x = c$. This algorithm\(^1\) automatically halves the step sizes successively until the two approximations $y(c; h)$ and $y(c; h/2)$ differ by less than the prescribed tolerance $\epsilon$. For a stopping procedure based on the relative error, Step 6 of the algorithm should be replaced by

**Step 6’** If \(\left| \frac{y_n - y_{n-1}}{y_n} \right| < \epsilon\), go to Step 10.

---

**CLASSICAL FOURTH-ORDER RUNGE–KUTTA SUBROUTINE**

**Purpose** To approximate the solution to the initial value problem

\[
y' = f(x, y), \quad y(x_0) = y_0
\]

for $x_0 \leq x \leq c$

**INPUT** $x_0, y_0, c, N$ (number of steps), PRNTR (= 1 to print a table)

**Step 1** Set step size $h = (c - x_0)/N, x = x_0, y = y_0$

**Step 2** For $i = 1$ to $N$, do Steps 3–5

**Step 3** Set

\[
k_1 = hf(x, y) \\
k_2 = hf\left(x + \frac{h}{2}, y + \frac{k_1}{2}\right) \\
k_3 = hf\left(x + \frac{h}{2}, y + \frac{k_2}{2}\right) \\
k_4 = hf(x + h, y + k_3)
\]

**Step 4** Set

\[
x = x + h \\
y = y + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\]

**Step 5** If PRNTR = 1, print $x$, $y$

---

\(^1\)Note that the form of the algorithm on page 136 is the same as that for the improved Euler’s method on page 128 except for Step 4, where the Runge–Kutta subroutine is called. More sophisticated stopping procedures are used in production-grade codes.
CLASSICAL FOURTH-ORDER RUNGE-KUTTA ALGORITHM WITH TOLERANCE

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To approximate the solution to the initial value problem $y' = f(x, y), \quad y(x_0) = y_0$ at $x = c$, with tolerance $\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>$x_0, y_0, c, \varepsilon, M$ (maximum number of iterations) \quad Step 1 \quad Set $z = y_0, \text{PRNTR} = 0$ \quad Step 2 \quad For $m = 0$ to $M$, do Steps 3–7 (or, to save time, start with $m &gt; 0$) \quad Step 3 \quad Set $N = 2^n$ \quad Step 4 \quad Call FOURTH-ORDER RUNGE-KUTTA SUBROUTINE \quad Step 5 \quad Print $h, y$ \quad Step 6 \quad If $</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Approximations of the solution to the initial value problem at $x = c$, using $2^n$ steps.</td>
</tr>
</tbody>
</table>

Example 2  Use the classical fourth-order Runge–Kutta algorithm to approximate the solution $\phi(x)$ of the initial value problem

$$y' = y, \quad y(0) = 1,$$

at $x = 1$ with a tolerance of 0.001.

Solution  The inputs are $x_0 = 0, y_0 = 1, c = 1, \varepsilon = 0.001$, and $M = 25$ (say). Since $f(x, y) = y$, the formulas in Step 3 of the subroutine become

$$k = hy, \quad k_2 = h\left(y + \frac{k_1}{2}\right), \quad k_3 = h\left(y + \frac{k_2}{2}\right), \quad k_4 = h(y + k_3).$$

The initial value for $N$ in this algorithm is $N = 1$, so

$$h = (1 - 0)/1 = 1.$$  

Thus, in Step 3 of the subroutine, we compute

$$k_1 = (1)(1) = 1, \quad k_2 = (1)(1 + 0.5) = 1.5, \quad k_3 = (1)(1 + 0.75) = 1.75, \quad k_4 = (1)(1 + 1.75) = 2.75,$$

and, in Step 4 of the subroutine, we get for the first approximation

$$y = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$= 1 + \frac{1}{6}[1 + 2(1.5) + 2(1.75) + 2.75]$$

$$= 2.70833,$$
where we have rounded to five decimal places. Because

$$|z - y| = |y_0 - y| = |1 - 2.70833| = 1.70833 > \varepsilon,$$

we start over and reset $N = 2$, $h = 0.5$.

Doing Steps 3 and 4 for $i = 1$ and 2, we ultimately obtain (for $i = 2$) the approximation

$$y = 2.71735.$$

Since $|z - y| = |2.70833 - 2.71735| = 0.00902 > \varepsilon$, we again start over and reset $N = 4$, $h = 0.25$. This leads to the approximation

$$y = 2.71821,$$

so that

$$|z - y| = |2.71735 - 2.71821| = 0.00086,$$

which is less than $\varepsilon = 0.001$. Hence $\phi(1) = e = 2.71821$. ♦

In Example 2 we were able to obtain a better approximation for $\phi(1) = e$ with $h = 0.25$ than we obtained in Section 3.6 using Euler's method with $h = 0.001$ (see Table 3.4, page 124) and roughly the same accuracy as we obtained in Section 3.6 using the improved Euler's method with $h = 0.01$ (see Table 3.5, page 127).

**Example 3** Use the fourth-order Runge–Kutta subroutine to approximate the solution $\phi(x)$ of the initial value problem

$$(15) \quad y' = y^2, \quad y(0) = 1,$$

on the interval $0 \leq x \leq 2$ using $N = 8$ steps (i.e., $h = 0.25$).

**Solution** Here the starting values are $x_0 = 0$ and $y_0 = 1$. Since $f(x, y) = y^2$, the formulas in Step 3 of the subroutine are

$$k_1 = hy^2, \quad k_2 = h\left(y + \frac{k_1}{2}\right)^2, \quad k_3 = h\left(y + \frac{k_2}{2}\right)^2, \quad k_4 = h(y + k_3)^2.$$

From the output, we find

$$
\begin{align*}
x &= 0.25 & y &= 1.33322, \\
x &= 0.50 & y &= 1.99884, \\
x &= 0.75 & y &= 3.97238, \\
x &= 1.00 & y &= 32.82820, \\
x &= 1.25 & y &= 4.09664 \times 10^{11}, \\
x &= 1.50 & y &= \text{overflow}.
\end{align*}
$$

What happened? Fortunately, the equation in (15) is separable, and, solving for $\phi(x)$, we obtain $\phi(x) = (1 - x)^{-1}$. It is now obvious where the problem lies: The true solution $\phi(x)$ is not defined
at \( x = 1 \). If we had been more cautious, we would have realized that \( \partial f/\partial y = 2y \) is not bounded for all \( y \). Hence, the existence of a unique solution is not guaranteed for all \( x \) between 0 and 2, and in this case, the method does not give meaningful approximations for \( x \) near (or greater than) 1.

**Example 4** Use the fourth-order Runge–Kutta algorithm to approximate the solution \( \phi(x) \) of the initial value problem

\[
y' = x - y^2, \quad y(0) = 1,
\]

at \( x = 2 \) with a tolerance of 0.0001.

**Solution**

This time we check to see whether \( \partial f/\partial y \) is bounded. Here \( \partial f/\partial y = -2y \), which is certainly unbounded in any vertical strip. However, let’s consider the qualitative behavior of the solution \( \phi(x) \). The solution curve starts at \((0,1)\), where \( \phi'(0) = 0 - 1 < 0 \), so \( \phi(x) \) begins decreasing and continues to decrease until it crosses the curve \( y = \sqrt{x} \). After crossing this curve, \( \phi(x) \) begins to increase, since \( \phi'(x) = x - \phi^2(x) > 0 \). As \( \phi(x) \) increases, it remains below the curve \( y = \sqrt{x} \). This is because if the solution were to get “close” to the curve \( y = \sqrt{x} \), then the derivative of \( \phi(x) \) would approach zero, so that overtaking the function \( \sqrt{x} \) is impossible.

Therefore, although the existence-uniqueness theorem does not guarantee a solution, we are inclined to try the algorithm anyway. The above argument shows that \( \phi(x) \) probably exists for \( x > 0 \), so we feel reasonably sure the fourth-order Runge–Kutta method will give a good approximation of the true solution \( \phi(x) \). Proceeding with the algorithm, we use the starting values \( x_0 = 0 \) and \( y_0 = 1 \). Since \( f(x, y) = x - y^2 \), the formulas in Step 3 of the subroutine become

\[
k_1 = h(x - y^2), \quad k_2 = h \left[ \left( x + \frac{h}{2} \right) - \left( y + \frac{k_1}{2} \right)^2 \right],
\]

\[
k_3 = h \left[ \left( x + \frac{h}{2} \right) - \left( y + \frac{k_2}{2} \right)^2 \right], \quad k_4 = h \left[ (x + h) - (y + k_3)^2 \right].
\]

In Table 3.6, we give the approximations \( y(2; 2^{-m+1}) \) for \( \phi(2) \) for \( m = 0, 1, 2, 3, \) and 4. The algorithm stops at \( m = 4 \), since

\[
|y(2; 0.125) - y(2; 0.25)| = 0.00000.
\]

Hence, \( \phi(2) \approx 1.25132 \) with a tolerance of 0.0001.

<table>
<thead>
<tr>
<th>Table 3.6</th>
<th>Classical Fourth-Order Runge–Kutta Approximation for ( \phi(2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{m} )</td>
<td>( h )</td>
</tr>
<tr>
<td>0</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
</tr>
</tbody>
</table>