I. First-Order Ordinary Differential Equations
7. Numerical Methods

C. David Levermore
Department of Mathematics
University of Maryland

August 2, 2022

Contents

Return to Main Webpage

7. Numerical Methods

7.1. **Numerical Approximation.** Analytic methods are either difficult or impossible to apply to many first-order differential equations. In such cases direction fields are the only graphical method which we have covered that can be applied. However, it can be hard to understand how any particular solution behaves from the direction field of its governing equation. If we are interested in understanding how a particular solution behaves then a numerical method can be used to construct an accurate approximation to the solution. This approximation then can be graphed much like an explicit solution.

Suppose we are interested in the solution $Y(t)$ of the initial-value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t,y)\,, \qquad y(t_I) = y_I\,, \tag{7.1}$$

over the time interval $[t_I, t_F]$ — i.e. for $t_I \le t \le t_F$. Here t_I is called the *initial time* while t_F is called the *final time*. A numerical method selects times $\{t_n\}_{n=0}^N$ such that

$$t_I = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = t_F\,,$$

and computes values $\{y_n\}_{n=0}^N$ such that

$$y_0 = Y(t_0) = y_I\,,$$
$$y_n \text{ approximates } Y(t_n) \text{ for } n = 1, 2, \cdots, N\,.$$

For good numerical methods, these approximations will improve as N increases. So for sufficiently large N we can plot the points $\{(t_n, y_n)\}_{n=0}^N$ in the (t,y)-plane and "connect the dots" to get an accurate picture of how $Y(t)$ behaves over the time interval $[t_I, t_F]$.

Here we will introduce a few basic numerical methods in simple settings. The numerical methods used in software packages such as MATLAB are generally far more sophisticated than those we will study here. They are however built upon the same fundamental ideas as the simpler methods we will study. Throughout this chapter we will make the following two basic simplifications.

- We will employ *uniform time steps*. This means that given N we set

$$h = \frac{t_F - t_I}{N}\,, \qquad \text{and} \qquad t_n = t_I + nh \quad \text{for } n = 0, 1, \cdots, N\,, \tag{7.2}$$

  where h is called the *step size*.
- We will employ *one-step methods*. This means that given $f(t,y)$ and h the value of y_{n+1} for $n = 0, 1, \cdots, N-1$ will depend only on y_n.

Sophisticated software packages use methods in which the step size is chosen adaptively. In other words, the choice of t_{n+1} will depend on the behavior of recent approximations — for example, on (t_n, y_n) and (t_{n-1}, y_{n-1}). Employing uniform time steps greatly simplifies the algorithms, and thereby simplifies the programming we have to do. If we do not like the way a run looks, we will simply try again with a larger N.

Similarly, sophisticated software packages sometimes use so-called *multi-step methods* for which the value of y_{n+1} for $n = m, m+1, \cdots, N-1$ will depend on y_n, y_{n-1}, $\cdots$, and y_{n-m} for some positive integer m. Employing one-step methods again simplifies the algorithms, and thereby simplifies the programming we have to do.

7.2. **Explicit and Implicit Euler Methods.** The simplest (and least accurate) numerical methods are the Euler methods. These can be derived many ways. Here we give a simple approach based on the definition of the derivative through difference quotients.

7.2.1. *Explicit Euler Method.* If we start with the fact that

$$\lim_{h\to 0} \frac{Y(t+h)-Y(t)}{h} = Y'(t) = f(t,Y(t))\,,$$

then for small positive h we have

$$\frac{Y(t+h)-Y(t)}{h} \approx f(t,Y(t))\,.$$

Upon solving this for $Y(t+h)$ we find that

$$Y(t+h) \approx Y(t) + hf(t,Y(t))\,.$$

If we let $t = t_n$ above (so that $t+h = t_{n+1}$) this is equivalent to

$$Y(t_{n+1}) \approx Y(t_n) + hf(t_n,Y(t_n))\,.$$

Because y_n and y_{n+1} approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$y_{n+1} = y_n + hf(t_n,y_n) \quad \text{for } n = 0,1,\cdots,N-1\,. \tag{7.3}$$

This so-called Euler method was introduced by Leonhard Euler in 1768.

In practice, the explicit Euler method is implemented by initializing $y_0 = y_I$ and then for $n = 0,\cdots,N-1$ cycling through the instructions

$$f_n = f(t_n,y_n)\,, \qquad y_{n+1} = y_n + hf_n\,,$$

where $t_n = t_I + nh$. You should know the explicit Euler method and be able to carry out one or two steps of it by hand.

Example. Let $Y(t)$ be the solution of the initial-value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = t^2 + y^2\,, \qquad y(0) = 1\,.$$

Use the explicit Euler method with $h = .1$ to approximate $Y(.2)$.

Solution. We initialize $t_0 = 0$ and $y_0 = 1$. The explicit Euler method then gives

$$\begin{aligned}
f_0 &= f(t_0,y_0) = 0^2 + 1^2 = 1\\
y_1 &= y_0 + hf_0 = 1 + .1\cdot 1 = 1.1\\
f_1 &= f(t_1,y_1) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22\\
y_2 &= y_1 + hf_1 = 1.1 + .1\cdot 1.22 = 1.1 + .122 = 1.222
\end{aligned}$$

Therefore $Y(.2) \approx y_2 = 1.222$. □

Remark. Of course, when many time steps are to be taken then the explicit Euler method should be implemented on a computer. However, when using a computer you should understand what it is doing, or what it is supposed to be doing. Without such understanding you will not be able to spot when the compter is returning nonsense, or how to fix the computer program when it is. Indeed, hand calculations like in the above example still play an important role in debugging computer code.

The explicit Euler method is implemented by the following MATLAB function M-file.

```
function [t,y] = EulerExplicit(f, tI, yI, tF, N)

t = zeros(N + 1, 1); y = zeros(N + 1, 1);
t(1) = tI; y(1) = yI; h = (tF - tI)/N;
for j = 1:N
t(j + 1) = t(j) + h;
y(j + 1) = y(j) + h*f(t(j), y(j));
end
```

This M-file assumes that an anonymous function f is defined that gives the right-hand side of the differential equation. For example, if the right-hand side of the differential equation is $t^2 + y^2$ then we would type

```
>>  f = @(t, y) t^2 + y^2
```

The values of tI, yI, tF, and N are the initial time t_I, initial value y_I, final time t_F, and the number of time steps N. Given that f is defined as above, the foregoing example can be carried out using this M-file by typing

```
>>  [t, y] = EulerExplicit(f, 0.0, 1.0, 0.2, 2)
```

The vector t would return the values (t(1), t(2), t(3)) = (t_0, t_1, t_2) = (0.0, 0.1, 0.2) and the vector y would return the values (y(1), y(2), y(3)) = (y_0, y_1, y_2) = (1.0, 1.1, 1.222).

Remark. There are some things you should notice. First, t(j) is t_{j-1} and y(j) is y_{j-1}, the approximation of $y(t_{j-1})$. In particular, y(j) is *not* the same as $Y(j)$, which denotes the solution $Y(t)$ evaluated at $t = j$. (You must pay attention to the font in which a letter is written!) The shift of the indices by one is needed because indexed variables in MATLAB begin with the index 1. In particular, t(1) and y(1) denote the initial time t_0 and value y_0. Consequently, all subsequent indices are shifted too, so that t(2) and y(2) denote t_1 and y_1, t(3) and y(3) denote t_2 and y_2, etc.

7.2.2. *Implicit Euler Method.* Alternatively, we could have started with the fact that

$$\lim_{h\to 0} \frac{Y(t) - Y(t-h)}{h} = Y'(t) = f(t, Y(t))\,.$$

Then for small positive h we have

$$\frac{Y(t) - Y(t-h)}{h} \approx f(t, Y(t))\,.$$

Upon solving this for $Y(t-h)$ we find that

$$Y(t-h) \approx Y(t) - hf(t, Y(t))\,.$$

If we let $t = t_{n+1}$ above (so that $t - h = t_n$) this is equivalent to

$$Y(t_{n+1}) - hf(t_{n+1}, Y(t_{n+1})) \approx Y(t_n)\,.$$

Because y_n and y_{n+1} approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$y_{n+1} - hf(t_{n+1}, y_{n+1}) = y_n \quad \text{for } n = 0, 1, \cdots, N-1\,. \tag{7.4}$$

This method is called the *implicit Euler* or *backward Euler* method. It is called the implicit Euler method because equation (7.4) implicitly relates y_{n+1} to y_n. It is called the backward Euler method because the difference quotient upon which it is based steps backward in time (from t to $t-h$). In contrast, the Euler method (7.3) sometimes called the *explicit Euler* or *forward Euler* method because it gives y_{n+1} explicitly and because the difference quotient upon which it is based steps forward in time (from t to $t+h$).

Remark. One step of the implicit Euler method will be much slower than one step of the explicit Euler method unless equation (7.4) can be explicitly solved for y_{n+1}. This can be done when $f(t,y)$ is a fairly simple function if y. For example, this can be done when $f(t,y)$ is linear or quadratic in either y or $\sqrt{y}$. In general equation (7.4) must be solved for y_{n+1} numerically (say by the Newton method), which takes time. However, there are equations for which the implicit Euler method outperforms the explicit Euler method because the explicit Euler method has to take so many more time steps that its speed advantage per time step cannot compensate.

Because of the complications discussed in the above remark, in the rest of this section we will focus on explicit methods. Any exercises on the implicit Euler method will involve an $f(t,y)$ for which equation (7.4) can be solved explicitly for y_{n+1}. The study of more complicated equations are faced in courses on numerical analysis.

7.3. **Explicit One-Step Methods Based on Taylor Approximation.** The explicit (or forward) Euler method can be understood as the first in a sequence of explicit methods that can be derived from the Taylor approximation formula. This view gives a better understanding of how errors arise and accumulate in a numerical approximation.

7.3.1. *Explicit Euler Method Revisited.* The explicit Euler method can be derived from the first-order Taylor approximation, which is also known as the tangent line approximation. This approximation states that if $Y(t)$ is twice continuously differentiable then

$$Y(t+h) = Y(t) + hY'(t) + O(h^2)\,. \tag{7.5}$$

Here the $O(h^2)$ means that the remainder vanishes at least as fast as h^2 as h tends to zero. It is clear from (7.5) that for small positive h we have

$$Y(t+h) \approx Y(t) + hY'(t)\,.$$

Because $Y(t)$ satisfies (7.1), this is the same as

$$Y(t+h) \approx Y(t) + hf(t,Y(t))\,.$$

If we let $t = t_n$ above (so that $t+h = t_{n+1}$) this is equivalent to

$$Y(t_{n+1}) \approx Y(t_n) + hf(t_n,Y(t_n))\,.$$

Because y_n and y_{n+1} approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$y_{n+1} = y_n + hf(t_n,y_n) \qquad \text{for } n = 0,1,\cdots,N-1\,, \tag{7.6}$$

which is exactly the Euler method (7.3). This view of the Euler method is illustrated by the Figure 7.1.

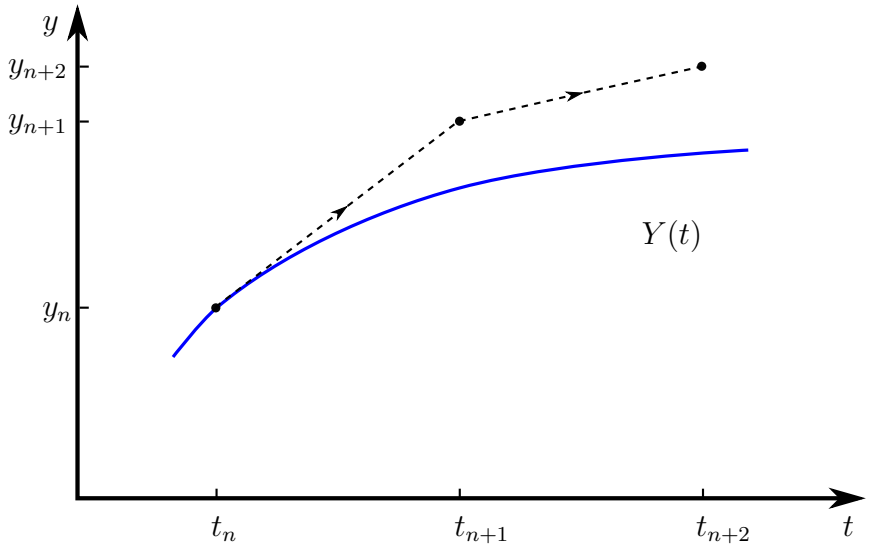

FIGURE 7.1. Illustration of the explicit Euler method.

7.3.2. *Local and Global Errors.* One advantage of viewing the Euler method through the tangent line approximation (7.5) is that we gain some understanding of how its error behaves as we increase N, the number of time steps — or what is equivalent by (7.2), as we decrease h, the step size. The $O(h^2)$ term in (7.5) represents the *local error*, which is error the approximation makes at each step.

Roughly speaking, if we halve the step size h then by (7.5) the local error will reduce by a factor of one quarter, while by (7.2) the number of steps N we must take to get to a prescribed time (say t_F) will double. If we assume that errors add (which is often the case) then the error at t_F will reduce by a factor of one half. In other words, doubling the number of time steps will reduce the error by about a factor of one half. Similarly, tripling the number of time steps will reduce the error by about a factor of one third. Indeed, it can be shown (but we will not do so) that the error of the explicit Euler method is $O(h)$ over the interval $[t_I, t_F]$. The best way to think about this is that if we take N steps and the error made at each step is $O(h^2)$ then we can expect that the acummulation of the local errors will lead to a *global error* of $O(h^2)N$. This is illustrated in Figure 7.2. Because (7.2) states that $hN = t_F - t_I$, which is a number that is independent of h and N, we see that global error of the explicit Euler method is $O(h)$. This was shown by Cauchy in 1824. Moreover, it can be shown that the error of the implicit Euler method behaves the same way.

Global error is a more meaningful concept than local error because it tells us how fast a method converges over the entire interval $[t_I, t_F]$. *Therefore we identify the order of a method by the order of its global error.* In particular, methods like the Euler methods with global errors of $O(h)$ are *first-order methods.* By reasoning similar to that given in the previous paragraph, methods whose local error is $O(h^{m+1})$ will have a global error of $O(h^{m+1})N = O(h^m)$ and thereby are m^{th}*-order methods.*

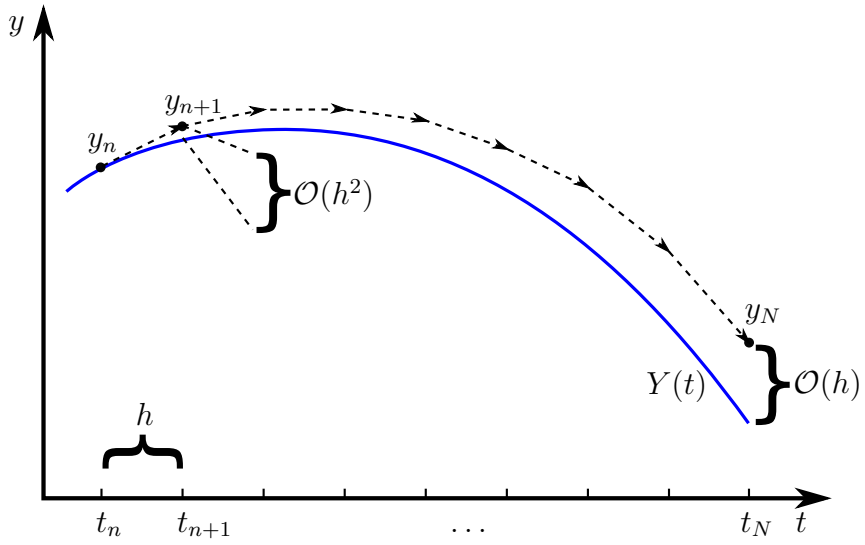

FIGURE 7.2. Illustration of global error arising through the accumulation of local errors for the explicit Euler method.

Higher-order methods are more complicated than the explicit Euler method. The hope is that this cost is overcome by the fact that its error improves faster as you increase N — or what is equivalent by (7.2), as you decrease h. For example, if we halve the step size h of a fourth-order method then the global error will reduce by a factor of $1/16$. Similarly, tripling the number of time steps will reduce the error by about a factor of $1/81$.

7.3.3. *Higher-Order Taylor-Based Methods.* The second-order Taylor approximation states that if $Y(t)$ is thrice continuously differentiable then

$$Y(t+h) = Y(t) + hY'(t) + \tfrac{1}{2}h^2Y''(t) + O(h^3)\,. \tag{7.7}$$

Here the $O(h^3)$ means that the remainder vanishes at least as fast as h^3 as h tends to zero. It is clear from (7.7) that for small positive h we have

$$Y(t+h) \approx Y(t) + hY'(t) + \tfrac{1}{2}h^2Y''(t)\,. \tag{7.8}$$

Because $Y(t)$ satisfies (7.1), we see by the chain rule from multivariable calculus that

$$\begin{aligned} Y''(t) = \frac{\mathrm{d}}{\mathrm{d}t}\big(Y'(t)\big) = \frac{\mathrm{d}}{\mathrm{d}t}f(t,Y(t)) &= \partial_t f(t,Y(t)) + Y'(t)\,\partial_y f(t,Y(t)) \\ &= \partial_t f(t,Y(t)) + f(t,Y(t))\,\partial_y f(t,Y(t))\,. \end{aligned}$$

Hence, equation (7.8) is the same as

$$Y(t+h) \approx Y(t) + hf(t,Y(t)) + \tfrac{1}{2}h^2\Big(\partial_t f(t,Y(t)) + f(t,Y(t))\,\partial_y f(t,Y(t))\Big)\,.$$

If we let $t = t_n$ above (so that $t + h = t_{n+1}$) this is equivalent to

$$Y(t_{n+1}) \approx Y(t_n) + hf(t_n,Y(t_n)) + \tfrac{1}{2}h^2\Big(\partial_t f(t_n,Y(t_n)) + f(t_n,Y(t_n))\,\partial_y f(t_n,Y(t_n))\Big)\,.$$

Because y_n and y_{n+1} approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$y_{n+1} = y_n + hf(t_n, y_n) + \tfrac{1}{2}h^2\Big(\partial_t f(t_n, y_n) + f(t_n, y_n)\,\partial_y f(t_n, y_n)\Big) \quad \text{for } n = 0, 1, \cdots, N-1\,. \tag{7.9}$$

We call this the second-order Taylor-based method.

Remark. We can generalize our derivation of the second-order Taylor-based method by using the m^{th}-order Taylor approximation to derive an explicit numerical method whose error is $O(h^m)$ over the interval $[t_I, t_F]$ — a so-called m^{th}-*order method*. However, the formulas for these methods grow in complexity. For example, the third-order method is

$$\begin{aligned} y_{n+1} = y_n &+ hf(t_n, y_n) + \tfrac{1}{2}h^2\Big(\partial_t f(t_n, y_n) + f(t_n, y_n)\,\partial_y f(t_n, y_n)\Big) \\ &+ \tfrac{1}{6}h^3\Big[\partial_{tt} f(t_n, y_n) + 2f(t_n, y_n)\,\partial_{yt} f(t_n, y_n) + f(t_n, y_n)^2\partial_{yy} f(t_n, y_n) \\ &\qquad + \Big(\partial_t f(t_n, y_n) + f(t_n, y_n)\,\partial_y f(t_n, y_n)\Big)\partial_y f(t_n, y_n)\Big] \\ &\text{for } n = 0, 1, \cdots, N-1\,. \end{aligned} \tag{7.10}$$

This complexity of these methods makes them far less practical for general algorithms than the next class of methods we will study.

7.4. **Explicit One-Step Methods Based on Quadrature.** The starting point for our next class of methods will be the Fundamental Theorem of Calculus — specifically, the fact

$$Y(t+h) - Y(t) = \int_t^{t+h} Y'(s)\,\mathrm{d}s\,.$$

Because $Y(t)$ satisfies (7.1), this becomes

$$Y(t+h) = Y(t) + \int_t^{t+h} f(s, Y(s))\,\mathrm{d}s\,. \tag{7.11}$$

In 1895 Carl Runge proposed using quadrature rules (numerical integration) to construct approximations to the definite integral above in the form

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = K(h, t, Y(t)) + O(h^{m+1})\,, \tag{7.12}$$

where m is some positive integer. The key point here is that $K(h, t, Y(t))$ depends on $Y(t)$, but does not depend on $Y(s)$ for any $s \neq t$. When approximation (7.12) is placed into (7.11) we obtain

$$Y(t+h) = Y(t) + K(h, t, Y(t)) + O(h^{m+1})\,.$$

If we let $t = t_n$ above (so that $t + h = t_{n+1}$) this is equivalent to

$$Y(t_{n+1}) = Y(t_n) + K(h, t_n, Y(t_n)) + O(h^{m+1})\,.$$

Because y_n and y_{n+1} approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$y_{n+1} = y_n + K(h, t_n, y_n) \quad \text{for } n = 0, 1, \cdots, N-1\,, \tag{7.13}$$

Hence, every approximation of the form (7.12) yields the m^{th}-order explicit one-step method (7.13) for approximating solutions of (7.1). Here we will present methods associated with four basic quadrature rules that are covered in most calculus courses: the left-hand rule, the trapezoidal rule, the midpoint rule, and the Simpson rule.

7.4.1. *Explicit Euler Method Revisited Again.* The left-hand rule approximates the definite integral on the left-hand side of (7.12) as

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = hf(t, Y(t)) + O(h^2)\,.$$

This approximation is already in the form (7.12) with $K(h, t, y) = hf(t, y)$. Method (7.13) thereby becomes

$$y_{n+1} = y_n + hf(t_n, y_n) \quad \text{for } n = 0, 1, \cdots, N-1\,,$$

which is exactly the explicit Euler method (7.3).

7.4.2. *Runge-Trapezoidal Method.* The trapezoidal rule approximates the definite integral on the left-hand side of (7.12) as

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = \frac{h}{2}\big[f(t, Y(t)) + f(t+h, Y(t+h))\big] + O(h^3)\,.$$

This approximation is not in the form (7.12) because of the $Y(t+h)$ on the right-hand side. If we approximate this $Y(t+h)$ by the explicit Euler method then we obtain

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = \frac{h}{2}\big[f(t, Y(t)) + f\big(t+h, Y(t) + hf(t, Y(t))\big)\big] + O(h^3)\,.$$

This approximation is in the form (7.12) with

$$K(h, t, y) = \frac{h}{2}\big[f(t, y) + f\big(t+h, y + hf(t, y)\big)\big]\,.$$

Method (7.13) thereby becomes

$$y_{n+1} = y_n + \frac{h}{2}\big[f(t_n, y_n) + f\big(t_{n+1}, y_n + hf(t_n, y_n)\big)\big] \quad \text{for } n = 0, 1, \cdots, N-1\,.$$

This is sometimes called the *improved Euler* method. However, that name is also used for other methods and is not very descriptive. Rather, we will call this the *Runge-trapezoidal* method because it was proposed by Runge based on the trapezoidal rule. This name makes the origins of the method clear.

In practice, the Runge-trapezoidal method is implemented by initializing $y_0 = y_I$ and then for $n = 0, \cdots, N-1$ cycling through the instructions

$$\begin{aligned} f_n &= f(t_n, y_n)\,, & \tilde{y}_{n+1} &= y_n + hf_n\,,\\ \tilde{f}_{n+1} &= f(t_{n+1}, \tilde{y}_{n+1})\,, & y_{n+1} &= y_n + \tfrac{1}{2}h[f_n + \tilde{f}_{n+1}]\,, \end{aligned}$$

where $t_n = t_I + nh$.

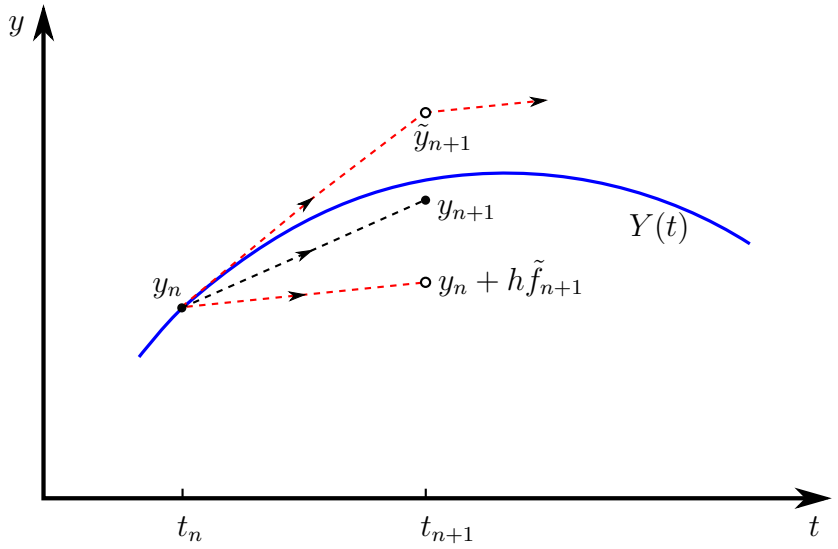

FIGURE 7.3. Illustration of the Runge-trapezoidal method. The method is described as follows: First evaluate y_{n+1} using the explicit Euler method, then find $\tilde{f}_{n+1}$ by evaluating $f(y,t)$ at $\tilde{y}_{n+1}$ and t_{n+1}. Finally y_{n+1} is the midpoint of $\tilde{y}_{n+1}$ and the "correction" $y_n + h\tilde{f}_{n+1}$. Notice how the line leaving $\tilde{y}_{n+1}$ is parallel to the segment between y_n and $y_n + h\tilde{f}_{n+1}$.

Example. Let $y(t)$ be the solution of the initial-value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = t^2 + y^2\,, \qquad y(0) = 1\,.$$

Use the Runge-trapezoidal method with $h = .2$ to approximate $y(.2)$.

Solution. We initialize $t_0 = 0$ and $y_0 = 1$. The Runge-trapezoidal method then gives

$$f_0 = f(t_0, y_0) = 0^2 + 1^2 = 1$$
$$\tilde{y}_1 = y_0 + hf_0 = 1 + .2 \cdot 1 = 1.2$$
$$\tilde{f}_1 = f(t_1, \tilde{y}_1) = (.2)^2 + (1.2)^2 = .04 + 1.44 = 1.48$$
$$y_1 = y_0 + \tfrac{1}{2}h\big[f_0 + \tilde{f}_1\big] = 1 + .1 \cdot (1 + 1.48) = 1 + .1 \cdot 2.48 = 1.248$$

We then have $y(.2) \approx y_1 = 1.248$. □

Remark. Notice that two steps of the explicit Euler method with $h = .1$ gave $y(.2) \approx 1.222$, while one step of the Runge-trapezoidal method with $h = .2$ gave $y(.2) \approx 1.248$, which is much closer to the exact value. As these two calculations required roughly the same computational effort, this shows the advantange of using the second-order method.

The Runge-trapezoidal method is implemented by the following MATLAB function M-file.

```
function [t,y] = RungeTrap(f, tI, yI, tF, N)

t = zeros(N + 1, 1); y = zeros(N + 1, 1);
t(1) = tI; y(1) = yI; h = (tF - tI)/N; hhalf = h/2;
for j = 1:N
t(j + 1) = t(j) + h;
fnow = f(t(j), y(j));
yplus = y(j) + h*fnow; fplus = f(t(j + 1), yplus);
y(j + 1) = y(j) + hhalf*(fnow + fplus);
end
```

Remark. Here t(j) and y(j) have the same meaning as they did in the M-file for the explicit Euler method. In particular, we have the same shift of the indices by one. Here we have introduced the so-called *working variables* fnow, yplus, and fplus to temporarily hold the values of f_{j-1}, $\tilde{y}_j$, and $\tilde{f}_j$ during each cycle of the loop. These values do not have to be saved, and so are overwritten with each new cycle. Here we have isolated the function evaluations for fnow and fplus into two separate instructions. This is good coding practice that makes adaptations easier. For example, you can replace the function calls to f(t,y) by explicit formulas in those two lines without changing the rest of the coding.

7.4.3. *Runge-Midpoint Method.* The midpoint rule approximates the definite integral on the left-hand side of (7.12) as

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = hf\big(t + \tfrac{1}{2}h, Y(t + \tfrac{1}{2}h)\big) + O(h^3)\,.$$

This approximation is not in the form (7.12) because of the $Y(t+\frac{1}{2}h)$ on the right-hand side. If we approximate this $Y(t + \frac{1}{2}h)$ by the explicit Euler method then we obtain

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = hf\big(t + \tfrac{1}{2}h, Y(t) + \tfrac{1}{2}hf(t, Y(t))\big) + O(h^3)\,.$$

This approximation is in the form (7.12) with

$$K(h, t, y) = hf\big(t + \tfrac{1}{2}h, y + \tfrac{1}{2}hf(t, y)\big)\,.$$

Method (7.13) thereby becomes

$$y_{n+1} = y_n + hf\big(t_{n+\frac{1}{2}}, y_n + \tfrac{1}{2}hf(t_n, y_n)\big) \quad \text{for } n = 0, 1, \cdots, N-1\,.$$

This is sometimes called the *modified Euler* method. However, that name is also used for other methods and is not very descriptive. Rather, we will call this the *Runge-midpoint* method because it was proposed by Runge based on the midpoint rule. This name makes the origins of the method clear.

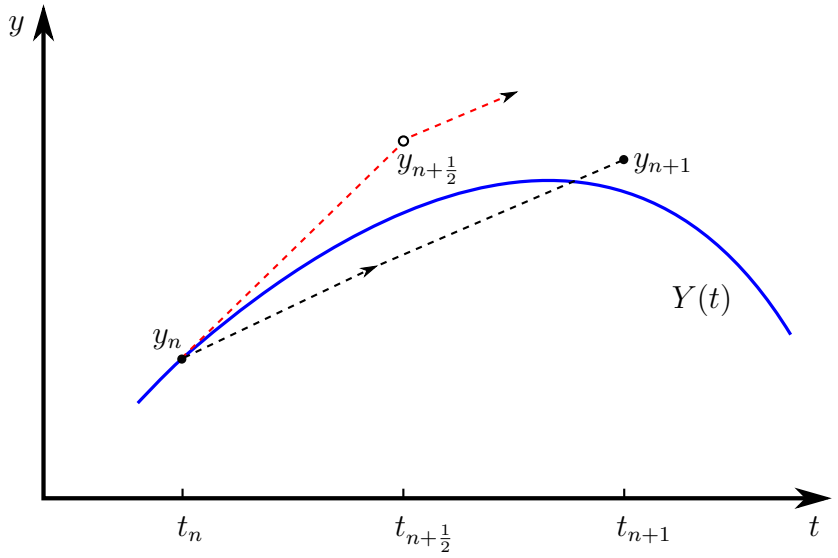

FIGURE 7.4. Illustration of the Runge-midpoint method. The method is described as follows: First evalute $y_{n+\frac{1}{2}}$ by taking the midpoint of the segment between y_n and the value $y_n + hf(y_n, t_n)$ predicted by the explicit Euler method. Next, find $f_{n+\frac{1}{2}}$ by evaluating $f(y,t)$ at $y_{n+\frac{1}{2}}$ and $t_{n+\frac{1}{2}}$. Fnally, find y_{n+1} by stepping from y_n in the direction of $f_{n+\frac{1}{2}}$, that is $y_{n+1} = y_n + hf_{n+\frac{1}{2}}$. Notice how the line leaving $y_{n+\frac{1}{2}}$ is parallel to the segment between y_n and y_{n+1}.

In practice, the Runge-midpoint method is implemented by initializing $y_0 = y_I$ and then for $n = 0, \cdots, N-1$ cycling through the instructions

$$\begin{aligned} f_n &= f(t_n, y_n)\,, & y_{n+\frac{1}{2}} &= y_n + \tfrac{1}{2}hf_n\,, \\ f_{n+\frac{1}{2}} &= f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})\,, & y_{n+1} &= y_n + hf_{n+\frac{1}{2}}\,, \end{aligned}$$

where $t_n = t_I + nh$ and $t_{n+\frac{1}{2}} = t_I + (n+\frac{1}{2})h$.

Remark. The half-integer subscripts on $t_{n+\frac{1}{2}}$, $y_{n+\frac{1}{2}}$, and $f_{n+\frac{1}{2}}$ indicate that those variables are associated with the time $t = t_n + \frac{1}{2}h$, which is halfway between the times t_n and t_{n+1}. While it may seem strange at first, this notational device is a handy way to help keep track of the meanings of different variables.

Example. Let $y(t)$ be the solution of the initial-value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = t^2 + y^2\,, \qquad y(0) = 1\,.$$

Use the Runge-midpoint method with $h = .2$ to approximate $y(.2)$.

Solution. We initialize $t_0 = 0$ and $y_0 = 1$. Then the Runge-midpoint method gives

$$\begin{aligned} f_0 &= f(t_0, y_0) = 0^2 + 1^2 = 1 \\ y_{\frac{1}{2}} &= y_0 + \tfrac{1}{2} h f_0 = 1 + .1 \cdot 1 = 1.1 \\ f_{\frac{1}{2}} &= f(t_{\frac{1}{2}}, y_{\frac{1}{2}}) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22\,, \\ y_1 &= y_0 + h f_{\frac{1}{2}} = 1 + .2 \cdot (1.22) = 1 + .244 = 1.244\,. \end{aligned}$$

We then have $y(.2) \approx y_1 = 1.244$. □

Remark. Notice that the Runge-trapezoidal method gave $y(.2) \approx 1.248$ while the Runge-midpoint method gave $y(.2) \approx 1.244$. The results are about the same because both methods are second-order. Here the Runge-trapezoidal method gave a better approximation. For other problems the Runge-midpoint method might give a better approximation.

The Runge-midpoint method is implemented by the following MATLAB function M-file.

```
function [t,y] = RungeMid(f, tI, yI, tF, N)

t = zeros(N + 1, 1); y = zeros(N + 1, 1);
t(1) = tI; y(1) = yI; h = (tF - tI)/N; hhalf = h/2;
for j = 1:N
thalf = t(j) + hhalf;
t(j + 1) = t(j) + h;
fnow = f(t(j), y(j));
yhalf = y(j) + hhalf*fnow; fhalf = f(thalf, yhalf);
y(j + 1) = y(j) + h*fhalf;
end
```

Remark. Here t(j) and y(j) have the same meaning as they did in the M-file for the explicit Euler method. In particular, we have the same shift of the indices by one. Here we have introduced the working variables fnow, thalf, yhalf, and fhalf to temporarily hold the values of f_{j-1}, $t_{j-\frac{1}{2}}$, $y_{j-\frac{1}{2}}$, and $f_{j-\frac{1}{2}}$ during each cycle of the loop. These values do not have to be saved, and so are overwritten with each new cycle.

7.4.4. *Classical Runge-Kutta Method.* The Simpson rule approximates the definite integral on the left-hand side of (7.12) as

$$\int_t^{t+h} f(s, Y(s))\,\mathrm{d}s = \frac{h}{6}\big[f(t, Y(t)) + 4f\big(t+\tfrac{1}{2}h, Y(t+\tfrac{1}{2}h)\big) + f\big(t+h, Y(t+h)\big)\big] + O(h^5)\,.$$

This approximation is not in the form (7.12) because of the $Y(t + \frac{1}{2}h)$ and $Y(t + h)$ on the right-hand side. If we approximate these with the explicit Euler method as we did before then we will degrade the local error to $O(h^3)$. We would like to find an approximation that is consistent with the $O(h^5)$ local error of the Simpson rule. In 1901 Wilhelm Kutta found such an approximation, which led to the so-called *Runge-Kutta*

method. We will not give a derivation of this method here. Such derivations can be found in numerical analysis books.

In practice the Runge-Kutta method is implemented by initializing $y_0 = y_I$ and then for $n = 0, \cdots, N-1$ cycling through the instructions

$$\begin{aligned} f_n &= f(t_n, y_n)\,, & \tilde{y}_{n+\frac{1}{2}} &= y_n + \tfrac{1}{2}hf_n\,, \\ \tilde{f}_{n+\frac{1}{2}} &= f(t_{n+\frac{1}{2}}, \tilde{y}_{n+\frac{1}{2}})\,, & y_{n+\frac{1}{2}} &= y_n + \tfrac{1}{2}h\tilde{f}_{n+\frac{1}{2}}\,, \\ f_{n+\frac{1}{2}} &= f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})\,, & \tilde{y}_{n+1} &= y_n + hf_{n+\frac{1}{2}}\,, \\ \tilde{f}_{n+1} &= f(t_{n+1}, \tilde{y}_{n+1})\,, & y_{n+1} &= y_n + \tfrac{1}{6}h\big[f_n + 2\tilde{f}_{n+\frac{1}{2}} + 2f_{n+\frac{1}{2}} + \tilde{f}_{n+1}\big]\,, \end{aligned}$$

where $t_n = t_I + nh$ and $t_{n+\frac{1}{2}} = t_I + (n+\frac{1}{2})h$.

Remark. This Runge-Kutta method requires four evaluations of $f(t,y)$ to advance each time step, whereas the second-order methods each required only two. Therefore it requires roughly twice as much computational work per time step as those methods.

Remark. Notice that because

$$\begin{aligned} y_n &\approx Y(t_n)\,, & f_n &\approx f\big(t_n, Y(t_n)\big) \\ \tilde{y}_{n+\frac{1}{2}} &\approx Y(t_n + \tfrac{1}{2}h)\,, & \tilde{f}_{n+\frac{1}{2}} &\approx f\big(t_n + \tfrac{1}{2}h, Y(t_n + \tfrac{1}{2}h)\big)\,, \\ y_{n+\frac{1}{2}} &\approx Y(t_n + \tfrac{1}{2}h)\,, & f_{n+\frac{1}{2}} &\approx f\big(t_n + \tfrac{1}{2}h, Y(t_n + \tfrac{1}{2}h)\big)\,, \\ \tilde{y}_{n+1} &\approx Y(t_n + h)\,, & \tilde{f}_{n+1} &\approx f\big(t_n + h, Y(t_n + h)\big)\,, \end{aligned}$$

we see that

$$y_{n+1} \approx Y(t_n) + \frac{h}{6}\big[f(t_n, Y(t_n)) + 4f\big(t_n + \tfrac{1}{2}h, Y(t_n + \tfrac{1}{2}h)\big) + f\big(t_n + h, Y(t_n + h)\big)\big]\,.$$

This Runge-Kutta method thereby looks consistent with the Simpson rule approximation. This argument does not show that the Runge-Kutta method is fourth order, but it is.

Example. Let $y(t)$ be the solution of the initial-value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = t^2 + y^2\,, \qquad y(0) = 1\,.$$

Use the Runge-Kutta method with $h = .2$ to approximate $y(.2)$.

Solution. We initialize $t_0 = 0$ and $y_0 = 1$. The Runge-Kutta method then gives

$$\begin{aligned}
f_0 &= f(t_0, y_0) = 0^2 + 1^2 = 1 \\
\tilde{y}_{\frac{1}{2}} &= y_0 + \tfrac{1}{2} h f_0 = 1 + .1 \cdot 1 = 1.1 \\
\tilde{f}_{\frac{1}{2}} &= f(t_{\frac{1}{2}}, \tilde{y}_{\frac{1}{2}}) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22 \\
y_{\frac{1}{2}} &= y_0 + \tfrac{1}{2} h \tilde{f}_{\frac{1}{2}} = 1 + .1 \cdot 1.22 = 1.122 \\
f_{\frac{1}{2}} &= f(t_{\frac{1}{2}}, y_{\frac{1}{2}}) = (.1)^2 + (1.122)^2 = .01 + 1.258884 = 1.268884 \\
\tilde{y}_1 &= y_0 + h f_{\frac{1}{2}} = 1 + .2 \cdot 1.268884 = 1 + .2517768 = 1.2517768 \\
\tilde{f}_1 &= f(t_1, \tilde{y}_1) = (.2)^2 + (1.2517768)^2 \approx .04 + 1.566945157 = 1.606945157 \\
y_1 &= y_0 + \tfrac{1}{6} h \big[f_0 + 2\tilde{f}_{\frac{1}{2}} + 2 f_{\frac{1}{2}} + \tilde{f}_1 \big] \\
&\approx 1 + .033333333 \big[1 + 2 \cdot 1.22 + 2 \cdot 1.268884 + 1.606945157 \big] .
\end{aligned}$$

We then have $y(.2) \approx y_1 \approx 1.252823772$. Of course, you would not be expected to carry out such arithmetic calculations to nine decimal places on an exam. □

Remark. One step of this Runge-Kutta method with $h = .2$ yielded the approximation $y(.2) \approx 1.252823772$. This is more accurate than the approximations we had obtained with either second-order method. However, that is not a fair comparison because the Runge-Kutta method required roughly twice the computational work. A better comparison would be with the approximation produced by two steps of either second-order method with $h = .1$.

Remark. You will not be required to memorize this method. You also will not be required to carry out one step of it on an exam or quiz because, as the above example illustrates, the arithmetic gets messy even for fairly simple differential equations. However, you should understand the implications of it being a fourth-order method — namely, the relationship between its error and the step size h. You also should be able to recognize the Runge-Kutta method if it is presented to you in MATLAB code.

This Runge-Kutta method is implemented by the following MATLAB function M-file.

```
function [t,y] = RungeKutta(f, tI, yI, tF, N)

t = zeros(N + 1, 1); y = zeros(N + 1, 1);
t(1) = tI; y(1) = yI; h = (tF - tI)/N; hhalf = h/2; hsixth = h/6;
for j = 1:N
thalf = t(j) + hhalf;
t(j + 1) = t(j) + h;
fnow = f(t(j), y(j));
yhalfone = y(j) + hhalf*fnow; fhalfone = f(thalf, yhalfone);
yhalftwo = y(j) + hhalf*fhalfone; fhalftwo = f(thalf, yhalftwo);
yplus = y(j) + h*fhalftwo; fplus = f(t(j + 1), yplus);
y(j + 1) = y(j) + hsixth*(fnow + 2*fhalfone + 2*fhalftwo + fplus);
end
```

Remark. Here t(j) and y(j) have the same meaning as they did in the M-file for the explicit Euler method. In particular, we have the same shift of the indices by one. Here we have introduced the working variables fnow, thalf, yhalfone, fhalfone, yhalftwo, fhalftwo, yplus, and fplus to temporarily hold the values of f_{j-1}, $t_{j-\frac{1}{2}}$, $\tilde{y}_{j-\frac{1}{2}}$, $\tilde{f}_{j-\frac{1}{2}}$, $y_{j-\frac{1}{2}}$, $f_{j-\frac{1}{2}}$, $\tilde{y}_j$, and $\tilde{f}_j$.

7.5. **General Explicit Runge-Kutta Methods.** All the methods presented in the previous section are members of the family of general Runge-Kutta methods. The MATLAB command "ode45" uses the Dormand-Prince method, which is another member of this Runge-Kutta family that was discovered in 1980! The Runge-Kutta family continues to be enlarged by new methods, some of which might replace the Dormand-Prince method in future versions of MATLAB. An introduction to these modern methods requires a graduate course in numerical analysis. Here we have the more modest goal of introducing those family members presented by Wilhelm Kutta in his 1901 paper.

Carl Runge had described just a few methods in his 1895 paper, but these included the Runge-trapezoid and Runge-midpoint methods. In 1900 Karl Heun presented a family of methods that included all of those studied by Runge as special cases. Heun characterized the computational effort of these methods by how many evaluations of $f(t, y)$ are needed to compute $K(h, t, y)$. A method that requires s evaluations of $f(t, y)$ is called an s-stage method. The explicit Euler method, for which $K(h, t, y) = hf(t, y)$, is the only one-stage method.

7.5.1. *Two-Stage Methods.* Heun considered the family of two-stage methods in the form

$$K(h, t, y) = \alpha_1 k_1 + \alpha_2 k_2 \,, \qquad \text{with} \quad \alpha_1 + \alpha_2 = 1 \,, \tag{7.14a}$$

where k_1 and k_2 are given by two evaluations of $f(t, y)$ as

$$k_1 = hf(t, y) \,, \qquad k_2 = hf(t + \beta h, y + \beta k_1) \,, \qquad \text{for some } \beta > 0 \,. \tag{7.14b}$$

Heun showed the two-stage method (7.14) is second-order for every $f(t, y)$ if and only if

$$\alpha_1 = 1 - \frac{1}{2\beta} \,, \qquad \alpha_2 = \frac{1}{2\beta} \,.$$

These include the Runge-trapezoidal method, which is given by $\alpha_1 = \alpha_2 = \frac{1}{2}$ and $\beta = 1$, and the Runge-midpoint method, which is given by $\alpha_1 = 0$, $\alpha_2 = 1$, and $\beta = \frac{1}{2}$. Heun also showed that no two-stage method (7.14) is third-order for every $f(t, y)$.

Remark. Second-order, two-stage methods are often called Heun methods in recognition of his work. Of these Heun favored the method given by $\alpha_1 = \frac{1}{4}$, $\alpha_2 = \frac{3}{4}$, and $\beta = \frac{2}{3}$, which is third-order in the special case when $\partial_y f = 0$.

7.5.2. *Three-Stage Methods.* Heun also considered families of three- and four-stage methods in his 1900 paper. However in 1901 Kutta introduced families of s-stage methods that are more general when $s \geq 3$. For example, Kutta considered the family of three-stage methods in the form

$$K(h,t,y) = \alpha_1 k_1 + \alpha_2 k_2 + \alpha_3 k_3\,, \qquad \text{with} \quad \alpha_1 + \alpha_2 + \alpha_3 = 1\,, \tag{7.15a}$$

where k_1, k_2, and k_3 are given by three evaluations of $f(t,y)$ as

$$\begin{aligned} k_1 &= hf(t,y)\,, \\ k_2 &= hf(t+\beta_2 h, y+\gamma_{21}k_1)\,, && \text{with} \quad \beta_2 = \gamma_{21}\,, \\ k_3 &= hf(t+\beta_3 h, y+\gamma_{31}k_1+\gamma_{32}k_2)\,, && \text{with} \quad \beta_3 = \gamma_{31}+\gamma_{32}\,. \end{aligned} \tag{7.15b}$$

Kutta showed the three-stage method (7.15) is second-order for every $f(t,y)$ if and only if

$$\alpha_2\beta_2 + \alpha_3\beta_3 = \tfrac{1}{2}\,;$$

and is third-order for every $f(t,y)$ if and only if in addition

$$\alpha_2\beta_2^2 + \alpha_3\beta_3^2 = \tfrac{1}{3}\,, \qquad \alpha_3\gamma_{32}\beta_2 = \tfrac{1}{6}\,.$$

Kutta also showed that no three-stage method (7.15) is fourth-order for every $f(t,y)$. Heun had shown the analogus results restricted to the case $\gamma_{31} = 0$. He favored the third-order method given by

$$\alpha_1 = \tfrac{1}{4}\,, \quad \alpha_2 = 0\,, \quad \alpha_3 = \tfrac{3}{4}\,, \quad \beta_2 = \gamma_{21} = \tfrac{1}{3}\,, \quad \beta_3 = \gamma_{32} = \tfrac{2}{3}\,, \quad \gamma_{31} = 0\,,$$

which is the third-order method requiring the fewest arithmetic operations. Kutta favored the third-order method given by

$$\alpha_1 = \tfrac{1}{6}\,, \quad \alpha_2 = \tfrac{2}{3}\,, \quad \alpha_3 = \tfrac{1}{6}\,, \quad \beta_2 = \gamma_{21} = \tfrac{1}{2}\,, \quad \beta_3 = 1\,, \quad \gamma_{31} = -1\,, \quad \gamma_{32} = 2\,,$$

which agrees with the Simpson rule in the special case when $\partial_y f = 0$.

7.5.3. *Four-Stage Methods.* Similarly, Kutta considered the family of four-stage methods in the form

$$K(h,t,y) = \alpha_1 k_1 + \alpha_2 k_2 + \alpha_3 k_3 + \alpha_4 k_4\,, \qquad \text{with} \quad \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1\,, \tag{7.16a}$$

where k_1, k_2, k_3, and k_4 are given by four evaluations of $f(t,y)$ as

$$\begin{aligned} k_1 &= hf(t,y)\,, \\ k_2 &= hf(t+\beta_2 h, y+\gamma_{21}k_1)\,, && \text{with} \quad \beta_2 = \gamma_{21}\,, \\ k_3 &= hf(t+\beta_3 h, y+\gamma_{31}k_1+\gamma_{32}k_2)\,, && \text{with} \quad \beta_3 = \gamma_{31}+\gamma_{32}\,, \\ k_4 &= hf(t+\beta_4 h, y+\gamma_{41}k_1+\gamma_{42}k_2+\gamma_{43}k_3)\,, && \text{with} \quad \beta_4 = \gamma_{41}+\gamma_{42}+\gamma_{43}\,. \end{aligned} \tag{7.16b}$$

Kutta showed the four-stage method (7.16) is second-order for every $f(t,y)$ if and only if

$$\alpha_2\beta_2 + \alpha_3\beta_3 + \alpha_4\beta_4 = \tfrac{1}{2}\,;$$

is third-order for every $f(t,y)$ if and only if in addition

$$\alpha_2\beta_2^2 + \alpha_3\beta_3^2 + \alpha_4\beta_4^2 = \tfrac{1}{3}\,, \qquad \alpha_3\gamma_{32}\beta_2 + \alpha_4\big(\gamma_{42}\beta_2 + \gamma_{43}\beta_3\big) = \tfrac{1}{6}\,;$$

and is fourth-order for every $f(t,y)$ if and only if in addition

$$\alpha_2\beta_2^3 + \alpha_3\beta_3^3 + \alpha_4\beta_4^3 = \tfrac{1}{4}\,, \qquad \alpha_3\gamma_{32}\beta_2^2 + \alpha_4\big(\gamma_{42}\beta_2^2 + \gamma_{43}\beta_3^2\big) = \tfrac{1}{12}\,,$$
$$\alpha_3\beta_3\gamma_{32}\beta_2 + \alpha_4\beta_4\big(\gamma_{42}\beta_2 + \gamma_{43}\beta_3\big) = \tfrac{1}{8}\,, \qquad \alpha_4\gamma_{43}\gamma_{32}\beta_2 = \tfrac{1}{24}\,.$$

Kutta also showed that no four-stage method (7.16) is fifth-order for every $f(t,y)$. Heun had shown the analogus results restricted to the case $\gamma_{31} = \gamma_{41} = \gamma_{42} = 0$. Kutta favored the classical Runge-Kutta method presented in the previous subsection, which is given by

$$\alpha_1 = \tfrac{1}{6}\,, \quad \alpha_2 = \tfrac{1}{3}\,, \quad \alpha_3 = \tfrac{1}{3}\,, \quad \alpha_4 = \tfrac{1}{6}\,,$$
$$\beta_2 = \gamma_{21} = \tfrac{1}{2}\,, \quad \beta_3 = \gamma_{32} = \tfrac{1}{2}\,, \quad \gamma_{31} = 0\,, \quad \beta_4 = \gamma_{43} = 1\,, \quad \gamma_{41} = \gamma_{42} = 0\,.$$

This is the fourth-order method that both requires the fewest arithmetic operations and is consistant with the Simpson rule.

7.5.4. *Higher-Stage Methods.* More generally, Kutta considered the family of s-stage methods in the form

$$K(h,t,y) = \sum_{j=1}^{s} \alpha_j k_j\,, \qquad \text{with} \quad \sum_{j=1}^{s} \alpha_j = 1\,, \tag{7.17a}$$

where k_j for $j = 1, \cdots, s$ are given by s evaluations of $f(t,y)$ as

(7.17b)

$$k_1 = hf(t,y)\,,$$
$$k_j = hf\left(t + \beta_j h\,,\; y + \sum_{i=1}^{j-1} \gamma_{ji} k_i\right)\,, \qquad \text{with} \quad \beta_j = \sum_{i=1}^{j-1} \gamma_{ji}\,, \quad \text{for } j = 2, \cdots, s\,.$$

Kutta showed that no five-stage method (7.17) is fifth-order for every $f(t,y)$. This result was surprising because for $s = 1$, 2, 3, and 4 there were s-stage methods that were s^{th}-order. Kutta then characterized those six-stage methods (7.17) that are fifth-order for every $f(t,y)$. We will not give the conditions he found here.

Remark. Programmable electronic computers were invented over fifty years after Runge, Heun, and Kutta carried out their work. Early numerical computations had less precision than they do today. Higher-order methods suffer from round-off error more than lower-order methods. Because round-off error is larger on machines with less precision, there was little advantage to using higher-order methods on early machines. As machines became more precise, the classical Runge-Kutta method became widely used to solve differential equations because it offers a nice balance between order and round-off error.

Remark. One of the most important developments in Runge-Kutta methods since their invention is *embedded methods*, which emerged in the 1950s. These methods maintain a prescribed error tolerance by selecting a different h for each time step based upon an error estimate made by comparing related Runge-Kutta methods of orders m and $m+1$. By "related" we mean that the methods are built from the same evaluations of $f(t,y)$, so that they can be computed simultaneously. The MATLAB command "ode45" uses

a fourth-order/fifth-order Runge-Kutta embedded method. Originally it used a fourth-order method invented by Fehlberg in 1969, sometimes denoted RKF4(5). Currently it uses a fifth-order method invented by J.R. Dormand and P.J. Prince in 1980, sometimes denoted RKDP5(4). This method might be replaced by a higher-order embedded method as faster machines with smaller round-off error become more common. One candidate to fill this role is an eighth-order method invented by Dormand and Prince in 1981, a seventh-order/eighth-order Runge-Kutta embedded method sometimes denoted RKDP8(7). There are other candidates.

Exercises on Numerical Methods

(1) Suppose you're working for a government agency and your work involves numerically solving differential equations. The regulations require your measurements to be accurate to within 0.0001. You're using an algorithm with fourth-order global error, and after running a simulation using three steps, you've estimated that the error does not exceed 13000. (It's three pretty big steps.) How many times must you halve the step size before you're within the required tolerance?

Short Answer
Solution

(2) You've used a numerical method with second-order global error to estimate the value of a solution of a differential equation, and using 4 time steps you've got an error not exceeding 0.25. How many times must you halve the step size to be sure the error is less than 0.0001? How does the answer change if the method were fifth-order instead?

Short Answer
Solution

(3) Suppose $y(t)$ is the solution of the differential equation $y' + y = t^2 + 1$ satisfying the initial condition $y(1) = 2$. Estimate $y(2)$ using Euler's method with a step size of 0.5.

Short Answer
Solution

(4) Suppose $y(t)$ solves the initial value problem

$$\dot{y} + \frac{1}{1+t}\, y = t^2\,, \qquad y(0) = 1\,.$$

Estimate $y(1)$ using Euler's method and four steps. [You can work to four decimal places of precision if you'd prefer to avoid fractions.] If you can, write a computer program that allows you to run Euler's method on this question with 100 steps.

Short Answer
Solution

(5) Suppose $y(t)$ solves the differential equation $y' + y = 2t$ with initial condition $y(2) = 0$.
   (a) Use the implicit Euler method with two steps to estimate $y(3)$.
   (b) Solve the equation and determine the actual value of $y(3)$ for the solution with a zero at $t = 2$.
   (c) What if we repeat part (a) with four steps? Does the estimate get any closer to the truth?

Short Answer
Solution

(6) Let $y(t)$ be a solution to $y' + \frac{1}{2}ty^2 = 2$ with $y(2) = 1$. Use implicit Euler with a step size $h = 1$ to estimate an approximate value of $y(5)$. [Let's focus on the

solution that will come up if we take positive square roots at every juncture possible.]

Short Answer
Solution

(7) Let $y(t)$ be the solution to the initial-value problem $y' = \sqrt{y}$, $y(0) = 3$. Using four steps, estimate $y(1)$ with the implicit Euler method. [Take positive square roots should the need arise, and use four decimals of precision.] [*Hint.* Though it doesn't look like it at first, $y + \alpha\sqrt{y}$ is the sort of thing that's amenable to completing the square, where α is some constant.]

Short Answer
Solution

(8) Consider the solution to the differential equation $\dot{y} = -100y$ with $y(0) = 1$.
   (a) Use explicit (forward, usual) Euler's method with a step size of 0.2 to estimate the value of $y(1)$.
   (b) Repeat part (a) with the same step size but use implicit (backwards) Euler instead.
   (c) What function are we approximating here, and which method got closer to the correct value?
   (d) Does increasing the number of steps help with any shortcomings you saw with either of the methods? Why or why not?

Short Answer
Solution

(9) Use the Runge-midpoint method to estimate $y(0.2)$ if $y(t)$ is the solution to $y' = \sqrt{y + t^2 + 1}$ with $y(0) = 0$. Use a step size of $h = 0.1$.

Short Answer
Solution

(10) Use the Runge-trapezoid method to calculate $y(1.4)$ if $y(t)$ is the solution to the differential equation $y' = \frac{1+y^2}{1+t^2}$ with $y(1) = 2$. Use a step size of $h = 0.2$.

Short Answer
Solution

(11) Use the Runge-Kutta method with to calculate $y(1.5)$ if $y(t)$ is the solution to $y' + y = \frac{1}{t}$ satisfying $y(1) = 1$. Use a step size of $h = 0.5$ (so just one step).

Short Answer
Solution

(12) Use the second-order Taylor method to estimate $y(2)$ if $y(t)$ is the solution to $\dot{y} = t + \frac{1}{y}$ with $y(1) = 2$. Use a step size of $h = 0.5$.

Short Answer
Solution

(13) The direction field for the equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = 5y(1-y) - \left(1 + \frac{\sin(2\pi t)}{2}\right)$$

may appear chaotic at first glance. We'll try to make sense of it in this exercise.

(a) Plot this direction field for $0 < t < 6$, $0 < y < 1$. Can you tell what is the long-term behavior of solutions from the picture?
(b) Use `ode45` to generate solutions to the equation with initial conditions $y(0) \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, again on the range $0 < t < 6$. Plot them all on the same graph, and speculate as to what's going on.
(c) Overlay the figures from part (a) and part (b) to see how the solutions followed the direction field.

Solution

(14) The goal of this problem is to use numerical methods to confirm our theorems regarding autonomous differential equations. We'll look at the equation

$$y' = -y^4 + 4y^3 - y^2 - 6y = -y(y+1)(y-2)(y-3)\,.$$

(a) Draw a phase line and indicate the kind of stationary solutions this equation has.
(b) Where should a solution with $y(0) = 1.2$ go as $t \to \infty$? Using a computer, perform Euler's method with 100 steps to estimate $y(10)$ to try to confirm this. Use `ode45` to plot a solution.
(c) Repeat the previous part using $y(0) = 2.1$.

Solution

(15) In this problem we'll try to "see" that the Runge-Kutta method has an error term of fourth order only by looking at its estimates for a specific problem. To do this we'll have to know what the error actually is, so we'll pick a straightforward differential equation whose solution we can get our hands on:

$$y' - 2y = e^t\,, \qquad y(0) = 2\,.$$

(a) Find the solution to this initial value problem; call it $y(t)$. Calculate $y(1)$.
(b) Using a computer programmed to perform the Runge-Kutta method, estimate $y(1)$ by using 1, 2, 10, 20, 32, 50, 64, and 100 steps. Make a table with the number of steps, the estimated value that Runge-Kutta gives, and the error (the absolute value of the difference between the answer to part (a) and the estimate). You'll need at least eight to ten digits of precision.
(c) For $h = 1$, 10, 32, and 50, calculate

$$\log\left(\frac{\text{error using } h \text{ steps}}{\text{error for } 2h \text{ steps}}\right) \cdot \frac{1}{\log(2)}\,.$$

What is this number telling us, and why do we expect it to be about 4 if it's the algorithm is fourth order? [*Note.* The fact that we get numbers near 4 for this problem is not enough evidence to conclude that the method is fourth order; in fact it's possible for these numbers to be smaller for specific problems. But generally speaking they will be around four.]

Solution

(16) As humans we sometimes take for granted that time can only move forwards, but when we're working with differential equations time can move both ways. We'll explore this in this problem, where we put the "backwards" in "backwards Euler method".

(a) Consider the differential equation

$$ty' - y = t^4 - 3\,,$$

with "terminal condition" $y(1) = 0$. Use the implicit Euler method

$$y_n = y_{n+1} - hf(t_{n+1}, y_{n+1})$$

with step size $h = \frac{1}{2}$ to estimate $y(0)$.

(b) Solve the initial value problem and determine what value of $y(0)$ would lead to a solution which has a root at $t = 1$. [Do you even need the specific solution to determine $y(0)$? Is something weird going on here?]

(c) Check that upping the number of steps from two to four improves the accuracy of the approximation.

Solution

(17) Let $y(t)$ be the solution to the differential equation $y' - 3y = t^2$ with $y(1) = 1$. Use the explicit Euler method with a step size of $\frac{1}{2}$ to estimate $y(0)$. Was this easier or harder than the previous problem?

Solution

(18) You've estimated the error of a fourth order method on an interval $[0, 10]$ with 20 time steps. If you increase to 60 time steps, by what factor will the error go down?

Short Answer
Solution

(19) You've estimated the error of a Forward Euler method on an interval $[0, 10]$ with 20 time steps. If you increase to 100 time steps, by what factor will the error go down?

Short Answer
Solution

(20) Suppose $x(t)$ solves the differential equation $x' = -2tx^2$ with initial condition x(1) = 1. Solve the above IVP for x(2) using:
a) Euler method with step size 0.5
b) Analytically.

Short Answer
Solution

(21) Use the explicit Euler method to plot the estimate $y(2)$ if $y(t)$ is the solution to $y' = t - y^3$ with $y(0) = 1$. Use 10 steps, 20 steps, and 30 steps and plot all three.

Solution

(22) Use the Explicit Euler and Runge-Kutta methods to estimate $y(2)$ if $y(t)$ is a solution to $y' = \frac{1}{t^2+y}$ with $y(0) = 1$ and 10 time steps.

Solution

Navigation to Other Chapters

This page may not work with every browser-driven pdf viewer. When it does work then it will enable you to link directly to any chapter. When it does not work then you can link to any chapter through the main webpage.