APPENDIX E

Numerical Methods

APPENDIX OUTLINE

E.1 INTRODUCTION TO NUMERICAL ALGORITHMS

Numerical methods for solving differential equations can be applied to both linear and nonlinear models, although their primary application is to nonlinear models. A nonlinear ordinary differential equation can be recognized by the fact that the dependent variable or its derivatives appears raised to a power or in a transcendental function. For example, the following equations are nonlinear.

$$y\ddot{y} + 5\dot{y} + y = 0 \qquad \dot{y} + \sin\, y = 0 \qquad \dot{y} + \sqrt{y} = 0$$

The algorithms presented here are simplified versions of the ones used by MATLAB and Simulink, and so an understanding of these methods will improve your understanding of these two programs. The numerical solution algorithms used by the MATLAB ODE solvers are very complicated. Therefore, we will limit our coverage to simple algorithms so that we can highlight the important issues to be considered when using numerical methods.

Numerical methods require that the derivatives in the model be described by finite-difference expressions, and that the resulting difference equations be solved in a step-by-step procedure. The issues related to these methods are

- What finite-difference expressions provide the best approximations for derivatives?
- What are the effects of step size used to obtain the approximations?
- What are the effects of roundoff error when solving the finite-difference equations?

We will explore these issues and make you aware of some pitfalls that can be encountered. Such difficulties are more likely to occur when the solution is rapidly changing with time, and can happen if the step size is not small compared to the smallest time constant of the system or the smallest oscillation period.

E.1.1 TEST CASES

We now develop three test cases, whose solution can be found analytically, to use for checking the results of our numerical methods.

1. The following equation is used to illustrate the effect of step size relative to the system time constant, which is $\tau = 1/10$.

$$\frac{dy}{dt} + 10y = 0 \qquad y(0) = 2$$

   The solution is $y(t) = 2e^{-10t}$.

2. The following equation is used to illustrate the effect of step size relative to the solution's oscillation period.

$$\frac{dy}{dt} = \sin \omega t \qquad y(0) = y_0$$

   The solution is

$$y(t) = y_0 + \frac{1 - \cos \omega t}{\omega}$$

   and the period is $2\pi/\omega$.

3. A *stiff* differential equation is one whose response changes rapidly over a time scale that is short compared to the time scale over which we are interested in the solution. For this reason, stiff equations present a challenge to solve numerically. The following equation has such characteristics.

$$\dot{y} + y = 0.001e^{10t} \qquad y(0) = 10$$

   The solution is

$$y(t) = 10e^{-t} + \frac{0.001}{11}\left(e^{10t} - e^{-t}\right)$$

   That part of the response due to the term e^{-t} is approximately 10 at $t = 0$ and decays quickly (it is approximately 0.2 at $t = 4$). However, the term due to e^{10t} is 9.09×10^{-5} at $t = 0$ but grows at a fast rate (it is 2.14×10^{13} at $t = 4$!). Thus it would be difficult for a plot to show the solution accurately over the range $0 \le t \le 4$. More importantly, a numerical solver would need a very small step size to compute the rapid changes due to the e^{10t} term, much smaller than the step size required to compute the slower response due to the e^{-t} term. The result can be a large accumulated error because of the small step size combined with the large number of steps required to obtain the full solution.

E.1.2 THE EULER METHOD

The essence of a numerical method is to convert the differential equation into a difference equation that can be programmed on a calculator or digital computer. Numerical algorithms differ partly as a result of the specific procedure used to obtain the difference equations. In general, as the accuracy of the approximation is increased, so is the complexity of the programming involved. It is important to understand the concept of "step size" and its effects on solution accuracy. To provide a simple introduction to these issues, we begin with the simplest numerical method, the *Euler method*.

Consider the equation

$$\frac{dy}{dt} = f(t, y) \tag{E.1.1}$$

where $f(t, y)$ is a known function. From the definition of the derivative,

$$\frac{dy}{dt} = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

If the time increment Δt is chosen small enough, the derivative can be replaced by the approximate expression

$$\frac{dy}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t} \tag{E.1.2}$$

Assume that the right-hand side of (E.1.1) remains constant over the time interval $(t, t + \Delta t)$, and replace (E.1.1) by the following approximation:

$$\frac{y(t + \Delta t) - y(t)}{\Delta t} = f(t, y)$$

or

$$y(t + \Delta t) = y(t) + f(t, y)\Delta t \tag{E.1.3}$$

The smaller Δt is, the more accurate are our two assumptions leading to (E.1.3). This technique for replacing a differential equation with a difference equation is the *Euler method*. The increment Δt is called the *step size*.

A more concise representation is obtained by using the following notation:

$$y_k = y(t_k) \qquad y_{k+1} = y(t_{k+1}) = y(t_k + \Delta t)$$

where $t_{k+1} = t_k + \Delta t$. In this notation, the Euler algorithm (E.1.3) is expressed as

$$y_{k+1} = y_k + f(t_k, y_k)\Delta t \tag{E.1.4}$$

The Euler Method and a Decaying Solution — EXAMPLE E.1.1

■ Problem

Use the Euler method to solve our first test case,

$$\frac{dy}{dt} + 10y = 0 \qquad y(0) = 2$$

which has the exact solution $y(t) = 2e^{-10t}$. Use a step size of $\Delta t = 0.02$, which is one-fifth of the time constant.

■ Solution

Here $f(t, y) = -10y$. Thus the Euler algorithm (E.1.4) in this case becomes

$$y_{k+1} = y_k - (10y_k)\Delta t$$

or

$$y_{k+1} = y_k - (10y_k)0.02 = 0.8y_k$$

We show the computations for the first few steps, using four significant figures. The exact value, obtained from $y(t) = 2e^{-10t}$, and the percent error are shown in the following table.

Step	Numerical solution	Exact solution	Percent error
$k = 0$	$y_1 = 0.8y_0 = 1.6$	1.637	2.3%
$k = 1$	$y_2 = 0.8y_1 = 1.28$	1.341	4.5%
$k = 2$	$y_3 = 0.8y_2 = 1.024$	1.098	6.7%
$k = 3$	$y_4 = 0.8y_3 = 0.8192$	0.8987	8.8%
$k = 4$	$y_5 = 0.8y_4 = 0.6554$	0.7358	10.9%
$k = 5$	$y_6 = 0.8y_5 = 0.5243$	0.6024	13%

Notice how the percent error grows with each step. This is because the calculated result from the previous step is not exact. The numerical and exact solutions are shown in Figure E.1.1, where the numerical solution is shown by the small circles and the exact solution is shown by the solid curve.

Another observation here is that the step size should be much smaller than the time constant τ. A commonly used rule of thumb is that $\Delta t \leq \tau/20$.

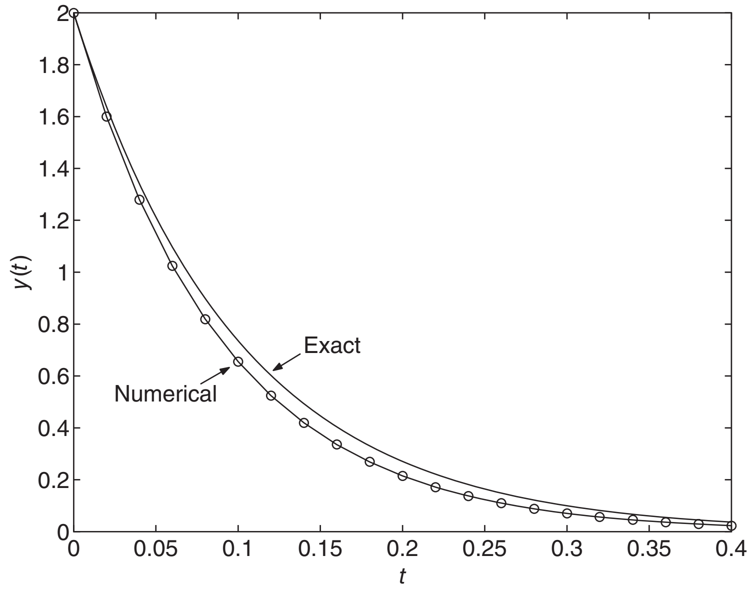

Figure E.1.1 Euler and exact solutions of $\dot{y} + 10y = 0$, $y(0) = 2$.

Roundoff Error There is another reason why the error increases with the number of steps. If we had retained six significant figures instead of four, we would have obtained $y_5 = 0.65536$ and $y_6 = 0.524288$. Even though computers can retain many more than four significant figures, nevertheless, they cannot represent numbers with infinite accuracy. Thus, the calculated solution obtained by computer at each step is in effect rounded off to a finite number of significant figures. This rounded number is then used in the calculations for the next step, and so on, just as we rounded the value of y_5 to 0.6554 before using it to compute y_6. Therefore, the error in the numerical solution will increase with the number of steps required to obtain the solution.

Thus, because roundoff error increases with each step, there is a trade-off between using a step size small enough to obtain an accurate solution yet not so small that many steps are required to obtain the solution.

Numerical methods have their greatest errors when trying to obtain solutions that are rapidly changing. The difficulties caused by an oscillating solution are illustrated in the following example.

The Euler Method and an Oscillating Solution

EXAMPLE E.1.2

■ Problem

Consider the following equation, which is our second test case.

$$\dot{y} = \sin t$$

for $y(0) = 0$ and $0 \le t \le 4\pi$. The exact solution is $y(t) = 1 - \cos t$ and its period is 2π. Solve this equation with Euler's method.

■ Solution

We choose a step size equal to one-thirteenth of the period, or $\Delta t = 2\pi/13$, so that we can compare the answer with that obtained by a method to be introduced later. The Euler algorithm (E.1.4) becomes

$$y_{k+1} = y_k + (\sin t_k)\frac{2\pi}{13}$$

For successive values of $k = 0, 1, 2, \ldots$, we have $t_k = 0, 2\pi/13, 4\pi/13, \ldots$. Retaining four significant figures, we have

$$y_1 = y_0 + (\sin t_0)\frac{2\pi}{13} = 0 + (\sin 0)\,\frac{2\pi}{13} = 0$$

$$y_2 = y_1 + (\sin t_1)\frac{2\pi}{13} = 0 + \left(\sin\frac{2\pi}{13}\right)\frac{2\pi}{13} = 0.2246$$

$$y_3 = y_2 + (\sin t_2)\frac{2\pi}{13} = 0.2246 + \left(\sin\frac{4\pi}{13}\right)\frac{2\pi}{13} = 0.6224$$

and so on. The numerical and exact solutions are shown in Figure E.1.2, where the numerical solution is shown by the small circles and the exact solution is shown by the solid curve. There is noticeable error, especially near the peaks and valleys, where the solution is rapidly changing.

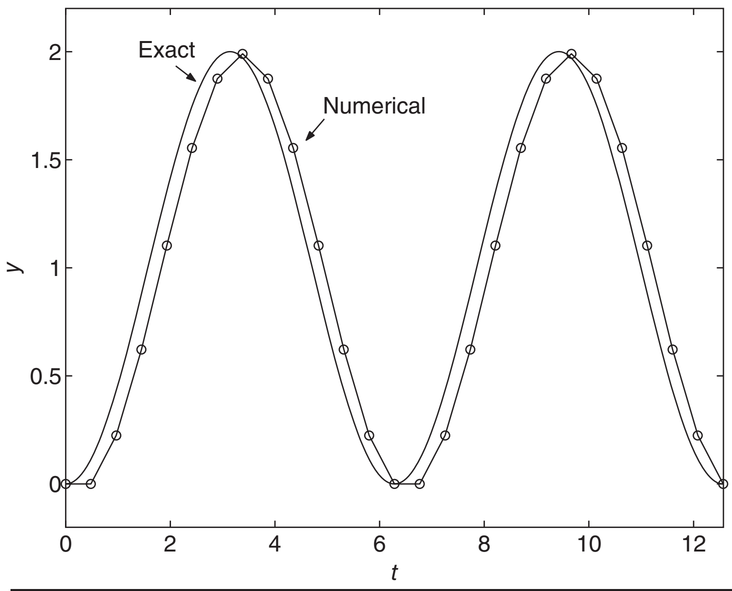

Figure E.1.2 Euler and exact solutions of $\dot{y} = \sin t$, $y(0) = 0$.

The accuracy of the Euler method can be improved by using a smaller step size. However, very small step sizes require longer run times and can result in a large accumulated error due to roundoff effects. So we seek better algorithms to use for more challenging applications.

E.1.3 PREDICTOR-CORRECTOR METHODS

We now consider *predictor-corrector* methods, which serve as the basis for many powerful algorithms. The Euler method can have a serious deficiency in problems where the variables are rapidly changing, because the method assumes the variables are constant over the time interval Δt. One way of improving the method is to use a better approximation to the right-hand side of the equation

$$\frac{dy}{dt} = f(t, y) \tag{E.1.5}$$

The Euler approximation is

$$y(t_{k+1}) = y(t_k) + \Delta t\, f[t_k, y(t_k)] \tag{E.1.6}$$

Suppose instead we use the average of the right-hand side of (E.1.5) on the interval (t_k, t_{k+1}). This gives

$$y(t_{k+1}) = y(t_k) + \frac{\Delta t}{2}(f_k + f_{k+1}) \tag{E.1.7}$$

where

$$f_k = f[t_k, y(t_k)] \tag{E.1.8}$$

with a similar definition for f_{k+1}. Equation (E.1.7) is equivalent to integrating (E.1.5) with the trapezoidal rule, whereas the Euler method is equivalent to integrating with the rectangular rule (see Figure E.1.3).

The difficulty with (E.1.7) is that f_{k+1} cannot be evaluated until $y(t_{k+1})$ is known, but this is precisely the quantity being sought. A way out of this difficulty is to use the Euler formula (E.1.6) to obtain a preliminary estimate of $y(t_{k+1})$. This estimate is then used to compute f_{k+1} for the use in (E.1.7) to obtain the required value of $y(t_{k+1})$.

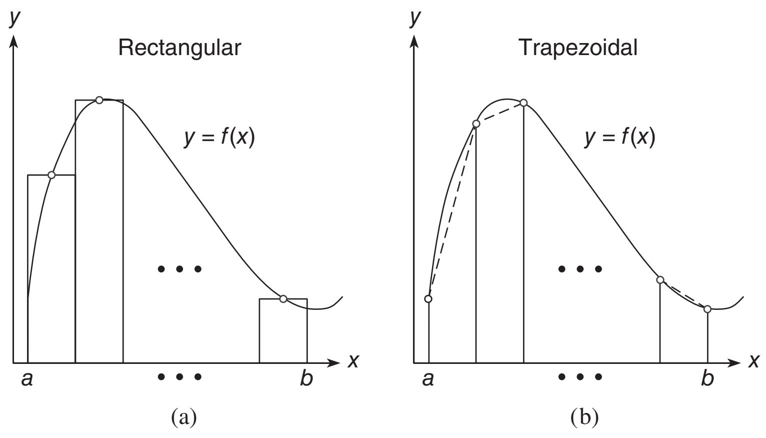

Figure E.1.3 Illustration of numerical integration by (a) the rectangular rule and (b) the trapezoidal rule.

The notation can be changed to clarify the method. Let $h = \Delta t$ and $y_k = y(t_k)$, and let x_{k+1} be the estimate of $y(t_{k+1})$ obtained from the Euler formula (E.1.6). Then, by omitting the t_k notation from the other equations, we obtain the following description of the predictor-corrector process:

$$\text{Euler predictor:} \quad x_{k+1} = y_k + h\, f(t_k, y_k) \tag{E.1.9}$$

$$\text{Trapezoidal corrector:} \quad y_{k+1} = y_k + \frac{h}{2}\left[f(t_k, y_k) + f(t_{k+1}, x_{k+1})\right] \tag{E.1.10}$$

This version of a predictor-corrector algorithm is sometimes called the *modified-Euler method*. However, note that any algorithm can be tried as a predictor or a corrector. Thus many other methods can also be classified as predictor-corrector.

The Modified-Euler Method and a Decaying Solution — EXAMPLE E.1.3

■ Problem

Use the modified-Euler method to solve our first test case:

$$\dot{y} = -10y \qquad y(0) = 2$$

for $0 \le t \le 0.5$. The exact solution is $y(t) = 2e^{-10t}$.

■ Solution

To illustrate the effect of the step size on the solution's accuracy, we use a step size $h = 0.02$, the same size used with the Euler method. The modified-Euler algorithm for this case has the following form.

$$x_{k+1} = y_k + h(-10y_k) = (1 - 10h)y_k = 0.8y_k$$

$$y_{k+1} = y_k + \frac{h}{2}(-10y_k - 10x_{k+1}) = 0.9y_k - 0.1x_{k+1}$$

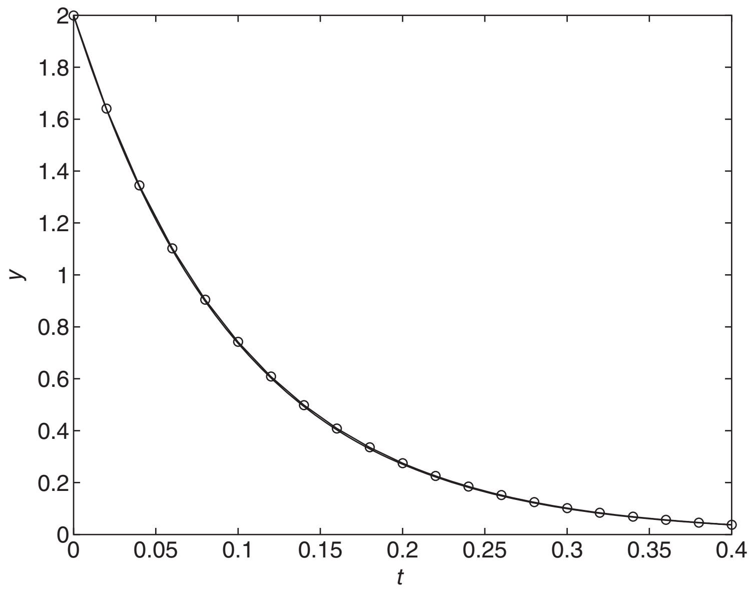

Figure E.1.4 Modified-Euler and exact solutions of $\dot{y} + 10y = 0$, $y(0) = 2$.

The following table shows the numerical and exact solutions, rounded to four significant figures, and the percent error, for a few steps.

Step	Numerical solution	Exact solution	Percent error
$k = 0$	$x_1 = 0.8y_0 = 1.6$		
	$y_1 = 0.9y_0 - 0.1x_1 = 1.64$	1.637	0.2%
$k = 1$	$x_2 = 0.8y_1 = 1.312$		
	$y_2 = 0.9y_1 - 0.1x_2 = 1.3448$	1.341	0.3%
$k = 2$	$x_3 = 0.8y_2 = 1.07584$		
	$y_3 = 0.9y_2 - 0.1x_3 = 1.102736$	1.098	0.4%

There is less error than with the Euler method using the same step size. Figure E.1.4 shows the results, with the numerical solution shown by the small circles and the exact solution by the solid line.

The modified-Euler method is a special case of the Runge-Kutta family of algorithms, to be discussed next. For purposes of comparison with the Runge-Kutta methods, we can express the modified-Euler method as follows:

$$y_{k+1} = y_k + \frac{1}{2}(g_1 + g_2) \tag{E.1.11}$$

$$g_1 = hf(t_k, y_k) \tag{E.1.12}$$

$$g_2 = hf(t_k + h, y_k + g_1) \tag{E.1.13}$$

E.1.4 RUNGE-KUTTA METHODS

The Taylor series representation forms the basis of several methods for solving differential equations, including the Runge-Kutta methods. The Taylor series may be used to represent the solution $y(t + h)$ in terms of $y(t)$ and its derivatives, as follows:

$$y(t + h) = y(t) + h\,\dot{y}(t) + \frac{1}{2}h^2\ddot{y}(t) + \cdots \tag{E.1.14}$$

The number of terms kept in the series determines its accuracy. The required derivatives are calculated from the differential equation. If these derivatives can be found, (E.1.14) can be used to march forward in time. In practice, the high-order derivatives can be difficult to calculate, and the series (E.1.14) is truncated at some term. The Runge-Kutta methods were developed because of the difficulty in computing the derivatives. These methods use several evaluations of the function $f(t, y)$ in a way that approximates the Taylor series. The number of terms in the series that is duplicated determines the order of the Runge-Kutta method. Thus, a fourth-order Runge-Kutta algorithm duplicates the Taylor series through the term involving h^4.

E.1.5 SECOND-ORDER RUNGE-KUTTA METHODS

The second-order Runge-Kutta methods express y_{k+1} as

$$y_{k+1} = y_k + w_1 g_1 + w_2 g_2 \tag{E.1.15}$$

where w_1 and w_2 are constant weighting factors, and

$$g_1 = hf(t_k, y_k) \tag{E.1.16}$$

$$g_2 = hf(t_k + \alpha h, y_k + \beta h f_k) \tag{E.1.17}$$

The family of second-order Runge-Kutta algorithms is categorized by the parameters α, β, w_1, and w_2. To duplicate the Taylor series through the h^2 term, these coefficients must satisfy the following:

$$w_1 + w_2 = 1 \qquad w_1\alpha = \frac{1}{2} \qquad w_2\beta = \frac{1}{2} \tag{E.1.18}$$

Thus one of the parameters can be chosen independently.

The modified-Euler algorithm, (E.1.11) through (E.1.13), is thus seen to be a second-order Runge-Kutta algorithm with $\alpha = \beta = 1$ and $w_1 = w_2 = 1/2$.

E.1.6 FOURTH-ORDER RUNGE-KUTTA METHODS

The family of fourth-order Runge-Kutta algorithms expresses y_{k+1} as

$$y_{k+1} = y_k + w_1 g_1 + w_2 g_2 + w_3 g_3 + w_4 g_4 \tag{E.1.19}$$

where
$$\begin{aligned} g_1 &= hf(t_k, y_k) \\ g_2 &= hf(t_k + \alpha_1 h, y_k + \alpha_1 g_1) \\ g_3 &= hf[t_k + \alpha_2 h, y_k + \beta_2 g_2 + (\alpha_2 - \beta_2)g_1] \\ g_4 &= hf[t_k + \alpha_3 h, y_k + \beta_3 g_2 + \gamma_3 g_3 + (\alpha_3 - \beta_3 - \gamma_3)g_1] \end{aligned} \tag{E.1.20}$$

Comparison with the Taylor series yields eight equations for the 10 parameters. Thus, two parameters can be chosen in light of other considerations. A number of different choices have been used. For example, the *classical* Runge-Kutta method, which reduces to Simpson's rule for integration if $f(t, y)$ is a function of only t, uses the following set of parameters.

$$\begin{aligned} w_1 = w_4 &= \tfrac{1}{6} & w_2 = w_3 &= \tfrac{1}{3} \\ \alpha_1 = \alpha_2 &= \tfrac{1}{2} & \beta_2 &= \tfrac{1}{2} \\ \gamma_3 = \alpha_3 &= 1 & \beta_3 &= 0 \end{aligned} \tag{E.1.21}$$

The Runge-Kutta algorithms are very tedious to compute by hand, so we not show the steps involved in Examples E.1.4 and E.1.5. The algorithms are easily programmed, however. The MATLAB programs for these examples, using the parameter values for the classical fourth-order Runge-Kutta algorithm, are given in Section E.2.

Runge-Kutta Method for an Oscillating Solution — EXAMPLE E.1.4

■ Problem

Illustrate how the fourth-order Runge-Kutta method works with an oscillating solution by using the method to solve our second test case:

$$\dot{y} = \sin t \qquad y(0) = 0$$

for $0 \le t \le 4\pi$. Use the parameter values given by (E.1.21).

■ Solution

To compare the results with those obtained with the Euler method, we will use the same step size $\Delta t = 2\pi/13$. The results are shown in Figure E.1.5, with the numerical solution shown by the small circles and the exact solution by the solid line. There is much less error than with the Euler method using the same step size.

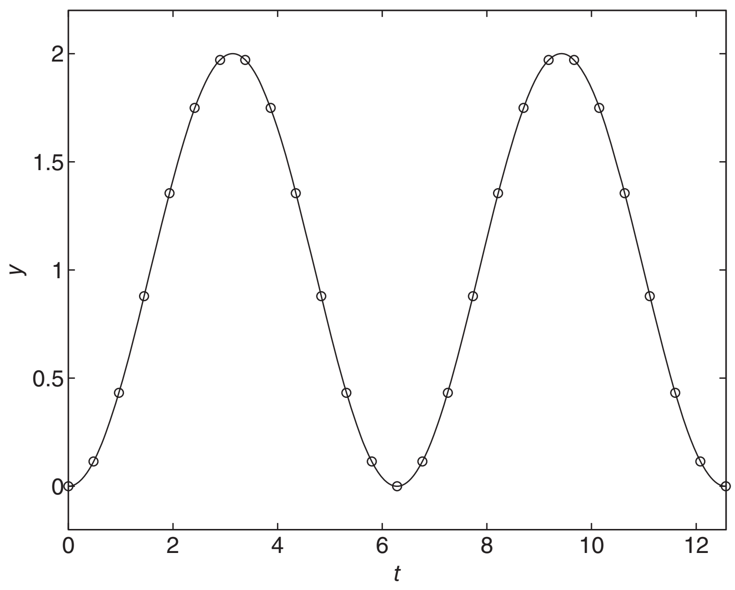

Figure E.1.5 Fourth-order Runge-Kutta and exact solutions of $\dot{y} = \sin t$, $y(0) = 0$.

EXAMPLE E.1.5

Runge-Kutta Method for a Stiff Equation

■ Problem

Our third test case is an example of an equation that requires a step size small enough to solve for the rapid changes in the solution, but for which many steps are needed to obtain the solution over the longer time interval. Thus an accurate numerical algorithm is needed to prevent large errors from accumulating. The problem is

$$\dot{y} + y = 0.001e^{10t} \qquad y(0) = 10$$

The closed form solution is

$$y(t) = 10e^{-t} + \frac{0.001}{11}(e^{10t} - e^{-t})$$

Compare this solution with that obtained with the fourth-order Runge-Kutta using the parameter values given by (E.1.21).

■ Solution

The following table gives the results for every twentieth step, along with the exact solution, to 11 decimal places. The numerical solution is correct to seven or more significant figures, which is quite good considering the wide range of values of y.

t	Exact solution	Numerical solution
0.2	8.18790483308	8.18790483325
0.4	6.70810299036	6.70810299083
0.6	5.52474181384	5.52474181600
0.8	44.76424497372	44.76424498796
1.0	5.68116694947	5.68116705275
1.2	17.80780486683	17.80780562797
1.4	111.79360941831	111.79361504050
1.6	809.84717596280	809.84721750362
1.8	5970.74107724873	5970.74138419445

Programming these algorithms is a very good way to improve your understanding of them. Examples of MATLAB programs that generated the data and the plots in this section are given in Section E.2.

E.2 PROGRAMMING NUMERICAL METHODS IN MATLAB

In this section, we first show how to program the Euler, modified-Euler, and Runge-Kutta methods in MATLAB. Although MATLAB has built-in solvers, learning to program such algorithms will improve your understanding of these methods.

E.2.1 PROGRAMMING THE EULER METHOD

The Euler algorithm for the equation $\dot{y} = f(t, y)$ is

$$y_{k+1} = y_k + f(t_k, y_k)\Delta t$$

where $t_{k+1} = t_k + \Delta t$.

This equation can be applied successively at the times t_k, for example, by putting it in a `for` loop in a MATLAB program.

The Euler Method for $\dot{y} = -10y$ **EXAMPLE E.2.1**

■ Problem

Use the Euler method to solve our first test case:

$$\dot{y} = -10y \qquad y(0) = 2$$

■ Solution

The following script file solves the problem and plots the solution over the range $0 \le t \le 0.5$. The exact solution is $y(t) = 2e^{-10t}$. For comparison purposes, we use the same step size as in Example E.1.1, which is $\Delta t = 0.02$.

```
delta = 0.02; y(1) = 2;
k = 0;
for time = [delta:delta:0.5]
   k = k + 1;
   y(k+1) = y(k) - 10*y(k)*delta;
end
t = [0:delta: 0.5];
y_exact = 2*exp(-10*t);
plot(t,y,'o',t,y_exact),xlabel('t'),ylabel('y')
```

Figure E.1.1 in Section E.1 shows the results. The numerical solution is shown by the small circles. The exact solution is shown by the solid line. There is some noticeable error. If we had used a step size equal to 0.005, for example, the error would not be noticeable on the plot.

E.2.2 PROGRAMMING THE MODIFIED-EULER METHOD

The modified-Euler algorithm for the equation $\dot{y} = f(t, y)$ is

$$\text{Euler predictor:} \quad x_{k+1} = y_k + hf(t_k, y_k)$$

$$\text{Trapezoidal corrector:} \quad y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, x_{k+1})]$$

EXAMPLE E.2.2 The Modified-Euler Method for $\dot{y} = -10y$

■ **Problem**

Use the modified-Euler method to solve our first test case:

$$\dot{y} = -10y \qquad y(0) = 2$$

■ **Solution**

The following script file solves the problem and plots the solution over the range $0 \le t \le 0.5$. The exact solution is $y(t) = 2e^{-10t}$. We use a step size $\Delta t = 0.02$ to compare the results with those of the Euler method.

```
delta = 0.02; y(1) = 2;
k = 0;
for time = [delta:delta: 0.5]
   k = k + 1;
   x(k+1) = y(k) - 10*delta*y(k);
   y(k+1) = y(k) - 10*(delta/2)*(y(k) + x(k+1));
end
t = [0:delta:0.5];
y_exact = 2*exp(-10*t);
plot(t,y,'o',t,y_exact),xlabel('t'),ylabel('y')
```

Figure E.1.4 in Section E.1 shows the results, with the numerical solution shown by the small circles and the exact solution by the solid line. There is less error than with the Euler method using the same step size.

E.2.3 PROGRAMMING THE RUNGE-KUTTA METHOD

The fourth-order Runge-Kutta algorithm is

$$y_{k+1} = y_k + w_1 g_1 + w_2 g_2 + w_3 g_3 + w_4 g_4$$

where

$$\begin{aligned} g_1 &= hf(t_k, y_k) \\ g_2 &= hf(t_k + \alpha_1 h, y_k + \alpha_1 g_1) \\ g_3 &= hf[t_k + \alpha_2 h, y_k + \beta_2 g_2 + (\alpha_2 - \beta_2) g_1] \\ g_4 &= hf[t_k + \alpha_3 h, y_k + \beta_3 g_2 + \gamma_3 g_3 + (\alpha_3 - \beta_3 - \gamma_3) g_1] \end{aligned}$$

We will use the parameters for the classical Runge-Kutta method, which are

$$\begin{aligned} w_1 = w_4 = \tfrac{1}{6} &\qquad w_2 = w_3 = \tfrac{1}{3} \\ \alpha_1 = \alpha_2 = \tfrac{1}{2} &\qquad \beta_2 = \tfrac{1}{2} \\ \gamma_3 = \alpha_3 = 1 &\qquad \beta_3 = 0 \end{aligned}$$

The Runge-Kutta Method for an Oscillating Solution

EXAMPLE E.2.3

■ Problem

Use the fourth-order Runge-Kutta method to solve our second test case:

$$\dot{y} = \sin t \qquad y(0) = 0$$

$0 \le t \le 4\pi$. The exact solution is $y(t) = 1 - \cos t$.

■ Solution

Here $f(t, y) = \sin t$ and thus is not a function of y. Therefore, the β and γ parameters are not needed for this problem and the g_i functions are

$$g_1 = h\sin(t_k)$$
$$g_2 = h\sin(t_k + \alpha_1 h)$$
$$g_3 = h\sin(t_k + \alpha_2 h)$$
$$g_4 = h\sin(t_k + \alpha_3 h)$$

We use a step size equal to one-thirteenth of the period, or $\Delta t = 2\pi/13$, so that we can compare the results with those obtained from the modified-Euler method. The following script file implements the method:

```
h = 2*pi/13;
w1 = 1/6; w2 = 1/3; w3 = 1/3; w4 = 1/6;
a1 = 1/2; a2 = 1/2; a3 = 1;
tk = 0;
% Set the initial condition.
y(1) = 0;
% Set the upper limit on the number of steps.
tf = 4*pi;
klimit = round(tf/h)-1;
for k = 0:klimit
   tk = k*h;
   g1 = h*sin(tk);
   g2 = h*sin(tk+a1*h);
   g3 = h*sin(tk+a2*h);
   g4 = h*sin(tk+a3*h);
   m = k+1;
   y(m+1) = y(m)+w1*g1+w2*g2+w3*g3+w4*g4;
end
t = [0:h:(klimit+1)*h];
% Compute the exact solution.
te = [0:0.01:tf];
ye = 1-cos(te);
plot(te,ye,t,y,'o'),xlabel('t'),ylabel('y'),axis([0 tf 0 2])
```

Figure E.1.5 in Section E.1 shows the results, with the numerical solution shown by the small circles and the exact solution by the solid line. There is less error than with the modified-Euler method using the same step size.

EXAMPLE E.2.4 The Runge-Kutta Method for a Stiff Equation

■ Problem

Use the fourth-order Runge-Kutta method to solve our third test case:

$$\dot{y} + y = 0.001e^{10t} \qquad y(0) = 10$$

for $0 \le t \le 1.8$. The exact solution is

$$y(t) = 10e^{-t} + \frac{0.001}{11}\left(e^{10t} - e^{-t}\right)$$

■ Solution

Here $f(t, y) = -y + 0.001e^{10t}$ and it is helpful to write a user-defined function file for $f(t, y)$. This file is

```
function ydot = stiff(t,y)
ydot = 0.001*exp(10*t)-y;
```

We try a step size of $h = 0.01$. The script file is

```
h = 0.01; tk = 0;
w1 = 1/6; w2 = 1/3; w3 = 1/3; w4 = 1/6;
a1 = 1/2; a2 = 1/2; a3 = 1;
b2 = 1/2; b3 = 0; gam3 = 1;
% Set the initial condition.
y(1) = 10;
% Set the upper limit on the number of steps.
tf = 1.8;
klimit = round(tf/h)-1;
for k = 0:klimit
   tk = k*h;
   yk = y(k+1);
   g1 = h*stiff(tk,yk);
   g2 = h*stiff(tk+a1*h,yk+a1*g1);
   g3 = h*stiff(tk+a2*h,yk+b2*g2+(a2-b2)*g1);
   g4 = h*stiff(tk+a3*h,yk+b3*g2+gam3*g3+(a3-b3-gam3)*g1);
   m = k+1;
   y(m+1) = y(m)+w1*g1+w2*g2+w3*g3+w4*g4;
end
% Compute the exact solution.
te = [0:0.01:tf];
ye = 10*exp(-te)+(0.001/11)*(exp(10*te)-exp(-te));
% Plot every 20th point of the numerical solution.
kp = 0;
for k = 1:20:klimit+1
   kp = kp+1;
   yp(kp) = y(k);
end
% Compute times for numerical solution, every 20th point.
tp = [0:20*h:tf];
plot(te,ye,tp,yp,'o'),xlabel('t'),ylabel('y')
```

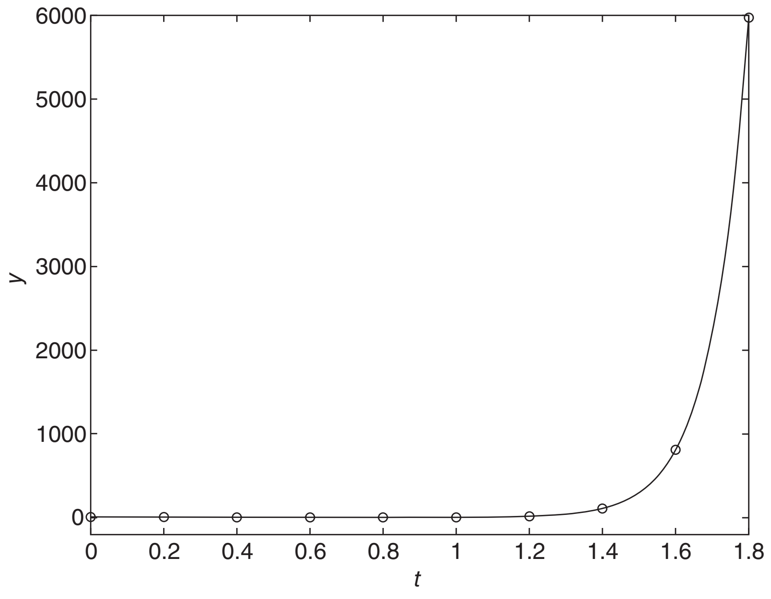

Figure E.2.1 Fourth-order Runge-Kutta and exact solutions of $\dot{y} + y = 0.001e^{10t}$, $y(0) = 10$.

Figure E.2.1 shows the results, with the numerical solution shown for every 20 steps by the small circles and the exact solution by the solid line. Notice the wide range of values for y, and yet the results are very accurate.

PROBLEMS

Section E.1 Introduction to Numerical Algorithms

E.1 a. Use the Euler method with a step size of $\Delta t = 0.02$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} + 5y = 0 \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.2 a. Use the Euler method with a step size of $\Delta t = 0.3$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = \cos t \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.3 a. Use the Euler method with a step size of $\Delta t = 0.1$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = 6 \sin 3t \quad y(0) = 10$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.4 a. Use the Euler method with a step size of $\Delta t = 0.025$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = 5e^{-4t} \quad y(0) = 2$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.5 a. Use the Euler method with a step size of $\Delta t = 0.01$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} + 3y = 5e^{4t} \quad y(0) = 10$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.6 a. Use the modified-Euler method with a step size of $\Delta t = 0.02$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} + 5y = 0 \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.7 a. Use the modified-Euler method with a step size of $\Delta t = 0.03$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = \cos t \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.8 a. Use the modified-Euler method with a step size of $\Delta t = 0.1$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = 6 \sin 3t \quad y(0) = 10$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.9 a. Use the modified-Euler method with a step size of $\Delta t = 0.025$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} = 5e^{-4t} \quad y(0) = 2$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.10 a. Use the modified-Euler method with a step size of $\Delta t = 0.01$ to solve the following equation for five steps, using four significant figures.

$$\dot{y} + 3y = 5e^{4t} \quad y(0) = 10$$

b. Use the closed-form solution to check the accuracy of the numerical method.

Section E.2 Programming Numerical Methods in MATLAB

E.11 Consider the system for lifting a mast, discussed in Chapter 3. The 70-ft-long mast weighs 500 lb. The winch applies a force $f = 380$ lb to the cable. The mast is supported initially at an angle of 30°, and the cable at A is initially horizontal. The equation of motion of the mast is

$$25{,}400\,\ddot{\theta} = -17{,}500 \cos\theta + \frac{626{,}000}{Q}\sin(1.33 + \theta)$$

where

$$Q = \sqrt{2020 + 1650 \cos(1.33 + \theta)}$$

Use a numerical method to solve for and plot $\theta(t)$ for $\theta(t) \le \pi/2$ rad.

E.12 a. Program the Euler method and use the program to solve the following equation for $0 \leq t \leq 1$. Use a step size of $\Delta t = 0.02$.

$$\dot{y} + 5y = 0 \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.13 a. Program the Euler method and use the program to solve the following equation for $0 \leq t \leq 12$. Use a step size of $\Delta t = 0.3$.

$$\dot{y} = \cos t \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.14 a. Program the Euler method and use the program to solve the following equation for $0 \leq t \leq 1$. Use a step size of $\Delta t = 0.025$.

$$\dot{y} = 5e^{-4t} \quad y(0) = 2$$

b. Use the closed-form solution check the accuracy of the numerical method.

E.15 a. Program the modified-Euler method and use the program to solve the following equation for $0 \leq t \leq 1$. Use a step size of $\Delta t = 0.02$.

$$\dot{y} + 5y = 0 \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.16 a. Program the modified-Euler method and use the program to solve the following equation for $0 \leq t \leq 12$. Use a step size of $\Delta t = 0.3$.

$$\dot{y} = \cos t \quad y(0) = 6$$

b. Use the closed-form solution to check the accuracy of the numerical method.

E.17 a. Program the modified-Euler method and use the program to solve the following equation for $0 \leq t \leq 1$. Use a step size of $\Delta t = 0.025$.

$$\dot{y} = 5e^{-4t} \quad y(0) = 2$$

b. Use the closed-form solution check the accuracy of the numerical method.

E.18 a. Program the fourth-order Runge-Kutta method and use the program to solve the following equation for $0 \leq t \leq 12$. Use a step size of $\Delta t = 0.3$.

$$\dot{y} = \cos t \quad y(0) = 6$$

b. Use the closed-form solution check the accuracy of the numerical method.

E.19 a. Program the fourth-order Runge-Kutta method and use the program to solve the following equation for $0 \leq t \leq 1$. Use a step size of $\Delta t = 0.01$.

$$\dot{y} + 3y = 5e^{4t} \quad y(0) = 10$$

b. Use the closed-form solution to check the accuracy of the numerical method.