A second-order ODE of the form
\[ y'' = g(x, y, y') \]
can be converted to a system of two first-order ODEs by a simple change of variables:
\[ u = y, \]
\[ v = y'. \]

The differential equations relating these variables (functions) are
\[ u' = v = f(x, u, v), \]
\[ v' = g(x, u, v). \]

The initial conditions for the original ODE,
\[ y(0) = \alpha_0, \quad y'(0) = \alpha_1, \]
become the initial conditions for the system, i.e.,
\[ u(0) = \alpha_0, \quad v(0) = \alpha_1. \]

### Example 13.1 Nonlinear Pendulum

Consider the nonlinear pendulum described in Example 13-A, with angular displacement \( y(x) \) given by
\[ y'' + \frac{c}{mL} y' + \frac{g}{L} \sin y = 0, \quad y(0) = a, \quad y'(0) = b. \]

Choosing \( g/L = 1 \) and \( c/(mL) = 0.3 \), \( a = \pi/2 \), and \( b = 0 \), we get the second-order ODE–IVP
\[ y'' = -0.3 y' - \sin y, \]
which can be converted to a system of first-order ODEs by means of the change of variables
\[ u = y, \]
\[ v = y'. \]

The differential equations relating these variables are
\[ u' = v = f(x, u, v), \]
\[ v' = -0.3 v - \sin u = g(x, u, v), \]
with initial conditions \( u(0) = \pi/2, v(0) = 0. \)

We investigate the application of Euler's method and the midpoint method to this and other systems of two first-order ODEs in the next section.
A higher order ODE may be converted to a system of first-order ODEs by a similar change of variables. The nth-order ODE

\[ y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}), \]

\[ y(0) = \alpha_0, \quad y'(0) = \alpha_1, \quad y''(0) = \alpha_2, \quad \ldots, \quad y^{(n-1)}(0) = \alpha_{n-1}, \]

becomes a system of first-order ODEs by the following change of variables:

\[ u_1 = y, \]
\[ u_2 = y', \]
\[ u_3 = y'', \]
\[ \vdots \]
\[ u_n = y^{(n-1)}. \]

The differential equations relating these variables are

\[ u_1' = u_2, \]
\[ u_2' = u_3, \]
\[ u_3' = u_4, \]
\[ \vdots \]
\[ u_n' = f(x, u_1, u_2, u_3, \ldots, u_n), \]

with the initial conditions

\[ u_1(0) = \alpha_0, \quad u_2(0) = \alpha_1, \quad u_3(0) = \alpha_2, \quad \ldots, \quad u_n(0) = \alpha_{n-1}. \]

We investigate the application of several of the methods from Chapter 12 to general systems of ODEs in Section 13.3. By utilizing MATLAB’s vector capabilities, only minor changes are required to the functions presented in Chapter 12.

### 13.2 SYSTEMS OF TWO FIRST-ORDER ODEs

Any of the methods for solving ODE–IVPs discussed in Chapter 12 can be generalized to apply to systems of equations. In this section, we consider systems of two first-order ODEs in detail, using the Euler and Runge–Kutta methods. In Section 13.3, we treat systems of arbitrary size, using MATLAB’s vector capabilities extensively.

#### 13.2.1 Euler’s Method for Solving Two ODE–IVPs

To apply the basic Euler’s method

\[ y_{i+1} = y_i + hf(x_i, y_i) \]

to the system of ODEs
we update the function $u$ using $f(x, u, \nu)$ and update $\nu$ using $g(x, u, \nu)$. The same step size $h$ is used for each function (since that refers to the spacing of the independent variable $x$):

$$u(i + 1) = u(i) + h f(x(i), u(i), \nu(i)),$$

$$\nu(i + 1) = \nu(i) + h g(x(i), u(i), \nu(i)).$$
13.2.2 Midpoint Method for Solving Two ODE-IVPs

The idea in generalizing Runge–Kutta methods to systems of two equations is the same as for Euler's method; that is, we update each unknown function \( u \) and \( v \), using the basic Runge–Kutta formulas and the appropriate right-hand-side function, \( f \) or \( g \), from the differential equation for the unknown:

\[
\begin{align*}
    u' & = f(x, u, v), \\
    v' & = g(x, u, v).
\end{align*}
\]

We rewrite the basic second-order Runge–Kutta formulas (the midpoint method),

\[
\begin{align*}
    k_1 & = h f(x_i, y_i), \\
    k_2 & = h f\left(x_i + \frac{1}{2} h, y_i + \frac{1}{2} k_1\right), \\
    y_{i+1} & = y_i + k_2,
\end{align*}
\]

using \( k_1 \) and \( k_2 \) to represent the update quantities for the unknown function \( u \) and calling the corresponding quantities for the function \( v \), \( m_1 \) and \( m_2 \). We must remember to update function \( f \) by the appropriate multiple of \( k_1 \) or \( k_2 \) and function \( g \) by the corresponding amount of \( m_1 \) or \( m_2 \). This means that \( k_1 \) and \( m_1 \) must be computed before \( k_2 \) and \( m_2 \) can be found. Thus,

\[
\begin{align*}
    k_1 & = h f(x_i, u_i, v_i), \\
    m_1 & = h g(x_i, u_i, v_i), \\
    k_2 & = h f\left(x_i + \frac{1}{2} h, u_i + \frac{1}{2} k_1, v_i + \frac{1}{2} m_1\right), \\
    m_2 & = h g\left(x_i + \frac{1}{2} h, u_i + \frac{1}{2} k_1, v_i + \frac{1}{2} m_1\right), \\
    u_{i+1} & = u_i + k_2, \\
    v_{i+1} & = v_i + m_2.
\end{align*}
\]

**MATLAB Function for Runge–Kutta Two-Step Method for Solving Two ODEs**

```matlab
function [ x, u, v ] = RK2_sys( f, g, a, b, u0, v0, n )
    h = (b-a)/n;   hh = h/2;
    x = (a+h : h : b);
    k1 = h*feval( f, a, u0, v0 );
    m1 = h*feval( g, a, u0, v0 );
    k2 = h*feval( f, a+hh, u0 +0.5*k1, v0 +0.5*m1);
    m2 = h*feval( g, a+hh, u0 +0.5*k1, v0 +0.5*m1);
    u(1) = u0 + k2;
    v(1) = v0 + m2;
```
for i = 1 : n-1
  k1 = h*feval(f, x(i), u(i), v(i));
  m1 = h*feval(g, x(i), u(i), v(i));
  k2 = h*feval(f, x(i) + hh, u(i) +0.5*k1, v(i) +0.5*m1);
  m2 = h*feval(g, x(i) + hh, u(i) +0.5*k1, v(i) +0.5*m1);
  u(i+1) = u(i) + k2;
  v(i+1) = v(i) + m2;
end
x = [a  x];  u = [ u0  u ];  v = [ v0  v ];

**Example 13.4 Nonlinear Pendulum using Runge–Kutta Method**

Consider again the system of ODEs obtained from the second-order ODE for the motion of the nonlinear pendulum described in Examples 13-A, 13.1, and 13.2, i.e.,

\[ u' = v = f(x, u, v), \]

\[ v' = -0.3v - \sin u = g(x, u, v), \]

with initial conditions

\[ u_0 = \pi/2; \quad v_0 = 0. \]

The motion for the first 15 seconds is shown in Fig. 13.4. The computed solutions for \( n = 50, n = 100, \) and \( n = 200 \) are indistinguishable. The motion of a linear pendulum and a nonlinear pendulum is illustrated in Fig. 13.5 for a larger initial displacement.

**FIGURE 13.4** Oscillations of a nonlinear pendulum.
13.3 SYSTEMS OF FIRST-ORDER ODE–IVPs

Systems of ODEs may arise directly from applications such as chemical reactions, predator–prey models, and many others. They also come from the conversion of higher order ODEs into system form.

Example 13.6 A Higher Order System of ODEs

Consider the equation

\[ y'''' = f(x, y, y', y'') = x + 2y - 3y' + 4y'' \]

with initial conditions

\[ y(0) = 4, \quad y'(0) = 3, \quad y''(0) = 2. \]

The system of ODEs is

\[ u_1' = f_1(x, u_1, u_2, u_3) = u_2, \]
\[ u_2' = f_2(x, u_1, u_2, u_3) = u_3, \]
\[ u_3' = f_3(x, u_1, u_2, u_3) = x + 2u_1 - 3u_2 + 4u_3. \]

For systems that come from a single higher order ODE, this structure for the right hand side is a direct result of the definitions of the transformed functions. For systems of ODEs in general, each of the right-hand-side functions \( f_1, f_2, \ldots \) may contain any or all of the indicated variables.
A system of ODEs can be expressed compactly in vector notation as

\[ \mathbf{u}' = \mathbf{f}(x, \mathbf{u}). \]

Since the components of the vectors, \( \mathbf{u} \) and \( \mathbf{f} \), are denoted by subscripts, we indicate the approximate solutions at the grid points as \( u_i(i) \), etc.

### 13.3.1 Euler’s Method for Solving Systems of ODEs

To apply the basic Euler method, \( y_{i+1} = y_i + h f(x_i, y_i) \), to the system of ODEs

\[ u_1'(i) = f_1(x, u_1, u_2, u_3), \]
\[ u_2'(i) = f_2(x, u_1, u_2, u_3), \]
\[ u_3'(i) = f_3(x, u_1, u_2, u_3), \]

we update the function \( u_1 \) using \( f_1 \), \( u_2 \) using \( f_2 \), and \( u_3 \) using \( f_3 \). The same step size \( h \) is used for each function. We have

\[ u_1(i+1) = u_1(i) + h f_1(x(i), u_1(i), u_2(i), u_3(i)), \]
\[ u_2(i+1) = u_2(i) + h f_2(x(i), u_1(i), u_2(i), u_3(i)), \]
\[ u_3(i+1) = u_3(i) + h f_3(x(i), u_1(i), u_2(i), u_3(i)). \]

### Example 13.7 Solving a Higher Order System using Euler’s Method

We apply Euler’s method with \( n = 2 \) to find an approximate solution of the system of ODEs

\[ u_1' = u_2, \]
\[ u_2' = u_3, \]
\[ u_3' = x + 2u_1 - 3u_2 + 4u_3, \]

with initial conditions \( u_1(0) = 4, u_2(0) = 3, \) and \( u_3(0) = 2 \) on \([0, 1]\).

The solution at \( i = 1 \) corresponds to \( x(i = 1) = 0.5 \):

\[ u_1(1) = u_1(0) + 0.5u_2(0) = 4 + 0.5(3) = 5.5, \]
The solution at \( i = 2 \) corresponds to \( x(i = 2) = 1.0 \):

\[
\begin{align*}
    u_1(2) &= u_1(1) + 0.5u_2(1) = 5.5 + 0.5(4) = 7.5, \\
    u_2(2) &= u_2(1) + 0.5u_3(1) = 4 + 0.5(5.5) = 6.75, \\
    u_3(2) &= u_3(1) + 0.5(x(1) + 2u_1(1) - 3u_2(1) + 4u_3(1)) \\
        &= 5.5 + 0.5(0.5 + 2(5.5) - 3(4) + 4(5.5)) = 11.25.
\end{align*}
\]

Example 13.8 Solving Another Higher Order System using Euler’s Method

Consider the system

\[
\begin{align*}
    u_1' &= u_2, & u_2' &= \frac{-2}{x} u_2, & u_3' &= u_4, & u_4' &= \frac{-2}{x} u_4,
\end{align*}
\]

with initial conditions

\[
\begin{align*}
    u_1(1) &= 10, & u_2(1) &= 0, & u_3(1) &= 0, & u_4(1) &= 1,
\end{align*}
\]

on \([1, 2]\) with \( n = 2 \) \((h = 0.5)\). The following calculations show the values of each component of the solution as a function of \( x \) (not the mesh index); at \( x = 3/2 \):

\[
\begin{align*}
    u_1(3/2) &= 10 + 0.5(0) = 10, & u_2(3/2) &= 0 + 0.5(0) = 0, \\
    u_3(3/2) &= 0 + 0.5(1) = 0.5, & u_4(3/2) &= 1 + 0.5(-2/1) = 0;
\end{align*}
\]

and at \( x = 2 \):

\[
\begin{align*}
    u_1(2) &= 10 + 0.5(0) = 10, & u_2(2) &= 0 + 0.5(0) = 0, \\
    u_3(2) &= 1/2 + 0.5(0) = 0.5, & u_4(2) &= 0 + 0.5(0) = 0.
\end{align*}
\]

These equations occur as part of the shooting method for solving the problem of finding the electrostatic potential between two concentric spheres, Example 14.1 in the next chapter.

13.3.2 Runge–Kutta Methods for Solving Systems of ODEs

The idea in generalizing Runge–Kutta methods for use on systems of equations is the same as for Euler’s method; that is, we update each unknown function \( u_1, u_2, \ldots \), using the basic Runge–Kutta formulas and the appropriate right-hand-side function \( f_1, f_2, \ldots \), from the differential equation for the unknown. We denote the two update parameters for a second order Runge–Kutta method as \( k \) and \( m \).

The basic second-order Runge–Kutta formulas (the midpoint method) are

\[
\begin{align*}
    k &= hf(x_i, y_i), & m &= hf\left(x_i + \frac{1}{2} h, y_i + \frac{1}{2} k\right), \\
    y_{i+1} &= y_i + m.
\end{align*}
\]
To apply these formulas to a system, we must compute $k$ and $m$ for each unknown function (i.e., for each component of the unknown vector $u$). In fact, $k$ must be computed for each unknown before $m$ can be found.

We illustrate the process for a system of three ODEs:

\[ \begin{align*}
    u_1' &= f_1(x, u_1, u_2, u_3), \\
    u_2' &= f_2(x, u_1, u_2, u_3), \\
    u_3' &= f_3(x, u_1, u_2, u_3).
\end{align*} \]

The values of the parameter $k$ for the unknown functions $u_1, u_2,$ and $u_3$ are

\[ \begin{align*}
    k_1 &= h f_1(x(i), u_1(i), u_2(i), u_3(i)), \\
    k_2 &= h f_2(x(i), u_1(i), u_2(i), u_3(i)), \\
    k_3 &= h f_3(x(i), u_1(i), u_2(i), u_3(i)).
\end{align*} \]

Similarly, the values of $m$ are $m_1, m_2,$ and $m_3$. Of course, to find the value of $m$ for the first ODE, we use $f_1$; however, we must evaluate $f_1$ at the appropriate values of $x, u_1, u_2,$ and $u_3$. Remembering that we are approximating the value of the unknown function employed in evaluating $f$ makes it clear that we approximate each $u$ using its value of $k$:

\[ \begin{align*}
    m_1 &= h f_1(x(i) + \frac{1}{2} h, u_1(i) + \frac{1}{2} k_1, u_2(i) + \frac{1}{2} k_2, u_3(i) + \frac{1}{2} k_3), \\
    m_2 &= h f_2(x(i) + \frac{1}{2} h, u_1(i) + \frac{1}{2} k_1, u_2(i) + \frac{1}{2} k_2, u_3(i) + \frac{1}{2} k_3), \\
    m_3 &= h f_3(x(i) + \frac{1}{2} h, u_1(i) + \frac{1}{2} k_1, u_2(i) + \frac{1}{2} k_2, u_3(i) + \frac{1}{2} k_3).
\end{align*} \]

Finally, the values of the unknown functions at the next grid point are found:

\[ \begin{align*}
    u_1(i + 1) &= u_1(i) + m_1, \\
    u_2(i + 1) &= u_2(i) + m_2, \\
    u_3(i + 1) &= u_3(i) + m_3.
\end{align*} \]

Example 13.9 Solving a Higher Order System using a Runge–Kutta Method

Let us use a Runge–Kutta method to find an approximate solution on the interval $[0, 1]$ of the system of ODEs

\[ \begin{align*}
    u_1' &= u_2, \\
    u_2' &= u_3, \\
    u_3' &= x + 2u_1 - 3u_2 + 4u_3,
\end{align*} \]

with initial conditions

\[ u_1(0) = 4, \quad u_2(0) = 3, \quad u_3(0) = 2. \]

With $n = 2$, we find the following values for $x, u_1, u_2,$ and $u_3$:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>4.0000</td>
<td>3.0000</td>
<td>2.0000</td>
</tr>
<tr>
<td>0.5</td>
<td>5.7500</td>
<td>4.8750</td>
<td>9.1250</td>
</tr>
<tr>
<td>1.0</td>
<td>9.7656</td>
<td>11.9844</td>
<td>32.8594</td>
</tr>
</tbody>
</table>
The following MATLAB function for the two-step Runge–Kutta method for systems relies on MATLAB’s characteristic of treating all variables as vectors. Thus, the only changes from the two-step Runge–Kutta program (midpoint method) in Chapter 12 that are required are the indexing of the matrix for the solution \( u \) at each step.

**MATLAB Function for Systems Using a Second-Order Runge–Kutta Method**

```matlab
function [ x , u ] = RK2_sys( f, tspan, u0, n)
% solve ODE-IVP using midpoint method (RK2)
% \( u' = f(x, u) \quad a \leq x \leq b \)
% interval of interest is given as tspan = [ a, b ]
% function \( f(x, u) \) returns a column vector of values
a = tspan(1);
b = tspan(2);
h = (b-a)/n;
x = (a+h : h : b);
k = h*feval( f, a, u0 )';
m = h*feval( f, a + h/2, u0 + k/2 )';
for i = 1 : n-1
    k = h*feval(f, x(i), u(i, : ) )';
    m = h*feval(f, x(i) + h/2, u(i, :) + k/2 )';
    u(i+1, :) = u(i, :) + m;
end
x = [a x];
u = [ u0,
    u ];
```

**Example 13.10 Solving a Higher Order System using a Runge–Kutta Method**

The system

\[ u_1' = u_2, \quad u_2' = \frac{-2}{x} u_2, \quad u_3' = u_4, \quad u_4' = \frac{-2}{x} u_4, \]

with initial conditions

\[ u_1(1) = 10, \quad u_2(1) = 0, \quad u_3(1) = 0, \quad u_4(1) = 1, \]

on the interval [1, 2], arises in the solution of the differential equation describing the electrostatic potential between two concentric spheres, one of radius 1 and the other of radius 2.
Using \( n = 2 \) (and \( h = 0.5 \)), we calculate the values of each component of the solution as a function of \( x \) (not the mesh index). First, we find \( k \) for each component:

\[
\begin{align*}
  k_1 &= 0.5(u_2(1)) = 0, & k_2 &= 0.5 \left( \frac{-2}{x} u_2(1) \right) = 0, \\
  k_3 &= 0.5(u_4(1)) = 0.5, & k_4 &= 0.5 \left( \frac{-2}{x} u_4(1) \right) = -1,
\end{align*}
\]

Next, we find \( m \) for each component:

\[
\begin{align*}
  m_1 &= 0.5(u_2(1) + 0.5k_2) = 0, \\
  m_2 &= 0.5 \left( \frac{-2}{1.25} \right)(u_2(1) + 0.5k_2) = 0, \\
  m_3 &= 0.5(u_4(1) + 0.5k_4) = 0.25, \\
  m_4 &= 0.5 \left( \frac{-2}{1.25} \right)(u_4(1) + 0.5k_4) = -0.4.
\end{align*}
\]

The approximate solution at \( x = 1.5 \) is

\[
\begin{align*}
  u_1(1.5) &= 10 + 0 = 10, & u_2(1.5) &= 0 + 0 = 0, \\
  u_3(1.5) &= 0 + 0.25 = 0.25, & u_4(1.5) &= 1 - 0.4 = 0.6.
\end{align*}
\]

Now we again find \( k \) for each component:

\[
\begin{align*}
  k_1 &= 0.5(u_2(1.5)) = 0, & k_2 &= 0.5 \left( \frac{-2}{1.5} u_2(1.5) \right) = 0, \\
  k_3 &= 0.5(u_4(1.5)) = 0.3, & k_4 &= 0.5 \left( \frac{-2}{1.5} u_4(1.5) \right) = -0.4.
\end{align*}
\]

Next, we again find \( m \) for each component:

\[
\begin{align*}
  m_1 &= 0.5(u_2(1.5) + 0.5k_2) = 0, \\
  m_2 &= 0.5 \left( \frac{-2}{1.75} \right)(u_2(1.5) + 0.5k_2) = 0, \\
  m_3 &= 0.5(u_4(1.5) + 0.5k_4) = 0.2, \\
  m_4 &= 0.5 \left( \frac{-2}{1.75} \right)(u_4(1.5) + 0.5k_4) = -0.2286.
\end{align*}
\]

The approximate solution at \( x = 1.0 \) is

\[
\begin{align*}
  u_1(1.0) &= 10 + 0 = 10, & u_2(1.0) &= 0 + 0 = 0, \\
  u_3(1.0) &= 0.25 + 0.2 = 0.45, & u_4(1.0) &= 0.6 - 0.2286 = 0.3714.
\end{align*}
\]

Using \( n = 20 \), the foregoing MATLAB function gives the results summarized in Table 13.1.
Table 13.1 ODE–IVP system for electrostatic potential problem.

<table>
<thead>
<tr>
<th></th>
<th>( u_1 )</th>
<th>( u_2 )</th>
<th>( u_3 )</th>
<th>( u_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>1.10</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.0907</td>
<td>0.8269</td>
</tr>
<tr>
<td>1.20</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.1664</td>
<td>0.6952</td>
</tr>
<tr>
<td>1.30</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.2304</td>
<td>0.5925</td>
</tr>
<tr>
<td>1.40</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.2854</td>
<td>0.5110</td>
</tr>
<tr>
<td>1.50</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.3330</td>
<td>0.4453</td>
</tr>
<tr>
<td>1.60</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.3747</td>
<td>0.3914</td>
</tr>
<tr>
<td>1.70</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.4115</td>
<td>0.3468</td>
</tr>
<tr>
<td>1.80</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.4443</td>
<td>0.3094</td>
</tr>
<tr>
<td>1.90</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.4735</td>
<td>0.2777</td>
</tr>
<tr>
<td>2.00</td>
<td>10.0000</td>
<td>0.0000</td>
<td>0.4999</td>
<td>0.2506</td>
</tr>
</tbody>
</table>

MATLAB Function for Systems Using a Fourth-Order Runge–Kutta Method

```matlab
% function f(x, u):
%
%   input: column vector x and row vector u
%   return: column vector of values for u'

a = tspan(1); b = tspan(2); h = (b-a) / n;
x = (a+h : h : b)';
k1 = h *feval( f, a, u0 )';
k2 = h *feval( f, a+h/2, u0+k1/2 )';
k3 = h *feval( f, a+h/2, u0+k2/2 )';
k4 = h *feval( f, a+h, u0+k3 )';
u(1, :) = u0 + k1/6 + k2/3 + k3/3 + k4/6;
for i = 1 : n-1
    k1 = h *feval( f, x(i), u(i,:) )';
    k2 = h *feval( f, x(i)+h/2, u(i,:)+k1/2 )';
    k3 = h *feval( f, x(i)+h/2, u(i,:)+k2/2 )';
    k4 = h *feval( f, x(i)+h, u(i,:)+k3 )';
    u(i+1, :) = u(i, :) + k1/6 + k2/3 + k3/3 + k4/6;
end
x = [a x];
u = [u0 u];
```
13.3.3 Multistep Methods for Systems

The basic two-step Adams–Bashforth method, in which \( y_0 \) is given by the initial condition for the differential equation, \( y_1 \) is found from a one-step method, such as a Runge–Kutta technique, and for \( i = 1, \ldots, n - 1 \) and \( h = \frac{b - a}{n} \),

\[
y_{i+1} = y_i + \frac{h}{2} [3f(x_i, y_i) - f(x_{i-1}, y_{i-1})],
\]
can be extended for use with a system of three ODEs

\[
\begin{align*}
   u'_1 &= f_1(x, u_1, u_2, u_3), \\
   u'_2 &= f_2(x, u_1, u_2, u_3), \\
   u'_3 &= f_3(x, u_1, u_2, u_3),
\end{align*}
\]
in a similarly straightforward manner. That is, \( u_1(i = 0), u_2(i = 0), u_3(i = 0) \) are given by the initial condition, \( u_1(i = 1), u_2(i = 1), u_3(i = 1) \) are found from a one-step method, and for \( i = 1, \ldots, n - 1 \) and \( h = \frac{b - a}{n} \),

\[
\begin{align*}
   u_1(i + 1) &= u_1(i) + \frac{h}{2} [3f_1(x(i), u_1(i), u_2(i), u_3(i)) \\
   &\quad \quad - f_1(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1))], \\
   u_2(i + 1) &= u_2(i) + \frac{h}{2} [3f_2(x(i), u_1(i), u_2(i), u_3(i)) \\
   &\quad \quad - f_2(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1))], \\
   u_3(i + 1) &= u_3(i) + \frac{h}{2} [3f_3(x(i), u_1(i), u_2(i), u_3(i)) \\
   &\quad \quad - f_3(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1))].
\end{align*}
\]

The Adams–Bashforth-Moulton predictor-corrector methods are extended for use with systems of ODEs in a similar manner. The third-order method is implemented in the MATLAB function `ABM3_sys` that follows (on p. 468). The dimensions of the vectors \( \mathbf{u}, \mathbf{u}_0, \) and `tspan` depend on the ODE system being solved, as illustrated in Examples 13.12 and 13.13.
There are ODE for which any error that occurs will increase, regardless of the numerical method employed. Such problems are called *ill conditioned*. As an illustration, consider the system

\[
\begin{align*}
    u_1' &= 2u_2 \\
    u_2' &= 2u_1
\end{align*}
\]

for which the general solution is

\[
\begin{align*}
    u_1 &= a e^{2x} + b e^{-2x} \\
    u_2 &= a e^{2x} - b e^{-2x}
\end{align*}
\]

With the initial conditions

\[
\begin{align*}
    u_1(0) &= 3 \\
    u_2(0) &= -3
\end{align*}
\]

we have \(a = 0, b = 3\). However, for any numerical error that occurs, a component of the positive exponential will be introduced and will eventually dominate the true solution.

Ill conditioning can also occur for a single first-order ODE, as the following problem shows. Consider the ODE

\[y' = 3y - t^2\]

for which the general solution is

\[y = Ce^{3t} + \frac{1}{3} t^2 + \frac{2}{9} t + \frac{2}{27}\]

If we take the initial condition as \(y(0) = \frac{2}{27}\), the exact solution is

\[y = \frac{1}{3} t^2 + \frac{2}{9} t + \frac{2}{27}\]

However, any error in the numerical solution process will introduce the exponential component which will eventually dominate the true solution. The exponential term is known as a *parasitic solution*.

An ODE in which there is a rapidly decaying transient solution also causes difficulties for numerical solution, requiring an extremely small step size in order to obtain an accurate solution. One source of such equations is in the description of a spring-mass system with large spring constants, hence these problems are known as *stiff ODE*. Stiff ODEs are very common in chemical kinetic studies, and also occur in many network analysis and simulation problems.
As an illustration, consider the system

\[ u' = 98u + 198v, \]
\[ v' = -99u + 199v, \]

with initial conditions \( u(0) = 1, v(0) = 0 \).

The exact solution is

\[ u(t) = 2 e^{-t} - e^{-100t}, \]
\[ v(t) = -e^{-t} + e^{-100t}. \]

It is also possible for a single first-order ODE to be stiff, as the following problem shows. Consider the ODE

\[ y' = \lambda(y - g(t)) + g'(t) \]

with \( \lambda \ll 0 \) and \( g(t) \) a smooth, slowly varying function. The solution is

\[ y = (y_0 - g(0)) e^{\lambda t} + g(t). \]

The first term in the solution will soon be insignificant compared with \( g(t) \), but stability will continue to be governed by \( h\lambda \), necessitating a very small step size.

For a system of equations

\[ y' = A(y - g(t)) + g'(t) \]

the eigenvalues of \( A \) correspond to \( \lambda \); if all of the eigenvalues have negative real parts, the solution will converge towards \( g(t) \) as \( t \to \infty \).

The simplest method for stiff problems is the backward Euler method

\[ y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}). \]

The error is amplified by \( (1 - h\lambda)^{-1} \) at each step, which is less than one if \( \text{Re}(\lambda) < 0 \). Thus, the backward Euler’s method is \( A \)-stable, according to the following definition.

A method is called \( A \)-stable if any solution produced when the method is applied (with fixed step size \( h > 0 \)) to the problem \( y' = \lambda y \) (with \( \lambda = \alpha + \beta i \) and \( \alpha < 0 \)) tends to zero as \( n \to \infty \).

Dahlquist (1963) showed that a multistep method that is \( A \)-stable cannot have order greater than two. The trapezoid method is the second-order multistep method with the smallest error constant (see summary for Chapter 12).

Since \( A \)-stability is difficult to achieve, a somewhat less restrictive stability condition, known as stiff-stability is often sufficient. Methods for stiff ODE are implicit and often require iterative techniques for their solution. Newton’s method may be used, with the required Jacobian either supplied by the user, or generated numerically. (See Gear, 1971 for further discussion.)
Convert Higher Order ODE to System of First-Order ODE: The $n$th-order ODE

\[ y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}), \]

\[ y(0) = \alpha_0, \quad y'(0) = \alpha_1, \quad y''(0) = \alpha_2, \quad \ldots, \quad y^{(n-1)}(0) = \alpha_{n-1}, \]

becomes a system of first-order ODEs by the following change of variables:

\[ u_1 = y, \quad u_2 = y', \quad u_3 = y'', \quad \ldots, \quad u_n = y^{(n-1)}. \]

The differential equations relating these variables are

\[ u_1' = u_2, \quad u_2' = u_3, \quad u_3' = u_4, \quad \ldots, \quad u_n' = f(x, u_1, u_2, u_3, \ldots, u_n), \]

with the initial conditions: $u_1(0) = \alpha_0, u_2(0) = \alpha_1, u_3(0) = \alpha_2, \ldots, u_n(0) = \alpha_{n-1}$.

Solve a System of Two First-Order ODEs $u' = f(x, u, v), v' = g(x, u, v)$:

Euler’s method updates $u$ using $f(x, u, v)$ and updates $v$ using $g(x, u, v)$:

\[ u_{i+1} = u_i + hf(x_i, u_i, v_i), \quad v_{i+1} = v_i + h g(x_i, u_i, v_i). \]

The midpoint method can be written using $k_1$ and $k_2$ to represent the update quantities for the unknown function $u$, $m_1$ and $m_2$ give the corresponding quantities for function $v$. We have

\[ k_1 = hf(x_i, u_i, v_i), \quad m_1 = hg(x_i, u_i, v_i), \]

\[ k_2 = hf \left( x_i + \frac{h}{2} u_i + \frac{1}{2} k_1, v_i + \frac{1}{2} m_1 \right), \quad m_2 = hg \left( x_i + \frac{h}{2} u_i + \frac{1}{2} k_1, v_i + \frac{1}{2} m_1 \right), \]

\[ u_{i+1} = u_i + k_2, \quad v_{i+1} = v_i + m_2. \]

Solve a System of Three First-Order ODEs

The basic two-step Adams-Bashforth method for a system of three ODEs

\[ u_1' = f_1(x, u_1, u_2, u_3), \quad u_2' = f_2(x, u_1, u_2, u_3), \quad u_3' = f_3(x, u_1, u_2, u_3), \]

is described as follows:

$u_1(i = 0), u_2(i = 0), u_3(i = 0)$ are given by the initial condition, $u_1(i = 1), u_2(i = 1)$, $u_3(i = 1)$ are found from a 1-step method, and for $i = 1, \ldots, n - 1$, and $h = \frac{b - a}{n}$:
\[ u_1(i + 1) = u_1(i) + \frac{h}{2} \left[ 3f_1(x(i), u_1(i), u_2(i), u_3(i) \right. \]
\[ \quad - f_1(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1)) \left. \right], \]
\[ u_2(i + 1) = u_2(i) + \frac{h}{2} \left[ 3f_2(x(i), u_1(i), u_2(i), u_3(i) \right. \]
\[ \quad - f_2(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1)) \left. \right], \]
\[ u_3(i + 1) = u_3(i) + \frac{h}{2} \left[ 3f_3(x(i), u_1(i), u_2(i), u_3(i) \right. \]
\[ \quad - f_3(x(i - 1), u_1(i - 1), u_2(i - 1), u_3(i - 1)) \left. \right]. \]

**SUGGESTIONS FOR FURTHER READING**

The suggested readings for Chapter 12 are also excellent references for the topics in this chapter. In addition, the following texts include discussion of applications of ODEs:


