6.3 SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS

Approximation of systems of differential equations can be done as a simple extension of the methodology for a single differential equation. Treating systems of equations greatly extends our ability to model interesting dynamical behavior.

The ability to solve systems of ordinary differential equations lies at the core of the art and science of computer simulation. In this section, we introduce two physical systems whose simulation has motivated a great deal of development of ODE solvers: the pendulum and orbital mechanics. The study of these examples will provide the reader some practical experience in the capabilities and limitations of the solvers.

The **order** of a differential equation refers to the highest order derivative appearing in the equation. A first-order system has the form

$$\begin{aligned} y_1' &= f_1(t, y_1, \ldots, y_n) \\ y_2' &= f_2(t, y_1, \ldots, y_n) \\ &\vdots \\ y_n' &= f_n(t, y_1, \ldots, y_n). \end{aligned}$$

In an initial value problem, each variable needs its own initial condition.

► EXAMPLE 6.13 Apply Euler's Method to the first-order system of two equations:

$$\begin{aligned} y_1' &= y_2^2 - 2y_1 \\ y_2' &= y_1 - y_2 - ty_2^2 \\ y_1(0) &= 0 \\ y_2(0) &= 1. \end{aligned} \tag{6.36}$$

Check that the solution of the system (6.36) is the vector-valued function

$$\begin{aligned} y_1(t) &= te^{-2t} \\ y_2(t) &= e^{-t}. \end{aligned}$$

For the moment, forget that we know the solution, and apply Euler's Method. The scalar Euler's Method formula is applied to each component in turn as follows:

$$\begin{aligned} w_{i+1,1} &= w_{i,1} + h(w_{i,2}^2 - 2w_{i,1}) \\ w_{i+1,2} &= w_{i,2} + h(w_{i,1} - w_{i,2} - t_i w_{i,2}^2). \end{aligned}$$

Figure 6.9 shows the Euler Method approximations of y_1 and y_2, along with the correct solution. The MATLAB code that carries this out is essentially the same as Program 6.1, with a few adjustments to treat y as a vector:

```
% Program 6.2  Vector version of Euler Method
% Input: interval inter, initial vector y0, number of steps n
% Output: time steps t, solution y
% Example usage: euler2([0 1],[0 1],10);
function [t,y]=euler2(inter,y0,n)
t(1)=inter(1); y(1,:)=y0;
h=(inter(2)-inter(1))/n;
for i=1:n
  t(i+1)=t(i)+h;
```

```
  y(i+1,:)=eulerstep(t(i),y(i,:),h);
end
plot(t,y(:,1),t,y(:,2));

function y=eulerstep(t,y,h)
%one step of the Euler Method
%Input: current time t, current vector y, step size h
%Output: the approximate solution vector at time t+h
y=y+h*ydot(t,y);

function z=ydot(t,y)
%right-hand side of differential equation
z(1)=y(2)^2-2*y(1);
z(2)=y(1)-y(2)-t*y(2)^2;
```

6.3.1 Higher order equations

A single differential equation of higher order can be converted to a system. Let

$$y^{(n)} = f(t, y, y', y'', \ldots, y^{(n-1)})$$

be an nth-order ordinary differential equation. Define new variables

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_n &= y^{(n-1)}, \end{aligned}$$

and notice that the original differential equation can be written

$$y_n' = f(t, y_1, y_2, \ldots, y_n).$$

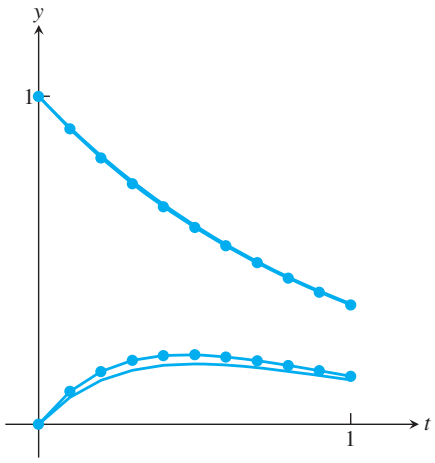

Figure 6.9 Equation (6.36) approximated by Euler Method. Step size $h = 0.1$. The upper curve is $y_1(t)$, along with its approximate solution $w_{i,1}$ (circles), while the lower curve is $y_2(t)$ and $w_{i,2}$.

Taken together, the equations

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ y_3' &= y_4 \\ &\vdots \\ y_{n-1}' &= y_n, \\ y_n' &= f(t, y_1, \ldots, y_n) \end{aligned}$$

convert the nth-order differential equation into a system of first-order equations, which can be solved by using methods like the Euler or Trapezoid Methods.

► EXAMPLE 6.14 Convert the third-order differential equation

$$y''' = a(y'')^2 - y' + yy'' + \sin t \tag{6.37}$$

to a system.

Set $y_1 = y$ and define the new variables

$$\begin{aligned} y_2 &= y' \\ y_3 &= y''. \end{aligned}$$

Then, in terms of first derivatives, (6.37) is equivalent to

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ y_3' &= a y_3^2 - y_2 + y_1 y_3 + \sin t. \end{aligned} \tag{6.38}$$

The solution $y(t)$ of the third-order equation (6.37) can be found by solving the system (6.38) for $y_1(t), y_2(t), y_3(t)$. ◄

Because of the possibility of converting higher-order equations to systems, we will restrict our attention to systems of first-order equations. Note also that a system of several higher-order equations can be converted to a system of first-order equations in the same way.

6.3.2 Computer simulation: the pendulum

Figure 6.10 shows a pendulum swinging under the influence of gravity. Assume that the pendulum is hanging from a rigid rod that is free to swing through 360 degrees. Denote by y the angle of the pendulum with respect to the vertical, so that $y = 0$ corresponds to straight down. Therefore, y and $y + 2\pi$ are considered the same angle.

Newton's second law of motion $F = ma$ can be used to find the pendulum equation. The motion of the pendulum bob is constrained to be along a circle of radius l, where l is the length of the pendulum rod. If y is measured in radians, then the component of acceleration tangent to the circle is ly'', because the component of position tangent to the circle is ly. The component of force along the direction of motion is $mg\sin y$. It is a restoring force, meaning that it is directed in the opposite direction from the displacement of the variable y. The differential equation governing the frictionless pendulum is therefore

$$mly'' = F = -mg\sin y. \tag{6.39}$$

This is a second-order differential equation for the angle y of the pendulum. The initial conditions are given by the initial angle $y(0)$ and angular velocity $y'(0)$.

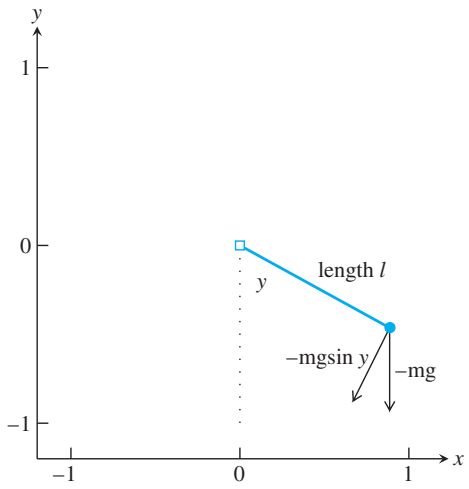

Figure 6.10 The pendulum. Component of force in the tangential direction is $F = -mg\sin y$, where y is the angle the pendulum bob makes with the vertical.

By setting $y_1 = y$ and introducing the new variable $y_2 = y'$, the second-order equation is converted to a first-order system:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -\frac{g}{l}\sin y_1 . \end{aligned} \tag{6.40}$$

The system is autonomous because there is no t dependence in the right-hand side. If the pendulum is started from a position straight out to the right, the initial conditions are $y_1(0) = \pi/2$ and $y_2(0) = 0$. In MKS units, the gravitational acceleration at the earth's surface is about 9.81m/sec^2. Using these parameters, we can test the suitability of Euler's Method as a solver for this system.

Figure 6.11 shows Euler's Method approximations to the pendulum equations with two different step sizes. The pendulum rod is assigned to be $l = 1$ meter in length. The smaller curve represents the angle y as a function of time, and the larger amplitude curve is the instantaneous angular velocity. Note that the zeros of the angle, representing the vertical position of the pendulum, correspond to the largest angular velocity, positive or negative. The pendulum is traveling fastest as it swings through the lowest point. When the pendulum is extended to the far right, the peak of the smaller curve, the velocity is zero as it turns from positive to negative.

The inadequacy of Euler's Method is apparent in Figure 6.11. The step size $h = 0.01$ is clearly too large to achieve even qualitative correctness. An undamped pendulum started with zero velocity should swing back and forth forever, returning to its starting position with a regular periodicity. The amplitude of the angle in Figure 6.11(a) is growing, which violates the conservation of energy. Using 10 times more steps, as in Figure 6.11(b), improves the situation at least visually, but a total of 10^4 steps are needed, an extreme number for the routine dynamical behavior shown by the pendulum.

A second-order ODE solver like the Trapezoid Method improves accuracy at a much lower cost. We will rewrite the MATLAB code to use the Trapezoid Method and take the opportunity to illustrate the ability of MATLAB to do simple animations.

The code `pend.m` that follows contains the same differential equation information, but `eulerstep` is replaced by `trapstep`. In addition, the variables `rod` and `bob` are introduced to represent the rod and pendulum bob, respectively. The MATLAB `set` command assigns attributes to variables. The `drawnow` command plots the `rod` and `bob` variables. Note that the erase mode of both variables is set to `xor`, meaning that when the plotted

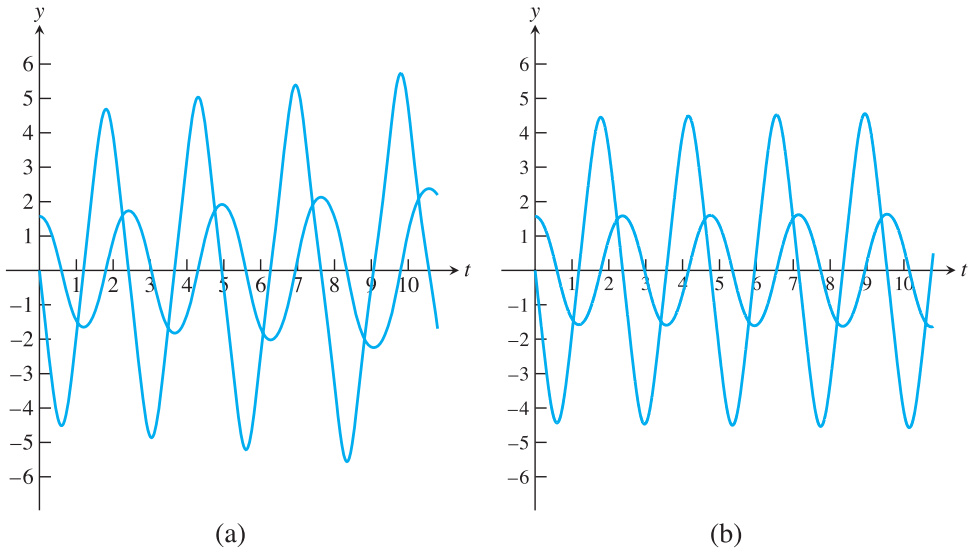

Figure 6.11 Euler Method applied to the pendulum equation (6.40). The curve of smaller amplitude is the angle y_1 in radians; the curve of larger amplitude is the angular velocity y_2. (a) Step size $h = 0.01$ is too large; energy is growing. (b) Step size $h = 0.001$ shows more accurate trajectories.

variable is redrawn somewhere else, the previous position is erased. Figure 6.10 is a screen shot of the animation. Here is the code:

```
% Program 6.3 Animation program for pendulum
% Inputs: time interval inter,
%  initial values ic=[y(1,1) y(1,2)], number of steps n
% Calls a one-step method such as trapstep.m
% Example usage: pend([0 10],[pi/2 0],200)
function pend(inter,ic,n)
h=(inter(2)-inter(1))/n; % plot n points in total
y(1,:)=ic;                    % enter initial conds in y
t(1)=inter(1);
set(gca,'xlim',[-1.2 1.2],'ylim',[-1.2 1.2], ...
  'XTick',[-1 0 1],'YTick',[-1 0 1], ...
  'Drawmode','fast','Visible','on','NextPlot','add');
cla;
axis square                % make aspect ratio 1 - 1
bob=line('color','r','Marker','.','markersize',40,...
    'erase','xor','xdata',[],'ydata',[]);
rod=line('color','b','LineStyle','-','LineWidth',3,...
    'erase','xor','xdata',[],'ydata',[]);
for k=1:n
  t(k+1)=t(k)+h;
  y(k+1,:)=trapstep(t(k),y(k,:),h);
  xbob=sin(y(k+1,1)); ybob= -cos(y(k+1,1));
  xrod=[0 xbob]; yrod=[0 ybob];
  set(rod,'xdata',xrod,'ydata',yrod)
  set(bob,'xdata',xbob,'ydata',ybob)
  drawnow; pause(h)
end

function y=trapstep(t,x,h)
%one step of the Trapezoid Method
```

```
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function z=ydot(t,y)
g=9.81;length=1;
z(1)=y(2);
z(2)=-(g/length)*sin(y(1));
```

Using the Trapezoid Method in the pendulum equation allows fairly accurate solutions to be found with larger step size. This section ends with several interesting variations on the basic pendulum simulation, which the reader is encouraged to experiment with in the Computer Problems.

▶ EXAMPLE 6.15 The damped pendulum.

The force of damping, such as air resistance or friction, is often modeled as being proportional and in the opposite direction to velocity. The pendulum equation becomes

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -\frac{g}{l}\sin y_1 - d y_2, \end{aligned} \tag{6.41}$$

where $d > 0$ is the damping coefficient. Unlike the undamped pendulum, this one will lose energy through damping and in time approach the limiting equilibrium solution $y_1 = y_2 = 0$, from any initial condition. Computer Problem 3 asks you to run a damped version of `pend.m`. ◀

▶ EXAMPLE 6.16 The forced damped pendulum.

Adding a time-dependent term to (6.41) represents outside forcing on the damped pendulum. Consider adding the sinusoidal term $A\sin t$ to the right-hand side of y_2', yielding

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -\frac{g}{l}\sin y_1 - d y_2 + A\sin t. \end{aligned} \tag{6.42}$$

This can be considered as a model of a pendulum that is affected by an oscillating magnetic field, for example.

A host of new dynamical behaviors becomes possible when forcing is added. For a two-dimensional autonomous system of differential equations, the Poincaré–Bendixson Theorem (from the theory of differential equations) says that trajectories can tend toward only regular motion, such as stable equilibria like the down position of the pendulum, or stable periodic cycles like the pendulum swinging back and forth forever. The forcing makes the system nonautonomous (it can be rewritten as a three-dimensional autonomous system, but not as a two-dimensional one), so that a third type of trajectories is allowed, namely, chaotic trajectories.

Setting the damping coefficient to $d = 1$ and the forcing coefficient to $A = 10$ results in interesting periodic behavior, explored in Computer Problem 4. Moving the parameter to $A = 15$ introduces chaotic trajectories. ◀

► EXAMPLE 6.17 The double pendulum.

The double pendulum is composed of a simple pendulum, with another simple pendulum hanging from its bob. If y_1 and y_3 are the angles of the two bobs with respect to the vertical, the system of differential equations is

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \frac{-3g\sin y_1 - g\sin(y_1 - 2y_3) - 2\sin(y_1 - y_3)(y_4^2 - y_2^2\cos(y_1 - y_3))}{3 - \cos(2y_1 - 2y_3)} - dy_2 \\ y_3' &= y_4 \\ y_4' &= \frac{2\sin(y_1 - y_3)[2y_2^2 + 2g\cos y_1 + y_4^2\cos(y_1 - y_3)]}{3 - \cos(2y_1 - 2y_3)}, \end{aligned}$$

where $g = 9.81$ and the length of both rods has been set to 1. The parameter d represents friction at the pivot. For $d = 0$, the double pendulum exhibits sustained nonperiodicity for many initial conditions and is mesmerizing to observe. See Computer Problem 8. ◄

6.3.3 Computer simulation: orbital mechanics

As a second example, we simulate the motion of an orbiting satellite. Newton's second law of motion says that the acceleration a of the satellite is related to the force F applied to the satellite as $F = ma$, where m is the mass. The law of gravitation expresses the force on a body of mass m_1 due to a body of mass m_2 by an inverse-square law

$$F = \frac{gm_1m_2}{r^2},$$

where r is the distance separating the masses. In the **one-body problem**, one of the masses is considered negligible compared with the other, as in the case of a small satellite orbiting a large planet. This simplification allows us to neglect the force of the satellite on the planet, so that the planet may be regarded as fixed.

Place the large mass at the origin, and denote the position of the satellite by (x, y). The distance between the masses is $r = \sqrt{x^2 + y^2}$, and the force on the satellite is central—that is, in the direction of the large mass. The direction vector, a unit vector in this direction, is

$$\left(-\frac{x}{\sqrt{x^2 + y^2}}, -\frac{y}{\sqrt{x^2 + y^2}}\right).$$

Therefore, the force on the satellite in terms of components is

$$(F_x, F_y) = \left(\frac{gm_1m_2}{x^2 + y^2}\frac{-x}{\sqrt{x^2 + y^2}}, \frac{gm_1m_2}{x^2 + y^2}\frac{-y}{\sqrt{x^2 + y^2}}\right). \tag{6.43}$$

Inserting these forces into Newton's law of motion yields the two second-order equations

$$\begin{aligned} m_1x'' &= -\frac{gm_1m_2x}{(x^2 + y^2)^{3/2}} \\ m_1y'' &= -\frac{gm_1m_2y}{(x^2 + y^2)^{3/2}}. \end{aligned}$$

Introducing the variables $v_x = x'$ and $v_y = y'$ allows the two second-order equations to be reduced to a system of four first-order equations:

$$
\begin{aligned}
x' &= v_x \\
v_x' &= -\frac{gm_2 x}{(x^2+y^2)^{3/2}} \\
y' &= v_y \\
v_y' &= -\frac{gm_2 y}{(x^2+y^2)^{3/2}}
\end{aligned}
\tag{6.44}
$$

The following MATLAB program `orbit.m` calls `eulerstep.m` and sequentially plots the satellite orbit.

```
%Program 6.4 Plotting program for one-body problem
%Inputs: time interval inter, initial conditions
% ic=[x0 vx0 y0 vy0], x position, x velocity, y pos, y vel,
% number of steps n, steps per point plotted p
% Calls a one-step method such as trapstep.m
% Example usage: orbit([0 100],[0 1 2 0],10000,5)
function z=orbit(inter,ic,n,p)
h=(inter(2)-inter(1))/n;              % plot n points
x0=ic(1);vx0=ic(2);y0=ic(3);vy0=ic(4);      % grab initial conds
y(1,:)=[x0 vx0 y0 vy0];t(1)=inter(1);       % build y vector
set(gca,'XLim',[-5 5],'YLim',[-5 5],'XTick',[-5 0 5],'YTick',...
  [-5 0 5],'Drawmode','fast','Visible','on');
cla;
sun=line('color','y','Marker','.','markersize',25,...
  'xdata',0,'ydata',0);
drawnow;
head=line('color','r','Marker','.','markersize',25,...
  'erase','xor','xdata',[],'ydata',[]);
tail=line('color','b','LineStyle','-','erase','none', ...
  'xdata',[],'ydata',[]);
%[px,py]=ginput(1);       % include these three lines
%[px1,py1]=ginput(1);     % to enable mouse support
%y(1,:)=[px px1-px py py1-py];   % 2 clicks set direction
for k=1:n/p
  for i=1:p
    t(i+1)=t(i)+h;
    y(i+1,:)=eulerstep(t(i),y(i,:),h);
  end
  y(1,:)=y(p+1,:);t(1)=t(p+1);
  set(head,'xdata',y(1,1),'ydata',y(1,3))
  set(tail,'xdata',y(2:p,1),'ydata',y(2:p,3))
  drawnow;
end

function y=eulerstep(t,x,h)
%one step of the Euler Method
y=x+h*ydot(t,x);

function z=ydot(t,x)
m2=3;g=1;mg2=m2*g;px2=0;py2=0;
px1=x(1);py1=x(3);vx1=x(2);vy1=x(4);
dist=sqrt((px2-px1)^2+(py2-py1)^2);
z=zeros(1,4);
z(1)=vx1;
z(2)=(mg2*(px2-px1))/(dist^3);
z(3)=vy1;
z(4)=(mg2*(py2-py1))/(dist^3);
```

Running the MATLAB script `orbit.m` immediately shows the limitations of Euler's Method for approximating interesting problems. Figure 6.12(a) shows the outcome of running `orbit([0 100],[0 1 2 0],10000,5)`. In other words, we follow the orbit over the time interval $[a, b] = [0, 100]$, the initial position is $(x_0, y_0) = (0, 2)$, the initial velocity is $(v_x, v_y) = (1, 0)$, and the Euler step size is $h = 100/10000 = 0.01$.

Solutions to the one-body problem must be conic sections—either ellipses, parabolas, or hyperbolas. The spiral seen in Figure 6.12(a) is a numerical artifact, meaning a misrepresentation caused by errors of computation. In this case, it is the truncation error of Euler's Method that leads to the failure of the orbit to close up into an ellipse. If the step size is cut by a factor of 10 to $h = 0.001$, the result is improved, as shown in Figure 6.12(b). It is clear that even with the greatly decreased step size, the accumulated error is noticeable.

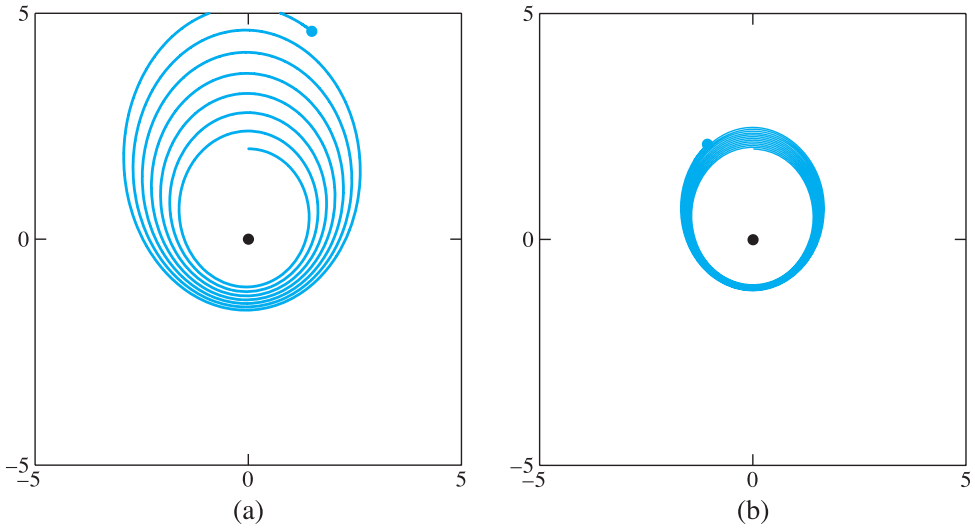

Figure 6.12 Euler's Method applied to one-body problem. (a) $h = 0.01$ and (b) $h = 0.001$.

Corollary 6.5 says that the Euler Method, in principle, can approximate a solution with as much accuracy as desired, if the step size h is sufficiently small. However, results like those represented by Figures 6.6 and 6.12 show that the method is seriously limited in practice.

Figure 6.13 shows the clear improvement in the one-body problem resulting from the replacement of the Euler step with the Trapezoid step. The plot was made by replacing the function `eulerstep` by `trapstep` in the foregoing code.

The one-body problem is fictional, in the sense that it ignores the force of the satellite on the (much larger) planet. When the latter is included as well, the motion of the two objects is called the two-body problem.

The case of three objects interacting gravitationally, called the **three-body problem**, holds an important position in the history of science. Even when all motion is confined to a plane (the **restricted** three-body problem) the long-term trajectories may be essentially unpredictable. In 1889, King Oscar II of Sweden and Norway held a competition for work proving the stability of the solar system. The prize was awarded to Henri Poincaré, who showed that it would be impossible to prove any such thing, due to phenomena seen even for three interacting bodies.

The unpredictability stems from **sensitive dependence on initial conditions**, a term which denotes the fact that small uncertainties in the initial positions and velocities can

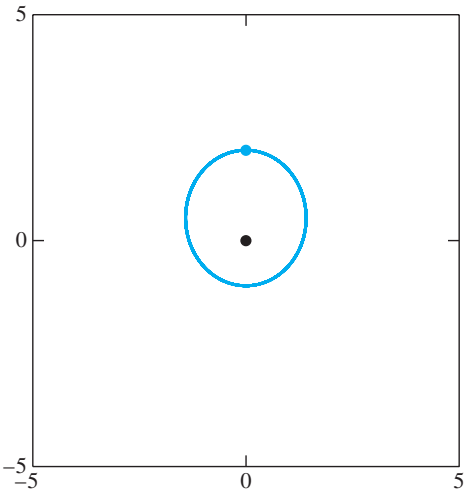

Figure 6.13 One-body problem approximated by the Trapezoid Method. Step size $h = 0.01$. The orbit appears to close, at least to the resolution visible in the plot.

lead to large deviations at a later time. In our terms, this is the statement that the solution of the system of differential equations is ill-conditioned with respect to the input of initial conditions.

The restricted three-body problem is a system of 12 equations, 4 for each body, that are also derived from Newton's second law. For example, the equations of the first body are

$$\begin{aligned} x_1' &= v_{1x} \\ v_{1x}' &= \frac{gm_2(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^{3/2}} + \frac{gm_3(x_3 - x_1)}{((x_3 - x_1)^2 + (y_3 - y_1)^2)^{3/2}} \\ y_1' &= v_{1y} \\ v_{1y}' &= \frac{gm_2(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^{3/2}} + \frac{gm_3(y_3 - y_1)}{((x_3 - x_1)^2 + (y_3 - y_1)^2)^{3/2}}. \end{aligned} \tag{6.45}$$

The second and third bodies, at (x_2, y_2) and (x_3, y_3), respectively, satisfy similar equations.

Computer Problems 9 and 10 ask the reader to computationally solve the two- and three-body problems. The latter problem illustrates severe sensitive dependence on initial conditions.

6.3 Exercises

1. Apply the Euler's Method with step size $h = 1/4$ to the initial value problem on $[0, 1]$.

(a) $\begin{cases} y_1' = y_1 + y_2 \\ y_2' = -y_1 + y_2 \\ y_1(0) = 1 \\ y_2(0) = 0 \end{cases}$ (b) $\begin{cases} y_1' = -y_1 - y_2 \\ y_2' = y_1 - y_2 \\ y_1(0) = 1 \\ y_2(0) = 0 \end{cases}$

(c) $\begin{cases} y_1' = -y_2 \\ y_2' = y_1 \\ y_1(0) = 1 \\ y_2(0) = 0 \end{cases}$ (d) $\begin{cases} y_1' = y_1 + 3y_2 \\ y_2' = 2y_1 + 2y_2 \\ y_1(0) = 5 \\ y_2(0) = 0 \end{cases}$

Find the global truncation errors of y_1 and y_2 at $t = 1$ by comparing with the correct solutions
(a) $y_1(t) = e^t \cos t$, $y_2(t) = -e^t \sin t$ (b) $y_1(t) = e^{-t} \cos t$, $y_2(t) = e^{-t} \sin t$
(c) $y_1(t) = \cos t$, $y_2(t) = \sin t$ (d) $y_1(t) = 3e^{-t} + 2e^{4t}$, $y_2(t) = -2e^{-t} + 2e^{4t}$.

2. Apply the Trapezoid Method with $h = 1/4$ to the initial value problems in Exercise 1. Find the global truncation error at $t = 1$ by comparing with the correct solutions.

3. Convert the higher-order ordinary differential equation to a first-order system of equations. (a) $y'' - ty = 0$ (Airy's equation) (b) $y'' - 2ty' + 2y = 0$ (Hermite's equation) (c) $y'' - ty' - y = 0$

4. Apply the Trapezoid Method with $h = 1/4$ to the initial value problems in Exercise 3, using $y(0) = y'(0) = 1$.

5. (a) Show that $y(t) = (e^t + e^{-t} - t^2)/2 - 1$ is the solution of the initial value problem $y''' - y' = t$, with $y(0) = y'(0) = y''(0) = 0$. (b) Convert the differential equation to a system of three first-order equations. (c) Use Euler's Method with step size $h = 1/4$ to approximate the solution on $[0, 1]$. (d) Find the global truncation error at $t = 1$.

6.3 Computer Problems

1. Apply Euler's Method with step sizes $h = 0.1$ and $h = 0.01$ to the initial value problems in Exercise 1. Plot the approximate solutions and the correct solution on $[0, 1]$, and find the global truncation error at $t = 1$. Is the reduction in error for $h = 0.01$ consistent with the order of Euler's Method?

2. Carry out Computer Problem 1 for the Trapezoid Method.

3. Adapt `pend.m` to model the damped pendulum. Run the resulting code with $d = 0.1$. Except for the initial condition $y_1(0) = \pi$, $y_2(0) = 0$, all trajectories move toward the straight-down position as time progresses. Check the exceptional initial condition: Does the simulation agree with theory? with a physical pendulum?

4. Adapt `pend.m` to build a forced, damped version of the pendulum. Run the Trapezoid Method in the following: (a) Set damping $d = 1$ and the forcing parameter $A = 10$. Set the step size $h = 0.005$ and the initial condition of your choice. After moving through some transient behavior, the pendulum will settle into a periodic (repeating) trajectory. Describe this trajectory qualitatively. Try different initial conditions. Do all solutions end up at the same "attracting" periodic trajectory? (b) Now increase the step size to $h = 0.01$, and repeat the experiment. Try initial condition $[\pi/2, 0]$ and others. Describe what happens, and give a reasonable explanation for the anomalous behavior at this step size.

5. Run the forced damped pendulum as in Computer Problem 4, but set $A = 12$. Use the Trapezoid Method with $h = 0.005$. There are now two periodic attractors that are mirror images of one another. Describe the two attracting trajectories, and find two initial conditions $(y_1, y_2) = (a, 0)$ and $(b, 0)$, where $|a - b| \leq 0.1$, that are attracted to different periodic trajectories. Set $A = 15$ to view chaotic motion of the forced damped pendulum.

6. Adapt `pend.m` to build a damped pendulum with oscillating pivot. The goal is to investigate the phenomenon of parametric resonance, by which the inverted pendulum becomes stable! The equation is

$$y'' + dy' + \left(\frac{g}{l} + A\cos 2\pi t\right)\sin y = 0,$$

where A is the forcing strength. Set $d = 0.1$ and the length of the pendulum to be 2.5 meters. In the absence of forcing $A = 0$, the downward pendulum $y = 0$ is a stable equilibrium, and the

inverted pendulum $y = \pi$ is an unstable equilibrium. Find as accurately as possible the range of parameter A for which the inverted pendulum becomes stable. (Of course, $A = 0$ is too small; it turns out that $A = 30$ is too large.) Use the initial condition $y = 3.1$ for your test, and call the inverted position "stable" if the pendulum does not pass through the downward position.

7. Use the parameter settings of Computer Problem 6 to demonstrate the other effect of parametric resonance: The stable equilibrium can become unstable with an oscillating pivot. Find the smallest (positive) value of the forcing strength A for which this happens. Classify the downward position as unstable if the pendulum eventually travels to the inverted position.

8. Adapt `pend.m` to build the double pendulum. A new pair of `rod` and `bob` must be defined for the second pendulum. Note that the pivot end of the second rod is equal to the formerly free end of the first rod: The (x, y) position of the free end of the second rod can be calculated by using simple trigonometry.

9. Adapt `orbit.m` to solve the two-body problem. Set the masses to $m_1 = 0.3$, $m_2 = 0.03$, and plot the trajectories with initial conditions $(x_1, y_1) = (2, 2)$, $(x_1', y_1') = (0.2, -0.2)$ and $(x_2, y_2) = (0, 0)$, $(x_2', y_2') = (-0.01, 0.01)$.

10. Adapt `orbit.m` to solve the three-body problem. Set the masses to $m_1 = 0.3$, $m_2 = m_3 = 0.03$. (a) Plot the trajectories with initial conditions $(x_1, y_1) = (2, 2)$, $(x_1', y_1') = (0.2, -0.2)$, $(x_2, y_2) = (0, 0)$, $(x_2', y_2') = (0, 0)$ and $(x_3, y_3) = (-2, -2)$, $(x_3', y_3') = (-0.2, 0.2)$. (b) Change the initial condition of x_1' to 0.20001, and compare the resulting trajectories. This is a striking visual example of sensitive dependence.

11. A remarkable three-body figure-eight orbit was discovered by C. Moore in 1993. In this configuration, three bodies of equal mass chase one another along a single figure-eight loop. Set the masses to $m_1 = m_2 = m_3 = 1$ and gravity $g = 1$. (a) Adapt `orbit.m` to plot the trajectory with initial conditions $(x_1, y_1) = (-0.970, 0.243)$, $(x_1', y_1') = (-0.466, -0.433)$, $(x_2, y_2) = (-x_1, -y_1)$, $(x_2', y_2') = (x_1', y_1')$ and $(x_3, y_3) = (0, 0)$, $(x_3', y_3') = (-2x_1', -2y_1')$. (b) Are the trajectories sensitive to small changes in initial conditions? Investigate the effect of changing x_3' by 10^{-k} for $1 \le k \le 5$. For each k, decide whether the figure-eight pattern persists, or a catastrophic change eventually occurs.

6.4 RUNGE–KUTTA METHODS AND APPLICATIONS

The Runge–Kutta Methods are a family of ODE solvers that include the Euler and Trapezoid Methods, and also more sophisticated methods of higher order. In this section, we introduce a variety of one-step methods and apply them to simulate trajectories of some key applications.

6.4.1 The Runge–Kutta family

We have seen that the Euler Method has order one and the Trapezoid Method has order two. In addition to the Trapezoid Method, there are other second-order methods of the Runge–Kutta type. One important example is the Midpoint Method.

Midpoint Method

$$w_0 = y_0$$
$$w_{i+1} = w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i)\right). \tag{6.46}$$

To verify the order of the Midpoint Method, we must compute its local truncation error. When we did this for the Trapezoid Method, we found the expression (6.31) useful:

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + \frac{\partial f}{\partial y}(t_i, y_i) f(t_i, y_i)\right) + \frac{h^3}{6} y'''(c). \quad (6.47)$$

To compute the local truncation error at step i, we assume that $w_i = y_i$ and calculate $y_{i+1} - w_{i+1}$. Repeating the use of the Taylor series expansion as for the Trapezoid Method, we can write

$$\begin{aligned} w_{i+1} &= y_i + hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, y_i)\right) \\ &= y_i + h\left(f(t_i, y_i) + \frac{h}{2}\frac{\partial f}{\partial t}(t_i, y_i) + \frac{h}{2} f(t_i, y_i)\frac{\partial f}{\partial y}(t_i, y_i) + O(h^2)\right). \end{aligned} \quad (6.48)$$

Comparing (6.47) and (6.48) yields

$$y_{i+1} - w_{i+1} = O(h^3),$$

so the Midpoint Method is of order two by Theorem 6.4.

Each function evaluation of the right-hand side of the differential equation is called a **stage** of the method. The Trapezoid and Midpoint Methods are members of the family of two-stage, second-order Runge–Kutta Methods, having form

$$w_{i+1} = w_i + h\left(1 - \frac{1}{2\alpha}\right) f(t_i, w_i) + \frac{h}{2\alpha} f(t_i + \alpha h, w_i + \alpha h f(t_i, w_i)) \quad (6.49)$$

for some $\alpha \neq 0$. Setting $\alpha = 1$ corresponds to the Explicit Trapezoid Method and $\alpha = 1/2$ to the Midpoint Method. Exercise 5 asks you to verify the order of methods in this family.

Figure 6.14 illustrates the intuition behind the Trapezoid and Midpoint Methods. The Trapezoid Method uses an Euler step to the right endpoint of the interval, evaluates the slope there, and then averages with the slope from the left endpoint. The Midpoint Method uses an Euler step to move to the midpoint of the interval, evaluates the slope there as $f(t_i + h/2, w_i + (h/2) f(t_i, w_i))$, and uses that slope to move from w_i to the new approximation w_{i+1}. These methods use different approaches to solving the same problem: acquiring a slope that represents the entire interval better than the Euler Method, which uses only the slope estimate from the left end of the interval.

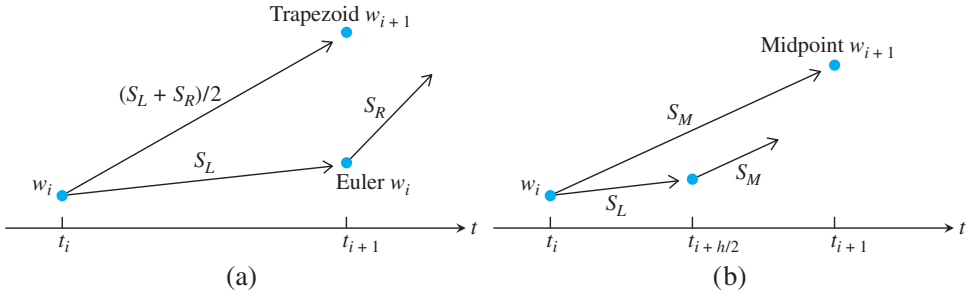

Figure 6.14 Schematic view of two members of the RK2 family. (a) The Trapezoid Method uses an average from the left and right endpoints to traverse the interval. (b) The Midpoint Method uses a slope from the interval midpoint.

SPOTLIGHT ON

Convergence The convergence properties of a fourth-order method, like RK4, are far superior to those of the order 1 and 2 methods we have discussed so far. Convergence here means how fast the (global) error of the ODE approximation at some fixed time t goes to zero as the step size h goes to zero. Fourth order means that for every halving of the step size, the error drops by approximately a factor of $2^4 = 16$, as is clear from Figure 6.15.

There are Runge–Kutta Methods of all orders. A particularly ubiquitous example is the method of fourth order.

Runge–Kutta Method of order four (RK4)

$$w_{i+1} = w_i + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4) \tag{6.50}$$

where

$$\begin{aligned} s_1 &= f(t_i, w_i) \\ s_2 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_1\right) \\ s_3 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_2\right) \\ s_4 &= f\left(t_i + h, w_i + hs_3\right). \end{aligned}$$

The popularity of this method stems from its simplicity and ease of programming. It is a one-step method, so that it requires only an initial condition to get started; yet, as a fourth-order method, it is considerably more accurate than either the Euler or Trapezoid Methods.

The quantity $h(s_1 + 2s_2 + 2s_3 + s_4)/6$ in the fourth-order Runge–Kutta Method takes the place of slope in the Euler Method. This quantity can be considered as an improved guess for the slope of the solution in the interval $[t_i, t_i + h]$. Note that s_1 is the slope at the left end of the interval, s_2 is the slope used in the Midpoint Method, s_3 is an improved slope at the midpoint, and s_4 is an approximate slope at the right-hand endpoint $t_i + h$. The algebra needed to prove that this method is order four is similar to our derivation of the Trapezoid and Midpoint Methods, but is a bit lengthy, and can be found, for example, in Henrici [1962]. We return one more time to differential equation (6.5) for purposes of comparison.

► EXAMPLE 6.18 Apply Runge–Kutta of order four to the initial value problem

$$\begin{cases} y' = ty + t^3 \\ y(0) = 1 \end{cases}. \tag{6.51}$$

Computing the global truncation error at $t = 1$ for a variety of step sizes gives the following table:

steps n	step size h	error at $t = 1$
5	0.20000	2.3788×10^{-5}
10	0.10000	1.4655×10^{-6}
20	0.05000	9.0354×10^{-8}
40	0.02500	5.5983×10^{-9}
80	0.01250	3.4820×10^{-10}
160	0.00625	2.1710×10^{-11}
320	0.00312	1.3491×10^{-12}
640	0.00156	7.2609×10^{-14}

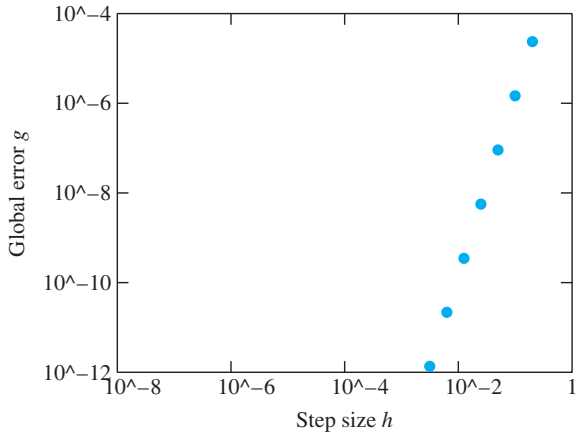

Figure 6.15 Error as a function of step size for Runge–Kutta of order 4. The difference between the approximate solution of (6.5) and the correct solution at $t = 1$ has slope 4 on a log–log plot, so is proportional to h^4, for small h.

Compare with the corresponding table for Euler's Method on page 286. The difference is remarkable and easily makes up for the extra complexity of RK4, which requires four function calls per step, compared with only one for Euler. Figure 6.15 displays the same information in a way that exhibits the fact that the global truncation error is proportional to h^4, as expected for a fourth-order method. ◀

6.4.2 Computer simulation: the Hodgkin–Huxley neuron

Computers were in their early development stages in the middle of the 20th century. Some of the first applications were to help solve hitherto intractable systems of differential equations.

A.L. Hodgkin and A.F. Huxley gave birth to the field of computational neuroscience by developing a realistic firing model for nerve cells, or neurons. They were able to approximate solutions of the differential equations model even with the rudimentary computers that existed at the time. For this work, they won the Nobel Prize in Biology in 1963.

The model is a system of four coupled differential equations, one of which models the voltage difference between the interior and exterior of the cell. The three other equations model activation levels of ion channels, which do the work of exchanging sodium and potassium ions between the inside and outside. The **Hodgkin–Huxley equations** are

$$\begin{aligned} Cv' &= -g_1 m^3 h(v - E_1) - g_2 n^4 (v - E_2) - g_3 (v - E_3) + I_{\text{in}} \\ m' &= (1 - m)\alpha_m (v - E_0) - m\beta_m (v - E_0) \\ n' &= (1 - n)\alpha_n (v - E_0) - n\beta_n (v - E_0) \\ h' &= (1 - h)\alpha_h (v - E_0) - h\beta_h (v - E_0), \end{aligned} \tag{6.52}$$

where

$$\alpha_m(v) = \frac{2.5 - 0.1v}{e^{2.5 - 0.1v} - 1},\ \beta_m(v) = 4e^{-v/18},$$

$$\alpha_n(v) = \frac{0.1 - 0.01v}{e^{1 - 0.1v} - 1},\ \beta_n(v) = \frac{1}{8} e^{-v/80},$$

and

$$\alpha_h(v) = 0.07e^{-v/20},\ \beta_h(v) = \frac{1}{e^{3 - 0.1v} + 1}.$$

The coefficient C denotes the capacitance of the cell, and I_{in} denotes the input current from other cells. Typical coefficient values are capacitance $C = 1$ microFarads, conductances $g_1 = 120$, $g_2 = 36$, $g_3 = 0.3$ siemens, and voltages $E_0 = -65$, $E_1 = 50$, $E_2 = -77$, $E_3 = -54.4$ millivolts.

The v' equation is an equation of current per unit area, in units of milliamperes/cm^2, while the three other activations m, n, and h are unitless. The coefficient C is the capacitance of the neuron membrane, g_1, g_2, g_3 are conductances, and E_1, E_2, and E_3 are the "reversal potentials," which are the voltage levels that form the boundary between currents flowing inward and outward.

Hodgkin and Huxley carefully chose the form of the equations to match experimental data, which was acquired from the giant axon of the squid. They also fit parameters to the model. Although the particulars of the squid axon differ from mammal neurons, the model has held up as a realistic depiction of neural dynamics. More generally, it is useful as an example of excitable media that translates continuous input into an all-or-nothing response. The MATLAB code implementing the model is as follows:

```
% Program 6.5 Hodgkin-Huxley equations
% Inputs: time interval inter,
% ic=initial voltage v and 3 gating variables, steps n
% Output: solution y
% Calls a one-step method such as rk4step.m
% Example usage: hh([0,100],[-65,0,0.3,0.6],2000);
function y=hh(inter,ic,n)
global pa pb pulse
inp=input('pulse start, end, muamps in [ ], e.g. [50 51 7]: ');
pa=inp(1);pb=inp(2);pulse=inp(3);
a=inter(1); b=inter(2); h=(b-a)/n; % plot n points in total
y(1,:)=ic;                         % enter initial conds in y
t(1)=a;
for i=1:n
  t(i+1)=t(i)+h;
  y(i+1,:)=rk4step(t(i),y(i,:),h);
end
subplot(3,1,1);
plot([a pa pa pb pb b],[0 0 pulse pulse 0 0]);
grid;axis([0 100 0 2*pulse])
ylabel('input pulse')
subplot(3,1,2);
plot(t,y(:,1));grid;axis([0 100 -100 100])
ylabel('voltage (mV)')
subplot(3,1,3);
plot(t,y(:,2),t,y(:,3),t,y(:,4));grid;axis([0 100 0 1])
ylabel('gating variables')
legend('m','n','h')
xlabel('time (msec)')

function y=rk4step(t,w,h)
%one step of the Runge-Kutta order 4 method
s1=ydot(t,w);
s2=ydot(t+h/2,w+h*s1/2);
s3=ydot(t+h/2,w+h*s2/2);
s4=ydot(t+h,w+h*s3);
y=w+h*(s1+2*s2+2*s3+s4)/6;

function z=ydot(t,w)
global pa pb pulse
```

```
c=1;g1=120;g2=36;g3=0.3;T=(pa+pb)/2;len=pb-pa;
e0=-65;e1=50;e2=-77;e3=-54.4;
in=pulse*(1-sign(abs(t-T)-len/2))/2;
% square pulse input on interval [pa,pb] of pulse muamps
v=w(1);m=w(2);n=w(3);h=w(4);
z=zeros(1,4);
z(1)=(in-g1*m*m*m*h*(v-e1)-g2*n*n*n*n*(v-e2)-g3*(v-e3))/c;
v=v-e0;
z(2)=(1-m)*(2.5-0.1*v)/(exp(2.5-0.1*v)-1)-m*4*exp(-v/18);
z(3)=(1-n)*(0.1-0.01*v)/(exp(1-0.1*v)-1)-n*0.125*exp(-v/80);
z(4)=(1-h)*0.07*exp(-v/20)-h/(exp(3-0.1*v)+1);
```

Without input, the Hodgkin–Huxley neuron stays quiescent, at a voltage of approximately E_0. Setting I_{in} to be a square current pulse of length 1 msec and strength 7 microamps is sufficient to cause a spike, a large depolarizing deflection of the voltage. This is illustrated in Figure 6.16. Run the program to check that 6.9 μA is not sufficient to cause a full spike. Hence, the all-or-nothing response. It is this property of greatly magnifying the effect of small differences in input that may explain the neuron's success at information processing. Figure 6.16(b) shows that if the input current is sustained, the neuron will fire a periodic volley of spikes. Computer Problem 10 is an investigation of the thresholding capabilities of this virtual neuron.

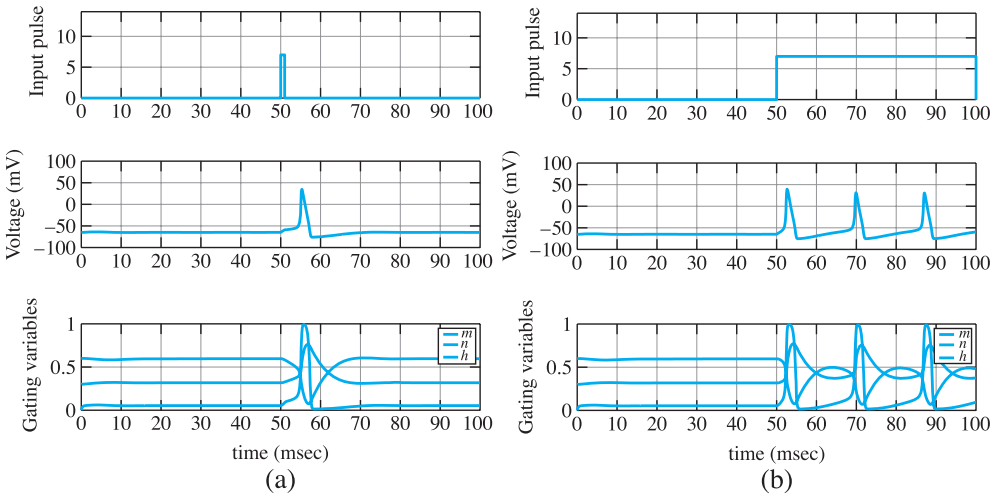

Figure 6.16 Screen shots of Hodgkin–Huxley program. (a) Square wave input of size $I_{\text{in}} = 7$ μA at time 50 msecs, 1 msec duration, causes the model neuron to fire once. (b) Sustained square wave, with $I_{\text{in}} = 7$ μA, causes the model neuron to fire periodically.

6.4.3 Computer simulation: the Lorenz equations

In the late 1950s, MIT meteorologist E. Lorenz acquired one of the first commercially available computers. It was the size of a refrigerator and operated at the speed of 60 multiplications per second. This unprecedented cache of personal computing power allowed him to develop and meaningfully evaluate weather models consisting of several differential equations that, like the Hodgkin–Huxley equations, could not be analytically solved.

The **Lorenz equations** are a simplification of a miniature atmosphere model that he designed to study Rayleigh-Bénard convection, the movement of heat in a fluid, such as air,

from a lower warm medium (such as the ground) to a higher cool medium (like the upper atmosphere). In this model of a two-dimensional atmosphere, a circulation of air develops that can be described by the following system of three equations:

$$\begin{aligned} x' &= -sx + sy \\ y' &= -xz + rx - y \\ z' &= xy - bz. \end{aligned} \tag{6.53}$$

The variable x denotes the clockwise circulation velocity, y measures the temperature difference between the ascending and descending columns of air, and z measures the deviation from a strictly linear temperature profile in the vertical direction. The Prandtl number s, the Rayleigh number r, and b are parameters of the system. The most common setting for the parameters is $s = 10, r = 28$, and $b = 8/3$. These settings were used for the trajectory shown in Figure 6.17, computed by order four Runge–Kutta, using the following code to describe the differential equation.

```
function z=ydot(t,y)
%Lorenz equations
s=10; r=28; b=8/3;
z(1)=-s*y(1)+s*y(2);
z(2)=-y(1)*y(3)+r*y(1)-y(2)
z(3)=y(1)*y(2)-b*y(3)
```

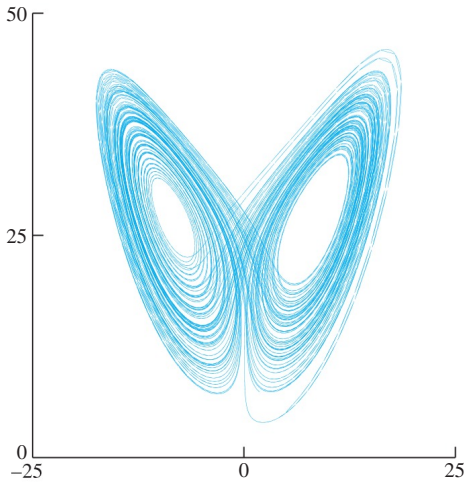

Figure 6.17 One trajectory of the Lorenz equations (6.53), projected to the *xz*-plane. Parameters are set to $s = 10, r = 28$, and $b = 8/3$.

The Lorenz equations are an important example because the trajectories show great complexity, despite the fact that the equations are deterministic and fairly simple (almost linear). The explanation for the complexity is similar to that of the double pendulum or three-body problem: sensitive dependence on initial conditions. Computer Problems 12 and 13 explore the sensitive dependence of this so-called chaotic attractor.

6.4 Exercises

1. Apply the Midpoint Method for the IVPs

   (a) $y' = t$ (b) $y' = t^2 y$ (c) $y' = 2(t+1)y$

   (d) $y' = 5t^4 y$ (e) $y' = 1/y^2$ (f) $y' = t^3/y^2$

with initial condition $y(0) = 1$. Using step size $h = 1/4$, calculate the Midpoint Method approximation on the interval $[0, 1]$. Compare with the correct solution found in Exercise 6.1.3, and find the global truncation error at $t = 1$.

2. Carry out the steps of Exercise 1 for the IVPs

$$\text{(a)} \quad y' = t + y \quad \text{(b)} \quad y' = t - y \quad \text{(c)} \quad y' = 4t - 2y$$

with initial condition $y(0) = 0$. The exact solutions were found in Exercise 6.1.4.

3. Apply fourth-order Runge–Kutta Method to the IVPs in Exercise 1. Using step size $h = 1/4$, calculate the approximation on the interval $[0, 1]$. Compare with the correct solution found in Exercise 6.1.3, and find the global truncation error at $t = 1$.

4. Carry out the steps of Exercise 3 for the IVPs in Exercise 2.

5. Prove that for any $\alpha \neq 0$, the method (6.49) is second order.

6. Consider the initial value problem $y' = \lambda y$. The solution is $y(t) = y_0 e^{\lambda t}$. (a) Calculate w_1 for RK4 in terms of w_0 for this differential equation. (b) Calculate the local truncation error by setting $w_0 = y_0 = 1$ and determining $y_1 - w_1$. Show that the local truncation error is of size $O(h^5)$, as expected for a fourth-order method.

7. Assume that the right-hand side $f(t, y) = f(t)$ does not depend on y. Show that $s_2 = s_3$ in fourth-order Runge–Kutta and that RK4 is equivalent to Simpson's Rule for the integral $\int_{t_i}^{t_i+h} f(s)\, ds$.

6.4 Computer Problems

1. Apply the Midpoint Method on a grid of step size $h = 0.1$ in $[0, 1]$ for the initial value problems in Exercise 1. Print a table of the t values, approximations, and global truncation error at each step.

2. Apply the fourth-order Runge–Kutta Method solution on a grid of step size $h = 0.1$ in $[0, 1]$ for the initial value problems in Exercise 1. Print a table of the t values, approximations, and global truncation error at each step.

3. Carry out the steps of Computer Problem 2, but plot the approximate solutions on $[0, 1]$ for step sizes $h = 0.1, 0.05$, and 0.025, along with the true solution.

4. Carry out the steps of Computer Problem 2 for the equations of Exercise 2.

5. Plot the fourth-order Runge–Kutta Method approximate solution on $[0, 1]$ for the differential equation $y' = 1 + y^2$ and initial condition (a) $y_0 = 0$ (b) $y_0 = 1$, along with the exact solution (see Exercise 6.1.7). Use step sizes $h = 0.1$ and 0.05.

6. Plot the fourth-order Runge–Kutta Method approximate solution on $[0, 1]$ for the differential equation $y' = 1 - y^2$ and initial condition (a) $y_0 = 0$ (b) $y_0 = -1/2$, along with the exact solution (see Exercise 6.1.8). Use step sizes $h = 0.1$ and 0.05.

7. Calculate the fourth-order Runge–Kutta Method approximate solution on $[0, 4]$ for the differential equation $y' = \sin y$ and initial condition (a) $y_0 = 0$ (b) $y_0 = 100$, using step sizes $h = 0.1 \times 2^{-k}$ for $0 \leq k \leq 5$. Plot the $k = 0$ and $k = 5$ approximate solutions along with the exact solution (see Exercise 6.1.15), and make a log–log plot of the error as a function of h.

8. Calculate the fourth-order Runge–Kutta Method approximate solution of the differential equation $y' = \sinh y$ and initial condition (a) $y_0 = 1/4$ on the interval $[0, 2]$ (b) $y_0 = 2$ on the interval $[0, 1/4]$, using step sizes $h = 0.1 \times 2^{-k}$ for $0 \le k \le 5$. Plot the $k = 0$ and $k = 5$ approximate solutions along with the exact solution (see Exercise 6.1.16), and make a log–log plot of the error as a function of h.

9. For the IVPs in Exercise 1, plot the global error of the RK4 method at $t = 1$ as a function of h, as in Figure 6.4.

10. Consider the Hodgkin–Huxley equations (6.52) with default parameters. (a) Find as accurately as possible the minimum threshold, in microamps, for generating a spike with a 1 msec pulse. (b) Does the answer change if the pulse is 5 msec long? (c) Experiment with the shape of the pulse. Does a triangular pulse of identical enclosed area cause the same effect as a square pulse? (d) Discuss the existence of a threshold for constant sustained input.

11. Adapt the `orbit.m` MATLAB program to animate a solution to the Lorenz equations by the order four Runge–Kutta Method with step size $h = 0.001$. Draw the trajectory with initial condition $(x_0, y_0, z_0) = (5, 5, 5)$.

12. Assess the conditioning of the Lorenz equations by following two trajectories from two nearby initial conditions. Consider the initial conditions $(x, y, z) = (5, 5, 5)$ and another initial condition at a distance $\Delta = 10^{-5}$ from the first. Compute both trajectories by fourth-order Runge–Kutta with step size $h = 0.001$, and calculate the error magnification factor after $t = 10$ and $t = 20$ time units.

13. Follow two trajectories of the Lorenz equations with nearby initial conditions, as in Computer Problem 12. For each, construct the binary symbol sequence consisting of 0 if the trajectory traverses the "negative x" loop in Figure 6.17 and 1 if it traverses the positive loop. For how many time units do the symbol sequences of the two trajectories agree?

Reality Check 6 The Tacoma Narrows Bridge

A mathematical model that attempts to capture the Tacoma Narrows Bridge incident was proposed by McKenna and Tuama [2001]. The goal is to explain how torsional, or twisting, oscillations can be magnified by forcing that is strictly vertical.

Consider a roadway of width $2l$ hanging between two suspended cables, as in Figure 6.18(a). We will consider a two-dimensional slice of the bridge, ignoring the dimension of the bridge's length for this model, since we are only interested in the side-to-side motion. At rest, the roadway hangs at a certain equilibrium height due to gravity; let y denote the current distance the center of the roadway hangs below this equilibrium.

Hooke's law postulates a linear response, meaning that the restoring force the cables apply will be proportional to the deviation. Let θ be the angle the roadway makes with the horizontal. There are two suspension cables, stretched $y - l\sin\theta$ and $y + l\sin\theta$ from equilibrium, respectively. Assume that a viscous damping term is given that is proportional to the velocity. Using Newton's law $F = ma$ and denoting Hooke's constant by K, the equations of motion for y and θ are as follows:

$$y'' = -dy' - \left[\frac{K}{m}(y - l\sin\theta) + \frac{K}{m}(y + l\sin\theta)\right]$$

$$\theta'' = -d\theta' + \frac{3\cos\theta}{l}\left[\frac{K}{m}(y - l\sin\theta) - \frac{K}{m}(y + l\sin\theta)\right].$$

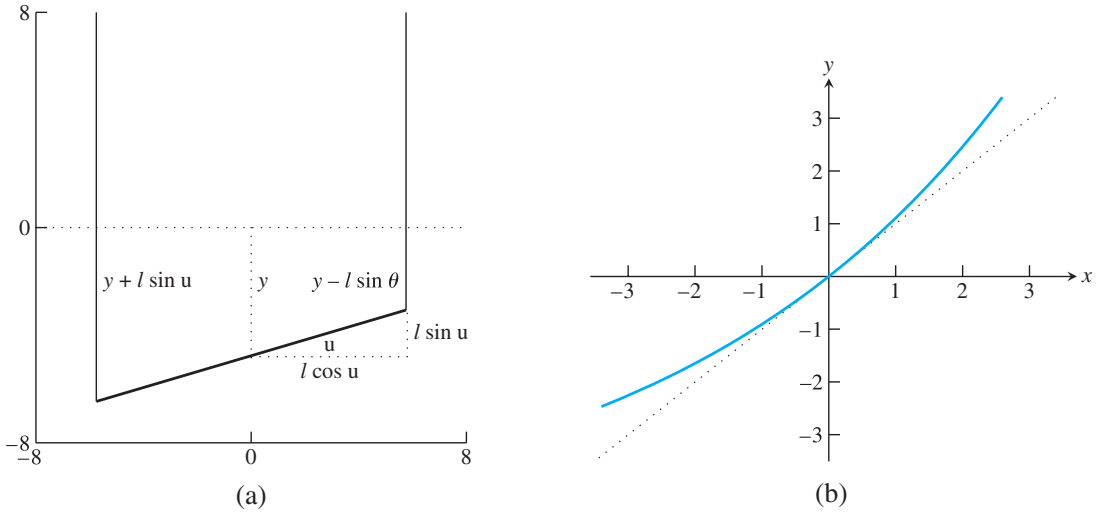

Figure 6.18 Schematics for the McKenna-Tuama model of the Tacoma Narrows Bridge. (a) Denote the distance from the roadway center of mass to its equilibrium position by y, and the angle of the roadway with the horizontal by θ. (b) Exponential Hooke's law curve $f(y) = (K/a)(e^{ay} - 1)$.

However, Hooke's law is designed for springs, where the restoring force is more or less equal whether the springs are compressed or stretched. McKenna and Tuama hypothesize that cables pull back with more force when stretched than they push back when compressed. (Think of a string as an extreme example.) They replace the linear Hooke's law restoring force $f(y) = Ky$ with a nonlinear force $f(y) = (K/a)(e^{ay} - 1)$, as shown in Figure 6.18(b). Both functions have the same slope K at $y = 0$; but for the nonlinear force, a positive y (stretched cable) causes a stronger restoring force than the corresponding negative y (slackened cable). Making this replacement in the preceding equations yields

$$y'' = -dy' - \frac{K}{ma}\left[e^{a(y-l\sin\theta)} - 1 + e^{a(y+l\sin\theta)} - 1\right]$$
$$\theta'' = -d\theta' + \frac{3\cos\theta}{l}\frac{K}{ma}\left[e^{a(y-l\sin\theta)} - e^{a(y+l\sin\theta)}\right]. \tag{6.54}$$

As the equations stand, the state $y = y' = \theta = \theta' = 0$ is an equilibrium. Now turn on the wind. Add the forcing term $0.2W \sin\omega t$ to the right-hand side of the y equation, where W is the wind speed in km/hr. This adds a strictly vertical oscillation to the bridge.

Useful estimates for the physical constants can be made. The mass of a one-foot length of roadway was about 2500 kg, and the spring constant K has been estimated at 1000 Newtons. The roadway was about 12 meters wide. For this simulation, the damping coefficient was set at $d = 0.01$, and the Hooke's nonlinearity coefficient $a = 0.2$. An observer counted 38 vertical oscillations of the bridge in one minute shortly before the collapse—set $\omega = 2\pi(38/60)$. These coefficients are only guesses, but they suffice to show ranges of motion that tend to match photographic evidence of the bridge's final oscillations. MATLAB code that runs the model (6.54) is as follows:

```
%Program 6.6 Animation program for bridge using IVP solver
%Inputs: time interval inter,
% ic=[y(1,1) y(1,2) y(1,3) y(1,4)],
% number of steps n, steps per point plotted p
%Calls a one-step method such as trapstep.m
%Example usage: tacoma([0 1000],[1 0 0.001 0],25000,5)
function tacoma(inter,ic,n,p)
```

```
clf                          % clear figure window
h=(inter(2)-inter(1))/n;
y(1,:)=ic;                   % enter initial conds in y
t(1)=inter(1);len=6;
set(gca,'XLim',[-8 8],'YLim',[-8 8], ...
   'XTick',[-8 0 8],'YTick',[-8 0 8], ...
   'Drawmode','fast','Visible','on','NextPlot','add');
cla;                         % clear screen
axis square                  % make aspect ratio 1-1
road=line('color','b','LineStyle','-','LineWidth',5,...
    'erase','xor','xdata',[],'ydata',[]);
lcable=line('color','r','LineStyle','-','LineWidth',1,...
    'erase','xor','xdata',[],'ydata',[]);
rcable=line('color','r','LineStyle','-','LineWidth',1,...
    'erase','xor','xdata',[],'ydata',[]);
for k=1:n
  for i=1:p
    t(i+1)=t(i)+h;
    y(i+1,:)=trapstep(t(i),y(i,:),h);
  end
  y(1,:)=y(p+1,:);t(1)=t(p+1);
  z1(k)=y(1,1);z3(k)=y(1,3);
  c=len*cos(y(1,3));s=len*sin(y(1,3));
  set(road,'xdata',[-c c],'ydata',[-s-y(1,1) s-y(1,1)])
  set(lcable,'xdata',[-c -c],'ydata',[-s-y(1,1) 8])
  set(rcable,'xdata',[c c],'ydata',[s-y(1,1) 8])
  drawnow; pause(h)
  end

function y=trapstep(t,x,h)
%one step of the Trapezoid Method
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function ydot=ydot(t,y)
len=6;a=0.2; W=80; omega=2*pi*38/60;
a1=exp(a*(y(1)-len*sin(y(3))));
a2=exp(a*(y(1)+len*sin(y(3))));
ydot(1)=y(2);
ydot(2)=-0.01*y(2)-0.4*(a1+a2-2)/a+0.2*W*sin(omega*t);
ydot(3)=y(4);
ydot(4)=-0.01*y(4)+1.2*cos(y(3))*(a1-a2)/(len*a);
```

Run `tacoma.m` with the default parameter values to see the phenomenon postulated earlier. If the angle θ of the roadway is set to any small nonzero value, vertical forcing causes θ to eventually grow to a macroscopic value, leading to significant torsion of the roadway. The interesting point is that there is no torsional forcing applied to the equation; the unstable "torsional mode" is excited completely by vertical forcing.

Suggested activities:

1. Run `tacoma.m` with wind speed $W = 80$ km/hr and initial conditions $y = y' = \theta' = 0$, $\theta = 0.001$. The bridge is stable in the torsional dimension if small disturbances in θ die out; unstable if they grow far beyond original size. Which occurs for this value of W?

2. Replace the trapezoid method by fourth-order Runge–Kutta to improve accuracy. Also, add new figure windows to plot $y(t)$ and $\theta(t)$.

3. The system is torsionally stable for $W = 50$ km/hr. Find the magnification factor for a small initial angle. That is, set $\theta(0) = 10^{-3}$ and find the ratio of the maximum angle $\theta(t)$, $0 \le t < \infty$, to $\theta(0)$. Is the magnification factor approximately consistent for initial angles $\theta(0) = 10^{-4}, 10^{-5}, \ldots$?

4. Find the minimum wind speed W for which a small disturbance $\theta(0) = 10^{-3}$ has a magnification factor of 100 or more. Can a consistent magnification factor be defined for this W?

5. Design and implement a method for computing the minimum wind speed in Step 4, to within 0.5×10^{-3} km/hr. You may want to use an equation solver from Chapter 1.

6. Try some larger values of W. Do all extremely small initial angles eventually grow to catastrophic size?

7. What is the effect of increasing the damping coefficient? Double the current value and compare the critical A when $\omega = 3$. Can you suggest possible changes in design that might have made the bridge less susceptible to torsion?

This project is an example of experimental mathematics. The equations are too difficult to derive closed-form solutions, and even too difficult to prove qualitative results about. Equipped with reliable ODE solvers, we can generate numerical trajectories for various parameter settings to illustrate the types of phenomena available to this model. Used in this way, differential equation models can predict behavior and shed light on mechanisms in scientific and engineering problems. ✔

6.5 VARIABLE STEP-SIZE METHODS

Up to this point, the step size h has been treated as a constant in the implementation of the ODE solver. However, there is no reason that h cannot be changed during the solution process. A good reason to want to change the step size is for a solution that moves between periods of slow change and periods of fast change. To make the fixed step size small enough to track the fast changes accurately may mean that the rest of the solution is solved intolerably slowly.

In this section, we discuss strategies for controlling the step size of ODE solvers. The most common approach uses two solvers of different orders, called embedded pairs.

6.5.1 Embedded Runge–Kutta pairs

The key idea of a variable step-size method is to monitor the error produced by the current step. The user sets an error tolerance that must be met by the current step. Then the method is designed to (1) reject the step and cut the step size if the error tolerance is exceeded, or (2) if the error tolerance is met, to accept the step and then choose a step size h that should be appropriate for the next step. The key need is for some way to approximate the error made on each step. First let's assume that we have found such a way and explain how to change the step size.

The simplest way to vary step size is to double or halve the step size, depending on the current error. Compare the error estimate e_i, or relative error estimate $e_i/|w_i|$, with the error