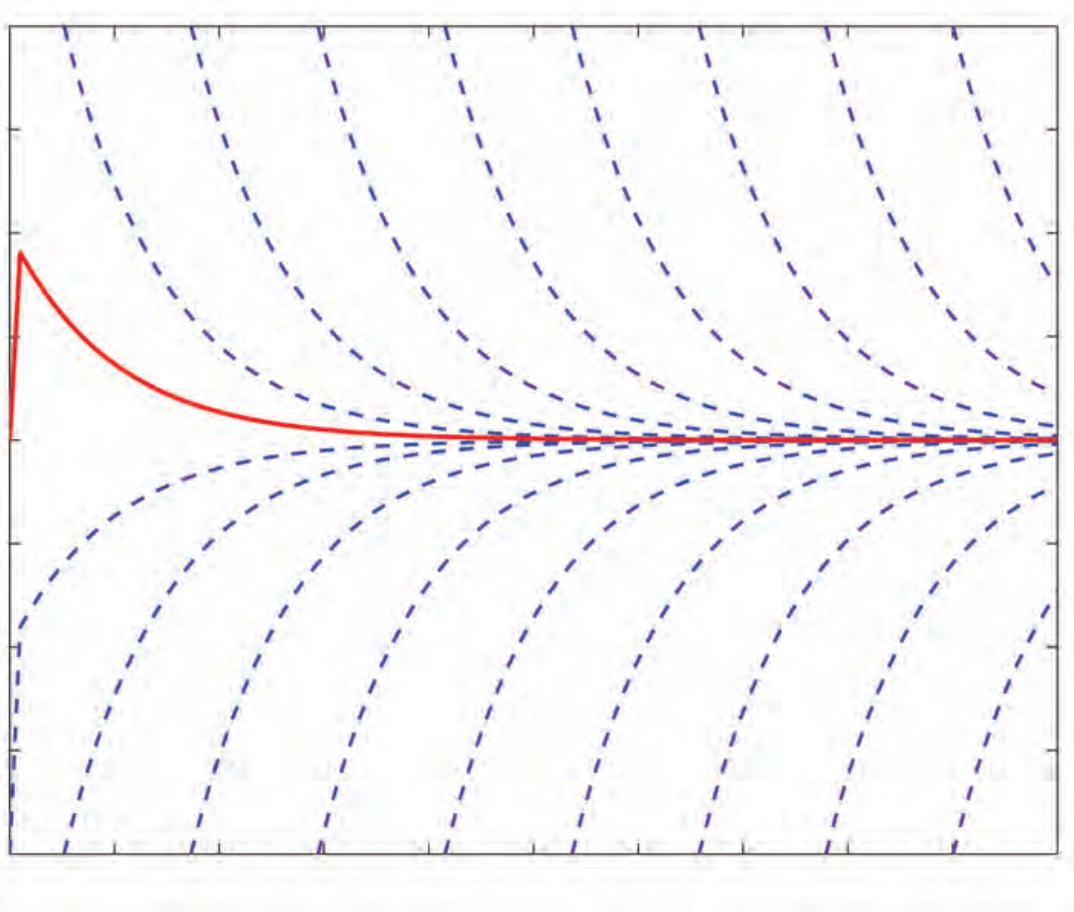
Unit V

Ordinary Differential Equations



In this unit we study initial value problems (IVPs) and boundary value problems (BVPs) for ordinary differential equations (ODEs).

We first review basic theory and algorithms for ODEs in Section 20.1, and then discuss Hamiltonian systems. Then we discuss some basics and some numerical methods for differential-algebraic equations (DAEs) in Section 20.4. Section 20.5 focuses on boundary value problems and their solution using shooting methods and finite difference methods. In Chapter 21, we revisit the problem of modeling the spread of an epidemic, considered in Chapter 19; this time we model the problem using differential equations. More experience with stability and control of ODEs is given in Chapter 22, where we study a differential equation modeling motion of a robot arm, investigating its positioning using a nonlinear equation, and the energy required for positioning using optimization. Finite element methods for BVPs are discussed in Chapter 23.

BASICS: To understand this unit, the following background is helpful:

- The mean value theorem and Taylor series from calculus.
- Basic facts about differential equations. This information can be found in a basic textbook on ODEs or in some specialized numerical textbooks such as [75] or [99].
- Numerical solution of ordinary differential equations by Euler's method, discussed in basic scientific computing textbooks.

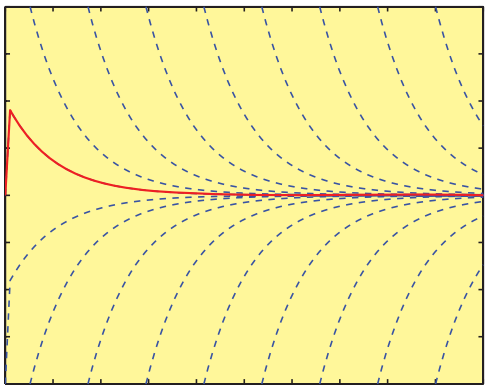
MASTERY: After you have worked through this unit, you should be able to do the following:

- Convert an ODE system to a "standard form" system.
- Use theorems to test whether there is a unique solution to an ODE.
- Determine whether an ODE is stable or stiff.
- Distinguish between local and global ODE error, estimate local error, and estimate a step length that keeps local error below a given tolerance.
- Derive the Euler method or the backward Euler method from Taylor series.
- Explain why the backward Euler method is sometimes more effective than the Euler method.
- Use Euler, backward Euler, or other Adams methods to solve differential equations.
- Plot ODE solutions vs. time, or make a phase plot of two components of a solution.
- Understand why predictor-corrector algorithms are used and be able to write one.
- Do order and stepsize control for an ODE method.
- Determine whether a numerical method for ODEs is stable.
- Know when to use the Gear methods instead of Adams or Runge–Kutta.

- Use a nonlinear equation solver to find a parameter in an ODE that satisfies a given condition.
- Define a Hamiltonian system and verify that a given system has a given Hamiltonian function.
- Incorporate a conservation principle into an ODE system by adding an extra variable.
- Put a DAE in standard form and use theorems to test whether a unique solution exists.
- Given an approximation to y' in terms of old function values, describe a numerical method for solving a DAE, using a finite difference formula to approximate y', and then using a nonlinear equation solver to compute y at the next timestep.
- Use theorems to test whether a unique solution to a BVP exists.
- Program a shooting method to solve a BVP.
- Formulate a delay differential equation.
- Use finite difference or finite elements to solve a BVP.
- Check Lyapunov stability of a system described by an ODE.
- Determine parameters in an ODE that minimize an energy function.

Chapter 20

Solution of Ordinary Differential Equations



In this chapter we study some algorithms for solving various types of ordinary differential equation (ODE) problems. We'll consider problems in which data values are known at a single initial time (Section 20.1) as well as those for which values are known at two different points in time or space (Section 20.5). In between, we'll discuss problems that have algebraic constraints (Section 20.4).

The goal is to understand the characteristics of the problems and the methods well enough to choose an appropriate method and assess the results it returns.

The problems we consider are these:

Initial Value Problem (IVP) in standard form: Given a function $\boldsymbol{f} : [0,T] \times \mathcal{R}^m \rightarrow \mathcal{R}^m$, and an $m \times 1$ vector of initial values $\boldsymbol{y}_0$, find a function $\boldsymbol{y} : [0,T] \rightarrow \mathcal{R}^m$ satisfying

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \tag{20.1}$$

$$\boldsymbol{y}(0) = \boldsymbol{y}_0 . \tag{20.2}$$

Higher-Order Initial Value Problem: Given a function $\boldsymbol{g} : [0,T] \times \mathcal{R}^{Km} \rightarrow \mathcal{R}^m$, and $m \times K$ initial values $\boldsymbol{u}_0$, find a function $\boldsymbol{u} : [0,T] \rightarrow \mathcal{R}^m$ satisfying

$$\boldsymbol{u}^{(K)}(t) = \boldsymbol{g}(t, \boldsymbol{u}(t), \boldsymbol{u}'(t), \ldots, \boldsymbol{u}^{(K-1)}(t)),$$

$$\boldsymbol{u}^{(j)}(0) = \boldsymbol{u}_0(:, j), \qquad j = 0, \ldots, K-1,$$

where $\boldsymbol{u}^{(j)}$ denotes the jth derivative of $\boldsymbol{u}$.

Boundary Value Problem (BVP): Given a function $\boldsymbol{f} : [0,T] \times \mathcal{R}^m \times \mathcal{R}^m \rightarrow \mathcal{R}^m$, and two $m \times 1$ vectors of values $\boldsymbol{u}_0$ and $\boldsymbol{u}_T$, find a function $\boldsymbol{u} : [0,T] \rightarrow \mathcal{R}^m$ satisfying

$$\boldsymbol{u}''(t) = \boldsymbol{f}(t, \boldsymbol{u}, \boldsymbol{u}'),$$

$$\boldsymbol{u}(0) = \boldsymbol{u}_0 ,$$

$$\boldsymbol{u}(T) = \boldsymbol{u}_T .$$

POINTER 20.1. Existence and Uniqueness of Solutions to IVPs for ODEs.

Before computing a numerical approximation to the solution to our ODE, it is important to know that such a solution exists! One condition guaranteeing existence and uniqueness of a continuous and differentiable solution on the interval $[0,a]$ is that the function $\boldsymbol{f}$ be **Lipschitz continuous**, meaning that there exists a constant L so that for all points t in $[0,a]$ and for all points $\boldsymbol{y}$ and $\widehat{\boldsymbol{y}}$ we have the bound

$$\|\boldsymbol{f}(t,\boldsymbol{y})-\boldsymbol{f}(t,\widehat{\boldsymbol{y}})\| \leq L\|\boldsymbol{y}-\widehat{\boldsymbol{y}}\|.$$

See standard textbooks such as [26, p. 251] for this and other existence results.

Differential-Algebraic Equation (DAE) in standard form: Given functions $\boldsymbol{M} : [0,T] \to \mathcal{R}^{m\times m}$, $\boldsymbol{A} : [0,T] \to \mathcal{R}^{m\times m}$, $\boldsymbol{f} : [0,T] \to \mathcal{R}^m$, and appropriate initial values $\boldsymbol{y}(0)$, find a function $\boldsymbol{y} : [0,T] \to \mathcal{R}^m$ satisfying

$$\boldsymbol{M}(t)\boldsymbol{y}'(t) = \boldsymbol{A}(t)\boldsymbol{y}(t)+\boldsymbol{f}(t).$$

20.1 Initial Value Problems for Ordinary Differential Equations

In this section, we review some standard ideas in the numerical solution of ODEs and discuss Hamiltonian systems.

20.1.1 Standard Form

We only work with initial value problems in **standard form**, as in (20.1), because that is what most software needs. Note that $\boldsymbol{y}'$ means the derivative with respect to t. In order to make sure there is no confusion between components of a vector and partial derivatives, in this chapter we denote the components of the vector $\boldsymbol{y}(t)$ by $y_{(j)}(t)$, so that writing our equations component by component yields

$$\begin{aligned} y'_{(1)}(t) &= f_1(t, y_{(1)}(t), \ldots, y_{(m)}(t)), \\ &\vdots \\ y'_{(m)}(t) &= f_m(t, y_{(1)}(t), \ldots, y_{(m)}(t)), \end{aligned}$$

with $y_{(1)}(0), \ldots, y_{(m)}(0)$ given numbers.

It is not essential that we start at $t = 0$; any value is fine.

One famous system of ODEs is **Volterra's predator/prey model**. We can think of this as modeling a population of foxes, who eat rabbits, coexisting with a population of

rabbits with an infinite food supply. If we denote the population of rabbits at time t by $r(t)$ and the population of foxes by $f(t)$, then the model is

$$\frac{dr(t)}{dt} = 2r(t) - \alpha r(t) f(t),$$

$$\frac{df(t)}{dt} = -f(t) + \alpha r(t) f(t),$$

$$r(0) = r_0,$$
$$f(0) = f_0.$$

The parameter α is the **encounter factor**, with $\alpha = 0$ meaning no interaction between rabbits and foxes.

Let $y_{(1)}(t) = r(t)$ and $y_{(2)}(t) = f(t)$. Then we can write this ODE system in standard form:

$$y'_{(1)}(t) = 2y_{(1)}(t) - \alpha y_{(1)}(t) y_{(2)}(t),$$
$$y'_{(2)}(t) = -y_{(2)}(t) + \alpha y_{(1)}(t) y_{(2)}(t).$$

The solution is not unique until we specify α and the initial population of rabbits and foxes.

CHALLENGE 20.1.

(a) Use one of MATLAB's solvers (See Pointer 20.2) to explore the behavior of the Volterra model for various values of α. Suppose that initially the population of rabbits is 20 and the population of foxes is 10. Graph the solutions for $0 \le t \le 2$ for $\alpha = 10^{-2}, 10^{-1}$, and 1.

(b) Suppose we want to find a value of α so that the final population of rabbits is 4:

$$r_\alpha(2) - 4 = 0.$$

Note that the result of our experiment in (a) tells us that there is such a value of α between 0.01 and 1. Write a program that uses `ode45` along with the nonlinear equation solver `fzero` to find α.

(c) Repeat part (b), using the "Events" option in the solver to find α.

Higher-order initial value problems, those involving derivatives of order greater than first, can be converted to standard form. For example, suppose

$$u''(t) = g(t, u, u'),$$

where g is a given function. Let $y_{(1)}(t) = u(t)$ and $y_{(2)}(t) = u'(t)$. Then

$$u'(t) = y'_{(1)}(t) = y_{(2)}(t),$$
$$u''(t) = y'_{(2)}(t) = g(t, y_{(1)}, y_{(2)}),$$

and we have a system in standard form.

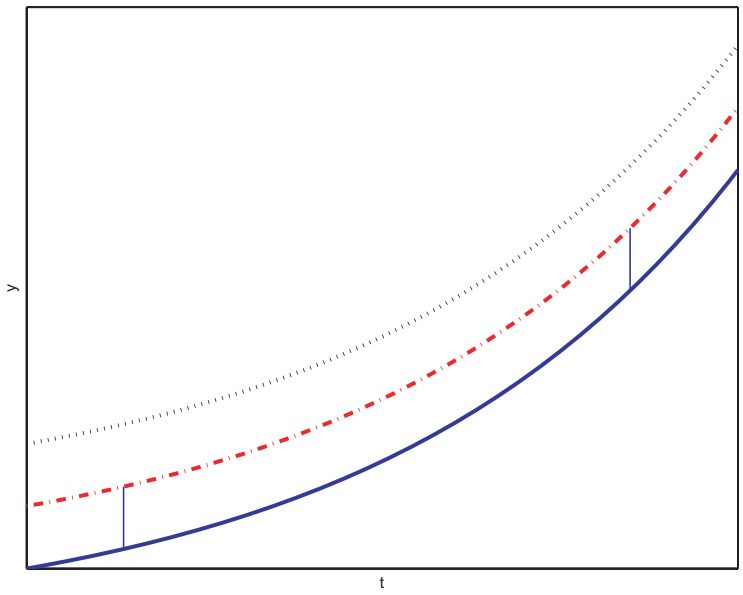


Figure 20.1. *Parallel solutions to an* ODE. *This graph shows three solutions to the same* ODE *but with different initial values* y_0. *This* ODE *is on the stability boundary.*

POINTER 20.2. ODE **Solvers in MATLAB.**
MATLAB has several solvers for differential equations, including:

`ode23`: Solves a nonstiff ODE using a low-order method.
`ode45`: Solves a nonstiff ODE using a medium-order method.
`ode23s`: Solves a stiff ODE using a low-order method.

The ODE solvers can do more complicated things, too; read the documentation carefully.

20.1.2 Solution Families and Stability

Given $\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t))$, the **family of solutions** is the set of all functions $\boldsymbol{y}$ that satisfy this equation.

Let's consider three examples:

	ODE	Solution	Jacobian matrix	Illustration
Example 1.	$y'(t) = e^t$	$y(t) = c - e^t$	$J(t) = 0$	Figure 20.1
Example 2.	$y'(t) = -y(t)$	$y(t) = ce^{-t}$	$J(t) = -1$	Figure 20.2
Example 3.	$y'(t) = y(t)$	$y(t) = ce^t$	$J(t) = 1$	Figure 20.3

In the table, c is an arbitrary constant, determined by the initial conditions, and the **Jacobian matrix** $\boldsymbol{J}$ is an $m \times m$ matrix with elements

$$J_{kj}(t, \boldsymbol{y}(t)) = \frac{\partial f_k(t, \boldsymbol{y})(t)}{\partial y_j}, \qquad k = 1, \ldots, m, \;\; j = 1, \ldots, m.$$

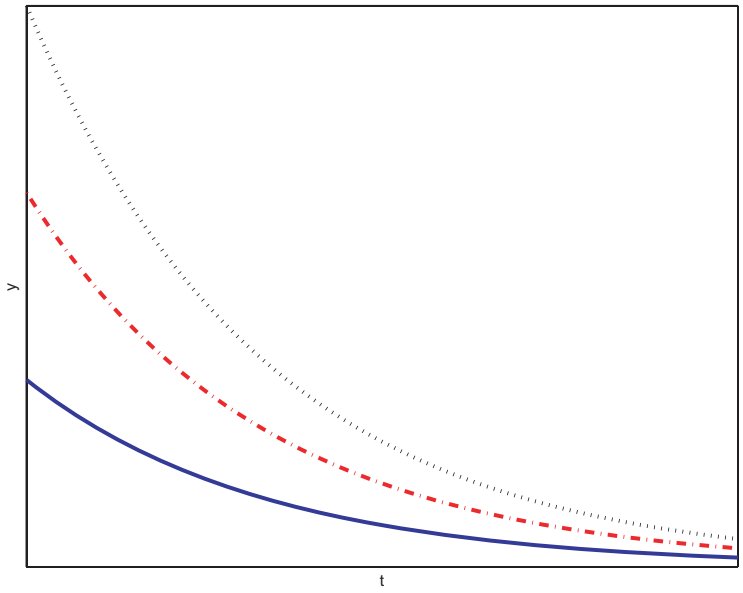


Figure 20.2. ODE *solutions that get closer to each other over time. The* ODE *is stable.*

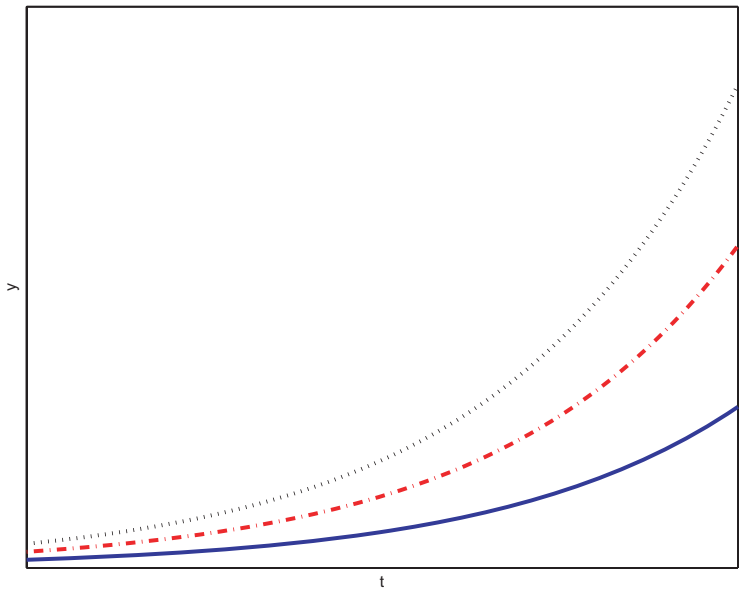


Figure 20.3. ODE *solutions that get further from each other. This* ODE *is unstable.*

For example, if

$$\boldsymbol{y}'(t) = \begin{bmatrix} 2y_{(1)}(t) + 3y_{(2)}^2(t) \\ -6t + 7y_{(2)}(t) \end{bmatrix},$$

then

$$\boldsymbol{J}(t,\boldsymbol{y}) = \begin{bmatrix} 2 & 6y_{(2)}(t) \\ 0 & 7 \end{bmatrix}.$$

The Jacobian matrix of a single ODE is 1×1, hence scalar, as shown in the table above. In this case,

$$J(t,y) = f_y(t,y(t)) = \partial f(t,y)/\partial y.$$

We say that a single ODE is

- **stable** at a point $(\hat{t}, \hat{y}(\hat{t}))$ if $f_y(\hat{t}, \hat{y}(\hat{t})) < 0$,
- **unstable** at a point $(\hat{t}, \hat{y}(\hat{t}))$ if $f_y(\hat{t}, \hat{y}(\hat{t})) > 0$,
- **stiff** at a point $(\hat{t}, \hat{y}(\hat{t}))$ if $f_y(\hat{t}, \hat{y}(\hat{t})) << 0$.

A system of ODEs is

- **stable** at a point $(\hat{t}, \hat{\boldsymbol{y}}(\hat{t}))$ if the real parts of all the eigenvalues of the matrix $\boldsymbol{J}(\hat{t}, \hat{\boldsymbol{y}}(\hat{t}))$ are negative,
- **stiff** at a point $(\hat{t}, \hat{\boldsymbol{y}}(\hat{t}))$ if the real parts of more than one eigenvalue of $\boldsymbol{J}(\hat{t}, \hat{\boldsymbol{y}}(\hat{t}))$ are negative and wildly different. In this case, the solution to the IVP is determined by processes occurring on time scales that are radically different from each other, and this forces numerical methods to take small timesteps in order to follow the solution. Stiff problems are common, for example, in chemical reactions and weather prediction.

Thus, stability is determined by the Jacobian matrix, which can change over time, so a system can be stable for one time and unstable for another.

CHALLENGE 20.2.

Suppose we have a system of differential equations $\boldsymbol{y}'(t) = \boldsymbol{f}(t,\boldsymbol{y}(t))$ with 3 components and a Jacobian matrix $\boldsymbol{J}(t,\boldsymbol{y}(t))$ with eigenvalues $4 - t^2$, $-t - it$, and $-t + it$, where $i = \sqrt{-1}$. For what values of t is the equation stable?

To illustrate the effects of stability, consider

$$\boldsymbol{y}'(t) = \boldsymbol{A}\boldsymbol{y}(t) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \boldsymbol{y}(t).$$

The solution is

$$\boldsymbol{y}(t) = b_1 e^{\lambda_1 t} \boldsymbol{z}_1 + b_2 e^{\lambda_2 t} \boldsymbol{z}_2,$$

where λ_j and $\boldsymbol{z}_j$ are eigenvalues and eigenvectors of $\boldsymbol{A}$:

$$\boldsymbol{A}\boldsymbol{z}_1 = \lambda_1 \boldsymbol{z}_1,$$
$$\boldsymbol{A}\boldsymbol{z}_2 = \lambda_2 \boldsymbol{z}_2.$$

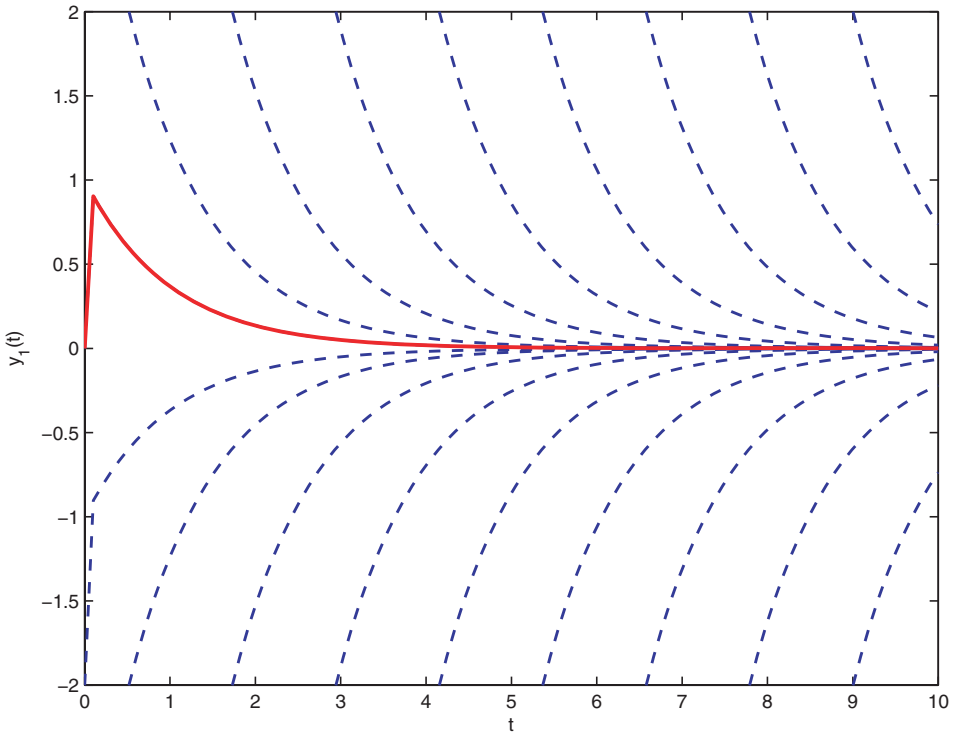


Figure 20.4. *Solution to the example stiff* ODE *problem. The red curve is the desired solution, and the blue dashed curves are in the same solution family; they solve the same* ODE *with different initial values.*

Let

$$\boldsymbol{A} = \begin{bmatrix} -1001 & 999 \\ 999 & -1001 \end{bmatrix}.$$

Then $\boldsymbol{y}'(t) = \boldsymbol{A}\boldsymbol{y}(t)$ has the Jacobian $\boldsymbol{J} = \boldsymbol{A}$ with eigenvalues $\lambda = -2, -2000$, so this system is stiff. Why does this trouble us? The solution is

$$\boldsymbol{y}(t) = b_1 e^{-2t} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + b_2 e^{-2000t} \begin{bmatrix} -1 \\ 1 \end{bmatrix},$$

where b_1 and b_2 are constants determined from the initial conditions. If $y_{(1)}(0) = 0$ and $y_{(2)}(0) = 2$, then

$$b_1 = 1,\ b_2 = 1,$$

and the solution is shown in Figure 20.4. Notice that even after our solution is nearly constant, other solutions in the family are changing rapidly. This causes trouble for numerical methods since they introduce small perturbations in the solution and may jump from one member of the solution family to another, as we will see in Figure 20.7. We can cope with this difficulty of rapidly changing solutions by using an ODE solver designed to handle stiff systems.

Note that stability for an ODE is the same as conditioning. Consider, for example, the stable system $y'(t) = -y(t)$ with $y_0 = 1$. Then the solution is $y(t) = e^{-t}$. A small perturbation in initial condition to $y_0 = 1 + \epsilon$ changes the solution to $y(t) = (1 + \epsilon)e^{-t}$,

POINTER 20.3. Stability vs. Conditioning.

The terminology used here differs somewhat from that discussed in Chapter 1. Usually we say that a problem is well-conditioned, not stable, if small changes in data lead to small changes in the result. We use the term stability to refer to algorithms. Since the use of the term stability in ODEs is older than the term conditioning, the standard terminology for ODEs violates our convention.

Thus, stability of solution families is different from stability of numerical algorithms used to solve them. Note that an algorithm that solves a well-conditioned (stable) problem may or may not be numerically stable.

---

so the problem is well-conditioned. On the other hand, the unstable system $y'(t) = y(t)$ with $y_0 = 1$ has the exact solution $y(t) = e^t$. A small perturbation to $y_0 = 1 + \epsilon$ changes the solution to $y(t) = (1 + \epsilon)e^t$. Thus an arbitrarily small change in input leads to a very different result for large t, and the problem is ill-conditioned. We'll consider other ways to measure stability in the case study of Chapter 22. Meanwhile, we investigate how a basic solution algorithm, Euler's method, works on stable and unstable problems.

20.2 Methods for Solving IVPs for ODEs

Now that we understand the characteristics of solutions to IVPs for ODEs, we introduce some methods for their numerical solution.

20.2.1 Euler's Method, Stability, and Error

We use Euler's method to illustrate the basic ideas behind numerical methods for ODEs. It is not a practical method because it is too slow, but it helps us understand the ideas. Three derivations of the method yield insight. Suppose that for some value t_n, we know the values $y_n = y(t_n)$ and $f_n = f(t_n, y_n)$. (Initially, $n = 0$, $t_0 = 0$, and we have the required data.) We use this information to step to a new point $t_{n+1} = t_n + h$.

Approach 1: Geometry

Euler's method can be derived from the left illustration in Figure 20.5. We have a function value y_n and a derivative value $f_n = f(t_n, y_n)$ at t_n. From these we obtain an approximation y_{n+1} by starting at y_n and moving along a line with a slope of f_n to $t = t_{n+1}$.

Approach 2: Polynomial interpolation

Euler's method can also be derived by determining the linear polynomial that matches the function value and derivative of y at t_n, and then evaluating it at $t = t_{n+1}$.

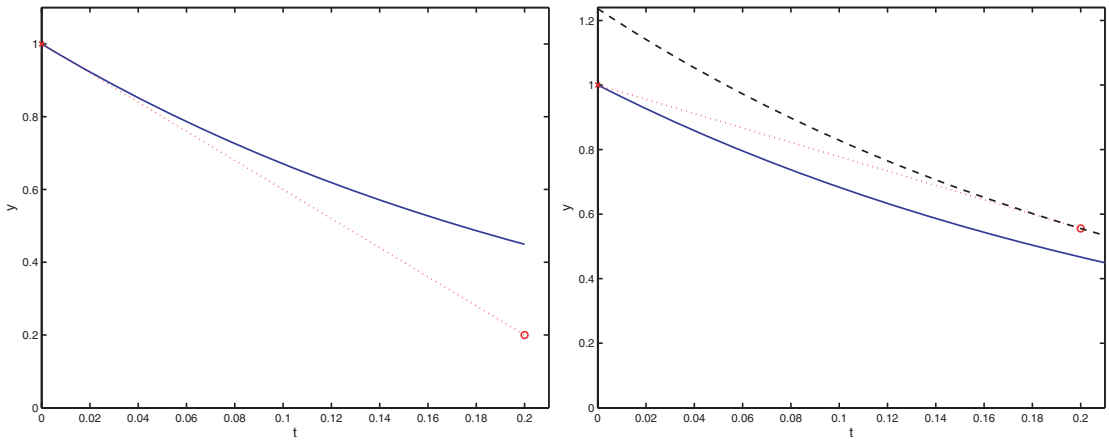


Figure 20.5. *Euler's method (left) and the backward Euler method (right) for solving an* ODE. *On the left, the blue curve is the true solution. An Euler step from $t = 0$, $y(0) = 1$ with stepsize $h = 0.2$ steps to the point marked with a circle. The red dotted line is tangent to the solution curve at $t = 0$, $y(0) = 1$, so its slope is $f(0, y(0))$. On the right, a backward Euler step from $t = 0$, $y(0) = 1$ with stepsize $h = 0.2$ steps to the point marked with a circle. The red dotted line passes through the point $t = 0$, $y(0) = 1$ and is tangent at $t = h$ to a solution curve (black dashed) for the same differential equation but with different initial condition.*

CHALLENGE 20.3.

Find the linear polynomial that passes through the point data (t_n, y_n) with slope f_n, and then evaluate it at t_{n+1}.

Approach 3: Taylor series

From Taylor series we know that if y is smooth enough, then

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2} y''(\xi)$$

for some point $t \le \xi \le t+h$. Since

$$y'(t) = f(t, y(t)),$$

this gives the approximation

$$y_{n+1} = y_n + hf_n,$$

where $h = t_{n+1} - t_n$. The difference between $y(t+h)$ and y_{n+1} is $O(h^2)$. It takes $1/h$ steps of Euler's method with stepsize h to walk from t to $t+1$, so the error per unit time is $O(h^1)$. Therefore, we say that Euler is a **first-order method**.[9]

However we derive it, we define Euler's method in Algorithm 20.1.

[9] The order of a method is always one less than the exponent of h in the error formula for a single step.

Algorithm 20.1 Euler's Method

Given: $\boldsymbol{y}_0$ and $t_0, t_1, \ldots, t_N$.
for $n = 0, \ldots, N-1$,
  $h_n = t_{n+1} - t_n$.
  $\boldsymbol{f}_n = \boldsymbol{f}(t_n, \boldsymbol{y}_n)$.
  $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n \boldsymbol{f}_n$.
end

CHALLENGE 20.4.
Apply Euler's method to $y'(t) = 1$, $y(0) = 0$, using a stepsize of $h = .1$ to obtain approximate values for $y(0.1), y(0.2), \ldots, y(1.0)$.

Stability of Euler's method and the backward Euler method

In using a method like Euler's for solving ODEs, there are three sources of error:

- rounding error: especially if the steps get too small.
- local error: the error introduced assuming that y_n is the true value.
- global error: how far we have strayed from our original solution curve, assuming no rounding error.

Consider a single ODE. For Euler's method, Taylor series (Approach 3) tells us that if y_n is correct, then

$$y(t_{n+1}) - y_{n+1} = \frac{h_n^2}{2} y''(\xi),$$

where $h_n = t_{n+1} - t_n$. Therefore the local error is $\frac{h_n^2}{2} y''(\xi)$, which converges to zero as $h_n \to 0$.

But as we iterate with Euler's method, y_n has some error, so from the relations

$$y_{n+1} = y_n + h_n f(t_n, y_n)$$

and

$$y(t_{n+1}) = y(t_n) + h_n f(t_n, y(t_n)) + \frac{h_n^2}{2} y''(\xi_n),$$

we obtain the error formula

$$y(t_{n+1}) - y_{n+1} = y(t_n) - y_n + h_n(f(t_n, y(t_n)) - f(t_n, y_n)) + \frac{h_n^2}{2} y''(\xi_n).$$

In this expression, the plum-colored terms are global errors at times t_n and t_{n+1}, and the red term is the local error. See Figure 20.6 for an illustration of local and global errors in an ODE.

Using the mean value theorem, the blue term can be written as

$$h_n(f(t_n, y(t_n)) - f(t_n, y_n)) = h_n f_y(\eta)(y(t_n) - y_n),$$

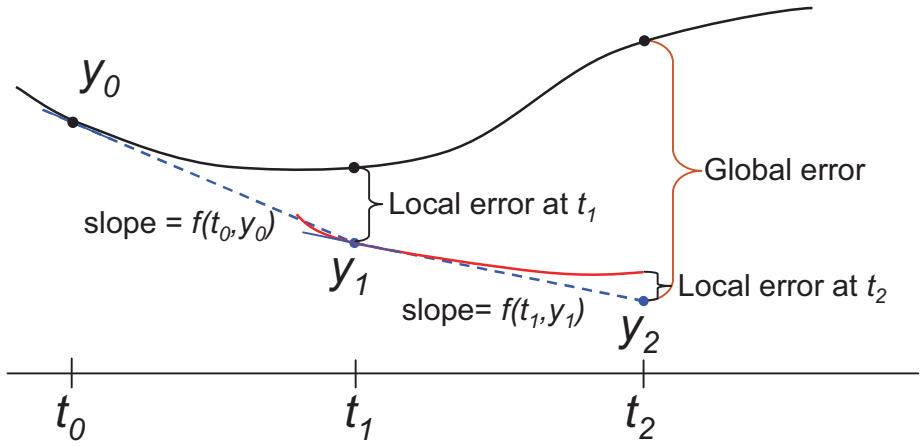


Figure 20.6. *Global errors accumulate. If we start on the solution curve at y_0 and take an Euler step, we obtain the value y_1, which is on a different curve in the solution family, and the local error is proportional to the stepsize squared. A second step takes us to y_2, which is on yet another solution curve. The error in moving from y_1 to y_2 is still proportional to the stepsize squared, but the global error can be much larger than the sum of the two local errors.*

where η is some point between t_n and t_{n+1}. Thus we have the expression

$$(\text{global error})_{n+1} = (1 + h_n f_y(\eta))\ (\text{global error})_n\ + (\text{local error})_n.$$

Therefore, the global errors are magnified if

$$|1 + h_n f_y(\eta)| > 1,$$

and we say that the Euler method is **unstable** in this case. The **stability interval for Euler's method**, the values of h_n for which the errors are not magnified at a given value of t, is defined by

$$-2 < h_n f_y(t) < 0$$

for a single equation. For a system of equations, the stability region contains the values of h_n for which the eigenvalues of $\boldsymbol{I} + h_n \boldsymbol{J}(t, y(t))$ are in the unit circle.

In contrast, in the method called the **backward Euler method** we define $\boldsymbol{y}_{n+1}$ as the solution to the equation

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}).$$

The geometry of the method is illustrated on the right in Figure 20.5, and we try the algorithm, summarized in Algorithm 20.2, in the next challenge.

Algorithm 20.2 The Backward Euler Method

Given: $\boldsymbol{y}_0$ and $t_0, t_1, \ldots t_N$.
for $n = 0, \ldots, N-1$
  Let $h_n = t_{n+1} - t_n$.
  Solve for $\boldsymbol{y}_{n+1}$ in the nonlinear equation

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n f(\boldsymbol{t}_{n+1}, \boldsymbol{y}_{n+1}).$$

end

CHALLENGE 20.5.
Let $y'(t) = -y(t)$. Show that the backward Euler method computes

$$y_{n+1} = y_n - h_n y_{n+1}.$$

Solve this equation for y_{n+1} and compute several iterates for $y(0) = 1$ and $h_n = 0.1$.

To determine the error and stability for the backward Euler method, note that Taylor series tells us that

$$\boldsymbol{y}(t) = \boldsymbol{y}(t+h) - h\boldsymbol{y}'(t+h) + \frac{h^2}{2}\boldsymbol{y}''(\xi),$$

where $\xi \in [t, t+h]$. Thus Taylor series says that the local error is $\frac{h^2}{2}\boldsymbol{y}''(\xi)$. This is a first-order method, just like Euler's method.

To determine the stability of the backward Euler method we again consider an ODE with a single equation. We know that

$$y(t_{n+1}) = y(t_n) + h_n f(t_{n+1}, y(t_{n+1})) + \frac{h_n^2}{2} y''(\xi_n), \qquad n = 0, 1, 2, \ldots,$$

and

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1}).$$

Therefore,

$$y(t_{n+1}) - y_{n+1} = y(t_n) - y_n + h_n(f(t_{n+1}, y(t_{n+1})) - f(t_{n+1}, y_{n+1})) + \frac{h_n^2}{2} y''(\xi_n).$$

Again, the plum-colored terms are global errors, and the red term is the local error. Using the mean value theorem, the blue term can be written as

$$h_n(f(t_{n+1}, y(t_{n+1})) - f(t_{n+1}, y_{n+1})) = h_n f_y(\eta)(y(t_{n+1}) - y_{n+1}).$$

Thus we have the expression

$$(1 - h_n f_y(\eta))\ (\text{global error})_{n+1} = (\text{global error})_n + (\text{local error})_n.$$

Therefore, the global errors are magnified if

$$|1 - h_n f_y(\eta)| < 1,$$

and we say that the backward Euler method is unstable in this case. The stability interval for the backward Euler method is

$$hf_y(t) < 0 \text{ or } hf_y(t) > 2$$

for a single equation. For a system of equations, the stability region is the region where all eigenvalues of $\boldsymbol{I} - h_n\boldsymbol{J}(t, \boldsymbol{y}(t))$ are outside the unit circle. For example, backward Euler is stable on the equation $y'(t) = -y(t)$ for all positive h.

Let's check stability for another method for solving ODEs.

CHALLENGE 20.6.

We want to solve the ODE

$$y'(t) + ay(t) = 0,$$

where $a > 0$ is a scalar, and $y(0)$ is given. We apply the **Crank–Nicholson method** to the ODE:

$$\frac{y^{n+1} - y^n}{h} + a\frac{y^{n+1} + y^n}{2} = 0,$$

where $y^n \approx y(nh)$ and $h > 0$ is the timestep. For what range of h values is the numerical method stable?

Figure 20.7 gives a geometric illustration of the importance of stability in solving ODEs.

20.2.2 Predictor-Corrector Methods

We could use our favorite nonlinear equation solver from Unit VI in Algorithm 20.2, but this is expensive. In rare occasions, as in Challenge 20.5, the nonlinear equation can be solved explicitly, but usually we use **functional iteration**, as shown in Algorithm 20.3. We could repeat the CE steps if the value of $\boldsymbol{y}_{n+1}$ has not converged (although reducing the stepsize is generally a safer cure). If repeated m times, we call this a **PE(CE)^m** scheme. Note that using this method usually fails to solve the nonlinear equation exactly, so this adds an additional source of error.

Let's see how this works on two examples.

Algorithm 20.3 Predictor-Corrector (PECE) Algorithm

Given: $\boldsymbol{y}_0$ and $t_0, t_1, \ldots t_N$.
for $n = 0, \ldots, N-1$
  P (predict): Guess $\boldsymbol{y}_{n+1}$ using a formula that only requires function values for times earlier than t_{n+1} (perhaps using Euler's method $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n \boldsymbol{f}_n$).
  E (evaluate): Evaluate $\boldsymbol{f}_{n+1} = \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})$.
  C (correct): Guess $\boldsymbol{y}_{n+1}$ using a formula that requires function values for times including t_{n+1} (perhaps using the backward Euler method $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n \boldsymbol{f}_{n+1}$).
  E (evaluate): Evaluate $\boldsymbol{f}_{n+1} = \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})$.
end

CHALLENGE 20.7.

Let

$$y'(t) = y^2(t) - 5t,$$
$$y(0) = 1.$$

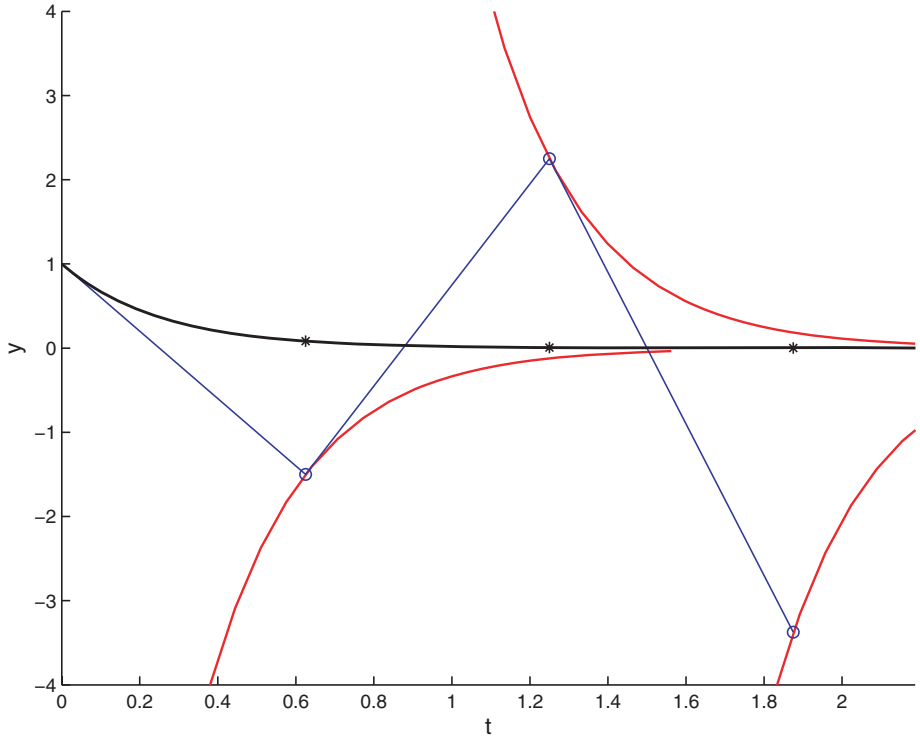


Figure 20.7. *A family of solutions to a stiff differential equation* $y'(t) = -4y(t)$. *The solution curve for* $y(0) = 1$ *is plotted in black, and other family members are plotted in red. We use Euler's method to approximate the true solution (marked with stars) by the values marked with circles. Notice that as we step with Euler's method, we move from one solution curve to another, and these curves can be qualitatively quite different. Therefore, our approximation, marked with blue circles, is quite different from the true solution. Lack of stability led to a very poor estimate.*

Apply a PECE scheme to this problem, using Euler and backward Euler with a stepsize $h = .1$, to obtain an approximation for $y(.1)$.

CHALLENGE 20.8.

Let

$$y'(t) = 10y^2(t) - 20,$$
$$y(0) = 1.$$

Apply a PECE scheme to this problem, using Euler and backward Euler with a stepsize $h = .1$, to obtain an approximation for $y(.1)$. What went wrong?

20.2.3 The Adams Family

Euler and backward Euler are the simplest examples of methods in the **Adams family**. Explicit methods like Euler's, which can be derived by polynomial interpolation at known function values, give **Adams–Bashforth** formulas, while those that use one unknown function value give **Adams–Moulton** formulas. To derive Adams–Bashforth formulas, notice that

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\, \mathrm{d}t\,,$$

and we can approximate the integrand by polynomial interpolation at the points

$$(t_n, f_n), \ldots, (t_{n-k+1}, f_{n-k+1})$$

for some integer k, as illustrated in Figure 20.8. For $k = 1$, this yields a fourth approach to deriving Euler's method: approximate the integral by the rectangle rule using the left endpoint so that

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))\, \mathrm{d}t \approx y_n + f(t_n, y_n)(t_{n+1} - t_n).$$

For Adams–Bashforth with $k = 2$, we interpolate the integrand by a polynomial of degree 1, with

$$\begin{aligned} p(t_n) &= f_n, \\ p(t_{n-1}) &= f_{n-1}, \end{aligned}$$

so letting $h_n = t_{n+1} - t_n$ and $h_{n-1} = t_n - t_{n-1}$, we obtain

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n-1} + \frac{f_n - f_{n-1}}{t_n - t_{n-1}}(t - t_{n-1})\, \mathrm{d}t \\ &= y_n + h_n f_{n-1} + \frac{f_n - f_{n-1}}{h_{n-1}} \frac{(h_n + h_{n-1})^2 - h_{n-1}^2}{2}\,. \end{aligned}$$

For Adams–Moulton, we approximate the integrand by polynomial interpolation at the points

$$(t_{n+1}, f_{n+1}), \ldots, (t_{n-k+2}, f_{n-k+2})$$

for some integer k. For $k = 1$, we derive the backward Euler method from the approximation:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))\, \mathrm{d}t \approx y_n + f(t_{n+1}, y_{n+1})(t_{n+1} - t_n).$$

For Adams–Moulton with $k = 2$, we interpolate the integrand by a polynomial of degree 1, with

$$\begin{aligned} p(t_n) &= f_n, \\ p(t_{n+1}) &= f_{n+1}, \end{aligned}$$

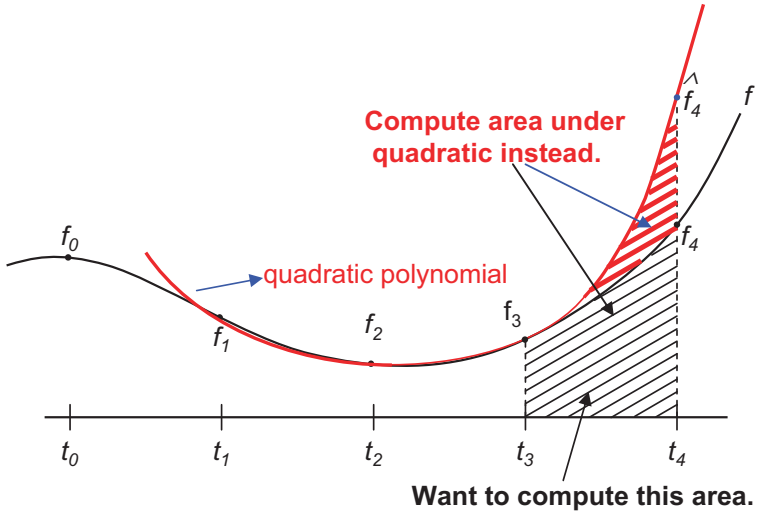


Figure 20.8. *Third-order Adams methods. Adams–Bashforth approximates the area under the curve from $t = t_3$ to $t = t_4$ by integrating the quadratic polynomial that interpolates the curve using the data f_1, f_2, and f_3. Adams–Moulton would interpolate f_2, f_3, and an estimate of f_4.*

so we obtain

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n+1} + \frac{f_n - f_{n+1}}{t_n - t_{n+1}}(t - t_{n+1})\,\mathrm{d}t \\ &= y_n + \frac{h_n}{2}(f_{n+1} + f_n), \end{aligned}$$

which is a generalization of the **trapezoidal rule for integration**.

Adams methods use an Adams–Bashforth formula as a predictor and an Adams–Moulton formula as a corrector in Algorithm 20.3. Some sample Adams formulas are given in Table 20.1. Note that the two families share function evaluations, so for a **PECE** scheme, the cost per step is only 2 evaluations of f, independent of k.

To make a PECE scheme practical, we need to add **stepsize control** and **error estimation**.

20.2.4 Some Ingredients in Building a Practical ODE Solver

We have two tools to help us control the size of the local error:

- We can change the stepsize h.
- We can change the order k (where the local error is $O(h^{k+1})$).

Therefore, most ODE solvers ask the user for a tolerance parameter and try to keep the local error below this tolerance. Note, as we saw in Figure 20.6, that controlling the local error says nothing about control of global error, unless we know something about the stability of the ODE.

Table 20.1. *Some Adams formulas, assuming equal stepsizes h.*

Some Adams–Bashforth Formulas:	Order	Local Error
$y_{n+1} = y_n + hf_n$	$k=1$	$\frac{h^2}{2}y^{(2)}(\eta)$
$y_{n+1} = y_n + \frac{h}{2}(3f_n - f_{n-1})$	$k=2$	$\frac{5h^3}{12}y^{(3)}(\eta)$
$y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2})$	$k=3$	$\frac{3h^4}{8}y^{(4)}(\eta)$
Some Adams–Moulton Formulas:	**Order**	**Local Error**
$y_{n+1} = y_n + hf_{n+1}$	$k=1$	$-\frac{h^2}{2}y^{(2)}(\eta)$
$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1})$	$k=2$	$-\frac{h^3}{12}y^{(3)}(\eta)$
$y_{n+1} = y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1})$	$k=3$	$-\frac{h^4}{24}y^{(4)}(\eta)$

To control local error, we need to have a means of estimating it. For example, consider the use of these formulas:

$$\begin{array}{ll} \text{P} & y^{\text{P}}_{n+1} = y_n + \frac{h_n}{2}(3f_n - f_{n-1}) \quad \text{(Adams–Bashforth)}, \\ \text{E} & f_{n+1} = f(t_{n+1}, y^{\text{P}}_{n+1}), \\ \text{C} & y^{\text{C}}_{n+1} = y_n + \frac{h_n}{2}(f_n + f_{n+1}) \quad \text{(Adams–Moulton)}, \\ \text{E} & f_{n+1} = f(t_{n+1}, y^{\text{C}}_{n+1}). \end{array}$$

To estimate the local error in this method, we compute, using Table 20.1,

$$y^{\text{P}}_{n+1} - y(t_{n+1}) = \frac{5h_n^3}{12}y^{(3)}(\eta_1),$$
$$y^{\text{C}}_{n+1} - y(t_{n+1}) = -\frac{h_n^3}{12}y^{(3)}(\eta_2),$$

where $y(t_{n+1})$ is the solution to $y' = f(t,y)$ with $y(t_n) = y_n$ rather than with $y(0) = y_0$. Therefore we see that

$$y^{\text{P}}_{n+1} - y^{\text{C}}_{n+1} = \frac{h_n^3}{12}(5y^{(3)}(\eta_1) + y^{(3)}(\eta_2)) \approx \frac{6h_n^3}{12}y^{(3)}(\eta),$$

whose magnitude is about 6 times that of the local error in the corrector. (The points η_1, η_2, and η are all in the interval $[t_n, t_{n+1}]$.) This gives us a practical means for estimating local error, since $|y^{\text{P}}_{n+1} - y^{\text{C}}_{n+1}|$ is computable: we now know that if $|y^{\text{P}}_{n+1} - y^{\text{C}}_{n+1}|$ is small enough, then the local error should be small, but if it is big we should worry about the local error.

Practical methods for solving ODEs use such estimates for the local error to determine whether the current choice of stepsize h_n is adequate. If the local error estimate is larger than that requested by the user, the stepsize is usually halved, and the step is repeated. If the local error estimate is much lower than requested, then the stepsize is doubled for the next step. By using factors of two, old function values can be reused; see Figure 20.9.

Let's get some practice with the idea of error estimation and stepsize control.

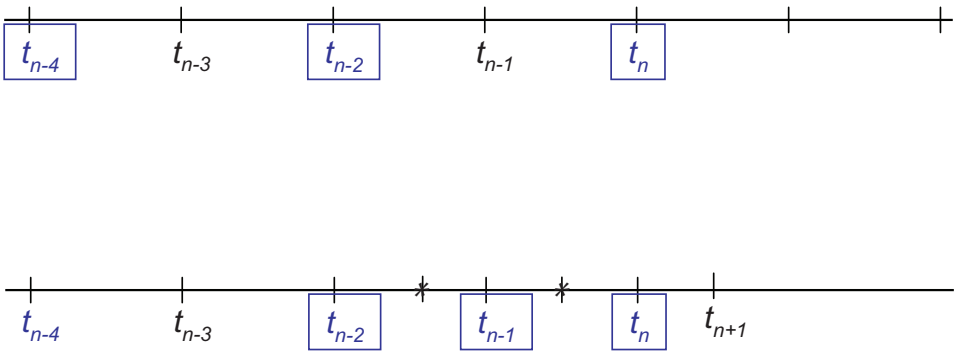


Figure 20.9. *(Top) Given data at t_{n-4}, t_{n-3}, t_{n-2}, t_{n-1}, and t_n, if we decide to increase the stepsize in our Adams formula by a factor of 2, we can reuse the function values at t_{n-4}, t_{n-2}, and t_n to obtain an approximation at t_{n+1}. (Bottom) If we reduce the stepsize by factor of two, then we interpolate to approximate the function values at the points marked with stars.*

CHALLENGE 20.9.

Suppose we have used a PECE method with predictor of order 4 (i.e., local error is proportional to $O(h^5)$, where h is the stepsize) and corrector of order 5. We want to keep the local error less than τ. Estimate the local error and explain how to alter the stepsize if necessary to achieve our local error criterion.

CHALLENGE 20.10.

Suppose we have used a PECE method with Adams–Bashforth and Adams–Moulton formulas of order 3 to form two estimates of $y(t_{n+1})$. How would you estimate the local error in the Adams–Moulton formula? How would you use that estimate to change h in order to keep the estimated local error less than a user-supplied local error tolerance τ without taking steps smaller than necessary?

We also change the order of the method to control error and get ourselves started. When we take our first step with an Adams method, we need to use Euler's method, because we have no history. When we take our second step, we can use the second-order Adams methods, because we have information about one old point. As we keep walking, we can use higher-order methods, by saving information about old function values. But there is a limit to the usefulness of history; generally, fifth-order methods are about as high as we go, and the order is kept smaller than this if the local error estimate indicates that the high-order derivatives are large.

CHALLENGE 20.11. (Extra)

- Write MATLAB statements to start an Adams algorithm. Begin using Euler's method, and gradually increase the order of the method to 3. Note that to control the error, the stepsizes are generally smaller when using lower-order methods.

POINTER 20.4. Controlling Error.
By changing h and k in our ODE solver, we control the local error. It is important to note that the user really cares about global error, not local error, but this depends on the stability of the differential equation. To estimate the global error, we might perform sensitivity analysis as discussed in the case study of Chapter 2, seeing how much the computed solution changes when the initial conditions are perturbed a bit. We might also experiment with perturbations to $\boldsymbol{f}$, but random noise introduced in $\boldsymbol{f}$ causes large-magnitude derivatives and therefore small stepsizes in the ODE solver.

- Write MATLAB statements to estimate the error and decide whether to change h.
- Write MATLAB statements to decide whether to cut back to a lower-order method.

Note that PECE using the Adams formulas is relatively inexpensive, since, no matter what the order of the method, we only need two new function evaluations per step. Thus there is an advantage to using higher-order methods, since they allow a larger stepsize (and thus fewer steps) with little additional work. One pitfall to watch for is that since PECE provides an approximate solution to a nonlinear equation, we should suspect that we have not solved very accurately if $|y^P_{n+1} - y^C_{n+1}|$ is large; in this case, the stepsize should be reduced.

20.2.5 Solving Stiff Problems

The Adams family with PECE is good for nonstiff problems, but using this method to solve a stiff problem may result in artificial oscillation. An illustration is given in Figure 20.7. Instead, a **Gear family** of formulas is effective for stiff problems and can also be used for nonstiff problems, although it is usually uses somewhat smaller stepsizes than the Adams family. See Pointer 20.11.

20.2.6 An Alternative to Adams Formulas: Runge–Kutta

Runge–Kutta methods build on the idea we used in the Euler method of predicting a new value by walking along a tangent to the curve. They just do it in a more complicated way.

For example, suppose we let

$$\begin{aligned} k_1 &= h_n f(t_n, y_n), \\ k_2 &= h_n f(t_n + \alpha h, y_n + \beta k_1), \\ y_{n+1} &= y_n + a k_1 + b k_2, \end{aligned}$$

where we choose α, β, a, b to make the formula as accurate as possible.

We expand everything in Taylor series,

$$y(t_{n+1}) = y(t_n) + y'(t_n)h_n + y''(t_n)\frac{h_n^2}{2} + O(h_n^3), \tag{20.3}$$

and match as many terms as possible. Note that

$$y'(t_n) = f(t_n, y_n) \equiv f,$$
$$y''(t_n) = \frac{\partial f}{\partial t}(t_n, y_n) + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}(t_n, y_n) \equiv f_t + f_y f,$$

so (20.3) can be written

$$y(t_{n+1}) = y(t_n) + f h_n + (f_t + f_y f)\frac{h_n^2}{2} + O(h_n^3). \tag{20.4}$$

Now we expand k_1 and k_2:

$$\begin{aligned} k_1 &= h_n f(t_n, y_n), \\ k_2 &= h_n[f(t_n, y_n) + \alpha h_n f_t + \beta k_1 f_y + O(h_n^2)] \\ &= h_n f(t_n, y_n) + \alpha h_n^2 f_t + \beta k_1 h_n f_y + O(h_n^3), \end{aligned}$$

so our Runge–Kutta method satisfies

$$\begin{aligned} y_{n+1} &= y_n + a k_1 + b k_2 \\ &= y_n + a h_n f + b(h_n f + \alpha h_n^2 f_t + \beta h_n f h_n f_y) + O(h_n^3) \\ &= y_n + (a+b) h_n f + \alpha b h_n^2 f_t + \beta b h_n^2 f_y f + O(h_n^3). \end{aligned}$$

We want to match the coefficients in this expansion to the coefficients in the Taylor series expansion of y in (20.4), so we set

$$\begin{aligned} a + b &= 1, \\ \alpha b &= \frac{1}{2}, \\ \beta b &= \frac{1}{2}. \end{aligned}$$

There are many solutions (in fact, an infinite number). One of them is Heun's method:

$$a = \frac{1}{2}, \quad b = \frac{1}{2}, \quad \alpha = 1, \quad \beta = 1.$$

This choice gives a **second-order Runge–Kutta method**. Note that the work per step is 2 evaluations of $\boldsymbol{f}$.

The most useful Runge–Kutta method is one of order 4:

$$\begin{aligned} \boldsymbol{k}_1 &= h_n \boldsymbol{f}(t_n, \boldsymbol{y}_n), \\ \boldsymbol{k}_2 &= h_n \boldsymbol{f}\left(t_n + \frac{h_n}{2}, \boldsymbol{y}_n + \frac{\boldsymbol{k}_1}{2}\right), \\ \boldsymbol{k}_3 &= h_n \boldsymbol{f}\left(t_n + \frac{h_n}{2}, \boldsymbol{y}_n + \frac{\boldsymbol{k}_2}{2}\right), \\ \boldsymbol{k}_4 &= h_n \boldsymbol{f}(t_n + h_n, \boldsymbol{y}_n + \boldsymbol{k}_3), \\ \boldsymbol{y}_{n+1} &= \boldsymbol{y}_n + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4). \end{aligned}$$

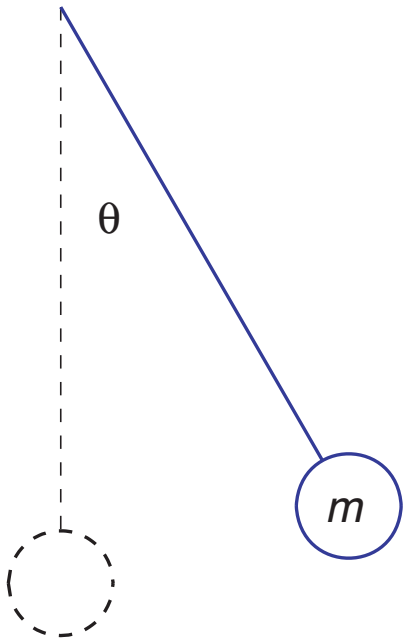


Figure 20.10. *Our simple harmonic oscillator, a pendulum. The pendulum is suspended at the origin and the string has length r. As the pendulum moves, its position is defined by $x(t) = p(t) = r\sin\theta(t)$, $y(t) = q(t) = -r\cos\theta(t)$ for some function $\theta(t)$.*

It requires 4 evaluations of f per step, and many pages for its derivation.

The fact that the Runge–Kutta methods use no old function values is both an advantage and a disadvantage. In contrast to PECE methods, it is easy to change the stepsize, since no old function values are needed. But PECE methods require only two function evaluations per iteration, regardless of order, so the Runge–Kutta methods use more work per iteration when the order is larger than 2.

20.3 Hamiltonian Systems

In some ODE systems, there is an associated **conservation law**, and if possible, we formulate the problem so that conservation is observed.

A **Hamiltonian system** is one for which there exists a scalar **Hamiltonian function** $H(\boldsymbol{y})$ so that

$$\boldsymbol{y}'(t) = \boldsymbol{D}\nabla_{\boldsymbol{y}} H(\boldsymbol{y}(t)), \tag{20.5}$$

where $\boldsymbol{D}$ is a block-diagonal matrix with blocks equal to

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

and $\nabla_{\boldsymbol{y}} H(\boldsymbol{y}(t))$ denotes the gradient of H with respect to the $\boldsymbol{y}$ variables.

For example, consider the pendulum pictured in Figure 20.10. The functions $p(t)$ and $q(t)$ that define the position of the pendulum bob satisfy

$$q'(t) = \omega p(t), \tag{20.6}$$
$$p'(t) = -\omega q(t), \tag{20.7}$$

where $\omega > 0$ is a fixed parameter. The length r of the string does not change, so we might want to preserve this invariance, or conservation law, in our numerical method, requiring that the Hamiltonian of the system

$$H(t) = \frac{\omega}{2}(p^2(t) + q^2(t))$$

remain constant, as it does for the true solution. To verify (20.5) for this example, note that if $\boldsymbol{y}(t) = [q(t), p(t)]^T$, then

$$\nabla_{\boldsymbol{y}} H(\boldsymbol{y}(t)) = \begin{bmatrix} \omega q(t) \\ \omega p(t) \end{bmatrix},$$

so that

$$\boldsymbol{y}'(t) = \boldsymbol{D}\nabla_{\boldsymbol{y}} H(\boldsymbol{y}(t)) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \omega q(t) \\ \omega p(t) \end{bmatrix},$$

which is (20.6)-(20.7). Note that differentiating with respect to t gives

$$\begin{aligned} H'(t) &= \frac{\omega}{2}(2p(t)p'(t) + 2q(t)q'(t)) \\ &= \frac{\omega}{2}(2p(t)(-\omega q(t)) + 2q(t)(\omega p(t))) \\ &= 0, \end{aligned}$$

so $H(t)$ must be constant; in other words, the quantity $H(t)$ is conserved or invariant. We can verify this a different way by writing the general solution to the problem

$$\begin{bmatrix} q(t) \\ p(t) \end{bmatrix} = \begin{bmatrix} \cos\omega t & \sin\omega t \\ -\sin\omega t & \cos\omega t \end{bmatrix} \begin{bmatrix} q(0) \\ p(0) \end{bmatrix},$$

and computing $p(t)^2 + q(t)^2$. The eigenvalues of the matrix defining the solution are imaginary numbers, so a small perturbation of the matrix can cause the quantity $H(t)$ to either grow or shrink, and this does not produce a useful estimate of $p(t)$ and $q(t)$. Therefore, in solving systems involving Hamiltonians (conserved quantities), it is important to also build conservation into the numerical method whenever possible.

If, as in the pendulum example, an ODE of the form

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)), \tag{20.8}$$

$$H(\boldsymbol{y}(t)) = 0, \tag{20.9}$$

is overdetermined, with more equations than unknown functions, then we can rewrite it as

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)) - \boldsymbol{G}(\boldsymbol{y}(t))z(t), \tag{20.10}$$

$$H(\boldsymbol{y}(t)) = 0, \tag{20.11}$$

where $z(t)$ is another function (added so that the system is not overdetermined) and $\boldsymbol{G}(\boldsymbol{y}(t))$ is a function whose derivative matrix is bounded away from singularity for all t. If we solve system (20.10)–(20.11) exactly, then we get $z(t) = 0$ and we recover our original solution. But if we solve it numerically, the conservation law $H(\boldsymbol{y}(t)) = 0$ forces $z(t)$ to be nonzero to compensate for numerical errors.

For our pendulum, for example, we can choose $\boldsymbol{G}(\boldsymbol{y}) = \omega \boldsymbol{y}$ and rewrite our harmonic oscillator example as

POINTER 20.5. Warning about Hamiltonian Systems.
Sometimes, adding conservation to an ODE system makes the problem too expensive to solve; for example, if the solution is rapidly oscillating, the conservation law forces very small stepsizes.

$$q'(t) = \omega p(t) - \omega q(t) z(t), \tag{20.12}$$
$$p'(t) = -\omega q(t) - \omega p(t) z(t), \tag{20.13}$$
$$0 = \frac{\omega}{2}(p^2(t) + q^2(t) - r^2). \tag{20.14}$$

Let's consider another example of a Hamiltonian.

CHALLENGE 20.12.
Derive the Hamiltonian system for

$$H(\mathbf{y}) = \frac{1}{2}y_1^2(t) + \frac{1}{2}y_2^2(t) + \frac{1}{2}y_3^2(t) + \frac{1}{2}y_4^2(t) + \frac{1}{2}y_1^2(t)y_2^2(t)y_3^2(t)y_4^2(t).$$

Adding an invariant, or conservation law, generally changes the ODE system to a system that includes nonlinear equations not involving derivatives as in (20.12)-(20.14) – a system of **differential-algebraic equations** (DAEs). Next we'll consider a little of the theory and computation of such DAEs.

20.4 Differential-Algebraic Equations

The general DAE has the form

$$\boldsymbol{F}(t, \widehat{\boldsymbol{y}}(t), \widehat{\boldsymbol{y}}'(t)) = \boldsymbol{0},$$

where $\boldsymbol{F}$ is a specified function. We'll consider a special case in which the problem can be written as

$$\boldsymbol{M}(t)\boldsymbol{y}'(t) = \boldsymbol{A}(t)\boldsymbol{y}(t) + \boldsymbol{f}(t). \tag{20.15}$$

The matrix in front of the derivatives is called the **mass matrix**. In the next challenge we convert a DAE to this **standard form**.

CHALLENGE 20.13.
Let the three variables $u(t)$, $v(t)$, and $w(t)$ be related by

$$u'(t) = 7u(t) - 6v(t) + 4t,$$
$$v'(t) = 4u(t) - 2v(t),$$
$$u(t) + v(t) + w(t) = 24.$$

Convert this to a system of the form $\boldsymbol{M}(t)\boldsymbol{y}'(t) = \boldsymbol{A}(t)\boldsymbol{y}(t) + \boldsymbol{f}(t)$.

POINTER 20.6. Existence and Uniqueness of Solutions to DAEs.

DAEs are a combination of ODEs and algebraic (linear or nonlinear) equations, so the subject of existence and uniqueness of solutions is somewhat complicated. We present a set of conditions for the DAE

$$\boldsymbol{M}(t)\boldsymbol{y}'(t) = \boldsymbol{A}(t)\boldsymbol{y}(t) + \boldsymbol{f}(t).$$

For any nonnegative integer ℓ, define the $(\ell+1)m \times (\ell+1)m$ matrix

$$\boldsymbol{P}_\ell(t) = \begin{bmatrix} \boldsymbol{M}(t) & & & & \\ \boldsymbol{M}'(t) - \boldsymbol{A}(t) & \boldsymbol{M}(t) & & & \\ \boldsymbol{M}''(t) - 2\boldsymbol{A}'(t) & 2\boldsymbol{M}'(t) - \boldsymbol{A}(t) & & & \\ \vdots & & \ddots & \ddots & \\ \boldsymbol{M}^{(\ell)}(t) - \ell\boldsymbol{A}^{(\ell-1)}(t) & \dots & \dots & \ell\boldsymbol{M}'(t) - \boldsymbol{A}(t) & \boldsymbol{M}(t) \end{bmatrix},$$

where the (i,j) block below the diagonal contains $\binom{i-1}{j-1}\boldsymbol{M}^{(i-j)} - \binom{i-1}{j}\boldsymbol{A}^{(i-j-1)}$. Define the $(\ell+1)m \times (\ell+1)m$ matrix

$$\boldsymbol{N}_\ell(t) = \begin{bmatrix} \boldsymbol{A}(t) & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} \\ \boldsymbol{A}'(t) & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} \\ \vdots & \vdots & & & \vdots \\ \boldsymbol{A}^{(\ell)}(t) & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} \end{bmatrix}.$$

Suppose that for some value of ℓ, there are integers m_a and m_d, with $m_a + m_d = m$, such that for all values of t in the interval of interest:

- $\boldsymbol{P}_\ell(t)$ and $\boldsymbol{N}_\ell(t)$ are continuously differentiable.
- $\operatorname{rank}(\boldsymbol{P}_\ell(t)) = (\ell+1)m - m_a$, and the m_a columns of some matrix $\boldsymbol{Z}(t)$ form a basis for the null space of $\boldsymbol{P}_\ell(t)^*$.
- Let $\widehat{\boldsymbol{N}}_\ell(t) = \boldsymbol{Z}(t)^*\boldsymbol{N}_\ell(t)[\boldsymbol{I}_n, \boldsymbol{0}, \dots, \boldsymbol{0}]^T$. Then $\operatorname{rank}(\widehat{\boldsymbol{N}}_\ell(t)) = m_a$, and the m_d columns of some matrix $\boldsymbol{T}(t)$ form a basis for the null space of $\widehat{\boldsymbol{N}}_\ell(t)$.
- There exists $\boldsymbol{W}(t)$ of dimension $m \times m_d$ such that $\operatorname{rank}(\boldsymbol{W}(t)^*\boldsymbol{M}(t)\boldsymbol{T}(t)) = m_d$.
- $\boldsymbol{Z}(t)$, $\boldsymbol{T}(t)$, and $\boldsymbol{W}(t)$ are continuously differentiable functions of t.

Then for every consistent initial condition the DAE has a unique solution. See Kunkel and Mehrmann [97, Theorem 3.52] for proof of this result as well as results for more general problems.

20.4.1 Some Basics

DAEs are classified by several parameters:

- m_a is the number of algebraic conditions in the DAE.
- m_d is the number of differential conditions in the DAE, and $m_a + m_d = m$.
- ℓ is the **strangeness** of the DAE.

Often a fourth parameter is considered: the **differential-index** of a DAE is the number of differentiations needed to convert the problem to an (explicit) system of ODEs. A system of ODEs has differential-index 0, and a system of algebraic equations $\boldsymbol{F}(\boldsymbol{y}) = \boldsymbol{0}$ has differential-index 1.

Existence and uniqueness of the solution to the DAE can be checked using the result in Pointer 20.6. For example, if $\boldsymbol{M}(t)$ in equation (20.15) has full rank for all t, then $\ell = 0$, $m_a = 0$, $\boldsymbol{Z} = [\,]$, and $\boldsymbol{W} = \boldsymbol{T} = \boldsymbol{I}_m$, and we have a system of ODEs. Alternatively, if $\boldsymbol{M} = \boldsymbol{0}$ and $\boldsymbol{A}(t)$ is full rank, then $\ell = 0$, $m_a = m$, $\boldsymbol{Z} = \boldsymbol{I}_m$, and $\boldsymbol{W} = \boldsymbol{T} = [\,]$, and we have a system of algebraic equations.

Let's consider a problem with strangeness $\ell = 1$.

CHALLENGE 20.14.

The **nonstationary Stokes equation** can be discretized in space to give the DAE

$$\begin{aligned} \boldsymbol{u}'(t) &= \boldsymbol{C}\boldsymbol{u}(t) + \boldsymbol{B}\boldsymbol{p}(t), \\ \boldsymbol{B}^T \boldsymbol{u}(t) &= \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{u}(t)$ is an $m_u \times 1$ vector of fluid velocities and $\boldsymbol{p}(t)$ is an $m_p \times 1$ vector of pressures, with $m_p < m_u$. Suppose $\boldsymbol{B}^T$ is a real full-rank matrix, and that the columns of the $m_u \times (m_u - m_p)$ matrix $\boldsymbol{X}$ are a basis for its null space. Verify the hypotheses of Pointer 20.6 for $\ell = 1$.

There is a major difference between DAEs and ODEs: For ODEs, it is easy to count how many initial conditions we need to uniquely determine the solution. For DAEs, it is not so simple; initial conditions may be inconsistent with the problem. For example, the equation

$$e^y = 5$$

is (trivially) a DAE and, of course, nonlinear equations like this require no initial conditions to specify the solution.

We now consider some numerical methods for solving DAEs.

20.4.2 Numerical Methods for DAEs

The main idea used to solve DAEs follows from what we know about data fitting. If we want to solve the DAE

$$\boldsymbol{F}(t, \boldsymbol{y}(t), \boldsymbol{y}'(t)) = \boldsymbol{0},$$

then we step from known values at $t = t_n, t_{n-1}, \ldots, t_{n-k}$ to unknown values at $t = t_{n+1}$ using our favorite approximation scheme to replace $\boldsymbol{y}'(t_{n+1})$ by

$$\boldsymbol{y}'(t_{n+1}) \approx \sum_{j=0}^{k} \alpha_i \boldsymbol{y}(t_{n+1-j}).$$

POINTER 20.7. DAE **Software.**

The MATLAB ODE solvers, including `ode23s`, for example, handle some DAEs. There are several high-quality packages specifically designed for DAEs, including:

- SUNDIALS from Lawrence Livermore National Laboratory (`www.llnl.gov.CASC/sundials`),
- GELDA and GENDA by Peter Kunkel and Volker Mehrmann (`www.math.tu-berlin.de/numerik/mt.NumMat`).

Other software references can be found in [97].

This gives us a nonlinear equation to solve for our approximation $\boldsymbol{y}_{n+1} \approx \boldsymbol{y}(t_{n+1})$:

$$\boldsymbol{F}\left(t_{n+1}, \boldsymbol{y}_{n+1}, \sum_{j=0}^{k} \alpha_i \boldsymbol{y}(t_{n+1-j})\right) = \boldsymbol{0}.$$

In principle, we can solve this equation using our favorite method from Chapter 24 (Newton-like, homotopy, etc.). In practice, there are a few complications:

- Stability is an important consideration. Usually a stiff method is used to ensure numerical stability.
- The nonlinear equation may fail to have a solution.
- Even if a solution exists, the method for solving the nonlinear equation may fail to converge.
- Automatic control of order and stepsize is even more difficult than for ODEs.

The bottom line is that you should not try to write your own solver for DAEs. Use high-quality software, as indicated in Pointer 20.7. We will use a MATLAB DAE solver in Chapter 21.

20.5 Boundary Value Problems for ODEs

Up to now, the solution to our ODE has always been made unique by specifying values of the variables at some fixed time t_0. We consider in this section **boundary value problems**, for which values are specified at two different times.

For example, consider the problem of determining $u(t)$ for $t \in (0,1)$ given that

$$u''(t) = 6u'(t) - tu(t) + u^2(t), \tag{20.18}$$
$$u(0) = 5, \tag{20.19}$$
$$u(1) = 2. \tag{20.20}$$

POINTER 20.8. Existence and Uniqueness of Solutions to BVPs for ODEs.
Consider a problem that can be expressed as

$$\mathcal{A}u(t) = -(a(t)u'(t))' + b(t)u'(t) + c(t)u(t) = f(t) \text{ for } t \in (0,1), \tag{20.16}$$

$$u(0) = u_0, \qquad u(1) = u_1, \tag{20.17}$$

where $a(t)$, $b(t)$, $c(t)$, and $f(t)$ are given differentiable functions and u_0 and u_1 are given numbers. The interval is set to $t \in (0,1)$, but this can be changed by rescaling.

It is important to know that the problem is well posed, in the sense that a unique solution exists (with two continuous derivatives), and that small changes in the data lead to small changes in the solution. There are various conditions on the coefficients that guarantee this. One set of sufficient conditions is

- $a(t) \geq a_0$ for $t \in [0,1]$, where a_0 is a number greater than 0.
- $c(t) \geq 0$ for $t \in [0,1]$.
- $\int_0^1 |f(t)|^2 \, dt$ is finite.

For further information, consult a standard textbook such as [99, Chap. 2].

If we convert this to a system of first order equations, we let $y_{(1)}(t) = u(t)$, $y_{(2)}(t) = u'(t)$ and obtain

$$y'_{(1)}(t) = y_{(2)}(t) \tag{20.21}$$

$$y'_{(2)}(t) = 6y_{(2)}(t) - ty_{(1)}(t) + y^2_{(1)}(t), \tag{20.22}$$

$$y_{(1)}(0) = 5, \tag{20.23}$$

$$y_{(1)}(1) = 2. \tag{20.24}$$

So we have values of $y_{(1)}$ at 0 and 1. If we had values of $y_{(1)}$ and $y_{(2)}$ at 0, we could use our IVP methods. But now we have a **boundary value problem**. What can we do? There are three alternatives:

- Adapt our IVP methods to this problem in a method called **shooting**.
- Develop new methods using **finite differences**.
- Develop new methods using **finite elements**.

We consider the first two ideas in the following two sections, and finite element methods in the case study of Chapter 23.

Before studying these methods, though, we illustrate the use of the ideas in Pointers 20.8 and 20.9 to obtain information about a BVP without actually solving the problem.

POINTER 20.9. Bounds on Solutions to BVPs for ODEs.

Two facts about BVPs in the form (20.16)-(20.17) can help us compute bounds on the solutions without computing the solutions themselves.

First, the **maximum principle** tells us that if

- the solution u exists and has two continuous derivatives on $[0,1]$,
- $f(t) \leq 0$ for $t \in (0,1)$,

then we can bound the solution:

- If $c(t) = 0$, then
$$\max_{t\in[0,1]} u(t) = \max(u_0, u_1).$$
- If $c(t) \geq 0$ for $t \in (0,1)$, then
$$\max_{t\in[0,1]} u(t) \leq \max(u_0, u_1, 0).$$

Second, the **monotonicity theorem for ODEs** helps us compare solutions to different ODEs without computing them. In particular, if

- u satisfies $\mathcal{A}u(t) = f(t)$ for $t \in [0,1]$, with $u(0) = u_0$ and $u(1) = u_1$,
- v satisfies $\mathcal{A}v(t) = g(t)$ for $f \in [0,1]$, with $v(0) = v_0$ and $v(1) = v_1$,
- $f(t) \leq g(t)$ for $t \in [0,1]$,
- $u_0 \leq v_0$,
- $u_1 \leq v_1$,

then $u(t) \leq v(t)$ for $t \in [0,1]$.

For further information, consult a standard textbook such as [99, Chap. 2].

CHALLENGE 20.15.

Consider the differential equation

$$-u''(t) + 8.125\pi \cot((1+t)\pi/8)u'(t) + \pi^2 u(t) = -3\pi^2 \text{ on } \Omega = (0,1)$$

with boundary conditions $u(0) = -2.0761$, $u(1) = -2.2929$. Discuss the solution: Does it exist? Is it unique? What are upper and lower bounds on the solution? Justify each of your answers by citing a theorem and verifying its hypotheses. (Hint: One bound can be obtained by comparing the solution to $v(t) = -3$.)

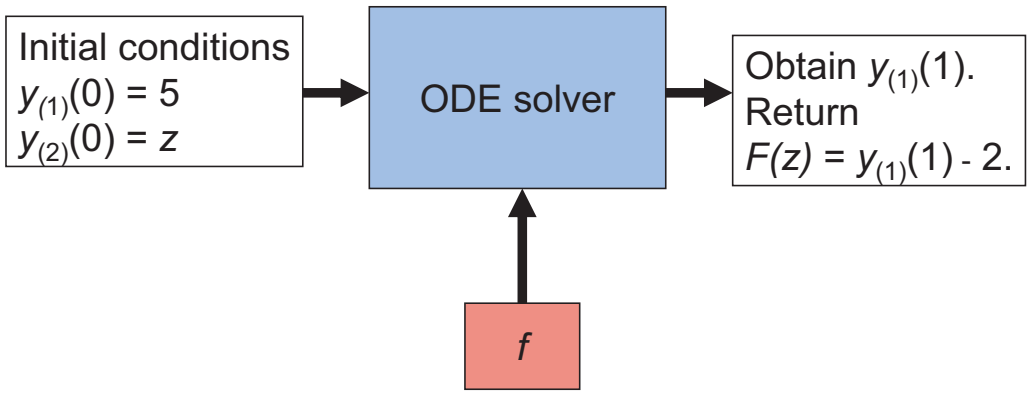


Figure 20.11. *The function evaluation $F(z)$ for the nonlinear equation solver in a shooting method to solve $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$, $y_{(1)}(0) = 5$, $y_{(1)}(1) = 2$. We return the value $F(z) = y_{(1)}(1) - 2$. When the computed value of $y_{(1)}(1) \approx 2$, then we have the correct initial condition z.*

20.5.1 Shooting Methods

When in doubt, guess. The idea behind shooting methods is to guess at the missing initial values, solve the IVP using our favorite method, and then use the results to improve our guess.

In fact, we recognize this as a nonlinear system of equations: to solve our example problem (20.18)–(20.20), we want to solve the nonlinear equation

$$F(z) = 0,$$

where z is the value we give to $y_{(2)}(0)$ and $F(z)$ is the difference between the value that our (IVP) ODE solver returns for $y_{(1)}(1)$ and the desired value, 2. So a **shooting method** involves using a nonlinear equation solver, as illustrated in Figure 20.11. In doing this we evaluate $F(z)$ by applying our favorite IVP-ODE solver to (20.21)–(20.23). Once we find the initial value z, then the IVP-ODE solver can estimate values $\boldsymbol{u}(t)$ for any t.
Some warnings:

- If the IVP is difficult to solve (for example, stiff), then it is difficult to get an accurate estimate of z.

- Our function evaluation for the nonlinear equation $F(z) = 0$ is **noisy**: it includes all of the rounding error and the global discretization error introduced by the IVP-ODE solver. The resulting wiggles in the values of F can cause the nonlinear equation solver to have trouble finding an accurate solution, and can also introduce multiple solutions where there is really only one; see Figure 20.12.

- If the interval of integration is long, these difficulties can be overwhelming and we need to go to more complicated methods; for example, **multiple shooting**.

Let's see how shooting works.

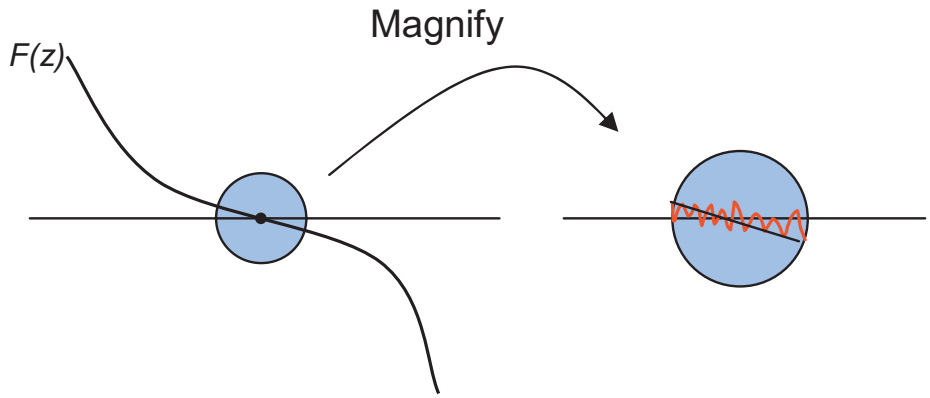


Figure 20.12. *In a shooting method, the evaluation of the function is noisy. Instead of computing points on the (true) black curve, we compute points on the red curve. Thus the computed function may have several zeros near the true solution.*

CHALLENGE 20.16.
Let

$$\begin{aligned} a''(t) &= a^2(t) - 5a'(t), \\ a(0) &= 5, \\ a(1) &= 2. \end{aligned}$$

Convert this to a system of the form $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}(t))$ and write a MATLAB program that uses a shooting method to solve it.

CHALLENGE 20.17.
Let

$$u''(t) = -\left(\frac{\pi}{2}\right)^2 (u(t) - t^2) + 2,$$

with $u(0) = u(1) = 1$. Write a MATLAB program to solve this problem using the shooting method.

CHALLENGE 20.18. (Extra)
Write a program to solve the BVP (20.18)-(20.20) using `ode45` and `fzero`.

20.5.2 Finite Difference Methods

Finite difference methods provide an alternative to shooting methods for BVPs. We derive two finite difference formulas in the following exercise.

CHALLENGE 20.19.
Suppose u has 4 continuous derivatives. Prove that

$$u'(t) = \frac{u(t+h) - u(t-h)}{2h} + O(h^2),$$
$$u''(t) = \frac{u(t-h) - 2u(t) + u(t+h)}{h^2} + O(h^2)$$

for small values of h.

Now return to our example problem in its original form (20.18)-(20.20). Given a large number n (for example, $n = 100$), let $h = 1/n$ and define

$$u_j \approx u(jh), \qquad j = 0, \ldots, n.$$

Then using the formulas in Challenge 20.19 we can approximate our original equation

$$u''(t) = 6u'(t) - tu(t) + u^2(t)$$

at $t = t_j$ $(0 < j < n)$ by

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = 6\frac{u_{j+1} - u_{j-1}}{2h} - t_j u_j + u_j^2.$$

Since we already know that

$$u_0 \approx u(0) = 5,$$
$$u_n \approx u(1) = 2,$$

we have a system of $n-1$ nonlinear equations in $n-1$ unknowns with a tridiagonal coefficient matrix. Defining

$$\begin{aligned} a_{jj} &= -\tfrac{2}{h^2} + jh, & j &= 1, \ldots, n-1, \\ a_{j,j+1} &= \tfrac{1}{h^2} - \tfrac{3}{h}, & j &= 1, \ldots, n-2, \\ a_{j,j-1} &= \tfrac{1}{h^2} + \tfrac{3}{h}, & j &= 2, \ldots, n-1, \\ a_{jk} &= 0, & k &\neq j, j-1, j+1, \end{aligned}$$

we have

$$\begin{bmatrix} a_{11} & a_{12} & & & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{n-2,n-3} & a_{n-2,n-2} & a_{n-2,n-1} \\ & & & & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{15}{h} - \frac{5}{h^2} \\ 0 \\ \cdot \\ \cdot \\ 0 \\ \frac{6}{h} - \frac{2}{h^2} \end{bmatrix} + \begin{bmatrix} u_1^2 \\ u_2^2 \\ \cdot \\ \cdot \\ u_{n-2}^2 \\ u_{n-1}^2 \end{bmatrix},$$

or

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b} + \begin{bmatrix} u_1^2 \\ u_2^2 \\ \cdot \\ \cdot \\ u_{n-2}^2 \\ u_{n-1}^2 \end{bmatrix}.$$

Now we can use our favorite method for nonlinear equations (Unit VI). If we don't have nonlinear terms, all we need to do is to solve a linear system (Unit II or VII).

Next we get some practice in forming the system of linear or nonlinear equations for our finite difference method.

CHALLENGE 20.20.

Let

$$u''(t) = u'(t) + 6u(t)$$

with $u(0) = 2$ and $u(1) = 3$. Let $h = 1/5$, and write a set of finite difference equations that approximate the solution to this problem at $t = jh$, $j = 0, \ldots, 5$.

CHALLENGE 20.21.

Suppose we solve the problem

$$\begin{aligned} u''(t) &= u'(t) - tu(t) + e^{u(t)}, \\ u(0) &= 1, \\ u(1) &= 0, \end{aligned}$$

using the finite difference method, approximating $u_i \approx u(ih)$, where $h = .01$, $i = 0, \ldots, 100$. We can use a nonlinear equation solver on the system $\boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{0}$, where there are 99 unknowns and 99 equations. Write the equations for $\boldsymbol{F}(\boldsymbol{u})$.

20.6 Summary

Some tips for solving the problems considered in this chapter are given in Pointer 20.10. In the next chapter we see how differential equations arise in models for problems such as the spread of an epidemic.

POINTER 20.10. Some Tips for Solving Differential Equations.

- IVPs for ODEs that arise in practice can be very difficult to solve.
  - When in doubt, use a stiff method.
  - If there is a **conservation law** or Hamiltonian, make sure to incorporate it into the formulation. (Otherwise, your user may be very unhappy with the numerical results.) But be aware that if you don't do this in a smart way, it may cause the ODE solver to take very small steps.
- We have just touched on the existence, uniqueness, and stability theory for ODEs and DAEs. If you need to solve an important problem, be ready to study these issues further before you go to the computer.
- Numerical solution of DAEs is still an evolving science, so watch the literature if you are working in this field.
- For BVPs, finite difference methods and finite element methods (Chapter 23) are the most commonly used methods.

POINTER 20.11. Further Reading.

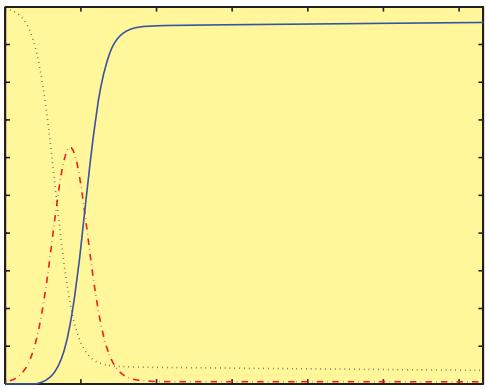
More information on numerical solution of ODEs can be found, for example, in Chapter 9 of Van Loan's book [148]. More Adams formulas are listed in Van Loan [148, p. 354].

The Gear family of formulas is discussed, for example, in [54] and used in the MATLAB stiff ODE solvers whose names end in 's'.

DAEs are discussed in the books by Brenan, Campbell, and Petzold [17] and by Kunkel and Mehrmann [97].

Chapter 21 / Case Study

More Models of Infection: It's Epidemic



In the case study of Chapter 19, we studied a model of the spread of an infection through a hospital ward. The ward was small enough that we could track each patient as an individual.

When the size of the population becomes large, it becomes impractical to use that kind of model, so in this case study we turn our attention to models which study the population as a whole.

As before, we divide the population into three groups: At day t, $I(t)$ is the proportion of the population that is infected and $S(t)$ is the proportion of the population that has never been infected. These quantities satisfy $0 \leq I(t) \leq 1$ and $0 \leq S(t) \leq 1$ for $t \geq 0$. The third part, $R(t)$, is the proportion of the population that was once infected but has now recovered, and it can be derived from these two: $R(t) = 1 - I(t) - S(t)$.

Models without Spatial Variation

In the models we studied before, the probability of an individual becoming infected depended on the status of the individual's neighbors. In the models in this section, we consider all individuals to be neighbors. This is equivalent to assuming a **well-mixed** model, in which all members of the population have contact with all others.

How might we model the three groups in the population? If the infection (or at least the contagious phase of the infection) lasts k days, then we might assume as an approximation that the rate of recovery is equal to the number infected divided by k. Thus, on average, $1/k$ of the infected individuals recover each day.

Let τ be the proportion of encounters between an infected individual and a susceptible individual that transmit the infection. Then the rate of new infections should increase as any of the parameters I, S, or τ increases, so we model this rate as $\tau I(t)S(t)$.

Next, we take the limit as the "timestep" Δt goes to zero, obtaining a system of ODEs. This gives us a simple but interesting Model 1:

$$\frac{dI(t)}{dt} = \tau I(t)S(t) - I(t)/k,$$
$$\frac{dS(t)}{dt} = -\tau I(t)S(t),$$

POINTER 21.1. Software.

The MATLAB function `ode23s` provides a good solver for the ODEs of Challenge 21.1. Most ODE software provides a mechanism for stopping the integration when some quantity goes to zero; in `ode23s` this is done by using the `Events` property in an option vector.

For Challenge 21.2, some ODE software, including `ode23s`, can be used to solve some DAEs; in MATLAB, this is done using the `Mass` property in the option vector.

In MATLAB, some DDEs can be solved using `dde23`.

$$\frac{dR(t)}{dt} = I(t)/k\,.$$

We start the model by assuming some proportion of infected individuals; for example, $I(0) = 0.005$, $S(0) = 1 - I(0)$, $R(0) = 0$.

CHALLENGE 21.1.

Solve Model 1 using `ode23s` for $k = 4$ and $\tau = .8$ for $t > 0$ until either $I(t)$ or $S(t)$ drops below 10^{-5}. Plot $I(t)$, $S(t)$, and $R(t)$ on a single graph. Report the proportion of the population that became infected and the maximum difference between $I(t) + S(t) + R(t)$ and 1.

Instead of using the equation $dR/dt = I/k$, we could have used the conservation principle

$$I(t) + S(t) + R(t) = 1$$

for all time. Substituting this for the dR/dt equation gives us an equivalent system of differential-algebraic equations (DAEs), studied in Section 20.4, and we call this Model 2.

CHALLENGE 21.2.

Redo Challenge 21.1 using Model 2 instead of Model 1. One way to do this is to differentiate the conservation principle and express the three equations of the model as $\boldsymbol{M}\boldsymbol{u}' = \boldsymbol{f}(t, \boldsymbol{u})$, where $\boldsymbol{M}$ is a 3×3 matrix and $\boldsymbol{u}$ is a function of t with three components. Another way is to use a DAE formulation.

There are many limitations in the model, but one of them is that the recovery rate is proportional to the current number of infections. This means that we are not very faithful to the hypothesis that each individual is infectious for k days. One way to model this more closely is to use a **delay differential equation** (DDE). We modify Model 1 by specifying that the rate of recovery at time t is equal to the rate of new infections at time $t - k$. This gives us a new model, Model 3:

$$\frac{dI(t)}{dt} = \tau I(t)S(t) - \tau I(t-k)S(t-k)\,,$$

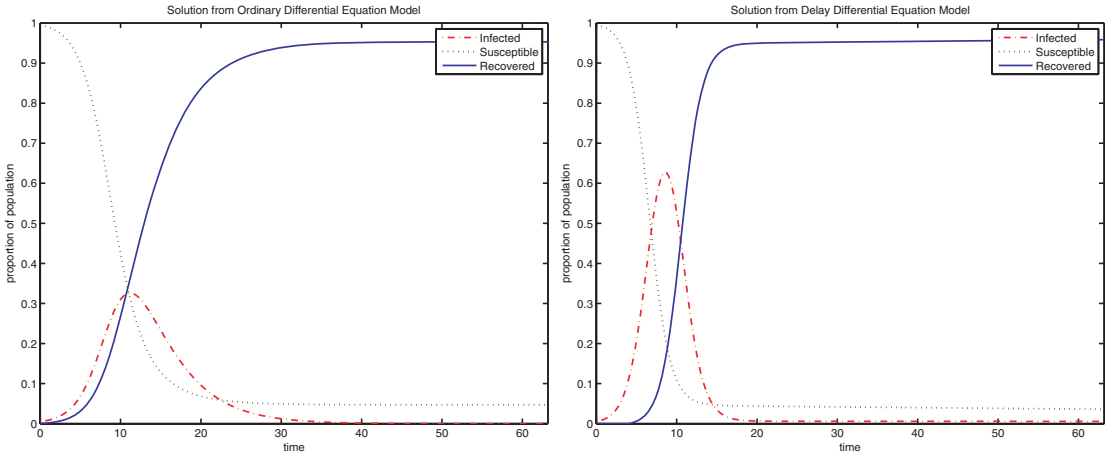


Figure 21.1. *Results of our models. (Left) Proportion of individuals infected by the epidemic from the* ODE *Model 1 or the* DAE *Model 2. (Right) Proportion of individuals infected by the epidemic from the* DDE *Model 3.*

$$\frac{dS(t)}{dt} = -\tau I(t)S(t),$$
$$\frac{dR(t)}{dt} = \tau I(t-k)S(t-k).$$

One disadvantage of this model is that we need to specify initial conditions not just at $t = 0$ but for $-k \le t \le 0$, so it requires a lot more information. A second disadvantage is that the functions I, S, and R are likely to have discontinuous derivatives (for example, at $t = 0$ and $t = k$, as we switch from dependence on the initial conditions to dependence only on the integration history). This causes solvers to do extra work at these points of discontinuity.

CHALLENGE 21.3.

Redo Challenge 21.1 with MATLAB's `dde23` using Model 3 instead of Model 1. For $t < 0$, use the initial conditions

$$I(t) = 0,\ S(t) = 1,\ R(t) = 0,$$

and let $I(0) = 0.005$, $S(0) = 1 - I(0)$, $R(0) = 0$. Note that these conditions match our previous ones, but have a jump at $t = 0$. Compare the results of the three models, as illustrated in Figure 21.1.

Models that Include Spatial Variation

Epidemics vary in space as well as time. They usually start in a single location and then spread, based on interaction of infected individuals with neighbors, as in the models of Chapter 19. The models of the previous section lose this characteristic. To recover it, we now let S, I, and R depend on a spatial coordinate (x, y) as well as t and see what such a model predicts.

Since people move in space, we introduce a **diffusion term** that allows infected individuals to affect susceptible individuals that are close to them in space. Diffusion adds a term $\delta((\partial^2 I)/(\partial x^2)+(\partial^2 I)/(\partial y^2))S$ to dI/dt, and subtracts the same term from dS/dt. This produces differential equations analogous to Model 1:

$$\begin{aligned}
\frac{\partial I(t,x,y)}{\partial t} &= \tau I(t,x,y)S(t,x,y) - I(t,x,y)/k \\
&\quad +\delta\left(\frac{\partial^2 I(t,x,y)}{\partial x^2}+\frac{\partial^2 I(t,x,y)}{\partial y^2}\right)S(t,x,y), \\
\frac{\partial S(t,x,y)}{\partial t} &= -\tau I(t,x,y)S(t,x,y) - \delta\left(\frac{\partial^2 I(t,x,y)}{\partial x^2}+\frac{\partial^2 I(t,x,y)}{\partial y^2}\right)S(t,x,y), \\
\frac{\partial R(t,x,y)}{\partial t} &= I(t,x,y)/k\,.
\end{aligned}$$

We assume that the initial values $I(0,x,y)$ and $S(0,x,y)$ are given, that we study the problem for $0 \le x \le 1$, $0 \le y \le 1$, and $t \ge 0$, and that there is no diffusion across the boundaries $x = 0$, $x = 1$, $y = 0$, and $y = 1$.

To solve this problem, Model 4, we **discretize** and approximate the solution at the points of a grid of size $n \times n$. Let $h = 1/(n-1)$ and let $x_i = ih$, $i = 0,\ldots,n-1$, and $y_j = jh$, $j = 0,\ldots,n-1$. Our variables are our approximations $I(t)_{ij} \approx I(t,x_i,y_j)$ and similarly for $S(t)_{ij}$ and $R(t)_{ij}$.

CHALLENGE 21.4.

(a) Use Taylor series expansions to show that we can approximate

$$\frac{\partial^2 I(t,x_i,y_j)}{\partial x^2} = \frac{I(t,x_{i-1},y_j) - 2I(t,x_i,y_j) + I(t,x_{i+1},y_j)}{h^2} + O(h^2)\,.$$

A similar expression can be derived for $\partial^2 I(t,x_i,y_j)/\partial y^2$.

(b) Form a vector $\widehat{\boldsymbol{I}}(t)$ from the approximate values of $I(t)$ by ordering the unknowns as I_{00}, $I_{01},\ldots,$ $I_{0,n-1}$, I_{10}, $I_{11},\ldots,I_{1,n-1},\ldots,$ $I_{n-1,0}$, $I_{n-1,1},\ldots,I_{n-1,n-1}$, where $I_{ij}(t) = I(t,x_i,y_j)$. In the same way, form the vectors $\widehat{\boldsymbol{S}}(t)$ and $\widehat{\boldsymbol{R}}(t)$ and derive the matrix $\boldsymbol{A}$ so that our discretized equations become Model 4:

$$\begin{aligned}
\frac{\partial \widehat{\boldsymbol{I}}(t)}{\partial t} &= \tau\widehat{\boldsymbol{I}}(t).*\widehat{\boldsymbol{S}}(t) - \widehat{\boldsymbol{I}}(t)/k + \delta(\boldsymbol{A}\widehat{\boldsymbol{I}}(t)).*\widehat{\boldsymbol{S}}(t), \\
\frac{\partial \widehat{\boldsymbol{S}}(t)}{\partial t} &= -\tau\widehat{\boldsymbol{I}}(t).*\widehat{\boldsymbol{S}}(t) - \delta(\boldsymbol{A}\widehat{\boldsymbol{I}}(t)).*\widehat{\boldsymbol{S}}(t), \\
\frac{\partial \widehat{\boldsymbol{R}}(t)}{\partial t} &= \widehat{\boldsymbol{I}}(t)/k\,,
\end{aligned}$$

where the notation $\widehat{\boldsymbol{I}}(t).*\widehat{\boldsymbol{S}}(t)$ means the vector formed by the product of each component of $\widehat{\boldsymbol{I}}(t)$ with the corresponding component of $\widehat{\boldsymbol{S}}(t)$. To form the approximation near the boundary, assume that the (Neumann) boundary conditions imply that $I(t,-h,y) = I(t,h,y)$,

$I(t,1+h,y) = I(t,1-h,y)$ for $0 \le y \le 1$, and similarly for S and R. Make the same type of assumption at the two other boundaries.

There are two ways to use this model. First, suppose we fix the timestep Δt and use Euler's method to approximate the solution; this means we approximate the solution at $t+\Delta t$ by the solution at t, plus Δt times the derivative at t. This gives us an iteration

$$\begin{aligned}
\widehat{\boldsymbol{I}}(t+\Delta t) &= \widehat{\boldsymbol{I}}(t) + \Delta t(\tau \widehat{\boldsymbol{I}}(t).*\widehat{\boldsymbol{S}}(t) - \widehat{\boldsymbol{I}}(t)/k + \delta(\boldsymbol{A}\widehat{\boldsymbol{I}}(t)).*\widehat{\boldsymbol{S}}(t)),\\
\widehat{\boldsymbol{S}}(t+\Delta t) &= \widehat{\boldsymbol{S}}(t) + \Delta t(-\tau \widehat{\boldsymbol{I}}(t).*\widehat{\boldsymbol{S}}(t) - \delta(\boldsymbol{A}\widehat{\boldsymbol{I}}(t)).*\widehat{\boldsymbol{S}}(t)),\\
\widehat{\boldsymbol{R}}(t+\Delta t) &= \widehat{\boldsymbol{R}}(t) + \Delta t(\widehat{\boldsymbol{I}}(t)/k).
\end{aligned}$$

This model is very much in the spirit of the models we considered in the case study of Chapter 19, except that it is deterministic rather than stochastic.

Alternatively, we could apply a more accurate ODE solver to this model, and we investigate this in the next challenge.

CHALLENGE 21.5.

(a) Set $n = 11$ (so that $h = 0.1$), $k = 4$, $\tau = 0.8$ and $\delta = 0.2$ and use an ODE solver to solve Model 4. For initial conditions, set $S(0,x,y) = 1$ and $I(0,x,y) = R(0,x,y) = 0$ at each point (x,y), except that $S(0,0.5,0.5) = I(0,0.5,0.5) = .5$. (For simplicity, you need only use I and S in the model, and you may derive $R(t)$ from these quantities.) Stop the simulation when either the average value of $\widehat{\boldsymbol{I}}(t)$ or $\widehat{\boldsymbol{S}}(t)$ drops below 10^{-5}. Form a plot similar to that of Challenge 21.1 by plotting the average value of $I(t)$, $S(t)$, and $R(t)$ vs time. Compare the results.

(b) Let's vaccinate the susceptible population at a rate

$$\frac{\nu S(t,x,y)I(t,x,y)}{I(t,x,y)+S(t,x,y)}.$$

This rate is the derivative of the vaccinated population $V(t,x,y)$ with respect to time, and this term is subtracted from $\partial S(t,x,y)/\partial t$. So now we model four segments of the population: susceptible $S(t)$, infected $I(t)$, recovered $R(t)$, and vaccinated $V(t)$. Your program can track three of these and derive the fourth from the conservation principle $S(t)+I(t)+R(t)+V(t) = 1$. Run this model with $\nu = 0.7$ and compare the results with those of Model 4.

POINTER 21.2. Further Reading.

Model 1 is the SIR model of Kermack and McKendrick, introduced in 1927. It is discussed, for example, by Britton [21].

DDEs such as those in Challenge 21.3 arise in many applications, including circuit analysis. To learn more about these problems, consult a textbook such as that by Bellman and Cooke [12] or by Hale and Lunel [67].

Stochastic differential equations are an active area of research. Higham [77] gives a good introduction to computational aspects and supplies references for further investigation.

The differential equations leading to Model 4 are presented, for example, by Callahan [24], following a model with one space dimension given in [87].

If you want to experiment further with Model 4, incorporate the delay recovery term in place of $-\widehat{I}(t)/k$.

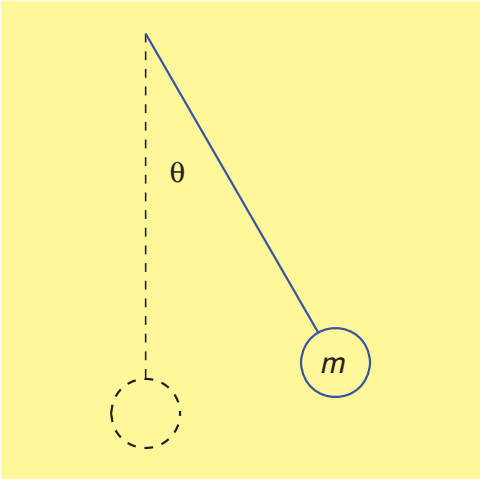
CHALLENGE 21.6. (Extra)

Include a delay in Model 4. Solve the resulting DDE model numerically and compare with the previous results.

In the models we used in the case study of Chapter 19, we incorporated some randomness to account for factors that were not explicitly modeled. We could also put randomness into our differential equation models, resulting in **stochastic differential equations**. See Pointer 21.2 for references on this subject.

Chapter 22 / Case Study

Robot Control: Swinging Like a Pendulum



(coauthored by Yalin E. Sagduyu)

Suppose we have a robot arm with a single joint, simply modeled as a damped driven pendulum. It is amazing how such a trivial system illustrates so many difficult concepts! In this project, we study the stability and behavior of this robot arm and develop a strategy to move the arm from one position to another using minimal energy.

The Model

We assume that the pendulum of length ℓ has a bob of mass m. Figure 22.1 shows the pendulum's position at some time t, with the variable $\theta(t)$ denoting the angle that the pendulum makes with the vertical axis at that time. The angle is measured in radians. The acceleration of the pendulum is proportional to the angular displacement from vertical, and we model the drag due to friction with the air as being proportional to velocity. This yields a second-order ordinary differential equation (ODE) for $t \geq 0$:

$$m\ell \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} + c\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} + mg\sin(\theta(t)) = u(t), \tag{22.1}$$

where g is the gravitational acceleration on an object at the surface of the earth, and c is the damping (frictional) constant. The term $u(t)$ defines the external force applied to the pendulum. In this project, we consider what happens in three cases: no external force, constant external force driving the pendulum to a final state, and then a force designed to minimize the energy needed to drive the robot arm from an initial position to an angle θ_f.

Stability and Controllability of the Robot Arm

The solution to equation (22.1) depends on relations among m, ℓ, c, g, and $u(t)$ and ranges from fixed amplitude oscillations for the **undamped** case ($c = 0$) to decays (oscillatory or

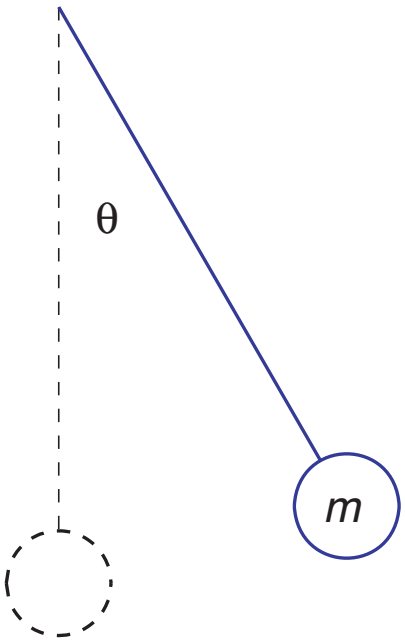


Figure 22.1. *We move this robot arm (pendulum), shown here at a position* $\theta(t)$.

strict) for the **damped** case ($c > 0$). Unfortunately, there is no simple analytical solution to the pendulum equation in terms of elementary functions unless we linearize the term $\sin(\theta(t))$ in (22.1) as $\theta(t)$, an approximation that is only valid for small values of $\theta(t)$. Despite the limitations of the linear approximation, the linearization enables us to find analytical solutions and also to apply the results of **linear control theory** to the specific problem of robot-arm control.

CHALLENGE 22.1.

Consider the **undriven damped pendulum** modeled by (22.1) with $u(t) = 0$ and $c > 0$. Linearize the second-order nonlinear differential equation using the approximation $\sin(\theta(t)) \approx \theta(t)$. Use the method in Section 20.1.1 to transform this equation into a first order system of ordinary differential equations of the form $\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y}$, where $\boldsymbol{A}$ is a 2×2 matrix, and the two components of the vector $\boldsymbol{y}(t)$ represent $y_{(1)}(t) = \theta(t)$, and $y_{(2)}(t) = \mathrm{d}\theta(t)/\mathrm{d}t$. Determine the eigenvalues of $\boldsymbol{A}$. Show that the damped system is **stable**, whereas the undamped system is not. (Recall from Section 20.1.2 that stability means that the real part of each eigenvalue of $\boldsymbol{A}$ is negative.) Use the eigenvalue information to show how the solutions behave in the damped and undamped systems.

Stability of the original differential equation (22.1) is more difficult to analyze than the stability of the linearized approximation to it, as we saw in Section 20.1.2. **Lyapunov stability** occurs when the total energy of an unforced (undriven), dissipative mechanical system decreases as the state of the system evolves in time. Therefore, the **state vector** $\boldsymbol{y}^T = [\theta(t), \mathrm{d}\theta(t)/\mathrm{d}t]$ approaches a constant value (**steady state**) corresponding to zero energy as time increases (i.e., $\boldsymbol{y}'(t) \to \boldsymbol{0}$ as $\boldsymbol{t} \to \infty$). According to Lyapunov's formulation,

the equilibrium point $\boldsymbol{y} = \boldsymbol{0}$ of a system described by the equation $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ is globally asymptotically stable if $\lim_{t\to\infty} \boldsymbol{y}(t) = \boldsymbol{0}$ for any choice of $\boldsymbol{y}(0)$.

Let $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ and let $\bar{\boldsymbol{y}}$ be a steady state solution of this differential equation. Terminology varies in the literature, but we use these definitions:

- A **positive definite Lyapunov function** v at $\bar{\boldsymbol{y}}(t)$ is a continuously differentiable function into the set of nonnegative numbers. It satisfies $v(\bar{\boldsymbol{y}}) = 0$, $v(\boldsymbol{y}(t)) > 0$ and

$$\frac{\mathrm{d}}{\mathrm{d}t} v(\boldsymbol{y}(t)) \leq 0$$

  for all $t > 0$ and all $\boldsymbol{y}$ in a neighborhood of $\bar{\boldsymbol{y}}$.

- An **invariant set** is a set for which the solution to the differential equation remains in the set when the initial state is in the set.

Suppose v is a positive definite Lyapunov function for a steady state solution $\bar{\boldsymbol{y}}$ of $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$. Then $\bar{\boldsymbol{y}}$ is **stable**. If in addition $\{\boldsymbol{y} : \mathrm{d}v(\boldsymbol{y}(t))/\mathrm{d}t = 0\}$ contains no invariant sets other than $\bar{\boldsymbol{y}}$, then $\bar{\boldsymbol{y}}$ is **asymptotically stable** [13]. This result guides our analysis.

Finding a Lyapunov function for a given problem can be difficult, but success yields important information. For unstable systems, small perturbations in the application of the external force can cause large changes in the behavior of the solution to the equation and therefore to the pendulum behavior, so the robot arm might behave erratically. Therefore, in practice we need to ensure that the system is stable.

CHALLENGE 22.2.

Consider the function

$$v(\theta, \mathrm{d}\theta/\mathrm{d}t) = \frac{(1 - \cos\theta)g}{\ell} + \frac{1}{2}\left(\frac{\mathrm{d}\theta}{\mathrm{d}t}\right)^2$$

for the pendulum described by (22.1). Show that it is a valid positive definite Lyapunov function for the undriven model. Investigate the stability of the solution $\theta(t) = 0, \mathrm{d}\theta(t)/\mathrm{d}t = 0$ for undamped systems and damped systems.

Consider the first-order system described by $\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u}$, where $\boldsymbol{A}$ is an $n \times n$ matrix and $\boldsymbol{B}$ is an $n \times m$ matrix. The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ may depend on time t, but in our example they do not. The system is **controllable** on $t \in [0, t_f]$ if given any initial state $\boldsymbol{y}(0)$ there exists a continuous function $\boldsymbol{u}(t)$ such that the solution of $\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u}$ satisfies $\boldsymbol{y}(t_f) = \boldsymbol{0}$. For controllability on any time interval, it is necessary and sufficient that the $n \times nm$ controllability matrix $[\boldsymbol{B}, \boldsymbol{A}\boldsymbol{B}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{B}]$ have rank n on that interval, and the rank can be computed using the methods in Chapter 5.

Controllability of the robot arm means that we can specify a force that drives it to any desired position. We investigate the controllability of the linearized pendulum model.

CHALLENGE 22.3.

Consider the linearized version of the driven (forced), damped pendulum system with constant force term u. Transform the corresponding differential equation to a first order ODE system of the form $\boldsymbol{y}' = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u}$. Specify the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and show that the system is controllable for both the damped and undamped cases.

Numerical Solution of the Initial Value Problem

We now develop some intuition for the behavior of the original model and the linearized model by comparing them under various experimental conditions.

For the numerical investigations in Challenges 22.4–22.6, assume that $m = 1$ kg, $\ell = 1$ m, and $g = 9.81\text{m/sec}^2$, with $c = 0$ for the undamped case and $c = 0.5$ kg-m/sec for the damped case.

First we investigate the effects of damping and of applied forces.

CHALLENGE 22.4.

For the initial conditions $\theta(0) = \pi/4$ and $\mathrm{d}\theta(0)/\mathrm{d}t = 0$, use an ODE solver to find the numerical solutions on the interval $t = [0, 30]$ for the nonlinear model (22.1) for

1. an undamped ($c = 0$), undriven ($u = 0$) pendulum,
2. a damped ($c > 0$), undriven ($u = 0$) pendulum, and
3. a damped ($c > 0$), driven pendulum with the applied forces $u = mg\sin(\theta_f)$, where $\theta_f = \pi/8, \pi/4, \pi/3$.

Repeat the same experiments for the linearized model of the pendulum and discuss the difference in behavior of the solutions. It helps to plot $\theta(t)$ for the corresponding linear and nonlinear models in the same figure, as in Figure 22.2.

Missing Data: Solution of the Boundary Value Problem

In Challenge 22.4, we solved the **initial value problem**, in which values of θ and $\mathrm{d}\theta/\mathrm{d}t$ were given at time $t = 0$. In many cases, we do not have the initial value for $\mathrm{d}\theta/\mathrm{d}t$; this value might not be **observable**. The missing initial condition prevents us from applying standard methods to solve initial value problems. Instead, we might have the value $\theta(t_B) = \theta_B$ at some other time t_B. Next we investigate two solution methods for this **boundary value problem**.

Recall from Section 20.5.1 that the idea behind the **shooting method** is to guess at the missing initial value $z = \mathrm{d}\theta(0)/\mathrm{d}t$, integrate equation (1) using our favorite method, and then use the results at the final time t_B to improve the guess. To do this systematically, we use a nonlinear equation solver to solve the equation $\rho(z) \equiv \theta_z(t_B) - \theta_B = 0$, where $\theta_z(t_B)$

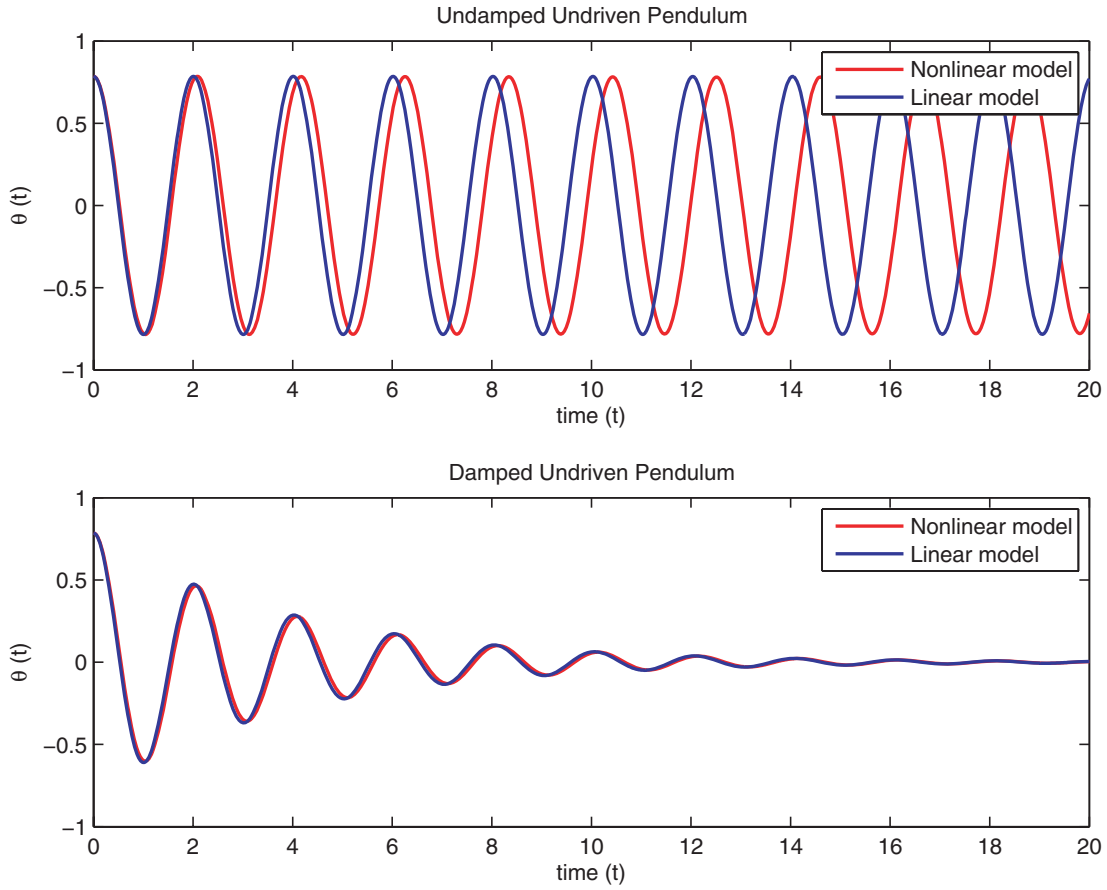


Figure 22.2. *The linear and nonlinear undriven models from Challenge 22.4.*

is the value reported by an initial value problem ODE solver for $\theta(t_B)$, given the initial condition $z = \mathrm{d}\theta(0)/\mathrm{d}t$.

The **finite difference method** of Section 20.5.2 is an alternate method to solve a boundary value problem. Choose a small time increment $h > 0$ and replace the first derivative in the linearized model of problem (1) by

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \approx \frac{\theta(t+h) - \theta(t-h)}{2h}$$

and second derivative by

$$\frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} \approx \frac{\theta(t+h) - 2\theta(t) + \theta(t-h)}{h^2}.$$

Let $n = t_B/h$, and write the equation for each value $\theta_j \approx \theta(jh)$, $j = 1,\ldots,n-1$. The **boundary conditions** can be stated as $\theta_0 = \theta(0)$, $\theta_n = \theta_B$. This method transforms the linearized version of the second-order differential equation (22.1) to a system of $n-1$ linear equations with $n-1$ unknowns. Assuming the solution to this linear system exists, we then use our favorite linear system solver to solve these equations.

CHALLENGE 22.5.

Consider the linearized model with constant applied force $u(t) = mg\sin(\pi/8)$ and damping constant $c = 0.5$. Suppose that we are given the boundary conditions $\theta(0) = \pi/32$ and $\theta(10) = \theta_B$, where θ_B is the value of the solution when $\mathrm{d}\theta(0)/\mathrm{d}t = 0$.

Apply the shooting method to find the solutions to the damped, driven linearized pendulum equation on the time interval $t = [0, 10]$. Try different initial guesses for $\mathrm{d}\theta(0)/\mathrm{d}t$ and compare the results.

Now use the finite difference method to solve this boundary value problem with $h = 0.01$. Use your favorite linear system solver to solve the resulting linear system of equations.

Compare the results of the shooting and finite difference methods with the solution to the original initial value problem.

Controlling the Robot Arm

Finally, we investigate how to design a forcing function that drives the robot arm from an initial position to some other desired position with the least expenditure of energy. We measure energy as the integral of the absolute force applied between time 0 and the convergence time t_c at which the arm reaches its destination:

$$e_f = \int_0^{t_c} |u(t)|\mathrm{d}t\,.$$

CHALLENGE 22.6.

Consider the damped, driven pendulum with applied force

$$u(t) = mg\sin(\theta_f) + m\ell b\,\mathrm{d}\theta(t)/\mathrm{d}t,$$

where $\theta_f = \pi/3$. This force is a particular **closed-loop control** with control parameter b, and it drives the pendulum position to θ_f. The initial conditions are given as $\theta(0) = \pi/4$ and $\mathrm{d}\theta(0)/\mathrm{d}t = 0$. Assume $c = 0.5$ as the damping constant, $t_c = 5$ sec as the time limit for achieving the position θ_f, and $h = 0.01$ as the time step length for numerical solutions.

We call a parameter b successful if the pendulum position satisfies $|\theta(t) - \theta_f| < 10^{-3}$ for $5 \le t \le 10$. Approximate the total energy by

$$\hat{e}_f \approx \sum_{k=1}^{5/h} |u(kh)|h\,.$$

Write a function that evaluates $\hat{e}_f$. The input to the function should be the control parameters b and the output should be the approximate total consumed energy $\hat{e}_f$.

For stability of the closed-loop control system, we impose the constraint $b < c/(m\ell)$, which make the real parts of the eigenvalues (of the linearized version) of the system strictly negative.

POINTER 22.1. Further Reading.

For Challenge 22.1, consult Section 20.1.1 for converting second derivative equations to a system of equations involving only first derivatives. An elementary linear algebra textbook [102] discusses computation of the eigenvalues and eigenvectors of a 2×2 matrix, and Section 20.1.2 gives an example of how to solve linear ODE systems once the eigensystem is known.

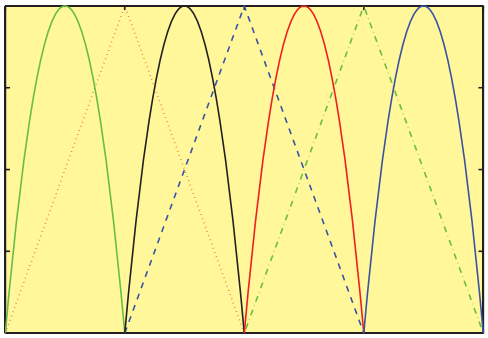
An ODE textbook [101] can serve as a reference on Lyapunov stability, used in Challenge 22.2. Control theory textbooks [116, 128] discuss stability plus the concept of controllability used in Challenge 22.3.

Challenge 22.6 relies on an ODE solver and a function for minimization of a function of a single variable under bound constraints (e.g., MATLAB's `fminbnd`).

Now use your favorite constrained minimization solver to select the control parameter b to minimize the energy function $\hat{e}_f(b)$. Display the optimal parameter and graph the resulting $\theta(t)$.

Chapter 23 / Case Study

Finite Differences and Finite Elements: Getting to Know You



Numerical solution of differential equations relies on two main methods: finite differences and finite elements. In this case study, we explore the nuts and bolts of the two methods for a simple **two-point boundary value problem**:

$$-(a(t)u'(t))' + c(t)u(t) = f(t) \text{ for } t \in (0,1),$$

with the functions a, c, and f given and $u(0) = u(1) = 0$. We assume that $a(t) \geq a_0$, where a_0 is a positive number, and $c(t) \geq 0$ for $t \in [0,1]$. Check Pointer 20.8 for conditions that guarantee the existence of a unique solution for our problem.

The Finite Difference Method

Let's rewrite our equation as

$$-a(t)u''(t) - a'(t)u'(t) + c(t)u(t) = f(t) \tag{23.1}$$

and approximate each derivative of u by a finite difference:

$$u'(t) = \frac{u(t) - u(t-h)}{h} + O(h),$$
$$u''(t) = \frac{u(t-h) - 2u(t) + u(t+h)}{h^2} + O(h^2).$$

(We'll compute $a'(t)$ analytically, so we won't need an approximation to it.)

The finite difference approach is to choose **grid points** $t_j = jh$, where $h = 1/(M-1)$ for some large integer M, and solve for $u_j \approx u(t_j)$ for $j = 1, \ldots, M-2$. We write one equation for each unknown, by substituting our finite difference approximations for u'' and u' into (23.1), and then evaluating the equation at $t = t_j$.

CHALLENGE 23.1.

Let $M = 6$, $a(t) = 1$, and $c(t) = 0$ and write the four finite difference equations for u at $t = .2, .4, .6$, and $.8$.

POINTER 23.1. Some Notes on the Challenges.

To debug your programs, it is helpful to experiment with the simplest test problem and a small number of grid points. Look ahead to Challenge 23.6 for sample problems.

Challenge 23.2 uses the MATLAB function `spdiags` to construct a sparse matrix. If you have never used sparse matrices in MATLAB, print the matrix `A` to see that the data structure for it contains the row index, column index, and value for each nonzero element. If you have never used `spdiags`, type `help spdiags` to see the documentation, and then try it on your own data to see exactly how the matrix elements are defined.

Use MATLAB's `quad` to compute the integrals for the entries in the matrix and right-hand side for the finite element formulations.

Before tackling the programming for Challenge 23.5 and 23.6, take some time to understand exactly where the nonzeros are in the matrix, and exactly what intervals of integration should be used. The programs are short, but it is easy to make mistakes if you don't understand what they compute.

In Challenge 23.7, we measure **work** by counting the number of multiplications. One alternative is to count the number of floating-point computations, but this usually gives a count of about twice the number of multiplications, since typically multiplications and additions are paired in computations. Computing time is another very useful measure of work, but it can be contaminated by the effects of other users or other processes on the computer.

In determining and understanding the convergence rate in Challenge 23.7, plotting the solutions or the error norms might be helpful.

Notice that the matrix constructed in Challenge 23.1 is tridiagonal. (See Pointer 5.2.) The full matrix requires $(M-2)^2$ storage locations, but, if we are careful, we can instead store all of the data in $O(M)$ locations by agreeing to store only the nonzero elements, along with their row and column indices. This is a standard technique for storing **sparse matrices**, those whose elements are mostly zero; see Chapter 27.

Let's see how this finite difference method is implemented.

CHALLENGE 23.2.

The MATLAB function `finitediff1.m`, found on the website, implements the finite difference method for our equation. The inputs are the parameter M and the functions `a`, `c`, and `f` that define the equation. Each of these functions takes a vector of points as input and returns a vector of function values. (The function `a` also returns a second vector of values of a'.) The outputs of `finitediff1.m` are a vector `ucomp` of computed estimates of u at the grid points `tmesh`, along with the matrix `A` and the right-hand side `g` from which `ucomp` was computed, so that `A ucomp = g`.

Add documentation to the function `finitediff1.m` so that a user could easily use it, understand the method, and modify the function if necessary. (See Section 4.1 for advice on documentation.)

There is a mismatch in `finitediff1.m` between our approximation to u'', which is second order in h, and our approximation to u', which is only first order. We can compute a better solution, for the same cost, by using a second-order (central difference) approximation to u', so next we make this change to our function.

CHALLENGE 23.3.

Define a central difference approximation to the first derivative by

$$u'(t) \approx \frac{u(t+h) - u(t-h)}{2h}.$$

Modify the function of Challenge 23.2 to produce a function `finitediff2.m` that uses this second-order accurate central difference approximation (studied in Challenge 20.19) in place of the first-order approximation.

The Finite Element Method

We'll use a **Galerkin method** with finite elements to solve our problem. In particular, we notice that

$$-(a(t)u'(t))' + c(t)u(t) = f(t) \text{ for } t \in (0,1)$$

implies that

$$\int_0^1 (-(a(t)u'(t))' + c(t)u(t))v(t)\,\mathrm{d}t = \int_0^1 f(t)v(t)\,\mathrm{d}t$$

for all functions v. Now we use integration by parts on the first term, recalling that our boundary values are zero, to obtain the equation

$$\int_0^1 a(t)u'(t)v'(t) + c(t)u(t)v(t)\,\mathrm{d}t = \int_0^1 f(t)v(t)\,\mathrm{d}t,$$

for all functions v. If a, c, and f are smooth functions (i.e., their first few derivatives exist), then the solution to our differential equation satisfies the boundary conditions and has a first derivative, with the integral of $(u'(t))^2$ on $[0,1]$ finite. We call the space of all such functions H_0^1, and that is also the space we draw v from.

How does this help us solve the differential equation? We choose a subspace S_h of H_0^1 that contains functions that are good approximations to every function in H_0^1, and we look for a function $u_h \in S_h$ so that

$$\int_0^1 a(t)u_h'(t)v_h'(t) + c(t)u_h(t)v_h(t)\,\mathrm{d}t = \int_0^1 f(t)v_h(t)\,\mathrm{d}t$$

for all functions $v_h \in S_h$. This gives us an approximate solution to our problem.

A common choice for S_h is the set of functions that are continuous and linear on each interval $[t_{j-1}, t_j] = [(j-1)h, jh]$, $j = 1, \ldots, M-1$ (piecewise linear elements), where

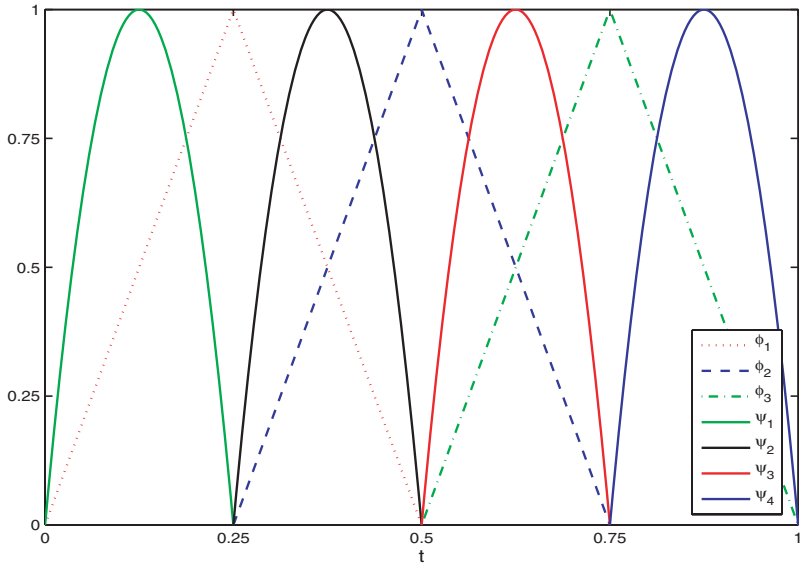


Figure 23.1. *The nonzero pieces of the three linear (ϕ) and four quadratic (ψ) basis functions for three interior grid points and four subintervals ($M = 5$).*

$h = 1/(M-1)$. We can construct our solution using any basis for S_h, but one basis is particularly convenient: the set of **hat functions** ϕ_j, $j = 1, \ldots, M-2$, where

$$\phi_j(t) = \begin{cases} \frac{t - t_{j-1}}{t_j - t_{j-1}}, & t \in [t_{j-1}, t_j], \\ \frac{t - t_{j+1}}{t_j - t_{j+1}}, & t \in [t_j, t_{j+1}], \\ 0 & \text{otherwise}. \end{cases}$$

These are designed to satisfy $\phi_j(t_j) = 1$ and $\phi_j(t_k) = 0$ if $j \neq k$; see Figure 23.1 for an illustration. Note that ϕ_j is nonzero only on the interval (t_{j-1}, t_{j+1}) (i.e., it has **small support**), but it is defined everywhere.

Then we can express our approximate solution u_h as

$$u_h(t) = \sum_{j=1}^{M-2} u_j \phi_j(t)$$

for some coefficients u_j, which happen to be approximate values for $u(t_j)$.

Define

$$\mathrm{a}(u, v) = \int_0^1 (a(t)u'(t)v'(t) + c(t)u(t)v(t))\, \mathrm{d}t,$$
$$(f, v) = \int_0^1 f(t)v(t)\, \mathrm{d}t.$$

Then our solution u satisfies

$$\mathrm{a}(u, v) = (f, v)$$

for all $v \in H_0^1$, and we demand that our approximate solution $u_h \in S_h$ satisfy

$$\mathrm{a}(u_h, v_h) = (f, v_h)$$

for all $v_h \in S_h$. In Challenge 23.4, we reduce this to a linear system of equations that can be solved for the coefficients u_j, and we implement our ideas in Challenge 23.5.

CHALLENGE 23.4.

(a) Since the functions ϕ_j form a basis for S_h, any function $v_h \in S_h$ can be written as

$$v_h(t) = \sum_{j=1}^{M-2} v_j \phi_j(t)$$

for some coefficients v_j. Show that if

$$\mathrm{a}(u_h, \phi_j) = (f, \phi_j) \tag{23.2}$$

for $j = 1, \ldots, M-2$, then

$$\mathrm{a}(u_h, v_h) = (f, v_h)$$

for all $v_h \in S_h$.

(b) Putting the unknowns u_j in a vector $\boldsymbol{u}$ we can write the system of equations resulting from (23.2) as $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{g}$, where the (j,k) entry in $\boldsymbol{A}$ is $\mathrm{a}(\phi_j, \phi_k)$ and the jth entry in $\boldsymbol{g}$ is (f, ϕ_j). Write this system of equations for $M = 6$, $a(t) = 1$, $c(t) = 0$, and compare with your solution to Challenge 23.1.

CHALLENGE 23.5.

Write a function `fe_linear.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise linear elements. Remember to store $\boldsymbol{A}$ as a sparse matrix.

It can be shown that the computed solution is within $O(h^2)$ of the exact solution, if the data is smooth enough. Better accuracy can be achieved if we use **higher-order elements**; for example, piecewise quadratic elements would produce a result within $O(h^3)$ for smooth data. A convenient basis for this set of elements is the piecewise linear basis plus $M-1$ quadratic functions ψ_j that are zero outside $[t_{j-1}, t_j]$ and satisfy

$$\begin{aligned} \psi_j(t_j) &= 0, \\ \psi_j(t_{j-1}) &= 0, \\ \psi_j(t_{j-1} + h/2) &= 1 \end{aligned}$$

for $j = 1, \ldots, M-1$. See Figure 23.1 for an illustration.

CHALLENGE 23.6.

Write a function `fe_quadratic.m` that has the same inputs and outputs as `finitediff1.m` but computes the finite element approximation to the solution using piecewise quadratic elements. In order to keep the number of unknowns comparable to the number in the previous functions, let the number of intervals be $m = \lfloor M/2 \rfloor$. When $M = 10$, for example, we have 5 quadratic basis functions (one for each subinterval) and 4 linear ones (one for each interior grid point). If you order the basis elements as $\psi_1, \phi_1, \ldots, \psi_{m-1}, \phi_{m-1}, \psi_m$ then the matrix $\boldsymbol{A}$ has 5 nonzero bands including the main diagonal. Compute one additional output `uval` which is the finite element approximation to the solution at the $m-1$ interior grid points and the m midpoints of each interval, where the $2m-1$ equally spaced points are ordered smallest to largest. (In our previous methods, this was equal to `ucomp`, but now the values at the midpoints of the intervals are a linear combination of the linear and quadratic elements.)

Now we have four solution algorithms, so we define a set of functions for experimentation:

$$
\begin{aligned}
u_1(t) &= t(1-t)e^t, \\
u_2(t) &= \begin{cases} u_1(t) & \text{if } t \le 2/3, \\ t(1-t)e^{2/3} & \text{if } t > 2/3, \end{cases} \\
u_3(t) &= \begin{cases} u_1(t) & \text{if } t \le 2/3, \\ t(1-t) & \text{if } t > 2/3, \end{cases} \\
a_1(t) &= 1, \\
a_2(t) &= 1+t^2, \\
a_3(t) &= \begin{cases} a_2(t) & \text{if } t \le 1/3, \\ t+7/9 & \text{if } t > 1/3, \end{cases} \\
c_1(t) &= 0, \\
c_2(t) &= 2, \\
c_3(t) &= 2t.
\end{aligned}
$$

For each particular choice of u, a, and c, we define f using (23.1).

CHALLENGE 23.7.

Use your four algorithms to solve 7 problems:

- a_1 with c_j $(j = 1,2,3)$ and true solution u_1.
- a_j $(j = 2,3)$ with c_1 and true solution u_1.
- a_1 and c_1 with true solution u_j $(j = 2,3)$.

Compute three approximations for each algorithm and each problem, with the number of unknowns in the problem chosen to be 9, 99, and 999. For each approximation, print

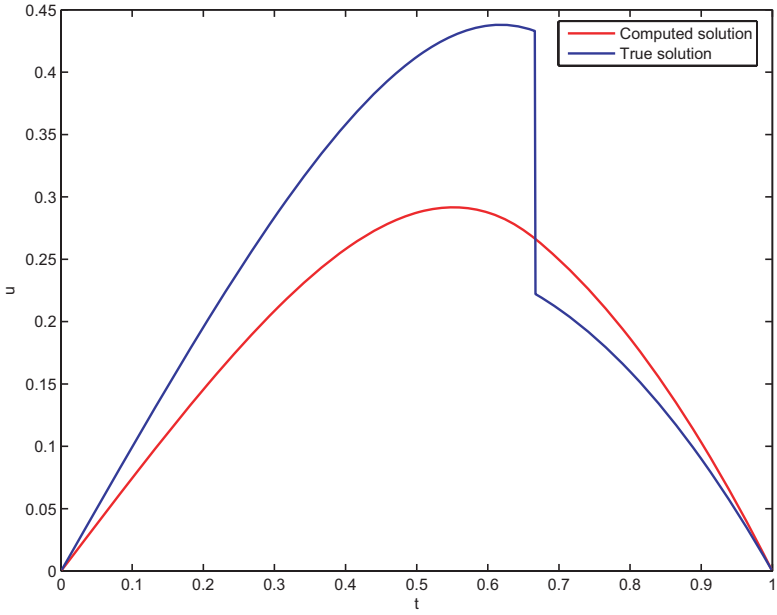


Figure 23.2. *The "solution" to the seventh test problem. We compute an accurate answer to a different problem.*

$\|\boldsymbol{u}_{computed} - \boldsymbol{u}_{true}\|_\infty$, where $\boldsymbol{u}_{true}$ is the vector of true values at the points jz, where $z = 1/10$, $1/100$, or $1/1000$, respectively.

Discuss the results:

- How easy is it to program each of the four methods? Estimate how much work MATLAB does to form and solve the linear systems. (The work to solve the tridiagonal systems should be about $5M$ multiplications, and the work to solve the 5-diagonal systems should be about $11M$ multiplications, so you just need to estimate the work in forming each system.)
- For each problem, note the observed convergence rate r: if the error drops by a factor of 10^r when M is increased by a factor of 10, then the observed convergence rate is r.
- Explain any deviations from the theoretical convergence rate: $r = 1$ and $r = 2$ for the two finite difference implementations, and $r = 2$ and $r = 3$ for the finite element implementations when measuring $(u - u_h, u - u_h)^{1/2}$. The solution to the last problem is shown in Figure 23.2.

In doing this work, we begin to understand the complexities of implementation of finite difference and finite element methods. We have left out many features that a practical implementation should contain. In particular, the algorithm should be adaptive, estimating the error on each grid interval and subdividing the intervals (or raising the order of polynomials) where the error is too high. This method can handle partial differential equations, too. Luckily, there are good implementations of these methods for two- and three-dimensional domains, so we don't need to write our own.

POINTER 23.2. Further Reading.

A good introduction to the theory of finite difference and finite element methods is given by Gockenbach [59]; for a more advanced treatment, see, for example, Larsson and Thomée [99, Chap. 2, Section 4.1, Section 5.1].