CHAPTER 15

Ordinary Differential Equations

CONTENTS

CHAPTER 13 motivated the problem of interpolation by transitioning from *analyzing* functions to *finding* functions. In problems like interpolation and regression, the unknown is a entire function $f(\vec{x})$, and the job of the algorithm is to fill in $f(\vec{x})$ at positions $\vec{x}$ where it is unknown.

In this chapter and the next, our unknown will continue to be a function f, but rather than filling in missing values we will solve more complex design problems like the following:

- Find f approximating some other function f_0 but satisfying additional criteria (smoothness, continuity, boundedness, etc.).
- Simulate some dynamical or physical relationship as $f(t)$ where t is time.
- Find f with similar values to f_0 but certain properties in common with a different function g_0.

In each of these cases, our unknown is a function f, but our criterion for success is more involved than "matches a given set of data points."

The theories of ordinary differential equations (ODEs) and partial differential equations (PDEs) involve the case where we wish to find a function $f(\vec{x})$ based on information about

or relationships between its derivatives. We inadvertently solved one problem in this class while studying quadrature: Given $f'(t)$, quadrature approximates $f(t)$ using integration.

In this chapter, we will consider *ordinary* differential equations and in particular *initial value problems*. In these problems, the unknown is a function $f(t) : \mathbb{R} \to \mathbb{R}^n$, given $f(0)$ and an equation satisfied by f and its derivatives. Our goal is to predict $f(t)$ for $t > 0$. We will provide examples of ODEs appearing in practice and then will describe common solution techniques.

15.1 MOTIVATION

ODEs appear in nearly every branch of science, and hence it is not difficult to identify target applications of solution techniques. We choose a few representative examples both from the computational and scientific literatures:

Example 15.1 (Newton's Second Law)**.** Continuing from §6.1.2, recall that Newton's Second Law of Motion states $\vec{F} = m\vec{a}$, that is, the total force on an object is equal to its mass times its acceleration. If we simulate n particles simultaneously as they move in three-dimensional space, we can combine all their positions into a single vector $\vec{x}(t) \in \mathbb{R}^{3n}$. Similarly, we can write a function $\vec{F}(t, \vec{x}, \vec{x}') \in \mathbb{R}^{3n}$ taking the current time, the positions of the particles, and their velocities and returning the total force on each particle divided by its mass. This function can take into account interrelationships between particles (e.g., gravitational forces, springs, or intermolecular bonds), external effects like wind resistance (which depends on $\vec{x}'$), external forces varying with time t, and so on. To find the positions of all the particles as functions of time, we can integrate Newton's second law forward in time by solving the equation $\vec{x}'' = \vec{F}(t, \vec{x}, \vec{x}')$. We usually are given the positions and velocities of all the particles at time $t = 0$ as a starting condition.

Example 15.2 (Protein folding)**.** On a small scale, the equations governing motions of molecules stem from Newton's laws or—at even smaller scales—the Schrödinger equation of quantum mechanics. One challenging case is that of *protein folding*, in which the geometric structure of a protein is predicted by simulating intermolecular forces over time. These forces take many nonlinear forms that continue to challenge researchers in computational biology due in large part to a variety of *time scales*: The same forces that cause protein folding and related phenomena also can make molecules vibrate rapidly, and the disparate time scales of these two different behaviors makes them difficult to capture simultaneously.

Example 15.3 (Gradient descent)**.** Suppose we wish to minimize an objective function $E(\vec{x})$ over all $\vec{x}$. Especially if E is a convex function, the most straightforward option for minimization from Chapter 9 is gradient descent with a constant step size or "learning rate." Since $-\nabla E(\vec{x})$ points in the direction along which E decreases the most from a given $\vec{x}$, we can iterate:

$$\vec{x}_{i+i} \equiv \vec{x}_i - h\nabla E(\vec{x}_i),$$

for fixed $h > 0$. We can rewrite this relationship as

$$\frac{\vec{x}_{i+1} - \vec{x}_i}{h} = -\nabla E(\vec{x}_i).$$

In the style of §14.3, we might think of $\vec{x}_k$ as a sample of a function $\vec{x}(t)$ at $t = hk$. Heuristically, taking $h \to 0$ motivates an ordinary differential equation

$$\vec{x}'(t) = -\nabla E(\vec{x}).$$

If we take $\vec{x}(0)$ to be an initial guess of the location where $E(\vec{x})$ is minimized, then this ODE is a continuous model of gradient descent. It can be thought of as the equation of a path smoothly walking "downhill" along a landscape provided by E.

For example, suppose we wish to solve $A\vec{x} = \vec{b}$ for symmetric positive definite A. From §11.1.1, this is equivalent to minimizing $E(\vec{x}) \equiv \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top\vec{x} + c$. Using the continuous model of gradient descent, we can instead solve the ODE $\vec{x}' = -\nabla E(\vec{x}) = \vec{b} - A\vec{x}$. As $t \to \infty$, we expect $\vec{x}(t)$ to better and better satisfy the linear system.

Example 15.4 (Crowd simulation)**.** Suppose we are writing video game software requiring realistic simulation of virtual crowds of humans, animals, spaceships, and the like. One way to generate plausible motion is to use differential equations. In this technique, the velocity of a member of the crowd is determined as a function of its environment; for example, in human crowds, the proximity of other humans, distance to obstacles, and so on can affect the direction a given agent is moving. These rules can be simple, but in the aggregate their interaction becomes complex. Stable integrators for differential equations underlie crowd simulation to avoid noticeably unrealistic or unphysical behavior.

15.2 THEORY OF ODES

A full treatment of the theory of ordinary differential equations is outside the scope of our discussion, and we refer the reader to [64] or any other basic text for details from this classical theory. We highlight relevant results here for development in future sections.

15.2.1 Basic Notions

The most general initial value problem takes the following form:

$$\begin{array}{rl} \text{Find} & f(t) : \mathbb{R}^+ \to \mathbb{R}^n \\ \text{satisfying} & F[t, f(t), f'(t), f''(t), \ldots, f^{(k)}(t)] = \vec{0} \\ \text{given} & f(0), f'(0), f''(0), \ldots, f^{(k-1)}(0). \end{array}$$

Here, F is some relationship between f and all its derivatives; we use $f^{(\ell)}$ to denote the ℓ-th derivative of f. The functions f and F can be *multidimensional*, taking on values in $\mathbb{R}^n$ rather than $\mathbb{R}$, but by convention and for convenience of notation we will omit the vector sign. We also will use the notation $\vec{y} \equiv f(t)$ as an alternative to writing $f(t)$ when the t dependence is implicit; in this case, derivatives will be notated $\vec{y}' \equiv f'(t)$, $\vec{y}'' \equiv f''(t)$, and so on.

Example 15.5 (Canonical ODE form)**.** Suppose we wish to solve the ODE $y'' = ty' \cos y$. In the general form above, the ODE can be written $F[t, y, y', y''] = 0$, where $F[t, a, b, c] \equiv tb \cos a - c$.

ODEs determine the *evolution* of f over time t; we know f and its derivatives at time $t = 0$ and wish to predict these quantities moving forward. They can take many forms even in a single variable. For instance, denote $y = f(t)$ for $y \in \mathbb{R}^1$. Then, examples of ODEs include the following:

Example ODE	Distinguishing properties
$y' = 1 + \cos t$	Can be solved by integrating both sides with respect to t; can be solved discretely using quadrature
$y' = ay$	Linear in y, no dependence on time t
$y' = ay + e^t$	Time- and value-dependent
$y'' + 3y' - y = t$	Involves multiple derivatives of y
$y'' \sin y = e^{ty'}$	Nonlinear in y and t

We will restrict most of our discussion to the case of *explicit* ODEs, in which the highest-order derivative can be isolated:

Definition 15.1 (Explicit ODE)**.** An ODE is *explicit* if can be written in the form

$$f^{(k)}(t) = F[t, f(t), f'(t), f''(t), \ldots, f^{(k-1)}(t)].$$

Certain *implicit* ODEs can be converted to explicit form by solving a root-finding problem, for example, using the machinery introduced in Chapter 8, but this approach can fail in the presence of multiple roots.

Generalizing a trick first introduced in §6.1.2, any explicit ODE can be converted to a first-order equation $f'(t) = F[t, f(t)]$ by adding to the dimensionality of f. This construction implies that it will be enough for us to consider algorithms for solving (multivariable) ODEs containing only a single time derivative. As a reminder of this construction for the second-order ODE $y'' = F[t, y, y']$, recall that

$$\frac{d^2y}{dt^2} = \frac{d}{dt}\left(\frac{dy}{dt}\right).$$

Defining an intermediate variable $z \equiv {}^{dy}/_{dt}$, we can expand to the following first-order system:

$$\frac{d}{dt}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} z \\ F[t, y, z] \end{pmatrix}.$$

More generally, if we wish to solve the explicit problem

$$f^{(k)}(t) = F[t, f(t), f'(t), f''(t), \ldots, f^{(k-1)}(t)]$$

for $f : \mathbb{R}^+ \to \mathbb{R}^n$, then instead we can solve the first-order ODE in dimension $n(k+1)$:

$$\frac{d}{dt}\begin{pmatrix} f_0(t) \\ f_1(t) \\ f_2(t) \\ \vdots \\ f_{k-1}(t) \end{pmatrix} = \begin{pmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ \vdots \\ F[t, f_0(t), f_1(t), \ldots, f_{k-1}(t)] \end{pmatrix}.$$

Here, we denote $f_i(t) : \mathbb{R} \to \mathbb{R}^n$ as the i-th derivative of $f_0(t)$, which satisfies the original ODE. To check, our expanded system above implies $f_1(t) = f_0'(t)$, $f_2(t) = f_1'(t) = f_0''(t)$, and so on; the final row encodes the original ODE.

This trick simplifies notation and allows us to emphasize first-order ODEs, but some care should be taken to understand that it does come with a cost. The expansion above replaces ODEs with potentially many derivatives with ODEs containing just one derivative but with much higher dimensionality. We will return to this trade-off between dimensionality and number of derivatives when designing methods specifically for second-order ODEs in §15.4.2.

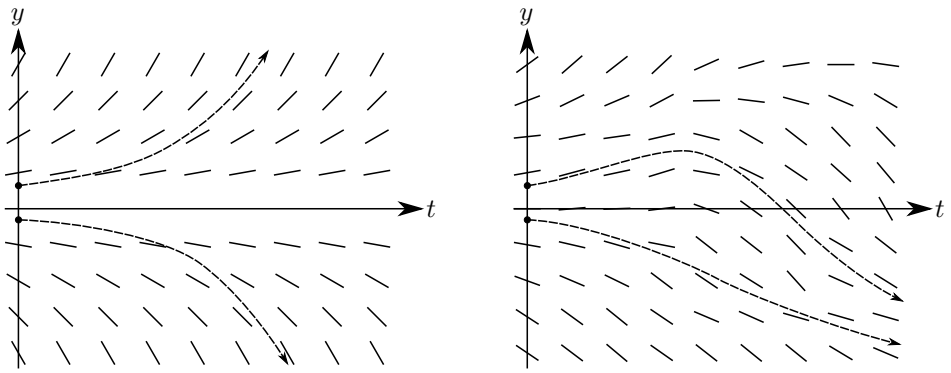


Figure 15.1 First-order ODEs in one variable $y' = F[t, y]$ can be visualized using *slope fields* on the (t, y) plane. Here, short line segments show the slope $F[t, y]$ at each sampled point; solution curves $y(t)$ shown as dotted lines start at $(0, y(0))$ and follow the slope field as their tangents. We show an example of a time-independent ("autonomous") ODE $y' = F[y]$ and an example of a time-dependent ODE $y' = F[t, y]$.

Example 15.6 (ODE expansion)**.** Suppose we wish to solve $y''' = 3y'' - 2y' + y$ where $y(t) : \mathbb{R}^+ \to \mathbb{R}$. This equation is equivalent to:

$$\frac{d}{dt}\begin{pmatrix} y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -2 & 3 \end{pmatrix}\begin{pmatrix} y \\ z \\ w \end{pmatrix}.$$

In the interests of making our canonical ODE problem as simple as possible, we can further restrict our consideration to *autonomous* ODEs. These equations are of the form $f'(t) = F[f(t)]$, that is, F has no dependence on t (or on higher-order derivatives of f, removed above). To reduce an ODE to this form, we use the fact $d/dt(t) = 1$. After defining a trivial function $g(t) = t$, the ODE $f'(t) = F[t, f(t)]$ can be rewritten as the autonomous equation

$$\frac{d}{dt}\begin{pmatrix} g(t) \\ f(t) \end{pmatrix} = \begin{pmatrix} 1 \\ F[g(t), f(t)] \end{pmatrix},$$

with an additional initial condition $g(0) = 0$.

It is possible to visualize the behavior and classification of low-dimensional ODEs in many ways. If the unknown $f(t)$ is a function of a single variable, then $F[f(t)]$ provides the slope of $f(t)$, as shown in Figure 15.1. For higher-order ODEs, it can be useful to plot $f(t)$ and its derivatives, shown for the equation of motion for a pendulum in Figure 15.2. In higher dimensions, it may be possible only to show example solution paths, as in Figure 15.3.

15.2.2 Existence and Uniqueness

Before we discretize the initial value ODE problem, we should acknowledge that not all differential equations are solvable, while others admit infinitely many solutions. Existence and uniqueness of ODE solutions can be challenging to prove, but without these properties

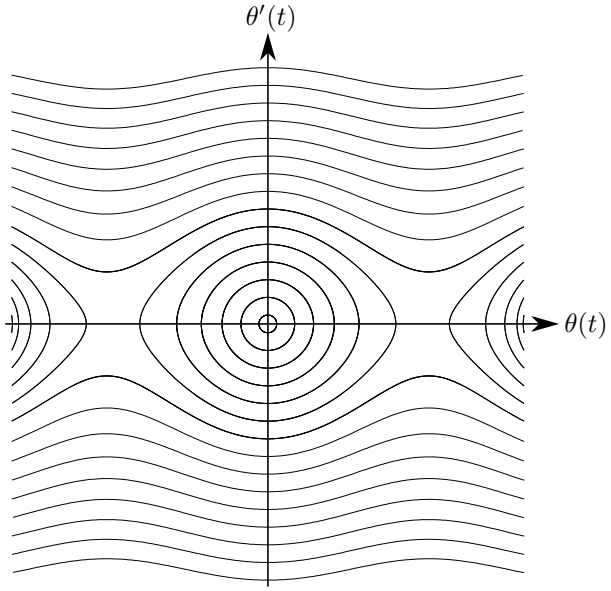


Figure 15.2 The *phase space diagram* of a pendulum, which satisfies the ODE $\theta'' = -\sin\theta$. Here, the horizontal axis shows position θ of the pendulum as it swings (as an angle from vertical), and the vertical axis shows the angular velocity θ'. Each path represents the motion of a pendulum with different starting conditions; the time t is not depicted. Rings indicate a swinging pendulum, while waves indicate that the pendulum is doing complete revolutions.

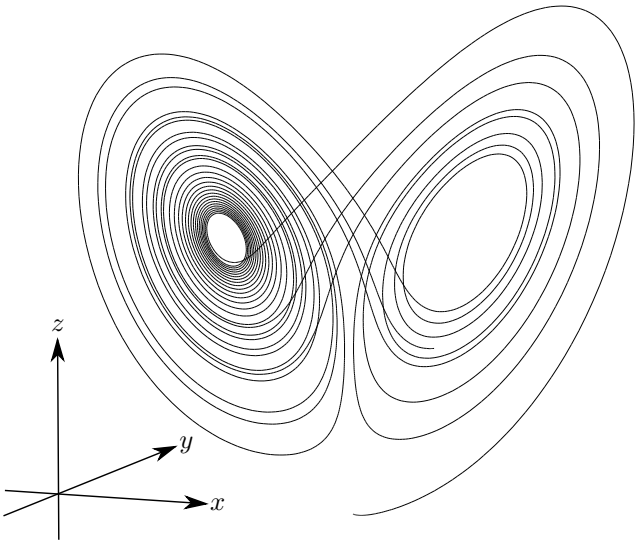


Figure 15.3 The *trace* of an ODE solution $(x(t), y(t), z(t))$ shows typical behavior without showing the velocity of the path or dependence on time t; here we show a solution to the *Lorenz equations* (known as a "Lorenz attractor") $x' = \sigma(y-x), y' = x(\rho - z) - y, z' = xy - \beta z$ integrated numerically ($\rho = 28, \sigma = 10, \beta = 8/3$).

we cannot hold numerical methods responsible for failure to recover a reasonable solution. Numerical ODE solvers can be thought of as filling the gap between knowing that a solution to a differential equation exists and being able to write this solution in closed form; checking existence and uniqueness is largely a function of how an ODE is written *before* discretization and usually is checked theoretically rather than algorithmically.

Example 15.7 (Unsolvable ODE)**.** Consider the equation $y' = 2y/t$, with $y(0) \neq 0$ given; the $1/t$ factor does not divide by zero because the ODE only has to hold for $t > 0$. Rewriting as

$$\frac{1}{y}\frac{dy}{dt} = \frac{2}{t}$$

and integrating with respect to t on both sides shows

$$\ln|y| = 2\ln t + c.$$

Exponentiating both sides shows $y = Ct^2$ for some $C \in \mathbb{R}$. In this expression, $y(0) = 0$, contradicting the initial conditions. Thus, this ODE has no solution with the given initial conditions.

Example 15.8 (Nonunique solutions)**.** Now, consider the same ODE with $y(0) = 0$. Consider $y(t)$ given by $y(t) = Ct^2$ for *any* $C \in \mathbb{R}$. Then, $y'(t) = 2Ct$ and

$$\frac{2y}{t} = \frac{2Ct^2}{t} = 2Ct = y'(t),$$

showing that the ODE is solved by this function regardless of C. Thus, solutions of this equation with the new initial conditions are nonunique.

There is a rich theory characterizing behavior and stability of solutions to ordinary differential equations. Under weak conditions on F, it is possible to show that an ODE $f'(t) = F[f(t)]$ has a solution; in the next chapter, we will see that showing existence and/or uniqueness for PDEs rather than ODEs does not benefit from this structure. One such theorem guarantees existence of a solution when F is not sharply sloped:

Theorem 15.1 (ODE existence and uniqueness)**.** Suppose F is continuous and Lipschitz, that is, $\|F[\vec{y}] - F[\vec{x}]\|_2 \leq L\|\vec{y} - \vec{x}\|_2$ for some fixed $L \geq 0$. Then, the ODE $f'(t) = F[f(t)]$ admits exactly one solution for all $t \geq 0$ regardless of initial conditions.

In our subsequent development, we will assume that the ODE we are attempting to solve satisfies the conditions of such a theorem. This assumption is realistic since the conditions guaranteeing existence and uniqueness are relatively weak.

15.2.3 Model Equations

One way to understand computational methods for integrating ODEs is to examine their behavior on well-understood *model equations*. Many ODEs locally can be approximated by these model equations, motivating our detailed examination of these simplistic test cases.

We start by introducing a model equation for ODEs with a single dependent variable. Given our simplifications in §15.2.1, we consider equations of the form $y' = F[y]$, where $y(t) : [0, \infty) \to \mathbb{R}$. Taking a linear approximation of F, we might define $y' = ay + b$ to be

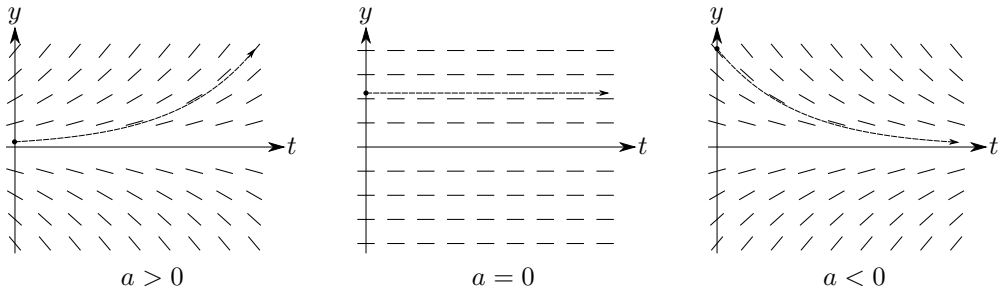


Figure 15.4 Three cases of the linear model equation $y' = ay$.

the model ODE, but we actually can fix $b = 0$. To justify using just one degree of freedom, define $\bar{y} \equiv y + b/a$. Then,

$$\begin{aligned}
\bar{y}' &= \left(y + \frac{b}{a}\right)' \text{ by definition of } \bar{y} \\
&= y' \text{ since the second term is constant with respect to } t \\
&= ay + b \text{ from the linearization} \\
&= a(\bar{y} - b/a) + b \text{ by inverting the definition of } \bar{y} \\
&= a\bar{y}.
\end{aligned}$$

This substitution satisfies $\bar{y}' = a\bar{y}$, showing that the constant b does not affect the qualitative behavior of the ODE. Hence, in the phenomenological study of model equations we safely take $b = 0$.

By the argument above, we locally can understand behavior of $y' = F[y]$ by studying the linear equation $y' = ay$. While the original ODE may not be solvable in closed form, applying standard arguments from calculus shows that the model equation is solved by the formula

$$y(t) = Ce^{at}.$$

Qualitatively, this formula splits into three cases, illustrated in Figure 15.4:

1. $a > 0$: Solutions get larger and larger; if $y(t)$ and $\hat{y}(t)$ both satisfy the ODE with slightly different starting conditions, as $t \to \infty$ they diverge.
2. $a = 0$: This system is solved by constant functions; solutions with different starting points stay the same distance apart.
3. $a < 0$: All solutions approach 0 as $t \to \infty$.

We say cases 2 and 3 are *stable*, in the sense that perturbing $y(0)$ yields solutions that do not diverge from each other over time; case 1 is *unstable*, since a small mistake in specifying the initial condition $y(0)$ will be amplified as time t advances.

Unstable ODEs generate ill-posed computational problems. Without careful consideration, we cannot expect numerical methods to generate usable solutions in this case, since theoretical solutions are already sensitive to perturbations of the input. On the other hand, stable problems are well-posed since small mistakes in $y(0)$ get diminished over time. Both cases are shown in Figure 15.5.

Extending to multiple dimensions, we study the linearized equation $\vec{y}' = A\vec{y}$; for simplicity, we will assume A is symmetric. As explained in §6.1.2, if $\vec{y}_1, \cdots, \vec{y}_k$ are eigenvectors of A

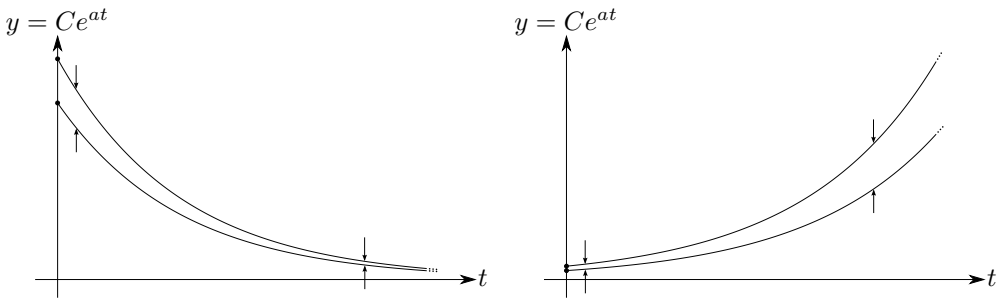


Figure 15.5 A stable ODE diminishes the difference between solutions over time t if $y(0)$ is perturbed, while an unstable ODE amplifies this difference.

with eigenvalues $\lambda_1, \ldots, \lambda_k$ and $\vec{y}(0) = c_1\vec{y}_1 + \cdots + c_k\vec{y}_k$, then $\vec{y}(t) = c_1e^{\lambda_1 t}\vec{y}_1 + \cdots + c_ke^{\lambda_k t}\vec{y}_k$. Based on this formula, the eigenvalues of A take the place of a in the one-dimensional model equation. From this result, it is not hard to intuit that a multivariable solution to $\vec{y}' = A\vec{y}$ is stable exactly when the eigenvalues of A are bounded above by zero.

As in the single-variable case, in reality we need to solve $\vec{y}' = F[\vec{y}]$ for general functions F. Assuming F is differentiable, we can approximate $F[\vec{y}] \approx F[\vec{y}_0] + J_F(\vec{y}_0)(\vec{y} - \vec{y}_0)$, yielding the model equation above after a shift. This argument shows that for short periods of time we expect behavior similar to the model equation with $A = J_F(\vec{y}_0)$, the Jacobian at $\vec{y}_0$.

15.3 TIME-STEPPING SCHEMES

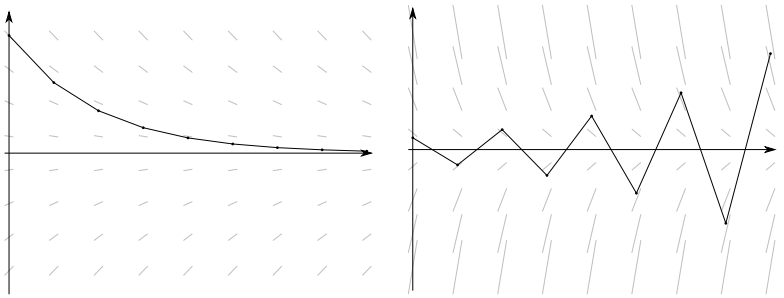
We now describe several methods for solving the nonlinear ODE $\vec{y}' = F[\vec{y}]$ given potentially nonlinear functions F. Given a "time step" h, our methods will generate estimates of $\vec{y}(t+h)$ given $\vec{y}(t)$ and F. Applying these methods iteratively generates estimates $\vec{y}_0 \equiv \vec{y}(t)$, $\vec{y}_1 \approx \vec{y}(t+h)$, $\vec{y}_2 \approx \vec{y}(t+2h)$, $\vec{y}_3 \approx \vec{y}(t+3h)$, and so on. We call algorithms for generating approximations of $\vec{y}(t)$ *time-stepping schemes* or *integrators*, reflecting the fact that they are integrating out the derivatives in the input equation.

Of key importance to our consideration is the idea of *stability*. Even if an ODE theoretically is stable using the definition from §15.2.3, the integrator may produce approximations that diverge at an exponential rate. Stability usually depends on the time step h; when h is too large, differential estimates of the quality of an integrator fail to hold, yielding unpredictable output. Stability, however, can compete with *accuracy*. Stable schemes may generate bad approximations of $\vec{y}(t)$, even if they are guaranteed not to have wild behavior. ODE integrators that are both stable and accurate tend to require excessive computation time, indicating that we must compromise between these two properties.

15.3.1 Forward Euler

Our first ODE integrator comes from our construction of the forward differencing scheme in §14.3.2:

$$F[\vec{y}_k] = \vec{y}'(t) = \frac{\vec{y}_{k+1} - \vec{y}_k}{h} + O(h).$$



Stable ($a = -0.4$) Unstable ($a = -2.3$)

Figure 15.6 Unstable and stable cases of forward Euler integration for the model equation $y' = ay$ with $h = 1$.

Solving this relationship for $\vec{y}_{k+1}$ shows

$$\boxed{\vec{y}_{k+1} = \vec{y}_k + hF[\vec{y}_k] + O(h^2) \approx \vec{y}_k + hF[\vec{y}_k].}$$

This *forward Euler* scheme applies the approximation on the right to estimate $\vec{y}_{k+1}$ from $\vec{y}_k$. It is one of the most computationally efficient strategies for time-stepping. Forward Euler is an *explicit* integrator, since there is an explicit formula for $\vec{y}_{k+1}$ in terms of $\vec{y}_k$ and F.

The forward Euler approximation of $\vec{y}_{k+1}$ holds to $O(h^2)$, so each step induces quadratic error. We call this the *localized truncation error* because it is the error induced by a single time step. The word "truncation" refers to the fact that we truncated a Taylor series to obtain the integrator. The iterate $\vec{y}_k$, however, already may be inaccurate thanks to accumulated truncation errors from previous iterations. If we integrate from t_0 to t with $k = O(1/h)$ steps, then the total error looks like $O(h)$. This estimate quantifies *global truncation error*, and thus we usually say that the forward Euler scheme is "first-order accurate."

The stability of forward Euler can be motivated by studying the model equation. We will work out the stability of methods in the one-variable case $y' = ay$, with the intuition that similar statements carry over to multidimensional equations by replacing a with a spectral radius. Substituting the one-variable model equation into the forward Euler scheme gives

$$y_{k+1} = y_k + ahy_k = (1 + ah)y_k.$$

Expanding recursively shows $y_k = (1+ah)^k y_0$. By this explicit formula for y_k in terms of y_0, the integrator is stable when $|1+ah| \leq 1$, since otherwise $|y_k| \to \infty$ exponentially. Assuming $a < 0$ (otherwise the theoretical problem is ill-posed), this condition takes a simpler form:

$$\begin{aligned} |1 + ah| \leq 1 &\iff -1 \leq 1 + ah \leq 1 \text{ by expanding the absolute value} \\ &\iff 0 \leq h \leq \frac{2}{|a|}, \text{ since } a < 0. \end{aligned}$$

This derivation shows that forward Euler admits a *time step restriction*. That is, the output of forward Euler integration can explode even if $y' = ay$ is stable, when h is too large.

Figure 15.6 illustrates what happens when the stability condition is obeyed or violated. When time steps are too large—or equivalently when $|a|$ is too large—the forward Euler method is not only inaccurate but also has very different qualitative behavior. For nonlinear ODEs this formula gives a guide for stability at least locally in time; globally h may have to be adjusted if the Jacobian of F becomes worse conditioned.

Certain well-posed ODEs require tiny time steps h for forward Euler to be stable. In this case, even though the forward Euler formula is computationally inexpensive for a single step, integrating to some fixed time t may be infeasible because so many steps are needed. Such ODEs are called *stiff*, inspired by stiff springs requiring tiny time steps to capture their rapid oscillations. One text defines stiff problems slightly differently (via [60]): "Stiff equations are problems for which explicit methods don't work" [57]. With this definition in mind, in the next section we consider an *implicit* method with no stability time step restriction, making it more suitable for stiff problems.

15.3.2 Backward Euler

We could have applied backward differencing at $\vec{y}_{k+1}$ to design an ODE integrator:

$$F[\vec{y}_{k+1}] = \vec{y}'(t+h) = \frac{\vec{y}_{k+1} - \vec{y}_k}{h} + O(h).$$

Isolating $\vec{y}_k$ shows that this integrator requires solving the following potentially nonlinear system of equations for $\vec{y}_{k+1}$:

$$\boxed{\vec{y}_{k+1} = \vec{y}_k + hF[\vec{y}_{k+1}].}$$

This equation differs from forward Euler integration by the evaluation of F at $\vec{y}_{k+1}$ rather than at $\vec{y}_k$. Because we have to solve this equation for $\vec{y}_{k+1}$, this technique, known as *backward Euler* integration, is an *implicit* integrator.

Example 15.9 (Backward Euler)**.** Suppose we wish to generate time steps for the ODE $\vec{y}' = A\vec{y}$, with fixed $A \in \mathbb{R}^{n\times n}$. To find $\vec{y}_{k+1}$ we solve the following system:

$$\vec{y}_k = \vec{y}_{k+1} - hA\vec{y}_{k+1} \implies \vec{y}_{k+1} = (I_{n\times n} - hA)^{-1}\vec{y}_k.$$

Backward Euler is first-order accurate like forward Euler by an identical argument. Its stability, however, contrasts considerably with that of forward Euler. Once again considering the model equation $y' = ay$, we write:

$$y_k = y_{k+1} - hay_{k+1} \implies y_{k+1} = \frac{y_k}{1-ha}.$$

To prevent exponential blowup, we enforce the following condition:

$$\frac{1}{|1-ha|} \leq 1 \iff h \leq \frac{2}{a} \text{ or } h \geq 0, \text{ for } a < 0.$$

We always choose $h \geq 0$, so backward Euler is *unconditionally* stable, as in Figure 15.7.

Even if backward Euler is stable, however, it may not be accurate. If h is too large, $\vec{y}_k$ will approach zero at the wrong rate. When simulating cloth and other physical materials that require lots of high-frequency detail to be realistic, backward Euler may exhibit undesirable dampening. Furthermore, we have to invert $F[\cdot]$ to solve for $\vec{y}_{k+1}$.

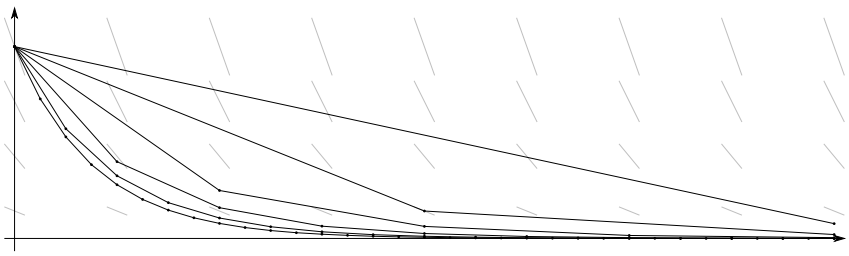


Figure 15.7 Backward Euler integration is unconditionally stable, so no matter how large a time step h with the same initial condition, the resulting approximate solution of $y' = ay$ does not diverge. While the output is stable, when h is large the result does not approximate the continuous solution $y = Ce^{at}$ effectively.

15.3.3 Trapezoidal Method

Suppose that in addition to having $\vec{y}_k$ at time t and $\vec{y}_{k+1}$ at time $t+h$, we also know $\vec{y}_{k+1/2}$ at the halfway point in time $t + h/2$. Then, by our derivation of centered differencing

$$\vec{y}_{k+1} = \vec{y}_k + hF[\vec{y}_{k+1/2}] + O(h^3).$$

In our derivation of error bounds for the trapezoidal rule in §14.2.3, we derived the following relationship via Taylor's theorem:

$$\frac{F[\vec{y}_{k+1}] + F[\vec{y}_k]}{2} = F[\vec{y}_{k+1/2}] + O(h^2).$$

Substituting this equality into the expression for $\vec{y}_{k+1}$ yields a second-order ODE integrator, the *trapezoid method*:

$$\boxed{\vec{y}_{k+1} = \vec{y}_k + h\frac{F[\vec{y}_{k+1}] + F[\vec{y}_k]}{2}}$$

Like backward Euler, this method is implicit since we must solve this equation for $\vec{y}_{k+1}$.

Example 15.10 (Trapezoidal integrator)**.** Returning to the ODE $\vec{y}' = A\vec{y}$ from Example 15.9, trapezoidal integration solves the system

$$\vec{y}_{k+1} = \vec{y}_k + h\frac{A\vec{y}_{k+1} + A\vec{y}_k}{2} \implies \vec{y}_{k+1} = \left(I_{n\times n} - \frac{hA}{2}\right)^{-1}\left(I_{n\times n} + \frac{hA}{2}\right)\vec{y}_k.$$

To carry out stability analysis on $y' = ay$, the example above shows time steps of the trapezoidal method satisfy

$$y_k = \left(\frac{1 + \frac{1}{2}ha}{1 - \frac{1}{2}ha}\right)^k y_0.$$

Consequently, the method is stable when

$$\left|\frac{1 + \frac{1}{2}ha}{1 - \frac{1}{2}ha}\right| < 1.$$

This inequality holds whenever $a < 0$ and $h > 0$, showing that the trapezoid method is unconditionally stable.

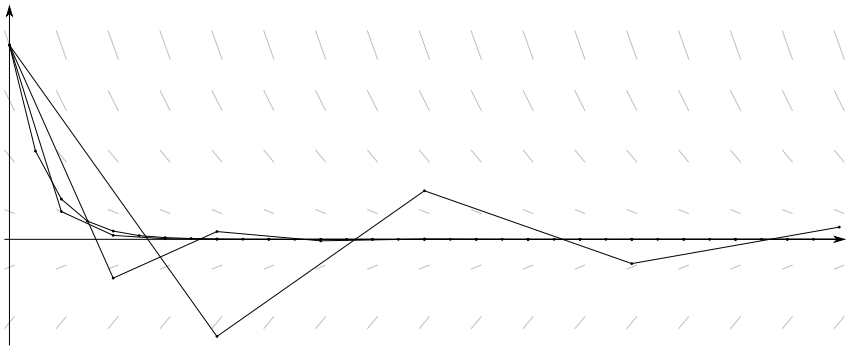


Figure 15.8 The trapezoidal method is unconditionally stable, so regardless of the step size h the solution curves always approach $y = 0$; when h is large, however, the output oscillates about zero as it decays.

Despite its higher order of accuracy with maintained stability, the trapezoid method has some drawbacks that make it less popular than backward Euler for large time steps. In particular, consider the ratio

$$R \equiv \frac{y_{k+1}}{y_k} = \frac{1 + \frac{1}{2}ha}{1 - \frac{1}{2}ha}.$$

When $a < 0$, for large enough h this ratio eventually becomes negative; as $h \to \infty$, we have $R \to -1$. As illustrated in Figure 15.8, this observation shows that if time steps h are too large, the trapezoidal method of integration tends to introduce undesirable oscillatory behavior not present in theoretical solutions Ce^{at} of $y' = ay$.

15.3.4 Runge-Kutta Methods

A class of integrators can be derived by making the following observation:

$$\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + \int_{t_k}^{t_k+h} \vec{y}'(t)\,dt \text{ by the Fundamental Theorem of Calculus} \\ &= \vec{y}_k + \int_{t_k}^{t_k+h} F[\vec{y}(t)]\,dt \text{ since } \vec{y} \text{ satisfies } \vec{y}'(t) = F[\vec{y}(t)].\end{aligned}$$

Using this formula outright does not help design a method for time-stepping, since we do not know $\vec{y}(t)$ *a priori*. Approximating the integral using quadrature rules from the previous chapter, however, produces a class of well-known strategies for ODE integration.

Suppose we apply the trapezoidal quadrature rule to the integral for $\vec{y}_{k+1}$. Then,

$$\vec{y}_{k+1} = \vec{y}_k + \frac{h}{2}(F[\vec{y}_k] + F[\vec{y}_{k+1}]) + O(h^3).$$

This is the formula we wrote for the trapezoidal method in §15.3.3. If we wish to find an explicit rather than implicit method with the accuracy of the trapezoidal time-stepping, however, we must replace $F[\vec{y}_{k+1}]$ with a high-accuracy approximation that is easier to evaluate:

$$\begin{aligned}F[\vec{y}_{k+1}] &= F[\vec{y}_k + hF[\vec{y}_k] + O(h^2)] \text{ by the forward Euler order of accuracy} \\ &= F[\vec{y}_k + hF[\vec{y}_k]] + O(h^2) \text{ by Taylor's theorem.}\end{aligned}$$

Since it gets scaled by h, making this substitution for $\vec{y}_{k+1}$ does not affect the order of approximation of the trapezoidal time step. This change results in a new approximation:

$$\boxed{\vec{y}_{k+1} = \vec{y}_k + \frac{h}{2}(F[\vec{y}_k] + F[\vec{y}_k + hF[\vec{y}_k]]) + O(h^3).}$$

Ignoring the $O(h^3)$ terms yields a new integrator known as *Heun's method*, which is second-order accurate and explicit.

To evaluate the stability of Heun's method for $y' = ay$ with $a < 0$, we expand

$$y_{k+1} = y_k + \frac{h}{2}(ay_k + a(y_k + hay_k)) = \left(\frac{1}{2}h^2a^2 + ha + 1\right) y_k.$$

From this substitution, the method is stable when

$$-1 \leq 1 + ha + \frac{1}{2}h^2a^2 \leq 1 \iff -4 \leq 2ha + h^2a^2 \leq 0.$$

The inequality on the right is equivalent to writing $h \leq \frac{2}{|a|}$, and the inequality on the left is always true for $h > 0$ and $a < 0$. Hence, the stability condition for Heun's method can be written $h \leq \frac{2}{|a|}$, the same as the stability condition for forward Euler.

Heun's method is an example of a *Runge-Kutta* integrator derived by starting from a quadrature rule and substituting Euler steps to approximate $F[\vec{y}_{k+\ell}]$, for $\ell > 0$. Forward Euler is a first-order accurate Runge-Kutta method, and Heun's method is second-order. A popular fourth-order Runge-Kutta method (abbreviated "RK4") is given by:

$$\boxed{\begin{aligned} \vec{y}_{k+1} &= \vec{y}_k + \frac{h}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \\ \text{where } \vec{k}_1 &= F\left[\vec{y}_k\right] \\ \vec{k}_2 &= F\left[\vec{y}_k + \frac{1}{2}h\vec{k}_1\right] \\ \vec{k}_3 &= F\left[\vec{y}_k + \frac{1}{2}h\vec{k}_2\right] \\ \vec{k}_4 &= F\left[\vec{y}_k + h\vec{k}_3\right] \end{aligned}}$$

This formula is constructed from Simpson's quadrature rule.

Runge-Kutta methods are popular because they are explicit but provide high degrees of accuracy. The cost of this accuracy, however, is that $F[\cdot]$ must be evaluated more times to carry out a single time step. Implicit Runge-Kutta integrators also have been constructed, for poorly conditioned ODEs.

15.3.5 Exponential Integrators

We have focused our stability and accuracy analyses on the model equation $y' = ay$. If this ODE is truly an influential test case, however, we have neglected a key piece of information: We know the solution of $y' = ay$ in closed form as $y = Ce^{at}$! We might as well incorporate this formula into an integration scheme to achieve 100% accuracy on the model equation. That is, we can design a class of integrators that achieves strong accuracy when $F[\cdot]$ is nearly linear, potentially at the cost of computational efficiency.

Assuming A is symmetric, using the eigenvector method from §15.2.3 we can write the solution of the ODE $\vec{y}' = A\vec{y}$ as $\vec{y}(t) = e^{At}\vec{y}(0)$, where e^{At} is a matrix encoding the transformation from $\vec{y}(0)$ to $\vec{y}(t)$ (see Exercise 6.10). Starting from this formula, integrating in time by writing $\vec{y}_{k+1} = e^{Ah}\vec{y}_k$ achieves perfect accuracy on the linear model equation; our strategy is to use this formula to construct integrators for the nonlinear case.

When F is smooth, we can attempt to factor the ODE $\vec{y}' = F[\vec{y}]$ as

$$\vec{y}' = A\vec{y} + G[\vec{y}],$$

where G is a nonlinear but small function and $A \in \mathbb{R}^{n\times n}$. Taking A to be the Jacobian of F makes this factorization agree with its first-order Taylor expansion. *Exponential integrators* integrate the $A\vec{y}$ part using the exponential formula and approximate the effect of the nonlinear G part separately.

We start by deriving a "variation of parameters" formula from the classical theory of ODEs. Rewriting the original ODE as $\vec{y}' - A\vec{y} = G[\vec{y}]$, suppose we multiply both sides by e^{-At} to obtain $e^{-At}(\vec{y}' - A\vec{y}) = e^{-At}G[\vec{y}]$. The left-hand side satisfies

$$e^{-At}(\vec{y}' - A\vec{y}) = \frac{d}{dt}\left(e^{-At}\vec{y}(t)\right),$$

after applying the identity $Ae^{At} = e^{At}A$ (see Exercise 15.2), implying $d/dt(e^{-At}\vec{y}(t)) = e^{-At}G[\vec{y}]$. Integrating this expression from 0 to t shows

$$e^{-At}\vec{y}(t) - \vec{y}(0) = \int_0^t e^{-A\tau}G[\vec{y}(\tau)]\,d\tau,$$

or equivalently,

$$\begin{aligned}\vec{y}(t) &= e^{At}\vec{y}(0) + e^{At}\int_0^t e^{-A\tau}G[\vec{y}(\tau)]\,d\tau \\ &= e^{At}\vec{y}(0) + \int_0^t e^{A(t-\tau)}G[\vec{y}(\tau)]\,d\tau.\end{aligned}$$

Slightly generalizing this formula shows

$$\vec{y}_{k+1} = e^{Ah}\vec{y}_k + \int_{t_k}^{t_k+h} e^{A(t_k+h-t)}G[\vec{y}(t)]\,dt.$$

Similar to the Runge-Kutta methods, exponential integrators apply quadrature to the integral on the right-hand side to approximate the time step to $\vec{y}_{k+1}$.

For example, the *first-order exponential integrator* applies forward Euler to the nonlinear G term by making a constant approximation $G[\vec{y}(t)] \approx G[\vec{y}_k]$, yielding

$$\vec{y}_{k+1} \approx e^{Ah}\vec{y}_k + \left[\int_0^h e^{A(h-t)}\,dt\right] G[\vec{y}_k].$$

As shown in Exercise 15.5, the integral can be taken in closed form, leading to the expression

$$\boxed{\vec{y}_{k+1} = e^{Ah}\vec{y}_k + A^{-1}(e^{Ah} - I_{n\times n})G[\vec{y}_k].}$$

Analyzing exponential integrators like this one requires techniques beyond using the linear model equation, since they are designed to integrate linear ODEs exactly. Intuitively, exponential integrators behave best when $G \approx 0$, but the cost of this high numerical performance is the use of a matrix exponential, which is difficult to compute or apply efficiently.

15.4 MULTIVALUE METHODS

The transformations in §15.2.1 reduced all explicit ODEs to the form $\vec{y}' = F[\vec{y}]$, which can be integrated using the methods introduced in the previous section. While all explicit ODEs *can* be written this way, however, it is not clear that they always *should* be when designing a high-accuracy integrator.

When we reduced k-th order ODEs to first order, we introduced new variables representing the first through $(k-1)$-st derivatives of the desired output function $\vec{y}(t)$. The integrators in the previous section then approximate $\vec{y}(t)$ and these $k-1$ derivatives with equal accuracy, since in some sense they are treated "democratically" in first-order form. A natural question is whether we can relax the accuracy of the approximated derivatives of $\vec{y}(t)$ without affecting the quality of the $\vec{y}(t)$ estimate itself.

To support this perspective, consider the Taylor series

$$\vec{y}(t_k + h) = \vec{y}(t_k) + h\vec{y}'(t_k) + \frac{h^2}{2}\vec{y}''(t_k) + O(h^3).$$

If we perturb $\vec{y}'(t_k)$ by some value on the order $O(h^2)$, the quality of this Taylor series approximation does not change, since

$$h \cdot [\vec{y}'(t_k) + O(h^2)] = h\vec{y}'(t_k) + O(h^3).$$

Perturbing $\vec{y}''(t_k)$ by a value on the order $O(h)$ has a similar effect, since

$$\frac{h^2}{2} \cdot [\vec{y}''(t_k) + O(h)] = \frac{h^2}{2}\vec{y}''(t_k) + O(h^3).$$

Based on this argument, *multivalue methods* integrate $\vec{y}^{(k)}(t) = F[t, \vec{y}'(t), \vec{y}''(t), \ldots, \vec{y}^{(k-1)}(t)]$ using less accurate estimates of the higher-order derivatives of $\vec{y}(t)$.

We will restrict our discussion to the second-order case $\vec{y}''(t) = F[t, \vec{y}, \vec{y}']$, the most common case for ODE integration thanks to Newton's second law $F = ma$. Extending the methods we consider to higher order, however, follows similar if notationally more complex arguments. For the remainder of this section, we will define a "velocity" vector $\vec{v}(t) \equiv \vec{y}'(t)$ and an "acceleration" vector $\vec{a} \equiv \vec{y}''(t)$. By the reduction to first order, we wish to solve the following order system:

$$\begin{aligned}
\vec{y}'(t) &= \vec{v}(t) \\
\vec{v}'(t) &= \vec{a}(t) \\
\vec{a}(t) &= F[t, \vec{y}(t), \vec{v}(t)].
\end{aligned}$$

Our goal is to derive integrators tailored to this system, evaluated based on the accuracy of estimating $\vec{y}(t)$ rather than $\vec{v}(t)$ or $\vec{a}(t)$.

15.4.1 Newmark Integrators

We begin by deriving the class of *Newmark* integrators following the development in [46]. Denote $\vec{y}_k$, $\vec{v}_k$, and $\vec{a}_k$ as position, velocity, and acceleration vectors at time t_k; our goal is to advance to time $t_{k+1} \equiv t_k + h$.

By the Fundamental Theorem of Calculus,

$$\vec{v}_{k+1} = \vec{v}_k + \int_{t_k}^{t_{k+1}} \vec{a}(t)\, dt.$$

We also can write $\vec{y}_{k+1}$ as an integral involving $\vec{a}(t)$ by deriving an error estimate developed in some proofs of Taylor's theorem:

$$\begin{aligned}
\vec{y}_{k+1} &= \vec{y}_k + \int_{t_k}^{t_{k+1}} \vec{v}(t)\,dt \text{ by the Fundamental Theorem of Calculus} \\
&= \vec{y}_k + [t\vec{v}(t)]_{t_k}^{t_{k+1}} - \int_{t_k}^{t_{k+1}} t\vec{a}(t)\,dt \text{ after integration by parts} \\
&= \vec{y}_k + t_{k+1}\vec{v}_{k+1} - t_k\vec{v}_k - \int_{t_k}^{t_{k+1}} t\vec{a}(t)\,dt \text{ by expanding the difference term} \\
&= \vec{y}_k + h\vec{v}_k + t_{k+1}\vec{v}_{k+1} - t_{k+1}\vec{v}_k - \int_{t_k}^{t_{k+1}} t\vec{a}(t)\,dt \text{ by adding and subtracting } h\vec{v}_k \\
&= \vec{y}_k + h\vec{v}_k + t_{k+1}(\vec{v}_{k+1} - \vec{v}_k) - \int_{t_k}^{t_{k+1}} t\vec{a}(t)\,dt \text{ after factoring out } t_{k+1} \\
&= \vec{y}_k + h\vec{v}_k + t_{k+1}\int_{t_k}^{t_{k+1}} \vec{a}(t)\,dt - \int_{t_k}^{t_{k+1}} t\vec{a}(t)\,dt \text{ since } \vec{v}'(t) = \vec{a}(t) \\
&= \vec{y}_k + h\vec{v}_k + \int_{t_k}^{t_{k+1}} (t_{k+1} - t)\vec{a}(t)\,dt.
\end{aligned}$$

Fix a constant $\tau \in [t_k, t_{k+1}]$. Then, we can write expressions for $\vec{a}_k$ and $\vec{a}_{k+1}$ using the Taylor series about τ:

$$\begin{aligned}
\vec{a}_k &= \vec{a}(\tau) + \vec{a}'(\tau)(t_k - \tau) + O(h^2) \\
\vec{a}_{k+1} &= \vec{a}(\tau) + \vec{a}'(\tau)(t_{k+1} - \tau) + O(h^2).
\end{aligned}$$

For any constant $\gamma \in \mathbb{R}$, scaling the expression for $\vec{a}_k$ by $1 - \gamma$, scaling the expression for $\vec{a}_{k+1}$ by γ, and summing shows

$$\begin{aligned}
\vec{a}(\tau) &= (1-\gamma)\vec{a}_k + \gamma\vec{a}_{k+1} + \vec{a}'(\tau)((\gamma - 1)(t_k - \tau) - \gamma(t_{k+1} - \tau)) + O(h^2) \\
&= (1-\gamma)\vec{a}_k + \gamma\vec{a}_{k+1} + \vec{a}'(\tau)(\tau - h\gamma - t_k) + O(h^2) \text{ after substituting } t_{k+1} = t_k + h.
\end{aligned}$$

Integrating $\vec{a}(t)$ from t_k to t_{k+1} yields the change in velocity. Substituting the approximation above shows:

$$\begin{aligned}
\vec{v}_{k+1} - \vec{v}_k = \int_{t_k}^{t_{k+1}} \vec{a}(\tau)\,d\tau &= (1-\gamma)h\vec{a}_k + \gamma h\vec{a}_{k+1} + \int_{t_k}^{t_{k+1}} \vec{a}'(\tau)(\tau - h\gamma - t_k)\,d\tau + O(h^3) \\
&= (1-\gamma)h\vec{a}_k + \gamma h\vec{a}_{k+1} + O(h^2),
\end{aligned}$$

where the second step holds because $(\tau - t_k) - h\gamma = O(h)$ for $\tau \in [t_k, t_{k+1}]$ and the interval of integration is of width h. Rearranging shows

$$\vec{v}_{k+1} = \vec{v}_k + (1-\gamma)h\vec{a}_k + \gamma h\vec{a}_{k+1} + O(h^2).$$

Starting again from the approximation we wrote for $\vec{a}(\tau)$—this time using a new constant β rather than γ—we can also develop an approximation for $\vec{y}_{k+1}$. To do so, we will work with the integrand in the Taylor estimate for $\vec{y}_{k+1}$:

$$\begin{aligned}
\int_{t_k}^{t_{k+1}} (t_{k+1} - t)\vec{a}(t)\,dt &= \int_{t_k}^{t_{k+1}} (t_{k+1} - \tau)((1-\beta)\vec{a}_k + \beta\vec{a}_{k+1} + \vec{a}'(\tau)(\tau - h\beta - t_k))\,d\tau \\
&\quad + O(h^3) \\
&= \frac{1}{2}(1-\beta)h^2\vec{a}_k + \frac{1}{2}\beta h^2\vec{a}_{k+1} + O(h^2) \text{ by a similar simplification.}
\end{aligned}$$

We can use this observation to write the Taylor series error estimate for $\vec{y}_{k+1}$ in a different form:

$$\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + h\vec{v}_k + \int_{t_k}^{t_{k+1}} (t_{k+1} - t)\vec{a}(t)\,dt \text{ from before} \\ &= \vec{y}_k + h\vec{v}_k + \left(\frac{1}{2} - \beta\right) h^2\vec{a}_k + \beta h^2\vec{a}_{k+1} + O(h^2).\end{aligned}$$

Summarizing this technical argument, we have derived the class of Newmark schemes, each characterized by the two fixed parameters γ and β:

$$\boxed{\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + h\vec{v}_k + \left(\frac{1}{2} - \beta\right) h^2\vec{a}_k + \beta h^2\vec{a}_{k+1} \\ \vec{v}_{k+1} &= \vec{v}_k + (1-\gamma)h\vec{a}_k + \gamma h\vec{a}_{k+1} \\ \vec{a}_k &= F[t_k, \vec{y}_k, \vec{v}_k]\end{aligned}}$$

This integrator is accurate up to $O(h^2)$ in each time step, making it globally first-order accurate. Depending on γ and β, the integrator can be implicit, since $\vec{a}_{k+1}$ appears in the expressions for $\vec{y}_{k+1}$ and $\vec{v}_{k+1}$.

Specific choices of β and γ yield integrators with additional properties:

- $\beta = \gamma = 0$ gives the *constant acceleration* integrator:

$$\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + h\vec{v}_k + \frac{1}{2}h^2\vec{a}_k \\ \vec{v}_{k+1} &= \vec{v}_k + h\vec{a}_k.\end{aligned}$$

  This integrator is explicit and holds exactly when the acceleration is a constant function of time.

- $\beta = 1/2, \gamma = 1$ gives the *constant implicit acceleration* integrator:

$$\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + h\vec{v}_k + \frac{1}{2}h^2\vec{a}_{k+1} \\ \vec{v}_{k+1} &= \vec{v}_k + h\vec{a}_{k+1}.\end{aligned}$$

  The velocity is stepped implicitly using backward Euler, giving first-order accuracy. The $\vec{y}$ update, however, can be written

$$\vec{y}_{k+1} = \vec{y}_k + \frac{1}{2}h(\vec{v}_k + \vec{v}_{k+1}),$$

  which coincides with the trapezoidal rule. Hence, this is our first example of a scheme where the velocity and position updates have different orders of accuracy. This technique, however, is still only globally first-order accurate in $\vec{y}$.

- $\beta = 1/4, \gamma = 1/2$ gives the following second-order trapezoidal scheme after some algebra:

$$\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + \frac{1}{2}h(\vec{v}_k + \vec{v}_{k+1}) \\ \vec{v}_{k+1} &= \vec{v}_k + \frac{1}{2}h(\vec{a}_k + \vec{a}_{k+1}).\end{aligned}$$

- $\beta = 0, \gamma = 1/2$ gives a second-order accurate *central differencing* scheme. In the canonical form, it is written

$$\vec{y}_{k+1} = \vec{y}_k + h\vec{v}_k + \frac{1}{2}h^2\vec{a}_k$$
$$\vec{v}_{k+1} = \vec{v}_k + \frac{1}{2}h(\vec{a}_k + \vec{a}_{k+1}).$$

  The method earns its name because simplifying the equations above leads to the alternative form:

$$\vec{v}_{k+1} = \frac{\vec{y}_{k+2} - \vec{y}_k}{2h}$$
$$\vec{a}_{k+1} = \frac{\vec{y}_{k+2} - 2\vec{y}_{k+1} + \vec{y}_k}{h^2}.$$

Newmark integrators are unconditionally stable when $4\beta > 2\gamma > 1$, with second-order accuracy exactly when $\gamma = 1/2$.

15.4.2 Staggered Grid and Leapfrog

A different way to achieve second-order accuracy in stepping $\vec{y}$ is to use centered differences about $t_{k+1/2} \equiv t_k + h/2$:

$$\vec{y}_{k+1} = \vec{y}_k + h\vec{v}_{k+1/2}.$$

Rather than attempting to approximate $\vec{v}_{k+1/2}$ from $\vec{v}_k$ and/or $\vec{v}_{k+1}$, we can process velocities $\vec{v}$ directly at *half* points on the grid of time steps.

A similar update steps forward the velocities with the same accuracy:

$$\vec{v}_{k+3/2} = \vec{v}_{k+1/2} + h\vec{a}_{k+1}.$$

A lower-order approximation suffices for the acceleration term since it is a higher-order derivative:

$$\vec{a}_{k+1} = F\left[t_{k+1}, \vec{x}_{k+1}, \frac{1}{2}(\vec{v}_{k+1/2} + \vec{v}_{k+3/2})\right].$$

This expression can be substituted into the equation for $\vec{v}_{k+3/2}$.

When $F[\cdot]$ has no dependence on $\vec{v}$, e.g., when simulating particles without wind resistance, the method is fully explicit:

$$\boxed{\begin{aligned}\vec{y}_{k+1} &= \vec{y}_k + h\vec{v}_{k+1/2}\\ \vec{a}_{k+1} &= F[t_{k+1}, \vec{y}_{k+1}]\\ \vec{v}_{k+3/2} &= \vec{v}_{k+1/2} + h\vec{a}_{k+1}\end{aligned}}$$

This is known as the *leapfrog* integrator, thanks to the staggered grid of times and the fact that each midpoint is used to update the next velocity or position.

A distinguishing property of the leapfrog scheme is its *time reversibility.** Assume we have used the leapfrog integrator to generate $(\vec{y}_{k+1}, \vec{v}_{k+3/2}, \vec{a}_{k+1})$. Starting at t_{k+1}, we might reverse the direction of time to step backward. The leapfrog equations give

$$\vec{v}_{k+1/2} = \vec{v}_{k+3/2} + (-h)\vec{a}_{k+1}$$
$$\vec{y}_k = \vec{y}_{k+1} - h\vec{v}_{k+1/2}.$$

These formulas invert the forward time step exactly. That is, if we run the leapfrog in reverse, we trace the solution back to where it started *exactly*, up to rounding error.

*Discussion of time reversibility contributed by Julian Kates-Harbeck.

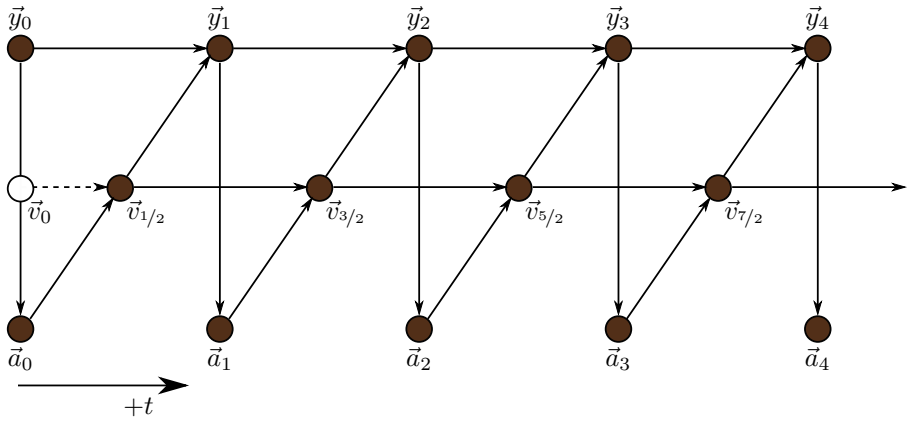


Figure 15.9 Leapfrog integration computes velocities at half time steps; here arrows denote dependencies between computed values. If the initial conditions specify $\vec{v}$ at $t = 0$, an initial half time step must be carried out to approximate $\vec{v}_{1/2}$.

A consequence of reversibility is that errors in position, energy, and angular momentum tend to cancel out over time as opposed to accumulating. For instance, for problems where the acceleration only depends on position, angular momentum is conserved exactly by leapfrog integration, and energy remains stable over time, whereas other even higher-order schemes can induce significant "drift" of these quantities. Symmetry, second-order accuracy for "first-order work" (i.e., the same amount of computation as forward Euler integration), and conservation properties make leapfrog integration a popular method for physical simulation. These properties classify the leapfrog method as a *symplectic* integrator, constructed to conserve the continuous structure of ODEs coming from Hamiltonian dynamics and related physical systems.

If $F[\cdot]$ has dependence on $\vec{v}$, then this "staggered grid" method becomes implicit. Such dependence on velocity often is symmetric. For instance, wind resistance changes sign if you reverse the direction in which you are moving. This property makes the matrices symmetric in the implicit step for updating velocities, enabling the application of conjugate gradients and related iterative methods.

15.5 COMPARISON OF INTEGRATORS

This chapter has introduced a sampling from the remarkably large pantheon of ODE integrators. Choosing the right ODE integrator for a given problem is a challenging task representing a careful balancing act between accuracy, stability, computational efficiency, and assorted special properties like reversibility. The table in Figure 15.10 compares the basic properties of the methods we considered.

In practice, it may require some experimentation to determine the proper integrator given an ODE problem; thankfully, most of the integrators we have introduced are relatively easy to implement. In addition to the generic considerations we have discussed in this chapter, additional "domain-specific" concerns also influence the choice of ODE integrators, including the following:

- In computer graphics and other fields prioritizing visual effect over reproducibility in the real world, it may be more important that a time-stepping method *looks* right

Integrator	*Section*	*Accuracy*	*Implicit or explicit?*	*Stability*	*Notes*
Forward Euler	§15.3.1	First	Explicit	Conditional	
Backward Euler	§15.3.2	First	Implicit	Unconditional	
Trapezoidal	§15.3.3	Second	Implicit	Unconditional	Large steps oscillate
Heun	§15.3.4	Second	Explicit	Conditional	
RK4	§15.3.4	Fourth	Explicit	Conditional	
1^{st}-order exponential	§15.3.5	First	Explicit	Conditional	Needs matrix exponential
Newmark	§15.4.1	First	Implicit	Conditional	For 2^{nd}-order ODE; 2^{nd}-order accurate when $\gamma = 1/2$; explicit when $\beta = \gamma = 0$
Staggered	§15.4.2	Second	Implicit	Conditional	For 2^{nd}-order ODE
Leapfrog	§15.4.2	Second	Explicit	Conditional	For 2^{nd}-order ODE; reversible; $F[\cdot]$ must not depend on $\vec{v}$

Figure 15.10 Comparison of ODE integrators.

than whether the numerical output is perfect. For instance, simulation tools for visual effects need to produce fluids, gases, and cloth that exhibit high-frequency swirls, vortices, and folds. These features may be dampened by a backward Euler integrator, even if it is more likely to be stable than other alternatives.

- Most of our analysis used Taylor series and other localized arguments, but *long-term behavior* of certain integrators can be favorable even if individual time steps are suboptimal. For instance, forward Euler integration tends to add energy to oscillatory ODEs, while backward Euler removes it. If we wish to simulate a pendulum swinging in perpetuity, neither of these techniques will suffice.

- Some ODEs operate in the presence of constraints. For instance, if we simulate a ball attached to a string, we may not wish for the string to stretch beyond its natural length. Methods like forward Euler and leapfrog integration can overshoot such constraints, so an additional projection step may be needed to enforce the constraints more exactly.

- A degree of adaptivity is needed for applications in which discrete events can happen during the course of solving an ODE. For instance, when simulating the dynamics of a piece of cloth, typically parts of the cloth can run into each other or into objects in their surroundings. These collision events can occur at fractional time steps and must be handled separately to avoid interpenetration of objects in a scene [5].

- For higher-quality animation and physical predictions, some ODE integrators can output not only the configuration at discrete time steps but also some indicator (e.g., an interpolatory formula) approximating continuous behavior between the time steps.

- If the function F in $\vec{y}' = F[\vec{y}]$ is smooth and differentiable, the derivatives of F can be used to improve the quality of time-stepping methods.

Many of these problems are difficult to handle efficiently in large-scale simulations and in cases where computational power is relatively limited.

15.6 EXERCISES

15.1 Some practice discretizing an ODE:

(a) Suppose we wish to solve the ODE $dy/dt = -\sin y$ numerically. For time step $h > 0$, write the implicit backward Euler equation for approximating y_{k+1} at $t = (k+1)h$ given y_k at $t = kh$.

(b) Write the Newton iteration for solving the equation from Exercise 15.1a for y_{k+1}.

15.2 We continue our discussion of the matrix exponential introduced in Exercise 6.10 and used in our discussion of exponential integrators. For this problem, assume $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix.

(a) Show that A commutes with e^{At} for any $t \geq 0$. That is, justify the formula $Ae^{At} = e^{At}A$.

(b) Recall that we can write

$$e^{At} = I_{n\times n} + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots.$$

For sufficiently small $h \geq 0$, prove a similar formula for matrix inverses:

$$(I_{n\times n} - hA)^{-1} = I_{n\times n} + hA + (hA)^2 + (hA)^3 + \cdots$$

(c) Which of the two series from Exercise 15.2b should converge faster? Based on this observation, compare the computational cost of a single backward Euler iteration (see Example 15.9) versus that of an iteration of the exponential integrator from §15.3.5 using these formulas.

15.3 Suppose we are solving a second-order ODE using the leapfrog integrator. We are given initial conditions $\vec{y}(0)$ and $\vec{v}(0)$, the position and velocity vectors at time $t = 0$. But, the leapfrog scheme maintains velocities at the half time steps. Propose a way to initialize $\vec{v}_{1/2}$ at time $t = h/2$, and argue that your initialization does not affect the order of accuracy of the leapfrog integrator if it is run for sufficiently many time steps.

15.4 Suppose we wish to approximate solutions to $\vec{y}'' = F[\vec{y}]$. Add together Taylor expansions for $\vec{y}(t+h)$ and $\vec{y}(t-h)$ to derive the *Verlet algorithm* for predicting $\vec{y}_{k+1}$ from $\vec{y}_k$ and $\vec{y}_{k-1}$, which induces $O(h^4)$ integration error in a single time step.

15.5 Verify the following formula used in §15.3.5 for symmetric $A \in \mathbb{R}^{n\times n}$:

$$\int_0^h e^{A(h-t)}\,dt = A^{-1}(e^{Ah} - I_{n\times n}).$$

Also, derive a global order of accuracy in the form $O(h^k)$ for some $k \in \mathbb{N}$ for the first-order exponential integrator.

15.6 In this problem, we will motivate an ODE used in computer graphics applications that does not come from Newton's laws. Throughout this problem, assume $f, g : [0,1] \to \mathbb{R}$ are differentiable functions with $g(0) = g(1) = 0$. We will derive continuous and discrete versions of the *screened Poisson equation*, used for smoothing (see, e.g., [24]).

(a) So far our optimization problems have been to find points $\vec{x}^* \in \mathbb{R}^n$ minimizing some function $h(\vec{x})$, but sometimes our unknown is an entire function. Thankfully, the "variational" approach still is valid in this case. Explain in words what the following energies, which take a function f as input, measure about f:

(i) $E_1[f] \equiv \int_0^1 (f(t) - f_0(t))^2\,dt$ for some fixed function $f_0 : [0,1] \to \mathbb{R}$.

(ii) $E_2[f] \equiv \int_0^1 (f'(t))^2\,dt$.

(b) For an energy functional $E[\cdot]$ like the two above, explain how the following expression for $dE(f;g)$ (the Gâteaux derivative of E) can be thought of as the "directional derivative of E at f in the g direction":

$$dE(f;g) = \frac{d}{d\varepsilon} E[f + \varepsilon g]\Big|_{\varepsilon=0}.$$

(c) Again assuming $g(0) = g(1) = 0$, derive the following formulae:

(i) $dE_1(f,g) = \int_0^1 2(f(t) - f_0(t))g(t)\,dt$.

(ii) $dE_2(f,g) = \int_0^1 -2f''(t)g(t)\,dt$.
Hint: Apply integration by parts to get rid of $g'(t)$; recall our assumption $g(0) = g(1) = 0$.

(d) Suppose we wish to approximate f_0 with a smoother function f. One reasonable model for doing so is to minimize $E[f] \equiv E_1[f] + \alpha E_2[f]$ for some $\alpha > 0$ controlling the trade-off between similarity to f_0 and smoothness. Using the result of 15.6c, argue informally that an f minimizing this energy should satisfy the differential equation $f(t) - f_0(t) = \alpha f''(t)$ for $t \in (0,1)$.

(e) Now, suppose we discretize f on $[0,1]$ using n evenly spaced samples $f^1, f^2, \ldots, f^n \in \mathbb{R}$ and f_0 using samples $f_0^1, f_0^2, \ldots, f_0^n$. Devise a discrete analog of $E[f]$ as a quadratic energy in the f^k's. For $k \notin \{1, n\}$, does differentiating E with respect to f_k yield a result analogous to Exercise 15.6d?

15.7 (Adapted from [21]) The swing angle θ of a pendulum under gravity satisfies the following ODE:

$$\theta'' = -\sin\theta,$$

where $|\theta(0)| < \pi$ and $\theta'(0) = 0$.

(a) Suppose $\theta(t)$ solves the ODE. Show that the following value (representing the energy of the system) is constant as a function of t:

$$E(t) \equiv \frac{1}{2}(\theta')^2 - \cos\theta.$$

(b) Many ODE integrators drift away from the desired output as time progresses over larger periods. For instance, forward Euler can add energy to a system by overshooting, while backward Euler tends to damp out motion and remove energy. In many computer graphics applications, quality long-term behavior can be prioritized, since large-scale issues cause visual artifacts. The class of *symplectic* integrators is designed to avoid this issue.

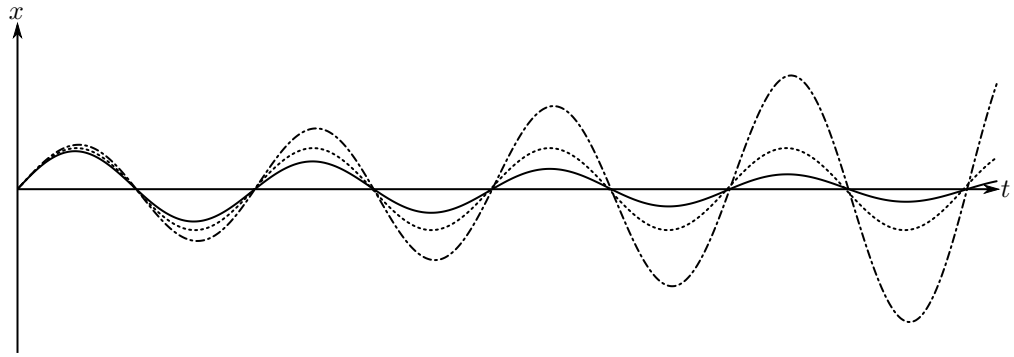


Figure 15.11 Three simulations of an undamped oscillator.

Denote $\omega \equiv \theta'$. The *symplectic Euler* scheme makes a series of estimates $\theta_0, \theta_1, \theta_2, \theta_3, \ldots$ and $\omega_0, \omega_1, \omega_2, \omega_3, \ldots$ at time $t = 0, h, 2h, 3h, \ldots$ using the following iteration:

$$\theta_{k+1} = \theta_k + h\omega_k$$
$$\omega_{k+1} = \omega_k - h \sin \theta_{k+1}.$$

Define

$$E_k \equiv \frac{1}{2}\omega_k^2 - \cos \theta_k.$$

Show that $E_{k+1} = E_k + O(h^2)$.

(c) Suppose we make the small-angle approximation $\sin \theta \approx \theta$ and decide to solve the linear ODE $\theta'' = -\theta$ instead. Now, symplectic Euler takes the following form:

$$\theta_{k+1} = \theta_k + h\omega_k$$
$$\omega_{k+1} = \omega_k - h\theta_{k+1}.$$

Write a 2×2 matrix A such that

$$\begin{pmatrix} \theta_{k+1} \\ \omega_{k+1} \end{pmatrix} = A \begin{pmatrix} \theta_k \\ \omega_k \end{pmatrix}.$$

(d) If we define $E_k \equiv \omega_k^2 + h\omega_k\theta_k + \theta_k^2$, show that $E_{k+1} = E_k$ in the iteration from 15.7c. In other words, E_k is constant from time step to time step.

15.8 Suppose we simulate a spring by solving the ODE $y'' = -y$ with $y(0) = 0$ and $y'(0) = 1$. We obtain the three plots of $y(t)$ in Figure 15.11 by using forward Euler, backward Euler, and symplectic Euler time integration. Determine which plot is which, and justify your answers using properties of the three integrators.

15.9 Suppose we discretize Schrödinger's equation for a particular quantum simulation yielding an ODE $\vec{x}' = A\vec{x}$, for $\vec{x}(t) \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n \times n}$. Furthermore, suppose that A is *self-adjoint* and *negative definite*, that is, A satisfies the following properties:

- Self-adjoint: $a_{ij} = \bar{a}_{ji}$, where $\overline{a + bi} = a - bi$.

- Negative definite: $\vec{\bar{x}}^\top A\vec{x} \leq 0$ (and is real) for all $\vec{x} \in \mathbb{C}^n \backslash \{\vec{0}\}$. Here we define $(\vec{x})_i \equiv \bar{x}_i$.

Derive a backward Euler formula for solving this ODE and show that each step can be carried out using conjugate gradients.
Hint: Before discretizing, convert the ODE to a real-valued system by separating imaginary and real parts of the variables and constants.

15.10 ("Phi functions," [89]) Exponential integrators made use of ODEs with known solutions to boost numerical quality of time integration. This strategy can be extended using additional closed-form solutions.

(a) Define $\varphi_k(x)$ recursively by defining $\varphi_0(x) \equiv e^x$ and recursively writing

$$\varphi_{k+1}(x) \equiv \frac{1}{x}\left(\varphi_k(x) - \frac{1}{k!}\right).$$

Write the Taylor expansions of $\varphi_0(x)$, $\varphi_1(x)$, $\varphi_2(x)$, and $\varphi_3(x)$ about $x = 0$.

(b) Show that for $k \geq 1$,

$$\varphi_k(x) = \frac{1}{(k-1)!}\int_0^1 e^{(1-\theta)x}\theta^{k-1}\,d\theta.$$

Hint: Use integration by parts to show that the recursive relationship from Exercise 15.10a holds.

(c) Check the following formula for $\varphi_k'(x)$ when $k \geq 1$:

$$\varphi_k'(x) = \frac{1}{x}\left[\varphi_k(x)(x-k) + \frac{1}{(k-1)!}\right].$$

(d) Show that the ODE

$$\vec{u}'(t) = L\vec{u}(t) + \vec{v}_0 + \frac{1}{1!}t\vec{v}_1 + \frac{1}{2!}t^2\vec{v}_2 + \frac{1}{3!}t^3\vec{v}_3 + \cdots$$

subject to $\vec{u}(0) = \vec{u}_0$ is solved by

$$\vec{u}(t) = \varphi_0(tL)\vec{u}_0 + t\varphi_1(tL)\vec{v}_0 + t^2\varphi_2(tL)\vec{v}_1 + t^3\varphi_3(tL)\vec{v}_2 + \cdots.$$

When L is a matrix, assume $\varphi_k(tL)$ is evaluated using the formula from Exercise 15.10b.

(e) Use the closed-form solution from Exercise 15.10d to propose an integrator for the ODE $\vec{y}' = A\vec{y} + \sum_{k=1}^{\ell}\frac{t^k}{k!}\vec{v}_k + G[\vec{y}]$ that provides exact solutions when $G[\vec{y}] \equiv \vec{0}$.

15.11 ("Fehlberg's method," [39] via notes by J. Feldman) We can approximate the error of an ODE integrator to help choose appropriate step sizes given a desired level of accuracy.

(a) Suppose we carry out a single time step of $\vec{y}' = F[\vec{y}]$ with size h starting from $\vec{y}(0) = \vec{y}_0$. Make the following definitions:

$$\begin{aligned}\vec{v}_1 &\equiv F[\vec{y}_0] \\ \vec{v}_2 &\equiv F[\vec{y}_0 + h\vec{v}_1] \\ \vec{v}_3 &\equiv F\left[\vec{y}_0 + \frac{h}{4}(\vec{v}_1 + \vec{v}_2)\right].\end{aligned}$$

We can write two estimates of $\vec{y}(h)$:

$$\begin{aligned}\vec{y}^{(1)} &\equiv \vec{y}_0 + \frac{h}{2}(\vec{v}_1 + \vec{v}_2) \\ \vec{y}^{(2)} &\equiv \vec{y}_0 + \frac{h}{6}(\vec{v}_1 + \vec{v}_2 + 4\vec{v}_3).\end{aligned}$$

Show that there is some $K \in \mathbb{R}$ such that $\vec{y}^{(1)} = \vec{y}(h) + Kh^3 + O(h^4)$ and $\vec{y}^{(2)} = \vec{y}(h) + O(h^4)$.

(b) Use this relationship to derive an approximation of the amount of error introduced per unit increase of time t if we use $\vec{y}^{(1)}$ as an integrator. If this value is too large, adaptive integrators reject the step and try again with a smaller h.