CHAPTER 14

Integration and Differentiation

CONTENTS

THE previous chapter developed tools for predicting values of a function $f(\vec{x})$ given a sampling of points $(\vec{x}_i, f(\vec{x}_i))$ in the domain of f. Such methods are useful in themselves for completing functions that are known to be continuous or differentiable but whose values only are sampled at a set of isolated points, but in some cases we instead wish to compute "derived quantities" from the sampled function. Most commonly, many applications must approximate the integral or derivatives of f rather than its values.

There are many applications in which numerical integration and differentiation play key roles for computation. In the most straightforward instance, some well-known functions are *defined* as integrals. For instance, the "error function" given by the cumulative distribution of a bell curve is defined as:

$$\operatorname{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$

Approximations of $\operatorname{erf}(x)$ are needed in statistical methods, and one reasonable approach to finding these values is to compute the integral above numerically.

Other times, numerical approximations of derivatives and integrals are part of a larger system. For example, methods we will develop in future chapters for approximating solutions to differential equations will depend strongly on discretizations of derivatives. In

computational electrodynamics, *integral equations* for an unknown function $\phi(\vec{y})$ given a kernel $K(\vec{x}, \vec{y})$ and function $f(\vec{x})$ are expressed as the relationship

$$f(\vec{x}) = \int_{\mathbb{R}^n} K(\vec{x}, \vec{y})\phi(\vec{y})\, d\vec{y}.$$

Equations in this form are solved for ϕ to estimate electric and magnetic fields, but unless the ϕ and K are very special we cannot hope to work with such an integral in closed form. Hence, these methods typically discretize ϕ and the integral using a set of samples and then solve the resulting discrete system of equations.

In this chapter, we will develop methods for numerical integration and differentiation given a sampling of function values. We also will suggest strategies to evaluate how well we can expect approximations of derivatives and integrals to perform, helping formalize intuition for their relative quality and efficiency in different circumstances or applications.

14.1 MOTIVATION

It is not hard to encounter applications of numerical integration and differentiation, given how often the tools of calculus appear in physics, statistics, and other fields. Well-known formulas aside, here we suggest a few less obvious places requiring algorithms for integration and differentiation.

Example 14.1 (Sampling from a distribution)**.** Suppose we are given a probability distribution $p(t)$ on the interval $[0, 1]$; that is, if we randomly sample values according to this distribution, we expect $p(t)$ to be proportional to the number of times we draw a value near t. A common task is to generate random numbers distributed like $p(t)$.

Rather than develop a specialized sampling method every time we receive a new $p(t)$, it is possible to leverage a single uniform sampling tool to sample from nearly any distribution on $[0, 1]$. We define the *cumulative distribution function* (CDF) of p to be

$$F(t) = \int_0^t p(x)\, dx.$$

If X is a random number distributed evenly in $[0, 1]$, one can show that $F^{-1}(X)$ is distributed like p, where F^{-1} is the inverse of F. That is, if we can approximate F or F^{-1}, we can generate random numbers according to an arbitrary distribution p.

Example 14.2 (Optimization)**.** Most of our methods for minimizing and finding roots of a function $f(\vec{x})$ require computing not only values $f(\vec{x})$ but also gradients $\nabla f(\vec{x})$ and even Hessians $H_f(\vec{x})$. BFGS and Broyden's method build up rough approximations of derivatives of f during optimization. When f changes rapidly in small neighborhoods, however, it may be better to approximate ∇f directly near the current iterate $\vec{x}_k$ rather than using values from potentially far-away iterates $\vec{x}_\ell$ for $\ell < k$, which can happen as BFGS or Broyden slowly build up derivative matrices.

Example 14.3 (Rendering)**.** The *rendering equation* from computer graphics and ray tracing is an integral equation expressing conservation of light energy [70]. As it was originally presented, the rendering equation states:

$$I(\vec{x}, \vec{y}) = g(\vec{x}, \vec{y}) \left[\varepsilon(\vec{x}, \vec{y}) + \int_S \rho(\vec{x}, \vec{y}, \vec{z}) I(\vec{y}, \vec{z})\, d\vec{z} \right].$$

Here $I(\vec{x}, \vec{y})$ is proportional to the intensity of light going from point $\vec{y}$ to point $\vec{x}$ in a scene. The functions on the right-hand side are:

$g(\vec{x}, \vec{y})$	A *geometry term* accounting, e.g., for objects occluding the path from $\vec{x}$ to $\vec{y}$
$\varepsilon(\vec{x}, \vec{y})$	The *light emitted* directly from $\vec{x}$ to $\vec{y}$
$\rho(\vec{x}, \vec{y}, \vec{z})$	A *scattering* term giving the amount of light scattered to point $\vec{x}$ by a patch of surface at location $\vec{z}$ from light located at $\vec{z}$
$S = \cup_i S_i$	The set of surfaces S_i in the scene

Many rendering algorithms can be described as approximate strategies for solving this integral equation.

Example 14.4 (Image processing)**.** Suppose we think of an image or photograph as a function of two variables $I(x, y)$ giving the brightness of the image at each position (x, y). Many classical image processing filters can be thought of as *convolutions*, given by

$$(I * g)(x, y) = \iint_{\mathbb{R}^2} I(u, v) g(x - u, y - v)\, du\, dv.$$

For example, to blur an image we can take g to be a Gaussian or bell curve; in this case $(I*g)(x, y)$ is a weighted average of the colors of I near the point (x, y). In practice, images are sampled on discrete grids of pixels, so this integral must be approximated.

Example 14.5 (Bayes' Rule)**.** Suppose X and Y are continuously valued random variables; we can use $P(X)$ and $P(Y)$ to express the probabilities that X and Y take particular values. Sometimes, knowing X may affect our knowledge of Y. For instance, if X is a patient's blood pressure and Y is a patient's weight, then knowing a patient has high weight may suggest that he or she also has high blood pressure. In this situation, we can write *conditional* probability distributions $P(X|Y)$ (read "the probability of X given Y") expressing such relationships.

A foundation of modern probability theory states that $P(X|Y)$ and $P(Y|X)$ are related by *Bayes' rule*

$$P(X|Y) = \frac{P(Y|X)P(X)}{\int P(Y|X)P(X)\, dX}.$$

Estimating the integral in the denominator can be a serious problem in machine learning algorithms where the probability distributions take complex forms. Approximate and often randomized integration schemes are needed for algorithms in parameter selection that use this value as part of a larger optimization technique [63].

14.2 QUADRATURE

We will begin by considering the problem of numerical integration, or *quadrature*. This problem—in a single variable—can be expressed as: "Given a sampling of n points from some function $f(x)$, find an approximation of $\int_a^b f(x)\, dx$." In the previous section, we presented some applications that reduce to exactly this problem.

There are a few variations of this setup that require slightly different treatment or adaptation:

- The endpoints a and b may be fixed, or we may wish to find a quadrature scheme that efficiently can approximate integrals for many (a, b) pairs.
- We may be able to query $f(x)$ at any x but wish to approximate the integral using relatively few samples, or we may be given a list of precomputed pairs $(x_i, f(x_i))$ and are constrained to using these data points in our approximation.

These considerations should be kept in mind as we design assorted quadrature techniques.

14.2.1 Interpolatory Quadrature

Many of the interpolation strategies developed in the previous chapter can be extended to methods for quadrature. Suppose we write a function $f(x)$ in terms of a set of basis functions $\phi_i(x)$:

$$f(x) = \sum_i a_i \phi_i(x).$$

Then, we can find the integral of f as follows:

$$\begin{aligned}
\int_a^b f(x)\,dx &= \int_a^b \left[\sum_i a_i \phi_i(x)\right] dx \text{ by definition of } f \\
&= \sum_i a_i \left[\int_a^b \phi_i(x)\,dx\right] \text{ by swapping the sum and the integral} \\
&= \sum_i c_i a_i \text{ if we make the definition } c_i \equiv \int_a^b \phi_i(x)\,dx.
\end{aligned}$$

In other words, the integral of $f(x)$ written in a basis is a weighted sum of the integrals of the basis functions making up f.

Example 14.6 (Monomials)**.** Suppose we write $f(x) = \sum_k a_k x^k$. We know

$$\int_0^1 x^k\,dx = \frac{1}{k+1}.$$

Applying the formula above, we can write

$$\int_0^1 f(x)\,dx = \sum_k \frac{a_k}{k+1}.$$

In the more general notation above, we have taken $c_k = \frac{1}{k+1}$. This formula shows that the integral of $f(x)$ in the monomial basis can be computed directly via a weighted sum of the coefficients a_k.

Integration schemes derived using interpolatory basis functions are known as *interpolatory quadrature* rules; nearly all the methods we will present below can be written this way. We can encounter a chicken-and-egg problem if the integral $\int \phi_i(x)\,dx$ itself is not known in closed form. Certain methods in higher-order finite elements deal with this problem by putting extra computational time into making a high-quality numerical approximation of the integral of a single ϕ_i. Then, since all the ϕ's have similar form, these methods apply change-of-coordinates formulas to compute integrals of the remaining basis functions. The canonical integral can be approximated offline using a high-accuracy scheme and then reused during computations where timing matters.

14.2.2 Quadrature Rules

Our discussion above suggests the following form for a *quadrature rule* approximating the integral of f on some interval given a set of sample locations x_i:

$$Q[f] \equiv \sum_i w_i f(x_i).$$

Different weights w_i yield different approximations of the integral, which we hope become increasingly similar as the x_i's are sampled more densely. From this perspective, the choices of $\{x_i\}$ and $\{w_i\}$ determine a quadrature rule.

The classical theory of integration suggests that this formula is a reasonable starting point. For example, the *Riemann integral* presented in introductory calculus takes the form

$$\int_a^b f(x)\,dx \equiv \lim_{\Delta x_k \to 0} \sum_k f(\tilde{x}_k)(x_{k+1} - x_k).$$

Here, the interval $[a, b]$ is partitioned into pieces $a = x_1 < x_2 < \cdots < x_n = b$, where $\Delta x_k = x_{k+1} - x_k$, and $\tilde{x}_k$ is any point in $[x_k, x_{k+1}]$. For a fixed set of x_k's before taking the limit, this integral is in the $Q[f]$ form above.

There are many ways to choose the form of $Q[\cdot]$, as we will see in the coming section and as we already have seen for interpolatory quadrature. If we can query f for its values anywhere, then the x_i's *and* w_i's can be chosen strategically to sample f in a near-optimal way, but even if the x_i's are fixed, there exist many ways to choose the weights w_i with different advantages and disadvantages.

Example 14.7 (Method of undetermined coefficients)**.** Suppose we fix $x_1, \ldots, x_n$ and wish to find a reasonable set of weights w_i so that $\sum_i w_i f(x_i)$ approximates the integral of f for reasonably smooth $f : [a, b] \to \mathbb{R}$. An alternative to interpolatory quadrature is the *method of undetermined coefficients*. In this strategy, we choose n functions $f_1(x), \ldots, f_n(x)$ whose integrals are known, and require that the quadrature rule recovers the integrals of these functions exactly:

$$\begin{aligned}
\int_a^b f_1(x)\,dx &= w_1 f_1(x_1) + w_2 f_1(x_2) + \cdots + w_n f_1(x_n) \\
\int_a^b f_2(x)\,dx &= w_1 f_2(x_1) + w_2 f_2(x_2) + \cdots + w_n f_2(x_n) \\
&\vdots \qquad\quad \vdots \\
\int_a^b f_n(x)\,dx &= w_1 f_n(x_1) + w_2 f_n(x_2) + \cdots + w_n f_n(x_n).
\end{aligned}$$

The n expressions above create an $n \times n$ linear system of equations for the unknown w_i's.

One common choice is to take $f_k(x) \equiv x^{k-1}$, that is, to make sure that the quadrature scheme recovers the integrals of low-order polynomials. As in Example 14.6,

$$\int_a^b x^k\,dx = \frac{b^{k+1} - a^{k+1}}{k+1}.$$

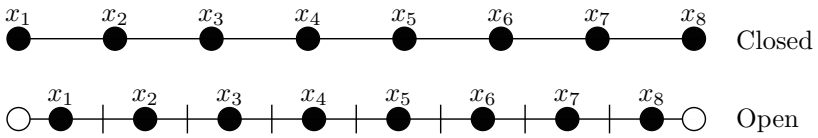

Figure 14.1 Closed and open Newton-Cotes quadrature schemes differ by where they place the samples x_i on the interval $[a, b]$; here we show the two samplings for $n = 8$.

Thus, we solve the following linear system of equations for the w_i's:

$$\begin{aligned} w_1 + w_2 + \cdots + w_n &= b - a \\ x_1 w_1 + x_2 w_2 + \cdots + x_n w_n &= \frac{b^2 - a^2}{2} \\ x_1^2 w_1 + x_2^2 w_2 + \cdots + x_n^2 w_n &= \frac{b^3 - a^3}{3} \\ \vdots \qquad & \quad \vdots \\ x_1^{n-1} w_1 + x_2^{n-1} w_2 + \cdots + x_n^{n-1} w_n &= \frac{b^n - a^n}{n}. \end{aligned}$$

In matrix form, this system is

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} b - a \\ \frac{1}{2}(b^2 - a^2) \\ \frac{1}{3}(b^3 - a^3) \\ \vdots \\ \frac{1}{n}(b^n - a^n) \end{pmatrix}.$$

This is the transpose of the Vandermonde system discussed in §13.1.1.

14.2.3 Newton-Cotes Quadrature

Quadrature rules that integrate the result of polynomial interpolation when the $x_i's$ are evenly spaced in $[a, b]$ are known as *Newton-Cotes* quadrature rules. As illustrated in Figure 14.1, there are two reasonable choices of evenly spaced samples:

- *Closed* Newton-Cotes quadrature places x_i's at a and b. In particular, for $k \in \{1, \ldots, n\}$ we take
$$x_k \equiv a + \frac{(k-1)(b-a)}{n-1}.$$
- *Open* Newton-Cotes quadrature does not place an x_i at a or b:
$$x_k \equiv a + \frac{k(b-a)}{n+1}.$$

The Newton-Cotes formulae compute the integral of the polynomial interpolant approximating the function on a to b through these points; the degree of the polynomial must be $n - 1$ to keep the quadrature rule well-defined. There is no inherent advantage to using

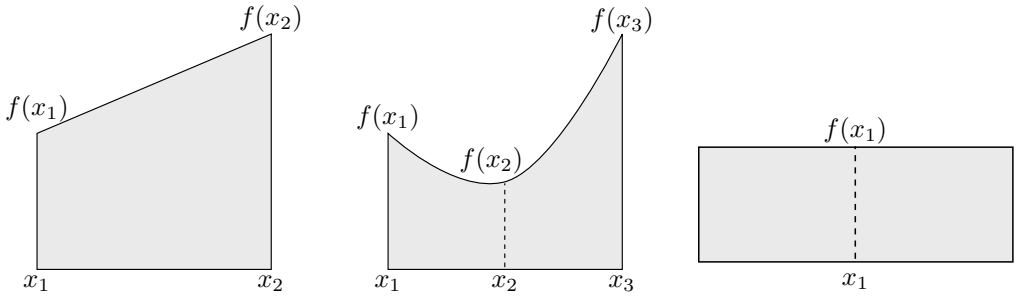

Figure 14.2 Newton-Cotes quadrature schemes; the approximated integral based on the $(x_i, f(x_i))$ pairs shown is given by the area of the gray region.

closed versus open Newton-Cotes rules; the choice between these options generally depends on which set of samples is available.

We illustrate the integration rules below in Figure 14.2. We will keep n relatively small to avoid oscillation and noise sensitivity that occur when fitting high-degree polynomials to a set of data points. Then, as in piecewise polynomial interpolation, we will then chain together small pieces into *composite* rules when integrating over a large interval $[a, b]$.

Closed rules. Closed Newton-Cotes quadrature strategies require $n \geq 2$ to avoid dividing by zero. The two lowest-order closed integrators are the most common:

- The *trapezoidal* rule for $n = 2$ (so $x_1 = a$ and $x_2 = b$) is constructed by linearly interpolating from $f(a)$ to $f(b)$. It effectively computes the area of a trapezoid via the formula
$$\int_a^b f(x)\,dx \approx (b-a)\frac{f(a)+f(b)}{2}.$$
- *Simpson's rule* is used for $n = 3$, with sample points
$$\begin{aligned} x_1 &= a \\ x_2 &= \frac{a+b}{2} \\ x_3 &= b. \end{aligned}$$
Integrating the parabola that goes through these three points yields
$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

Open rules. By far the most common rule for open quadrature is the *midpoint rule*, which takes $n = 1$ and approximates an integral with the signed area of a rectangle through the midpoint of the integration interval $[a, b]$:
$$\int_a^b f(x)\,dx \approx (b-a)f\left(\frac{a+b}{2}\right).$$

Larger values of n yield formulas similar to Simpson's rule and the trapezoidal rule.

Composite integration. We usually wish to integrate $f(x)$ with more than one, two, or three sample points x_i. To do so, we can construct a composite rule out of the midpoint or trapezoidal rules, as illustrated in Figure 14.3, by summing up smaller pieces along each interval. For example, if we subdivide $[a, b]$ into k intervals, then we can take $\Delta x \equiv \frac{b-a}{k}$ and $x_i \equiv a + (i-1)\Delta x$. Then, the composite midpoint rule is

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{k} f\left(\frac{x_{i+1} + x_i}{2}\right)\Delta x.$$

Similarly, the composite trapezoidal rule is

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \sum_{i=1}^{k}\left(\frac{f(x_i) + f(x_{i+1})}{2}\right)\Delta x \\ &= \Delta x\left(\frac{1}{2}f(a) + f(x_2) + f(x_3) + \cdots + f(x_k) + \frac{1}{2}f(b)\right) \\ &\qquad \text{after reorganizing the sum.}\end{aligned}$$

An alternative derivation of the composite midpoint rule applies the interpolatory quadrature formula from §14.2.1 to piecewise constant interpolation; the composite version of the trapezoidal rule comes from piecewise linear interpolation.

The composite version of Simpson's rule, also illustrated in Figure 14.3, chains together three points at a time to make parabolic approximations. Adjacent parabolas meet at every other x_i and may not share tangents. After combining terms, this quadrature rule becomes:

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{6}\left[f(a) + 4f(x_2) + 2f(x_3) + 4f(x_4) + 2f(x_5) + \cdots + 4f(x_k) + f(b)\right].$$

Accuracy. We have developed a number of quadrature rules that combine the same set of $f(x_i)$'s with different weights to obtain potentially unequal approximations of the integral of f. Each approximation is based on a different interpolatory construction, so it is unclear that any of these rules is better than any other. Thus, we need to compute error estimates characterizing their respective behavior. We will study the basic Newton-Cotes integrators above to show how such comparisons might be carried out.

First, consider the midpoint quadrature rule on a single interval $[a, b]$. Define $c \equiv \frac{1}{2}(a+b)$. The Taylor series of f about c is:

$$f(x) = f(c) + f'(c)(x-c) + \frac{1}{2}f''(c)(x-c)^2 + \frac{1}{6}f'''(c)(x-c)^3 + \frac{1}{24}f''''(c)(x-c)^4 + \cdots$$

After integration, by symmetry about c, the odd-numbered derivatives drop out:

$$\int_a^b f(x)\,dx = (b-a)f(c) + \frac{1}{24}f''(c)(b-a)^3 + \frac{1}{1920}f''''(c)(b-a)^5 + \cdots$$

The first term of this sum is exactly the estimate of $\int_a^b f(x)\,dx$ provided by the midpoint rule, so based on this formula we can conclude that this rule is accurate up to $O(\Delta x^3)$.

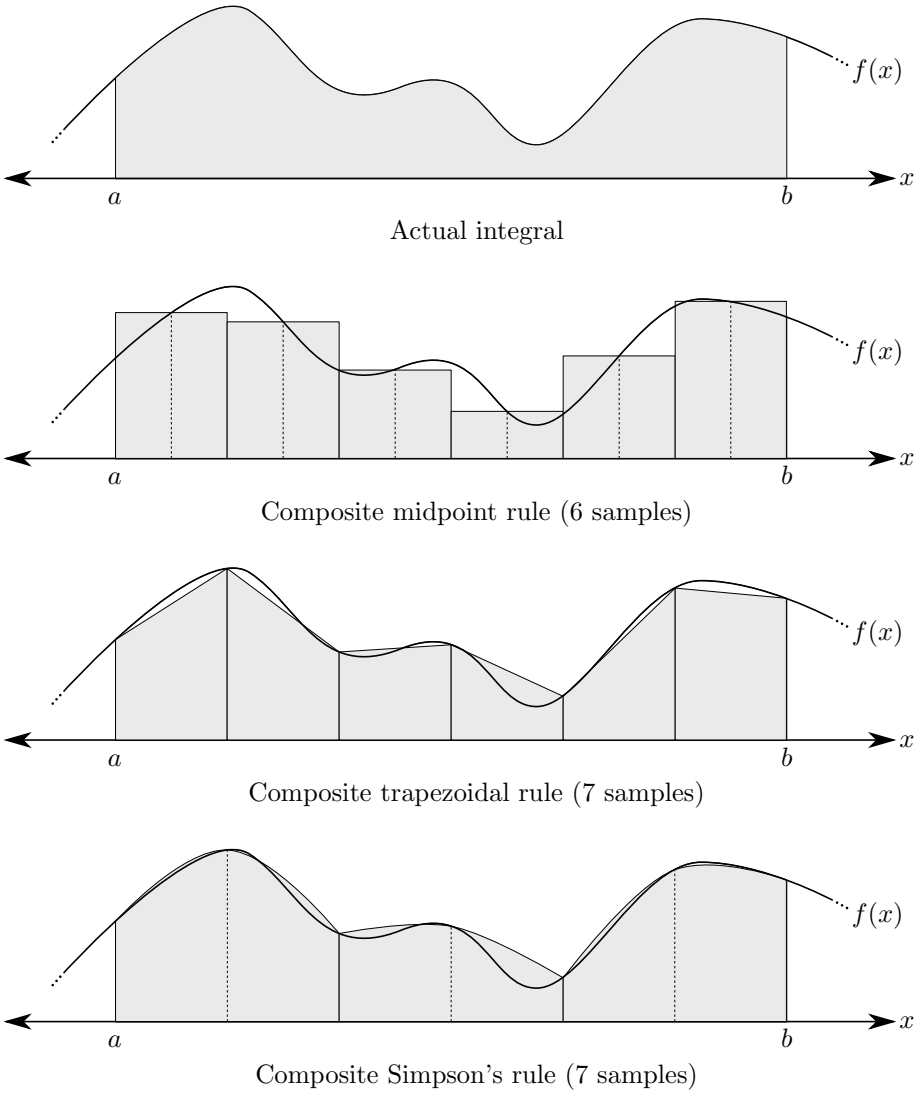

Figure 14.3 Composite Newton-Cotes quadrature rules; each rule is marked with the number of samples $(x_i, f(x_i))$ used to approximate the integral over six subintervals.

Continuing, plugging a and b into the Taylor series for $f(x)$ about c shows:

$$f(a) = f(c) + f'(c)(a-c) + \frac{1}{2}f''(c)(a-c)^2 + \frac{1}{6}f'''(c)(a-c)^3 + \cdots$$
$$f(b) = f(c) + f'(c)(b-c) + \frac{1}{2}f''(c)(b-c)^2 + \frac{1}{6}f'''(c)(b-c)^3 + \cdots$$

Adding these together and multiplying both sides by $\frac{1}{2}(b-a)$ shows:

$$\begin{aligned}(b-a)\frac{f(a)+f(b)}{2} &= f(c)(b-a) + \frac{1}{4}f''(c)(b-a)((a-c)^2 + (b-c)^2) + \cdots \\ &= f(c)(b-a) + \frac{1}{8}f''(c)(b-a)^3 + \cdots \text{ by definition of } c.\end{aligned}$$

The $f'(c)$ term vanishes for the first line by substituting $c = \frac{1}{2}(a+b)$. Now, the left-hand side is the trapezoidal rule integral estimate, and the right-hand side agrees with the Taylor series for $\int_a^b f(x)\,dx$ up to the cubic term. Hence, the trapezoidal rule is also $O(\Delta x^3)$ accurate in a single interval. A similar argument provides error estimate for Simpson's rule; after somewhat more involved algebra, one can show Simpson's rule has error scaling like $O(\Delta x^5)$.

We pause here to highlight a surprising result: The trapezoidal and midpoint rules have the same order of accuracy! Examining the third-order term shows that the midpoint rule is approximately two times more accurate than the trapezoidal rule, making it marginally preferable for many calculations. This observation seems counterintuitive, since the trapezoidal rule uses a linear approximation while the midpoint rule uses a constant approximation. As you will see in Exercise 14.1, however, the midpoint rule recovers the integrals of linear functions, explaining its extra degree of accuracy.

A notable caveat applies to this sort of analysis. Taylor's theorem only applies when Δx is *small*; otherwise, the analysis above is meaningless. When a and b are far apart, to return to the case of small Δx, we can divide $[a, b]$ into many intervals of width Δx and apply the composite quadrature rules. The total number of intervals is $^{b-a}/_{\Delta x}$, so we must multiply error estimates by $^{1}/_{\Delta x}$ in this case. Hence, the following orders of accuracy hold:

- Composite midpoint: $O(\Delta x^2)$
- Composite trapezoid: $O(\Delta x^2)$
- Composite Simpson: $O(\Delta x^4)$

14.2.4 Gaussian Quadrature

In some applications, we can choose the locations x_i where f is sampled. In this case, we can optimize not only the weights for the quadrature rule but also the locations x_i to get the highest quality. This observation leads to challenging but theoretically appealing quadrature rules, such as the Gaussian quadrature technique explored below.

The details of this technique are outside the scope of our discussion, but we provide one path to its derivation. Generalizing Example 14.7, suppose that we wish to optimize $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$ simultaneously to increase the order of an integration scheme. Now we have $2n$ instead of n unknowns, so we can enforce equality for $2n$ examples:

$$\int_a^b f_1(x)\,dx = w_1 f_1(x_1) + w_2 f_1(x_2) + \cdots + w_n f_1(x_n)$$

$$\int_a^b f_2(x)\,dx = w_1 f_2(x_1) + w_2 f_2(x_2) + \cdots + w_n f_2(x_n)$$
$$\vdots \qquad \vdots$$
$$\int_a^b f_{2n}(x)\,dx = w_1 f_{2n}(x_1) + w_2 f_{2n}(x_2) + \cdots + w_n f_{2n}(x_n).$$

Since both the x_i's and the w_i's are unknown, this system of equations is not linear and must be solved using more involved methods.

Example 14.8 (Gaussian quadrature)**.** If we wish to optimize weights and sample locations for polynomials on the interval $[-1, 1]$, we would have to solve the following system of polynomials [58]:

$$\begin{aligned} w_1 + w_2 &= \int_{-1}^{1} 1\,dx = 2 \\ w_1 x_1 + w_2 x_2 &= \int_{-1}^{1} x\,dx = 0 \\ w_1 x_1^2 + w_2 x_2^2 &= \int_{-1}^{1} x^2\,dx = \frac{2}{3} \\ w_1 x_1^3 + w_2 x_2^3 &= \int_{-1}^{1} x^3\,dx = 0. \end{aligned}$$

Systems like this can have multiple roots and other degeneracies that depend not only on the f_i's (typically polynomials) but also on the interval over which the integral is approximated. These rules are not *progressive*, in that the x_i's chosen to integrate using n data points have little in common with those used to integrate using k data points when $k \neq n$. So, it is difficult to reuse data to achieve a better estimate with this quadrature rule. On the other hand, Gaussian quadrature has the highest possible degree of accuracy for fixed n. *Kronrod* quadrature rules adapt Gaussian points to the progressive case but no longer have the highest possible order of accuracy.

14.2.5 Adaptive Quadrature

Our discussion of Gaussian quadrature suggests that the placement of the x_i's can affect the quality of a quadrature scheme. There still is one piece of information we have not used, however: the function values $f(x_i)$. That is, different classes or shapes of functions may require different integration methods, but so far our algorithms have not attempted to detect this structure into account in any serious way.

With this situation in mind, *adaptive* quadrature strategies examine the current estimate of an integral and generate new x_i's where the integrand appears to be undersampled. Strategies for adaptive integration often compare the output of multiple quadrature techniques, e.g., trapezoid and midpoint, with the assumption that they agree where sampling of f is sufficient, as illustrated in Figure 14.4. If they do not agree to some tolerance on a given interval, an additional sample point is generated and the integral estimates are updated.

Figure 14.5 outlines a bisection technique for adaptive quadrature. The idea is to subdivide intervals in which the integral estimate appears to be inaccurate recursively. This method must be accompanied with special consideration when the level of recursion is too deep, accounting for the case of a function $f(x)$ that is noisy even at tiny scale.

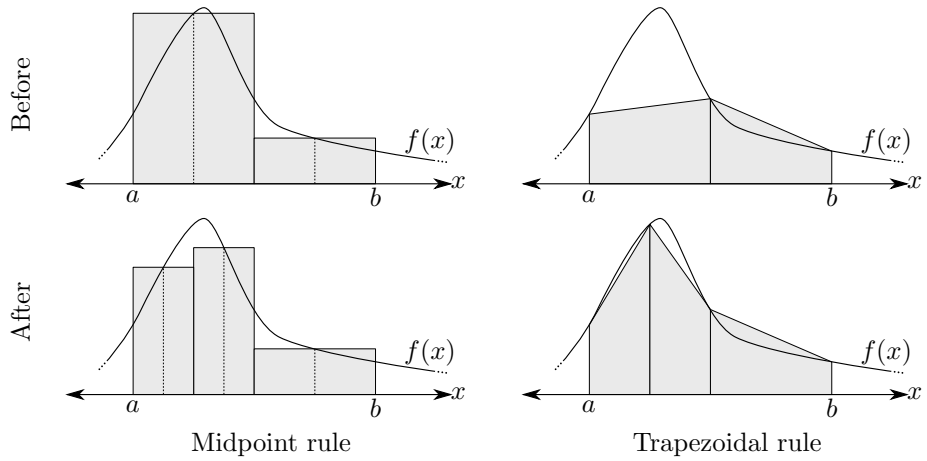

Figure 14.4 The trapezoidal and midpoint rules disagree considerably on the left subinterval (top), so adaptive quadrature methods subdivide in that region to get better accuracy (bottom).

```
function RECURSIVE-QUADRATURE(f(x), a, b, ε0)
   I ← QUADRATURE-RULE(f(x), a, b)
   E ← ERROR-ESTIMATE(f(x), I, a, b)
   if E < ε0 then
      return I
   else
      c ← ½(a + b)
      I1 ← RECURSIVE-QUADRATURE(f(x), a, c, ε0)
      I2 ← RECURSIVE-QUADRATURE(f(x), c, b, ε0)
      return I1 + I2
```

Figure 14.5 An outline for recursive quadrature via bisection. This method can use any of the quadrature rules discussed in this chapter; error estimates can be constructed, e.g., by evaluating the difference between using different quadrature rules for the same interval. The parameter ε_0 is a tolerance for the quality of the quadrature rule.

```
function MONTE-CARLO-INTEGRAL(f(x⃗), Ω ⊆ [a,b]^n, p)
  c, d ← 0          ▷ Number of points inside Ω and average value
  for k ← 1, 2, ..., p                          ▷ Sample p points
    x⃗ ← UNIFORM-RANDOM([a,b]^n)
    if INSIDE(x⃗, Ω) then                        ▷ Otherwise reject
      c ← c + 1
      d ← d + f(x⃗)
  v ← (c/p)(b − a)^n                            ▷ Estimate of |Ω|
  y ← d/c                                       ▷ Average observed f(x⃗)
return vy
```

Figure 14.6 Pseudocode for Monte Carlo integration of a function $f(\vec{x}) : \Omega \to \mathbb{R}$.

14.2.6 Multiple Variables

Many times we wish to integrate functions $f(\vec{x})$ where $\vec{x} \in \mathbb{R}^n$. For example, when $n = 2$ we might integrate over a rectangle by computing

$$\int_a^b \int_c^d f(x, y)\, dx\, dy.$$

More generally, we might wish to find an integral $\int_\Omega f(\vec{x})\, d\vec{x}$, where Ω is some subset of $\mathbb{R}^n$.

A "curse of dimensionality" makes integration more difficult as the dimension increases. The number of sample locations $\vec{x}_i$ of $f(\vec{x})$ needed to achieve comparable quadrature accuracy for an integral in $\mathbb{R}^n$ increases exponentially in n. This observation may be disheartening but is somewhat reasonable: The more input dimensions for f, the more samples are needed to understand its behavior in all dimensions.

This issue aside, one way to extend single-variable integration to $\mathbb{R}^k$ is via the *iterated integral*. For example, if $f(x, y)$ is a function of two variables, suppose we wish to find $\int_a^b \int_c^d f(x, y)\, dx\, dy$. For fixed y, we can approximate the inner integral over x using a one-dimensional quadrature rule; then, we integrate these values over y using another quadrature rule. The inner and outer integrals both induce some error, so we may need to sample the $\vec{x}_i$'s more densely than in one dimension to achieve desired output quality.

Alternatively, just as we subdivided $[a, b]$ into intervals, we can subdivide Ω into triangles and rectangles in 2D, polyhedra or boxes in 3D, and so on and use interpolatory quadrature rules in each piece. For instance, one popular option is to integrate barycentric interpolants (§13.2.2), since this integral is known in closed form.

When n is high, however, it is not practical to divide the domain as suggested. In this case, we can use the randomized *Monte Carlo method*. In the most basic version of this method, we generate k random points $\vec{x}_i \in \Omega$ with uniform probability. Averaging the values $f(\vec{x}_i)$ and scaling the result by the volume $|\Omega|$ of Ω yields an approximation of $\int_\Omega f(\vec{x})\, d\vec{x}$:

$$\int_\Omega f(\vec{x})\, d\vec{x} \approx \frac{|\Omega|}{k} \sum_{i=1}^{k} f(\vec{x}_i).$$

This approximation converges like $1/\sqrt{k}$ as more sample points are added—independent of the dimension of Ω! So, in large dimensions the Monte Carlo estimate is preferable to the deterministic quadrature methods above. A proof of convergence requires some notions from probability theory, so we refer the reader to [103] or a similar reference for discussion.

One advantage of Monte Carlo techniques is that they are easily implemented and extended. Figure 14.6 provides a pseudocode implementation of Monte Carlo integration over

a region $\Omega \subseteq [a,b]^n$. Even if we do not have a method for producing uniform samples in Ω directly, the more general integral can be carried out by sampling in the box $[a,b]^n$ and rejecting those samples outside Ω. This sampling is inappropriate when Ω is small relative to the bounding box $[a,b]^n$, since the odds of randomly drawing a point in Ω decrease in this case. To improve conditioning of this case, more advanced techniques *bias* their samples of $[a,b]^n$ based on evidence of where Ω takes the most space and where $f(\vec{x})$ is nontrivial.

Iterated integration can be effective for low-dimensional problems, and Monte Carlo methods show the greatest advantage in high dimensions. In between these two regimes, the choice of integrators is less clear. One compromise that samples less densely than iterated integration without resorting to randomization is the *sparse grid* or *Smolyak grid* method, designed to reduce the effect of the curse of dimensionality on numerical quadrature. We refer the reader to [114, 47] for discussion of this advanced technique.

14.2.7 Conditioning

We have evaluated the quality of a quadrature method by bounding its accuracy like $O(\Delta x^k)$ for small Δx. By this metric, a set of quadrature weights with large k is preferable. Another measure discussed in [58] and elsewhere, however, balances out the accuracy measurements obtained using Taylor arguments by considering the *stability* of a quadrature method under perturbations of the function being integrated.

Consider the quadrature rule $Q[f] \equiv \sum_i w_i f(x_i)$. Suppose we perturb f to some other $\hat{f}$. Define $\|f - \hat{f}\|_\infty \equiv \max_{x\in[a,b]} |f(x) - \hat{f}(x)|$. Then,

$$\begin{aligned}
\frac{|Q[f] - Q[\hat{f}]|}{\|f - \hat{f}\|_\infty} &= \frac{|\sum_i w_i (f(x_i) - \hat{f}(x_i))|}{\|f - \hat{f}\|_\infty} \\
&\leq \frac{\sum_i |w_i||f(x_i) - \hat{f}(x_i)|}{\|f - \hat{f}\|_\infty} \text{ by the triangle inequality} \\
&\leq \|\vec{w}\|_\infty \text{ since } |f(x_i) - \hat{f}(x_i)| \leq \|f - \hat{f}\|_\infty \text{ by definition.}
\end{aligned}$$

According to this bound, the most *stable* quadrature rules are those with small weights $\vec{w}$.

If we increase the order of quadrature accuracy by augmenting the degree of the polynomial used in Newton-Cotes quadrature, the conditioning bound $\|\vec{w}\|_\infty$ generally becomes less favorable. In degenerate circumstances, the w_i's even can take *negative* values, echoing the degeneracies of high-order polynomial interpolation. Thus, in practice we usually prefer composite quadrature rules summing simple estimates from many small subintervals to quadrature from higher-order interpolants, which can be unstable under numerical perturbation.

14.3 DIFFERENTIATION

Numerical integration is a relatively stable problem, in that the influence of any single value $f(x)$ on $\int_a^b f(x)\,dx$ shrinks to zero as a and b become far apart. Approximating the derivative of a function $f'(x)$, on the other hand, has no such property. From the Fourier analysis perspective, one can show that the integral $\int f(x)\,dx$ generally has lower frequencies than f, while differentiating to produce f' amplifies the frequency content of f, making sampling constraints, conditioning, and stability particularly challenging for approximating f'.

Despite the challenging circumstances, approximations of derivatives usually are relatively easy to implement and can be stable under sufficient smoothness assumptions. For

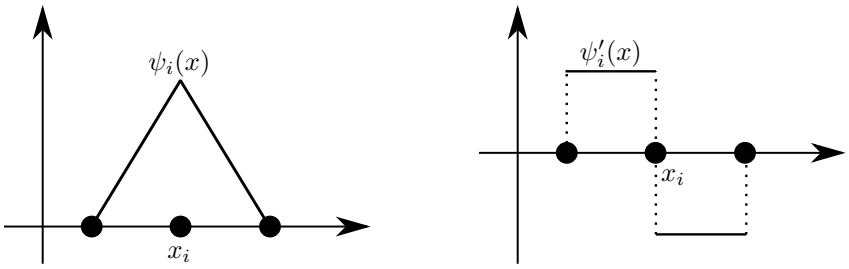

Figure 14.7 If a function is written in the basis of piecewise-linear "hat" functions $\psi_i(x)$, then its derivative can be written in the basis of piecewise constant functions $\psi_i'(x)$.

example, while developing the secant rule, Broyden's method, and so on we used approximations of derivatives and gradients to help guide optimization routines with success on a variety of objectives.

Here, we will focus on approximating f' for $f : \mathbb{R} \to \mathbb{R}$. Finding gradients and Jacobians usually is carried out by differentiating in one dimension at a time, effectively reducing to the one-dimensional problem.

14.3.1 Differentiating Basis Functions

From a mathematical perspective, perhaps the simplest use case for numerical differentiation involves functions that are constructed using interpolation formulas. As in §14.2.1, if $f(x) = \sum_i a_i \phi_i(x)$, then by linearity

$$f'(x) = \sum_i a_i \phi_i'(x).$$

In other words, the functions ϕ_i' form a *basis* for derivatives of functions written in the ϕ_i basis!

This phenomenon often connects different interpolatory schemes, as in Figure 14.7. For example, piecewise linear functions have piecewise constant derivatives, polynomial functions have polynomial derivatives of lower degree, and so on. In future chapters, we will see that this structure strongly influences discretizations of differential equations.

14.3.2 Finite Differences

A more common situation is that we have a function $f(x)$ that we can query but whose derivatives are unknown. This often happens when f takes on a complex form or when a user provides $f(x)$ as a subroutine without analytical information about its structure.

The definition of the derivative suggests a reasonable approximation

$$f'(x) \equiv \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

As we might expect, for a finite $h > 0$ with small $|h|$ the expression in the limit provides an approximation of $f'(x)$.

To substantiate this intuition, use Taylor series to write

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2} f''(x)h^2 + \cdots.$$

Rearranging this expression shows

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h).$$

Thus, the following *forward difference approximation* of f' has linear convergence:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Similarly, flipping the sign of h shows that *backward differences* also have linear convergence:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}.$$

We can improve this approximation by combining the forward and backward estimates. By Taylor's theorem,

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + \cdots \\ f(x-h) &= f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 + \cdots \\ \implies f(x+h) - f(x-h) &= 2f'(x)h + \frac{1}{3}f'''(x)h^3 + \cdots \\ \implies \frac{f(x+h) - f(x-h)}{2h} &= f'(x) + O(h^2). \end{aligned}$$

Hence, *centered differences* approximate $f'(x)$ with quadratic convergence; this is the highest order of convergence we can expect to achieve with a single divided difference. We can, however, achieve more accuracy by evaluating f at other points, e.g., $x + 2h$, at the cost of additional computation time, as explored in §14.3.3.

Approximations of higher-order derivatives can be derived via similar constructions. If we add together the Taylor expansions of $f(x+h)$ and $f(x-h)$, then

$$\begin{aligned} f(x+h) + f(x-h) &= 2f(x) + f''(x)h^2 + O(h^3) \\ \implies \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} &= f''(x) + O(h). \end{aligned}$$

To construct similar combinations for higher derivatives, one trick is to notice that our second derivative formula can be factored differently:

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h}.$$

That is, the second derivative approximation is a "finite difference of finite differences." One way to interpret this formula is shown in Figure 14.8. When we compute the forward difference approximation of f' between x and $x+h$, we can think of this slope as living at $x + h/2$; we similarly can use backward differences to place a slope at $x - h/2$. Finding the slope between these values puts the approximation back on x.

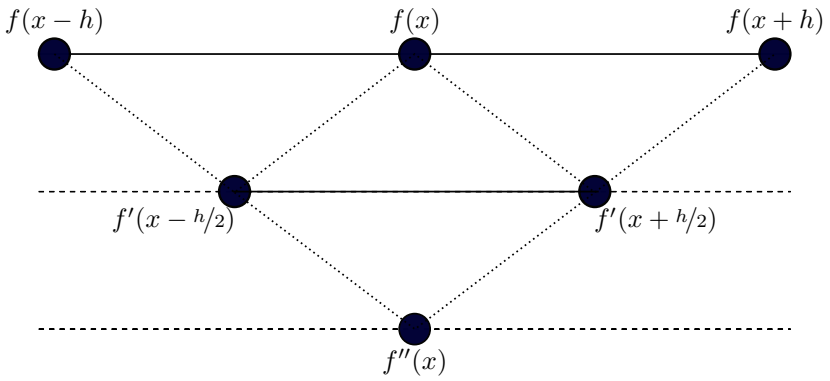

Figure 14.8 Computing the second derivative $f''(x)$ by divided differences can be thought of as applying the same divided difference rule once to approximate f' and a second time to approximate f''.

14.3.3 Richardson Extrapolation

One way to improve convergence of the approximations above is *Richardson extrapolation*. As an example of a more general pattern, suppose we wish to use forward differences to approximate $f'(x)$. For fixed $x \in \mathbb{R}$, define

$$D(h) \equiv \frac{f(x+h) - f(x)}{h}.$$

We have argued that $D(h)$ approaches $f'(x)$ as $h \to 0$. Furthermore, the difference between $D(h)$ and $f'(x)$ scales like $O(h)$.

More specifically, from our discussion in §14.3.2, $D(h)$ takes the form

$$D(h) = f'(x) + \frac{1}{2}f''(x)h + O(h^2).$$

Suppose we know $D(h)$ and $D(\alpha h)$ for some $0 < \alpha < 1$. Then,

$$D(\alpha h) = f'(x) + \frac{1}{2}f''(x)\alpha h + O(h^2).$$

We can combine these two relationships in matrix form as

$$\begin{pmatrix} 1 & \frac{1}{2}h \\ 1 & \frac{1}{2}\alpha h \end{pmatrix} \begin{pmatrix} f'(x) \\ f''(x) \end{pmatrix} = \begin{pmatrix} D(h) \\ D(\alpha h) \end{pmatrix} + O(h^2).$$

Applying the inverse of the 2×2 matrix on the left,

$$\begin{aligned} \begin{pmatrix} f'(x) \\ f''(x) \end{pmatrix} &= \begin{pmatrix} 1 & \frac{1}{2}h \\ 1 & \frac{1}{2}\alpha h \end{pmatrix}^{-1} \left[\begin{pmatrix} D(h) \\ D(\alpha h) \end{pmatrix} + O(h^2) \right] \\ &= \frac{1}{1-\alpha} \begin{pmatrix} -\alpha & 1 \\ \frac{2}{h} & -\frac{2}{h} \end{pmatrix} \left[\begin{pmatrix} D(h) \\ D(\alpha h) \end{pmatrix} + O(h^2) \right] \\ &= \frac{1}{1-\alpha} \begin{pmatrix} -\alpha & 1 \\ \frac{2}{h} & -\frac{2}{h} \end{pmatrix} \begin{pmatrix} D(h) \\ D(\alpha h) \end{pmatrix} + \begin{pmatrix} O(h^2) \\ O(h) \end{pmatrix}. \end{aligned}$$

Focusing on the first row, we took two $O(h)$ approximations of $f'(x)$ using $D(h)$ and combined them to make an $O(h^2)$ approximation! This clever technique is a method for *sequence acceleration*, improving the order of convergence of the approximation $D(h)$. The

same method is applicable to many other problems including numerical integration, as explored in Exercise 14.9. Richardson extrapolation even can be applied recursively to make higher and higher order approximations of the same quantity.

Example 14.9 (Richardson extrapolation)**.** Suppose we wish to approximate $f'(1)$ for $f(x) = \sin x^2$. To carry out Richardson extrapolation, we will use the function

$$D(h) = \frac{\sin(1+h)^2 - \sin 1^2}{h}.$$

If we take $h = 0.1$ and $\alpha = 0.5$, then

$$D(0.1) = 0.941450167\ldots$$
$$D(0.1 \cdot 0.5) = 1.017351587\ldots$$

These approximations both hold up to $O(h)$. The $O(h^2)$ Richardson approximation is

$$\frac{1}{1-0.5}\left(-0.5D(0.5) + D(0.1 \cdot 0.5)\right) = 1.0932530067\ldots$$

This approximation is a closer match to the ground truth value $f'(1) \approx 1.0806046117\ldots$.

14.3.4 Choosing the Step Size

We showed that the error of Richardson extrapolation shrinks more quickly as $h \to 0$ than the error of divided differences. We have not justified, however, why this scaling matters. The Richardson extrapolation derivative formula requires *more* arithmetic than divided differences, so at first glance it may seem to be of limited interest. That is, in theory we can avoid depleting a fixed error budget in computing numerical derivatives equally well with both formulas, even though divided differences will need a far smaller h.

More broadly, unlike quadrature, numerical differentiation has a curious property. It appears that any formula above can be arbitrarily accurate without extra computational cost by choosing a sufficiently small h. This observation is appealing from the perspective that we can achieve higher-quality approximations without additional computation time.

The catch, however, is that implementations of arithmetic operations usually are inexact. The smaller the value of h, the more similar the values $f(x)$ and $f(x+h)$ become, to the point that they are indistinguishable in finite-precision arithmetic. Dividing by very small $h > 0$ induces additional numerical instability. Thus, there is a range of h values that are not large enough to induce significant discretization error and not small enough to generate numerical problems. Figure 14.9 shows an example for differentiating a simple function in IEEE floating-point arithmetic.

Similarly, suppose as in §14.2.7 that due to noise, rather than evaluating $f(x)$, we receive perturbed values from a function $\hat{f}(x)$ satisfying $\|f - \hat{f}\|_\infty \leq \varepsilon$. Then, we can bound the error of computing a difference quotient:

$$\left|\frac{\hat{f}(x+h) - \hat{f}(x)}{h} - f'(x)\right| \leq \left|\frac{\hat{f}(x+h) - \hat{f}(x)}{h} - \frac{f(x+h) - f(x)}{h}\right| + O(h)$$

by our previous bound

$$\leq \left|\frac{(\hat{f}(x+h) - f(x+h)) - (\hat{f}(x) - f(x))}{h}\right| + O(h)$$

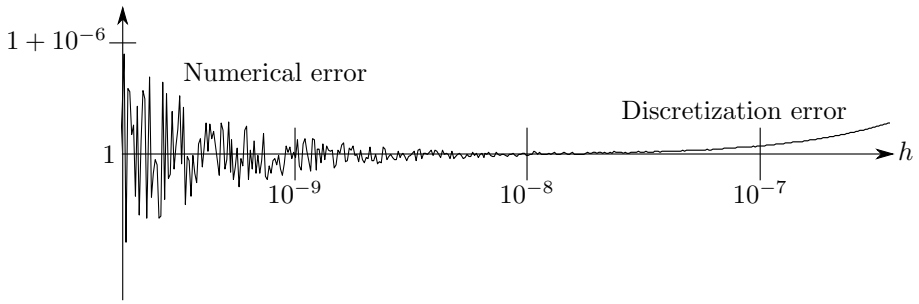

Figure 14.9 The finite difference $1/h(f(x+h)-f(x))$ as a function of h for $f(x) = x^2/2$, computed using IEEE floating-point arithmetic; when h is too small, the approximation suffers from numerical issues, while large h yields discretization error. The horizontal axis is on a logarithmic scale, and the vertical axis scales linearly.

$$\leq \frac{2\varepsilon}{h} + O(h) \text{ since } \|f - \hat{f}\|_\infty \leq \varepsilon.$$

For fixed $\varepsilon > 0$, this bound *degrades* if we take $h \to 0$. Instead, we should choose h to balance the $2\varepsilon/h$ and $O(h)$ terms to get minimal error. That is, if we cannot compute values of $f(x)$ exactly, taking larger $h > 0$ can actually *improve* the quality of the estimate of $f'(x)$. Exercise 14.6f has a similar conclusion about a method for numerical integration.

14.3.5 Automatic Differentiation

As we have seen, typical algorithms for numerical differentiation are relatively fast since they involve little more than computing a difference quotient. Their main drawback is numerical, in that finite-precision arithmetic and/or inexact evaluation of functions fundamentally limit the quality of the output. Noisy or rapidly varying functions are thus difficult to differentiate numerically with any confidence.

On the other end of the spectrum between computational efficiency and numerical quality lies the technique of *automatic differentiation* ("autodiff"), which is not subject to any discretization error [8]. Instead, this technique takes advantage of the chain rule and other properties of derivatives to compute them exactly.

"Forward" automatic differentiation is particularly straightforward to implement. Suppose we have two variables u and v, stored using floating-point values. We store alongside these variables additional values $u' \equiv du/dt$ and $v' \equiv dv/dt$ for some independent variable t; in some programming languages, we alternatively can define a new data type holding pairs of values $[u, u']$ and $[v, v']$. We can define an algebra on these pairs that encodes typical operations:

$$\begin{aligned}
[u, u'] + [v, v'] &\equiv [u + v, u' + v'] \\
c[u, u'] &\equiv [cu, cu'] \\
[u, u'] \cdot [v, v'] &\equiv [uv, uv' + u'v] \\
[u, u'] \div [v, v'] &\equiv \left[\frac{u}{v}, \frac{vu' - uv'}{v^2}\right] \\
\exp([u, u']) &\equiv [e^u, u'e^u]
\end{aligned}$$

$$\begin{aligned} \ln([u, u']) &\equiv \left[\ln u, \frac{u'}{u}\right] \\ \cos([u, u']) &\equiv [\cos u, -u' \sin u] \\ \vdots \quad & \quad \vdots \end{aligned}$$

Starting with the pair $t \equiv [t_0, 1]$—since ${}^{dt}/_{dt} = 1$—we can evaluate a function $f(t)$ and its derivative $f'(t)$ simultaneously using these rules. If they are implemented in a programming language supporting operator overloading, the additional derivative computations can be completely transparent to the implementer.

The method we just described builds up the derivative $f'(t)$ in parallel with building $y = f(t)$. "Backward" automatic differentiation is an alternative algorithm that can require fewer function evaluations in exchange for more memory usage and a more complex implementation. This technique constructs a graph representing the steps of computing $f(t)$ as a sequence of elementary operations. Then, rather than starting from the fact ${}^{dt}/_{dt} = 1$ and working forward to ${}^{dy}/_{dt}$, backward automatic differentiation starts with ${}^{dy}/_{dy} = 1$ and works backward from the same rules to replace the denominator with dt. Backward automatic differentiation can avoid unnecessary computations, particularly when y is a function of multiple variables. For instance, suppose we can write $f(t_1, t_2) = f_1(t_1) + f_2(t_2)$; in this case, backward automatic differentiation does not need to differentiate f_1 with respect to t_2 or f_2 with respect to t_1. The *backpropagation* method for neural networks in machine learning is a special case of backward automatic differentiation.

Automatic differentiation is widely regarded as an under-appreciated numerical technique, yielding *exact* derivatives of functions with minimal implementation effort. It is particularly valuable when prototyping software making use of optimization methods requiring derivatives or Hessians, avoiding having to recompute derivatives by hand every time an objective function is adjusted. The cost of this convenience, however, is computational efficiency, since in effect automatic differentiation methods do not simplify expressions for derivatives but rather apply the most obvious rules.

14.3.6 Integrated Quantities and Structure Preservation

Continuing in our consideration of alternatives to numerical differentiation, we outline an approach that has gained popularity in the geometry and computer graphics communities for dealing with curvature and other differential measures of shape.

As we have seen, a typical pattern from numerical analysis is to prove that properties of approximated derivatives hold as $\Delta x \to 0$ for some measure of spacing Δx. While this type of analysis provides intuition relating discrete computations to continuous notions from calculus, it neglects a key fact: In reality, we must fix $\Delta x > 0$. Understanding what happens in the $\Delta x > 0$ regime can be equally important to the $\Delta x \to 0$ limit, especially when taking coarse approximations. For example, in computational geometry, it may be desirable to link measures like curvature of smooth shape directly to discrete values like lengths and angles that can be computed on complexes of polygons.

With this new view, some techniques involving derivatives, integrals, and other quantities are designed with *structure preservation* in mind, yielding "discrete" rather than "discretized" analogs of continuous quantities [53]. That is, rather than asking that structure from continuous calculus emerges as $\Delta x \to 0$, we design differentiators and integrators for which certain theorems from continuous mathematics hold exactly.

One central technique in this domain is the use of *integrated quantities* to encode derivatives. As a basic example, suppose we are sampling $f(t)$ and have computed

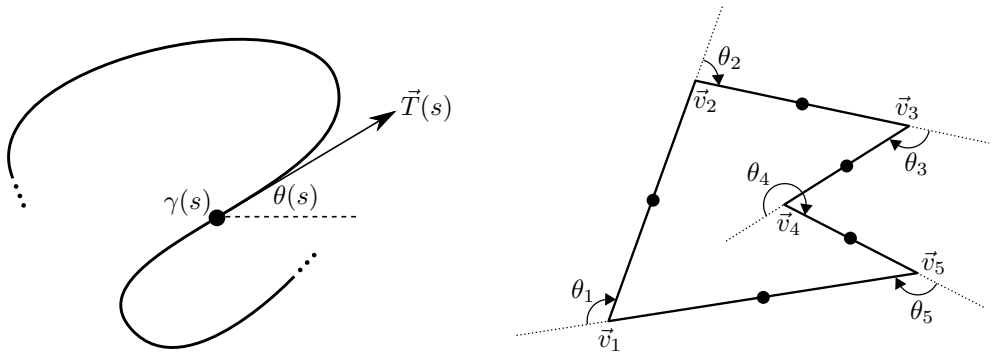

Figure 14.10 Notation for Example 14.10; each curve segment Γ_i is the union of the two half-segments adjacent to $\vec{v}_i$, bounded by the marked midpoints.

$f(t_1), f(t_2), \ldots, f(t_k)$ for some discrete set of times $t_1 < t_2 < \cdots < t_k$. Rather than using divided differences to approximate the derivative f', we can use the Fundamental Theorem of Calculus to show

$$\int_{t_i}^{t_{i+1}} f'(t)\,dt = f(t_{i+1}) - f(t_i).$$

This formula may not appear remarkable beyond first-year calculus, but it encodes a deep idea. The difference $f(t_{i+1}) - f(t_i)$ on the right side is computable *exactly* from the samples $f(t_1), f(t_2), \ldots, f(t_k)$, while the quantity on the left is an averaged version of the derivative f'. By substituting integrated versions of f' into computations whenever possible, we can carry out discrete analogs of continuous calculus for which certain theorems and properties hold exactly rather than in the limit.

Example 14.10 (Curvature of a 2D curve, [53])**.** In the continuous theory of differential geometry, a smooth curve Γ on the two-dimensional plane can be parameterized as a function $\gamma(s) : \mathbb{R} \to \mathbb{R}^2$ satisfying $\gamma'(s) \neq \vec{0}$ for all s. Assume that $\|\gamma'(s)\|_2 = 1$ for all s; such an arc length parameterization is always possible by moving along the curve with constant speed. Then, Γ has unit tangent vector $\vec{T}(s) \equiv \gamma'(s)$. If we write $\vec{T}(s) \equiv (\cos\theta(s), \sin\theta(s))$ for angle $\theta(s)$, then the *curvature* of $\gamma(s)$ is given by the derivative $\kappa(s) \equiv \theta'(s)$. This notation is illustrated in Figure 14.10 alongside notation for the discretization below.

Suppose Γ is closed, that is, $\gamma(s_0) = \gamma(s_1)$ for some $s_0, s_1 \in \mathbb{R}$. In this case, the *turning number theorem* from topology states

$$\int_{s_0}^{s_1} \kappa(s)\,ds = 2\pi k,$$

for some integer k. Intuitively, this theorem represents the fact that $\vec{T}(s_0) = \vec{T}(s_1)$, and hence θ took some number of loops around the full circle.

A typical discretization of a two-dimensional curve is as a sequence of line segments $\vec{v}_i \leftrightarrow \vec{v}_{i+1}$. Approximating $\kappa(s)$ on such a curve can be challenging, since κ is related to the second derivative γ''. Instead, suppose at each joint $\vec{v}_i$ we define the *integrated curvature* over the two half-segments around $\vec{v}_i$ to be the turning angle θ_i given by the π minus the angle between the two segments adjacent to $\vec{v}_i$.

Partition the discretization of Γ into pairs of half-segments Γ_i. Then, if Γ is closed,

$$\begin{aligned}\int_\Gamma \kappa\, ds &= \sum_i \int_{\Gamma_i} \kappa\, ds \text{ by breaking into individual terms} \\ &= \sum_i \theta_i \text{ by definition of integrated curvature} \\ &= 2\pi k,\end{aligned}$$

where the final equality comes from the fact that the discrete Γ is a polygon, and we are summing its exterior angles. That is, for this choice of discrete curvature, the turning number theorem holds *exactly* even for coarse approximations of Γ, rather than becoming closer and closer to true as the lengths $|\Gamma_i| \to 0$. In this sense, the integrated turning-angle curvature has more properties in common with the continuous curvature of a curve $\gamma(s)$ than an inexact but convergent discretization coming from divided differences.

The example above shows a typical structure-preserving treatment of a derivative quantity, in this case the curvature of a two-dimensional curve, accompanied by a discrete structure—the turning number theorem—holding without taking any limit as $\Delta x \to 0$. We have not shown, however, that the value θ_i—or more precisely some non-integrated pointwise approximation like $\theta_i/|\Gamma_i|$—actually converges to the curvature of Γ. This type of convergence does *not* always hold, and in some cases it is impossible to preserve structure exactly and converge as $\Delta x \to 0$ simultaneously [128]. Such issues are the topic of active research at the intersection of numerical methods and geometry processing.

14.4 EXERCISES

14.1 Show that the midpoint rule is exact for the function $f(x) = mx + c$ along any interval $x \in [a, b]$.

14.2 (Suggested by Y. Zhao) Derive α, β, and x_1 such that the following quadrature rule holds exactly for polynomials of degree ≤ 2 :

$$\int_0^2 f(x)\, dx \approx \alpha f(0) + \beta f(x_1).$$

14.3 Suppose we are given a quadrature rule of the form $\int_0^1 f(x)\, dx \approx af(0) + bf(1)$ for some $a, b \in \mathbb{R}$. Propose a corresponding composite rule for approximating $\int_0^1 f(x)\, dx$ given $n+1$ closed sample points $y_0 \equiv f(0), y_1 \equiv f(1/n), y_2 \equiv f(2/n), \ldots, y_n \equiv f(1)$.

14.4 Some quadrature problems can be solved by applying a suitable change of variables:

(a) Our strategies for quadrature break down when the interval of integration is not of finite length. Derive the following relationships for $f : \mathbb{R} \to \mathbb{R}$:

$$\begin{aligned}\int_{-\infty}^{\infty} f(x)\, dx &= \int_{-1}^{1} f\left(\frac{t}{1-t^2}\right)\frac{1+t^2}{(1-t^2)^2}\, dt \\ \int_0^{\infty} f(x)\, dx &= \int_0^1 \frac{f(-\ln t)}{t}\, dt \\ \int_c^{\infty} f(x)\, dx &= \int_0^1 f\left(c + \frac{t}{1-t}\right)\cdot\frac{1}{(1-t)^2}\, dt.\end{aligned}$$

How can these formulas be used to integrate over intervals of infinite length? What might be a drawback of evenly spacing t samples?

(b) Suppose $f : [-1, 1] \to \mathbb{R}$ can be written:

$$f(\cos\theta) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta).$$

Then, show:

$$\int_{-1}^{1} f(x)\,dx = a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2}.$$

This formula provides a way to integrate a function given its Fourier series [25].

14.5 The methods in this chapter for differentiation were limited to single-valued functions $f : \mathbb{R} \to \mathbb{R}$. Suppose $g : \mathbb{R}^n \to \mathbb{R}^m$. How would you use these techniques to approximate the Jacobian Dg? How does the timing of your approach scale with m and n?

14.6 ("Lanczos differentiator," [77]) Suppose $f(t)$ is a smooth function.

(a) Suppose we sample $f(t)$ at $t = kh$ for $k \in \{-n, -n+1, \ldots, 0, \ldots, n\}$, yielding samples $y_{-n} = f(-nh), y_{-n+1} = f((-n+1)h), \ldots, y_n = f(nh)$. Show that the parabola $p(t) = at^2 + bt + c$ optimally fitting these data points via least-squares satisfies

$$p'(0) = \frac{\sum_k k y_k}{h \sum_k k^2}.$$

(b) Use this formula to propose approximations of $f'(0)$ when $n = 1, 2, 3$.

(c) Motivate the following formula for "differentiation by integration":

$$f'(0) = \lim_{h\to 0} \frac{3}{2h^3} \int_{-h}^{h} t f(t)\,dt.$$

This formula provides one connection between numerical methods for integration and differentiation.

(d) Show that when $h > 0$,

$$\frac{3}{2h^3} \int_{-h}^{h} t f(t)\,dt = f'(0) + O(h^2).$$

(e) Denote $D_h f \equiv \frac{3}{2h^3} \int_{-h}^{h} t f(t)\,dt$. Suppose thanks to noise we actually observe $f^\varepsilon(t)$ satisfying $|f(t) - f^\varepsilon(t)| \le \varepsilon$ for all t. Show the following relationship:

$$|D_h f^\varepsilon - f'(0)| \le \frac{3\varepsilon}{2h} + O(h^2).$$

(f) Suppose the second term in Exercise 14.6e is bounded above by $Mh^2/10$; this is the case when $|f'''(t)| \le M$ everywhere [54]. Show that with the right choice of h, the integral approximation from Exercise 14.6e is within $O(\varepsilon^{2/3})$ of $f'(0)$.
Note: Your choice of h effectively trades off between numerical approximation error from using the "differentiation by integration" formula and noise approximating f with f^ε. This property makes the Lanczos approximation effective for certain noisy functions.

14.7 Propose an extension of forward automatic differentiation to maintaining first and second derivatives in triplets $[u, u', u'']$. Provide analogous formulas for the operations listed in §14.3.5 given $[u, u', u'']$ and $[v, v', v'']$.

14.8 The problem of numerical differentiation is challenging for noisy functions. One way to stabilize such a calculation is to consider multiple samples simultaneously [1]. For this problem, assume $f : [0, 1] \to \mathbb{R}$ is differentiable.

(a) By the Fundamental Theorem of Calculus, there exists $c \in \mathbb{R}$ such that

$$f(x) = c + \int_0^x f'(\bar{x})\, d\bar{x}.$$

Suppose we sample $f(x)$ at evenly spaced points $x_0 = 0, x_1 = h, x_2 = 2h, \ldots, x_n = 1$ and wish to approximate the first derivative $f'(x)$ at $x_1 - h/2, x_2 - h/2, \ldots, x_n - h/2$. If we label our samples of $f'(x)$ as $a_1, \ldots, a_n$, write a least-squares problem in the a_i's and an additional unknown c approximating this integral relationship.

(b) Propose a Tikhonov regularizer for this problem.

(c) We also could have written

$$f(x) = \tilde{c} - \int_x^1 f'(\bar{x})\, d\bar{x}.$$

Does your approximation of $f'(\bar{x})$ change if you use this formula?

14.9 The *Romberg* quadrature rules are derived by applying Richardson extrapolation (§14.3.3) to numerical integration. Here, we will derive Romberg integration for $f : [a, b] \to \mathbb{R}$.

(a) Suppose we divide $[a, b]$ into 2^k subintervals for $k \geq 0$. Denote by $T_{k,0}$ the result of applying the composite trapezoidal rule to $f(x)$ to this subdivision. Show that there exists a constant C dependent on f but not k such that:

$$\int_a^b f(x)\, dx = T_{k,0} + Ch^2 + O(h^4),$$

where $h(k) = (b-a)/2^k$. For this problem, you may assume that f is infinitely differentiable and that the Taylor series for f centered at any $c \in [a, b]$ is convergent.

(b) Use Richardson extrapolation to derive an estimate $T_{k,1}$ of the integral that is accurate up to $O(h^4)$.
Hint: Combine $T_{k,0}$ and $T_{k-1,0}$.

(c) Assume that the error expansion for the trapezoidal rule continues in a similar fashion:

$$\int_a^b f(x)\, dx = T_{k,0} + C_2h^2 + C_4h^4 + C_6h^6 + \cdots.$$

By iteratively applying Richardson extrapolation, propose values $T_{k,j}$ for $j \leq k$ that can be used to achieve arbitrarily high-order estimates of the desired integral.
Hint: You should be able to define $T_{k,j}$ as a linear combination of $T_{k,j-1}$ and $T_{k-1,j-1}$.

14.10 Give examples of closed and open Newton-Cotes quadrature rules with negative coefficients for integrating $f(x)$ on $[0, 1]$. What unnatural properties can be exhibited by these approximations?

14.11 Provide a sequence of differentiable functions $f_k : [0, 1] \to \mathbb{R}$ and a function $f : [0, 1] \to \mathbb{R}$ such that $\max_{x\in[0,1]} |f_k(x)-f(x)| \to 0$ as $k \to \infty$ but $\max_{x\in[0,1]} |f_k'(x)-f'(x)| \to \infty$. What does this example imply about numerical differentiation when function values are noisy? Is a similar counterexample possible for integration when f and the f_k's are differentiable?