Chapter 5

Differentiation and Integration

Two fundamental problems of univariate Calculus are Differentiation and Integration. In a first Calculus course we learn systematic approaches to Differentiation that can be used to compute almost all derivatives of interest, based on limits and on the rules of differentiation, specifically the chain rule. For Integration we learn how to "invert" differentiation (find antiderivatives) and how to compute definite integrals from their definition (through Riemann sums). We discuss first techniques for computing derivatives all based on ideas from Calculus then techniques for computing definite integrals. Finally, we discuss the facilities in MATLAB for differentiation and integration.

5.1 Differentiation

In Calculus we learned that one definition of the derivative function $f'(x)$ of a differentiable function $f(x)$ is

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

For this definition to be valid we need that $f(x)$ be defined in an interval centered on the point x, as the parameter h can be either positive or negative in this definition. If we assume a little more, for example that the function f also has a second derivative at x, then we can show using Taylor series that for h sufficiently small

$$f'(x) - \frac{f(x+h) - f(x)}{h} \approx \frac{h}{2} f''(x)$$

so as $h \to 0$ the difference quotient tends to the derivative like $O(h)$.

This forms the basis for the best known and simplest method for approximating a derivative numerically: if we want to compute the value of $f'(a)$, we approximate it using the one-sided difference $\dfrac{f(a+h) - f(a)}{h}$ for a chosen value of h. So, how do we choose h assuming that we can calculate $f(a+h)$ for any value of h? The limit definition seems to imply that we can choose it sufficiently small to achieve whatever accuracy of the computed $f'(a)$ that we desire. In practice, we are limited by the effects of catastrophic cancelation. When h is small we are proposing to calculate two values of $f(x)$ at close arguments hence guaranteeing that almost always there is heavy cancelation in which many significant digits are lost. Then, we propose to divide the result by a very small number hence grossly magnifying the effect. In practice, it turns out that values of h much smaller than about $\sqrt{\epsilon}$ cannot be used as, then, roundoff effects will dominate.

Example 5.1.1. We use the expression $f'(1) \approx \dfrac{f(1+h) - f(1)}{h}$ to approximate the derivative of $f(x) = x^4$ at $x = 1$. We choose the sequence $h = 10^{-n}$, $n = 0, 1, \ldots, 16$; note that the square root of machine precision is $\sqrt{\epsilon_{\text{DP}}} \approx 10^{-8}$. The results are displayed in the second column of table 5.1.

Observe that until rounding error begins to dominate, at about $n = 9$, the error is reduced by about a factor of ten from one row to the next just as is the step size, h; that is, the error behaves linearly (at first order) with h.

n	First Order	Second Order
0	15	8
1	4.64100000000001	4.04000000000000
2	4.06040100000000	4.00040000000000
3	4.00600400099949	4.00000399999972
4	4.00060003999947	4.00000003999923
5	4.00006000043085	4.00000000040368
6	4.00000599976025	3.99999999994849
7	4.00000060185590	4.00000000011502
8	4.00000004230350	4.00000000344569
9	4.00000033096148	4.00000010891688
10	4.00000033096148	4.00000033096148
11	4.00000033096148	4.00000033096148
12	4.00035560232936	4.00013355772444
13	3.99680288865056	3.99902333469981
14	3.99680288865056	3.99680288865056
15	4.44089209850063	4.21884749357559
16	0	2.22044604925031

Table 5.1: Derivative of x^4 at $x = 1$ by first and second order differences

The last column of table 5.1 is generated using central differences. Assuming that the function $f(x)$ has continuous third derivatives in a neighborhood of the point x, it is easy to show using Taylor series that

$$f(x+h) - f(x-h) \approx 2hf'(x) + \frac{h^3}{3}f'''(x)$$

so the error in the central difference approximation

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

behaves like h^2. We see the appropriate second order behavior in table 5.1. As we divide the step size by a factor of ten the error is divided by a factor of one hundred, until roundoff error dominates. this happens earlier than for the first order approximation because the actual error is much smaller so easier to dominate.

Problem 5.1.1. *Use Taylor series to show that* $\dfrac{f(x+h) - f(x)}{h} - f'(x) \approx \dfrac{h}{2}f''(x)$.

Problem 5.1.2. *Use Taylor series to show that central difference approximation* $f'(x) \approx \dfrac{f(x+h) - f(x-h)}{2h}$ *is second order in* h.

Problem 5.1.3. *Reproduce table 5.1. Produce a similar table for the derivative of* $f(x) = e^x$ *at* $x = 1$. *When does roundoff error begin to dominate?*

In reality we would normally calculate the derivative of a given function directly (using symbolic differentiation or automatic differentiation, see section 5.3.1). Where we need numerical differentiation is when we need an approximation to the derivative of a function which is tabulated. Suppose

we have data $\{(x_i, f_i)\}_{i=0}^N$ where we assume that the points $x_i = x_0 + ih$ are equally spaced and we assume that $f_i = f(x_i), i = 0, 1 \ldots, N$ for some unknown smooth function $f(x)$. The aim is to compute an approximation $f_i' \approx f'(x_i), i = 0, 1 \ldots, N$. The simplest approach is to use the second order centered difference $f_i' \approx \dfrac{f_{i+1} - f_{i-1}}{2h}, i = 1, 2, \ldots, N-1$ but that leaves open the question of how to produce an accurate approximation to the derivatives at the endpoints f_0' and f_N'.

How can we deal with variably spaced points x_i (and, incidentally, with derivatives at endpoints)? The simplest way to compute an approximation to f_i' is to fit a quadratic polynomial $p_2(x)$ to the data $(x_{i-1}, f_{i-1}), (x_i, f_i), (x_{i+1}, f_{i+1})$ then differentiate it and evaluate at x_i. (For f_0' proceed similarly but interpolate the data points $(x_0, f_0), (x_1, f_1), (x_2, f_2)$, differentiate, then evaluate at x_0.) Write $p_2(x) = A(x-x_i)^2 + B(x-x_i) + C$ then $f_i' \approx p_2'(x_i) = B$. Interpolating $p_2(x_j) = f_j, j = i-1, i, i+1$, and defining $h_{i+1} = x_{i+1} - x_i$ and $h_i = x_i - x_{i-1}$ we find that

$$f_i' \approx p_2'(x_i) = B = \frac{h_i^2(f_{i+1} - f_i) + h_{i+1}^2(f_i - f_{i-1})}{h_i h_{i+1}(h_{i+1} + h_i)}$$

Note, $B = \dfrac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}}$ when $h_i = h_{i+1}$. (The MATLAB function `gradient` uses this central difference approximation except at the endpoints where a one-sided difference is used.) Note, this central difference is a second order approximation to the derivative at the point midway between x_{i-1} and x_{i+1}, and this is the point x_i only when the points are equally spaced. Otherwise, to leading order the error behaves like $(h_{i+1} - h_i)$.

Problem 5.1.4. *Show that* $B = \dfrac{h_i^2(f_{i+1} - f_i) + h_{i+1}^2(f_i - f_{i-1})}{h_i h_{i+1}(h_{i+1} + h_i)}$ *in the formula for* $p_2(x) = A(x - x_i)^2 + B(x - x_i) + C$.

Problem 5.1.5. *Show, using Taylor series, that* $f'(x_i) - \dfrac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} = \dfrac{(h_{i+1} - h_i)}{2} f''(x_i)$. *to leading order.*

So, using quadratic interpolation gives a disappointing result for unequally spaced meshes. It is tempting to increase the degree of polynomial interpolation, by including additional points, to achieve a more accurate result. But, recall that high degree polynomial interpolation is potentially inaccurate and taking derivatives increases the inaccuracy. We prefer to use cubic spline interpolation to the data and to differentiate the result to give a derivative at any point in the interval defined by the data. Unlike the methods described above this is a global method; that is, we can only write the derivative in terms of all the data. If $S_{3,N}(x)$ is the not-a-knot cubic spline interpolating the data $\{(x_i, f_i)\}_{i=0}^N$, then the interpolation error behaves like $\max\limits_{i=1,\ldots,N} h_i^4$ everywhere in the interval. The error in the derivative $S_{3,N}'(x)$ behaves like $\max\limits_{i=1,\ldots,N} h_i^3$.

5.2 Integration

Consider the definite integral

$$I(f) \equiv \int_a^b f(x)\, dx$$

Assume that the function $f(x)$ is continuous on the closed interval $[a, b]$, so that the integral $I(f)$ exists. We develop efficient methods for computing approximations to the integral using only values of the integrand $f(x)$ at points $x \in [a, b]$. We'll study methods for finding integration rules, of which the midpoint rule is our first example, and for finding the error associated with these rules. We'll also consider composite versions of these rules (obtained by dividing the interval $[a, b]$ into subintervals

and using the basic rule on each) and we'll study the errors associated with them. Finally we'll see how to choose the subintervals in the composite rules adaptively to make the error as small as we need.

We need the property of the definite integral $I(f)$ that it is a linear functional[1], that is

$$I(\alpha f(x) + \beta g(x)) = \alpha I(f(x)) + \beta I(g(x))$$

for any constants α and β and any functions $f(x)$ and $g(x)$ for which the integrals $I(f(x))$ and $I(g(x))$ exist. In integral notation this equation provides the standard linearity result

$$\int_a^b (\alpha f(x) + \beta g(x))\, dx = \alpha \int_a^b f(x)\, dx + \beta \int_a^b g(x)\, dx$$

for integrals.

5.2.1 Integrals and the Midpoint Rule

To approximate the integral $I(f)$ we can integrate exactly piecewise polynomial approximations of $f(x)$ on the interval $[a, b]$. As in Figure 5.1, partition the interval $[a, b]$ into N subintervals $[x_{i-1}, x_i]$, $i = 1, 2, \cdots, N$, where $a = x_0 < x_1 < \cdots < x_N = b$ and use the **additivity property of integrals**:

$$I(f) = \int_{x_0}^{x_1} f(x)\, dx + \int_{x_1}^{x_2} f(x)\, dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)\, dx$$

The first, and simplest, approximation to $f(x)$ that we consider is a piecewise constant on a

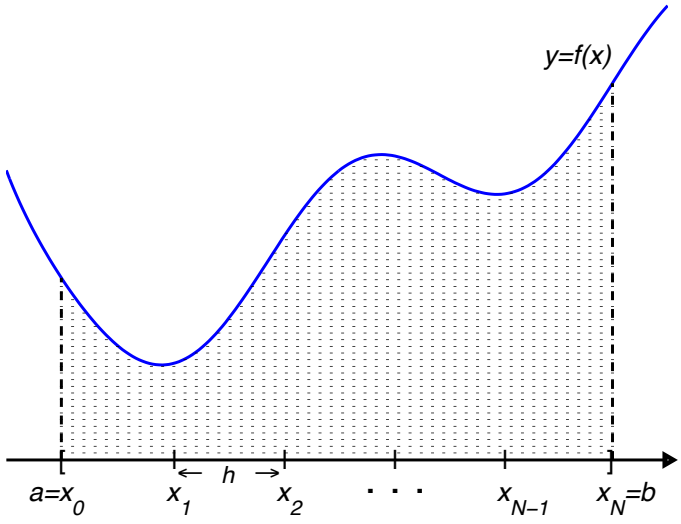

Figure 5.1: Partition of $[a, b]$ into N equal pieces of length h

subinterval. On the interval $[x_{i-1}, x_i]$, we approximate $f(x)$ by its value at the midpoint, $\frac{x_{i-1} + x_i}{2}$, of the interval. This turns out to be a pretty good choice. We have

$$\int_{x_{i-1}}^{x_i} f(x)\, dx \approx \int_{x_{i-1}}^{x_i} f\left(\frac{x_{i-1} + x_i}{2}\right) dx = (x_i - x_{i-1}) f\left(\frac{x_{i-1} + x_i}{2}\right)$$

[1] A functional maps functions to numbers. A definite integral provides a classic example of a functional; this functional assigns to each function a number that is the definite integral of that function.

which is the **midpoint rule**; the quantity $(x_i - x_{i-1})f\left(\frac{x_{i-1}+x_i}{2}\right)$ is the area of the rectangle of width $(x_i - x_{i-1})$ and height $f\left(\frac{x_{i-1}+x_i}{2}\right)$, see Figure 5.2. Hence, using the additivity property of definite integrals we obtain the **composite midpoint rule**:

$$\begin{aligned} I(f) &= \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)\,dx \\ &\approx (x_1 - x_0)f\left(\frac{x_0+x_1}{2}\right) + (x_2 - x_1)f\left(\frac{x_1+x_2}{2}\right) + \cdots + (x_N - x_{N-1})f\left(\frac{x_{N-1}+x_N}{2}\right) \\ &= \sum_{i=1}^{N}(x_i - x_{i-1})f\left(\frac{x_{i-1}+x_i}{2}\right) \end{aligned}$$

In the case when the points x_i are equally spaced, that is, $x_i \equiv a + ih$, $i = 0, 1, \cdots, N$, and $h = x_i - x_{i-1} \equiv \frac{b-a}{N}$, this formula reduces to

$$\begin{aligned} I(f) &= \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)\,dx \\ &\approx hf\left(x_{\frac{1}{2}}\right) + hf\left(x_{\frac{3}{2}}\right) + \cdots + hf\left(x_{\frac{2N-1}{2}}\right) \\ &= h\sum_{i=1}^{N} f\left(x_{\frac{2i-1}{2}}\right) \\ &\equiv R_{CM}(f,h) \end{aligned}$$

the composite midpoint rule, where for any real value s we have $x_s = a + sh$.

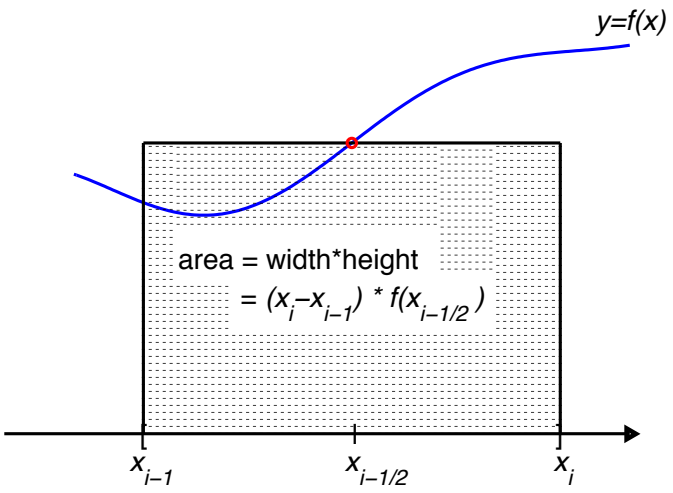

Figure 5.2: Geometry of the Midpoint Rule. The midpoint of the interval $[x_{i-1}, x_i]$ is $x_{i-\frac{1}{2}} \equiv \frac{x_{i-1}+x_i}{2}$.

A similar approach may be used to derive the **trapezoidal** rule. On the interval $[x_{i-1}, x_i]$, the constant approximation of the midpoint rule is replaced by a straight line approximation to $f(x)$ passing through the points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$. Then, the area of the trapezoid defined by this line (and the x-axis) is the approximation to the integral

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx (x_i - x_{i-1})\left(\frac{f(x_{i-1}) + f(x_i)}{2}\right),$$

see Figure 5.3.

The corresponding composite trapezoidal rule for equally spaced intervals is given by

$$\begin{aligned} I(f) &= \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)\,dx \\ &\approx h\sum_{i=1}^{N}\left(\frac{f(x_{i-1})+f(x_i)}{2}\right) \\ &\equiv R_{CT}(f,h) \end{aligned}$$

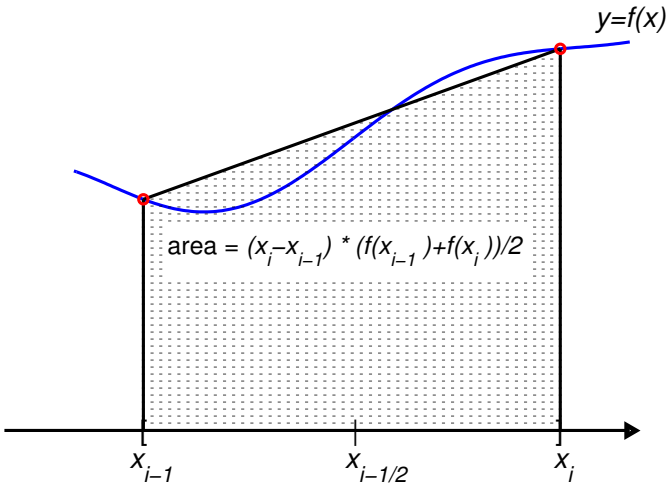

Figure 5.3: Geometry of the Trapezoidal Rule. The area of the trapezoid is the width of the interval multiplied by the average height of the trapezoid.

5.2.2 Quadrature Rules

Generally, a **quadrature rule**[2] (such as the midpoint rule for any interval of integration) has the form

$$R(f) \equiv \sum_{i=0}^{N} w_i f(x_i)$$

for given **points** $x_0 < x_1 < \cdots < x_N$ and **weights** $w_0, w_1, \cdots, w_N$.

Similarly to an integral, a quadrature rule $R(f)$ is a linear functional, that is we have a linearity rule:

$$R(\alpha f(x) + \beta g(x)) = \alpha R(f(x)) + \beta R(g(x))$$

or, in summation notation,

$$\sum_{i=0}^{N} w_i(\alpha f(x_i) + \beta g(x_i)) = \alpha \sum_{i=0}^{N} w_i f(x_i) + \beta \sum_{i=0}^{N} w_i g(x_i)$$

To approximate a definite integral $I(f)$ where we don't know an antiderivative for $f(x)$, a good choice is to integrate a simpler function $q(x)$ whose antiderivative we do know and that approximates the function $f(x)$ well. From the linearity of the functional $I(f)$, we have

$$I(q) = I(f + [q - f]) = I(f) + I(q - f)$$

[2]The term "quadrature" refers to a mathematical process that constructs a square whose area equals the area under a given curve. The problem of "squaring a circle" is an example of an ancient quadrature problem; construct a square whose area equals that of a given circle.

So that the error in approximating the true integral $I(f)$ by the integral of the approximation $I(q)$ can be expressed as

$$I(q) - I(f) = I(q - f);$$

that is, the error in approximating the definite integral of the function $f(x)$ by using the definite integral of the approximating function $q(x)$ is the definite integral of the error in approximating $f(x)$ by $q(x)$. If $q(x)$ approximates $f(x)$ well, that is if the error $q(x) - f(x)$ is in some sense small, then the error $I(q - f)$ in the integral $I(q)$ approximating $I(f)$ will also be small, because the integral of a small function is always relatively small; see Problem 5.2.1.

Example 5.2.1. Consider the definite integral $I(f) = \int_c^d f(x)\,dx$. Interpolation can be used to determine polynomials $q(x)$ that approximate the function $f(x)$ on $[c, d]$. As we have seen, the choice of the function $q_0(x)$ as a polynomial of degree $N = 0$ (that is, a constant approximation) interpolating the function $f(x)$ at the midpoint $x = \dfrac{c+d}{2}$ of the interval $[c, d]$ gives the midpoint rule.

Example 5.2.2. As another example, consider the function $q_1(x)$ which is the polynomial of degree one (that is, the straight line) that interpolates the function $f(x)$ at the integration interval endpoints $x = c$ and $x = d$; that is, the polynomial $q_1(x)$ is chosen so that the interpolating conditions

$$\begin{aligned} q_1(c) &= f(c) \\ q_1(d) &= f(d) \end{aligned}$$

are satisfied. The Lagrange form of the interpolating straight line is

$$q_1(x) = \ell_1(x)f(c) + \ell_2(x)f(d)$$

where the Lagrange basis functions are

$$\begin{aligned} \ell_1(x) &= \frac{x-d}{c-d} \\ \ell_2(x) &= \frac{x-c}{d-c} \end{aligned}$$

Because the function values $f(c)$ and $f(d)$ are constants, due to linearity we obtain

$$I(q_1) = I(\ell_1)f(c) + I(\ell_2)f(d),$$

and since $I(\ell_1) = I(\ell_2) = \dfrac{d-c}{2}$, we have

$$I(q_1) = w_1 f(c) + w_2 f(d)$$

where $w_1 = w_2 = \dfrac{d-c}{2}$. This is the **trapezoidal** rule

$$R_T(f) \equiv \frac{d-c}{2}f(c) + \frac{d-c}{2}f(d) = \frac{d-c}{2}[f(c) + f(d)]$$

Problem 5.2.1. *Consider approximating* $\int_a^b f(x)\,dx$ *by* $\int_a^b q(x)\,dx$ *where* $\max_{x\in[a,b]} |f(x) - q(x)| = \epsilon$. *Show that* $|\int_a^b f(x)\,dx - \int_a^b q(x)\,dx| \le \epsilon|b - a|$.

Problem 5.2.2. *For the basis functions* $\ell_1(x) = \dfrac{x-d}{c-d}$ *and* $\ell_2(x) = \dfrac{x-c}{d-c}$ *show that* $I(\ell_1) = I(\ell_2) = \dfrac{d-c}{2}$ *in two separate ways:*

(1) algebraically, by direct integration over the interval $[c, d]$*,*

(2) geometrically, by calculating the areas under the graphs of $\ell_1(x)$ *and* $\ell_2(x)$ *on the interval* $[c, d]$*.*

Problem 5.2.3. *Plot* $q_1(x) = \ell_1(x)f(c) + \ell_2(x)f(d)$ *and show that the area under the graph of* $q_1(x)$ *over the interval* $[c, d]$ *is the area of a trapezoid. Hence, compute its value.*

Problem 5.2.4. *Proceeding analogously to the derivation of the composite midpoint rule to derive the composite trapezoidal rule.*

Error in the Trapezoidal and Composite Trapezoidal Rules

Next, we need to see precisely how the error in approximating a definite integral $I(f)$ depends on the integrand $f(x)$. We use the trapezoidal rule to develop this analysis. We need the Integral Mean Value Theorem: If the functions $g(x)$ and $w(x)$ are continuous on the closed interval $[a, b]$ and $w(x)$ is nonnegative on the open interval (a, b), then for some point $\eta \in (a, b)$:

$$\int_a^b w(x)g(x)\,dx = \left\{\int_a^b w(x)\,dx\right\} g(\eta)$$

If the second derivative $f''(x)$ exists and is continuous on $[c, d]$, the error in the trapezoidal rule is

$$\begin{aligned} I(f) - R_T(f) &= I(f) - I(q_1) \\ &= I(f - q_1) \\ &= \int_c^d \{\text{the error in linear interpolation}\}\,dx \\ &= \int_c^d \frac{(x-c)(x-d)}{2} f''(\xi_x)\,dx \end{aligned}$$

where ξ_x is an unknown point in the interval $[c, d]$ whose location depends both on the integrand $f(x)$ and on the value of x. Here we have used the formula for the error in polynomial interpolation developed in Chapter 4. Now, since the function $-(x-c)(x-d) > 0$ for all $x \in (c, d)$ we can apply the Integral Mean Value Theorem. Thus, we obtain

$$\begin{aligned} I(f) - R_T(f) &= -\int_c^d \frac{(x-c)(d-x)}{2} f''(\xi_x)\,dx \\ &= -\frac{(d-c)^3}{12} f''(\eta) \end{aligned}$$

where η is an (unknown) point located in the open interval (c, d). Note, the point η is necessarily unknown or else the formula $I(f) = R(f) - \dfrac{(d-c)^3}{12} f''(\eta)$ could be used to evaluate *any* integral $I(f)$ exactly from explicit evaluations of $f(x)$ and of its second derivative $f''(x)$. However in the example that follows we see one way to determine the point η for some functions $f(x)$. Note that though we can determine the point η in this case, in most cases we cannot. Also, note that there may be more than one such point η.

Example 5.2.3. For $f(x) = \dfrac{(x-c)^3}{6}$ explicitly determine each of the values $f''(x)$, $R_T(f)$ and $I(f)$. Use these values to determine the "unknown" point $\eta \in (c,d)$ so that the equation

$$I(f) - R_T(f) = -\frac{(d-c)^3}{12} f''(\eta)$$

is satisfied.

Solution: For $f(x) = \dfrac{(x-c)^3}{6}$ we have $f''(x) = (x-c)$ and

$$\begin{aligned} R_T(f) &= \frac{(d-c)}{2}\left(0 + \frac{(d-c)^3}{6}\right) = \frac{(d-c)^4}{12} \\ I(f) &= \int_c^d f(x)\,dx = \frac{(d-c)^4}{24} \end{aligned}$$

Now, substituting these values in expression for the error in $R_T(f)$, we obtain

$$-\frac{1}{24}(d-c)^4 = I(f) - R_T(f) = -\frac{(d-c)^3}{12} f''(\eta) = -\frac{(d-c)^3}{12}(\eta - c)$$

so that

$$\eta - c = \frac{d-c}{2}$$

and solving this equation for η yields the explicit value

$$\eta = c + \frac{d-c}{2} = \frac{d+c}{2}$$

that is, the midpoint of the interval $[c,d]$.

Now, we develop a composite quadrature formula for $\int_a^b f(x)\,dx$ using the trapezoidal rule. Recall, $h = x_i - x_{i-1}$ and approximate $\int_{x_{i-1}}^{x_i} f(x)\,dx$ by the trapezoidal rule

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx \frac{(x_i - x_{i-1})}{2}[f(x_{i-1}) + f(x_i)] = \frac{h}{2}[f(x_{i-1}) + f(x_i)] = R_T(f)$$

So, the error is

$$I(f) - R_T(f) = \int_{x_{i-1}}^{x_i} f(x)\,dx - \frac{h}{2}[f(x_{i-1}) + f(x_i)] = -\frac{h^3}{12} f''(\eta_i)$$

where η_i is an unknown point in (x_{i-1}, x_i), and hence in (a,b). So, we have

$$I(f) \equiv \int_a^b f(x)\,dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)\,dx \approx \sum_{i=1}^{N} \frac{h}{2}[f(x_{i-1}) + f(x_i)] \equiv R_{CT}(f,h)$$

which is the **composite trapezoidal rule**.

Next, we will need the Generalized Mean Value Theorem for Sums. We'll begin with a simple example.

Example 5.2.4. (Mean Value Theorem for Sums) Show that if the function $f(x)$ is continuous on the interval $[a,b]$, if the weights w_0 and w_1 are nonnegative numbers with $w_0 + w_1 > 0$, and if the points x_0 and x_1 both lie in $[a,b]$, then there is a point $\eta \in [a,b]$ for which

$$w_0 f(x_0) + w_1 f(x_1) = \{w_0 + w_1\} f(\eta)$$

Solution: Let $m = \min_{x\in[a,b]} f(x)$ and $M = \max_{x\in[a,b]} f(x)$. These extrema exist because the function $f(x)$ is continuous on the closed interval $[a, b]$. Since the weights w_0 and w_1 are nonnegative, we have

$$\{w_0 + w_1\}\, m \le w_0 f(x_0) + w_1 f(x_1) \le \{w_0 + w_1\} M$$

Because $w_0 + w_1 > 0$, we can divide throughout by this factor to give

$$m \le \frac{w_0 f(x_0) + w_1 f(x_1)}{w_0 + w_1} \le M$$

which is a weighted average of the function values $f(x_1)$ and $f(x_2)$. Because the function $f(x)$ is continuous on $[a, b]$, the Intermediate Value Theorem (see Problem 6.1.1) states that there is a point $\eta \in [a, b]$ for which

$$f(\eta) = \frac{w_0 f(x_0) + w_1 f(x_1)}{w_0 + w_1}$$

which gives the required result.

A generalization of the above argument shows that if the function $g(x)$ is continuous on the interval $[a, b]$, the weights $\{w_i\}_{i=0}^N$ are all nonnegative numbers with $\sum_{i=0}^N w_i > 0$, and the points $\{x_i\}_{i=0}^N$ all lie in $[a, b]$, then for some point $\eta \in [a, b]$:

$$\sum_{i=0}^{N} w_i g(x_i) = \left\{ \sum_{i=0}^{N} w_i \right\} g(\eta)$$

This result, the Generalized Mean Value Theorem for Sums, may be proved directly or via a simple induction argument using the result of Example 5.2.4.

Now, assuming that $f''(x)$ is continuous on the interval $[a, b]$, the error in the composite trapezoidal rule is

$$\begin{aligned}
I(f) - R_{CT}(f) &= \int_a^b f(x)\,dx - \sum_{i=1}^{N} \frac{h}{2}[f(x_{i-1}) + f(x_i)] \\
&= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)\,dx - \sum_{i=1}^{N} \frac{h}{2}[f(x_{i-1}) + f(x_i)] \\
&= \sum_{i=1}^{N} \left(\int_{x_{i-l}}^{x_i} f(x)\,dx - \frac{h}{2}[f(x_{i-1}) + f(x_i)] \right) \\
&= -\frac{h^3}{12} \sum_{i=1}^{N} f''(\eta_i) \\
&= -\frac{h^3}{12} N f''(\eta)
\end{aligned}$$

for some unknown point $\eta \in (a, b)$. Here, we have used the Generalized Mean Value Theorem for Sums. Now, $Nh = (b - a)$, so this last expression reduces to

$$I(f) - R_{CT}(f) = -\frac{h^2}{12}(b - a) f''(\eta)$$

So, as $N \to \infty$ and $h \to 0$ simultaneously in such a way that the value $Nh = (b - a)$ is fixed, the error in the composite trapezoidal rule decreases like h^2.

Problem 5.2.5. *For $f(x) = \dfrac{(x-c)^2}{2}$, verify that the formula for the error in $R_T(f)$ in Example 5.2.3 gives the correct result. That is, calculate the error as the difference between the rule for this integrand $f(x)$ and the integral for this integrand, and show that you get the same expression as you do by evaluating the error expression for this integrand.*

Problem 5.2.6. *Argue that the trapezoidal rule is exact when used to find the area under any straight line. Use two approaches:*

(1) Use the polynomial interpolation uniqueness theorem to determine the error explicitly.

(2) Exploit the fact that the error expression depends on the second derivative $f''(x)$.

Problem 5.2.7. *Compute $\int_0^1 (x^2 + x - 1)\,dx$ and approximate it using the trapezoidal rule. Hence, compute the error exactly and also compute it from the error expression.*

Problem 5.2.8. *This problem asks you to repeat the steps in the error analysis of the composite trapezoidal rule outlined in this section.*

(1) Use the formula for the error in polynomial interpolation to show that the error in the trapezoidal rule is

$$\int_c^d f(x)\,dx - \frac{(d-c)}{2}[f(c) + f(d)] = -\frac{1}{2}\int_c^d w_1(x) f''(\xi_x)\,dx$$

for some point $\xi_x \in (c, d)$, where $w_1(x) = (x - c)(d - x)$

(2) Use the Mean Value Theorem for integrals to show that, for some point $\eta \in (c, d)$, we have

$$\int_c^d w_1(x) f''(x)\,dx = f''(\eta)\int_c^d w_1(x)\,dx$$

(3) Finally, show that $\int_c^d w_1(x)\,dx = \dfrac{(d-c)^3}{6}$

Problem 5.2.9. *By similar arguments to those used to develop the error in the composite trapezoidal rule, show that for the integral $I(f) = \int_a^b f(x)\,dx$ and step size $h = \frac{b-a}{N}$, the error in the composite midpoint rule $R_{CM}(f)$ is given by*

$$I(f) - R_{CM}(f) = \frac{h^2}{24}(b - a) f''(\eta)$$

for some unknown point $\eta \in (a, b)$. [Hint: Recall, for the integral $I(f) = \int_c^d f(x)\,dx$ the midpoint rule is $R_M(f) = (d - c) f\left(\frac{c+d}{2}\right)$. You should use the fact that the error is $I(f) - R_M(f) = \frac{(d-c)^3}{24} f''(\tau)$ for some unknown point $\tau \in (c, d)$.]

Interpolatory Quadrature

Rather than add points in the interval $[a, b]$ by making composite versions of simple rules such as the midpoint and trapezoidal rules, we may also generalize these rules by adding more interpolation points and using a higher degree interpolating polynomial. Let $x_0 < x_1 < \cdots < x_N$ and let $q_N(x)$ be the polynomial of degree N interpolating the data $\{(x_i, f(x_i))\}_{i=0}^N$. The Lagrange form of the interpolating polynomial is

$$q_N(x) = \sum_{i=0}^N f(x_i)\ell_i(x)$$

where the Lagrange basis functions $\ell_i(x)$ are defined in Chapter 4. Exploiting linearity, we have

$$I(f) \approx I(q_N) = I(\sum_{i=0}^N f(x_i)\ell_i(x)) = \sum_{i=0}^N I(\ell_i(x)) f(x_i) = \sum_{i=0}^N w_i f(x_i) \equiv R(f)$$

where the weights

$$w_i = I(\ell_i(x)) \equiv \int_a^b \ell_i(x)\,dx$$

$R(f)$ is an **interpolatory quadrature rule**. When the nodes x_i are equally spaced in $[a,b]$, so $x_i = a + ih$ where $h \equiv \dfrac{b-a}{N}$, we obtain the Newton–Cotes rules. In particular, the *closed* $(N+1)$–point Newton–Cotes rule has the points $x_0, x_1, \cdots, x_N$ as points; note, this list *includes* the interval endpoints $x_0 = a$ and $x_N = b$. The *open* $(N-1)$–point Newton–Cotes rule has the points $x_1, x_2, \cdots, x_{N-1}$ as points; note, this list *excludes* the endpoints $x_0 = a$ and $x_N = b$.

The open 1–point Newton–Cotes rule is the midpoint rule, the closed 2–point Newton–Cotes rule is the trapezoidal rule. The closed 3–point Newton–Cotes rule is **Simpson's rule**:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

The closed 4–point Newton–Cotes rule is Weddle's rule, see Problem 5.2.17.

Recall that $R(f) = I(q)$, so $I(f) - R(f) = I(f) - I(q) = I(f-q)$. Hence, $R(f)$ cannot accurately approximate $I(f)$ when $I(f-q)$ is large, which can occur only when $f-q$ is large. This can happen when using many equally spaced interpolation points, see Runge's example in Section 4.2. Integrating the polynomial interpolant used there to approximate $f(x) = \dfrac{1}{1+x^2}$ would correspond to using an 11–point closed Newton–Cotes rule to approximate $\int_{-5}^{5} f(x)\,dx$, with the possibility of a resulting large error.

However, $f-q$ can be large yet $I(f-q)$ be zero. Consider the error in the midpoint and Simpson's rules. The midpoint rule is derived by integrating a constant interpolating the data $\left(\dfrac{(a+b)}{2}, f\left(\dfrac{(a+b)}{2}\right)\right)$. This interpolant is exact only for constants, so we would anticipate that the error would be zero only for constant integrands because only then does $f-q \equiv 0$. But, like the trapezoidal rule, the midpoint rule turns out to be exact for all straight lines, see Problem 5.2.13. How can a polynomial approximation $q(x)$ of $f(x)$ that is exact for only constants yield a quadrature rule that is also exact for all straight lines? Similarly, Simpson's rule may be derived by integrating a quadratic interpolating function to the data $(a, f(a)), \left(\dfrac{(a+b)}{2}, f\left(\dfrac{(a+b)}{2}\right)\right), (b, f(b))$, see Problem 5.2.12. This quadratic interpolating function is exact for all functions f that are quadratic polynomials, yet the quadrature rule derived from integrating this interpolating function turns out to be exact for all cubic polynomials, see Problem 5.2.13. How can a polynomial approximation q of f that is exact for only quadratic polynomials yield a quadrature rule that is also exact for all cubic polynomials? In both of these cases $I(f-q) = 0$ when $f-q$ is not identically zero and, indeed, some values of $f(x) - q(x)$ are potentially large.

These rules exhibit a form of *superconvergence*; that is, the rules integrate exactly all polynomials of a certain higher degree than is to be anticipated from their construction. Indeed, all Newton–Cotes rules (closed or open) with an odd number of points exhibit this type of superconvergence; that is, they each integrate exactly all polynomials of degree one higher than the degree of the polynomial integrated to derive the rule. Gaussian quadrature rules, to be discussed in Section 5.2.4, yield the ultimate in superconvergent quadrature rules, integrating polynomials of degree almost twice the number of points.

Problem 5.2.10. *In general, an interpolatory quadrature rule based on $N+1$ nodes integrates exactly all polynomials of degree N. Why?*

Problem 5.2.11. *What is the highest degree general polynomial that*

(1) the $(N+1)$–point closed Newton–Cotes rules will integrate exactly?

(2) the $(N-1)$–point open Newton–Cotes rules will integrate exactly?

Problem 5.2.12. *Derive Simpson's rule for $\int_{-1}^{1} f(x)\,dx$ by constructing and integrating a quadratic interpolating polynomial to the data $(-1, f(-1))$, $(0, f(0))$ and $(1, f(1))$.*

Problem 5.2.13. *Consider the midpoint and Simpson's rules for the interval $[-1, 1]$. Show that*

- *the midpoint rule is exact for all straight lines*
- *Simpson's rule is exact for all cubic polynomials*

Degree of Precision and the Method of Undetermined Coefficients

Here we present an alternative, but related, way to derive quadrature rules and a theorem which simplifies determining the error in some rules.

Degree of Precision

Definition 5.2.1. Degree of precision (DOP). *The rule $R(f) = \sum_{i=0}^{N} w_i f(x_i)$ approximating the definite integral $I(f) = \int_a^b f(x)\,dx$ has DOP $= m$ if $\int_a^b f(x)\,dx = \sum_{i=0}^{N} w_i f(x_i)$ whenever $f(x)$ is a polynomial of degree at most m, but $\int_a^b f(x)\,dx \neq \sum_{i=0}^{N} w_i f(x_i)$ for some polynomial $f(x)$ of degree $m+1$.*

The following theorem gives an equivalent test for the DOP as the above definition.

Theorem 5.2.1. The rule $R(f) = \sum_{i=0}^{N} w_i f(x_i)$ approximating the definite integral $I(f) = \int_a^b f(x)\,dx$ has DOP $= m$ if $\int_a^b x^r\,dx = \sum_{i=0}^{N} w_i x_i^r$ for $r = 0, 1, \cdots, m$, but $\int_a^b x^{m+1}\,dx \neq \sum_{i=0}^{N} w_i x_i^{m+1}$.

From a practical point of view, if a quadrature rule $R_{hi}(f)$ has a higher DOP than another rule $R_{lo}(f)$, then $R_{hi}(f)$ is generally considered more accurate than $R_{lo}(f)$ because it integrates exactly higher degree polynomials and hence potentially integrates exactly more accurate polynomial approximations to f. (In practice, $R_{lo}(f)$ is sometimes more accurate than $R_{hi}(f)$, but for most integrands $f(x)$ the rule $R_{hi}(f)$ will be more accurate than the rule $R_{lo}(f)$.) These considerations will be important in our discussion of adaptive integration in Section 5.2.3.

The DOP concept may be used to derive quadrature rules directly using the *Method of Undetermined Coefficients*. With the points $x_i, i = 0, 1, \cdots, N$, given, consider the rule $I(f) = \int_{-1}^{1} f(x)\,dx \approx \sum_{i=0}^{N} w_i f(x_i) = R(f)$. Note that we have chosen a special (canonical) interval $[-1, 1]$ here. The weights (the undetermined coefficients), $w_i, i = 0, 1, \cdots, N$, are chosen to maximize the DOP, by solving the following *equations of precision* (starting from the first and leaving out no equations)

$$\begin{aligned}
\int_{-1}^{1} 1\,dx &= \sum_{i=0}^{N} w_i 1 \\
\int_{-1}^{1} x\,dx &= \sum_{i=0}^{N} w_i x_i \\
&\vdots \\
\int_{-1}^{1} x^m\,dx &= \sum_{i=0}^{N} w_i x_i^m
\end{aligned}$$

for the weights w_i. Suppose each of these equations is satisfied, but the next equation is not satisfied; that is,

$$\int_{-1}^{1} x^{m+1}\,dx \neq \sum_{i=0}^{N} w_i x_i^{m+1}\,.$$

Then the DOP corresponds to the last power of x for which we succeeded in satisfying the corresponding equation of precision, so DOP$= m$.

Example 5.2.5. Suppose that the quadrature rule

$$R(f) = w_0 f(-1) + w_1 f(0) + w_2 f(+1)$$

estimates the integral $I(f) \equiv \int_{-1}^{1} f(x)\,dx$. What choice of the weights w_0, w_1 and w_2 maximizes the DOP of the rule?
Solution: Create a table listing the values of $I(x^m)$ and $R(x^m)$ for $m = 0, 1, 2, \cdots$.

m	$I(x^m)$	$R(x^m)$
0	2	$w_0 + w_1 + w_2$
1	0	$-w_0 + w_2$
2	$\frac{2}{3}$	$w_0 + w_2$
3	0	$-w_0 + w_2$
4	$\frac{2}{5}$	$w_0 + w_2$

To determine the three free parameters, w_0, w_1 and w_2, we solve the first three equations of precision (to give us DOP≥ 2). That is we solve $I(x^m) = R(x^m)$ for $m = 0, 1, 2$:

$$\begin{aligned} 2 &= w_0 + w_1 + w_2 \\ 0 &= -w_0 + w_2 \\ \frac{2}{3} &= w_0 + w_2 \end{aligned}$$

These three equations have the unique solution (corresponding to Simpson's rule):

$$w_0 = w_2 = \frac{1}{3}, w_1 = \frac{4}{3}.$$

However, this rule has DOP $= 3$ not DOP $= 2$ because for this choice of weights $I(x^3) = R(x^3)$ too; that is, the first four equations of precision are satisfied. (Indeed, $I(x^m) = R(x^m) = 0$ for all odd powers $m \geq 0$.) DOP$= 3$ because if DOP$= 4$ then the equations $w_0 + w_2 = \frac{2}{3}$ and $w_0 + w_2 = \frac{2}{5}$ would both be satisfied, a clear contradiction.

Problem 5.2.14. *Use the linearity of integrals and quadrature rules to prove that the two definitions of DOP are equivalent.*

Problem 5.2.15. *For the integral $\int_{-1}^{1} f(x)\,dx$, find the DOP of each of the trapezoidal, midpoint and Simpson's rule.*

Problem 5.2.16. *Derive the midpoint and trapezoidal rules by fixing the points x_i and computing the weights w_i to maximize the DOP by the method of undetermined coefficients. What is the DOP in each case?*

Problem 5.2.17. *The four–point closed Newton–Cotes rule is also known as* Weddle's rule *or the* $\frac{3}{8}$th's rule. *For the interval $[-1, 1]$ use the equally spaced points $x_0 = -1$, $x_1 = -\frac{1}{3}$, $x_2 = +\frac{1}{3}$ and $x_3 = +1$ and determine the weights w_0, w_1, w_2 and w_3 to maximize the DOP of the rule. What is the DOP?*

Problem 5.2.18. *The two–point open Newton–Cotes rule uses points x_1 and x_2 of Problem 5.2.17. Determine the weights w_1 and w_2 to maximize the DOP of the rule. What is the DOP?*

Peano's Theorem and the Error in Integration

The next theorem, due to Peano, relates the error in using a quadrature rule to the DOP of the rule.

Theorem 5.2.2. *Peano's theorem.* Let the rule $R(f) \equiv \sum_{i=0}^{N} w_i f(x_i)$ approximating the integral $I(f) = \int_a^b f(x)\,dx$ have DOP$= m$. Suppose that the integrand $f(x)$, and its first $m+1$ derivatives $f'(x), f''(x), \cdots, f^{(m+1)}(x)$ exist and are continuous on the closed interval $[a,b]$. Then, there exists a function $K(x)$, the Peano kernel, that does not depend on the integrand $f(x)$ nor on its derivatives, for which

$$I(f) - R(f) = \int_a^b K(x) f^{(m+1)}(x)\,dx$$

When the Peano kernel $K(x)$ does not change sign on the interval of integration, using the Integral Mean Value Theorem it can be shown that there is a simpler relation

$$I(f) - R(f) = \kappa f^{(m+1)}(\eta)$$

where η is some unknown point in (a,b). In this relation, *the Peano constant* κ does not depend on the integrand f nor on its derivatives. The kernel $K(x)$ does not change sign on the interval of integration for the trapezoidal, midpoint or Simpson rules. Also, it does not change sign for the Gauss and Lobatto rules to be met later.

To calculate κ, we may use this relation directly with the special choice of integrand $f(x) = x^{m+1}$, as in the example that follows. To see this, observe that since κ is a constant we need to substitute for f in the relation a function whose $(m+1)^{\text{st}}$ derivative is a constant whatever the value of η. The only possible choice is a polynomial of exact degree $m+1$. We make the simplest such choice, $f(x) = x^{m+1}$. Observe that if we have just checked the DOP of the rule we will already have calculated m, $I(x^{m+1})$ and $R(x^{m+1})$.

Example 5.2.6. Calculate the Peano constant κ for Simpson's rule $R(f) = \dfrac{1}{3}\{f(-1) + 4f(0) + f(1)\}$ estimating the integral $I(f) = \displaystyle\int_{-1}^{1} f(x)\,dx$.
Solution: Simpson's rule has DOP $= 3$, so Peano's theorem tells us that

$$I(f) - R(f) = \int_{-1}^{1} K(x) f^{(4)}(x)\,dx = \kappa f^{(4)}(\eta)$$

Choose the integrand $f(x) = x^4$ so that the fourth derivative $f^{(4)}(x) = 4! = 24$ no longer involves x. From Example 5.2.5, $I(x^4) = \dfrac{2}{5}$ and $R(x^4) = \dfrac{2}{3}$, so $\dfrac{2}{5} - \dfrac{2}{3} = 24\kappa$; that is, $\kappa = -\dfrac{1}{90}$.

A higher DOP for a quadrature rule is associated with a larger number of integrand evaluations, but as the DOP increases the rule becomes more accurate. So, not surprisingly, greater accuracy comes at higher cost. The appropriate balance between cost and accuracy is problem dependent, but most modern, general purpose software (with the notable exception of MATLABś `integral`) uses integration rules of a higher DOP than any we have encountered so far. However, this software does not use the high DOP Newton–Cotes rules (that is, those with many points) because the weights, w_i, of these rules oscillate in sign for large numbers of points N. This oscillation in the weights can lead to (catastrophic) cancelation when summing the rule for smooth slowly changing integrands hence resulting in a significant loss of numerical accuracy.

We end this section with a reworking of Example 5.2.6 for an interval of integration of length h. This way we can observe the behavior of the error as $h \to 0$. This is of use when considering composite rules made up of rules on N intervals each of length h as for the composite trapezoidal and midpoint rules. Also this analysis enables us to understand the effect of bisecting intervals in the globally adaptive integration algorithm to be described in the next section.

Example 5.2.7. Use the DOP approach to determine Simpson's rule and the Peano constant, for approximating the integral $I(f) = \int_{-\frac{h}{2}}^{\frac{h}{2}} f(x)\,dx$.

Solution: Simpson's rule for approximating $I(f)$ has the form $R(f) = w_0 f(-\frac{h}{2}) + w_1 f(0) + w_2 f(\frac{h}{2})$. The usual table is

m	$I(x^m)$	$R(x^m)$
0	h	$w_0 + w_1 + w_2$
1	0	$(-w_0 + w_2)\frac{h}{2}$
2	$\frac{h^3}{12}$	$(w_0 + w_2)\frac{h^2}{4}$
3	0	$(-w_0 + w_2)\frac{h^3}{8}$
4	$\frac{h^5}{80}$	$(w_0 + w_2)\frac{h^4}{16}$

The first three equations of precision

$$\begin{aligned} h &= w_0 + w_1 + w_2 \\ 0 &= -w_0 + w_2 \\ \frac{h}{3} &= w_0 + w_2 \end{aligned}$$

have the unique solution

$$w_0 = w_2 = \frac{h}{6}, \quad w_1 = \frac{4h}{6}$$

Simpson's rule has DOP = 3 not DOP = 2 because $I(x^3) = R(x^3) = 0$. Peano's theorem tells us that the error

$$I(f) - R(f) = \kappa f^{(4)}(\eta)$$

To determine Peano's constant κ consider the special choice $f(x) = x^4$. From the table above, $I(x^4) = \frac{h^5}{80}$ and $R(x^4) = \left(\frac{h}{3}\right)\frac{h^4}{16} = \frac{h^5}{48}$, so $\frac{h^5}{80} - \frac{h^5}{48} = 24\kappa$; that is, $\kappa = -\frac{h^5}{2880}$.

Problem 5.2.19. *Calculate Peano's constant κ for the trapezoidal rule approximating the integral $I(f) = \int_a^b f(x)\,dx$. [Consider the special choice of integrand $f(x) = x^{m+1}$ where m is the DOP of the trapezoidal rule.]*

Problem 5.2.20. *Calculate Peano's constant κ for the midpoint rule when used to approximate $I(f) = \int_{-h}^{h} f(x)\,dx$.*

Problem 5.2.21. *For any quadrature rule with DOP $= m$ approximating $\int_{-\frac{h}{2}}^{\frac{h}{2}} f(x)\,dx$ and with Peano kernel that does not change sign on $[-\frac{h}{2}, \frac{h}{2}]$, show that the error has the form $\bar{\kappa} h^{m+2} f^{(m+1)}(\eta)$ where $\bar{\kappa}$ is a constant independent of the integrand $f(x)$ and of h. [Hint: We can assume the error has the form $\kappa f^{(m+1)}(\eta)$ then we show that κ always has a factor h^{m+2}.]*

Problem 5.2.22. *Write down a composite Simpson rule for the integral $\int_a^b f(x)\,dx$ where Simpson's rule is to be used on each subinterval $[x_{i-1}, x_i]$ of the interval $[a, b]$. Derive the error term for the composite rule. [Hint: Use the Generalized Mean Value Theorem for sums and the analysis of the error in Simpson's rule in Example 5.2.7.]*

5.2.3 Adaptive Integration

In a course on the calculus of one variable, we meet limits of Riemann sums as a method for defining definite integrals. The composite midpoint, trapezoidal and Simpson's rules are all Riemann sums, as are the Gauss and Lobatto rules that we will meet later.

Thinking in terms of the composite midpoint rule, if we let $h \to 0$, by exploiting the fact that the rule is a Riemann sum we can show that the integral exists, assuming only that the function f is continuous on the interval of integration. Also, this observation shows that as long as the function f is continuous on the interval of integration, using the composite midpoint rule with a sufficiently small step size h will give an accurate approximation to the integral. But, it gives us no information as to the rate of convergence of the composite midpoint rule to the integral as $h \to 0$. In contrast, from the expression for the error in the composite midpoint rule we know that if the second derivative of the integrand $f''(x)$ is continuous on the interval of integration then the composite midpoint rule converges to the integral at a rate proportional to h^2 as $h \to 0$. So, additional smoothness in the integrand $f(x)$ ensures a reasonable rate of convergence. Indeed, if we only know that the first derivative of the integrand $f'(x)$ is continuous on the interval of integration (that is, we have no information about the second derivative), we can use a form Peano's theorem to show that the composite midpoint rule converges to the integral at a rate proportional at least to h as $h \to 0$. However, more smoothness (e.g. continuous third derivative of the integrand) does not permit us to improve on the $O(h^2)$ convergence rate.

For difficult integrals, that is for integrals where the integrand is not slowly varying everywhere on the interval of integration, it is tempting to use a composite quadrature rule with subintervals of equal length h, since we know, from the discussion above, that, for the composite quadrature rules we have seen, making $h \to 0$ also forces the error to zero. However, this approach is usually very inefficient. Subintervals sufficiently short that the quadrature rule will integrate accurately the integrand where it is worst behaved are usually far too short for those parts of the integration interval $[a, b]$ where the integrand is relatively well behaved; that is, the error on these latter subintervals will be very small and determining the integral this unnecessarily accurately on these subintervals can be very inefficient, "wasting" many integrand evaluations. So, most modern, general purpose codes for integration are **adaptive**. That is, they adapt to the integrand's behavior the length of the subintervals on which the quadrature rules are applied, hence aiming to use a small number of subintervals of varying and appropriate lengths, and so not to "waste" integrand evaluations. The two basic types of adaptive algorithms are *global* and *local*. To illustrate the use of quadrature rules and to identify what else we need to study, we outline a **global adaptive integration algorithm**.

Let the problem be to estimate the integral $I(f, T) = \int_T f(x)\,dx$ where T is the interval of integration. Suppose we have quadrature rules $R_1(f, T^*)$ and $R_2(f, T^*)$ that each approximate the integral $I(f, T^*)$ for any specified subinterval T^* of T. Furthermore, assume that the rule R_2 is "more accurate" than the rule R_1. By "more accurate" we mean one of the following:

- The DOP of the rule R_2 is greater than the DOP of the rule R_1. Generally this implies that the error for R_2 is expected to be smaller than the error for R_1 for most integrands on most intervals of integration.
- The rule R_2 is the same as R_1 but it is applied on the two halves of the interval. So, if the DOP of R_1 is p the expected error in R_2 on half the interval is a factor of $\left(\frac{1}{2}\right)^{p+2}$ smaller than for R_1. Since we expect approximately the same error on each half interval, the error on both half intervals combined will be a factor of $2\left(\frac{1}{2}\right)^{p+2}$ smaller than for R_1. For example,

if we use Simpson's rule $p = 3$ so the error in Simpson's rule at half the step is expected to be a factor of $2\frac{1}{32} = \frac{1}{16}$ smaller than for the basic rule, and only two extra integrand evaluations are needed to evaluate it (the other three are involved from the basic rule).

Traditionally, the error in using the rule $R_1(f, T^*)$ to approximate the integral $I(f, T^*)$ is estimated by

$$E(f, T^*) \equiv |I(f, T^*) - R_1(f, T^*)| \approx |R_2(f, T^*) - R_1(f, T^*)|$$

where latter approximation is usually justified by arguing that the rule $R_2(f, T^*)$ is a very accurate approximation of the integral $I(f, T^*)$. We approximate the integral $\int_a^b f(x)\,dx$ to a specified accuracy tolerance tol; that is, we want $E(f) \leq tol$ where $E(f)$ is our estimate of the global error in the integration.

At any stage of the algorithm the original interval T has been divided into a number of subintervals $T_i, i = 1, 2, \cdots, n$. Suppose that the rules $R_1(f, T_i)$ computed so far have been accumulated in $R(f)$ and the corresponding accumulated error estimates are held in $E(f)$. In Figure 5.4, we show the case $n = 5$. Assume that $E(f) > tol$; if $E(f) < tol$ the algorithm will terminate. For

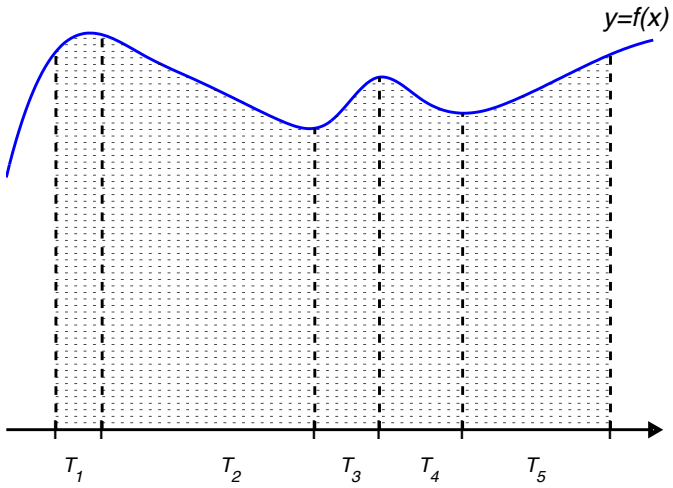

Figure 5.4: One step of a global adaptive integration strategy.

the sake of argument, we suppose that the error estimate with largest magnitude (out of the set of error estimates E_1, E_2, E_3, E_4 and E_5) is E_2 corresponding to the subinterval T_2. Then, we bisect the interval T_2 into two equal subintervals denoted T_{2L} (for Left) and T_{2R} (for Right). Next, we estimate the integral and the magnitude of the error on these subintervals. Finally, we modify the estimates of the overall definite integral and the error as appropriate. Thus, before the bisection of interval T_2 we have

$$\begin{aligned} R(f) &= R_1 + R_2 + R_3 + R_4 + R_5 \\ E(f) &= E_1 + E_2 + E_3 + E_4 + E_5 \end{aligned}$$

and after the bisection of T_2 we have (remember, := is the programming language assignment operation, not mathematical equality)

$$\begin{aligned} R(f) &:= R(f) - R_2 + (R_{2L} + R_{2R}) &= R_1 + R_{2L} + R_{2R} + R_3 + R_4 + R_5 \\ E(f) &:= E(f) - E_2 + (E_{2L} + E_{2R}) &= E_1 + E_{2L} + E_{2R} + E_3 + E_4 + E_5 \end{aligned}$$

Next, we must check whether $E(f) \leq tol$. If it is, we are done; if it is not, we again search for the subinterval T_i of T with the largest error estimate E_i in magnitude and proceed as above. In Figure 5.5, we present an implementation of the approach described above. For simplicity, in this

Global Adaptive Integration

Input: integrand, $f(x)$
endpoints, a, b
accuracy tolerance, tol

Output: integral approximation, $R(f) \approx \int_a^b f(x)\,dx$
error estimate, $E(f)$

$T(1) := T = [a, b]; last := 1$
Compute $R1(f, T(1)), R2(f, T(1))$
$R(f) = R(f, T(1)) := R1(f, T(1))$
$E(f) = E(f, T(1)) := |R2(f, T(1)) - R1(f, T(1))|$
while $E(f) > tol$ do
  Find the index i of the largest error estimate $E(f, T(i))$
  $Rold := R(f, T(i)); Eold := E(f, T(i))$
  $last := last + 1$
  $T(last) :=$ the first half of the interval $T(i)$
  $T(i) :=$ the second half of the interval $T(i)$
  Compute $R(f, T(i)), R(f, T(last)), E(f, T(i)), E(f, T(last))$
  $R(f) := R(f) + (R(f, T(i)) + R(f, T(last)) - Rold)$
  $E(f) := E(f) + (E(f, T(i)) + E(f, T(last)) - Eold)$
end while

Figure 5.5: Pseudocode *Global Adaptive Integration*.

implementation the subintervals T_{2L} and T_{2R} in the example above would be called $T(2)$ and $T(last)$, respectively.

In the algorithm in Figure 5.5 we have made a small change from the description given above. We have arranged to make the subtraction of the values from the discarded intervals and the addition of the values from the newly created intervals together so that at each iteration we are adding a (potentially small) correction to the current approximation to the integral and the associated error. This way we avoid possible catastrophic cancelation when computing the integral and error.

The global adaptive integration algorithm in Figure 5.5 terminates for all integrands $f(x)$ that are sufficiently smooth. When the algorithm terminates, $R(f)$ approximates the integral $I(f)$ and $E(f)$ estimates the error. The estimate, but not the actual error, is guaranteed to be smaller than the accuracy tolerance tol. Here, tol is assumed to be greater than ϵ_{DP}; if this is not the case then the accuracy requirement cannot be met. To choose rules R_1 and R_2 valid for any interval T^* that might arise in the adaptive integration algorithm, first we choose rules R_1 and R_2 appropriate for a canonical interval, then we transform the rules from the canonical interval to the interval T^*.

Problem 5.2.23. *Show that the composite midpoint rule $R_{CM}(f, h)$ is a Riemann sum.*

Problem 5.2.24. *In the example above, argue why the new value of the error estimate $E(f)$ is normally expected to be smaller than the previous value of $E(f)$? [Hint: Argue that, if the DOP's of the integration rules R_1 and R_2 are both greater than one, then $E_{2L} + E_{2R} < E_2$ usually.]*

Problem 5.2.25. *Under the assumptions of Problem 5.2.24, argue that the globally adaptive integration algorithm outlined in this section will normally terminate.*

5.2.4 Gauss and Lobatto Rules

High order integration rules used widely in adaptive quadrature are mainly of Gaussian type. Here, we describe two such classes of rules used in various software packages. First, we discuss the Gauss rules which are open, and their extension to the (open) Gauss-Kronrod rules used for estimating the error. Then, we discuss the Lobatto rules which are closed, that is they include the endpoints of the range of integration.

Gauss Rules

The most popular choices for rules for adaptive integration are the Gauss rules, for which the canonical interval is $[-1,+1]$; here, $I(f) \equiv \int_{-1}^{+1} f(x)\,dx \approx \sum_{i=0}^{N} w_i f(x_i) \equiv R(f)$ where all of the weights w_i, $i = 0, 1, \cdots, N$, and all of the points x_i, $i = 0, 1, \cdots, N$, are chosen to maximize the DOP. In Problems 5.2.26 — 5.2.28, you are asked to find the weights and points in some simple Gauss rules. The equations for the weights and points are nonlinear though not difficult to solve. In practice, Gauss rules with much larger numbers of points than in these problems are generally used. Deriving these practical rules by maximizing the DOP and solving the resulting large systems of nonlinear equations would be difficult, and tedious. Fortunately there are systematic approaches to deriving these rules based on a more sophisticated mathematical theory and the values of the weights and points have been tabulated for values N as large as are ever likely to be needed in practice.

Let us now consider the properties of the Gauss rules. For each value of $N \geq 0$, for the $(N+1)$–point Gauss rule we have:

(1) All the weights $w_i > 0$.

(2) All the points $x_i \in (-1,+1)$.

(3) **Symmetry** – The points x_i are placed symmetrically around the origin and the weights w_i are correspondingly symmetric. For N odd, the points satisfy $x_0 = -x_N, x_1 = -x_{N-1}$ etc. and the weights satisfy $w_0 = w_N, w_1 = w_{N-1}$ etc. For N even, the points and weights satisfy the same relations as for N odd, plus $x_{N/2} = 0$.

(4) The points $\overline{x}_i$ of the N–point Gauss rule interlace the points x_i of the $(N+1)$–point Gauss rule: $-1 < x_0 < \overline{x}_0 < x_1 < \overline{x}_1 < x_2 < \cdots < x_{N-1} < \overline{x}_{N-1} < x_N < +1$.

(5) The Gauss rules are interpolatory quadrature rules; that is, after the points $x_0, x_1, \cdots, x_N$ have been determined, then the weights $w_0, w_1, \cdots, w_N$ may be computed by integrating over the interval $[-1,+1]$ the polynomial of degree N, $q_N(x)$, that interpolates the integrand $f(x)$ at the points $x_0, x_1, \cdots, x_N$.

(6) The DOP of the $(N+1)$–point Gauss rule is $2N+1$.

If the points $x_0, x_1, \cdots, x_N$ had been fixed arbitrarily, then, by analogy with those rules we have derived previously, with $N+1$ free weights we would expect DOP $= N$. But for the Gauss quadrature rules the points are chosen to increase the DOP to $2N+1$. (This shouldn't be a surprise: the Gauss quadrature rule has a total of $2(N+1)$ unknowns, taking the weights and the points together, and it is plausible that they can be chosen to solve all the $2(N+1)$ equations of precision $R(x^k) = I(x^k)$ for $k = 0, 1, \cdots, 2N+1$.)

So, this describes how to choose the rule R_1. Typically, values of N in the range 4 to 30 are used in real-life applications. Given that the rule R_1 has been chosen to be a $(N+1)$–point Gauss rule with DOP $= 2N+1$, a sensible choice for the rule R_2 is the $(N+2)$–point Gauss rule which has DOP $= 2(N+2)+1 = (2N+1)+2$; that is, its DOP is 2 higher than for the corresponding Gauss $(N+1)$–point rule. Also, the interlacing property is important because it guarantees a good "sampling of the integration interval" by the points that the quadrature rule and the error estimate together provide.

But, for the same expense as evaluating the integrand $f(x)$ at an additional $N+2$ points to compute the Gauss $(N+2)$–point rule, we can achieve DOP $= 3N+2$ in R_2 by using a Gauss–Kronrod (GK) rule. Without going into details, this GK rule reuses the integrand evaluations from the Gauss $(N+1)$–point rule but with a new set of weights and adds $N+2$ new points and related weights. Then, the $N+2$ unknown points and the $(N+1)+(N+2) = 2N+3$ weights are used to maximize the DOP of the GK rule. The properties of the GK rules are very similar to those of the Gauss rules, including: all the points lie in the integration interval $(-1,+1)$, all the weights are positive, and there is an interlacing property of the points of the $(N+1)$–point Gauss rule with the $N+2$ new points of the corresponding $(2N+3)$–point GK rule.

Problem 5.2.26. *You are to derive Gauss rules by solving the equations of precision*

(1) Show that the Gauss one–point rule is the midpoint rule.

(2) By the symmetry properties the Gauss two–point rule has the simplified form:

$$\int_{-1}^{1} f(x)\,dx \approx w_0 f(x_0) + w_0 f(-x_0)$$

Find w_0 and x_0 to maximize the DOP. Find the Peano constant κ.

Problem 5.2.27. *By the symmetry properties the Gauss three–point rule has the simplified form:*

$$\int_{-1}^{1} f(x)\,dx \approx w_0 f(x_0) + w_1 f(0) + w_0 f(-x_0)$$

Compute the values w_0, w_1 and x_0 to maximize the DOP. Find the Peano constant κ.

Problem 5.2.28. *What is the simplified form of the Gauss four–point rule implied by the symmetry properties? How many unknowns are there? Find the unknowns by maximizing the DOP.*

Problem 5.2.29. *Why do we want all the points in the Gauss rules to lie in the interval $(-1,+1)$? Why do we want all the weights in the Gauss rules to be positive? [Hint: Since DOP > 0, we must satisfy the first equation of precision $\sum_{i=0}^{N} w_i = 2$.]*

Problem 5.2.30. *The Gauss one–point rule is the midpoint rule. Show that the corresponding Gauss–Kronrod three-point rule is the same as the Gauss three–point rule.*

Lobatto Rules

Another popular choices of rules for adaptive integration are the Lobatto rules, for which the canonical interval is again $[-1,+1]$. These rules are close relatives of the Gauss rules. The (N+1)-point rule has the form $I(f) \equiv \int_{-1}^{+1} f(x)\,dx \approx w_0 f(-1) + \sum_{i=1}^{N-1} w_i f(x_i) + w_N f(1) \equiv R(f)$ where the all weights w_i, $i = 0, 1, \cdots, N$, and all the unknown points x_i, $i = 1, 2, \cdots, N-1$, are chosen to maximize the DOP. The points ± 1 are included since the *endpoints* of the interval of integration are always used in Lobatto integration. In Problems 5.2.31— 5.2.33, you are asked to find the weights and points in some simple Lobatto rules. In reality, Lobatto rules with larger numbers of points than in these problems are often used.

Let us now consider the properties of the Lobatto rules. For each value of $N \geq 0$, for the $(N+1)$–point Lobatto rule we have:

(1) All the weights $w_i > 0$.

(2) The points $x_i \in (-1,+1), 1 = 1, 2, \ldots, N-1$.

(3) **Symmetry** – The points x_i are placed symmetrically around the origin and the weights w_i are correspondingly symmetric. For N odd, the points satisfy $x_1 = -x_{N-1}, x_2 = -x_{N-2}$ etc. and the weights satisfy $w_0 = w_N, w_1 = w_{N-1}, w_2 = w_{N-2}$ etc. For N even, the points and weights satisfy the same relations as for N odd, plus $x_{N/2} = 0$.

(4) The points $\overline{x}_i$ of the N–point Lobatto rule interlace the points x_i of the $(N+1)$–point Lobatto rule: $-1 < x_1 < \overline{x}_1 < x_2 < \overline{x}_2 < x_3 < \cdots < x_{N-2} < \overline{x}_{N-2} < x_{N-1} < +1$. Note that the points ± 1 are points for all the Lobatto rules.

(5) The Lobatto rules are interpolatory quadrature rules; that is, after the points $x_1, x_2, \cdots, x_{N-1}$ have been determined, then the weights $w_0, w_1, \cdots, w_N$ may be computed by integrating over the interval $[-1, +1]$ the polynomial of degree N, $q_N(x)$, that interpolates the integrand $f(x)$ at the points $-1, x_1, x_2 \cdots, x_{N-1}, 1$.

(6) The DOP of the $(N+1)$–point Lobatto rule is $2N-1$.

If the points $x_1, x_2, \cdots, x_{N-1}$ had been fixed arbitrarily, then, by analogy with those rules we have derived previously, with $N+1$ free weights we would expect DOP$= N$, or in some (symmetric) cases DOP$=N+1$. But in the Lobatto quadrature rules the points are chosen to increase the DOP to $2N-1$. (This shouldn't be a surprise: the Lobatto $(N+1)$–point quadrature rule has a total of $2N$ unknowns, taking the weights and the points together, and it is plausible that they can be chosen to solve all the $2N$ equations of precision $R(x^k) = I(x^k)$ for $k = 0, 1, \cdots, 2N-1$.)

So, if we are using Lobatto rules for adaptive integration this describes how to choose the rule R_1. Typically, values of N in the range 4 to 10 are used in real-life applications. For the rule R_2, given that the rule R_1 has been chosen to be an $(N+1)$–point Lobatto rule with DOP $= 2N-1$, a sensible choice is the $(N+2)$–point Lobatto rule which has DOP $= 2(N+1) - 1 = (2N-1) + 2$; that is, its DOP is 2 higher than for the corresponding Lobatto $(N+1)$–point rule. The resulting interlacing property is important because it guarantees a good "sampling of the integration interval" by the points that the quadrature rule and the error estimate together provide. Also, there are two endpoints in common, so the total number of points to compute the integral and error estimate is $(N+1) + (N+2) - 2 = 2N+1$. In addition, it is possible to organize the computation so that integrand evaluations at the ends of subintervals are shared across intervals hence effectively reducing the number of evaluations for the integral and error estimate to $2N$ per interval.

Problem 5.2.31. *You are to derive Lobatto rules by solving the equations of precision*

(1) Show that the Lobatto three–point rule is Simpson's rule.

(2) By the symmetry properties the Lobatto four–point rule has the simplified form:

$$\int_{-1}^{1} f(x)\,dx \approx w_0 f(-1) + w_1 f(-x_1) + w_1 f(x_1) + w_0 f(1)$$

Find w_0, w_1 and x_1 to maximize the DOP. Find the Peano constant κ for each rule.

Problem 5.2.32. *By the symmetry properties the Lobatto five–point rule has the simplified form:*

$$\int_{-1}^{1} f(x)\,dx \approx w_0 f((-1) + w_1 f(-x_1) + w_2 f(0) + w_1 f(x_1) + w_0 f(1)$$

Compute the values w_0, w_1, w_2 and x_1 to maximize the DOP. Find the Peano constant κ for this rule.

Problem 5.2.33. *What is the simplified form of the Lobatto six–point rule implied by the symmetry properties? How many unknowns are there? Find the unknowns by maximizing the DOP.*

Problem 5.2.34. *Why do we want all the points in the Lobatto rules to lie in the interval $(-1, +1)$? Why do we want all the weights in the Lobatto rules to be positive? [Hint: Since DOP > 0, we must satisfy the first equation of precision $\sum_{i=0}^{N} w_i = 2$.]*

Problem 5.2.35. *Explain how you know (without deriving the formula) that the Lobatto four– point rule is not Weddle's (four–point) closed Newton-Cotes rule.*

Comparing Gauss and Lobatto rules

In terms of degree of precision the Gauss N–point rule is comparable to the Lobatto $(N+1)$–point rule; the DOP $= 2N + 1$ in each case. If we consider the number of points where we must evaluate the integrand (for an amount of accuracy delivered) as the measure of efficiency then it seems that the Gauss rules are slightly the more efficient.

However, in the context of adaptive quadrature we observe that the point corresponding to $x = 1$ in one interval is the same as the point corresponding to $x = -1$ in the next. So, assuming we keep the values of the integrand at the interval endpoints after we have evaluated them and reuse them where appropriate then the cost of evaluating a Lobatto rule is reduced by about one integrand evaluation giving about the same cost as the Gauss rule with the same DOP. A further saving for the Lobatto rules arises in computing the error estimates as the integrand evaluations at the interval endpoints have already been calculated and if we reuse them in the Lobatto $(N+2)$–point rule the cost of the Lobatto error estimate is one less integrand evaluation than the for the corresponding Gauss rule. So, overall an efficient computation with the Lobatto rules seems likely to be at least as efficient as one with the Gauss rules.

An alternative approach to estimating the error in the Lobatto rules is to use a Lobatto-Kronrod rule. So, for a Lobatto $(N+1)$–point rule we construct a Lobatto-Kronrod $(2N+1)$–point rule reusing all the points in the Lobatto rule and adding N interior points symmetrically interlacing the interior points of the Lobatto rule. The cost of this is the same as using the Lobatto $(N+2)$–point rule but the DOP is higher.

There are other factors involved:

- We have discussed using Gauss and Lobatto rules of the same degree of precision. This implies that the rules behave similarly as the step size tends to zero but the actual error depends also on the size of the Peano constant. For example, the Gauss two-point rule and Simpson's rule (the Lobatto three–point rule) have the same degree of precision but the Peano constant for the Gauss two–point rule is $1/135$ and that for Simpson's rule (the Lobatto three–point rule) is $1/90$ hence the Gauss two–point rule is about 50% more accurate on the same interval. The sizes of the Peano constants for other Gauss and Lobatto rules of the same DOP similarly favor the Gauss rules.

- Suppose we want to compute the integral

$$\int_0^1 \frac{\cos x}{\sqrt{x}}\, dx$$

  which exists even though there is a singular point at $x = 0$. If we use a Gauss rule in our adaptive quadrature scheme it will never need the integrand value at $x = 0$, and will, with some effort, compute an accurate value. It takes some effort because the adaptive scheme will subdivide the interval a number of times near $x = 0$ before it computes the answer sufficiently accurately to meet the user's error tolerance. In contrast, using a Lobatto rule in the adaptive algorithm will fail immediately.

There are ways round the latter difficulty which may be superior even for the Gauss rules. One, relatively simple, approach is to expand the integrand in a series about the singularity then integrate the series over a short interval near the singularity and the original integrand by your

favorite adaptive scheme elsewhere. For example, in the case of $\int_0^1 \frac{\cos x}{\sqrt{x}}\,dx$, we could expand $\cos x$ in Taylor series about $x = 0$, then split the interval as follows:

$$\int_0^1 \frac{\cos x}{\sqrt{x}}\,dx = \int_0^\delta \frac{\cos x}{\sqrt{x}}\,dx + \int_\delta^1 \frac{\cos x}{\sqrt{x}}\,dx \approx \int_0^\delta \frac{1 - x^2/2}{\sqrt{x}}\,dx + \int_\delta^1 \frac{\cos x}{\sqrt{x}}\,dx$$

then choose δ sufficiently small so the series is accurate enough but not so small that the second integral is too irregular near the point $x = \delta$. Then the first integral may be calculated using simple Calculus and the second by your favorite adaptive scheme.

Problem 5.2.36. *Compare the accuracies of the following pairs of rules*

- *The Gauss three-point rule and the Lobatto four point rule*
- *The Gauss four–point rule and the Lobatto five–point rule*

Problem 5.2.37. *How accurate is the expression $(1-x^2/2)$ as an approximation to $\cos x$ on the interval $[0, \delta]$? How accurate would be the Taylor series with one more non-zero term? Calculate the approximation $\int_0^\delta \frac{1-x^2/2}{\sqrt{x}}\,dx$ and the corresponding approximation using one more non-zero term in the Taylor series. How small must δ be for these approximations to agree to four significant digits? [Hint: The Taylor series for $\cos x$ is an alternating series.*

5.2.5 Transformation from a Canonical Interval

When describing adaptive integration it was assumed that we have available quadrature rules for any interval of integration. However, almost all our derivations and discussions of quadrature rules have been for canonical intervals such as $[-1, 1]$ and $[-h, h]$. So, we consider how to transform a quadrature rule for a canonical interval of integration, chosen here as $[-1, 1]$, to a quadrature rule on a general interval of integration $[a, b]$.

The standard change of variable formula of integral calculus using the transformation $x = g(t)$ is

$$\int_{g(c)}^{g(d)} f(x)\,dx = \int_c^d f(g(t))g'(t)\,dt$$

One possible approach is to choose the transformation $g(t)$ as the straight line interpolating the points $(-1, a)$ and $(+1, b)$; that is, using the Lagrange form of the interpolating polynomial,

$$g(t) = a\frac{t - (+1)}{(-1) - (+1)} + b\frac{t - (-1)}{(+1) - (-1)} = \frac{(b-a)}{2}t + \frac{(b+a)}{2}\,.$$

The transformation $g(t)$ maps the canonical interval onto the interval of interest.

For this transformation $g(t)$ we have $g'(t) = \dfrac{(b-a)}{2}$ and the change of variable formula reads

$$\int_a^b f(x)\,dx = \frac{(b-a)}{2}\int_{-1}^{+1} f\left(\frac{(b-a)}{2}t + \frac{(b+a)}{2}\right)dt$$

Now, assume that for the canonical interval $[-1, 1]$ we have a quadrature rule

$$I^*(h) \equiv \int_{-1}^{+1} h(t)\,dt \approx \sum_{i=0}^{N} w_i^* h(t_i^*) \equiv R^*(h)$$

then, substituting for $h(t) = f(g(t))$, we have

$$\begin{aligned} I(f) \equiv \int_a^b f(x)\,dx &= \frac{(b-a)}{2}\int_{-1}^{+1} f\left(\frac{(b-a)}{2}t + \frac{(b+a)}{2}\right)\,dt \\ &= \frac{(b-a)}{2} I^*(f\circ g) \approx \frac{(b-a)}{2} R^*(f\circ g) \\ &= \frac{(b-a)}{2}\sum_{i=0}^{N} w_i^* f\left(\frac{(b-a)}{2}t_i^* + \frac{(b+a)}{2}\right) = \sum_{i=0}^{N} w_i f(x_i) \equiv R(f) \end{aligned}$$

where in $R(f)$ the weights $w_i = \dfrac{(b-a)}{2}w_i^*$ and the points $x_i = \dfrac{(b-a)}{2}t_i^* + \dfrac{(b+a)}{2}$.

The DOP of the transformed quadrature rule is the same as the DOP of the canonical quadrature rule. The error term for the canonical quadrature rule may be transformed using $g(t)$ to obtain the error term for the transformed quadrature rule. However, this error term can more easily be determined by applying Peano's theorem directly.

Problem 5.2.38. *Here we see how to use the transformation presented in this section to use rules derived on a canonical interval for integration on a general interval*

(1) Transform the Gauss 2–point quadrature rule given for the canonical interval $[-1,+1]$ to the interval $[a,b]$. Show that the DOP of the transformed quadrature rule is 3 and determine Peano's constant.

(2) Repeat (1) for the Gauss 3–point quadrature rule.

Problem 5.2.39. *Transform the Gauss 2–point quadrature rule for the canonical interval $[-h,+h]$ to the interval $[a,a+h]$. Show that the DOP of the transformed quadrature rule is 3.*

5.3 Matlab Notes

MATLAB has several functions that can be used for differentiation and integration, both symbolically and numerically. Some functions that are relevant to the topics discussed in this chapter include:

`syms`	used to declare a variable is symbolic
`diff`	used to evaluate derivative of function defined by symbolic variables, or to calculate differences between adjacent elements in vector of floating point numbers
`gradient`	used to approximate derivative of a function
`int`	used to evaluate integral of function defined by symbolic variables
`integral`	used to approximate integral, using low order global adaptive quadrature
`quadl`	used to approximate integral, using adaptive Lobatto quadrature
`quadgk`	used to approximate integral, using adaptive Gauss-Kronrod quadrature
`trapz`	used to approximate integral from tabular data

5.3.1 Differentiation

Symbolic Differentiation

When we need to find the derivative $f'(x)$ of a given function $f(x)$ then usually our starting point should be the MATLAB symbolic algebra package which is a part of Student MATLAB. (The package is derived from the Maple symbolic package.)

MATLAB "knows" the derivative of all the mathematical functions that it can represent and evaluate. These functions include the trigonometric, exponential, logarithmic, and hyperbolic functions and their inverses, and many other less familiar functions. It uses the "rules" of differentiation (product, quotient, chain rule etc.) to compute the derivative of combinations of functions.

For example, the sequence of MATLAB instructions

```
syms z
f = atan(3*z);
diff(f)
```

produces the result

```
ans =

3/(1+9*z^2)
```

which you can verify is correct. Observe that you need to indicate to MATLAB that you are using symbolic variables, by specifying them in the `syms` line. Without this specification MATLAB would not know which of its two `diff` functions to apply.

The more complicated sequence

```
syms z
f = sin(4*cos(3*z))/exp(5*(1+z^2));
diff(f)
```

produces the (correct) result

```
ans =

-12*cos(4*cos(3*z))*sin(3*z)/exp(5+5*z^2)-10*sin(4*cos(3*z))/exp(5+5*z^2)*z
```

MATLAB has the ability to "simplify" the result, which is taken to mean to represent the answer using as short a string of symbols as possible. So, the MATLAB statements

```
syms z
f = sin(4*cos(3*z))/exp(5*(1+z^2));
simplify(diff(f))
```

computes the "simplified" result

```
ans =

-2*(6*cos(4*cos(3*z))*sin(3*z)+5*sin(4*cos(3*z))*z)*exp(-5-5*z^2)
```

Though this simplification seems trivial, MATLAB went through a long sequence of steps and tried a variety of "tricks" to arrive at it. To see what MATLAB tried replace `simplify` by `simple`.

In cases where the "function" that we wish to differentiate is represented by a program (as is often the case in real world applications), it has become standard practice recently to use *automatic differentiation*. This process is not available directly in MATLAB though there are several packages that implement versions of it. These packages work by parsing the program then using the chain and other differentiation rules automatically to produce either numerical values of a derivative or a program to evaluate the derivative.

Numerical Differentiation

In those cases where we need to resort to numerical differentiation we can use the MATLAB function `gradient` which is designed for approximating the gradient of a multivariate function from a table of values, but can be used in our one dimensional case.

Example 5.3.1. Evaluating $\sin(x)$ on the interval $[0, \frac{\pi}{2}]$ using an equispaced mesh with mesh size $h = \pi/20$ and calculating the derivative (gradient) and the error at each mesh point, we obtain the results in Table 5.2 using the MATLAB code

```
N = 10;
h = pi/(2 * N);
x = 0 : h: pi/2;
f = sin(x);
fx = gradient(f, h);
err = fx - cos(x);
for i = 1 : N + 1
    disp(sprintf('%d %d %d', x(i), fx(i), err(i)))
end
```

Observe the lower accuracy at the right end point due to using first order differences there as against second order differences at internal points. (This problem persists for smaller step sizes.) The second argument in `gradient` is the step size. The step size may be replaced by a vector of locations for an unequally spaced mesh.

x	gradient	error
0	9.958927e-001	-4.107265e-003
1.570796e-001	9.836316e-001	-4.056698e-003
3.141593e-001	9.471503e-001	-3.906241e-003
4.712389e-001	8.873469e-001	-3.659600e-003
6.283185e-001	8.056941e-001	-3.322847e-003
7.853982e-001	7.042025e-001	-2.904275e-003
9.424778e-001	5.853711e-001	-2.414190e-003
1.099557e+000	4.521258e-001	-1.864659e-003
1.256637e+000	3.077478e-001	-1.269215e-003
1.413717e+000	1.557919e-001	-6.425178e-004
1.570796e+000	7.837846e-002	7.837846e-002

Table 5.2: Using `gradient` to approximate the derivative of $\sin x$

Alternatively, we can fit with a cubic spline and differentiate the resulting function. This requires us to use the function `unmkpp` to determine the coefficients of the cubic polynomial pieces. To be specific, given a set of data points, (x_i, f_i), the following MATLAB commands compute the spline and its coefficients:

```
S = spline(x, f);
[x, a, N, k] = unmkpp(S);
```

Here, `N` is the number of polynomial pieces, and `a` is an `N`×`k` array whose ith row contains the coefficients of the ith polynomial. That is, if

$$S_{3,N}(x) = \begin{cases} a_{11}(x-x_0)^3 + a_{12}(x-x_0)^2 + a_{13}(x-x_0) + a_{14} & x_0 \le x \le x_1 \\ a_{21}(x-x_1)^3 + a_{22}(x-x_1)^2 + a_{23}(x-x_1) + a_{24} & x_1 \le x \le x_2 \\ \vdots & \vdots \\ a_{N1}(x-x_{N-1})^3 + a_{N2}(x-x_{N-1})^2 + a_{N3}(x-x_{N-1}) + a_{N4} & x_{N-1} \le x \le x_N \end{cases}$$

then `unmkpp` returns the array

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & a_{N4} \end{bmatrix}.$$

Example 5.3.2. Evaluating $\sin(x)$ on the interval $[0, \frac{\pi}{2}]$ using an equispaced mesh with mesh size $h = \pi/20$ and calculating the derivative and the error at each mesh point using the MATLAB function `spline`, we obtain the results in Table 5.3 using the MATLAB code

```
N = 10;
h = pi/(2 * N);
x = 0 : h : pi/2;
f = sin(x);
S = spline(x, f);
[x, a] = unmkpp(S);
for i = 1 : N
    der(i) = a(i, 3);
end
der(N + 1) = 3*a(N, 1)* h^2 + 2*a(N, 2)*h + a(N, 3);
err = der - cos(x);
for i = 1 : N + 1
    disp(sprintf('%d %d %d', x(i), der(i), err(i)))
end
```

Note the special treatment of the last point which is only at the *end* of an interval. Observe is Table 5.3 that the errors using `spline` are far smaller than using `gradient` on the same mesh. Also note that the errors near the end of the interval are larger than in the interior, even though they are of the same order of accuracy.

x	spline derivative	error
0	1.000105e+000	1.054555e-004
1.570796e-001	9.876558e-001	-3.251156e-005
3.141593e-001	9.510611e-001	4.570561e-006
4.712389e-001	8.910015e-001	-5.048342e-006
6.283185e-001	8.090146e-001	-2.437651e-006
7.853982e-001	7.071052e-001	-1.599607e-006
9.424778e-001	5.877798e-001	-5.496783e-006
1.099557e+000	4.540022e-001	1.167249e-005
1.256637e+000	3.089666e-001	-5.039545e-005
1.413717e+000	1.566181e-001	1.836456e-004
1.570796e+000	-6.873579e-004	-6.873579e-004

Table 5.3: Using `spline` to approximate the derivative of $\sin x$

Problem 5.3.1. *Tabulate the function* $\ln x$ *at integer values on the interval* $[1, 10]$ *From your tabulated values approximate the derivative of* $\ln x$ *at the points* $1, 2 \ldots, 10$ *using the* MATLAB *function* `gradient`*. Tabulate the approximate derivatives and their errors. Also, calculate the approximate derivative by fitting to the data with the* MATLAB *function* `spline` *and differentiating the result. Again, tabulate the approximate derivatives and their errors. Which approach delivers the more accurate approximations to the derivative?*

5.3.2 Integration

Symbolic Integration

Consider first computing antiderivatives of integrable functions. In Calculus we learned first how to compute antiderivatives of a number of simple functions, for example powers, trigonometric and exponential functions then we learned a number of techniques to extend the range of functions for which we could compute antiderivatives. These techniques included substitution, integration by parts and partial fractions. In each of these cases our approach was to reduce the computation of an antiderivative to looking up a linear combination of well-known antiderivatives. This is essentially the technique used by MATLAB's symbolic integrator `int`. Given a function, it attempts to transform the function into a linear combination of functions each of which is in MATLAB's lookup table of antiderivatives. The difference between your attempts to find antiderivatives and MATLAB's is that MATLAB's `int` knows many more transformations and has a much longer lookup table than do you.

Consider the following MATLAB code for computing the antiderivatives of four integrands.

```
syms z
int(sin(z) * cos(z))
int(sin(4*z) * cos(3*z))
int(sin(z)^2 * cos(z)^3)
int((1/(1-z)) * (1/(1 + z + z^2)))
int((sin(z)/(1 - z)) * (1/(1 + z + z^2)))
int((sin(z)/(1 - z)) * (1/(1 + cos(z^2) + z^2)))
```

This code produces the following somewhat edited output.

```
ans =
1/2*sin(z)^2

ans =
-1/14*cos(7*z)-1/2*cos(z)

ans =
-1/5*sin(z)*cos(z)^4+1/15*cos(z)^2*sin(z)+2/15*sin(z)

ans =
-1/3*log(z-1)+1/6*log(z^2+z+1)+1/3*3^(1/2)*atan(1/3*(2*z+1)*3^(1/2))

ans =
-1/3*sinint(z-1)*cos(1)-1/3*cosint(z-1)*sin(1)-1/9*i*(3/2+1/2*i*3^(1/2))*3^(1/2)*
(sinint(z+1/2-1/2*i*3^(1/2))*cos(1/2-1/2*i*3^(1/2))-cosint(z+1/2-1/2*i*3^(1/2))*
sin(1/2-1/2*i*3^(1/2)))+1/9*i*(3/2-1/2*i*3^(1/2))*3^(1/2)*(sinint(z+1/2+1/2*i*3^(1/2))*
cos(1/2+1/2*i*3^(1/2))-cosint(z+1/2+1/2*i*3^(1/2))*sin(1/2+1/2*i*3^(1/2)))

Warning: Explicit integral could not be found.
> In sym.int at 58
  In symbint at 7
ans =
int(sin(z)/(1-z)/(1+cos(z^2)+z^2),z)
```

the first antidifferentiation used integration by parts followed by a trigonometric identity, the second used trigonometric identities followed by substitution and lookup, the third used trigonometric identities followed by integration by parts, and the fourth used partial fractions. Observe that the difference between the fourth and fifth integrands is the introduction of a `sin(z)` in the numerator but the effect on the result is dramatic. You will observe that the result is complex and involves two MATLAB functions `sinint` and `cosint`. These stand for the Sine Integral and the Cosine Integral, respectively, and they are "well-known" functions that are in MATLAB's lookup tables and whose

values MATLAB can evaluate. The sixth integrand is a modification of the fifth, replacing a z by a $\cos(z^2)$. This is a function for which MATLAB cannot find the antiderivative, a warning is given and the line of code is repeated.

Consider next the definite integral. We take the set of integrands above and integrate the first three over $[0, 1]$ and the last three over $[2, 3]$ (all these integrals exist). The code is

```
syms z
int(sin(z) * cos(z), 0, 1)
int(sin(4*z) * cos(3*z), 0, 1)
int(sin(z)^2 * cos(z)^3, 0, 1)
int((1/(1 - z)) * (1/(1 + z + z^2)), 2, 3)
int((sin(z)/(1 - z)) * (1/(1 + z + z^2)), 2, 3)
int((sin(z)/(1 - z)) * (1/(1 + cos(z^2) + z^2)), 2, 3)
```

and the (edited) results are

```
ans =
1/2*sin(1)^2

ans =
-1/14*cos(7)-1/2*cos(1)+4/7

ans =
-1/5*sin(1)*cos(1)^4+1/15*cos(1)^2*sin(1)+2/15*sin(1)

ans =
-1/3*log(2)+1/6*log(13)+1/3*3^(1/2)*atan(7/3*3^(1/2))-1/6*log(7)-1/3*3^(1/2)*atan(5/3*3^(1/2))

ans =
The value of this integral has been omitted by the authors.
The expression takes over twelve lines to reproduce.

Warning: Explicit integral could not be found.
> In sym.int at 58
  In symbint at 13

ans =
int(sin(z)/(1-z)/(1+cos(z^2)+z^2),z = 2, 3)
```

Of course, often you want a numerical value when you evaluate a definite integral. For the above integrals the values can be calculated as follows

```
a = int(sin(z) * cos(z), 0, 1); double(a)
a = int(sin(4*z) * cos(3*z), 0, 1); double(a)
a = int(sin(z)^2 * cos(z)^3, 0, 1); double(a)
a = int((1/(1 - z)) * (1/(1 + z + z^2)), 2, 2); double(a)
a = int((sin(z)/(1 - z)) * (1/(1 + z + z^2)), 2, 3); double(a)
a = int((sin(z)/(1-z)) * (1/(1+cos(z^2) +z^2)), 2, 3); double(a)
```

and the results are

```
ans =
   0.35403670913679

ans =
   0.24742725746998
```

```
ans =
   0.11423042636636

ans =
     0

ans =
 -0.04945332203255 - 0.00000000000000i

Warning: Explicit integral could not be found.
> In sym.int at 58
  In symbint at 19

ans =
  -0.06665303913176
```

The first five of these values were calculated by arithmetically evaluating the expressions above. Note that the result for the fifth integral is real (as it must be for a real function integrated over a real interval) even though the antiderivative is complex (the imaginary parts cancel to give zero). The sixth integral is computed differently. As we have already seen, MATLAB cannot find the antiderivative. It produces a numerical value by resorting to a numerical integration scheme such as those described in the next section. In fact, all these values may be calculated simply by using numerical integration schemes.

Finally, let us consider the improper integrals represented by the following code

```
int(1/z, 0, 1)
int(1/sqrt(z), 0, 1)
int(1/z, -1, 1)
int(1/z, 1, Inf)
int(1/z^2, 1, Inf)
```

where `Inf` is the MATLAB function representing infinity. These give

```
ans =
Inf

ans =
2

ans =
NaN

ans =
Inf

ans =
1
```

The first, third and fourth integrals are not convergent whereas the second and fifth are convergent. It is interesting that MATLAB spots that there is a problem with the third integral – it does more than simply substitute the limits of integration into an antiderivative.

Problem 5.3.2. *Use the* MATLAB *function* `int` *to compute the definite integrals*

(a) $\displaystyle\int_1^2 \frac{\ln x}{1+x}\,dx$ (b) $\displaystyle\int_0^3 \frac{1}{1+t^2+t^4}\,dt$

(c) $\displaystyle\int_0^{1/2} \sin\left(e^{t/2}\right)\,dt$ (d) $\displaystyle\int_0^4 \sqrt{1+\sqrt{x}}\,dx$

Numerical Integration

When attempting to compute approximations of

$$I(f) = \int_a^b f(x)\,dx\,, \tag{5.1}$$

three basic situations can occur:

1. $f(x)$ is a fairly simple, known function.
2. $f(x)$ is a known function, but is complicated by the fact that it contains various parameters that can change depending on the application.
3. $f(x)$ is not known explicitly. Instead, only a table of $(x_i, f(x_i))$ values is known.

In this section we describe how to use MATLAB for each of these situations. We also discuss how to handle certain special situations, such as infinite limits of integration, and integrand singularities.

Explicitly Known, Simple Integrand

Suppose the integrand, $f(x)$ is explicitly known. In this case, MATLAB provides three functions for definite integration, `integral`, `quadgk`, and `quadl`. The functions `integral` and `quadgk` implement adaptive integration using a 7-point Gauss rule with a 15-point Kronrod extension. The function `quadl` implements a lower order method, the Lobatto 4-point rule, with a 7-point Kronrod extension.

Since `integral` and `quadgk` use the same basic approach, the rest of this section will only refer to `integral`. The basic usage of `integral` and `quadl` is

```
I = integral(fun, a, b);
I = quadl(fun, a, b);
```

In order to use these, we must first define the function that is to be integrated. This can be achieved in several ways, including anonymous functions and function M–files. We illustrate how to use each of these with `integral`, for simple integrands, in the following example.

Example 5.3.3. Consider the simple example

$$\int_0^\pi (2\sin(x) - 4\cos(x))\,dx$$

Here are two ways to define $f(x) = 2\sin(x) - 4\cos(x)$, and how they can be used with `integral` to compute an approximate integral.

- Since the integrand is a fairly simple function, it can be easily defined as an anonymous function, with the result and the endpoints of the interval of integration then passed to `integral`:

```
f = @(x) 2*sin(x) - 4*cos(x);
I = integral(f, 0, pi)
```

- A second option is to write a function M–file that evaluates the integrand, such as:

```
function f = MyFun( x )
%
%  This function evaluates f(x) = 2*sin(x) - 4*cos(x)
%
%  Input:  x = vector of x-values at which f(x) is to be evaluated.
%
%  Output: f = vector containing the f(x) values.
%
f = 2*sin(x) - 4*cos(x);
```

Then we can use a "function handle" to tell `integral` about `MyFun`

```
I = integral(@MyFun, 0, pi)
```

When the integrand is a simple function, like the previous example, the anonymous approach is usually easiest to use. For more complicated integrands, function M–files may be necessary. It is important to note that, regardless of the approach used, evaluation of the integrand must be vectorized; that is, it must be able to compute $f(x)$ at a vector of x-values as required by the functions `integral` and `quadl`. MATLAB requires this since using vectorized arithmetic where possible is far more efficient than using scalar arithmetic. In numerical integration we know that we always need to evaluate the integrand at all the points in the integration rule (a vector) so there is no reason not to use vectorization.

Example 5.3.4. Suppose we want to compute an approximation of

$$\int_0^\pi x \sin(x)\, dx$$

Since the integrand in this case is very simple, we define $f(x)$ as an anonymous function. Our first attempt might be to use the statement:

```
f = @(x) x*sin(x);
```

This is a legal MATLAB statement, however we can only use it to compute $f(x)$ for scalar values x. For example, the statements

```
f(0)
f(pi/2)
f(pi)
```

will execute without error, and compute accurate approximations of the true values 0, $\frac{\pi}{2}$, and 0, respectively. However, if we try to execute the statements

```
x = [0 pi/2 pi];
f(x)
```

then an error will occur, stating that "inner matrix dimensions must agree". The same error will occur if we try to use `integral` or `quadl` with this anonymous function because, for reasons of efficiency, the software wants the integrand values at all the points in the rule simultaneously and so the computation `x*sin(x)` must be vectorized. A vectorized implementation is given by:

```
f = @(x) x.*sin(x);
```

In this case, we can compute $f(x)$ for vectors of x-values, and use `integral` and `quadl`, as in the previous example, without error.

Example 5.3.5. Suppose we want to compute an approximation of

$$\int_0^{2\pi} (\cos x)^2 (\sin x)^2 \, dx$$

The following MATLAB statements can be used to approximate this integral:

```
f = @(x) ( (cos(x)).^2 ) .* ( sin(x).^2 );
I = integral(f, 0, 2*pi)
```

Note that we must use vector operations (dot multiply and dot exponentiation) in the definition of the integrand.

Example 5.3.6. Suppose we want to compute an approximation of

$$\int_1^e \frac{1}{x\,(1 + (\ln x)^2)} \, dx$$

The following MATLAB statements can be used to approximate this integral:

```
f = @(x) 1 ./ (x .* (1 + log(x).^2));
I = integral(f, 1, exp(1))
```

Notice again that we must use vector operations (dot divide, dot multiply and dot exponentiation) in the definition of the integrand. Note, the MATLAB `log` function is the natural logarithm, and the `exp` function is the natural exponential. Thus, `exp(1)` computes $e^1 = e$.

As with most MATLAB functions, it is possible to provide more information to, and get more information from `integral` and `quadl`. For example, the basic calling sequence

```
I = integral(f, a, b)
```

uses a default absolute error tolerance of 10^{-6}. We can override this default value by using the calling sequence

```
I = integral(f, a, b, 'AbsTol', tol)
```

where a value for `tol` must be predefined. See the documentation on `integral` for information on other default values that can be modified.

The amount of work required by `integral` or `quadl` is determined by the total number of integrand evaluations that must be calculated. One way to obtain this information is to use a function m-file to evaluate $f(x)$, and update a counter in the m-file that is declared to be a *gobal variable*. Specifically, we can include the following statements at the beginning of the function:

```
global nevals
nevals = nevals + 1;
```

Then, just before using integral, declare `nevals` as a global variable, and initialize it to 0. For example, if we add the above two lines of code to `MyFun`, we can then we can use the following:

```
global nevals
nevals = 0;
I = integral(@MyFun, 0, pi)
```

In this case, `I` contains the approximation of the integral, and `nevals` is the total number of integrand evaluations needed to compute `I`.

Problem 5.3.3. *Use* `integral` *and* `quadl` *to compute approximations to the given integrals.*

(a) $\displaystyle\int_1^2 \frac{\ln x}{1+x}\,dx$ *(b)* $\displaystyle\int_0^3 \frac{1}{1+t^2+t^4}\,dt$

(c) $\displaystyle\int_0^{1/2} \sin\left(e^{t/2}\right)\,dt$ *(d)* $\displaystyle\int_0^4 \sqrt{1+\sqrt{x}}\,dx$

In each case, report on the number of function evaluations required by each method. Check your results against the exact values of these integrals that you computed in Problem 5.3.2.

Explicitly Known, Complicated Integrand

Suppose the integrand, $f(x)$, is known, but that it contains one or more parameters that may change. Such a situation is best illustrated with an example.

A random variable X has a *continuous distribution* if there exists a nonnegative function, f, such that for any interval $[a, b]$,

$$\text{Prob}(a \le X \le b) = \int_a^b f(x)\,dx\,.$$

The function $f(x)$ is called the *probability density function* (PDF). An important PDF is the *normal* (sometimes called a *Gaussian* or *bell curve*) distribution, which is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\,,$$

where μ is the mean and σ^2 is the variance.

To compute probabilities for this distribution, we need to calculate integrals where the integrand depends not only on the variable x, but also on the parameters μ and σ. We might try to write a function M–file that has input parameters x, μ and σ, such as:

```
function f = NormalPDF(x, mu, sigma)
%
% Compute f(x) for normal PDF, with mean mu and standard deviation sigma.
%
c = 1 / (sigma*sqrt(2*pi));
d = (x - mu).^2 / (2*sigma^2);
f = c * exp( -d );
```

However, if we try to use `integral` to compute probabilities, say $\text{Prob}(-1 \le X \le 1)$:

```
p = integral(@NormalPDF, -1, 1)
```

then MATLAB will print an error message complaining that `mu` and `sigma` have not been defined. The difficulty here is that we would like to be able to specify values for `mu` and `sigma` without editing the function `NormalPDF`. One way to do this is to set specific values for μ and σ, and then use a function handle to pass the information to `integral`. For example, suppose we want to compute the integral when $\mu = 0$ and $\sigma = 1$. Then we can do the following:

```
mu = 0;
sigma = 1;
p = integral(@(x)NormalPDF(x, mu, sigma), -1, 1)
```

If we want to recompute the integral with different values of μ and σ, we repeat the above three statements, with new values of `mu` and `sigma`; for example:

```
mu = -2;
sigma = 5;
p = integral(@(x)NormalPDF(x, mu, sigma), -1, 1)
```

An alternative to is to use *nested functions*; that is, a function nested inside another function M–file. For example, consider the following function M–file:

```
function p = TestProbs(a, b, mu, sigma)
%
%    p = TestProbs(a, b, mu, sigma)
%
% Compute a <= prob(X) <= b for normal distribution
% with mean mu and standard deviation sigma.
%
p = integral(@NormalPDF, a, b);

  function f = NormalPDF(x)
  %
  % Compute f(x) for normal PDF, with mean mu, standard deviation sigma.
  %
  c = 1 / (sigma*sqrt(2*pi));
  d = (x - mu).^2 / (2*sigma^2);
  f = c * exp( -d );
  end

end
```

Nested functions have access to variables in their parent function. Thus, since `mu` and `sigma` are defined as input to `TestProbs`, they can also be used in the nested function `NormalPDF` without having to pass them as input. Note that both the nested function `NormalPDF`, and its parent function, `TestProbs`, must be closed with `end` statements[3].

Problem 5.3.4. *Suppose a set of test scores are distributed normally with mean $\mu = 78$ and standard deviation $\sigma = 10$.*

(a) Plot the probability density function, including marks on the x-axis denoting the mean, and $\mu \pm 2 * \sigma$.

(b) Implement `TestProbs` *and use it to answer the following questions:*

What percentage of the scores are between 0 and 100? Does this make sense?

What percentage of the scores are between 0 and 60?

What percentage of scores are between 90 and 100?

What percentage of scores are within 2 standard deviations of the mean?

[3]Actually, any function M–file in MATLAB can be closed with an `end` statement, but it is not always necessary; it is for nested functions.

Problem 5.3.5. *Modify* `TestProbs` *so that it can accept as input vectors* `mus` *and* `sigmas` *(which have the same number of entries), and returns as output a vector* `p` *containing the probabilities Prob*$(a \le X \le b)$ *corresponding to* `mus(i)` *and* `sigmas(i)`, $i = 1, 2, \ldots$

Problem 5.3.6. *Modify* `TestProbs` *so that instead of using a nested function, it uses the function handle approach:*

```
p = integral(@(x)NormalPDF(x, mu, sigma), a, b);
```

As with the previous problem, your code should be able to accept as input vectors `mus` *and* `sigmas` *(which have the same number of entries), and returns as output a vector* `p` *containing the probabilities Prob*$(a \le X \le b)$ *corresponding to* `mus(i)` *and* `sigmas(i)`, $i = 1, 2, \ldots$

Problem 5.3.7. *The intensity of light with wavelength* λ *travelling through a diffraction grating with* n *slits at an angle* θ *is given by*

$$I(\theta) = n^2 \frac{\sin^2 k}{k^2}, \quad \text{where} \quad k = \frac{\pi n d \sin\theta}{\lambda},$$

and d *is the distance between adjacent slits. A helium-neon laser with wavelength* $\lambda = 632.8 * 10^{-9} m$ *is emitting a narrow band of light, given by* $-10^{-6} < \theta < 10^{-6}$, *through a grating with* $10,000$ *slits spaced* $10^{-4} m$ *apart. Estimate the total light intensity,*

$$\int_{-10^{-6}}^{10^{-6}} I(\theta)\, d\theta$$

emerging from the grating. You should write a function that can compute $I(\theta)$ *for general values of* n, d *and* λ, *and which uses a nested function to evaluate the integrand (similar to the approach used in* `TestProbs`*).*

Integration of Tabular Data

Suppose that we do not know the integrand explicitly, but we can obtain function values at certain discrete points. In this case, instead of using `integral` or `quadl`, we could first fit a curve through the data, and then integrate the resulting curve. We can use any of the methods discussed in Chapter 4, but the most common approach is to use a piecewise polynomial, such as a spline. That is,

- Suppose we are given a set of data points, (x_i, f_i).
- Fit a spline, $S(x)$, through this data. Keep in mind that $S(x)$ is a piecewise polynomial on several intervals. In particular,

$$S(x) = \begin{cases} p_1(x) & x_0 \le x \le x_1 \\ p_2(x) & x_1 \le x \le x_2 \\ \vdots & \vdots \\ p_N(x) & x_{N-1} \le x \le x_N \end{cases}$$

- Then

$$\int_a^b f(x)\, dx \approx \int_a^b S(x)\, dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} S(x)\, dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} p_i(x)\, dx$$

Since each $p_i(x)$ is a polynomial, we can compute the integrals by hand. For example, if we were to use a linear spline, then the data are connected by straight lines, and

$$\int_{x_{i-1}}^{x_i} p_i(x)\, dx = \text{ area under a line } = \text{ area of a trapezoid.}$$

This approach is equivalent to using the composite trapezoidal rule. MATLAB provides a function that can be used to integrate tabular data using the composite trapezoidal rule, called `trapz`. Its basic usage is illustrated by:

```
I = trapz(x, y)
```

where `x` and `y` are vectors containing the points and the data values, respectively. Note that `trapz` does not estimate the error, and thus should not be used without using another method to provide an error estimate.

Instead of a linear spline, we might consider using a cubic spline interpolating function. In this case, each $p_i(x)$ is a cubic polynomial, which can be integrated exactly by any integration rule with $DOP \geq 3$. We will use Simpson's rule to obtain the following function to integrate tabular data:

```
function I = quad3(x, f)
%
%         I = quad3(x, f);
%
%  This function integrates tabular data using cubic splines.
%
%  Input:  x, f  - vectors containing the tabular data
%  Output:     I  - approximation of the integral
%
S = spline(x, f);
N = length(x);
I = 0;
for i = 1:N-1
  h = x(i+1) - x(i);
  I = I + (h/6)*(f(i) + 4 * ppval(S, (x(i) + x(i+1))/2) + f(i+1));
end
```

The code `quad3` has been written this way for clarity. It could be made more efficient by taking advantage of the repeated function values (at the subinterval ends) in the sum and by vectorizing the call to `ppval` over all the subinterval midpoints. This neat approach exploiting the properties of integration rules allows us to avoid the more direct, but longwinded, approach of extracting the cubic polynomial form of the spline on each subinterval using the MATLAB function `unmkpp` then integrating the resulting polynomials directly.

We can use our function `quad3` to provide an error estimate for `trapz`. Generally, we would expect `quad3` to be the more accurate as it uses cubic splines in contrast to the linear spline used by `trapz`. If we want an error estimate for `quad3`, we need another rule of at least the same order of accuracy, for example that discussed in Problem 5.3.8.

Example 5.3.7. The *dye dilution method* is used to measure cardiac output. Dye is injected into the right atrium and flows through the heart into the aorta. A probe inserted into the aorta measures the concentration of the dye leaving the heart. Let $c(t)$ be the concentration of the dye at time t. Then the cardiac output is given by

$$F = \frac{A}{\int_0^T c(t)\,dt},$$

where A is the amount of dye initially injected, and T is the time elapsed.

In this problem, $c(t)$ is not known explicitly; we can only obtain measurements at fixed times. Thus, we obtain a set of tabular data, (t_i, c_i). In order to compute F, we can use either MATLAB's `trapz` function, or our own `quad3` to approximate the integral.

For example, suppose a 5-mg bolus dye is injected into a right atrium. The concentration of the dye (in milligrams per liter) is measured in the aorta at one-second intervals. The collected data is shown in the table:

t	0	1	2	3	4	5	6	7	8	9	10
$c(t)$	0	0.4	2.8	6.5	9.8	8.9	6.1	4.0	2.3	1.1	0

Using `trapz` and `quad3` we can compute approximations of F as follows:

```
t = 0:10;
c = [0 0.4 2.8 6.5 9.8 8.9 6.1 4.0 2.3 1.1 0];
A = 5;
F1 = A / trapz(t, c)
F3 = A / quad3(t, c)
```

We see that $F1 \approx 0.1193$L/s and $F3 \approx 0.1192$L/s. These results would seem to imply that we can be fairly confident of the first three digits in both results. Note however that the data is only given to two digits. Assuming the data is accurate to the digits shown, you should be cautious about assuming significantly more accuracy in the integral than you observe in the data.

Problem 5.3.8. *Modify the* MATLAB *function* `quad3` *so that it uses* `pchip` *instead of* `spline`*. Why is the modification simple? Use the modified function to integrate the tabular data in Example 5.3.7. Does the result that you obtain increase your confidence that the value of the integral is 0.119L/s to three digits? Why or why not?*

Improper Integrals

In calculus, the term *improper integral* is used for situations where either $\pm\infty$ is one of the limits of integration, or if the integrand, $f(x)$, is not defined at a point in the interval (i.e., $f(x)$ has a *singularity*).

Example 5.3.8. Since the function $f(x) = \dfrac{1}{\sqrt{x}}$ is not defined at $x = 0$, we say the integrand for

$$\int_0^1 \frac{1}{\sqrt{x}}\,dx$$

is singular at the endpoint $x = 0$ (that is, it has an endpoint singularity).

The function $f(x) = \dfrac{\sin x}{x}$ is also not defined at $x = 0$, so the integrand of

$$\int_{-1}^1 \frac{\sin x}{x}\,dx$$

has a singularity within the interval of integration (i.e., not at an endpoint).

An example of an infinite interval of integration is

$$\int_1^\infty \frac{1}{x^2}\,dx$$

All of the integrals in this example can be computed using calculus techniques, and have well-defined solution.

The MATLAB function `integral` can generally be used to compute approximations of these types of improper integrals.

Endpoint Singularities

It is possible to use `integral` and `quadl` directly on problems with endpoint singularities. A divide by zero warning may sometimes be displayed in the MATLAB command window, but it is not necessarily a fatal error, and a quite accurate result is generally computed.

Example 5.3.9. The following MATLAB statements

```
f = @(x) 1 ./ sqrt(x);
I = integral(f, 0, 1)
```

compute the approximation $\int_0^1 \frac{1}{\sqrt{x}}\,dx \approx 1.999999999999763$, which is within the default absolute error tolerance of 10^{-10}.

Interior Singularities

If the singularity is not at an endpoint, but is within the interval of integration, then `integral` and `quadl` may not find the singularity during the computation, and thus they may compute an approximation of the integral without incurring a fatal error. However, because the quadrature points are chosen adaptively, it is possible that a fatal divide by zero error will be encountered, causing `integral` and `quadl` to fail. Therefore, if interior singularities are known, then it is best to split the integral so that all singularities occur at end points.

Example 5.3.10. If we attempt to use `integral` or `quadl` directly to compute an approximation of the convergent integral $\int_{-0.5}^1 \frac{\sin x}{x}\,dx$:

```
f = @(x) sin(x) ./ x;
I1 = integral(f, -0.5, 1)
I2 = quadl(f, -0.5, 1)
```

For both methods, the (removable) singularity at $x = 0$ is not encountered during the computation, and both compute an accurate approximation; `integral` gives the value 1.439190488410250, and `quadl` gives the value 1.439190488409625.

On the other hand if we try the same direct approach to compute $\int_{-1}^1 \frac{\sin x}{x}\,dx$,

```
f = @(x) sin(x) ./ x;
I1 = integral(f, -1, 1)
I2 = quadl(f, -1, 1)
```

then the singularity at $x = 0$ is not encountered when using `integral`; the computed approximation is 1.892166140734366. However, in the case of `quadl`, an NaN (not a number) results when the integrand is evaluated at the 0.

If an integral contains an interior singularity, it is safest to split the interval of integration at the singular point, and calculate the sum of the resulting two integrals. Because `integral` and `quadl` can handle endpoint singularities quite well, the individual integrals can then be computed fairly safely. Note, the singularity here is "removable" as $\lim_{x\to 0} \frac{\sin x}{x} = 1$ so we could easily avoid the problem by defining the integrand to have value one at $x = 0$.

Example 5.3.11. Although the previous example illustrated that `integral` and `quadl` do not fail when computing an approximation of the integral $\int_{-0.5}^{1} \frac{\sin x}{x}\, dx$, since we recognize there is a singularity at $x = 0$, it is safest to compute the approximation as follws:

```
f = @(x) sin(x) ./ x;
I1 = integral(f, -0.5, 0) + integral(f, 0, 1)
I2 = quadl(f, -0.5, 0) + quadl(f, 0, 1)
```

giving the value 1.439190488410250 when using `integral`, and 1.439190488410243 when using `quadl`, which agree with the values above in Example 5.3.10 to more than the default absolute accuracy request of `tol = 1.0e-6`. Similarly, an approximation of the integral $\int_{-1}^{1} \frac{\sin x}{x}\, dx$ should be computed using:

```
f = @(x) sin(x) ./ x;
I1b = integral(f, -1, 0) + integral(f, 0, 1)
I2b = quadl(f, -1, 0) + quadl(f, 0, 1)
```

which give the values 1.892166140734366 and 1.892166140734353, respectively.

Infinite Limits of Integration

The MATLAB function `quadl` cannot be used directly with infinite limits of integration, but a variety of techniques can be used to to modify the integral so that `quadl` can be used. The basic idea is to use an algebraic substitution that replaces the infinite limit of integration by a finite value. One approach that often works well is to use a substitution such as $x = \frac{1}{t}$. This is illustrated in the following example.

Example 5.3.12. Consider the integral $\int_{1}^{\infty} \frac{x^3}{x^5 + 2}\, dx$. If we attempt the following:

```
f = @(x) (x.^3)./(x.^5+2);
I = quadl(f, 1, Inf)
```

then MATLAB produces a warning: "Minimum step size reached; singularity possible", and returns `NaN` for the integral approximation. However, if we use the substitution $x = \frac{1}{t}$, we obtain

$$\int_{1}^{\infty} \frac{x^3}{x^5 + 2}\, dx = \int_{1}^{0} \frac{(1/t)^3}{(1/t)^5 + 2} \left(\frac{-1}{t^2}\right) dt = \int_{0}^{1} \frac{1}{1 + 2t^5}\, dt,$$

This can be easily integrated with `quadl` using the MATLAB statements:

```
f = @(t) 1 ./ (1 + 2*t.^5);
I = quadl(f, 0, 1)
```

giving the value 0.826798251972840.

Note that for some problems a substitution such as $x = \frac{1}{t}$ or $x = \frac{1}{t^2}$ might cause the transformed integral to have an endpoint singularity, but this usually is not a problem since `quadl` can work fairly safely for such situations.

The MATLAB function `integral` can be used with infinite limits of integration. The code recognizes `Inf` or `-Inf` and does an algebraic substitution automatically.

Example 5.3.13. Repeating the previous example

$$\int_1^\infty \frac{x^3}{x^5+2}\,dx$$

with `integral`, we can simply use:

```
f = @(x) (x.^3)./(x.^5+2);
I = integral(f, 1, Inf)
```

to obtain the value 0.826798251977108.

A sometimes-used approach (but one we do not recommend) to dealing with integrals with infinite limits is to truncate the interval then use a finite interval of integration method.

Example 5.3.14. Continuing the previous example, we could approximate "infinity" by 10, and the MATLAB commands

```
f = @(x) (x.^3) ./ (x.^5 + 2);
I = integral(f, 1, 10)
```

give the value 0.726798585306805, which is not a very good approximation. If, instead we approximate "infinity" by 100 we get 0.816798251977442, and by 1000 we get 0.825798251977103. This is very slowly convergent integral, so this slow convergence of the integrals over truncated intervals is to be expected.

To make the integral converge faster we can "subtract out the singularity".

Example 5.3.15. Continuing the previous example, we could subtract out the singularity $\frac{1}{x^2}$, which is the limit of the integrand as $x \to \infty$. This gives

$$\int_1^\infty \frac{x^3}{x^5+2}\,dx = \int_1^\infty \frac{1}{x^2}\,dx - \int_1^\infty \frac{2}{x^2(x^5+2)}\,dx = 1 - \int_1^\infty \frac{2}{x^2(x^5+2)}\,dx$$

We then compute the integral using MATLAB script

```
f = @(x) 2 ./ (x.^2.*(x.^5 + 2));
I = 1 - integral(f, 1, 10)
```

which gives 0.826798585306805. This time approximating infinity by ten gives a much more accurate result as by subtracting out the singularity we have produced a much faster converging integral.

Problem 5.3.9. *Determine the singular points for each of the following integrands, and compute approximations of each of the integrals.*

(a) $\displaystyle\int_0^1 \frac{e^{-x}}{\sqrt{1-x}}\,dx$ *(b)* $\displaystyle\int_4^5 \frac{1}{(5-t)^{2/5}}\,dt$ *(c)* $\displaystyle\int_0^1 \frac{1}{\sqrt{x}(1+x)}\,dx$

Problem 5.3.10. *Use each of the* MATLAB *functions* `integral` *and* `quadl` *to compute the integral* $\displaystyle\int_0^1 \frac{1}{\sqrt{x}}\,dx$ *for the absolute error tolerances* $tol = 10^{-i}$, $i = 1, 2, \ldots, 12$ *in turn. Tabulate your results and their absolute errors. What do you observe from your tabulated results? What else did you observe when running your* MATLAB *script.*

Problem 5.3.11. *An approximation of the integral* $\displaystyle\int_{-0.5}^{1} \frac{e^x - 1}{x}\,dx$ *is to be computed using:*

```
f = @(x) (exp(x)-1) ./ x;
I = integral(f, -0.5, 1)
```

and of the integral $\displaystyle\int_{-1}^{1} \frac{e^x - x}{x}\,dx$ *similarly. What do you observe? There is a singularity. Is it removable? Can you split the range of integration appropriately?*
Repeat your experiment using `quadl` *in place of* `integral`.
To five digits what is the correct value of both integrals? Why?

Problem 5.3.12. *Consider the integral* $\displaystyle\int_{1}^{7} \frac{\sin(x-4)}{(x-4)}\,dx$. *At what value* x *is the integrand singular? Attempt to use* `integral` *and* `quadl` *directly to compute an approximation of this integral. Now, split the interval of integration into two parts at the singular point and repeat your experiment. Comment on your results.*

Problem 5.3.13. *Compute approximations to the following using* `integral` *and* `quadl`. *In the case of* `quadl`, *use the substitution* $x = \dfrac{1}{t}$. *Compare the results.*

(a) $\displaystyle\int_{1}^{\infty} \frac{\sin x}{x^2}\,dx$ *(b)* $\displaystyle\int_{1}^{\infty} \frac{2}{\sqrt{x}(1+x)}\,dx$ *(c)* $\displaystyle\int_{-\infty}^{3} \frac{1}{x^2+9}\,dx$

Problem 5.3.14. *Compute an approximation of the integral*

$$\int_{0}^{\infty} \frac{1}{\sqrt{x}(1+x)}\,dx$$

Use both `integral` *and* `quadl`.

Problem 5.3.15. *In a single* MATLAB *program use each of the functions* `integral` *and* `quadl` *to compute all three of the definite integrals:*

1. $\displaystyle\int_{0}^{8} [\,e^{-3x} - \cos(5\pi x)\,]\,dx$
2. $\displaystyle\int_{-1}^{2} \left(\left| x - \frac{1}{\sqrt{3}} \right| + \left| x + \frac{1}{\sqrt{2}} \right| \right) dx$
3. $\displaystyle\int_{0}^{1} x^{-\frac{2}{3}}\,dx$

setting the error tolerance to give first and absolute error of `tol=1.0e-6` *and then* `tol=1.0e-12` *In each case keep track of the number of integrand evaluations used to compute the integral. Do you get consistent results with the different tolerances and integrators? If* MATLAB *produces warnings, can you explain these?*