Chapter 5

Integrals and derivatives

In the preceding chapters we introduced the central elements of computer programming using Python and solved some simple physics problems using what we learned. You will get plenty of further opportunities to polish your programming skills, but our main task from here on is to learn about the ideas and techniques of computational physics, the physical and mathematical insights that allow us to perform accurate calculations of physical quantities on the computer.

One of the most basic but also most important applications of computers in physics is the evaluation of integrals and derivatives. Numerical evaluation of integrals is a particularly crucial topic because integrals occur widely in physics calculations and, while some integrals can be done analytically in closed form, most cannot. They can, however, almost always be done on a computer. In this chapter we examine a number of different techniques for evaluating integrals and derivatives, as well as taking a brief look at the related operation of interpolation.

5.1 Fundamental methods for evaluating integrals

Suppose we are given a mathematical function and we wish to evaluate its integral over a specified domain. Let us consider initially the simplest case, the integral of a function of a single variable over a finite range. We will study a number of techniques for the numerical evaluation of such integrals, but we start with the most basic—and also most widely used—the trapezoidal rule.

5.1.1 The trapezoidal rule

Suppose we have a function $f(x)$ and we want to calculate its integral with respect to x from $x = a$ to $x = b$, which we denote $I(a, b)$:

$$I(a,b) = \int_a^b f(x)\,\mathrm{d}x. \tag{5.1}$$

This is equivalent to calculating the area under the curve of $f(x)$ from a to b. There is no known way to calculate such an area exactly in all cases on a computer, but

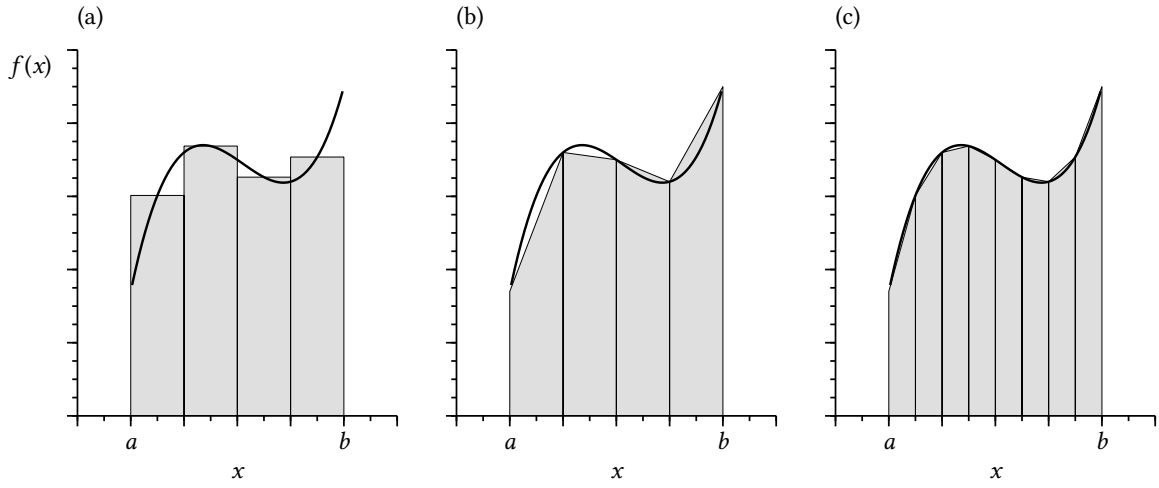


Figure 5.1: Estimating the area under a curve. (a) A simple scheme for estimating the area under a curve by dividing the area into rectangular slices. The gray shaded area approximates the area under the curve, though not very well. (b) The trapezoidal rule approximates the area as a set of trapezoids, and is usually more accurate. (c) With a larger number of slices, the shaded area is a better approximation to the true area under the curve.

we can do it approximately by the method shown in Fig. 5.1a: we divide the area up into rectangular slices, calculate the area of each one, and then add them up. This, however, is a pretty poor approximation. The area under the rectangles is not very close to the area under the curve.

A better approach, which involves very little extra work, is that shown in Fig. 5.1b, where the area is divided into trapezoids rather than rectangles. The area under the trapezoids is a considerably better approximation to the area under the curve, and this approach, though simple, often gives perfectly adequate results.

Suppose we divide the interval from a to b into N slices or steps, so that each slice has width $h = (b-a)/N$. Then the right-hand side of the kth slice falls at $a+kh$, and the left-hand side falls at $a+kh-h = a+(k-1)h$. Thus the area of the trapezoid for this slice is

$$A_k = \tfrac{1}{2}h\bigl[f(a+(k-1)h) + f(a+kh)\bigr]. \tag{5.2}$$

This is the *trapezoidal rule*. It gives us a trapezoidal approximation to the area under one slice of our function.

Our approximation for the area under the whole curve is the sum of the areas of

the trapezoids for all N slices:

$$\begin{aligned} I(a,b) \simeq \sum_{k=1}^{N} A_k &= \tfrac{1}{2}h \sum_{k=1}^{N} \bigl[f(a+(k-1)h) + f(a+kh) \bigr] \\ &= h\bigl[\tfrac{1}{2}f(a) + f(a+h) + f(a+2h) + \ldots + \tfrac{1}{2}f(b) \bigr] \\ &= h\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a+kh) \right]. \end{aligned} \tag{5.3}$$

This is the *extended trapezoidal rule*—it is the extension to many slices of the basic trapezoidal rule of Eq. (5.2). Being slightly sloppy in our usage, however, we will often refer to it simply as the trapezoidal rule. Note the structure of the formula: the quantity inside the square brackets is a sum over values of $f(x)$ measured at equally spaced points in the integration domain, and we take a half of the values at the start and end points but one times the value at all the interior points.

The trapezoidal rule is only an approximation to the area under the curve—it is clear from Fig. 5.1 that the trapezoids do not follow the curve perfectly. We can improve the approximation by using a larger number of slices N, as shown in Fig. 5.1c, although the program will also take longer to reach an answer because there are more terms in the sum to evaluate. We examine the variation in accuracy with N in more detail in Section 5.2.

Example 5.1: Integrating a function

Let us use the trapezoidal rule to calculate the integral of the function $x^4 - 2x + 1$ from $x = 0$ to $x = 2$. This is actually an integral we can do by hand, which means we don't really need to do it using the computer in this case, but it is a good first example because we can check easily if our program is working and how accurate an answer it gives.

Here is a program to do the integration using the trapezoidal rule with $N = 10$ slices:

File: `trapezoidal.py`

```
def f(x):
    return x**4 - 2*x + 1

N = 10
a = 0.0
b = 2.0
h = (b-a)/N

s = 0.5*f(a) + 0.5*f(b)
for k in range(1,N):
    s += f(a+k*h)
print(h*s)
```

This is a straightforward translation of the trapezoidal rule formula into computer code: we create a function that calculates the integrand, set up all the constants used, evaluate the sum in Eq. (5.3) term by term, and then multiply it by h and print it out. If we run the program it prints

```
4.50656
```

The correct answer is

$$\int_0^2 (x^4 - 2x + 1)\,\mathrm{d}x = \left[\tfrac{1}{5}x^5 - x^2 + x\right]_0^2 = 4.4. \tag{5.4}$$

So our calculation is moderately accurate but not exceptionally so—the answer is off by about 2%.

We can make the calculation more accurate by increasing the number of slices N. If we increase N to 100 and run the program again we get 4.40107, which is now accurate to 0.02%, which is pretty good. And if we use $N = 1000$ we get 4.40001, which is accurate to 0.0002%. In Section 5.2 we will study in more detail the accuracy of the trapezoidal rule.

Exercise 5.1: In the online resources you will find a file called `velocities.txt`, which contains two columns of numbers, the first representing time t in seconds and the second the x-velocity in meters per second of a particle, measured once every second from $t = 0$ to $t = 100$. The first few lines look like this:

```
0       0
1       0.069478
2       0.137694
3       0.204332
4       0.269083
5       0.331656
```

Write a program to do the following:

a) Read in the data and, using the trapezoidal rule, calculate the approximate distance traveled by the particle in the x direction as a function of time. See Section 2.4.3 on page 55 if you want a reminder of how to read data from a file.

b) Extend your program to make a graph that shows the original velocity curve and the distance traveled as a function of time, both on the same plot.

5.1.2 Simpson's rule

The trapezoidal rule is the simplest of numerical integration methods, requiring only a few lines of code to implement, but it is often perfectly adequate for calculations

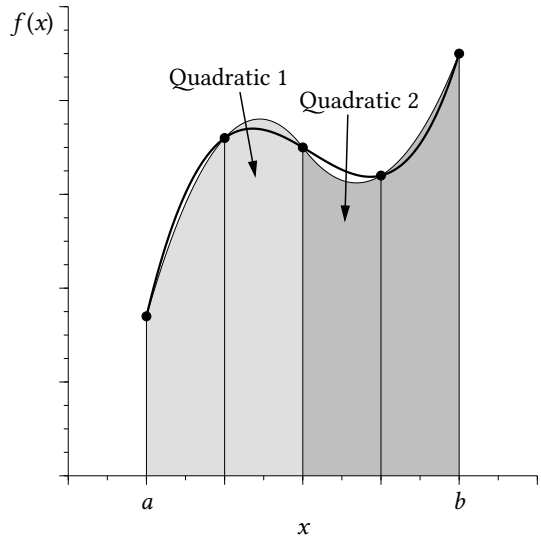


Figure 5.2: Simpson's rule. Simpson's rule involves fitting quadratic curves to pairs of slices and then calculating the area under the quadratics.

where no great accuracy is required. It happens frequently in physics calculations that we don't need an answer accurate to many significant figures and in such cases the ease and simplicity of the trapezoidal rule can make it the method of choice. One should not turn up one's nose at simple methods like this; they play an important role and are used widely. Moreover, the trapezoidal rule is the basis for several other more sophisticated methods of evaluating integrals, including the adaptive methods that we will study in Section 5.3 and the Romberg integration method of Section 5.4.

However, there are also cases where greater accuracy is required. As we have seen we can increase the accuracy of the trapezoidal rule by increasing the number N of steps used in the calculation. But in some cases a very large number of steps may be needed to achieve the desired accuracy, which means the calculation can become slow. There are other, more advanced schemes for calculating integrals that can achieve high accuracy with a smaller number of steps and quicker running time. In this section we study one such scheme, Simpson's rule.

In effect, the trapezoidal rule estimates the area under a curve by approximating the curve with straight-line segments—see Fig. 5.1b. We can often get a better result if we approximate the function instead with curves of some kind. Simpson's rule does this using quadratic curves, as shown in Fig. 5.2. In order to specify a quadratic completely one needs three points, not just two as with a straight line. So in this method we take a pair of adjacent slices and fit a quadratic through the three points that mark the boundaries of those slices. In Fig. 5.2 there are two quadratics, fitted to four slices. Simpson's rule involves approximating the integrand with quadratics in

this way, then calculating the area under those quadratics, which gives an approximation to the area under the true curve.

Suppose, as before, that our integrand is $f(x)$ and the spacing of adjacent points is h. And suppose for the purposes of argument that we have three points at $x = -h$, 0, and $+h$. If we fit a quadratic $Ax^2 + Bx + C$ through these points, then by definition we will have

$$f(-h) = Ah^2 - Bh + C, \qquad f(0) = C, \qquad f(h) = Ah^2 + Bh + C. \tag{5.5}$$

Solving these equations simultaneously for A, B, and C gives

$$A = \frac{1}{h^2}\bigl[\tfrac{1}{2}f(-h) - f(0) + \tfrac{1}{2}f(h)\bigr], \quad B = \frac{1}{2h}\bigl[f(h) - f(-h)\bigr], \quad C = f(0), \tag{5.6}$$

and the area under the curve of $f(x)$ from $-h$ to $+h$ is given approximately by the area under the quadratic:

$$\int_{-h}^{h} (Ax^2 + Bx + C)\,\mathrm{d}x = \tfrac{2}{3}Ah^3 + 2Ch = \tfrac{1}{3}h\bigl[f(-h) + 4f(0) + f(h)\bigr]. \tag{5.7}$$

This is *Simpson's rule*. It gives us an approximation to the area under two adjacent slices of our function. Note that the final formula for the area involves only the value of the function at evenly spaced points, just as with the trapezoidal rule. So to use Simpson's rule we don't actually have to fit a quadratic—we just plug numbers into this formula and it gives us an answer. This makes Simpson's rule almost as simple to use as the trapezoidal rule, and yet Simpson's rule often gives much more accurate results, as we will see.

To use Simpson's rule to perform a general integral we note that Eq. (5.7) does not depend on the fact that our three points lie at $x = -h$, 0, and $+h$. If we were to slide the curve along the x-axis to either higher or lower values, the area underneath it would not change. So we can use the same rule for any three uniformly spaced points. Applying Simpson's rule involves dividing the domain of integration into many slices and using the rule to separately estimate the area under successive pairs of slices, then adding the estimates for all pairs to get the final answer. If, as before, we are integrating from $x = a$ to $x = b$ in slices of width h then the three points bounding the first pair of slices fall at $x = a$, $a + h$ and $a + 2h$, those bounding the second pair at $a + 2h$, $a + 3h$, $a + 4h$, and so forth. Then the approximate value of the entire integral is given by

$$\begin{aligned} I(a,b) \simeq\ & \tfrac{1}{3}h\bigl[f(a) + 4f(a+h) + f(a+2h)\bigr] \\ & + \tfrac{1}{3}h\bigl[f(a+2h) + 4f(a+3h) + f(a+4h)\bigr] + \ldots \\ & + \tfrac{1}{3}h\bigl[f(a+(N-2)h) + 4f(a+(N-1)h) + f(b)\bigr]. \end{aligned} \tag{5.8}$$

Note that the total number of slices must be even for this to work. Collecting terms together, we now have

$$\begin{aligned} I(a,b) &\simeq \tfrac{1}{3}h\bigl[f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \ldots + f(b)\bigr] \\ &= \tfrac{1}{3}h\biggl[f(a) + f(b) + 4\sum_{\substack{k \text{ odd} \\ 1\ldots N-1}} f(a+kh) + 2\sum_{\substack{k \text{ even} \\ 2\ldots N-2}} f(a+kh)\biggr]. \end{aligned} \tag{5.9}$$

This formula is sometimes called the *extended Simpson's rule*, by analogy with the extended trapezoidal rule of Section 5.1.1, although for the sake of brevity we will just refer to it as Simpson's rule.

The sums over odd and even values of k can be conveniently accomplished in Python using a for loop of the form "`for k in range(1,N,2)`" for the odd terms or "`for k in range(2,N,2)`" for the even terms. Alternatively, we can rewrite Eq. (5.9) as

$$I(a,b) \simeq \tfrac{1}{3}h\biggl[f(a) + f(b) + 4\sum_{k=1}^{N/2} f(a+(2k-1)h) + 2\sum_{k=1}^{N/2-1} f(a+2kh)\biggr], \tag{5.10}$$

and just use an ordinary for loop (although this form is usually less convenient).

Comparing Eqs. (5.3) and (5.9) we see that Simpson's rule is modestly more complicated than the trapezoidal rule, but not enormously so. Programs using it are still straightforward to create.

As an example of the use of Simpson's rule, suppose we apply it with $N = 10$ slices to the integral from Example 5.1, $\int_0^2 (x^4 - 2x + 1)\,\mathrm{d}x$, whose true value, as we saw, is 4.4. As shown in Exercise 5.2, Simpson's rule gives an answer of 4.400427 in this case, which is already accurate to better than 0.01%, orders of magnitude better than the trapezoidal rule with $N = 10$. Results for $N = 100$ and $N = 1000$ are better still—see the exercise.

If you need an accurate answer for an integral, Simpson's rule is a good choice in many cases, giving precise results with relatively little effort. Alternatively, if you need to evaluate an integral quickly—perhaps because you will be evaluating many integrals as part of a larger calculation—then Simpson's rule may again be a good choice, since it can give accurate answers even with only a small number of steps.

Exercise 5.2:

a) Write a program to calculate an approximate value for the integral $\int_0^2 (x^4 - 2x + 1)\,\mathrm{d}x$ from Example 5.1, but using Simpson's rule with 10 slices instead of the trapezoidal rule. You may wish to base your program on the trapezoidal rule program on page 135.

b) Run the program and compare your result to the known correct value of 4.4. What is the fractional error on your calculation?

c) Modify the program to use a hundred slices instead, then a thousand. Note the improvement in the result. How do the results compare with those from Example 5.1 for the trapezoidal rule with the same numbers of slices?

Exercise 5.3: Consider the integral

$$E(x) = \int_0^x \mathrm{e}^{-t^2}\,\mathrm{d}t.$$

a) Write a program to calculate $E(x)$ for values of x from 0 to 3 in steps of 0.1. Choose for yourself what method you will use for performing the integral and a suitable number of slices.

b) When you are convinced your program is working, extend it further to make a graph of $E(x)$ as a function of x. If you want to remind yourself of how to make a graph, consult Section 3.1, starting on page 86.

There is no known way to perform this particular integral analytically, so numerical approaches are the only way forward.

Exercise 5.4: The diffraction limit of a telescope

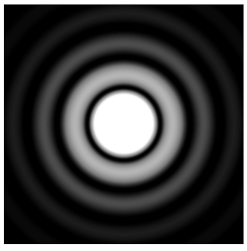
Our ability to resolve detail in astronomical observations is limited by the diffraction of light in our telescopes. Light from stars can be treated effectively as coming from a point source at infinity. When such light, with wavelength λ, passes through the circular aperture of a telescope (which we will assume to have unit radius) and is focused by the telescope in the focal plane, it produces not a single dot, but a circular diffraction pattern consisting of a central spot surrounded by a series of concentric rings. The intensity of the light in this diffraction pattern is given by

$$I(r) = I_0 \left(\frac{J_1(kr)}{kr}\right)^2,$$

where I_0 is a constant, r is the distance in the focal plane from the center of the diffraction pattern, $k = 2\pi/\lambda$, and $J_1(x)$ is a Bessel function. The Bessel functions $J_m(x)$ are given by

$$J_m(x) = \frac{1}{\pi}\int_0^\pi \cos(m\theta - x\sin\theta)\,\mathrm{d}\theta,$$

where m is a nonnegative integer and $x \geq 0$.



The diffraction pattern produced by a point source of light.

a) Write a Python function `J(m,x)` that calculates the value of $J_m(x)$ using Simpson's rule with $N = 1000$ points. Use your function in a program to make a plot, on a single graph, of the Bessel functions J_0, J_1, and J_2 as a function of x from $x = 0$ to $x = 20$.

b) Write a second program that makes a density plot of the intensity of the circular diffraction pattern of a point light source with $\lambda = 500$ nm, in a square region of the focal plane, using the formula given above. Your picture should cover values of r from zero up to about 1 μm.

If you have done the calculation correctly, your density plot should look something like the figure shown here.

Hint 1: You may find it useful to know that $\lim_{x\to 0} J_1(x)/x = \frac{1}{2}$. Hint 2: The central spot in the diffraction pattern is so bright that it may be difficult to see the rings around it on the

computer screen. If you run into this problem a simple way to deal with it is to use one of the other color schemes for density plots described in Section 3.3. The "`hot`" scheme works well. For a more sophisticated solution to the problem, the `imshow` function has an additional argument `vmax` that allows you to set the value that corresponds to the brightest point in the plot. For instance, if you say "`imshow(x,vmax=0.1)`", then elements in `x` with value 0.1, or any greater value, will produce the brightest (most positive) color on the screen. By lowering the `vmax` value, you can reduce the total range of values between the minimum and maximum brightness, and hence increase the sensitivity of the plot, making subtle details visible. For this exercise a value of `vmax=0.01` appears to work well. (There is also a `vmin` argument that can be used to set the value that corresponds to the dimmest (most negative) color.)

5.2 Errors on integrals

Our numerical integrals are only approximations. As with most numerical calculations there is usually a rounding error when we calculate an integral, as described in Section 4.2, but this is not the main source of error. The main source of error is the so-called *approximation error*—the fact that our integration rules themselves are only approximations to the true integral. Both the trapezoidal and Simpson rules calculate the area under an approximation (either linear or quadratic) to the integrand, not the integrand itself. How big an error does this approximation introduce?

Consider again an integral $\int_a^b f(x)\,\mathrm{d}x$, and let us look first at the trapezoidal rule of Eq. (5.3). To simplify our notation a little, let us define $x_k = a + kh$ as a shorthand for the positions at which we evaluate the integrand $f(x)$. We will refer to these positions as *sample points*. Now consider one particular slice of the integral, the one that falls between x_{k-1} and x_k, and let us perform a Taylor expansion of $f(x)$ about x_{k-1} thus:

$$f(x) = f(x_{k-1}) + (x - x_{k-1})f'(x_{k-1}) + \tfrac{1}{2}(x - x_{k-1})^2 f''(x_{k-1}) + \ldots \tag{5.11}$$

where f' and f'' denote the first and second derivatives of f respectively. Integrating this expression from x_{k-1} to x_k gives

$$\begin{aligned}\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = f(x_{k-1}) \int_{x_{k-1}}^{x_k} \mathrm{d}x + f'(x_{k-1}) \int_{x_{k-1}}^{x_k} (x - x_{k-1})\,\mathrm{d}x \\ + \tfrac{1}{2} f''(x_{k-1}) \int_{x_{k-1}}^{x_k} (x - x_{k-1})^2\,\mathrm{d}x + \ldots\end{aligned} \tag{5.12}$$

Now we make the substitution $u = x - x_{k-1}$, which gives

$$\begin{aligned}\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x &= f(x_{k-1}) \int_0^h \mathrm{d}u + f'(x_{k-1}) \int_0^h u\,\mathrm{d}u + \tfrac{1}{2} f''(x_{k-1}) \int_0^h u^2\,\mathrm{d}u + \ldots \\ &= hf(x_{k-1}) + \tfrac{1}{2}h^2 f'(x_{k-1}) + \tfrac{1}{6}h^3 f''(x_{k-1}) + \mathrm{O}(h^4),\end{aligned} \tag{5.13}$$

where $\mathrm{O}(h^4)$ denotes the rest of the terms in the series, those in h^4 and higher, which we are neglecting.

We can do a similar expansion around $x = x_k$ and again integrate from x_{k-1} to x_k to get

$$\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = hf(x_k) - \tfrac{1}{2}h^2 f'(x_k) + \tfrac{1}{6}h^3 f''(x_k) - \mathrm{O}(h^4). \tag{5.14}$$

Then, taking the average of Eqs. (5.13) and (5.14), we get

$$\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = \tfrac{1}{2}h[f(x_{k-1}) + f(x_k)] + \tfrac{1}{4}h^2[f'(x_{k-1}) - f'(x_k)] + \tfrac{1}{12}h^3[f''(x_{k-1}) + f''(x_k)] + \mathrm{O}(h^4). \tag{5.15}$$

Finally, we sum this expression over all slices k to get the full integral that we want:

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &= \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x \\ &= \tfrac{1}{2}h \sum_{k=1}^{N} [f(x_{k-1}) + f(x_k)] + \tfrac{1}{4}h^2[f'(a) - f'(b)] \\ &\quad + \tfrac{1}{12}h^3 \sum_{k=1}^{N} [f''(x_{k-1}) + f''(x_k)] + \mathrm{O}(h^4).\end{aligned} \tag{5.16}$$

Let us take a close look at this expression to see what is going on.

The first sum on the right-hand side of the equation is precisely equal to the trapezoidal rule, Eq. (5.3). When we use the trapezoidal rule, we evaluate only this sum and discard all the terms following. The size of the discarded terms—the rest of the series—measures the amount we would have to add to the trapezoidal rule value to get the true value of the integral. In other words it is equal to the error we incur when we use the trapezoidal rule, the so-called approximation error.

In the second term, the term in h^2, notice that almost all of the terms have canceled out of the sum, leaving only the first and last terms, the ones evaluated at a and b. Although we haven't shown it, a similar cancellation happens for the terms in h^4, h^6, and all even powers of h.

Now take a look at the term in h^3 and note the following useful fact: the sum in this term is itself, to within an overall constant, just the trapezoidal rule approximation to the integral of $f''(x)$ over the interval from a to b. Specifically, if we take Eq. (5.16) and make the substitution $f(x) \to f''(x)$ on the left-hand side, we get

$$\int_a^b f''(x)\,\mathrm{d}x = \tfrac{1}{2}h \sum_{k=1}^{N} [f''(x_{k-1}) + f''(x_k)] + \mathrm{O}(h^2). \tag{5.17}$$

Multiplying by $\frac{1}{6}h^2$ and rearranging, we then get

$$\begin{aligned}\tfrac{1}{12}h^3 \sum_{k=1}^{N} [f''(x_{k-1}) + f''(x_k)] &= \tfrac{1}{6}h^2 \int_a^b f''(x)\,\mathrm{d}x + \mathrm{O}(h^4) \\ &= \tfrac{1}{6}h^2[f'(b) - f'(a)] + \mathrm{O}(h^4),\end{aligned} \tag{5.18}$$

since the integral of $f''(x)$ is just $f'(x)$. Substituting this result into Eq. (5.16) and canceling some terms, we find that

$$\int_a^b f(x)\,\mathrm{d}x = \tfrac{1}{2}h\sum_{k=1}^{N}[f(x_{k-1}) + f(x_k)] + \tfrac{1}{12}h^2[f'(a) - f'(b)] + \mathrm{O}(h^4). \tag{5.19}$$

Thus, to leading order in h, the value of the terms dropped when we use the trapezoidal rule, which equals the approximation error δ on the integral, is

$$\delta = \tfrac{1}{12}h^2\bigl[f'(a) - f'(b)\bigr]. \tag{5.20}$$

This is the *Euler–Maclaurin formula* for the error on the trapezoidal rule. More correctly it is the first term in the Euler–Maclaurin formula; the full formula keeps the terms to all orders in h. We can see from Eq. (5.19) that the next term in the series is of order h^4. We might imagine it would be of order h^3, but the h^3 term cancels out, and in fact it is fairly straightforward to show that only even powers of h survive in the full formula at all orders, so the next term after h^4 is h^6, then h^8, and so forth. So long as h is small, however, we can neglect the h^4 and higher terms—the leading term, Eq. (5.20), is usually enough.

Equation (5.19) tells us that the trapezoidal rule is a *first-order* integration rule, which means it is accurate up to and including terms proportional to h and the leading-order approximation error is of order h^2. That is, a first-order rule is *accurate* to $\mathrm{O}(h)$ and has an *error* $\mathrm{O}(h^2)$.

In addition to approximation error, there is also a rounding error on our calculation. As discussed in Section 4.2, this rounding error will have approximate size ϵ times the value of the integral, where ϵ is the machine precision, which is about 10^{-16} in current versions of Python.[1] Equation (5.20) tells us that the approximation error gets smaller as h gets smaller, so we can make our integral more accurate by using smaller h or, equivalently, a larger number N of slices. However, there is no point making h so small that the approximation error becomes smaller than the rounding error. Further decreases in h beyond this point will only make our program slower, by increasing the number of terms in the sum for Eq. (5.3), without improving the accuracy of our calculation significantly, since accuracy will be dominated by the rounding error.

Thus decreases in h will only help us up to the point at which the approximation

[1] One might imagine that the rounding error would be larger than this because the trapezoidal rule involves a sum of terms in Eq. (5.3) and each term will incur its own rounding error, the individual errors accumulating over the course of the calculation. However, standard results for random variables tell us that the size of such cumulative errors goes up only as $\sqrt{N}$, while the trapezoidal rule equation (5.3) includes a factor of h, which falls off as $1/N$. The net result is that the theoretical cumulative error on the trapezoidal rule actually decreases as $1/\sqrt{N}$, rather than increasing, so the final error is well approximated by the error incurred on the final operation of the calculation, which will have size ϵ times the final value.

and rounding errors are roughly equal, which is the point where

$$\tfrac{1}{12}h^2\big[f'(a) - f'(b)\big] \simeq \epsilon \int_a^b f(x)\,\mathrm{d}x. \tag{5.21}$$

Rearranging for h we get

$$h \simeq \sqrt{\frac{12\int_a^b f(x)\,\mathrm{d}x}{f'(a) - f'(b)}}\,\epsilon^{1/2}. \tag{5.22}$$

Or we can set $h = (b-a)/N$ to get

$$N \simeq (b-a)\sqrt{\frac{f'(a) - f'(b)}{12\int_a^b f(x)\,\mathrm{d}x}}\,\epsilon^{-1/2}. \tag{5.23}$$

Thus if, for example, all the factors except the last are of order unity, then rounding error will become important when $N \simeq 10^8$. This is the point at which the accuracy of the trapezoidal rule reaches the limits of accuracy of the computer. There is no point increasing the number of integration slices beyond this point; the calculation will not become any more accurate. However, $N = 10^8$ would be an unusually large number of slices for the trapezoidal rule—it would be rare to use such a large number when equivalent accuracy can be achieved using much smaller N with a more accurate rule such as Simpson's rule. In most practical situations, therefore, we will be in the regime where approximation error is the dominant source of inaccuracy for the trapezoidal rule and it is safe to assume that rounding error can be ignored.

We can do an analogous error analysis for Simpson's rule. The algebra is similar but more tedious. Here we will just quote the results. For an integral over the interval from a to b, the approximation error is given to leading order by

$$\delta = \tfrac{1}{180}h^4\big[f'''(a) - f'''(b)\big]. \tag{5.24}$$

Thus Simpson's rule is a third-order integration rule—two orders better than the trapezoidal rule—with a fourth-order approximation error. For small values of h this means that the error on Simpson's rule will typically be much smaller than the error on the trapezoidal rule and it explains why Simpson's rule gave such superior results in our example calculations (see Section 5.1.2).

The rounding error for Simpson's rule is again of order $\epsilon\int_a^b f(x)\,\mathrm{d}x$ and the equivalent of Eq. (5.23) is

$$N = (b-a)\left(\frac{f'''(a) - f'''(b)}{180\int_a^b f(x)\,\mathrm{d}x}\right)^{1/4}\epsilon^{-1/4}. \tag{5.25}$$

If, again, the leading factors are roughly of order unity, this implies that rounding error will become important when $N \simeq 10\,000$. Beyond this point Simpson's rule is so

accurate that it exceeds the accuracy of the computer itself and there is no point using larger values of N. By contrast with the case for the trapezoidal rule, $N = 10\,000$ is not an unusually large number of slices to use in a calculation. Calculations with ten thousand slices can often be done in a fraction of a second. Thus it is worth bearing this result in mind: there is no point using more than a few thousand slices with Simpson's rule because the calculation will reach the limits of precision of the computer and larger values of N will do no further good.

Finally in this section, let us note that while Simpson's rule does in general give superior accuracy, it is not always guaranteed to do better than the trapezoidal rule, since the errors on the trapezoidal and Simpson rules also depend on derivatives of the integrand function via Eqs. (5.20) and (5.24). It would be possible, for instance, for $f'''(a)$ by bad luck to be large in some particular instance, making the error in Eq. (5.24) similarly large, and possibly worse than the error for the trapezoidal rule. It is fair to say that Simpson's rule usually gives better results than the trapezoidal rule, but the prudent scientist will bear in mind that it can do worse on occasion.

5.2.1 Practical estimation of errors

The Euler–Maclaurin formula of Eq. (5.20), or the equivalent for Simpson's rule in Eq. (5.24), allows us to calculate the error on our integrals provided we have a known closed-form expression for the integrand $f(x)$, so that we can calculate the derivatives that appear in the formulas. Unfortunately, in many cases—perhaps most—we have no such expression. For instance, the integrand may not be a mathematical function at all but a set of measurements made in the laboratory, or it might itself be the output of another computer program. In such cases Eq. (5.20) or (5.24) will not work. There is, however, still a way to calculate the error.

Suppose, as before, that we are evaluating an integral over the interval from $x = a$ to $x = b$ and let us assume that we are using the trapezoidal rule—it makes the argument simpler, although the method described here extends to Simpson's rule too. Let us perform the integral with some number of steps N_1, so that the step size is $h_1 = (b-a)/N_1$, and let us denote by I_1 the value of the integral that we calculate.

Here is the trick: we now double the number of steps and perform the integral again. That is we define a new number of steps $N_2 = 2N_1$ and a new step size $h_2 = (b-a)/N_2 = \frac{1}{2}h_1$ and we reevaluate the integral using the trapezoidal rule, giving a new answer I_2, which will normally be more accurate than the previous one. As we have seen, the trapezoidal rule introduces an error of order $\mathrm{O}(h^2)$, which means that when we half the value of h we *quarter* the size of the error. Knowing this fact allows us to estimate how big the error is.

Suppose that the true value of the integral is I. The difference between the true value and our first estimate I_1 is equal by definition to the error on that estimate, which as we have said is proportional to h^2, so let us write it as ch^2, where c is a constant. Then I and I_1 are related by $I = I_1 + ch_1^2$, neglecting higher-order terms.

We can also write a similar formula for our second estimate I_2 of the integral, with N_2 steps: $I = I_2 + ch_2^2$. Equating the two expressions for I we then get

$$I_1 + ch_1^2 = I_2 + ch_2^2, \tag{5.26}$$

or

$$I_2 - I_1 = ch_1^2 - ch_2^2 = 3ch_2^2, \tag{5.27}$$

where we have made use of the fact that $h_1 = 2h_2$. Rearranging this expression now gives the error δ_2 on the second estimate of the integral:

$$\delta_2 = ch_2^2 = \tfrac{1}{3}(I_2 - I_1). \tag{5.28}$$

As we have written it, this expression can be either positive or negative, depending on which way the error happens to go. If we want only the absolute size of the error then we can take the absolute value $\frac{1}{3}|I_2 - I_1|$, which in Python would be done using the built-in function `abs`.

This method gives us a simple way to estimate the error on the trapezoidal rule without using the Euler–Maclaurin formula. Indeed, even in cases where we could in principle use the Euler–Maclaurin formula because we know the mathematical form of the integrand, it is often simpler to use the method of Eq. (5.28) instead—it is easy to program and gives reliable answers.

The same principle can be applied to integrals evaluated using Simpson's rule too. The equivalent of Eq. (5.28) in that case turns out to be

$$\delta_2 = \tfrac{1}{15}(I_2 - I_1). \tag{5.29}$$

The derivation is left to the reader (see Exercise 5.5).

Exercise 5.5: Error on Simpson's rule

Following the same line of argument that led to Eq. (5.28), show that the error on an integral evaluated using Simpson's rule is given, to leading order in h, by Eq. (5.29).

Exercise 5.6: Write a program, or modify an earlier one, to once more calculate the value of the integral $\int_0^2 (x^4 - 2x + 1)\,dx$ from Example 5.1, using the trapezoidal rule with 20 slices, but this time have the program also print an estimate of the error on the result, calculated using the method of Eq. (5.28). To do this you will need to evaluate the integral twice, once with $N_1 = 10$ slices and then again with $N_2 = 20$ slices. Then Eq. (5.28) gives the error. How does the error calculated in this manner compare with a direct computation of the error as the difference between your value for the integral and the known true value of 4.4? Why do the two not agree perfectly?

5.3 Choosing the number of steps

So far we have not specified how the number N of steps used in our integrals is to be chosen. In our example calculations we just chose round numbers and looked to see if the results seemed reasonable. This is fine for quick calculations, but for serious physics we want a more principled approach. In some calculations we may know in advance how many steps we want to use. Sometimes we have a "budget," a certain amount of computer time that we can spend on a calculation, and our goal is simply to make the most accurate calculation we can in the given amount of time. If we know, for instance, that we have time to do a thousand steps, then that's what we do.

But a more common situation is that we want to calculate the value of an integral to a given accuracy, such as four decimal places, and we would like to know how many steps will be needed. So long as the desired accuracy does not exceed the fundamental limit set by the machine precision of our computer—the rounding error that limits all calculations—then it should always be possible to meet our goal by using a large enough number of steps. At the same time, we want to avoid using more steps than are necessary, since more steps take more time and our calculation will be slower. Ideally we would like an N that gives us the accuracy we want and no more.

A simple way to achieve this is to start with a small value of N and repeatedly increase it until we achieve the accuracy we want. As we saw in Section 5.2.1, there is a simple formula, Eq. (5.28), for calculating the error on an integral when we double the number of steps. By using this formula with repeated doublings we can evaluate an integral to exactly the accuracy we want.

The procedure is straightforward. We start off by evaluating the integral with some small number of steps N_1. For instance, we might choose $N_1 = 10$. Then we double the number to $N_2 = 2N_1$, evaluate the integral again, and apply Eq. (5.28) to calculate the error. If the error is small enough to satisfy our accuracy requirements, then we're done—we have our answer. If not, we double again to $N_3 = 2N_2$ and we keep on doubling until we achieve the required accuracy. The error on the ith step of the process is given by the obvious generalization of Eq. (5.28):

$$\delta_i = \tfrac{1}{3}(I_i - I_{i-1}), \tag{5.30}$$

where I_i is the ith estimate of the integral. This method is an example of an *adaptive integration* method, one that varies its own parameters to get a desired answer.

A particularly nice feature of this method is that when we double the number of steps we do not actually have to recalculate the entire integral again. We can reuse our previous calculation rather than just throwing it away. To see this, take a look at Fig. 5.3. The top part of the figure depicts the locations of the sample points, the values of x at which the integrand is evaluated in the trapezoidal rule. The sample points are regularly spaced, and bear in mind that the first and last points are treated differently from the others—the trapezoidal rule formula, Eq. (5.3), specifies that the

values of $f(x)$ at these points are multiplied by a factor of $\frac{1}{2}$ where the values at the interior points are multiplied by 1.

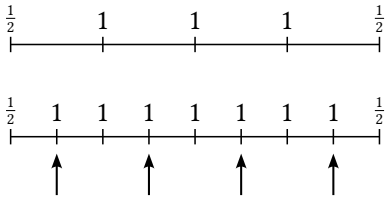


Figure 5.3: Doubling the number of steps in the trapezoidal rule. Top: we evaluate the integrand at evenly spaced points as shown, with the value at each point being multiplied by the appropriate factor. Bottom: when we double the number of steps, we effectively add a new set of points, half way between the previous points, as indicated by the arrows.

The lower part of the figure shows what happens when we double the number of slices. This adds an additional set of sample points half way between the old ones, as indicated by the arrows. Note that the original points are still included in the calculation and still carry the same multiplying factors as before—$\frac{1}{2}$ at the ends and 1 in the middle—while the new points are all multiplied by a simple factor of 1. Thus we have all of the same terms in our trapezoidal rule sum that we had before, terms that we have already evaluated, but we also have a set of new ones, which we have to add into the sum to calculate its full value. In the jargon of computational physics we say that the sample points for the first estimate of the integral are *nested* inside the points for the second estimate.

To put this in mathematical terms, consider the trapezoidal rule at the ith step of the calculation. Let the number of slices at this step be N_i and the width of a slice be $h_i = (b-a)/N_i$, and note that on the previous step there were half as many slices of twice the width, so that $N_{i-1} = \frac{1}{2}N_i$ and $h_{i-1} = 2h_i$. Then

$$\begin{aligned} I_i &= h_i\biggl[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N_i-1} f(a+kh_i)\biggr] \\ &= h_i\biggl[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{\substack{k\text{ even}\\ 2\ldots N_i-2}} f(a+kh_i) + \sum_{\substack{k\text{ odd}\\ 1\ldots N_i-1}} f(a+kh_i)\biggr]. \end{aligned} \tag{5.31}$$

But

$$\sum_{\substack{k\text{ even}\\ 2\ldots N_i-2}} f(a+kh_i) = \sum_{k=1}^{N_i/2-1} f(a+2kh_i) = \sum_{k=1}^{N_{i-1}-1} f(a+kh_{i-1}), \tag{5.32}$$

and hence

$$I_i = \tfrac{1}{2}h_{i-1}\biggl[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N_{i-1}-1} f(a+kh_{i-1})\biggr] + h_i \sum_{\substack{k\text{ odd}\\ 1\ldots N_i-1}} f(a+kh_i). \tag{5.33}$$

But the term $h_{i-1}[\ldots]$ in this equation is precisely the trapezoidal rule estimate I_{i-1} of the integral on the previous iteration of the process, so

$$I_i = \tfrac{1}{2}I_{i-1} + h_i \sum_{\substack{k\text{ odd}\\ 1\ldots N_i-1}} f(a+kh_i). \tag{5.34}$$

In effect, our old estimate gives us half of the terms in our trapezoidal rule sum and we only have to calculate the other half. In this way we avoid ever recalculating

any term that has already been calculated, meaning that each term is calculated only once, regardless of how many levels of the calculation it is used in. This means it takes only about as much work to calculate I_i going through all the successive levels $I_1, I_2, I_3, \ldots$ as it does to calculate I_i outright using the ordinary trapezoidal rule. Thus we pay very little extra price in terms of the running time of our program to use this adaptive method and we gain the significant advantage of a guarantee in the accuracy of the integral.

The entire process is as follows:

1. Choose an initial number of steps N_1 and decide on the target accuracy for the value of the integral. Calculate the first approximation I_1 to the integral using the chosen value of N_1 with the standard trapezoidal rule formula, Eq. (5.3).
2. Double the number of steps and use Eq. (5.34) to calculate an improved estimate of the integral. Also calculate the error on that estimate from Eq. (5.30).
3. If the absolute magnitude of the error is less than the target accuracy for the integral, stop. Otherwise repeat from step 2.

The sum over odd values of k in Eq. (5.34) can be conveniently performed in Python with a for loop of the form "`for k in range(1,N,2)`".

We can also derive a similar method for integrals evaluated using Simpson's rule. Again we double the number of steps on each iteration of the process and the equivalent of Eq. (5.30) is

$$\delta_i = \tfrac{1}{15}(I_i - I_{i-1}). \tag{5.35}$$

The equivalent of Eq. (5.34) is a little more complicated. We define

$$S_i = \tfrac{1}{3}\biggl[f(a) + f(b) + 2 \sum_{\substack{k \text{ even} \\ 2\ldots N_i-2}} f(a+kh_i)\biggr], \tag{5.36}$$

and

$$T_i = \tfrac{2}{3} \sum_{\substack{k \text{ odd} \\ 1\ldots N_i-1}} f(a+kh_i). \tag{5.37}$$

Then we can show that

$$S_i = S_{i-1} + T_{i-1}\,, \tag{5.38}$$

and

$$I_i = h_i(S_i + 2T_i). \tag{5.39}$$

Thus for Simpson's rule the complete process is:

1. Choose an initial number of steps and a target accuracy, and calculate the sums S_1 and T_1 from Eqs. (5.36) and (5.37) and the initial value I_1 of the integral from Eq. (5.39).
2. Double the number of steps then use Eqs. (5.37), (5.38), and (5.39) to calculate the new values of S_i and T_i and the new estimate of the integral. Also calculate the error on that estimate from Eq. (5.35).

3. If the absolute magnitude of the error is less than the target accuracy for the integral, stop. Otherwise repeat from step 2.

Again notice that on each iteration of the process you have to calculate only one sum, Eq. (5.37), which includes only those terms in the Simpson's rule formula that have not previously been calculated. As a result, the complete calculation of I_i takes very little more computer time than the basic Simpson rule.

5.4 Romberg integration

We can do even better than the adaptive method of the last section with only a little more effort. Let us go back to the trapezoidal rule again. We have seen that the leading-order error on the trapezoidal rule, at the ith step of the adaptive method, can be written as ch_i^2 for some constant c and is given by Eq. (5.30) to be

$$ch_i^2 = \tfrac{1}{3}(I_i - I_{i-1}). \tag{5.40}$$

But by definition the true value of the integral is $I = I_i + ch_i^2 + \mathrm{O}(h_i^4)$, where we are including the $\mathrm{O}(h_i^4)$ term to remind ourselves of the next term in the series—see Eq. (5.19). (Recall that there are only even-order terms in this series.) So in other words

$$I = I_i + \tfrac{1}{3}(I_i - I_{i-1}) + \mathrm{O}(h_i^4). \tag{5.41}$$

But this expression is now accurate to third order, and has a fourth-order error, which is as accurate as Simpson's rule,[2] and yet we calculated it using only our results from the trapezoidal rule, with hardly any extra work; we are just reusing numbers we already calculated while carrying out the repeated doubling procedure of Section 5.3.

---

[2] In fact, though it is not obvious, Eq. (5.41) is precisely equivalent to Simpson's rule. Using Eq. (5.3) for I_i and I_{i-1}, we have

$$\begin{aligned} I &= I_i + \tfrac{1}{3}(I_i - I_{i-1}) + \mathrm{O}(h_i^4) = \tfrac{4}{3}I_i - \tfrac{1}{3}I_{i-1} + \mathrm{O}(h_i^4) \\ &= \tfrac{4}{3}h_i\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N_i-1} f(a+kh_i)\right] - \tfrac{1}{3}h_{i-1}\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N_{i-1}-1} f(a+kh_{i-1})\right] + \mathrm{O}(h_i^4) \\ &= \tfrac{4}{3}h_i\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N_i-1} f(a+kh_i)\right] - \tfrac{2}{3}h_i\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{\frac{1}{2}N_i-1} f(a+2kh_i)\right] + \mathrm{O}(h_i^4), \end{aligned}$$

where we have made use of $h_{i-1} = 2h_i$ and $N_{i-1} = \frac{1}{2}N_i$ in the third line. Gathering terms and noting that the final sum is over only the even slices at locations $2kh_i$, we find that

$$I = \tfrac{1}{3}h_i\left[f(a) + f(b) + 4\sum_{\substack{k\ \text{odd} \\ 1\ldots N_i-1}} f(a+kh_i) + 2\sum_{\substack{k\ \text{even} \\ 2\ldots N_i-2}} f(a+kh_i)\right] + \mathrm{O}(h_i^4),$$

which is precisely Simpson's rule, Eq. (5.9). Similarly, the higher-order approximants of Eq. (5.51) are equivalent to the so-called Newton-Cotes rules, higher-order integration rules that we discuss in Section 5.5.

We can take this process further. Let us refine our notation a little and define

$$R_{i,1} = I_i\,, \qquad R_{i,2} = I_i + \tfrac{1}{3}(I_i - I_{i-1}) = R_{i,1} + \tfrac{1}{3}(R_{i,1} - R_{i-1,1}). \tag{5.42}$$

Then, from Eq. (5.41),

$$I = R_{i,2} + c_2 h_i^4 + \mathrm{O}(h_i^6), \tag{5.43}$$

where c_2 is another constant and we have made use of the fact that the series for I contains only even powers of h_i. Analogously,

$$I = R_{i-1,2} + c_2 h_{i-1}^4 + \mathrm{O}(h_{i-1}^6) = R_{i-1,2} + 16 c_2 h_i^4 + \mathrm{O}(h_i^6). \tag{5.44}$$

Since these last two equations both give expressions for I we can equate them and rearrange to get

$$c_2 h_i^4 = \tfrac{1}{15}(R_{i,2} - R_{i-1,2}) + \mathrm{O}(h_i^6). \tag{5.45}$$

Substituting this expression back into (5.43) gives

$$I = R_{i,2} + \tfrac{1}{15}(R_{i,2} - R_{i-1,2}) + \mathrm{O}(h_i^6). \tag{5.46}$$

Now we have an estimate accurate to fifth order, with a sixth-order error!

We can continue this process, computing higher and higher order error terms and getting more and more accurate results. In general, if $R_{i,m}$ is an estimate calculated at the ith round of the doubling procedure and accurate to order h^{2m-1}, with an error of order h^{2m}, then

$$I = R_{i,m} + c_m h_i^{2m} + \mathrm{O}\big(h_i^{2m+2}\big), \tag{5.47}$$

$$I = R_{i-1,m} + c_m h_{i-1}^{2m} + \mathrm{O}\big(h_{i-1}^{2m+2}\big) = R_{i-1,m} + 4^m c_m h_i^{2m} + \mathrm{O}\big(h_i^{2m+2}\big). \tag{5.48}$$

Equating the two and rearranging we have

$$c_m h_i^{2m} = \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m}) + \mathrm{O}\big(h_i^{2m+2}\big), \tag{5.49}$$

and substituting this into Eq. (5.47) gives

$$I = R_{i,m+1} + \mathrm{O}\big(h_i^{2m+2}\big), \tag{5.50}$$

where

$$R_{i,m+1} = R_{i,m} + \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m}), \tag{5.51}$$

which is accurate to order h^{2m+1} with an error of order h^{2m+2}.

The calculation also gives us an estimate of the error—Eq. (5.49) is precisely the error on $R_{i,m}$ (see Eq. (5.47))—and hence we can say how accurate our results are. To make use of these results in practice we do the following:

1. We calculate our first two estimates of the integral using the regular trapezoidal rule: $I_1 \equiv R_{1,1}$ and $I_2 \equiv R_{2,1}$.

2. From these we calculate the more accurate estimate $R_{2,2}$ using Eq. (5.51). This is as much as we can do with only the two starting estimates.
3. Now we calculate the next trapezoidal rule estimate $I_3 \equiv R_{3,1}$ and from this, with Eq. (5.51), we calculate $R_{3,2}$, and then $R_{3,3}$.
4. At each successive stage we compute one more trapezoidal rule estimate $I_i \equiv R_{i,1}$, and from it, with very little extra effort, we can calculate $R_{i,2} \ldots R_{i,i}$.
5. For each estimate we can also calculate the error, Eq. (5.49), which allows us to halt the calculation when the error on our estimate of the integral meets some desired target.

Perhaps a picture will help make the process clearer. This diagram shows which values $R_{i,m}$ are needed to calculate further Rs:

$$
\begin{array}{lllll}
I_1 \equiv R_{1,1} & & & & \\
\qquad\searrow & & & & \\
I_2 \equiv R_{2,1} \rightarrow & R_{2,2} & & & \\
\qquad\searrow & \qquad\searrow & & & \\
I_3 \equiv R_{3,1} \rightarrow & R_{3,2} \rightarrow & R_{3,3} & & \\
\qquad\searrow & \qquad\searrow & \qquad\searrow & & \\
I_4 \equiv R_{4,1} \rightarrow & R_{4,2} \rightarrow & R_{4,3} \rightarrow & R_{4,4} & \\
\qquad\searrow & \qquad\searrow & \qquad\searrow & \qquad\searrow &
\end{array}
$$

Each row here lists one trapezoidal rule estimate I_i followed by the other higher-order estimates it allows us to make. The arrows show which previous estimates go into the calculation of each new one via Eq. (5.51). Note how each fundamental trapezoidal rule estimate I_i allows us to go one step further with calculating the $R_{i,m}$. The most accurate estimate we get from the whole process is the very last one: if we do n levels of the process, then the last estimate is $R_{n,n}$ and is accurate to order h_n^{2n}.

Errors on our estimates are given by Eq. (5.49). If we are being picky, however, we should note that the equation gives us the error on every estimate *except* the last one in each row (which is the one we really care about). The equation says that the error on $R_{n,n}$ would be $(R_{n,n} - R_{n-1,n})/(4^n - 1)$ but there is no $R_{n-1,n}$ so we cannot use the formula in this case. In practice this means we have to content ourselves with the error estimate for the penultimate entry in each row, which is normally bigger than the error on the final entry. The best we can say is that the final entry in the row is our most accurate estimate of the integral and that its error is at least as good as the error for the entry that precedes it, which is given by Eq. (5.49). This is not ideal, but in practice it is usually good enough.

This whole procedure is called *Romberg integration*. It is essentially an "add-on" to our earlier trapezoidal rule scheme: all the tough work is done in the trapezoidal rule calculations and the Romberg integration takes almost no extra computer time (although it does involve extra programming). The payoff is a value for the integral that is accurate to much higher order in h than the simple trapezoidal rule (or even than Simpson's rule). And when used in an adaptive scheme that halts the calculation once the required accuracy is reached, it can significantly reduce the time needed to

evaluate integrals because it reduces the number of trapezoidal rule steps we have to do.

The method does have its limitations. We are in essence calculating the value of our integral by making a series expansion in powers of the step size h. This means that the method works best in cases where such power series converge rapidly. If one needs hundreds of terms in the series to get good convergence then the method is not going to give us any advantage over the simple trapezoidal rule. This can happen if the integrand $f(x)$ is poorly behaved, containing wild fluctuations, for instance, or singularities, or if it is noisy. If your integrand displays these types of pathologies then Romberg integration is not a good choice. The simpler adaptive trapezoidal method of Section 5.3 will give better results. In cases where the integrand is smooth and well-behaved, however, Romberg integration can give significantly more accurate results significantly faster than either the trapezoidal or Simpson rules.

Romberg integration is an example of the more general technique of *Richardson extrapolation*, in which high-order estimates of quantities are calculated iteratively from lower-order ones. We will see another application of Richardson extrapolation in Section 5.10.5, when we apply it to numerical differentiation.

Exercise 5.7: Consider the integral

$$I = \int_0^1 \sin^2 \sqrt{100x}\, \mathrm{d}x.$$

a) Write a program that uses the adaptive trapezoidal rule method of Section 5.3 and Eq. (5.34) to calculate the value of this integral to an approximate accuracy of $\delta = 10^{-6}$ (i.e., correct to six digits after the decimal point). Start with one single integration slice and work up from there to two, four, eight, and so forth. Have your program print out the number of slices, its estimate of the integral, and its estimate of the error on the integral, for each value of the number of slices N, until the target accuracy is reached. (Hint: You should find the result is around $I = 0.45$.)

b) Now modify your program to evaluate the same integral using the Romberg integration technique described in this section. Have your program print out a triangular table of values, as on page 152, of all the Romberg estimates of the integral. Calculate the error on your estimates using Eq. (5.49) and again continue the calculation until you reach an accuracy of $\delta = 10^{-6}$. You should find that the Romberg method reaches the required accuracy considerably faster than the trapezoidal rule alone.

Exercise 5.8: Write a program that uses the adaptive Simpson's rule method of Section 5.3 and Eqs. (5.35) to (5.39) to calculate the same integral as in Exercise 5.7, again to an approximate accuracy of $\delta = 10^{-6}$. Starting this time with two integration slices, work up from there to four, eight, and so forth, printing out the results at each step until the required accuracy is reached. You should find you reach that accuracy for a significantly smaller number of slices than with the trapezoidal rule calculation in part (a) of Exercise 5.7, but a somewhat larger number than with the Romberg integration of part (b).

5.5 HIGHER-ORDER INTEGRATION METHODS

As we have seen, the trapezoidal rule is based on approximating an integrand $f(x)$ with straight-line segments, while Simpson's rule uses quadratics. We can create higher-order (and hence potentially more accurate) rules by using higher-order polynomials, fitting $f(x)$ with cubics, quartics, and so forth. The general form of the trapezoidal and Simpson rules is

$$\int_a^b f(x)\,\mathrm{d}x \simeq \sum_{k=1}^{N} w_k f(x_k), \tag{5.52}$$

where the x_k are the positions of the sample points at which we calculate the integrand and the w_k are some set of weights. In the trapezoidal rule, Eq. (5.3), the first and last weights are $\frac{1}{2}$ and the others are all 1, while in Simpson's rule the weights are $\frac{1}{3}$ for the first and last slices and alternate between $\frac{4}{3}$ and $\frac{2}{3}$ for the other slices—see Eq. (5.9).

For higher-order rules the basic form is the same: after fitting to the appropriate polynomial and integrating we end up with a set of weights that multiply the values $f(x_k)$ of the integrand at evenly spaced sample points. Here are the weights up to quartic order:

Degree	Polynomial	Coefficients
1 (trapezoidal rule)	Straight line	$\frac{1}{2}, 1, 1, \ldots, 1, \frac{1}{2}$
2 (Simpson's rule)	Quadratic	$\frac{1}{3}, \frac{4}{3}, \frac{2}{3}, \frac{4}{3}, \ldots, \frac{4}{3}, \frac{1}{3}$
3	Cubic	$\frac{3}{8}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \ldots, \frac{9}{8}, \frac{3}{8}$
4	Quartic	$\frac{14}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \frac{28}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \ldots, \frac{64}{45}, \frac{14}{45}$

Higher-order integration rules of this kind are called *Newton–Cotes formulas* and in principle they can be extended to any order we like.

However, we can do better still. We note that the trapezoidal rule is *exact* if the function being integrated is actually a straight line, because then the straight-line approximation isn't an approximation at all. Similarly, Simpson's rule is exact if the function being integrated is a quadratic, and the kth Newton–Cotes rule is exact if the function being integrated is a degree-k polynomial.

But if we have N sample points, then presumably that means we could just fit one $(N-1)$th-degree polynomial to the whole integration interval, and get an integration method that is exact for $(N-1)$th-degree polynomials—and for any lower-degree polynomials as well. (Note that it's $N-1$ because you need three points to fit a quadratic, four for a cubic, and so forth.)

But we can do even better than this. We have been assuming here that the sample points are evenly spaced. Methods with evenly spaced points are simple to program, and they have the advantage that it is easy to increase the number of points by adding new points half way between the old ones, as we saw in Section 5.3. However, it is also possible to derive integration methods that use unevenly spaced points and,

while they lack some of the advantages above, they have others of their own. In particular, they can give very accurate answers with only a small number of points, making them especially suitable for cases where we need to do integrals very fast, or where evaluation of the integrand itself takes a long time.

Suppose then that we broaden our outlook to include rules of the form of Eq. (5.52), but where we are allowed to vary not only the weights w_k but also the positions x_k of the sample points. Any choice of positions is allowed, including ones that are not evenly spaced. As we have said, it is possible to create an integration method that is exact for polynomials up to degree $N-1$ with N equally spaced points. Varying the positions of the points gives us N extra degrees of freedom, which suggests that it might be possible to create an integration rule that is exact for polynomials up to degree $2N-1$ if all of those degrees of freedom are chosen correctly. For large values of N this could give us the power to fit functions very accurately indeed, and hence to do very accurate integrals. It turns out that it is indeed possible to do this and the developments lead to the superbly accurate integration method known as *Gaussian quadrature*, which we describe in the next section.

5.6 Gaussian quadrature

The derivation of the Gaussian quadrature method has two parts. First, we will see how to derive integration rules with unevenly spaced sample points x_k. Then we will choose the particular set of points that give the optimal integration rule.

5.6.1 Nonuniform sample points

Suppose we are given a nonuniform set of N points x_k and we wish to create an integration rule of the form (5.52) that calculates integrals over a given interval from a to b, based only on the values $f(x_k)$ of the integrand at those points. In other words, we want to choose weights w_k so that Eq. (5.52) works for general $f(x)$. To do this, we will fit a single polynomial through the values $f(x_k)$ and then integrate that polynomial from a to b to calculate an approximation to the true integral. To fit N points we need to use a polynomial of degree $N-1$. The fitting can be done using the *method of interpolating polynomials*.

Consider the following quantity:

$$\begin{aligned}\phi_k(x) &= \prod_{\substack{m=1\ldots N\\ m\neq k}} \frac{(x-x_m)}{(x_k-x_m)} \\ &= \frac{(x-x_1)}{(x_k-x_1)} \times \ldots \times \frac{(x-x_{k-1})}{(x_k-x_{k-1})} \times \frac{(x-x_{k+1})}{(x_k-x_{k+1})} \times \ldots \times \frac{(x-x_N)}{(x_k-x_N)},\end{aligned} \tag{5.53}$$

which is called a *Lagrange interpolating polynomial*. Note that the numerator contains one factor for each sample point except the point x_k. Thus $\phi_k(x)$ is a polynomial

in x of degree $N-1$. For values of k from 1 to N, Eq. (5.53) defines N different such polynomials.

You can confirm for yourself that if we evaluate $\phi_k(x)$ at one of the sample points $x = x_m$ we get

$$\phi_k(x_m) = \begin{cases} 1 & \text{if } m = k, \\ 0 & \text{if } m \neq k, \end{cases} \tag{5.54}$$

or, to be more concise,

$$\phi_k(x_m) = \delta_{km}\,, \tag{5.55}$$

where δ_{km} is the Kronecker delta—the quantity that is 1 when $k = m$ and zero otherwise.

Now consider the following expression:

$$\Phi(x) = \sum_{k=1}^{N} f(x_k)\,\phi_k(x). \tag{5.56}$$

Since it is a linear combination of polynomials of degree $N-1$, this entire quantity is also a polynomial of degree $N-1$. And if we evaluate it at any one of the sample points $x = x_m$ we get

$$\Phi(x_m) = \sum_{k=1}^{N} f(x_k)\,\phi_k(x_m) = \sum_{k=1}^{N} f(x_k)\,\delta_{km} = f(x_m), \tag{5.57}$$

where we have used Eq. (5.55).

In other words $\Phi(x)$ is a polynomial of degree $N-1$ that fits the integrand $f(x)$ at all of the sample points. This is exactly the quantity we were looking for to create our integration rule. Moreover, the polynomial of degree $N-1$ that fits a given N points is unique: it has N free coefficients and our points give us N constraints, so the coefficients are completely determined. Hence $\Phi(x)$ is not merely *a* polynomial that fits our points, it is *the* polynomial. There are no others.

To calculate an approximation to our integral, all we have to do now is integrate $\Phi(x)$ from a to b thus:

$$\begin{aligned} \int_a^b f(x)\,\mathrm{d}x \simeq \int_a^b \Phi(x)\,\mathrm{d}x &= \int_a^b \sum_{k=1}^{N} f(x_k)\phi_k(x)\,\mathrm{d}x \\ &= \sum_{k=1}^{N} f(x_k) \int_a^b \phi_k(x)\,\mathrm{d}x, \end{aligned} \tag{5.58}$$

where we have interchanged the order of the sum and integral in the second line. Comparing this expression with Eq. (5.52) we now see that the weights we need for our integration rule are given by

$$w_k = \int_a^b \phi_k(x)\,\mathrm{d}x. \tag{5.59}$$

In other words we have found a general method for creating an integration rule of the form (5.52) for any set of sample points x_k: we set the weights w_k equal to the integrals of the interpolating polynomials, Eq. (5.53), over the integration domain.

There is no general closed-form formula for the integrals of the interpolating polynomials. In some special cases it is possible to perform the integrals exactly, but often it is not, in which case we may have to perform them on the computer, using one of our other integration methods, such as Simpson's rule or Romberg integration. This may seem to defeat the point of our calculation, which was to find an integration method that did not rely on uniformly spaced sample points, and here we are using Simpson's rule, which has uniformly spaced points. But in fact the exercise is not as self-defeating as it may appear. The point to notice is that we have to calculate the weights w_k only once, and then we can use them in Eq. (5.52) to integrate as many different functions as we like. So we may have to put some effort into the calculation of the weights, using, say, Simpson's rule with very many slices to get as accurate an answer as possible. But we only have to do it once, and thereafter other integrals can be done rapidly and accurately using Eq. (5.52).

In fact, it's better than this. Once one has calculated the weights for a particular set of sample points and domain of integration, it is possible to map those weights and points onto any other domain and get an integration rule of the form (5.52) without having to recalculate the weights. Typically one gives sample points and weights arranged in a standard interval, which for historical reasons is usually taken to be the interval from $x = -1$ to $x = +1$. Thus to specify an integration rule one gives a set of sample points in the range $-1 \le x_k \le 1$ and a set of weights

$$w_k = \int_{-1}^{1} \phi_k(x)\, \mathrm{d}x. \tag{5.60}$$

If we want to integrate over any domain other than the one from -1 to $+1$, we map these values to that other domain. Since the area under a curve does not depend on where that curve is along the x line, the sample points can be slid up and down the x line *en masse* and the integration rule will still work fine. If the desired domain is wider or narrower than the interval from -1 to $+1$ then we need to spread the points out or squeeze them together. The correct rule for mapping the points to a general domain that runs from $x = a$ to $x = b$ is:

$$x'_k = \tfrac{1}{2}(b-a)x_k + \tfrac{1}{2}(b+a). \tag{5.61}$$

Similarly the weights do not change if we are simply sliding the sample points up or down the x line, but if the width of the integration domain changes then the value of the integral will increase or decrease by a corresponding factor, and hence the weights have to be rescaled thus:

$$w'_k = \tfrac{1}{2}(b-a)w_k. \tag{5.62}$$

Once we have calculated the rescaled positions and weights then the integral itself is given by

$$\int_a^b f(x)\,\mathrm{d}x \simeq \sum_{k=1}^{n} w'_k f(x'_k). \tag{5.63}$$

5.6.2 Sample points for Gaussian quadrature

The developments of the previous section solve part of our problem. Given the positions of the sample points x_k they tell us how to choose the weights w_k, but we still need to choose the sample points. As we speculated at the end of Section 5.5, it is possible to chose these points so as to create a rule that gives the exact integral of any polynomial function up to degree $2N-1$ with just N sample points. The derivation is based on the mathematics of Legendre polynomials.

The Legendre polynomial $P_N(x)$ is an Nth-degree polynomial in x that has the property

$$\int_{-1}^{1} x^k P_N(x)\,\mathrm{d}x = 0 \qquad \text{for all integer } k \text{ in the range } 0 \le k < N \tag{5.64}$$

and satisfies the normalization condition

$$P_N(1) = 1. \tag{5.65}$$

Thus, for instance, $P_0(x) =$ constant, and the constant is fixed by (5.65) to give $P_0(x) = 1$. Similarly, $P_1(x)$ is a linear function $ax + b$ satisfying

$$\int_{-1}^{1} (ax+b)\,\mathrm{d}x = 0. \tag{5.66}$$

Carrying out the integral, we find that $b = 0$ and a is fixed by (5.65) to be 1, giving $P_1(x) = x$. By similar arguments we can derive expressions for as many polynomials as we like. The next two are $P_2(x) = \frac{1}{2}(3x^2-1)$ and $P_3(x) = \frac{1}{2}(5x^3-3x)$, and you can find tables online or elsewhere that list them to higher order.

Now suppose that $q(x)$ is any polynomial of degree less than N, so that it can be written $q(x) = \sum_{k=0}^{N-1} c_k x^k$ for some set of coefficients c_k. Then

$$\int_{-1}^{1} q(x)P_N(x)\,\mathrm{d}x = \sum_{k=0}^{N-1} c_k \int_{-1}^{1} x^k P_N(x)\,\mathrm{d}x = 0, \tag{5.67}$$

by Eq. (5.64). Thus, for any N, $P_N(x)$ is orthogonal to every polynomial of lower degree. A further property of the Legendre polynomials, which we will use shortly, is that for all N the polynomial $P_N(x)$ has N real roots that all lie in the interval from -1 to 1. That is, there are N values of x in this interval for which $P_N(x) = 0$.

We make use of the Legendre polynomials as follows. Our goal is to find an integration rule of the form

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \simeq \sum_{k=1}^{N} w_k f(x_k). \tag{5.68}$$

(As we saw in the previous section, if we can find sample points and weights for an integral like this over the standard interval from −1 to 1 then we can do integrals over any other interval by a simple change of variables, Eqs. (5.61) and (5.62).) Suppose that the integrand $f(x)$ is a polynomial in x of degree $2N-1$ or less. If we divide $f(x)$ by the Legendre polynomial $P_N(x)$, then we get

$$f(x) = q(x)P_N(x) + r(x), \tag{5.69}$$

where $q(x)$ and $r(x)$ are both polynomials of degree $N-1$ or less. Thus our integral can be written

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \int_{-1}^{1} q(x)P_N(x)\,\mathrm{d}x + \int_{-1}^{1} r(x)\,\mathrm{d}x = \int_{-1}^{1} r(x)\,\mathrm{d}x, \tag{5.70}$$

where the term in $q(x)$ vanishes because of (5.67). This means that to find the integral of the polynomial $f(x)$ we have only to find the integral of the polynomial $r(x)$, which always has degree $N-1$ or less.

But we already know how to solve this problem. As we saw in Section 5.6.1, for any choice of the N sample points x_k, a polynomial of degree $N-1$ or less can be fitted exactly using the interpolating polynomials $\phi_k(x)$, Eq. (5.53), and then the fit can be integrated to give

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \int_{-1}^{1} r(x)\,\mathrm{d}x = \sum_{k=1}^{N} w_k r(x_k), \tag{5.71}$$

where

$$w_k = \int_{-1}^{1} \phi_k(x)\,\mathrm{d}x. \tag{5.72}$$

(See Eq. (5.60) on page 157.) Note that, unlike Eq. (5.68), the equality in Eq. (5.71) is now an exact one, because the polynomial fit is exact.

Thus we have a method for integrating any polynomial of degree $2N-1$ or less exactly over the interval from −1 to 1: we divide by the Legendre polynomial $P_N(x)$ and then integrate the remainder polynomial $r(x)$ using any set of N sample points we choose plus the corresponding weights.

This, however, is not a very satisfactory method. In particular the polynomial division is rather complicated to perform. However, we can simplify the procedure by noting that, so far, the positions of our sample points are unconstrained and we can pick them in any way we please. So consider again an integration rule of the form (5.68) and make the substitution (5.69), to get

$$\sum_{k=1}^{N} w_k f(x_k) = \sum_{k=1}^{N} w_k q(x_k) P_N(x_k) + \sum_{k=1}^{N} w_k r(x_k). \tag{5.73}$$

But we know that $P_N(x)$ has N zeros between −1 and 1, so let us choose our N sample points x_k to be exactly the positions of these zeros. That is, let x_k be the kth root of

the Legendre polynomial $P_N(x)$. In that case, $P_N(x_k) = 0$ for all k and Eq. (5.73) becomes simply

$$\sum_{k=1}^{N} w_k f(x_k) = \sum_{k=1}^{N} w_k r(x_k). \tag{5.74}$$

Combining with Eq. (5.71), we then have

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \sum_{k=1}^{N} w_k f(x_k), \tag{5.75}$$

where the equality is exact. Now we have a integration rule of the standard form that allows us to integrate any polynomial function $f(x)$ of degree $2N - 1$ or less from -1 to 1 and get an exact answer (except for rounding error).

Let us pause for a moment to take that in: this integration rule will give the exact value for the integral, even though we only measure the function at N points. Upon reflection, this is a remarkable result. If we measure a polynomial of any degree greater than $N - 1$ at only N points then we do not have enough information to actually reconstruct the polynomial—a polynomial of degree N or greater has at least $N+1$ coefficients, so N measurements are not enough to fix all the degrees of freedom. Nonetheless, even though we cannot tell what the polynomial is, we *can* calculate its integral.

The calculation of the positions of the zeros of $P_N(x)$ does take some work. For any $n \geq 1$ the Legendre polynomials are known to satisfy the recurrence relation

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x), \tag{5.76}$$

and starting from $P_0(x) = 1$ and $P_1(x) = x$ we can use this formula to calculate $P_2(x)$, then $P_3(x)$, $P_4(x)$, and so on until we reach the desired $P_N(x)$. This gives us the value of $P_N(x)$ for any x exactly, apart from rounding error. Then the zeros can be found using, for instance, Newton's method. (If you are not familiar with Newton's method, we study it in detail in the next chapter, in Section 6.3.5.)

5.6.3 Weights for Gaussian quadrature

Once we have the sample points, the final step of our derivation is to calculate the weights w_k for our integration rule from Eq. (5.60) on page 157, which says that $w_k = \int_{-1}^{1} \phi_k(x)\,\mathrm{d}x$. Recall that the Lagrange interpolating polynomial $\phi_k(x)$ is given by Eq. (5.53) to be

$$\phi_k(x) = \prod_{\substack{m=1\ldots N \\ m\neq k}} \frac{(x - x_m)}{(x_k - x_m)}. \tag{5.77}$$

Since we have chosen the x_m to be the roots of the polynomial $P_N(x)$, we have, by definition,

$$P_N(x) = A \prod_{m=1}^{N} (x - x_m), \tag{5.78}$$

where A is a constant equal to the coefficient of x^N in P_N. If we define

$$\psi_k(x) = A \prod_{\substack{m=1\ldots N \\ m\neq k}} (x - x_m), \tag{5.79}$$

then $P_N(x) = (x - x_k)\psi_k(x)$ and (5.77) becomes

$$\phi_k(x) = \frac{\psi_k(x)}{\psi_k(x_k)} = \frac{P_N(x)/(x - x_k)}{P'_N(x_k)}, \tag{5.80}$$

where $P'_N(x)$ is the derivative of $P_N(x)$ and we have made use of

$$\psi_k(x_k) = \lim_{x\to x_k} \frac{P_N(x)}{x - x_k} = P'_N(x_k) \tag{5.81}$$

by l'Hopital's rule. Thus the weight w_k is

$$w_k = \int_{-1}^{1} \phi_k(x)\,\mathrm{d}x = \frac{1}{P'_N(x_k)} \int_{-1}^{1} \frac{P_N(x)}{x - x_k}\,\mathrm{d}x. \tag{5.82}$$

The value of the remaining integral can be calculated from the recurrence relation in Eq. (5.76). Multiplying this relation throughout by $P_n(y)$ for any y we have

$$(n+1)P_{n+1}(x)P_n(y) = (2n+1)xP_n(x)P_n(y) - nP_{n-1}(x)P_n(y). \tag{5.83}$$

Subtracting this equation and the same equation with x and y exchanged, we get

$$\begin{aligned}(n+1)[P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)] = (2n+1)(x-y)P_n(x)P_n(y) + \\ + n[P_n(x)P_{n-1}(y) - P_{n-1}(x)P_n(y)],\end{aligned} \tag{5.84}$$

and summing over n from 1 to $N-1$ and rearranging, we get

$$\begin{aligned}(x-y)\sum_{n=1}^{N-1}(2n+1)P_n(x)P_n(y) &= \sum_{n=1}^{N-1}(n+1)[P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)] \\ &\quad - \sum_{n=1}^{N-1} n[P_n(x)P_{n-1}(y) - P_{n-1}(x)P_n(y)] \\ &= N[P_N(x)P_{N-1}(y) - P_{N-1}(x)P_N(y)] - (x-y),\end{aligned} \tag{5.85}$$

with all the other terms on the right having canceled. Rearranging again, we arrive at the *Christoffel-Darboux formula* for the Legendre polynomials:

$$\sum_{n=0}^{N-1}(2n+1)P_n(x)P_n(y) = N\frac{P_N(x)P_{N-1}(y) - P_{N-1}(x)P_N(y)}{x-y}. \tag{5.86}$$

Note how the term $x - y$ on the right-hand side of (5.85) has been absorbed into the $n = 0$ term in the sum on the left, exploiting the fact that $P_0(x) = P_0(y) = 1$.

Setting $y = x_k$ and noting that $P_N(x_k) = 0$ because x_k is a root, Eq. (5.86) becomes

$$\sum_{n=0}^{N-1}(2n+1)P_n(x)P_n(x_k) = N\frac{P_N(x)P_{N-1}(x_k)}{x - x_k}. \tag{5.87}$$

Now we integrate both sides with respect to x from -1 to 1, noting that, because $P_0(x) = 1$ and $P_n(x)$ for $n \geq 1$ is orthogonal to $P_0(x)$, all terms in the sum on the left vanish except for the $n = 0$ term, and we get

$$\begin{aligned}\int_{-1}^{1}\sum_{n=0}^{N-1}(2n+1)P_n(x)P_n(x_k)\,\mathrm{d}x &= P_0(x_k)\int_{-1}^{1}P_0(x)\,\mathrm{d}x = 2\\ &= NP_{N-1}(x_k)\int_{-1}^{1}\frac{P_N(x)}{x - x_k}\,\mathrm{d}x,\end{aligned} \tag{5.88}$$

or

$$\int_{-1}^{1}\frac{P_N(x)}{x - x_k}\,\mathrm{d}x = \frac{2}{NP_{N-1}(x_k)}. \tag{5.89}$$

Substituting this result into Eq. (5.82), we arrive at an expression for the integration weights:

$$w_k = \frac{2}{NP'_N(x_k)P_{N-1}(x_k)}. \tag{5.90}$$

Finally, we can if we wish simplify this expression by making use of another standard recurrence relation for the Legendre polynomials:

$$(x^2 - 1)P'_N(x) = N[xP_N(x) - P_{N-1}(x)]. \tag{5.91}$$

Setting $x = x_k$ and noting again that $P_N(x_k) = 0$, we get

$$P'_N(x_k) = \frac{NP_{N-1}(x_k)}{1 - x_k^2}, \tag{5.92}$$

and combining this result with (5.90), we derive two further forms for w_k thus:

$$w_k = \frac{2}{(1 - x_k^2)[P'_N(x_k)]^2}, \tag{5.93}$$

$$w_k = \frac{2(1 - x_k^2)}{[NP_{N-1}(x_k)]^2}. \tag{5.94}$$

The first of these is the form one sees most often in books and online, but we find the second one more convenient in practice. Since one calculates the Legendre polynomials from the recurrence relation (5.76) starting from $P_0(x)$ and working up, one always calculates $P_{N-1}(x)$ en route to calculating $P_N(x)$, meaning we can evaluate (5.94) easily using results we already have at hand after finding the roots of $P_N(x)$.

Figure 5.4 shows what our sample points and weights look like for the cases $N = 10$ and $N = 100$. Note how the points get closer together at the edges while at the same time the weights get smaller.

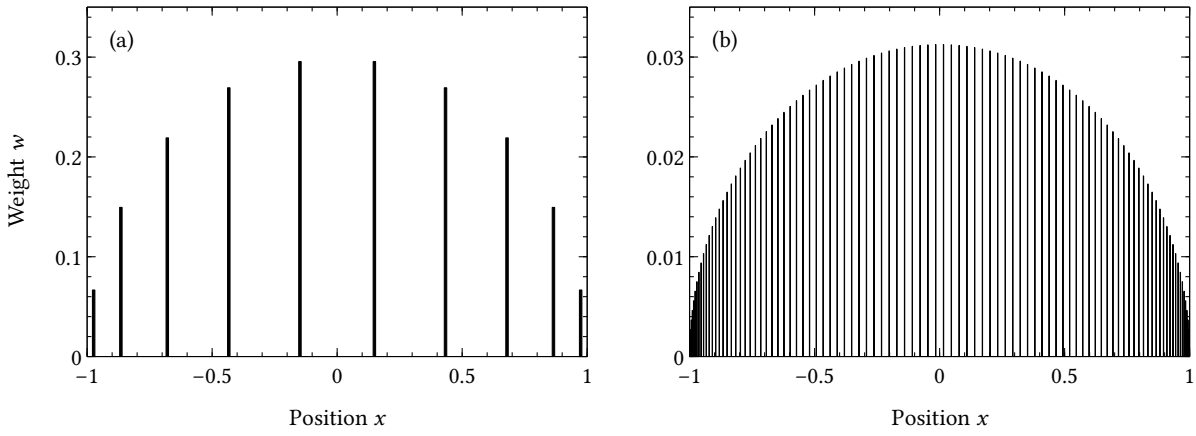


Figure 5.4: Sample points and weights for Gaussian quadrature. The positions and heights of the bars represent the sample points and their associated weights for Gaussian quadrature with (a) $N = 10$ and (b) $N = 100$.

Combining everything we have learned, our final integration rule, based on the sample points x_k and weights w_k, is called *Gaussian quadrature*[3] and although its derivation is quite complicated, using it in practice is beautifully simple: given the values x_k and w_k for your chosen N, you can integrate any function over the interval -1 to 1 by simply performing the sum in Eq. (5.68). For integrals over any other interval, you rescale the values using Eqs. (5.61) and (5.62) and then perform the sum in Eq. (5.63). Note that, although we have derived the method by considering integrals of polynomial functions, its use is not restricted to polynomials. It can be used to integrate functions of any kind. It does not give exact answers for non-polynomial functions, but it is highly accurate nonetheless, even when using a relatively small number of sample points.

The only difficult part of the method is finding the values of x_k and w_k in the first place. As described above, we can calculate them using the recurrence relation (5.76) plus Newton's method, combined with the formula for the weights, Eq. (5.94). The calculation is not complicated, but getting it right does require some care. You are welcome to give it a try, but we also provide Python functions to do it in Appendix C and in the online resources in the file `gaussxw.py`. We will make use of these functions extensively in this book. Example 5.2 below shows how to use them.

[3]It is called "Gaussian" because it was pioneered by the legendary mathematician Carl Friedrich Gauss. "Quadrature" is an old (19th century) name for numerical integration—Gauss's work predates the invention of computers, to a time when people did numerical integrals by hand, meaning they were *very* concerned about getting the best answers when N is small. When you are doing calculations by hand, Simpson's rule with $N = 1000$ is not an option.

EXAMPLE 5.2: GAUSSIAN INTEGRAL OF A SIMPLE FUNCTION

Consider again the integral we did in Example 5.1, $\int_0^2 (x^4-2x+1)\,dx$, whose true value, as we saw, is 4.4. Here is a program to evaluate the same integral using Gaussian quadrature. Just to emphasize the impressive power of the method, we will perform the calculation with only $N = 3$ sample points:

File: gaussint.py

```
from gaussxw import gaussxw

def f(x):
    return x**4 - 2*x + 1

N = 3
a = 0.0
b = 2.0

# Calculate the sample points and weights
x,w = gaussxw(N,a,b)

# Perform the integration
s = 0.0
for k in range(N):
    s += w[k]*f(x[k])

print(s)
```

For this program to work you must have a copy of the file `gaussxw.py` in the same folder as the program itself.

The function `gaussxw` takes three arguments, which are the value of N plus the limits a and b of the integration interval. The latter two are optional—if they are omitted the function assumes the standard interval where $a = -1$ and $b = 1$. Note also how the function returns two variables, not just one. We discussed functions of this type in Section 2.6 but this is the first time we have seen one in use. In this case the variables are arrays, `x` and `w`, containing the sample points and weights for Gaussian quadrature on N points over the interval from a to b.

The program above is very simple—no more complicated than the program for the trapezoidal rule in Example 5.1—yet when we run it, it prints the following:

```
4.4
```

The program has calculated the answer exactly, with just three sample points. This is not a mistake, or luck, or a coincidence. It's exactly what we expect. Gaussian integration on N points gives exact answers for the integrals of polynomial functions up to and including polynomials of degree $2N-1$, which for $N = 3$ means degree five. The function $x^4 - 2x + 1$ that we are integrating here is a degree-four polynomial, so we expect the method to return an exact answer, and it does. Nonetheless, the

performance of the method does seem almost magical in this case: the program has evaluated the integrand at just three points and from those three values alone it is, amazingly, able to calculate the integral of the entire function exactly.

This is the strength of Gaussian quadrature: it can give remarkably accurate answers, even with small numbers of sample points. This makes it especially useful in situations where you cannot afford to use large numbers of points, either because you need to be able to calculate an answer very quickly or because evaluating your integrand takes a long time even for just a few points.

The method does have its disadvantages. In particular, because the sample points are not uniformly distributed it takes more work if we want to employ the trick of repeatedly doubling N, as we did in Section 5.3, to successively improve the accuracy of the integral—if we change the value of N then all the sample points and weights have to be recalculated, and the entire sum over points, Eq. (5.52), has to be redone. We cannot reuse the calculations for old sample points as we did with the trapezoidal rule. In the language of computational physics we say that the sample points are not nested.

Exercise 5.9: Heat capacity of a solid

Debye's theory of solids gives the heat capacity of a solid at temperature T to be

$$C_V = 9V\rho k_B \left(\frac{T}{\theta_D}\right)^3 \int_0^{\theta_D/T} \frac{x^4 e^x}{(e^x - 1)^2}\, dx,$$

where V is the volume of the solid, ρ is the number density of atoms, k_B is Boltzmann's constant, and θ_D is the so-called *Debye temperature*, a property of solids that depends on their density and speed of sound.

a) Write a Python function `cv(T)` that calculates C_V for a given value of the temperature, for a sample consisting of 1000 cubic centimeters of solid aluminum, which has a number density of $\rho = 6.022 \times 10^{28}\,\mathrm{m}^{-3}$ and a Debye temperature of $\theta_D = 428\,\mathrm{K}$. Use Gaussian quadrature to evaluate the integral, with $N = 50$ sample points.

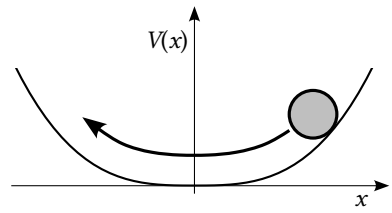
b) Use your function to make a graph of the heat capacity as a function of temperature from $T = 5\,\mathrm{K}$ to $T = 500\,\mathrm{K}$.

Exercise 5.10: Period of an anharmonic oscillator

The simple harmonic oscillator crops up in many places. Its behavior can be studied readily using analytic methods and it has the important property that its period is a constant, independent of the amplitude of oscillation, making it useful, for instance, for keeping time in watches and clocks.

Frequently in physics, however, we also come across anharmonic oscillators, whose period varies with amplitude and whose behavior cannot usually be calculated analytically. A general

classical oscillator can be thought of as a particle in a concave potential well. When disturbed, the particle will rock back and forth in the well:



The harmonic oscillator corresponds to a quadratic potential $V(x) \propto x^2$. Any other form gives an anharmonic oscillator. (Thus there are many different kinds of anharmonic oscillator, depending on the exact form of the potential.)

One way to calculate the motion of an oscillator is to write down the equation for the conservation of energy in the system. If the particle has mass m and position x, then the total energy is equal to the sum of the kinetic and potential energies thus:

$$E = \tfrac{1}{2}m\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + V(x).$$

Since the energy must be constant over time, this equation is effectively a (nonlinear) differential equation linking x and t.

Let us assume that the potential $V(x)$ is symmetric about $x = 0$ and let us set our anharmonic oscillator going with amplitude a. That is, at $t = 0$ we release it from rest at position $x = a$ and it swings back towards the origin. Thus at $t = 0$ we have $\mathrm{d}x/\mathrm{d}t = 0$ and the equation above reads $E = V(a)$, which gives us the total energy of the particle in terms of the amplitude.

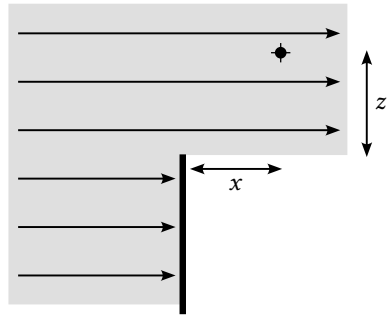
a) When the particle reaches the origin for the first time, it has gone through one quarter of a period of the oscillator. By rearranging the equation above for $\mathrm{d}x/\mathrm{d}t$ and then integrating with respect to t from 0 to $\frac{1}{4}T$, show that the period T is given by

$$T = \sqrt{8m}\int_0^a \frac{\mathrm{d}x}{\sqrt{V(a) - V(x)}}.$$

b) Suppose the potential is $V(x) = x^4$ and the mass of the particle is $m = 1$. Write a Python function that calculates the period of the oscillator for given amplitude a using Gaussian quadrature with $N = 20$ points, then use your function to make a graph of the period for amplitudes ranging from $a = 0$ to $a = 2$.

c) You should find that the oscillator gets faster as the amplitude increases, even though the particle has further to travel for larger amplitude. And you should find that the period diverges as the amplitude goes to zero. How do you explain these results?

Exercise 5.11: Suppose a plane wave of wavelength λ, such as light or a sound wave, is blocked by an object with a straight edge, represented by the solid bar at the bottom of this figure:



The wave will be diffracted at the edge and the resulting intensity at the position (x, z) marked by the dot is given by near-field diffraction theory to be

$$I = \frac{I_0}{8}\bigl([2C(u)+1]^2 + [2S(u)+1]^2\bigr),$$

where I_0 is the intensity of the wave before diffraction and

$$u = x\sqrt{\frac{2}{\lambda z}}, \qquad C(u) = \int_0^u \cos\bigl(\tfrac{1}{2}\pi t^2\bigr)\,\mathrm{d}t, \qquad S(u) = \int_0^u \sin\bigl(\tfrac{1}{2}\pi t^2\bigr)\,\mathrm{d}t.$$

Write a program to calculate I/I_0 and make a plot of it as a function of x in the range -5 m to 5 m for the case of a sound wave with wavelength $\lambda = 1$ m, measured $z = 3$ m past the straight edge. Calculate the integrals using Gaussian quadrature with $N = 50$ points. You should find significant variation in the intensity of the diffracted sound—enough that you could easily hear the effect if sound were diffracted, say, at the edge of a tall building.

5.6.4 Errors on Gaussian quadrature

In our study of the trapezoidal rule we derived an expression, the Euler–Maclaurin formula of Eq. (5.20), for the approximation error on the value of an integral. There exists a corresponding expression for Gaussian quadrature but it is, unfortunately, ungainly and not easy to use in practice. What it does tell us, however, is that Gaussian quadrature is impressively accurate. Roughly speaking, the approximation error—the difference between the value of an integral calculated using Gaussian quadrature and the true value of the same integral, neglecting rounding error—improves by a factor of c/N^2 when we increase the number of samples by *just one*, where c is a constant whose value depends on the detailed shape of the integrand and the size of the domain of integration. Thus, for instance, if we go from $N = 10$ to $N = 11$ our estimate of the integral will improve by a factor of order a hundred. This means that we converge extremely quickly on the true value of the integral, and in practice it is rarely necessary to use more than a few tens of points, or at most perhaps a hundred, to get an estimate of an integral accurate to the limits of precision of the computer.

There are some caveats. An important one is that it must be possible to capture the shape of the integrand well by looking only at its values at the sample points.

When one is calculating an integral using a relatively small number of sample points, the points will inevitably be far apart, which leaves room for the function to vary significantly between them. Since Gaussian quadrature looks only at the values at the sample points and nowhere else, substantial variation between points is not taken into account in calculating the value of the integral. Thus for rapidly varying functions one needs to use enough sample points to capture the variation, and in such cases larger values of N may be warranted.

Another issue is that there is no direct equivalent of Eq. (5.28) for estimating the error in practice. As we have said, however, the error improves by a factor of c/N^2 when the number of samples is increased by one, which is typically a substantial improvement if N is reasonably large. And if we *double* the value of N then we compound many such improvements, giving an overall reduction in the error by a factor of something like N^{-2N}, which is typically a huge improvement.

If we make a Gaussian estimate I_N of the true value I of an integral using N sample points, then $I = I_N + \delta_N$, where δ_N is the approximation error. If we double the number of samples to $2N$, we have $I = I_{2N} + \delta_{2N}$. Equating the two expressions for I and rearranging, we have

$$\delta_N - \delta_{2N} = I_{2N} - I_N. \tag{5.95}$$

But, as we have argued, the error is expected to improve by a large factor when we double the number of sample points, meaning that $\delta_{2N} \ll \delta_N$. So, to a good approximation,

$$\delta_N \simeq I_{2N} - I_N. \tag{5.96}$$

Another way of saying this is that I_{2N} is so much better an estimate of the true value of the integral than I_N that for the purposes of estimating the error we can treat it as if it were the true value, so that $I_{2N} - I_N$ is a good estimate of the error.

We can use Eq. (5.96) in an adaptive integration method where we double the number of sample points at each step, calculating the error and repeating until the desired target accuracy is reached. Such a method is not entirely satisfactory, for a couple of reasons. First, when we double the number of sample points from N to $2N$, Eq. (5.96) gives us only the error on the *previous* estimate of the integral I_N, not on the new estimate I_{2N}. This means that we end up doubling N one more time than is strictly necessary to achieve the desired accuracy, and the final value for the integral will probably be significantly more accurate than we really need it to be, which means we have wasted time on unnecessary calculations. Second, we have to perform the entire calculation of the integral anew for each new value of N. As mentioned earlier, and unlike the adaptive trapezoidal method of Section 5.3, we cannot reuse the results of earlier calculations to speed up the computation. So an adaptive calculation of this type would be slower than just a single instance of Gaussian quadrature. On the other hand, it is straightforward to show that the total number of terms in all the sums we perform, over all steps of the process, is never greater than twice the final value of N used, which means that the adaptive procedure costs us no more than about twice

the effort required for the simple Gaussian quadrature. Moreover, as we have said, we rarely need to go beyond $N = 100$ to get a highly accurate answer, so the number of times we double N is typically rather small. If we start with, say, $N = 10$, we will probably only have to double three or four times. The net result is that, despite the extra work, Gaussian quadrature is often more efficient than methods like the trapezoidal rule or Simpson's rule in terms of overall time needed to get an answer to a desired degree of accuracy.

An alternative, though more complex, solution to the problem of estimating the error in Gaussian quadrature is to use *Gauss–Kronrod quadrature*, a variant of Gaussian quadrature based on the properties of Stieltjes polynomials, which provides not only an accurate estimate of the integral (though not quite as accurate as ordinary Gaussian quadrature) but also an estimate of the error. We will not use Gauss–Kronrod quadrature in this book, but the interested reader can find a discussion, with some derivations, in Appendix A, Section A.1.

5.7 Choosing an integration method

We have studied a number of integration methods in this chapter: the trapezoidal rule and Simpson's rule as well as adaptive versions of both, Romberg integration, and Gaussian integration. You might ask at this point which of all these methods is the best? Which one should you use, in practice, if you need to evaluate an integral?

There is no one answer to this question. Which method you should use depends on the particular problem confronting you. A good general principle, however, is that higher-order methods such as Romberg and Gaussian integration—methods that allow you to make accurate estimates of integrals using relatively few sample points—work best when applied to smooth, well-behaved functions. If your function is not smooth or is poorly behaved in some way, then simpler methods, and particularly the trapezoidal rule, are the way to go. The reason is that any integration method knows only about the value of the integrand at its sample points. If the integrand varies significantly in between the sample points, then that variation will not be reflected in the computed value of the integral, which can lead to inaccurate results. If you are evaluating an integral using only ten or twenty sample points, it is crucial that those points give a good picture of the integrand—if you join up the dots the result should capture most of the shape of the function. If it does not then methods using few sample points will not do a good job. Conversely, for smooth functions whose shape *can* be captured with relatively few sample points, methods such as Romberg and Gaussian integration are excellent choices, providing impressive accuracy with very little computational effort.

Bearing these principles in mind, here is a guide to the kinds of problems each of our integration methods is good for.

The trapezoidal rule: The trapezoidal rule of Section 5.1.1 is trivial to program and hence is a good choice when you need a quick answer for an integral. It is not

very accurate, but sometimes you don't need great accuracy. It uses equally spaced sample points, which is appropriate for applications such as integrating data from laboratory experiments that are sampled at uniform time intervals. The trapezoidal rule is also a good choice for poorly behaved functions—those that vary widely or rapidly, contain singularities, or are noisy. It is usually a better choice for such functions than the other methods we have considered. In its adaptive form (Section 5.3) it can also give us a guaranteed accuracy for an integral, although it may take more computer time to achieve that accuracy than other methods.

Simpson's rule: Simpson's rule (Section 5.1.2) has many of the benefits of the trapezoidal rule, such as simplicity of programming and equally spaced sample points. It gives greater accuracy than the trapezoidal rule with the same number of sample points, or the same accuracy with fewer points, but relies on higher-order approximation of the integrand, which can lead to problems if the integrand is noisy or otherwise not smooth—use it with caution if you are unsure of the nature of your integrand. Its adaptive form (Section 5.3) provides a result of guaranteed accuracy, and does so faster than the equivalent trapezoidal rule calculation, but again may be less suitable for poorly behaved integrands.

Romberg integration: When using equally spaced sample points, Romberg integration (Section 5.4) is the quintessential higher-order integration method. It gives exceptionally accurate estimates of integrals with a minimum number of sample points, plus error estimates that allow you to halt the calculation once you have achieved a desired accuracy. Since it relies on extrapolating answers from measurements of the integrand at only a few points, however, Romberg integration will not work well for wildly varying integrands, noisy integrands, or integrands with mathematically pathological behaviors like singularities. It is best applied to smooth functions whose form can be determined accurately from only a small number of sample points.

Gaussian quadrature: Gaussian quadrature (Section 5.6) has many of the same advantages as Romberg integration (potentially very high accuracy from small numbers of sample points) but also the same disadvantages (poor performance for badly behaved integrands). It is also simple to program, as simple as any of the other methods we have considered. The hard work of the method lies in the calculation of the integration points and weights, which is normally done for you by standard software, and the Gaussian integral itself requires only the evaluation of a single sum in the form of Eq. (5.68). It has the additional advantage over Romberg integration of still higher-order accuracy and indeed, in a certain formal sense, it is the highest-order, and hence potentially most accurate, integration rule available. The price you pay for this is that the integration points are unequally spaced. If you need equally spaced points, then Gaussian quadrature is not the method for you.

Armed with these guidelines, you should be able to choose a suitable integration method for most problems you come up against.

5.8 Integrals over infinite ranges

Often in physics we encounter integrals over infinite ranges, like $\int_0^\infty f(x)\,dx$. The techniques we have seen so far do not work for these integrals because we would need an infinite number of sample points to span an infinite range. The solution to this problem is to change variables. For an integral over the range from 0 to ∞ the standard change of variables is

$$z = \frac{x}{1+x} \qquad \text{or equivalently} \qquad x = \frac{z}{1-z}. \tag{5.97}$$

Then $dx = dz/(1-z)^2$ and

$$\int_0^\infty f(x)\,dx = \int_0^1 \frac{1}{(1-z)^2} f\Big(\frac{z}{1-z}\Big)\,dz, \tag{5.98}$$

which can be done using any of the techniques discussed earlier in the chapter, including the trapezoidal and Simpson rules, or Gaussian quadrature.

This is not the only change of variables we can use, however. In fact, a change of the form

$$z = \frac{x}{c+x} \tag{5.99}$$

would work for any value of c, or $z = x^\gamma/(1+x^\gamma)$ for any γ, or any of a range of other possibilities. Some choices typically work better than others for particular integrals and sometimes you have to play around with things a little to find what works for a given problem, but Eq. (5.97) is often a good first guess. (See Exercise 5.19 for a counterexample.)

To do an integral over a range from some nonzero value a to ∞ we can use a similar approach, but make two changes of variables, first to $y = x - a$, which shifts the start of the integration range to 0, and then $z = y/(1+y)$ as in Eq. (5.97). Or we can combine both changes into a single one:

$$z = \frac{x-a}{1+x-a} \qquad \text{or} \qquad x = \frac{z}{1-z} + a, \tag{5.100}$$

and again $dx = dz/(1-z)^2$, so that

$$\int_a^\infty f(x)\,dx = \int_0^1 \frac{1}{(1-z)^2} f\Big(\frac{z}{1-z} + a\Big)\,dz. \tag{5.101}$$

Integrals from $-\infty$ to a can be done in a similar way using the substitution

$$z = \frac{1}{1-x+a} \qquad \text{or} \qquad x = \frac{z-1}{z} + a, \tag{5.102}$$

which gives

$$\int_{-\infty}^a f(x)\,dx = \int_0^1 \frac{1}{z^2} f\Big(\frac{z-1}{z} + a\Big)\,dz. \tag{5.103}$$

For integrals that run from $-\infty$ to ∞ we can split the integral into two parts, one from $-\infty$ to 0 and one from 0 to ∞, and then use the tricks above for the two integrals separately. Or we can put the split at some other point a and perform separate integrals from $-\infty$ to a and from a to ∞. Alternatively, one could use a single change of variables, such as

$$x = \frac{z}{1-z^2}, \qquad dx = \frac{1+z^2}{(1-z^2)^2}\,dz, \tag{5.104}$$

which would give

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-1}^{1} \frac{1+z^2}{(1-z^2)^2} f\Big(\frac{z}{1-z^2}\Big)\,dz. \tag{5.105}$$

Another possibility, perhaps simpler, is

$$x = \tan z, \qquad dx = \frac{dz}{\cos^2 z}, \tag{5.106}$$

which gives

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-\pi/2}^{\pi/2} \frac{f(\tan z)}{\cos^2 z}\,dz. \tag{5.107}$$

EXAMPLE 5.3: INTEGRATING OVER AN INFINITE RANGE

Let us calculate the value of the following integral using Gaussian quadrature:

$$I = \int_0^{\infty} e^{-t^2} dt. \tag{5.108}$$

We make the change of variables given in Eq. (5.97) and the integral becomes

$$I = \int_0^1 \frac{e^{-z^2/(1-z)^2}}{(1-z)^2}\,dz. \tag{5.109}$$

We can modify our program from Example 5.2 to perform this integral using Gaussian quadrature with $N = 50$ sample points:

File: `intinf.py`

```
from gaussxw import gaussxw
from math import exp

def f(z):
    return exp(-z**2/(1-z)**2)/(1-z)**2

N = 50
a = 0.0
b = 1.0
x,w = gaussxw(N,a,b)
```

```
s = 0.0
for k in range(N):
    s += w[k]*f(x[k])
print(s)
```

If we run this program it prints

```
0.8862269254528359
```

In fact, the value of this integral is known exactly to be $\frac{1}{2}\sqrt{\pi} = 0.8862269254527580\ldots$ Again we see the impressive accuracy of the Gaussian quadrature method: with just 50 sample points, we have calculated an estimate of the integral that is correct to 13 decimal places.

Exercise 5.12: The Stefan–Boltzmann constant

The Planck theory of thermal radiation tells us that in the (angular) frequency interval ω to $\omega + \mathrm{d}\omega$, a black body of unit area radiates electromagnetically an amount of thermal energy equal to $I(\omega)\,\mathrm{d}\omega$ per second, where

$$I(\omega) = \frac{\hbar}{4\pi^2 c^2} \frac{\omega^3}{(\mathrm{e}^{\hbar\omega/k_B T} - 1)}.$$

Here $\hbar$ is Planck's constant over 2π, c is the speed of light, and k_B is Boltzmann's constant.

a) Show that the total rate at which energy is radiated by a black body per unit area, over all frequencies, is

$$W = \frac{k_B^4 T^4}{4\pi^2 c^2 \hbar^3} \int_0^\infty \frac{x^3}{\mathrm{e}^x - 1}\,\mathrm{d}x.$$

b) Write a program to evaluate the integral in this expression. Explain what method you used, and how accurate you think your answer is.

c) Even before Planck gave his theory of thermal radiation around the turn of the 20th century, it was known that the total energy W given off by a black body per unit area per second followed Stefan's law: $W = \sigma T^4$, where σ is the Stefan–Boltzmann constant. Use your value for the integral above to compute a value for the Stefan–Boltzmann constant (in SI units) to three significant figures. Check your result against the known value, which you can find online or in books. You should get good agreement.

Exercise 5.13: Quantum uncertainty in the harmonic oscillator

In units where all the constants are 1, the wavefunction of the nth energy level of the one-dimensional quantum harmonic oscillator—i.e., a spinless point particle in a quadratic potential well—is given by

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}}\,\mathrm{e}^{-x^2/2}\,H_n(x),$$

for $n = 0 \ldots \infty$, where $H_n(x)$ is the nth Hermite polynomial. Hermite polynomials satisfy a relation somewhat similar to that for Fibonacci numbers, although more complex:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x).$$

The first two Hermite polynomials are $H_0(x) = 1$ and $H_1(x) = 2x$.

a) Write a user-defined function `H(n,x)` that calculates $H_n(x)$ for given x and any integer $n \geq 0$. Use your function to make a plot that shows the harmonic oscillator wavefunctions for $n = 0$, 1, 2, and 3, all on the same graph, in the range $x = -4$ to $x = 4$. Hint: There is a function `factorial` in the `math` package that calculates the factorial of an integer.

b) Make a separate plot of the wavefunction for $n = 30$ from $x = -10$ to $x = 10$. Hint: If your program takes too long to run in this case, then you're doing the calculation wrong—the program should take only a second or so to run.

c) The quantum uncertainty in the position of a particle in the nth level of a harmonic oscillator can be quantified by its root-mean-square position $\sqrt{\langle x^2 \rangle}$, where

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2 |\psi_n(x)|^2 \, dx.$$

Write a program that evaluates this integral using Gaussian quadrature with 100 points, then calculates the uncertainty (i.e., the root-mean-square position of the particle) for a given value of n. Use your program to calculate the uncertainty for $n = 5$. You should get an answer in the vicinity of $\sqrt{\langle x^2 \rangle} = 2.3$.

5.9 MULTIPLE INTEGRALS

Integrals over more than one variable are common in physics problems and can be tackled using generalizations of the methods we have already seen. Consider for instance the integral

$$I = \int_0^1 \int_0^1 f(x,y) \, dx \, dy. \tag{5.110}$$

We can rewrite this by defining a function $F(y)$ thus

$$F(y) = \int_0^1 f(x,y) \, dx. \tag{5.111}$$

Then

$$I = \int_0^1 F(y) \, dy. \tag{5.112}$$

Thus one way to do the multiple integral numerically is first to evaluate $F(y)$ for a suitable set of y values, which means performing the integral in Eq. (5.111), then use those values of $F(y)$ to do the integral in Eq. (5.112). For instance, if we do the

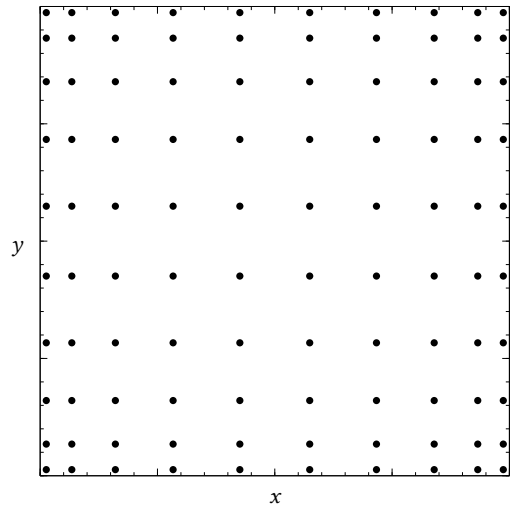


Figure 5.5: Sample points for Gaussian quadrature in two dimensions. If one applies Eq. (5.114) to integrate the function $f(x, y)$ in two dimensions, using Gaussian quadrature with $N = 10$ points along each axis, the resulting set of sample points in the two-dimensional space looks like this.

integrals by Gaussian quadrature with the same number N of points for both x and y integrals, we have

$$F(y) \simeq \sum_{i=1}^{N} w_i f(x_i, y) \qquad \text{and} \qquad I \simeq \sum_{j=1}^{N} w_j F(y_j). \tag{5.113}$$

Alternatively, we can substitute the first sum into the second to get the *Gauss–Legendre product formula*:

$$I \simeq \sum_{i=1}^{N} \sum_{j=1}^{N} w_i w_j f(x_i, y_j). \tag{5.114}$$

This expression has a form similar to the standard integration formula for single integrals, Eq. (5.52), with a sum over values of the function $f(x, y)$ at a set of sample points, multiplied by appropriate weights. Equation (5.114) represents a kind of two-dimensional version of Gaussian quadrature, with weights $w_i w_j$ distributed over a two-dimensional grid of points as shown in Fig. 5.5.

Once you look at it this way, however, you realize that in principle there is no reason why the sample points have to be on a grid. They could be anywhere—we can use any set of 2D locations and suitable weights that give a good estimate of the integral. Just as Gaussian quadrature gives the best choice of points for an integral in one dimension, so we can ask what the best choice is for two dimensions, or for

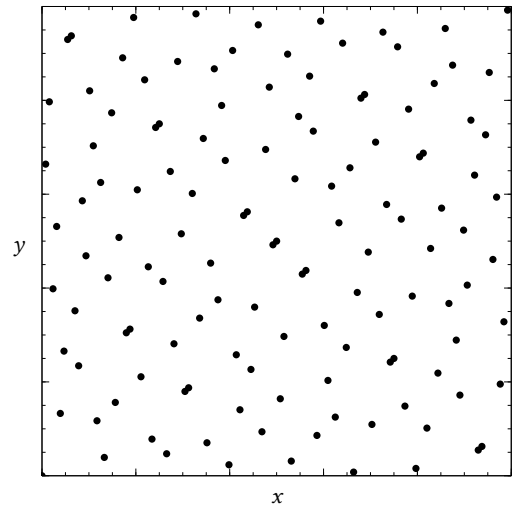


Figure 5.6: 128-point Sobol sequence. The Sobol sequence is one example of a low-discrepancy point set that gives good results for integrals in high dimensions. This figure shows a Sobol sequence of 128 points in two dimensions.

higher dimensions like three or four. It turns out, however, that the answer to this question is not known in general. There are some results for special cases, but no general answer. Various point sets have been proposed for use with 2D integrals that appear to give reasonable results, but there is no claim that they are the best possible choices. Typically they are selected because they have some other desirable properties, such as nesting, and not because they give the most accurate answer. One common choice of point set is the *Sobol sequence*, shown for $N = 128$ points in Fig. 5.6. Sobol sequences and similar sets of points are known as *low-discrepancy point sets* or sometimes *quasi-random point sets* (although the latter name is a poor one because there is nothing random about them). Another common way to choose the sample points is to choose them completely randomly, which leads to the method known as Monte Carlo integration. Choosing points at random may seem like an odd idea, but we will see that it can be a useful approach for certain types of integrals, particularly integrals over very many variables. We will look at Monte Carlo integration in Section 10.2, after we study random number generators.

In the integral of Eq. (5.110) the limits of both integrals are constant, which makes the domain of integration rectangular in xy space. It is not uncommon, however, for the limits of one integral to depend on the other, as here:

$$I = \int_0^1 \mathrm{d}y \int_0^y \mathrm{d}x\, f(x,y). \tag{5.115}$$

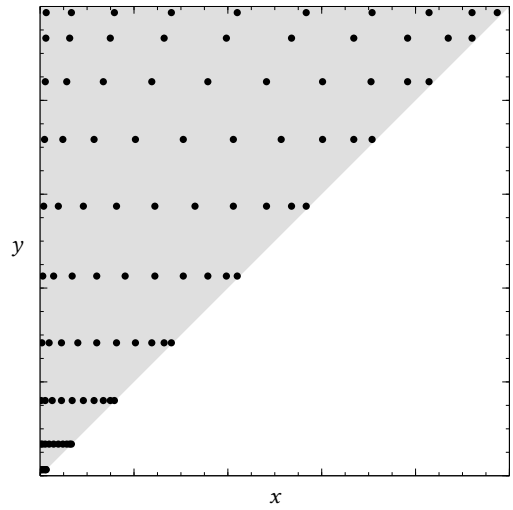


Figure 5.7: Integration over a non-rectangular domain. When the limits of multiple integrals depend on one another they can produce arbitrarily shaped domains of integration. This figure shows the triangular domain that results from the integral in Eq. (5.115). The gray region is the domain of integration. Note how the points become squashed together towards the bottom of the plot.

We can use the same approach as before to evaluate this integral. We define

$$F(y) = \int_0^y f(x, y)\, \mathrm{d}x, \tag{5.116}$$

so that

$$I = \int_0^1 F(y)\, \mathrm{d}y, \tag{5.117}$$

and then do both integrals with any method we choose, such as Gaussian quadrature. The result, again, is a two-dimensional integration rule, but now with the sample points arranged in a triangular space as shown in Fig. 5.7.

This method will work, and will probably give reasonable answers, but it is not ideal. In particular note how the sample points are cramped together in the lower left corner of the integration domain but much farther apart at the top. This means, all other things being equal, that we will have lower accuracy for the part of the integral at the top. It would be better if the accuracy were roughly uniform.

And things can get more complicated still. Suppose the domain of integration takes some more elaborate shape like Fig. 5.8. We will not come across any examples this complicated in this book, but if we did there would be various techniques we could use. One is the Monte Carlo integration method mentioned above, which we study in detail in Section 10.2. Another is to set the integrand to zero everywhere

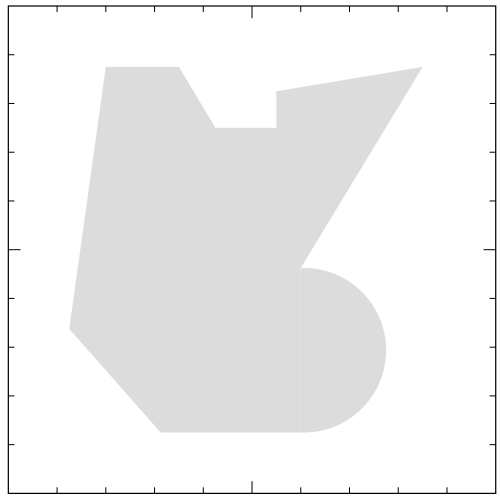
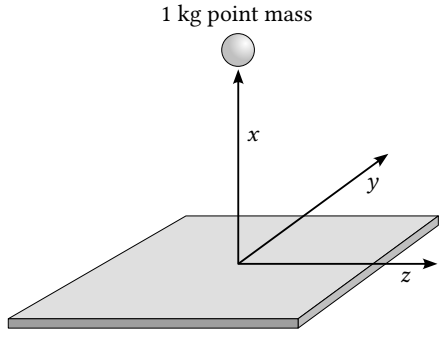


Figure 5.8: A complicated integration domain. Integration domains can be arbitrarily complicated in their shapes. They can even contain holes, or take on complex topologies in higher dimensions such as tori or knotted topologies.

outside the domain of integration and then integrate it using a standard method over some larger, regularly shaped domain, such as a rectangle, that completely encloses the irregular one. There are more sophisticated techniques as well, but we will not need them for the moment.

Exercise 5.14: Gravitational pull of a uniform sheet

A uniform square sheet of metal is floating motionless in space:



The sheet is 10 m on a side and of negligible thickness, and it has a mass of 10 metric tonnes.

a) Consider the gravitational force due to the plate felt by a point mass of 1 kg a distance z from the center of the square in the direction perpendicular to the sheet, as shown above. Show that the component of the force along the z-axis is

$$F_z = G\sigma z \iint_{-L/2}^{L/2} \frac{\mathrm{d}x\,\mathrm{d}y}{(x^2+y^2+z^2)^{3/2}},$$

where $G = 6.674 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ is Newton's gravitational constant and σ is the mass per unit area of the sheet.

b) Write a program to calculate and plot the force as a function of z from $z = 0$ to $z = 10$ m. For the double integral use (double) Gaussian quadrature, as in Eq. (5.114), with 100 sample points along each axis.

c) You should see a smooth curve, except at very small values of z, where the force should drop off suddenly to zero. This drop is not a real effect, but an artifact of the way we have done the calculation. Explain briefly where this artifact comes from and suggest a strategy to remove it, or at least to decrease its size.

This calculation can thought of as a model for the gravitational pull of a galaxy. Most of the mass in a spiral galaxy (such as our own Milky Way) lies in a thin plane or disk, and the gravitational pull exerted by that plane on bodies outside the galaxy can be calculated by methods like the ones we have employed here.

5.10 Derivatives

The opposite of a numerical integral is a numerical derivative. Numerical derivatives are used less than numerical integrals, in part because derivatives of known functions can always be calculated analytically, so there is less need for numerical methods, but they are nonetheless important in certain applications, including the solution of partial differential equations and the training of artificial intelligence models. Like integrals, there are a range of methods for calculating derivatives, from the simple but not very accurate to sophisticated higher-order approximations. We look at each of these in this section, along with the technique of "automatic differentiation," which plays a big role in machine learning.

5.10.1 Forward and backward differences

The standard definition of a derivative, the one you see in calculus books, is

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}. \tag{5.118}$$

The basic method for calculating numerical derivatives is a computational implementation of this formula. We cannot take the limit $h \to 0$ in practice, but we can make h very small and then calculate

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h) - f(x)}{h}. \tag{5.119}$$

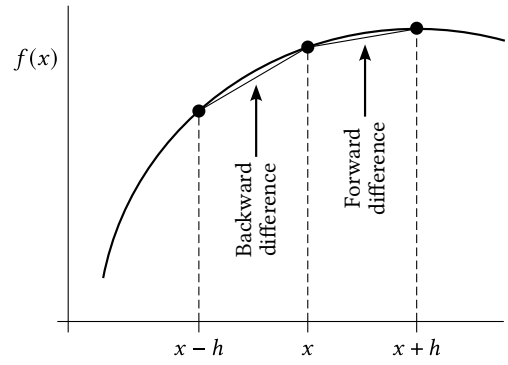


Figure 5.9: Forward and backward differences. The forward and backward differences provide two different approximations to the derivative of a function $f(x)$ at the point x in terms of the slopes of small segments measured in the forward (i.e., positive) direction from x and the backward (negative) direction, respectively.

This approximation to the derivative is called the *forward difference* because it is measured in the forward (i.e., positive) direction from the point of interest x. You can think of it in geometric terms as shown in Fig. 5.9—it is the slope of the curve $f(x)$ measured over a small interval of width h in the forward direction from x.

There is also the *backward difference*, which has the mirror image definition

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x) - f(x-h)}{h}. \tag{5.120}$$

The forward and backward differences typically give about the same answer and in many cases you can use either. Most often one uses the forward difference, but there are a few special cases where one is preferred over the other, particularly when there is a discontinuity in the derivative of the function at the point x or when the domain of the function is bounded and you want the value of the derivative on the boundary, in which case only one or other of the two difference formulas will work. The rest of the time, however, there is little to choose between them.

Before using either the forward or backward difference we must choose a value for h and there is some art to getting the value right. To work out what value to use we need to look at the errors and inaccuracies involved in calculating numerical derivatives.

5.10.2 Errors

Calculations of derivatives using forward and backward differences are not perfectly accurate. There are two sources of error. The first is rounding error of the type discussed in Section 4.2. The second is the approximation error that arises because we cannot take the limit $h \to 0$, so our differences are not really true derivatives. By

contrast with numerical integrals, where, as we have seen, rounding error is usually negligible, it turns out that both sources of error are important when we calculate a derivative.

To understand why this is, let us focus on the forward difference and consider the Taylor expansion of $f(x)$ about x:

$$f(x+h) = f(x) + hf'(x) + \tfrac{1}{2}h^2 f''(x) + \ldots \tag{5.121}$$

where f' and f'' denote the first and second derivatives of f. Rearranging this expression, we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \tfrac{1}{2}hf''(x) + \ldots \tag{5.122}$$

When we calculate the forward difference we calculate only the first part on the right-hand side, and neglect the term in $f''(x)$ and all higher terms. The size of these neglected terms measures the approximation error on the forward difference. Thus, to leading order in h, the absolute magnitude of the approximation error is $\frac{1}{2}h\,|f''(x)|$, which is linear in h so that, as we would expect, we should get more accurate answers if we use smaller values of h.

But now here is the problem: as we saw in Section 4.2, subtracting numbers from one another on a computer can give rise to big rounding errors (in fractional terms) if the numbers are close to one another—so-called catastrophic cancellation. And that is exactly what happens here. The numbers $f(x+h)$ and $f(x)$ that we are subtracting will be very close to one another if we make h small, and we will get a large rounding error as a result. This puts us in a difficult situation: we want to make h small to make the forward difference approximation as accurate as possible, but if we make it too small we will get a large rounding error. To get the best possible answer, we are going to have to find a compromise.

In Section 4.2 we saw that the computer can typically calculate a number such as $f(x)$ to an accuracy of $\epsilon f(x)$, where ϵ is the machine precision, which is typically about 10^{-16} in Python. Since $f(x+h)$ is normally close in value to $f(x)$, the accuracy of our value for $f(x+h)$ will also be about the same, and the absolute magnitude of the total rounding error on $f(x+h) - f(x)$ will, in the worst case, be about $2\epsilon|f(x)|$. It might be better than this if the two errors go in opposite directions and happen to cancel out, but we cannot assume that this will be the case. Then the worst-case rounding error on the complete forward difference, Eq. (5.119), will be $2\epsilon|f(x)|/h$.

Meanwhile, the approximation error is, as we have said, $\frac{1}{2}h\,|f''(x)|$ to leading order in h, from Eq. (5.122), which means that the total error δ on our derivative, in the worst case, is

$$\delta = \frac{2\epsilon|f(x)|}{h} + \tfrac{1}{2}h\left|f''(x)\right|. \tag{5.123}$$

We want to find the value of h that minimizes this error, so we differentiate with respect to h and set the result equal to zero, which gives

$$-\frac{2\epsilon|f(x)|}{h^2} + \tfrac{1}{2}\left|f''(x)\right| = 0, \tag{5.124}$$

or equivalently

$$h = \sqrt{4\epsilon \left| \frac{f(x)}{f''(x)} \right|}. \tag{5.125}$$

Substituting this value back into Eq. (5.123) we find that the error on our derivative is

$$\delta = h\,|f''(x)| = \sqrt{4\epsilon \left| f(x) f''(x) \right|}. \tag{5.126}$$

Thus, for instance, if $f(x)$ and $f''(x)$ are of order 1, we should choose h to be roughly of order $\sqrt{\epsilon}$, which will be about 10^{-8}, and the final error on our result will also be about $\sqrt{\epsilon}$ or 10^{-8}. A similar analysis can be applied to the backward difference, and gives the same result.

In other words, we can get about half of the usual numerical precision on our derivatives but not better. If the machine precision is, as here, about 16 digits, then we can get 8 digits of precision on our derivatives. This is substantially poorer than most of the calculations we have seen so far in this book, and could be a significant source of error for calculations that require high accuracy.

5.10.3 CENTRAL DIFFERENCES

We have seen that forward and backward differences are not very accurate. What can we do to improve the situation? A simple improvement is to use the *central difference*:

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}. \tag{5.127}$$

The central difference is similar to the forward and backward differences, approximating the derivative using the difference between two values of $f(x)$ at points a distance h apart. What has changed is that the two points are now placed symmetrically around x, one at a distance $\frac{1}{2}h$ in the forward (i.e., positive) direction and the other at a distance $\frac{1}{2}h$ in the backward (negative) direction.

To calculate the approximation error on the central difference we write two Taylor series:

$$f(x + \tfrac{1}{2}h) = f(x) + \tfrac{1}{2}hf'(x) + \tfrac{1}{8}h^2 f''(x) + \tfrac{1}{48}h^3 f'''(x) + \ldots \tag{5.128}$$

$$f(x - \tfrac{1}{2}h) = f(x) - \tfrac{1}{2}hf'(x) + \tfrac{1}{8}h^2 f''(x) - \tfrac{1}{48}h^3 f'''(x) + \ldots \tag{5.129}$$

Subtracting the second expression from the first and rearranging for $f'(x)$, we get

$$f'(x) = \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h} - \tfrac{1}{24}h^2 f'''(x) + \ldots \tag{5.130}$$

To leading order the magnitude of the error is now $\frac{1}{24}h^2\,|f'''(x)|$, which is one order in h better than before. There is also, as before, a rounding error. Its size is unchanged

from our previous calculation, having magnitude $2\epsilon|f(x)|/h$, so the magnitude of the total error on our estimate of the derivative is

$$\delta = \frac{2\epsilon|f(x)|}{h} + \tfrac{1}{24}h^2\left|f'''(x)\right|. \tag{5.131}$$

Differentiating to find the minimum and rearranging, we find that the optimal value of h is

$$h = \left(24\epsilon\left|\frac{f(x)}{f'''(x)}\right|\right)^{1/3}, \tag{5.132}$$

and substituting this back into Eq. (5.131) we find the optimal error itself to be

$$\delta = \tfrac{1}{8}h^2\left|f'''(x)\right| = \left(\tfrac{9}{8}\epsilon^2[f(x)]^2\left|f'''(x)\right|\right)^{1/3}. \tag{5.133}$$

Thus, for instance, if $f(x)$ and $f'''(x)$ are of order 1, the ideal value of h is going to be around $\epsilon^{1/3}$, which is typically about 10^{-5}, but the error will be around $\epsilon^{2/3}$, or about 10^{-10}.

Thus the central difference is indeed more accurate than the forward and backward differences, by a factor of 100 or so in this case, although we get this accuracy by using a *larger* value of h. This may seem surprising, but it is the correct result.

Example 5.4: Derivative of a sampled function

As an example of the central difference, suppose we are given the values of a function $f(x)$ measured at regularly spaced sample points a distance h apart—see Fig. 5.10. One often gets such samples from data collected in the laboratory, for example. And suppose we want to calculate the derivative of f at one of the points (case (a) in the figure). We could use a forward or backward difference based on the sample at x and one of the adjacent ones, or we could use a central difference. However, if we use a central difference, which is based on points equally spaced on either side of x, then we must use the points at $x + h$ and $x - h$. We cannot, as in Eq. (5.127), use points at $x + \frac{1}{2}h$ and $x - \frac{1}{2}h$ because there are no such points—we only have the samples we are given. The formula for the central difference in this case will thus be

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h) - f(x-h)}{2h}. \tag{5.134}$$

This means that the interval between the points we use is $2h$ for the central difference, but only h for the forward and backward differences. So which will give a better answer? The central difference because it is a better approximation or the forward difference because of its smaller interval?

From Eq. (5.126) we see that the error on the forward difference is $h\,|f''(x)|$ and from Eq. (5.133) the error on the central difference—with h replaced by $2h$—is $\frac{1}{2}h^2\,|f'''(x)|$. Which is smaller depends on the value of h. For the central difference to give the more accurate answer, we require $\frac{1}{2}h^2\,|f'''(x)| < h\,|f''(x)|$ or

$$h < 2\left|\frac{f''(x)}{f'''(x)}\right|. \tag{5.135}$$

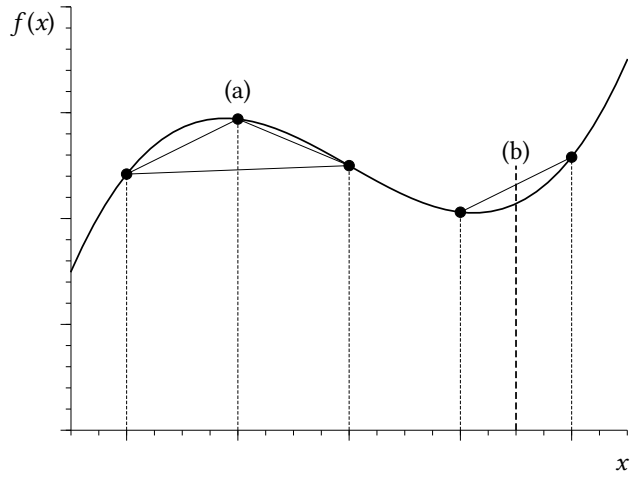


Figure 5.10: Derivative of a sampled function. (a) If we only know the function at a set of sample points spaced a distance h apart then we must chose between calculating the forward or backward difference between adjacent samples, or the central difference between samples $2h$ apart (straight lines). We cannot calculate a central difference using the standard formula, Eq. (5.127), because we do not know the value of the function at $x \pm \frac{1}{2}h$. (b) We can, however, calculate the central difference at a point half way between two samples using the standard formula.

If h is larger than this then the forward difference is actually the better approximation in this case.

But now suppose that instead of calculating the value of the derivative at one of the sample points, we calculate it at a point x that lies half way between two of the samples—case (b) in Fig. 5.10. Viewed from that point we do have samples at $x + \frac{1}{2}h$ and $x - \frac{1}{2}h$, so now we can use the original form of the central difference, Eq. (5.127), with an interval only h wide, as with the forward difference. This calculation will give a more accurate answer, but only at the expense of calculating the result at a point in between the samples.

Exercise 5.15: Even when we can find the value of $f(x)$ for any value of x, the forward difference can still be more accurate than the central difference for sufficiently large h. For what values of h will the approximation error on the forward difference of Eq. (5.119) be smaller than on the central difference of Eq. (5.127)?

Exercise 5.16: Create a user-defined function `f(x)` that returns the value $1 + \frac{1}{2}\tanh 2x$, then use a central difference to calculate the derivative of the function in the range $-2 \le x \le 2$.

Calculate an analytic formula for the derivative and make a graph with your numerical result and the analytic answer on the same plot. It may help to plot the exact answer as lines and the numerical one as dots. (Hint: In Python the tanh function is found in the `math` package and it is called simply `tanh`.)

5.10.4 Higher-order approximations for derivatives

One way to think about the numerical derivatives of the previous sections is that we are fitting a straight line through two points, such as the points $f(x)$ and $f(x+h)$, and then asking about the slope of that line at the point x. The trapezoidal rule of Section 5.1.1 does a similar thing for integrals, approximating a curve by a straight line between sample points and estimating the area under the curve using that line. We saw that we can make a higher-order—and usually better—approximation to an integral by fitting a quadratic or higher-order polynomial instead of a straight line, and this led to the Simpson and Newton–Cotes rules for integrals. We can take a similar approach with derivatives by fitting a polynomial to a set of sample points and then calculating the derivative of the polynomial at x.

Consider, for example, fitting a quadratic curve $y = ax^2 + bx + c$ to the function $f(x)$. We require three sample points to make the fit, and suppose for example that we are interested in the derivative at $x = 0$, so we place our three points at $-h$, 0, and $+h$, for some h that we choose. Requiring that our quadratic is equal to $f(x)$ at these three points gives us three equations thus:

$$ah^2 - bh + c = f(-h), \qquad c = f(0), \qquad ah^2 + bh + c = f(h), \tag{5.136}$$

In principle, we can now solve these equations for the three parameters a, b, and c. (This is the same calculation that we did in Section 5.1.2 for Simpson's rule.) However, in this case, we don't need the whole solution, because we don't need all of the parameters. Given the quadratic fit $y = ax^2 + bx + c$, the derivative of the curve at the point $x = 0$ is

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \Big[2ax + b\Big]_{x=0} = b. \tag{5.137}$$

So we need only the one parameter b, which we can get from Eq. (5.136) by subtracting the first equation from the third to give $2bh = f(h) - f(-h)$ and rearranging. Thus our approximation for the derivative at $x = 0$ is

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(h) - f(-h)}{2h}. \tag{5.138}$$

We have done this calculation for the derivative at $x = 0$, but the same result applies at any other point—we can slide the whole function up or down the x-axis, to put any point x at the origin and then calculate the derivative from the formula above. Or, equivalently, we can just write

$$\frac{\mathrm{d}f}{\mathrm{d}x} \simeq \frac{f(x+h) - f(x-h)}{2h} \tag{5.139}$$

Degree	$f(-\frac{5}{2}h)$	$f(-2h)$	$f(-\frac{3}{2}h)$	$f(-h)$	$f(-\frac{1}{2}h)$	$f(0)$	$f(\frac{1}{2}h)$	$f(h)$	$f(\frac{3}{2}h)$	$f(2h)$	$f(\frac{5}{2}h)$	Error
1					-1		1					$O(h^2)$
2				$-\frac{1}{2}$				$\frac{1}{2}$				$O(h^2)$
3			$\frac{1}{24}$		$-\frac{27}{24}$		$\frac{27}{24}$		$-\frac{1}{24}$			$O(h^4)$
4		$\frac{1}{12}$		$-\frac{2}{3}$				$\frac{2}{3}$		$-\frac{1}{12}$		$O(h^4)$
5	$-\frac{3}{640}$		$\frac{25}{384}$		$-\frac{75}{64}$		$\frac{75}{64}$		$-\frac{25}{384}$		$\frac{3}{640}$	$O(h^6)$

Table 5.1: Coefficients for numerical derivatives. The coefficients for central approximations to the first derivative of $f(x)$ at $x = 0$. To derive the full expression for an approximation, multiply the samples listed in the top row of the table by the coefficients in one of the other rows, then divide by h. For instance, the cubic approximation would be $\left[\frac{1}{24}f(-\frac{3}{2}h) - \frac{27}{24}f(-\frac{1}{2}h) + \frac{27}{24}f(\frac{1}{2}h) - \frac{1}{24}f(\frac{3}{2}h)\right]/h$. For derivatives at points other than $x = 0$ the same coefficients apply—one just uses the appropriate sample points around the value x of interest. The final column of the table gives the order of the approximation error on the derivative.

for general x.

This is the correct result for the quadratic approximation, but it is a disappointing one, since Eq. (5.139) is nothing other than the central difference approximation for sample points $2h$ apart, which we already saw in Eq. (5.134). In other words, the higher-order approximation has not helped us in this case.

However, going to still higher orders does help. If we use a cubic or quartic approximation, we do get improved estimates of the derivative. At higher orders there is a distinction between the odd- and even-order approximations. For the odd-order ones the sample points fall at "half-way" points, as with the central difference of Eq. (5.127). For instance, to get the four sample points required for a cubic approximation, symmetrically distributed about zero, we would choose them to fall at $x = -\frac{3}{2}h$, $-\frac{1}{2}h$, $\frac{1}{2}h$, and $\frac{3}{2}h$. For even-order approximations, on the other hand, the samples fall at "integer" points. The five points for the quartic approximation, for instance, fall at $-2h$, $-h$, 0, h, and $2h$. The methodology for deriving the higher-order approximations follows the same pattern as for the quadratic case: we write down the required value of the polynomial at each of the sample points, which gives us a set of simultaneous equations in the polynomial coefficients. As before, we actually need only one of those coefficients, the coefficient of the linear term in the polynomial. Solving for this coefficient gives us our expression for the derivative. At each order the expression is a linear combination of the samples, divided by h. We will not go through the derivations in detail, but Table 5.1 gives the coefficients of the combinations for the first five approximations.

Each of the approximations given in the table is exact, apart from rounding error, if the function being differentiated is actually a polynomial of the appropriate (or lower) degree, so that the polynomial fit is a perfect one. Most of the time, however, this will not be the case and there will be an approximation error involved in calculating the derivative. One can calculate this error to leading order for each of the approximations by a method analogous to our calculations for the forward, backward, and central differences: we perform Taylor expansions about $x = 0$ to derive

expressions for $f(x)$ at each of the sample points, then plug these expressions into the formula for the derivative. The order in h of the resulting error is listed in the final column of Table 5.1. As before, this approximation error must be balanced against the rounding error and a suitable value of h chosen to minimize the overall error in the derivative.

An interesting point to notice about Table 5.1 is that the coefficient for $f(0)$ in all the approximations is zero. The value of the function exactly at the point of interest never plays a role in the evaluation of the derivative. Another (not unrelated) point is that the order in h of the error given in the final column does not go up uniformly with the degree of the polynomial—it is the same for the even-degree polynomials as for the next-lower odd-degree ones. We saw a special case of this result for the quadratic: the quadratic fit just gives us an ordinary central difference and therefore necessarily has an error $\mathrm{O}(h^2)$, the same as the central difference derived from the linear fit. In general, the odd-degree approximations give us slightly more accurate results than the even-degree ones—the error is of the same order in h but the constant of proportionality is smaller. On the other hand, the odd-degree approximations require samples at the half-way points, as we have noted, which can be inconvenient. As discussed in Example 5.4, we sometimes have samples at only the "integer" points, in which case we must use the even-degree approximations.

5.10.5 Richardson extrapolation

An alternative way to compute higher-order approximations to derivatives is to use Richardson extrapolation. We encountered Richardson extrapolation previously in our study of Romberg integration in Section 5.4, where it provided us with a way to increase the accuracy of integral evaluations, while doing only a little extra computational work. Richardson extrapolation can also be applied to derivatives. We illustrate the approach here using central differences, although it can also be applied to forward and backward differences.

Equation (5.130) on page 182 tells us that the basic central difference has an approximation error $\mathrm{O}(h^2)$ to leading order. In fact, noting the alternating signs in the Taylor expansion of Eq. (5.129), it is straightforward to see that the central difference only has error terms of even order in h and no odd-order terms. So the central difference for a function $f(x)$ can be written

$$f'(x) = \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h} + ch^2 + \mathrm{O}(h^4), \tag{5.140}$$

where c is a constant. Suppose we make an estimate—call it D_1—of the derivative of $f(x)$ using a central difference with step size h_1:

$$D_1 = \frac{f(x + \frac{1}{2}h_1) - f(x - \frac{1}{2}h_1)}{h_1}. \tag{5.141}$$

Then from (5.140) we have

$$f'(x) = D_1 + ch_1^2 + \mathrm{O}(h_1^4). \tag{5.142}$$

Now let us halve our step size to $h_2 = \frac{1}{2}h_1$ and repeat the calculation to get another estimate D_2 such that

$$f'(x) = D_2 + ch_2^2 + \mathrm{O}(h_2^4). \tag{5.143}$$

Equations (5.142) and (5.143) are both expressions for the same derivative, so we can equate them to get

$$D_1 + ch_1^2 + \mathrm{O}(h_1^4) = D_2 + ch_2^2 + \mathrm{O}(h_2^4), \tag{5.144}$$

or

$$ch_2^2 = \tfrac{1}{3}(D_2 - D_1) + \mathrm{O}(h_2^4), \tag{5.145}$$

where we have made use of $h_1 = 2h_2$. Substituting this expression back into Eq. (5.143), we now have

$$f'(x) = D_2 + \tfrac{1}{3}(D_2 - D_1) + \mathrm{O}(h_2^4), \tag{5.146}$$

which is accurate to order h^3 and has a error of order h^4, two orders better than the original central difference.

We can take this argument further. Suppose we compute a whole series of central differences, $D_1, D_2, D_3 \ldots$, each with half the step size of the previous one, and let us define a notation similar to the one we used for Romberg integration in Section 5.4, such that

$$R_{i,1} = D_i\,, \qquad R_{i,2} = D_i + \tfrac{1}{3}(D_i - D_{i-1}) = R_{i,1} + \tfrac{1}{3}(R_{i,1} - R_{i-1,1}). \tag{5.147}$$

Then, from Eq. (5.146),

$$f'(x) = R_{i,2} + c_2 h_i^4 + \mathrm{O}(h_i^6), \tag{5.148}$$

where c_2 is another constant and we have made use of the fact that the series for the central difference contains only even powers of h. Similarly,

$$f'(x) = R_{i-1,2} + c_2 h_{i-1}^4 + \mathrm{O}(h_{i-1}^6) = R_{i-1,2} + 16c_2 h_i^4 + \mathrm{O}(h_i^6). \tag{5.149}$$

Equating (5.148) and (5.149) and rearranging, we get

$$c_2 h_i^4 = \tfrac{1}{15}(R_{i,2} - R_{i-1,2}) + \mathrm{O}(h_i^6). \tag{5.150}$$

And substituting this expression back into (5.148) gives

$$f'(x) = R_{i,2} + \tfrac{1}{15}(R_{i,2} - R_{i-1,2}) + \mathrm{O}(h_i^6). \tag{5.151}$$

Now we have eliminated the h_i^4 term and generated an estimate of the derivative accurate to fifth order, with a sixth-order error.

We can continue this process, calculating higher and higher order terms and getting more and more accurate results. The derivation follows the same the lines as for Romberg integration and gives similar looking formulas, with a recurrence relation for the Richardson estimates of

$$R_{i,m+1} = R_{i,m} + \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m}), \tag{5.152}$$

and an error on $R_{i,m}$ of

$$c_m h_i^{2m} = \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m}). \tag{5.153}$$

To make use of these results we do the following:

1. First we calculate our initial central differences $D_1 \equiv R_{1,1}$ and $D_2 \equiv R_{2,1}$, using the standard formula, Eq. (5.127).
2. From these we calculate the more accurate estimate $R_{2,2}$ using Eq. (5.152). This is as much as we can do with only the two starting estimates.
3. Now we calculate the next central difference $D_3 \equiv R_{3,1}$ and from this, with Eq. (5.152), we calculate $R_{3,2}$, and then $R_{3,3}$.
4. At each successive stage we compute one more central difference $D_i \equiv R_{i,1}$, and from it, with very little extra effort, we can calculate $R_{i,2} \ldots R_{i,i}$.
5. For each estimate we can also calculate the error, Eq. (5.153).

We can now continue this process until the error reaches a desired level of accuracy then stop, or we can simply continue until the value of h decreases to the point where rounding error makes further calculations pointless, which for the central difference is around $h = 10^{-5}$ (see Section 5.10.3).

The structure of the calculation can be represented with a diagram similar to the one we used for Romberg integration:

$$\begin{array}{lllllll}
D_1 \equiv R_{1,1} & & & & & & \\
 & \searrow & & & & & \\
D_2 \equiv R_{2,1} & \rightarrow & R_{2,2} & & & & \\
 & \searrow & & \searrow & & & \\
D_3 \equiv R_{3,1} & \rightarrow & R_{3,2} & \rightarrow & R_{3,3} & & \\
 & \searrow & & \searrow & & \searrow & \\
D_4 \equiv R_{4,1} & \rightarrow & R_{4,2} & \rightarrow & R_{4,3} & \rightarrow & R_{4,4} \\
 & \searrow & & \searrow & & \searrow & & \searrow
\end{array}$$

Each row here corresponds to one central difference estimate D_i followed by the other higher-order estimates derived from it. The most accurate estimate we get from the whole process is the very last one: if we do n levels of the process, then the last estimate is $R_{n,n}$, which is accurate to order h_n^{2n}.

5.10.6 Second derivatives

So far our discussion has focused on calculation of the first derivative of a function $f(x)$, but we can also calculate numerical approximations to the second and

higher derivatives. The second derivative is, by definition, the derivative of the first derivative, so we can calculate it by applying our first-derivative formulas twice. For example, starting with the central difference formula, Eq. (5.127), we can write expressions for the first derivative at $x + \frac{1}{2}h$ and $x - \frac{1}{2}h$ thus:

$$f'(x+\tfrac{1}{2}h) \simeq \frac{f(x+h)-f(x)}{h}, \qquad f'(x-\tfrac{1}{2}h) \simeq \frac{f(x)-f(x-h)}{h}. \tag{5.154}$$

Then we apply the central difference again to get an expression for the second derivative:

$$\begin{aligned} f''(x) &\simeq \frac{f'(x+\frac{1}{2}h) - f'(x-\frac{1}{2}h)}{h} \\ &= \frac{[f(x+h)-f(x)]/h - [f(x)-f(x-h)]/h}{h} \\ &= \frac{f(x+h)-2f(x)+f(x-h)}{h^2}. \end{aligned} \tag{5.155}$$

This is the simplest approximation for the second derivative. We will use it extensively in Chapter 9 for solving second-order differential equations. Higher-order approximations exist too, but we will not use them in this book.

We can also calculate the error on Eq. (5.155). We perform two Taylor expansions of $f(x)$ thus:

$$f(x+h) = f(x) + hf'(x) + \tfrac{1}{2}h^2 f''(x) + \tfrac{1}{6}h^3 f'''(x) + \tfrac{1}{24}f''''(x) + \ldots \tag{5.156}$$

$$f(x-h) = f(x) - hf'(x) + \tfrac{1}{2}h^2 f''(x) - \tfrac{1}{6}h^3 f'''(x) + \tfrac{1}{24}f''''(x) - \ldots \tag{5.157}$$

Adding them together and rearranging, we find that

$$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} - \tfrac{1}{12}h^2 f''''(x) + \ldots \tag{5.158}$$

The first term on the right is our formula for the second derivative, Eq. (5.155), and the remainder of the terms measure the error. Thus, to leading order, the absolute error inherent in our approximation to the second derivative is $\frac{1}{12}h^2\,|f''''(x)|$. As before, we also need to take rounding error into account, which contributes an error of roughly $\epsilon|f(x)|$ on each value of $f(x)$, where ϵ is the machine precision, so that, in the worst case, the total rounding error in the numerator of (5.155) is $4\epsilon|f(x)|$ and the rounding error on the whole expression is $4\epsilon|f(x)|/h^2$. Then the complete error on the derivative is

$$\delta = \frac{4\epsilon|f(x)|}{h^2} + \tfrac{1}{12}h^2\left|f''''(x)\right|. \tag{5.159}$$

Differentiating with respect to h and setting the result to zero then gives an optimum value of h of

$$h = \left(48\epsilon\left|\frac{f(x)}{f''''(x)}\right|\right)^{1/4}. \tag{5.160}$$

Substituting this expression back into Eq. (5.159) gives the size of the error to be

$$\delta = \tfrac{1}{6}h^2\left|f''''(x)\right| = \left(\tfrac{4}{3}\epsilon\left|f(x)f''''(x)\right|\right)^{1/2}. \tag{5.161}$$

So if, for instance, $f(x)$ and $f''''(x)$ are of order 1, the error will be roughly of order $\sqrt{\epsilon}$, which is typically about 10^{-8}. This is about the same accuracy as we found for the forward and backward difference approximations to the first derivative in Section 5.10.2. Thus our expression for the second derivative is not very accurate—about as good as, but not better than, the forward difference. As mentioned above, there are higher-order approximations for the second derivative that can give more accurate answers. But for our purposes Eq. (5.155) will be good enough.

We can use the same method to derive expressions for higher derivatives too. For instance, the third derivative is given by

$$f'''(x) \simeq \frac{f(x+\frac{3}{2}h) - 3f(x+\frac{1}{2}h) + 3f(x-\frac{1}{2}h) - f(h-\frac{3}{2}h)}{h^3}, \tag{5.162}$$

and there are similar formulas for fourth and higher derivatives too. In this book, however, we will only need derivatives up to the second.

5.10.7 Partial derivatives

We will come across a number of situations in this book where we need to calculate partial derivatives—derivatives of a function of several variables with respect to only one of those variables. The calculation of such partial derivatives is a simple generalization of the calculation of ordinary derivatives. If you have a function $f(x,y)$ of two variables, for instance, then the central difference approximations to derivatives with respect to x and y are

$$\frac{\partial f}{\partial x} = \frac{f(x+\frac{1}{2}h, y) - f(x-\frac{1}{2}h, y)}{h}, \tag{5.163}$$

$$\frac{\partial f}{\partial y} = \frac{f(x, y+\frac{1}{2}h) - f(x, y-\frac{1}{2}h)}{h}. \tag{5.164}$$

By analogy with our approach for the second derivative in Section 5.10.6 we can also calculate second derivatives with respect to either variable, or a mixed second derivative with respect to both, which is given by

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{f(x+\frac{1}{2}h, y+\frac{1}{2}h) - f(x-\frac{1}{2}h, y+\frac{1}{2}h) - f(x+\frac{1}{2}h, y-\frac{1}{2}h) + f(x-\frac{1}{2}h, y-\frac{1}{2}h)}{h^2}. \tag{5.165}$$

We leave the derivation to the avid reader.

5.10.8 Derivatives of noisy data

Suppose we have some measurements of a quantity that, when plotted on a graph, look like Fig. 5.11a. Perhaps they come from an experiment in the lab, for instance.

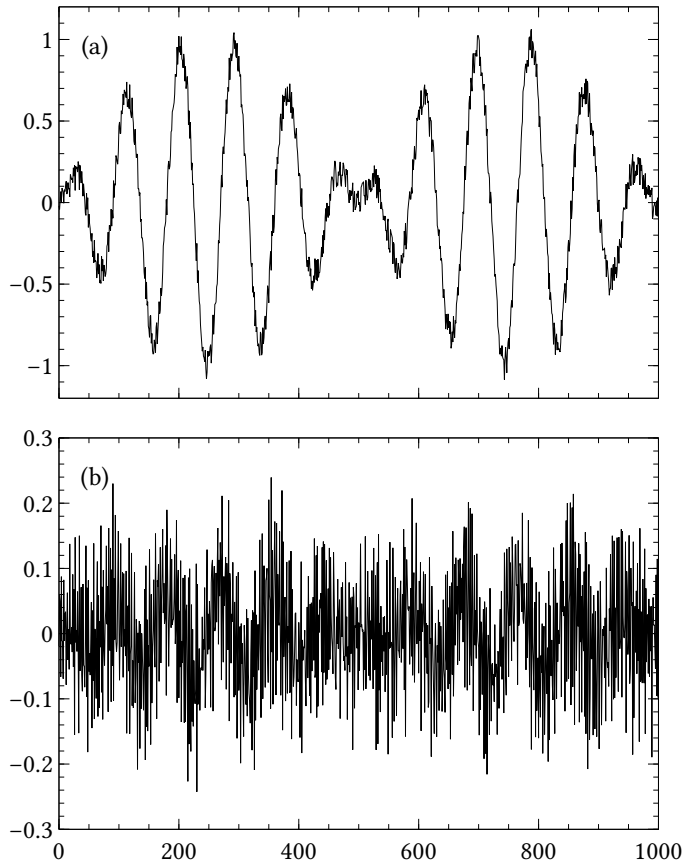


Figure 5.11: Derivative of noisy data. (a) An example of a noisy data set. The data plotted in this graph have a clear underlying form, but contain some random noise or experimental error as well. (b) The derivative of the same data calculated using a forward difference. The action of taking the derivative amplifies the noise and makes the underlying form difficult to discern.

The overall shape of the curve is clear from the figure, but there is some noise in the data, so the curve is not completely smooth.

Now suppose we calculate the first derivative of this curve. We write a program to calculate, say, the forward difference at each point and plot the values we get. The result is shown in Fig. 5.11b. As you can see, taking the derivative has made our noise problem much worse. Now it is almost impossible to see the shape of the curve. This is a known problem with numerical derivatives. If there is any noise in the curve you are differentiating, then it can be greatly exaggerated by taking the derivative, perhaps to the point where the results are useless.

The reason for the problem is easy to see if you zoom in on a small portion of the

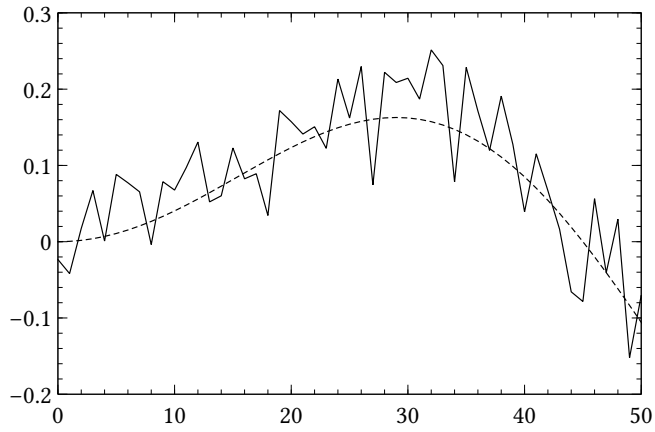


Figure 5.12: An expanded view of the noisy data. The jagged line in this plot is an enlargement of the first portion of the data from Fig. 5.11a, while the dotted line is a guess about the form of the underlying curve, without the noise.

original data, as shown in Fig. 5.12. In this figure the solid line represents the actual data, and the dotted line is a sketch of what the underlying curve, without the noise, probably looks like. (We do not usually know the underlying curve, so this is just a guess.) When viewed close-up like this, we can see that, because of the noise, the slope of the noisy line is very steep in some places, and completely different from the slope of the underlying curve. Although the noisy curve follows the general form of the underlying one, its derivative does not. So now, when we calculate the derivative, we generate spurious large values where there should be none.

Unfortunately, this kind of issue is common with physics data, and this is one of the reasons why numerical derivatives are less used than numerical integrals. There are some things we can do to mitigate the problem, although they all also decrease the accuracy of our results:

1. The simplest thing we can do is increase the value of h. We can treat the noise in the same way that we treat rounding error and calculate an optimum value for h that balances the error from the noise against the error in our approximation of the derivative. The end result is a formula similar to Eq. (5.125) for the forward difference or Eq. (5.132) for the central difference, but with the machine precision ϵ replaced by the fractional error introduced into the data by the noise (which is the inverse of the signal-to-noise ratio).
2. Another approach is to fit a curve to a portion of the data near the point where we want the derivative, then differentiate the curve. For instance, we might fit a quadratic or a cubic, then differentiate that. We do not, however, fit a quadratic to just three sample points or a cubic to just four, as we did in Section 5.10.4. Instead we find the curve that best approximates a larger number of

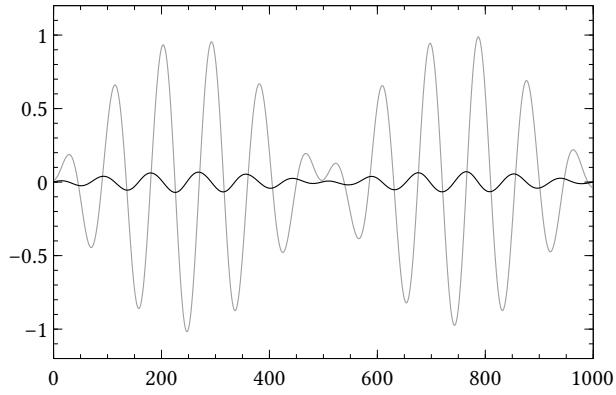


Figure 5.13: Smoothed data and an improved estimate of the derivative. The gray curve in this plot is a version of the data from Fig. 5.11a that has been smoothed to remove noise using a Fourier transform method. The black curve shows the numerical derivative of the smoothed function, which is a significant improvement over Fig. 5.11b.

points, even though it will not typically pass exactly through all those points. In effect, we are trying to find an approximation to the underlying smooth curve depicted in Fig. 5.12. The derivative of this curve then gives an estimate of the true derivative of the data without noise. Methods for fitting curves to data like this are discussed in Section 11.5.

3. A third approach is to smooth the data in some other manner before differentiating, which can be done, for instance, using Fourier transforms, which we study in Chapter 7. (See Exercise 7.4 for an example of Fourier smoothing.) Figure 5.13 shows a version of the data from Fig. 5.11 that has been smoothed in this way, and the corresponding derivative, which is much cleaner now.

5.11 AUTOMATIC DIFFERENTIATION

A completely different approach to the numerical calculation of derivatives is *automatic differentiation*. Automatic differentiation is a technique that allows us to calculate an exact value (apart from rounding error) for the derivative of any function for which we can write computer code. Automatic differentiation can be used to calculate the derivative of a simple polynomial function for instance, or something as complicated as a sum or integral computed with many loops and operations. However, it cannot be used for calculating the derivative of something like experimental data—it applies only to functions calculated on the computer. But for these it is far more accurate than typical numerical derivatives, such as forward or backward differences, and does not even involve a great deal of computational effort. Automatic differentiation is a mainstay of a number of important technologies, particularly in

machine learning and artificial intelligence—it is a crucial component of the algorithms used to train these systems.

Automatic differentiation with respect to a variable t works by replacing any other variable x that depends on t with a pair of values $[x, x']$, where x is the value of the variable at some t and x' is its derivative with respect to t at the same point. For instance, if we are calculating the value of $x = t^2$ at $t = 1$ we would actually store the pair of values $[1, 2]$, because $x = t^2 = 1$ and $x' = 2t = 2$.

Automatic differentiation further replaces each elementary mathematical operation in a program with a generalized version that takes as input these value/derivative pairs and produces similar pairs as output. For instance, the operation of multiplying a variable by a constant $x \to cx$ would be replaced by the operation $[x, x'] \to [cx, cx']$. Or consider the operation of taking a power $x \to x^c$. Applying the chain rule, the derivative of x^c with respect to t is

$$\frac{\mathrm{d}}{\mathrm{d}t}(x^c) = cx^{c-1}\frac{\mathrm{d}x}{\mathrm{d}t}, \tag{5.166}$$

so our operation $x \to x^c$ becomes $[x, x'] \to [x^c, cx^{c-1}x']$.

Here is a list of common operations:

$$\begin{array}{lll}
x \to x + c & \text{becomes} & [x, x'] \to [x + c, x'] \\
x \to cx & \text{becomes} & [x, x'] \to [cx, cx'] \\
x \to x^c & \text{becomes} & [x, x'] \to [x^c, cx^{c-1}x'] \\
x \to c^x & \text{becomes} & [x, x'] \to [c^x, (c^x \ln c)\, x'] \\
x \to \log x & \text{becomes} & [x, x'] \to [\log x, x'/x] \\
x \to \sin x & \text{becomes} & [x, x'] \to [\sin x, x' \cos x] \\
x \to \cos x & \text{becomes} & [x, x'] \to [\cos x, -x' \sin x] \\
x \to \tan x & \text{becomes} & [x, x'] \to [\tan x, x' \sec^2 x]
\end{array}$$

These are all unary operations, ones that operate on a single variable x. We can also define generalizations of binary mathematical operations, such as addition. For two variables x and y the standard addition operation that maps $x, y \to x+y$ becomes $[x, x'], [y, y'] \to [x + y, x' + y']$. A more complex example is multiplication: if we multiply x and y together, then, by the chain rule, the derivative of their product is

$$\frac{\mathrm{d}}{\mathrm{d}t}(xy) = x\frac{\mathrm{d}y}{\mathrm{d}t} + y\frac{\mathrm{d}x}{\mathrm{d}t}. \tag{5.167}$$

So the normal multiplication operation $x, y \to xy$ becomes $[x, x'], [y, y'] \to [xy, xy' + yx']$. Here is a list of common binary operations:

$$\begin{array}{lll}
x, y \to x + y & \text{becomes} & [x, x'], [y, y'] \to [x + y, x' + y'] \\
x, y \to x - y & \text{becomes} & [x, x'], [y, y'] \to [x - y, x' - y'] \\
x, y \to xy & \text{becomes} & [x, x'], [y, y'] \to [xy, xy' + yx'] \\
x, y \to \dfrac{x}{y} & \text{becomes} & [x, x'], [y, y'] \to \left[\dfrac{x}{y}, \dfrac{yx' - xy'}{y^2}\right] \\
x, y \to x^y & \text{becomes} & [x, x'], [y, y'] \to [x^y, (x^y \log x)x' + yx^{y-1}y']
\end{array}$$

And so forth. The general rule for unary operations $U(x)$ is

$$x \to U(x) \quad \text{becomes} \quad [x, x'] \to [U(x), U'(x)\, x'],$$

where U' denotes the derivative of U with respect to its argument. The general rule for binary operations $B(x, y)$ is

$$x, y \to B(x, y) \quad \text{becomes} \quad [x, x'], [y, y'] \to [B(x, y), B_x(x, y)\, x' + B_y(x, y)\, y'],$$

where B_x and B_y denote the partial derivatives of B with respect to x and y.

Armed with this machinery, we can now combine elementary operations to calculate any more complicated function that can be expressed as a sequence of such operations—which means essentially any function we can evaluate on a computer. For instance, suppose we want to calculate the function $f(x) = 3x^2 + \mathrm{e}^x$. We could do this by performing the following sequence of operations:

$$u_1 = x^2, \tag{5.168a}$$
$$u_2 = 3u_1, \tag{5.168b}$$
$$u_3 = \mathrm{e}^x, \tag{5.168c}$$

then

$$f(x) = u_2 + u_3. \tag{5.169}$$

With automatic differentiation, each mathematical operation—raising to the power of 2, multiplying by 3, taking the exponential, and adding the two final terms—would be performed using the generalized operations listed above and the end result would be a pair $[f, f']$, representing the value of f and its derivative.

Now here is the final trick: to calculate the derivative $f'(t)$ at any value of t, we simply evaluate our function $f(x)$ with argument $[t, 1]$ for whatever value of t we are interested in. By definition this is the correct value/derivative pair for the independent variable t, because $\mathrm{d}t/\mathrm{d}t = 1$ always. So if we use this pair as the argument of our function, we will get out a pair representing the value of the function $f(t)$ and its derivative $f'(t)$, at the chosen value of t.

Example 5.5: Automatic differentiation

An example should help to make this clear. Let us store our value/derivative pairs as two-element lists in Python of the form `[value,derivative]`. We begin by defining some functions to perform basic mathematical operations on these lists:

File: `autodiff.py`

```
from math import exp

def multiply(x,c):
    vx,dx = x
    return [c*vx,c*dx]
```

```
def power(x,c):
    vx,dx = x
    return [vx**c,c*vx**(c-1)*dx]

def exponential(x):
    vx,dx = x
    return [exp(vx),exp(vx)*dx]

def add(x,y):
    vx,dx = x
    vy,dy = y
    return [vx+vy,dx+dy]
```

Each function takes one or two pairs, plus potentially some other arguments, and returns a single pair as result. Now we can combine these elementary operations to calculate a more complicated function, such as our function $f(x) = 3x^2 + \mathrm{e}^x$:

```
def f(x):
    u1 = power(x,2)
    u2 = multiply(u1,3)
    u3 = exponential(x)
    return add(u2,u3)
```

Now we call this function with argument $[t, 1]$ to compute the derivative at t. For instance, to calculate the derivative of $f(t)$ at $t = \frac{1}{2}$ we could write:

```
print(f([0.5,1]))
```

If we run the whole program, it prints:

```
[2.398721270700128, 4.648721270700128]
```

The first number is $f\left(\frac{1}{2}\right)$ and the second is $f'\left(\frac{1}{2}\right)$. We can check the results:

$$f(t) = 3t^2 + \mathrm{e}^t = \tfrac{3}{4} + \mathrm{e}^{1/2} = 2.3987\ldots, \tag{5.170}$$

$$f'(t) = 6t + \mathrm{e}^t = 3 + \mathrm{e}^{1/2} = 4.6487\ldots, \tag{5.171}$$

so our program has indeed correctly found both the value of the function and its derivative.

Note that there is no approximation involved in calculating derivatives this way, other than the rounding error inherent in all computer calculations. The answers we get are accurate to the limits of machine precision. The method only applies, as we have said, to functions that can be calculated on the computer—we cannot apply this kind of differentiation to lab data for example. But it is applicable in many situations in computational physics and in science and engineering more broadly.

Automatic differentiation can also be extended to the calculation of higher derivatives by keeping track of more derivatives for each variable. For instance, if we are interested in the second derivative, then each variable x gets replaced by a three-valued triple $[x, x', x'']$, and the general rule for unary operations $U(x)$ is

$$x \to U(x) \quad \text{becomes} \quad [x, x', x''] \to [U(x), U'(x)\,x', U''(x)\,(x')^2 + U'(x)\,x''],$$

while the rule for binary operations $B(x, y)$ is

$$\begin{aligned} x, y \to B(x, y) \quad \text{becomes} \quad & [x, x', x''], [y, y', y''] \to [B(x, y), B_x(x, y)\,x' + B_y(x, y)\,y', \\ & B_{xx}(x, y)\,(x')^2 + 2B_{xy}(x, y)\,x'y' + B_{yy}(x, y)\,(y')^2 + B_x(x, y)\,x'' + B_y(x, y)\,y''], \end{aligned}$$

where B_x and B_y are first derivatives of $B(x, y)$ with respect to its two arguments and B_{xx}, B_{xy}, and B_{yy} are second derivatives.

Implementing automatic differentiation does require writing many individual user-defined functions to perform elementary mathematical operations, like the ones in Example 5.5 above. Because of the wide use of automatic differentiation in both science and commercial applications, however, people have created Python packages that define large collections of such functions for you, to save you the effort. If you plan to use automatic differentiation extensively, you may want to look at these packages. Examples include TensorFlow, PyTorch, and JAX.

Exercise 5.17: In Exercise 5.13 we encountered the Hermite polynomials, which are defined iteratively by

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x), \quad \text{with } H_0(x) = 1 \text{ and } H_1(x) = 2x.$$

a) Write a Python function `H(n,x)` to compute the value and derivative of the nth Hermite polynomial at the point x using automatic differentiation.

b) Use your function to verify the values of the derivatives $H_3'(\frac{1}{2}) = -6$ and $H_{10}'(\frac{1}{2}) = 129\,620$.

c) Make a plot of the derivative of $H_{100}(x)$ between $x = -2$ and $x = 2$.

Hint: If you program is taking a long time to run, then you are probably doing the calculation the wrong way. It should only take a second or two to finish.

5.12 INTERPOLATION

We tackle one more topic in this chapter, namely *interpolation*, which is not directly related to integrals and derivatives, but uses similar mathematical methods, making this a good moment to look at it.

Suppose you are given the value of a function $f(x)$ at two points $x = a, b$ and you want to estimate the value at another point x in between. There are a number of

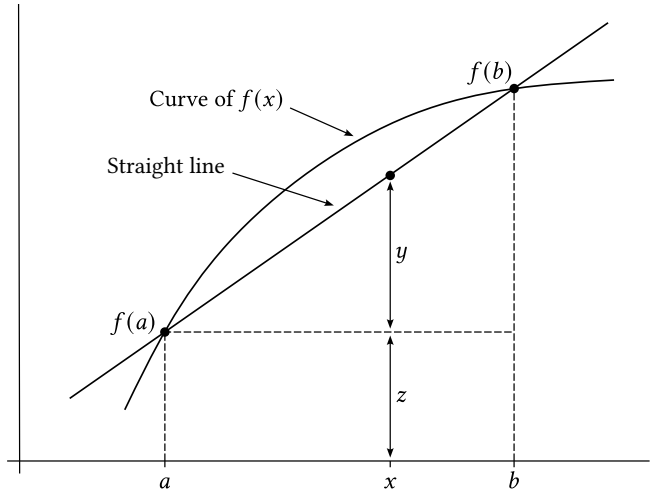


Figure 5.14: Linear interpolation. The value of $f(x)$ in between the two known points at $x = a$ and $x = b$ is estimated by assuming a straight line from $f(a)$ to $f(b)$.

ways of making such estimates, of which the simplest is *linear interpolation*, which is illustrated in Fig. 5.14. We assume our function follows a straight line from $f(a)$ to $f(b)$, which in most cases is an approximation—likely the function follows some sort of curve—but if we make this assumption then we can calculate $f(x)$ with some elementary geometry.

The slope of the straight-line approximation is

$$m = \frac{f(b) - f(a)}{b - a}, \tag{5.172}$$

and the distance marked y on the figure is given in terms of this slope by $y = m(x-a)$. The distance marked z is equal to $f(a)$, so

$$\begin{aligned} f(x) \simeq y + z &= \frac{f(b) - f(a)}{b - a}(x - a) + f(a) \\ &= \frac{(b - x)f(a) + (x - a)f(b)}{b - a}. \end{aligned} \tag{5.173}$$

This is the fundamental formula of linear interpolation. In fact, this same formula can also be used to *extrapolate* the function to points outside the interval from a to b, although one should not extrapolate too far. The further you go, the less likely it is that the extrapolation will be accurate.

Another way to look at the linear interpolation formula is to define two positive weights

$$u_1 = b - x, \qquad u_2 = x - a, \tag{5.174}$$

in terms of which, Eq. (5.173) can be written as

$$f(x) \simeq \frac{u_1 f(a) + u_2 f(b)}{u_1 + u_2}. \tag{5.175}$$

In other words, linear interpolation is equivalent to taking a weighted average of the two values $f(a)$ and $f(b)$, with weights that depend on the distances between the point x and the ends of the interval a, b. As we will see in the following discussion, all interpolation formulas eventually boil down to some kind of weighted average over the known values of the function.

How accurate is the linear interpolation formula? The calculation of the error is similar to that for derivatives, making use of two Taylor expansions:

$$f(a) = f(x) + (a-x)f'(x) + \tfrac{1}{2}(a-x)^2 f''(x) + \ldots \tag{5.176}$$

$$f(b) = f(x) + (b-x)f'(x) + \tfrac{1}{2}(b-x)^2 f''(x) + \ldots \tag{5.177}$$

Multiplying the first of these by $b-x$ and the second by $a-x$, then subtracting one from the other, the terms in $f'(x)$ cancel and we get

$$(b-x)f(a) + (x-a)f(b) = (b-a)f(x) + \tfrac{1}{2}(a-x)(b-x)(a-b)f''(x) + \ldots, \tag{5.178}$$

which can be rearranged to read

$$f(x) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a} + (a-x)(b-x)\,f''(x) + \ldots \tag{5.179}$$

The first term on the right-hand side is our linear interpolation formula; the rest of the terms are the error. Note that the leading-order error term vanishes as x tends to either a or b, so that either $b-x$ or $a-x$ becomes small. And, assuming $f''(x)$ varies slowly, the error will be largest in the middle of the interval. If we denote the width of the interval by $b-a=h$, then when we are in the middle we have $x-a = b-x = \frac{1}{2}h$ and the absolute magnitude of the leading-order error is $\frac{1}{4}h^2\,|f''(x)|$. Thus, like the central difference formula for a first derivative, the worst-case error on a linear interpolation is $\mathrm{O}(h^2)$, and we can make the interpolation more accurate by making h smaller.

By contrast with the case of derivatives, however, we do not need to be particularly careful about rounding error when using linear interpolation. The interpolation formula, Eq. (5.173), involves the *sum* of values of $f(x)$ at two closely spaced points, not the difference, so we don't normally run into the accuracy problems that plague calculations based on subtractions (like calculations of derivatives).

Can we do better than linear interpolation? Not if we know the value of the function $f(x)$ at only two points—there is no better approximation in that case. If we know the function at more than two points there are several ways to improve on linear interpolation. The most obvious is to interpolate with higher-order polynomials, using the method known as *Lagrange interpolation*. If we have three points,

for instance, we can fit a quadratic through them, which will usually give a better match to the underlying curve. Four points allows us to fit a cubic, and so on. A straightforward way to fit such polynomials is using the Lagrange interpolating polynomials $\phi_k(x)$ of Eq. (5.53) on page 155, which for N points at $x_1 \ldots x_N$ are a set of N polynomials, each of degree $N-1$. In Eq. (5.56) and the accompanying discussion we showed that the unique polynomial of degree $N-1$ that fits the function $f(x)$ at all of the N points is given by

$$\Phi(x) = \sum_{k=1}^{N} f(x_k)\phi_k(x), \tag{5.180}$$

and we can use the value of this function as our interpolation.

Note that for the special case where $f(x_k) = 1$ for all k, the unique polynomial that goes through all the points is trivially just $\Phi(x) = 1$, and hence (5.180) tells us that $\sum_{k=1}^{N} \phi_k(x) = 1$ for all x: the sum of the Lagrange polynomials at any x is always 1. Given this fact, we can see that Eq. (5.180) again takes the form of a weighted average over the known values $f(x_k)$ with weights $\phi_k(x)$ that sum to one.

When the number of points becomes large, however, the Lagrange interpolation approach breaks down. If we have a large number N of points then you might think the best thing to do would be to fit an $(N-1)$th order polynomial through them, but it turns out this does not work because very high order polynomials tend to have a lot of wiggles in them[4] and can deviate from the fitted points badly in the intervals between points. It is better in this case to fit many lower-order polynomials such as quadratics or cubics to smaller sets of adjacent points. Unfortunately, the naive implementation of such a scheme gives rather uneven interpolations because the slope of the interpolation changes at the join-points between polynomials. A more satisfactory approach is to fit polynomials to the measured points and the derivatives at their ends, so that one gets a function that goes through the points and has a smooth slope everywhere. Such interpolations are called *splines*. The most widely used type are *cubic splines*. We will not need these methods in this book however, so we will go into them further here.

5.12.1 Interpolation in two or more dimensions

Sometimes we need to interpolate in more than one dimension. For instance, we might have an image, such as an astronomical image, represented as a grid of pixels of varying intensity, and we want to estimate what the intensity is between the pixels. We can do this using *bilinear interpolation*.

We have a function $f(x, y)$ defined in a two-dimensional space and we know its value on the points of a square or rectangular grid. Figure 5.15 shows a sketch of one

[4] This is primarily a problem when the sample points are evenly spaced. For unevenly spaced points it may be possible to get good results using high-degree polynomials.

of the rectangles in the grid and let us write the values of the function at the four corners as

$$f_{11} = f(x_1, y_1), \quad f_{21} = f(x_2, y_1), \quad f_{12} = f(x_1, y_2), \quad f_{22} = f(x_2, y_2). \tag{5.181}$$

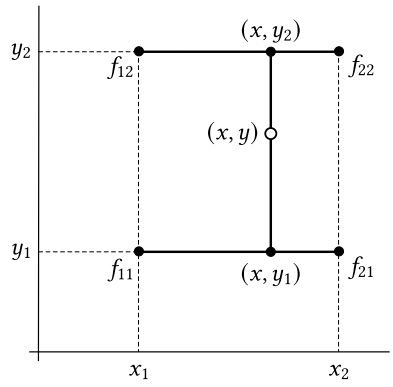


Figure 5.15: Bilinear interpolation. In bilinear interpolation we interpolate first along the top and bottom lines of a rectangle and then vertically between the resulting two points.

Our goal is to estimate the value of the function at a general point (x, y) inside the rectangle. The idea of bilinear interpolation is that we first interpolate linearly along the two horizontal lines at top and bottom of the rectangle to estimate the values at the points (x, y_1) and (x, y_2), and then interpolate vertically between those two points to estimate $f(x, y)$.

Thus, using our linear interpolation formula, Eq. (5.173), we have

$$f(x, y_1) = \frac{(x_2 - x)f_{11} + (x - x_1)f_{21}}{x_2 - x_1}, \tag{5.182a}$$

$$f(x, y_2) = \frac{(x_2 - x)f_{12} + (x - x_1)f_{22}}{x_2 - x_1}, \tag{5.182b}$$

and then

$$\begin{aligned} f(x, y) &= \frac{(y_2 - y)f(x, y_1) + (y - y_1)f(x, y_2)}{y_2 - y_1} \\ &= \frac{u_{11}f_{11} + u_{21}f_{21} + u_{12}f_{12} + u_{22}f_{22}}{u_{11} + u_{21} + u_{12} + u_{22}}, \end{aligned} \tag{5.183}$$

where

$$u_{11} = (x_2 - x)(y_2 - y), \tag{5.184a}$$

$$u_{21} = (x - x_1)(y_2 - y), \tag{5.184b}$$

$$u_{12} = (x_2 - x)(y - y_1), \tag{5.184c}$$

$$u_{22} = (x - x_1)(y - y_1). \tag{5.184d}$$

Thus, as with our previous interpolation formulas, our estimate of $f(x, y)$ is a weighted average of the known values of the function.

One might imagine that there are two different ways to perform bilinear interpolation, either by interpolating horizontally first then vertically, or *vice versa*. In fact, however, both of these lead to same formula, given above, and hence there is only one way to do the calculation.

A different type of two-dimensional interpolation arises when the known values of the function are not on a rectilinear grid, but are arranged in some other, possibly irregular fashion. In this case one can interpolate between trios of points forming triangles. Any location $\mathbf{r}$ within the area covered by the known points falls inside (or on the edge of) at least one such triangle, and when it does it divides the triangle

into three smaller triangles with areas u_1, u_2, u_3 as shown in Fig. 5.16. We define the interpolated value of the function $f(\mathbf{r})$ within the triangle by

$$f(\mathbf{r}) = \frac{u_1 f_1 + u_2 f_2 + u_3 f_3}{u_1 + u_2 + u_3}, \tag{5.185}$$

where f_1, f_2, f_3 are the values at the corners. Again this takes the form of a weighted average. It also takes the values f_1, f_2, and f_3, as it should, at the three corners of the large triangle, and it interpolates linearly between them when we are within the triangle.

The areas of the three small triangles are most easily calculated by vector methods. The area of any triangle is given in terms of the lengths a, b of any two of its edges and the angle θ between them by $\frac{1}{2}ab\sin\theta = \frac{1}{2}|\mathbf{a}\times\mathbf{b}|$, where $\mathbf{a}$ and $\mathbf{b}$ are the vectors along the edges and the $\times$ symbol represents the vector cross-product. The factor of $\frac{1}{2}$ is not important—we can multiply u_1, u_2, and u_3 by 2 and Eq. (5.185) is unaffected—so let us drop the $\frac{1}{2}$. Then the value of u_1 can be calculated from the cross-product of the edge vectors $\mathbf{r}-\mathbf{r}_3$ and $\mathbf{r}_2-\mathbf{r}_3$ thus:

$$u_1 = |(\mathbf{r}-\mathbf{r}_3)\times(\mathbf{r}_2-\mathbf{r}_3)| = |\mathbf{r}\times(\mathbf{r}_2-\mathbf{r}_3)+\mathbf{r}_2\times\mathbf{r}_3|, \tag{5.186}$$

where we have made use of $\mathbf{r}_3\times\mathbf{r}_2 = -\mathbf{r}_2\times\mathbf{r}_3$ and $\mathbf{r}_3\times\mathbf{r}_3 = 0$. For a fixed grid of points, the quantities $\mathbf{r}_2-\mathbf{r}_3$ and $\mathbf{r}_2\times\mathbf{r}_3$ can be calculated ahead of time and stored to allow quick interpolation, and we can write similar expressions for u_2 and u_3:

$$u_2 = |\mathbf{r}\times(\mathbf{r}_3-\mathbf{r}_1)+\mathbf{r}_3\times\mathbf{r}_1|, \tag{5.187}$$

$$u_3 = |\mathbf{r}\times(\mathbf{r}_1-\mathbf{r}_2)+\mathbf{r}_1\times\mathbf{r}_2|. \tag{5.188}$$

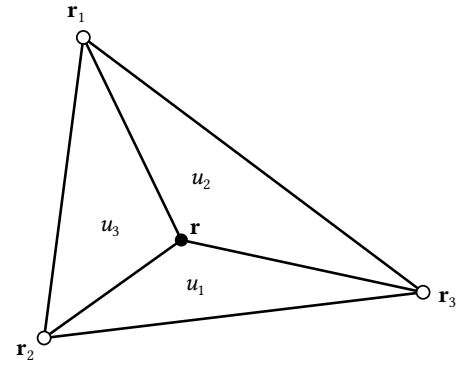


Figure 5.16: Weights for interpolation within a triangle. Any point $\mathbf{r}$ within a triangle divides the triangle into three smaller ones, whose areas u_1, u_2, u_3 can be used as weights for interpolation.

The cross-product can be calculated in Python using the `cross` function from `numpy`. The cross-product of a pair of two-dimensional vectors in the xy plane is always in the z-direction, so when given two-dimensional vectors as inputs the `cross` function returns only a single floating-point value, equal to the z-component of the cross-product. For example, we can calculate the area of a triangle with corners at $(0,0)$, $(5,0)$, and $(2,4)$ thus:

```
from numpy import array,cross

a = array([5,0],float)
b = array([2,4],float)
print(abs(cross(a,b))/2)
```

Note the use of the `abs` function here, since we want the magnitude of the cross-product. When we run this program it correctly prints "`10.0`".

Though it may not be immediately obvious, this triangular interpolation is a linear interpolation, since Eqs. (5.186) to (5.188) are linear in $\mathbf{r}$. An alternative, geometric way to look at the interpolation is to think of $f(\mathbf{r})$ as defining the height of a two-dimensional surface as a function of position (x, y) and the known points at the corners of the triangles as points on this surface. Then our interpolation scheme is equivalent to interpolating within each triangle using the flat plane that passes exactly through the points at the three corners of the triangle. The latter is clearly a linear interpolation, and since there is only one plane that passes through the corners, it is the unique linear interpolation that takes the correct values at the corners. The interpolation defined in Eq. (5.185) is also linear and also takes the correct values at the corners, hence the two interpolations must be the same. Triangular linear interpolation, viewed in this geometric way, forms one of the mathematical pillars of computer animation and video games, where surfaces are commonly represented by triangular meshes and the surface itself approximated using interpolation.

Both bilinear interpolation and triangular interpolation can be generalized to higher dimensions. The generalization of bilinear interpolation in three dimensions, for example, is a natural one: you interpolate linearly first along the x axis, then the y axis, and finally the z axis. The resulting formulas are an obvious extension of Eqs. (5.183) and (5.184). The generalization of the triangular interpolation to three dimensions involves interpolating within tetrahedra and is analogous to the two-dimensional version. Any point within a tetrahedron divides it into four smaller tetrahedra, whose areas can be used as the weights in a weighted average over the values of the function at the four corners.

We will not need these methods in this book, however. For our purposes, two dimensions will be enough.

CHAPTER SUMMARY

- The **trapezoidal rule** is the simplest of methods for evaluating integrals on a computer. It approximates a function with straight-line segments and then calculates the area underneath those straight lines.
- For sample points spaced a distance h apart the trapezoidal rule gives an answer accurate to order h, with an error of order h^2.
- **Simpson's rule** approximates the function using quadratics instead of straight lines and usually gives a more accurate answer than the trapezoidal rule. It is accurate to order h^3 with an error of order h^4.
- **Adaptive versions** of the trapezoidal rule and Simpson's rule allow you to compute an answer to a required accuracy by increasing the number of sample points until you hit the desired target.

- **Romberg integration** is an extension of the trapezoidal rule that uses **Richardson extrapolation** to increase the accuracy of integrals, in many cases substantially improving on the accuracy of either the trapezoidal rule or Simpson's rule.
- **Gaussian quadrature** abandons equally spaced sample points and creates an integration rule that is superbly accurate, and also simple to use, but at the expense of unevenly spaced points.
- Integrals over **infinite ranges** cannot be performed directly on a computer, but they can be done by first making a change of variables so that the infinite range becomes a finite one.
- **Multiple integrals** can be performed by nesting integrals one inside another. This can become computationally demanding for high-dimensional integrals. Integrals in more than three or four dimensions can be challenging.
- **Derivatives** can be calculated numerically using **forward**, **backward**, or **central differences**. Central differences are generally more accurate than forward or backward ones.
- More accurate values for derivatives can be calculated using a range of **higher-order approximations** or by applying **Richardson extrapolation** again.
- **Second derivatives** and other higher derivatives are calculated as derivatives of derivatives.
- An alternative approach to calculating derivatives is **automatic differentiation**, which allows one to calculate exact values (apart from rounding error), but is limited to derivatives of functions that can be evaluated on the computer. It cannot be used, for example, to differentiate experimental measurements.
- **Interpolation** is the process of estimating the value of a function between known sample points. The simplest approach is **linear interpolation**, which assumes a straight line between points.
- Higher-order interpolations can be calculated using **Lagrange interpolating polynomials**, although caution should be used when using polynomials of high degree, which can lead to numerical instability.
- **Interpolation in two or more dimensions** can be performed using bilinear interpolation within rectangles or linear interpolation within triangles (or the equivalent in three or more dimensions).

FURTHER EXERCISES

5.18 The value of π: It can be shown that

$$\int_0^\infty \frac{dx}{x^2+1} = \frac{\pi}{2}.$$

Write a program to calculate the value of π to 10 decimal places using this result. Use an adaptive integration method that guarantees the required accuracy.

5.19 The gamma function: A commonly occurring function in physics calculations is the gamma function $\Gamma(a)$, which is defined by the integral

$$\Gamma(a) = \int_0^\infty x^{a-1}e^{-x}\,dx.$$

There is no closed-form expression for the gamma function, but one can calculate its value for given a by performing the integral above numerically. You have to be careful how you do it, however, if you wish to get an accurate answer.

a) Write a program to make a graph of the value of the integrand $x^{a-1}e^{-x}$ as a function of x from $x = 0$ to $x = 5$, with three separate curves for $a = 2$, 3, and 4, all on the same axes. You should find that the integrand starts at zero, rises to a maximum, and then decays again for each curve.

b) Show analytically that the maximum falls at $x = a - 1$.

c) Most of the area under the integrand falls near the maximum, so to get an accurate value of the gamma function we need to do a good job of this part of the integral. We can change the integral from 0 to ∞ to one over a finite range from 0 to 1 using the change of variables in Eq. (5.97), but this tends to squash the peak towards the edge of the $[0, 1]$ range and does a poor job of evaluating the integral accurately. We can do a better job by making a different change of variables that puts the peak in the middle of the integration range, around $\frac{1}{2}$. We will use the change of variables

$$z = \frac{x}{c+x}.$$

For what value of x does this change of variables give $z = \frac{1}{2}$? Hence what is the appropriate choice of the parameter c that puts the peak of the integrand for the gamma function at $z = \frac{1}{2}$?

d) Before we can calculate the gamma function, there is another detail we need to attend to. The integrand $x^{a-1}e^{-x}$ can be difficult to evaluate because the factor x^{a-1} can become very large and the factor e^{-x} very small, causing numerical overflow or underflow, or both, for some values of x. Write $x^{a-1} = e^{(a-1)\ln x}$ to derive an alternative expression for the integrand that does not suffer from these problems (or at least not so much). Explain why your new expression is better than the old one.

e) Now, using the change of variables above and the value of c you have chosen, write a user-defined function `gamma(a)` to calculate the gamma function for arbitrary argument a. Use whatever integration method you feel is appropriate. Test your function by using it to calculate and print the value of $\Gamma(\frac{3}{2})$, which is known to be equal to $\frac{1}{2}\sqrt{\pi} \simeq 0.886$.

f) For integer values of a it can be shown that $\Gamma(a)$ is equal to the factorial of $a-1$. Use your Python function to calculate $\Gamma(3)$, $\Gamma(6)$, and $\Gamma(10)$. You should get answers closely equal to $2! = 2$, $5! = 120$, and $9! = 362\,880$.

5.20 Gauss's law: Consider a function $f(x,y,z)$ in three-dimensional space. The integral I of that function over a sphere of radius R centered on the origin can be calculated by writing x, y, z in spherical polar coordinates $x = R\sin\theta\sin\phi$, $y = R\sin\theta\cos\phi$, $z = R\cos\theta$ and then using

$$I = \int_0^{2\pi}\int_0^{\pi} f(x,y,z)\,R^2\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi.$$

a) Suppose we have three electric charges of 1 coulomb each at positions $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ in otherwise empty space. Write an expression for the electric field $\mathbf{E}$ that results from these three charges at an arbitrary point $\mathbf{r}$. Hence show that the radial component E_r of that field, meaning the component in the direction directly away from the origin, is given by

$$E_r = \frac{1}{4\pi\epsilon_0}\,\frac{\mathbf{r}}{|\mathbf{r}|}\cdot\left[\frac{\mathbf{r}-\mathbf{r}_1}{|\mathbf{r}-\mathbf{r}_1|^3} + \frac{\mathbf{r}-\mathbf{r}_2}{|\mathbf{r}-\mathbf{r}_2|^3} + \frac{\mathbf{r}-\mathbf{r}_3}{|\mathbf{r}-\mathbf{r}_3|^3}\right],$$

where ϵ_0 is the permittivity of the vacuum.

b) Let $f(x,y,z) = E_r$. Write a program to evaluate the integral I defined above for this choice of f in the case where

$$\mathbf{r}_1 = (0,0,0), \qquad \mathbf{r}_2 = (1,0,0), \qquad \mathbf{r}_3 = (0,1,0),$$

and the radius of the sphere is $R = 2$. Use whatever integration method you feel is appropriate.

c) Gauss's law tells us that the integral of the radial component of the field should be equal to the total charge inside the sphere divided by the permittivity ϵ_0. Verify that your integral gives the correct value.

d) Try a few other positions for the three charges, some inside the sphere and some outside, and recompute the integral. Verify that Gauss's law is obeyed in each case.

5.21 Rearranging Eq. (5.19) on page 143 into a slightly more conventional form, we have:

$$\int_a^b f(x)\,\mathrm{d}x = h\left[\tfrac{1}{2}f(a) + \tfrac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a+kh)\right] + \tfrac{1}{12}h^2\bigl[f'(a) - f'(b)\bigr] + \mathrm{O}(h^4).$$

This result gives a value for the integral on the left which has an error of order h^4—a factor of h^2 better than the error on the trapezoidal rule and as good as Simpson's rule. We can use this formula as a new rule for evaluating integrals, distinct from any of the others we have seen in this chapter. We might call it the "Euler–Maclaurin rule."

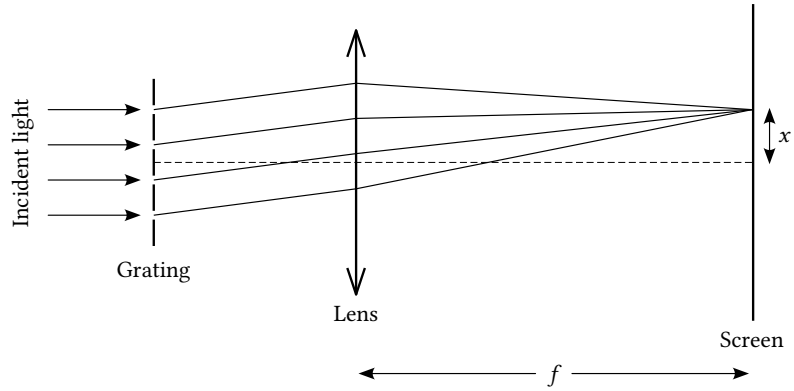
a) Write a program to calculate the value of the integral $\int_0^2 (x^4 - 2x + 1)\,\mathrm{d}x$ using this formula. (This is the same integral that we studied in Example 5.1, whose true value is 4.4.) The order-h term in the formula is just the ordinary trapezoidal rule; the h^2 term involves the derivatives $f'(a)$ and $f'(b)$, which you should evaluate using central differences, centered on a and b respectively. Note that the size of the interval you use for calculating the central differences does not have to equal the value of h used in the

trapezoidal rule part of the calculation. An interval of about 10^{-5} gives good values for the central differences.

Use your program to evaluate the integral with $N = 10$ slices and compare the accuracy of the result with that obtained from the trapezoidal rule alone with the same number of slices.

b) Good though it is, this integration method is not much used in practice. Suggest a reason why not.

5.22 Diffraction gratings: Light with wavelength λ is incident on a diffraction grating of total width w, gets diffracted, is focused with a lens of focal length f, and falls on a screen:



Theory tells us that the intensity of the diffraction pattern on the screen, a distance x from the central axis of the system, is given by

$$I(x) = \left| \int_{-w/2}^{w/2} \sqrt{q(u)}\, e^{i2\pi xu/\lambda f}\, du \right|^2,$$

where $q(u)$ is the *intensity transmission function* of the diffraction grating at a distance u from the central axis, defined as the fraction of the incident light that the grating lets through at that point.

a) Consider a grating with transmission function $q(u) = \sin^2 \alpha u$. What is the separation of the "slits" in this grating, expressed in terms of α?

b) Write a Python function `q(u)` that returns the transmission function $q(u) = \sin^2 \alpha u$ as above at position u for a grating whose slits have separation 20 μm.

c) Use your function in a program to calculate and graph the intensity of the diffraction pattern produced by such a grating having ten slits in total, if the incident light has wavelength $\lambda = 500$ nm. Assume the lens has a focal length of 1 meter and the screen is 10 cm wide. You can use whatever method you think appropriate for doing the integral. Once you have made your choice you will also need to decide the number of sample points you will use. What criteria play into this decision?

Notice that the integrand in the equation for $I(x)$ is complex, so you will have to use complex variables in your program. As mentioned in Section 2.2.5, there is a version of the `math` package for use with complex variables called `cmath`. In particular you may find the `exp` function from `cmath` useful because it can calculate exponentials of complex arguments.

d) Create a visualization of how the diffraction pattern would look on the screen using a density plot (see Section 3.3). Your plot should look something like this:

e) Modify your program further to make pictures of the diffraction patterns produced by gratings with the following profiles:

i) A transmission profile that obeys $q(u) = \sin^2 \alpha u \, \sin^2 \beta u$, with α as before and the same total grating width w, and $\beta = \frac{1}{2}\alpha$.

ii) Two "square" slits, meaning slits with 100% transmission through the slit and 0% transmission everywhere else. Calculate the diffraction pattern for non-identical slits, one 10 μm wide and the other 20 μm wide, with a 60 μm gap between the two.

5.23 A more advanced adaptive method for the trapezoidal rule: In Section 5.3 we studied an adaptive version of the trapezoidal rule in which the number of steps is increased—and the width h of the slices correspondingly decreased—until the calculation gives a value for the integral accurate to some desired level. Although this method varies h, it still calculates the integral at any individual stage of the process using slices of equal width throughout the domain of integration. In this exercise we look at a more sophisticated adaptive method that uses different step sizes in different parts of the domain, which can be useful particularly for poorly behaved functions that vary rapidly in certain regions but not others. Remarkably, this method is not much more complicated to program than the ones we have already seen, if one knows the right tricks. Here is how the method works.

Suppose we wish to evaluate the integral $I = \int_a^b f(x)\,\mathrm{d}x$ and we want an error of no more than ϵ on our answer. To put that another way, if we divide up the integral into slices of width h then we require an accuracy per slice of

$$h\,\frac{\epsilon}{b-a} = h\delta,$$

where $\delta = \epsilon/(b-a)$ is the target accuracy per unit interval.

We start by evaluating the integral using the trapezoidal rule with just a single slice of width $h_1 = b - a$. Let us call the estimate of the integral from this calculation I_1. Usually I_1 will not be very accurate, but that does not matter. Next we make a second estimate I_2 of the integral, again using the trapezoidal rule but now with two slices of width $h_2 = \frac{1}{2}h_1$ each. Equation (5.28) tells us that the error on this second estimate is $\frac{1}{3}(I_2 - I_1)$ to leading order. If the absolute value of this error is smaller than the required accuracy ϵ then our calculation is complete and we need go no further. I_2 is a good enough estimate of the integral.

Most likely, however, this will not be the case; the accuracy will not be good enough. If so, then we divide the integration interval into two equal parts of size $\frac{1}{2}(b-a)$ each, and we repeat the process above in each part separately, calculating estimates I_1 and I_2 using one and two slices respectively, estimating the error, and checking to see if it is less than the required accuracy, which is now $\frac{1}{2}(b-a)\delta = \frac{1}{2}\epsilon$.

We keep on repeating this process, dividing each slice in half and in half again, as many times as necessary to achieve the desired accuracy in every slice. Different slices may be divided different numbers of times, and hence we may end up with different sized slices in

different parts of the integration domain. The method automatically uses whatever size and number of slices is appropriate in each region.

Write a program using this method to calculate the integral

$$I = \int_0^{10} \frac{\sin^2 x}{x^2}\,dx,$$

to an accuracy of $\epsilon = 10^{-4}$. You should carry out the following steps.

a) Start by writing a function to calculate the integrand $f(x) = (\sin^2 x)/x^2$. Note that the limiting value of the integrand at $x = 0$ is 1. You will probably have to include this point as a special case in your function using an if statement.

b) The best way to perform the integration itself is to use recursion, the ability of a Python function to call itself—see Section 2.6.1. Write a function `step(x1,x2,f1,f2)` that takes as arguments the beginning and end points x_1, x_2 of a slice and the values $f(x_1), f(x_2)$ of the integrand at those two points, and returns the value of the integral from x_1 to x_2. This function should evaluate the two estimates I_1 and I_2 of the integral from x_1 to x_2, calculated with one and two slices respectively, and the error $\frac{1}{3}(I_2 - I_1)$. If this error meets the target value, which is $(x_2 - x_1)\delta$, then the calculation is complete and the function simply returns the value I_2. If the error fails to meet the target, then the function calls itself, twice, to evaluate the integral separately on the first and second halves of the interval and returns the sum of the two results. (And then *those* functions can call themselves, and so forth, subdividing the integral as many times as necessary to reach the required accuracy.)

c) As icing on the cake, when the error target is met and the function returns a value for the integral in the current slice, it can, in fact, return a slightly better value than the estimate I_2. Since you will already have calculated the value of the integrand $f(x)$ at x_1, x_2, and the midpoint $x_m = \frac{1}{2}(x_1 + x_2)$ in order to evaluate I_2, you can use those results to compute the improved Simpson's rule estimate, Eq. (5.7), for this slice. You just return the value $\frac{1}{6}h[f(x_1)+4f(x_m)+f(x_2)]$ instead of the trapezoidal rule estimate $\frac{1}{4}h[f(x_1)+2f(x_m)+f(x_2)]$ (where $h = x_2 - x_1$). This involves very little extra work, but gives a value that is more accurate by two orders in h. (Technically, this is an example of "local extrapolation," although it is perhaps not obvious what we are extrapolating in this case. We will discuss local extrapolation again when we study adaptive methods for the solution of differential equations in Section 8.4.)

d) Why does the function `step(x1,x2,f1,f2)` take not only the positions x_1 and x_2 as arguments, but also the values $f(x_1)$ and $f(x_2)$? Since we know the function $f(x)$, we could just calculate these values from x_1 and x_2. Nonetheless, it is a smart move to include the values of $f(x_1)$ and $f(x_2)$ as arguments to the function. Why?

e) Modify your program to make a plot of the integrand with dots added showing where the edges of each integration slice lie. You should see larger slices in portions of the integrand that follow reasonably straight lines (because the trapezoidal rule gives an accurate value for straight-line integrands) and smaller slices in portions with more curvature.

5.24 Electric field of a charge distribution: Suppose we have a distribution of charges and we want to calculate the resulting electric field. One way to do this is to first calculate the

electric potential ϕ and then take its gradient. For a point charge q at the origin, the electric potential at a distance r from the origin is $\phi = q/4\pi\epsilon_0 r$ and the electric field is $\mathbf{E} = -\nabla\phi$.

a) You have two charges, of ± 1 C, 10 cm apart. Write a program to calculate the resulting electric potential on a 1 m × 1 m square plane surrounding the charges and passing through them. Calculate the potential at 1 cm spaced points in a grid and make a visualization on the screen of the potential using a density plot.

b) Now calculate the partial derivatives of the potential with respect to x and y and hence find the electric field in the xy plane. Make a visualization of the field also. This is a little trickier than visualizing the potential, because the electric field has both magnitude and direction. One way to do it might be to make two density plots, one for the magnitude, and one for the direction, the latter using the "`hsv`" color scheme in `pyplot`, which is a rainbow scheme that passes through all the colors but starts and ends with the same shade of red, which makes it suitable for representing things like directions or angles that go around the full circle and end up where they started. A more sophisticated visualization might use the `quiver` function from `pyplot`, which draws a grid of arrows with direction and length that you specify.

5.25 Differentiating by integrating: If you are familiar with the calculus of complex variables, you may find the following technique useful and interesting.

Suppose we have a function $f(z)$ whose value we know not only on the real line but also for complex values of its argument. Then we can calculate derivatives of that function at any point z_0 by performing a contour integral, using the Cauchy derivative formula:

$$\left(\frac{\mathrm{d}^m f}{\mathrm{d}z^m}\right)_{z=z_0} = \frac{m!}{2\pi\mathrm{i}} \oint \frac{f(z)}{(z-z_0)^{m+1}}\,\mathrm{d}z,$$

where the integral is performed counterclockwise around any contour in the complex plane that surrounds the point z_0 but contains no poles in $f(z)$. Since numerical integration is significantly easier and more accurate than numerical differentiation, this formula provides us with a method for calculating derivatives—and especially multiple derivatives—accurately by turning them into integrals.

Suppose, for example, that we want to calculate derivatives of $f(z)$ at $z = 0$. Let us apply the Cauchy formula above using the trapezoidal rule to calculate the integral along a circular contour centered on the origin with radius 1. The trapezoidal rule will be slightly different from the version we are used to because the value of the interval h is now a complex number, and moreover is not constant from one slice of the integral to the next—it stays constant in modulus, but its argument changes from one slice to another.

We will divide our contour integral into N slices with sample points z_k distributed uniformly around the circular contour at positions $z_k = \mathrm{e}^{\mathrm{i}2\pi k/N}$ for $k = 0 \ldots N$. Then the distance between consecutive sample points is

$$h_k = z_{k+1} - z_k = \mathrm{e}^{\mathrm{i}2\pi(k+1)/N} - \mathrm{e}^{\mathrm{i}2\pi k/N},$$

and, introducing the shorthand $g(z) = f(z)/z^{m+1}$ for the integrand, the trapezoidal rule ap-

proximation to the integral is

$$\begin{aligned}\oint g(z)\,\mathrm{d}z &\simeq \sum_{k=0}^{N-1} \tfrac{1}{2}\bigl[g(z_{k+1})+g(z_k)\bigr]\bigl[\mathrm{e}^{\mathrm{i}2\pi(k+1)/N}-\mathrm{e}^{\mathrm{i}2\pi k/N}\bigr]\\ &= \tfrac{1}{2}\Biggl[\sum_{k=0}^{N-1} g(z_{k+1})\,\mathrm{e}^{\mathrm{i}2\pi(k+1)/N} - \sum_{k=0}^{N-1} g(z_k)\,\mathrm{e}^{\mathrm{i}2\pi k/N}\\ &\qquad - \sum_{k=0}^{N-1} g(z_{k+1})\,\mathrm{e}^{\mathrm{i}2\pi k/N} + \sum_{k=0}^{N-1} g(z_k)\,\mathrm{e}^{\mathrm{i}2\pi(k+1)/N}\Biggr].\end{aligned}$$

Noting that $z_N = z_0$, the first two sums inside the brackets cancel each other in their entirety, and the remaining two sums are equal except for trivial phase factors, so the entire expression simplifies to

$$\oint g(z)\,\mathrm{d}z \simeq \tfrac{1}{2}\bigl[\mathrm{e}^{\mathrm{i}2\pi/N}-\mathrm{e}^{-\mathrm{i}2\pi/N}\bigr]\sum_{k=0}^{N-1} g(z_k)\,\mathrm{e}^{\mathrm{i}2\pi k/N} \simeq \frac{2\pi\mathrm{i}}{N}\sum_{k=0}^{N-1} f(z_k)\,\mathrm{e}^{-\mathrm{i}2\pi km/N},$$

where we have used the definition of $g(z)$ again. Combining this result with the Cauchy formula, we then have

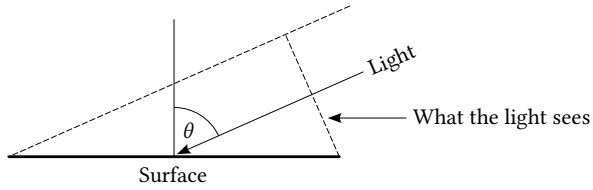
$$\left(\frac{\mathrm{d}^m f}{\mathrm{d}z^m}\right)_{z=0} \simeq \frac{m!}{N}\sum_{k=0}^{N-1} f(z_k)\,\mathrm{e}^{-\mathrm{i}2\pi km/N}.$$

Write a program to calculate the first twenty derivatives of $f(z) = \mathrm{e}^{2z}$ at $z = 0$ using this formula with $N = 10\,000$. You will need to use the version of the `exp` function from the `cmath` package, which can handle complex arguments. You may also find the function `factorial` from the `math` package useful; it calculates factorials of integer arguments.

The correct value for the mth derivative in this case is easily shown to be 2^m, so it should be straightforward to tell if your program is working—the results should be powers of two, 2, 4, 8, 16, 32, etc. You should find that it is possible to get reasonably accurate results for all twenty derivatives rapidly using this technique. If you use standard difference formulas for the derivatives, on the other hand, you will find that you can calculate only the first three or four derivatives accurately before the numerical errors become so large that the results are useless. In this case, therefore, the Cauchy formula gives better results.

The sum $\sum_k f(z_k)\,\mathrm{e}^{-\mathrm{i}2\pi km/N}$ that appears in the formula above is known as the *discrete Fourier transform* of the complex samples $f(z_k)$. There exists an elegant technique for evaluating the Fourier transform for many values of m simultaneously, known as the *fast Fourier transform*, which could be useful in cases where the direct evaluation of the formula is slow. We will study the fast Fourier transform in Chapter 7.

5.26 Image processing and the STM: When light strikes a surface, the amount falling per unit area depends not only on the intensity of the light, but also on the angle of incidence. If the direction the light is coming from makes an angle θ to the normal, then the light only "sees" $\cos\theta$ of area per unit of actual area on the surface:



So the intensity of illumination is $a\cos\theta$, if a is the raw intensity of the light. This simple physical law is a central element of 3D computer graphics. It allows us to calculate how light falls on three-dimensional objects and hence how they will look when illuminated from various angles.

Suppose, for instance, that we are looking down on the Earth from above and we see mountains. We know the height of the mountains $w(x,y)$ as a function of position in the plane, so the equation for the Earth's surface is simply $z = w(x,y)$, or equivalently $z - w(x,y) = 0$, and the normal vector $\mathbf{v}$ to the surface is given by the gradient of $z - w(x,y)$ thus:

$$\mathbf{v} = \nabla[z - w(x,y)] = \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{pmatrix} [z - w(x,y)] = \begin{pmatrix} -\partial w/\partial x \\ -\partial w/\partial y \\ 1 \end{pmatrix}.$$

Now suppose we have incident light represented by a vector $\mathbf{a}$ that points toward the source of the light and has magnitude equal to the intensity. The dot product of the vectors $\mathbf{a}$ and $\mathbf{v}$ is

$$\mathbf{a}\cdot\mathbf{v} = |\mathbf{a}|\,|\mathbf{v}|\cos\theta,$$

where θ is the angle between the vectors. Employing the cosine rule discussed above, the intensity of illumination of the surface of the mountains is then

$$I = |\mathbf{a}|\cos\theta = \frac{\mathbf{a}\cdot\mathbf{v}}{|\mathbf{v}|} = \frac{-a_x(\partial w/\partial x) - a_y(\partial w/\partial y) + a_z}{\sqrt{(\partial w/\partial x)^2 + (\partial w/\partial y)^2 + 1}}.$$

Let us take the simple case where the light is shining horizontally with unit intensity, and the direction it is coming from makes an angle ϕ to the east-west axis, so that $\mathbf{a} = (\cos\phi, \sin\phi, 0)$. Then our intensity of illumination simplifies to

$$I = -\frac{\cos\phi\,(\partial w/\partial x) + \sin\phi\,(\partial w/\partial y)}{\sqrt{(\partial w/\partial x)^2 + (\partial w/\partial y)^2 + 1}}.$$

If we can calculate the derivatives of the height $w(x,y)$ and we know ϕ we can calculate the intensity at any point.

a) In the online resources you will find a file called `altitude.txt`, which contains the altitude $w(x,y)$ in meters above sea level (or depth below sea level) of the surface of the Earth, measured on a grid of points (x,y). Write a program that reads this file and stores the data in an array. Then calculate the derivatives $\partial w/\partial x$ and $\partial w/\partial y$ at each grid point. Explain what method you used to calculate them and why. (Hint: You will probably have to use more than one method to get every grid point, because awkward things happen at the edges of the grid.) To calculate the derivatives you will need to know the value of h, the distance in meters between grid points, which is about 30 000 m in this case.[5]

[5] It is actually not precisely constant because we are representing the spherical Earth on a flat map, but $h = 30\,000$ m will give reasonable results.

b) Now, using your values for the derivatives, calculate the intensity for each grid point, with $\phi = 45^\circ$, and make a density plot of the resulting values in which the brightness of each dot depends on the corresponding intensity value. If you get it working right, the plot should look like a relief map of the world—you should be able to see the continents and mountain ranges in 3D. (Common problems include a map that is upside-down or sideways, or a relief map that is "inside-out," meaning the high regions look low and *vice versa*. Work with the details of your program until you get a map that looks right to you.)

Hint: Note that the intensity I in the formula above can be either positive or negative—it ranges from +1 to −1. What does a negative intensity mean? It means that the area in question is *in shadow*—it lies on the wrong side of the mountain to receive any light at all. You could represent this by coloring these areas of the map completely black, although in practice you will get a nicer-looking image (if arguably less true-to-life) by simply using a continuous range of grays from +1 to −1.

c) There is another file in the online resources called `stm.txt`, which contains a grid of values from scanning tunneling microscope measurements of the (111) surface of silicon. A scanning tunneling microscope (STM) is a device that measures the shape of surfaces at the atomic level by tracking a sharp tip over the surface and measuring quantum tunneling current as a function of position. The end result is a grid of values that represent the height of the surface as a function of position and the data in the file `stm.txt` contain just such a grid of values. Modify the program you just wrote to visualize the STM data and hence create a 3D picture of what the silicon surface looks like. The value of h for the derivatives in this case is around $h = 2.5$ (in arbitrary units).