

CHAPTER 5

INTEGRALS AND DERIVATIVES

IN THE preceding chapters we looked at the basics 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 wish to evaluate the integral of a given function. Let us consider initially the simplest case, the integral of a function of a single variable over a finite range. We will study a range of techniques for the numerical evaluation of such integrals, but we start with the most basic—and also most widely used—the trapezoidal rule.¹

¹Also called the trapezium rule in British English.

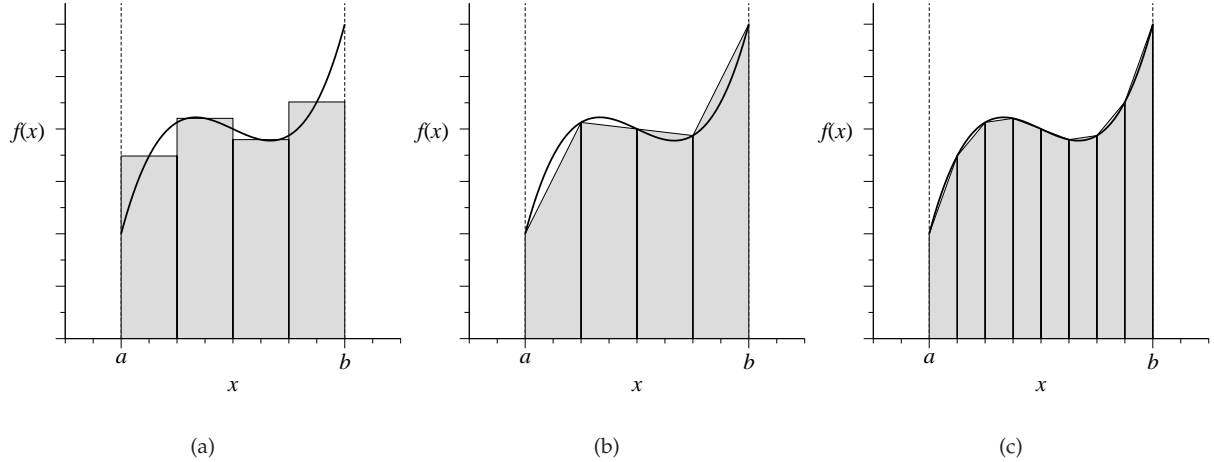


Figure 5.1: Estimating the area under a curve. (a) A simple scheme for estimating the area under a curve by dividing the area into rectangular slices. The gray shaded area approximates the area under the curve, though not very well. (b) The trapezoidal rule approximates the area as a set of trapezoids,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

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 $I(a, b)$:

$$I(a, b) = \int_a^b f(x) \, dx. \quad (5.1)$$

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 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 k th 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

$$A_k = \frac{1}{2}h[f(a + (k - 1)h) + f(a + kh)]. \quad (5.2)$$

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

Now our approximation for the area under the whole curve is the sum of the areas of the trapezoids for all N slices:

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

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.

EXAMPLE 5.1: INTEGRATING A FUNCTION

Let us use the trapezoidal rule to calculate the integral of $x^4 - 2x + 1$ 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's 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 for the integral $I(a, b)$ 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

$$\int_0^2 (x^4 - 2x + 1)dx = \left[\frac{1}{5}x^5 - x^2 + x \right]_0^2 = 4.4. \quad (5.4)$$

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

We can make the calculation more accurate by increasing the number of slices. As shown in Fig. 5.1c, we approximate the area under the curve better when N is larger, though the program will also take longer to reach an answer because there are more terms in the sum to evaluate. If we increase the number of slices to $N = 100$ 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 on-line 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 time $t = 0$ 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:

- Read in the data and, using the trapezoidal rule, calculate from them the approximate distance traveled by the particle in the x direction as a function of time. See Section 2.4.3 on page 57 if you want a reminder of how to read data from a file.
- Extend your program to make a graph that shows, on the same plot, both the original velocity curve and the distance traveled as a function of time.

5.1.2 SIMPSON'S RULE

The trapezoidal rule is the simplest of numerical integration methods, taking only a few lines of code as we have seen, but it is often perfectly adequate for calculations 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, particularly for integrands that are rapidly varying, 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 while still arriving at an answer quickly. 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 exactly 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

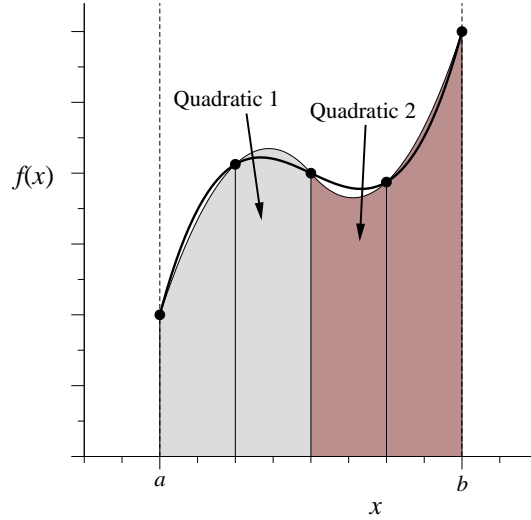


Figure 5.2: Simpson's rule. Simpson's rule involves fitting quadratic curves to pairs of slices and then calculating the area under the quadratics.

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 denoted $f(x)$ 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:

$$f(-h) = Ah^2 - Bh + C, \quad f(0) = C, \quad f(h) = Ah^2 + Bh + C. \quad (5.5)$$

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

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

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

$$\int_{-h}^h (Ax^2 + Bx + C) dx = \frac{2}{3}Ah^3 + 2Ch = \frac{1}{3}h[f(-h) + 4f(0) + f(h)]. \quad (5.7)$$

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 h and 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 worry about the details of fitting 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

$$\begin{aligned} I(a, b) \simeq & \frac{1}{3}h[f(a) + 4f(a + h) + f(a + 2h)] \\ & + \frac{1}{3}h[f(a + 2h) + 4f(a + 3h) + f(a + 4h)] + \dots \\ & + \frac{1}{3}h[f(a + (N - 2)h) + 4f(a + (N - 1)h) + f(b)]. \end{aligned} \quad (5.8)$$

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

$$\begin{aligned} I(a, b) \simeq & \frac{1}{3}h[f(a) + 4f(a + h) + 2f(a + 2h) + 4f(a + 3h) + \dots + f(b)] \\ = & \frac{1}{3}h \left[f(a) + f(b) + 4 \sum_{\substack{k \text{ odd} \\ 1 \dots N-1}} f(a + kh) + 2 \sum_{\substack{k \text{ even} \\ 2 \dots N-2}} f(a + kh) \right]. \end{aligned} \quad (5.9)$$

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

$$I(a, b) \simeq \frac{1}{3}h \left[f(a) + f(b) + 4 \sum_{k=1}^{N/2} f(a + (2k-1)h) + 2 \sum_{k=1}^{N/2-1} f(a + 2kh) \right], \quad (5.10)$$

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

Comparing these equations to Eq. (5.3) 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, suppose we apply Simpson's rule with $N = 10$ slices to the integral from Example 5.1, $\int_0^2 (x^4 - 2x + 1)dx$, whose true value, as we saw, is 4.4. As shown in Exercise 5.2, this gives an answer of 4.400427, 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 moderately accurate answers even with only a small number of steps.

Exercise 5.2:

- Write a program to calculate an approximate value for the integral $\int_0^2 (x^4 - 2x + 1) dx$ 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 142.
- Run the program and compare your result to the known correct value of 4.4. What is the fractional error on your calculation?
- 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

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

- 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.
- 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, you should consult Section 3.1, starting on page 88.

Note that 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'll 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 central spot surrounded by a series of concentric rings. The intensity of the light in this diffraction pattern is given by

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

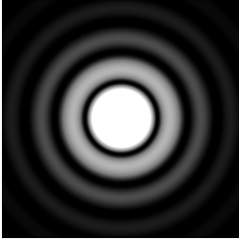
where r is the distance in the focal plane from the center of the diffraction pattern, $k = 2\pi/\lambda$, and $J_1(x)$ is a Bessel function. The Bessel functions $J_m(x)$ are given by

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

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

- 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 $x = 20$.
- Make 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 $1 \mu\text{m}$.

Hint 1: You may find it useful to know that $\lim_{x \rightarrow 0} J_1(x)/x = \frac{1}{2}$. 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



The diffraction pattern produced by a point source of light when viewed through a telescope.

elements in x with value 0.1, or any greater value, will produce the brightest (most positive) color on the screen. By lowering the v_{\max} 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. (There is also a v_{\min} argument that can be used to set the value that corresponds to the dimmest (most negative) color.) For this exercise a value of $v_{\max}=0.01$ appears to work well.

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 $\int_a^b f(x) dx$, and let us first look 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:

$$f(x) = f(x_{k-1}) + (x - x_{k-1})f'(x_{k-1}) + \frac{1}{2}(x - x_{k-1})^2 f''(x_{k-1}) + \dots \quad (5.11)$$

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

$$\begin{aligned} \int_{x_{k-1}}^{x_k} f(x) dx &= f(x_{k-1}) \int_{x_{k-1}}^{x_k} dx + f'(x_{k-1}) \int_{x_{k-1}}^{x_k} (x - x_{k-1}) dx \\ &\quad + \frac{1}{2} f''(x_{k-1}) \int_{x_{k-1}}^{x_k} (x - x_{k-1})^2 dx + \dots \end{aligned} \quad (5.12)$$

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

$$\begin{aligned} \int_{x_{k-1}}^{x_k} f(x) dx &= f(x_{k-1}) \int_0^h du + f'(x_{k-1}) \int_0^h u du + \frac{1}{2} f''(x_{k-1}) \int_0^h u^2 du + \dots \\ &= hf(x_{k-1}) + \frac{1}{2} h^2 f'(x_{k-1}) + \frac{1}{6} h^3 f''(x_{k-1}) + O(h^4), \end{aligned} \quad (5.13)$$

where $O(h^4)$ 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

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

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

$$\begin{aligned} \int_{x_{k-1}}^{x_k} f(x) \, dx &= \frac{1}{2}h[f(x_{k-1}) + f(x_k)] + \frac{1}{4}h^2[f'(x_{k-1}) - f'(x_k)] \\ &\quad + \frac{1}{12}h^3[f''(x_{k-1}) + f''(x_k)] + O(h^4). \end{aligned} \quad (5.15)$$

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

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

Let's take a close look at this expression to see what's 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 notice 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 $f(x) \rightarrow f''(x)$ on the left-hand side, we get

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

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

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

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

$$\int_a^b f(x) dx = \frac{1}{2}h \sum_{k=1}^N [f(x_{k-1}) + f(x_k)] + \frac{1}{12}h^2 [f'(a) - f'(b)] + O(h^4). \quad (5.19)$$

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

$$\epsilon = \frac{1}{12}h^2 [f'(a) - f'(b)]. \quad (5.20)$$

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's 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, meaning 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 $O(h)$* and has an *error $O(h^2)$* .

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 C times the value of the integral, where C is the error constant, which is about 10^{-16} in current versions of Python.² Equation (5.20) tells us that the

²One 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. As shown in Section 4.2 and Eq. (4.7), however, 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 it is safe to say that the final error is no greater than the error incurred by the final operation in the calculation, which will have size C times the final value.

Motivation to work with the approx. error as a condition to my code stop to integrate.

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 little point in making h so small that the approximation error becomes much 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 and rounding errors are roughly equal, which is the point where

$$\frac{1}{12}h^2[f'(a) - f'(b)] \simeq C \int_a^b f(x) dx. \quad (5.21)$$

Rearranging for h we get

$$h \simeq \sqrt{\frac{12 \int_a^b f(x) dx}{f'(a) - f'(b)}} C^{1/2}. \quad (5.22)$$

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

$$N \simeq (b - a) \sqrt{\frac{f'(a) - f'(b)}{12 \int_a^b f(x) dx}} C^{-1/2}. \quad (5.23)$$

Thus if, for example, all the factors except the last are of order unity, then rounding error will become important when $N \simeq 10^8$. Looked at another way, this is the point at which the accuracy of the trapezoidal rule reaches the “machine precision,” the maximum accuracy with which the computer can represent the result. 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 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’ll just quote the results. For an integral over the interval from a to b the approximation error is given to leading order by

$$\epsilon = \frac{1}{90}h^4[f'''(a) - f'''(b)]. \quad (5.24)$$

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 $C \int_a^b f(x) dx$ and the equivalent of Eq. (5.23) is

$$N = (b - a)^4 \sqrt{\frac{f'''(a) - f'''(b)}{90 \int_a^b f(x) dx}} C^{-1/4}. \quad (5.25)$$

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 its accuracy exceeds the machine precision of the computer 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 be done easily 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 would be 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 its equivalent for Simpson's rule, 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 we cannot differentiate the function and 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's assume that we are using the trapezoidal rule, since 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 $h_1 = (b - a) / N_1$.

Then here's 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, which will normally be more accurate than the previous one. As we have seen, the trapezoidal rule introduces an error of order $O(h^2)$, which means when we half the value of h we quarter the size of our error. Knowing this fact allows us to estimate how big the error is.

Suppose that the true value of our integral is I and let us denote our first estimate using the trapezoidal rule with N_1 steps by I_1 . The difference between the true value and the estimate, which is the error on the estimate, 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

$$I_1 + ch_1^2 = I_2 + ch_2^2, \quad (5.26)$$

or

$$I_2 - I_1 = ch_1^2 - ch_2^2 = 3ch_2^2, \quad (5.27)$$

where we have made use of the fact that $h_1 = 2h_2$. Rearranging this expression then gives the error ϵ_2 on the second estimate of the integral to be

$$\epsilon_2 = ch_2^2 = \frac{1}{3}(I_2 - I_1). \quad (5.28)$$

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 in practice 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

$$\epsilon_2 = \frac{1}{15}(I_2 - I_1). \quad (5.29)$$

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 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 double 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 i th step of the process is given by the obvious generalization of Eq. (5.28):

$$\epsilon_i = \frac{1}{3}(I_i - I_{i-1}), \quad (5.30)$$

where I_i is the i th 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 don't 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.

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 previous points, as indicated by the arrows.

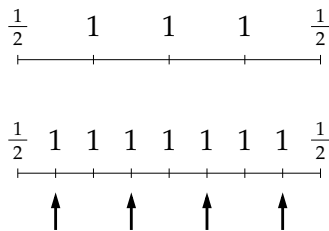


Figure 5.3: Doubling the number of steps in the trapezoidal rule. Top: We evaluate the integrand at evenly spaced points as shown, with the value at each point being multiplied by the appropriate factor. Bottom: when we double the number of steps, we effectively add a new set of points, half way between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

tween 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 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 i th 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

$$\begin{aligned} I_i &= h_i \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N_i-1} f(a + kh_i) \right] \\ &= h_i \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{\substack{k \text{ even} \\ 2 \dots N_i-2}} f(a + kh_i) + \sum_{\substack{k \text{ odd} \\ 1 \dots N_i-1}} f(a + kh_i) \right]. \end{aligned} \quad (5.31)$$

But

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

and hence

$$I_i = \frac{1}{2}h_{i-1} \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N_{i-1}-1} f(a + kh_{i-1}) \right] + h_i \sum_{\substack{k \text{ odd} \\ 1 \dots N_i-1}} f(a + kh_i). \quad (5.33)$$

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

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

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 in our sums is calculated only once, regardless of how many levels of the calculation it's 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, \dots 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

$$\epsilon_i = \frac{1}{15}(I_i - I_{i-1}). \quad (5.35)$$

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

$$S_i = \frac{1}{3} \left[f(a) + f(b) + 2 \sum_{\substack{k \text{ even} \\ 2 \dots N_i-2}} f(a + kh_i) \right], \quad (5.36)$$

and

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

Then we can show that

$$S_i = S_{i-1} + T_{i-1}, \quad (5.38)$$

and

$$I_i = h_i(S_i + 2T_i). \quad (5.39)$$

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 only have to calculate 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 i th 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

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

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

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

But this expression is now accurate to third order, and has only a fourth order error, which is as accurate as Simpson's rule, 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.

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

$$R_{i,1} = I_i, \quad R_{i,2} = I_i + \frac{1}{3}(I_i - I_{i-1}) = R_{i,1} + \frac{1}{3}(R_{i,1} - R_{i-1,1}). \quad (5.42)$$

Then, from Eq. (5.41),

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

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,

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

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

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

Substituting this expression back into (5.43) gives

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

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

We can continue this process, canceling out 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 i th round of the doubling procedure and accurate to order h^{2m-1} , with an error of order h^{2m} , then

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

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

Equating the two and rearranging we have

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

and substituting this into Eq. (5.47) gives

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

where

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

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:

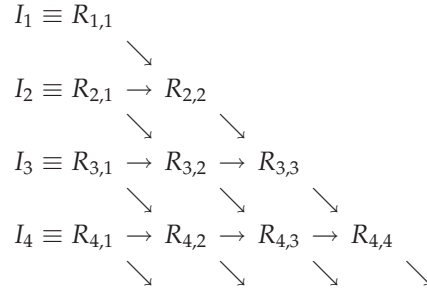
1. We calculate our first two estimates of the integral using the regular trapezoidal rule: $I_1 \equiv R_{1,1}$ and $I_2 \equiv R_{2,1}$.

i : corre para áreas do trapézio
 m : corre para precisão

$m = 1$, não há precisão na área
 avaliada. Só a aplic. do mét. trapézio

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 $R_{i,2} \dots R_{i,i}$.
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:



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're being picky, however, we should point out 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's usually good enough.

This whole procedure is called *Romberg integration*. It's 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 value (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 example of Richardson extrapolation in Section 8.5.5, when we apply it to the solution of differential equations.

Exercise 5.7: Consider the integral

$$I = \int_0^1 \sin^2 \sqrt{100x} \, dx$$

- 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 $\epsilon = 10^{-6}$ (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 161, 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 $\epsilon = 10^{-6}$. 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 $\epsilon = 10^{-6}$. 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

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

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	$\frac{1}{2}, 1, 1, \dots, 1, \frac{1}{2}$
2 (Simpson's rule)	Quadratic	$\frac{1}{3}, \frac{4}{3}, \frac{2}{3}, \frac{4}{3}, \dots, \frac{4}{3}, \frac{1}{3}$
3	Cubic	$\frac{3}{8}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \frac{9}{8}, \frac{9}{8}, \frac{3}{4}, \dots, \frac{9}{8}, \frac{3}{8}$
4	Quartic	$\frac{14}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \frac{28}{45}, \frac{64}{45}, \frac{8}{15}, \frac{64}{45}, \dots, \frac{64}{45}, \frac{14}{45}$

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. The point to notice is 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 k th 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. This has some significant advantages. Methods with evenly spaced points are relatively simple to program, and it's 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 small numbers of points, making them particularly 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 into 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, and particularly ones that are not equally 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 then 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 indeed that it is 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 nonuniform sample points x_k . Then we will choose the particular set of nonuniform 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:

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

which is called an *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,

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

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

So now consider the following expression:

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

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

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

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:

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

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

$$w_k = \int_a^b \phi_k(x) dx. \quad (5.59)$$

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 simply set the weights w_k

equal to the integrals of the interpolating polynomials, Eq. (5.53), over the domain of integration.

There is no general closed-form formula for the integrals of the interpolating polynomials.³ 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 didn't 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 important point to notice is that we only have to calculate the weights w_k once, and then we can use them in Eq. (5.52) to integrate as many different functions over the given integration domain 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's 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 $-1 \leq x_k \leq 1$ and a set of weights

$$w_k = \int_{-1}^1 \phi_k(x) dx. \quad (5.60)$$

If we want to use any integration domain other than the one from -1 to $+1$, we map these values to that other domain. Since the area under a curve doesn't 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 also need to spread the points out or squeeze them together. The correct rule

³One can in principle expand Eq. (5.53) and then integrate the resulting expression term by term, since powers of x can be integrated in closed form. However, the result would be a sum of 2^{N-1} different terms, which would be intractable even for the fastest computers, for relatively modest values of N .

for mapping the points to a general domain that runs from $x = a$ to $x = b$ is:

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

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:

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

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

$$\int_a^b f(x) dx \simeq \sum_{k=1}^n w'_k f(x'_k). \quad (5.63)$$

5.6.2 SAMPLE POINTS FOR GAUSSIAN QUADRATURE

The developments of the previous section solve half 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 argued in Section 5.5, in the best case it should be possible to choose our N points so that our integration rule is exact for all polynomial integrands up to and including polynomials of degree $2N - 1$. The proof that this is indeed possible, and the accompanying derivation of the positions, is not difficult, but it is quite long and it's not really important for our purposes. If you want to see it, it's given in Appendix C on page 514. Here we'll just look at the results, which definitely are important and useful.

The bottom line is this: to get an integration rule accurate up to the highest possible degree of $2N - 1$, the sample points x_k should be chosen to coincide with the zeros of the N th Legendre polynomial $P_N(x)$, rescaled if necessary to the window of integration using Eq. (5.61), and the corresponding weights w_k are

$$w_k = \left[\frac{2}{(1-x^2)} \left(\frac{dP_N}{dx} \right)^{-1} \right]_{x=x_k}, \quad (5.64)$$

also rescaled if necessary, using Eq. (5.62).

This method is called *Gaussian quadrature*⁴ and although it might sound rather formidable from the description above, in practice it's beautifully simple: given the values x_k and w_k for your chosen N , all you have to do is rescale

⁴It's called "Gaussian" because it was pioneered by the legendary mathematician Carl

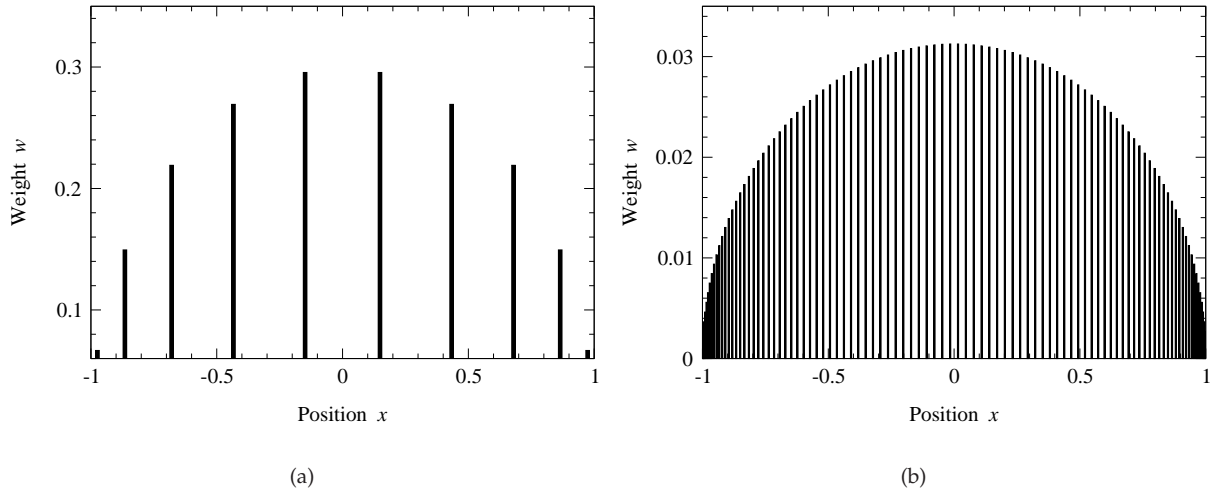


Figure 5.4: Sample points and weights for Gaussian quadrature. The positions and heights of the bars represent the sample points and their associated weights for Gaussian quadrature with (a) $N = 10$ and (b) $N = 100$.

them if necessary using Eqs. (5.61) and (5.62) and then perform the sum in Eq. (5.63).

The only catch is finding the values in the first place. In principle the results quoted above tell us everything we need to know but in practice the zeros of the Legendre polynomials are not trivial to compute. Tables containing values of x_k and w_k up to about $N = 20$ can be found in books or on-line,⁵ or they can be calculated for any N using a suitable computer program. Python functions to perform the calculation are given in Appendix E and also in the on-line resources in the file `gaussxw.py`—Example 5.2 below shows how to use them. Figure 5.4 shows what the 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.

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’re doing calculations by hand, Simpson’s rule with $N = 1000$ is not an option.

⁵See for example Abramowitz, M. and Stegun, I. A., eds., *Handbook of Mathematical Functions*, Dover Publishing, New York (1974).

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's 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, then map them
# to the required integration domain
x,w = gaussxw(N)
xp = 0.5*(b-a)*x + 0.5*(b+a)
wp = 0.5*(b-a)*w

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

Note how the function `gaussxw(N)` returns two variables, not just one. We discussed functions of this type in Section 2.6 but this is the first time we've 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 standard interval from -1 to $+1$. Notice also how we mapped the points and the weights from the standard interval to our desired integration domain: we have used Python's ability to perform calculations with entire arrays to achieve the mapping in just two lines.

There is also an alternative function `gaussxwab(N, a, b)` that calculates the positions and weights and then does the mapping for you. To use this function, we would say "from `gaussxw` import `gaussxwab`", then

```

x,w = gaussxwab(N,a,b)
s = 0.0
for k in range(N):
    s += w[k]*f(x[k])

```

It's worth noting that the calculation of the sample points and weights takes quite a lot of work—the functions above may take a second or so to complete the calculation. That's fine if you call them only once in your program, but you should avoid calling them many times or you may find your program runs slowly. Thus, for instance, if you need to do many integrals over different domains of integration, you should call the function `gaussxw` once to calculate the sample points over the standard interval from -1 to $+1$ and then map the points yourself to the other integration domains you need. Calling `gaussxwab` separately for each different integration domain would be slow and wasteful, since it would needlessly recalculate the zeros of the Legendre polynomial each time.

Our Gaussian quadrature program is quite simple—only a little 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 means degree five when $N = 3$. The function $x^4 - 2x + 1$ that we are integrating here is a degree-four polynomial, so we expect the method to return the exact answer of 4.4, and indeed it does. Nonetheless, the performance of the program 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 deduce 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 would say that the sample points are not nested.⁶

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.

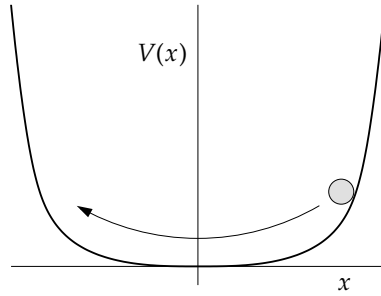
- Write a Python function `cv(T)` that calculates C_V for a given value of the temperature, for a sample consisting of 1000 cubic centimeters of solid aluminum, which has a number density of $\rho = 6.022 \times 10^{28} \text{ m}^{-3}$ and a Debye temperature of $\theta_D = 428 \text{ K}$. Use Gaussian quadrature to evaluate the integral, with $N = 50$ sample points.
- Use your function to make a graph of the heat capacity as a function of temperature from $T = 5 \text{ K}$ to $T = 500 \text{ K}$.

⁶There are other methods, such as *Gauss–Kronrod quadrature* and *Clenshaw–Curtis quadrature*, which have nonuniformly distributed sample points and still permit nesting, although these methods have their own disadvantages. Gauss–Kronrod quadrature permits only one step of nesting: it provides two sets of integration points, one nested inside the other, but no way to generate subsequent points nested inside those. Two sets of points are enough to make error estimates, via a formula analogous to Eq. (5.28), but one cannot keep on doubling the number of points to reduce the error below a given target, as with the adaptive method of Section 5.3. Clenshaw–Curtis quadrature does permit nesting over an arbitrary number of steps, but is not based on an integration rule of the simple form (5.52). Instead the method uses a more complicated formula whose evaluation involves, among other steps, performing a Fourier transform, which is more computationally demanding, and hence slower, than the simple sum used in Gaussian quadrature. In addition, neither Gauss–Kronrod quadrature nor Clenshaw–Curtis quadrature achieves the level of accuracy provided by Gaussian quadrature, although both are highly accurate and probably good enough for most purposes. Gauss–Kronrod quadrature in particular is widely used in mathematical software to compute definite integrals, because of its ability to provide both good accuracy and error estimates. Gauss–Kronrod quadrature is discussed further in Appendix C.

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 of oscillation is a constant, independent of its amplitude, 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:

$$E = \frac{1}{2}m \left(\frac{dx}{dt} \right)^2 + V(x).$$

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

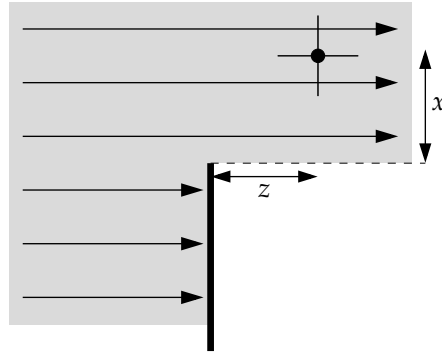
Let us assume that the potential $V(x)$ is symmetric about $x = 0$ and let us set our anharmonic oscillator going with amplitude a . That is, at $t = 0$ we release it from rest at position $x = a$ and it swings back towards the origin. Then at $t = 0$ we have $dx/dt = 0$ and the equation above reads $E = V(a)$, 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 dx/dt and then integrating with respect to t from 0 to $\frac{1}{4}T$, show that the period T is given by

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

- 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 line 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

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

where I_0 is the intensity of the wave before diffraction and

$$u = x \sqrt{\frac{2}{\lambda z}}, \quad C(u) = \int_0^u \cos \frac{1}{2} \pi t^2 dt, \quad S(u) = \int_0^u \sin \frac{1}{2} \pi t^2 dt.$$

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.

5.6.3 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 the function being integrated must be reasonably smooth. 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 of the function at the sample points and nowhere else, substantial variation between points is not taken into account in calculating the value of the integral. If, on the other hand, the function is relatively smooth, then the samples we take will give a good approximation of the function’s behavior and Gaussian quadrature will work well. 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 $I = I_N + \epsilon_N$, where ϵ_N is the approximation error. And if we double the number of samples to $2N$ we have, $I = I_{2N} + \epsilon_{2N}$. Equating the two expressions for I and rearranging, we have

$$\epsilon_N - \epsilon_{2N} = I_{2N} - I_N. \quad (5.65)$$

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 $\epsilon_{2N} \ll \epsilon_N$. So, to a good approximation,

$$\epsilon_N \simeq I_{2N} - I_N. \quad (5.66)$$

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. (5.66) 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. (5.66) 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 always end up doubling N one more time than is strictly necessary to achieve the desired accuracy, and that 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's 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 our 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 short discussion, with some derivations, in Appendix C.

5.7 CHOOSING AN INTEGRATION METHOD

We have studied a number of different 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 well ask at this point which of all these methods is the best? Which one should I use, in practice, if I 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 about the value of the integrand only 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’re evaluating an integral using only ten or twenty sample points, it’s 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 high-order methods using few sample points will not do a good job.

Bearing this principle 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’s not very accurate, but sometimes you don’t need great accuracy. It uses equally spaced sample points, which is appropriate for problems 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, 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 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 highly accurate estimates from small numbers of sample points) as well as 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. (5.52). 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 don't work for these integrals because we'd 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

$$z = \frac{x}{1+x} \quad \text{or equivalently} \quad x = \frac{z}{1-z}. \quad (5.67)$$

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

$$\int_0^\infty f(x) dx = \int_0^1 \frac{1}{(1-z)^2} f\left(\frac{z}{1-z}\right) dz, \quad (5.68)$$

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

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

$$z = \frac{x}{c+x} \quad (5.69)$$

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. (5.67) is often a good first guess. (See Exercise 5.17 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. (5.67). Or we can combine both changes into a single one:

$$z = \frac{x-a}{1+x-a} \quad \text{or} \quad x = \frac{z}{1-z} + a, \quad (5.70)$$

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

$$\int_a^\infty f(x) dx = \int_0^1 \frac{1}{(1-z)^2} f\left(\frac{z}{1-z} + a\right) dz. \quad (5.71)$$

Integrals from $-\infty$ to a can be done the same way—just substitute $z \rightarrow -z$.

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

$$x = \frac{z}{1-z^2}, \quad dx = \frac{1+z^2}{(1-z^2)^2} dz, \quad (5.72)$$

which would give

$$\int_{-\infty}^{\infty} f(x) dx = \int_{-1}^1 \frac{1+z^2}{(1-z^2)^2} f\left(\frac{z}{1-z^2}\right) dz. \quad (5.73)$$

Another possibility, perhaps simpler, is

$$x = \tan z, \quad dx = \frac{dz}{\cos^2 z}, \quad (5.74)$$

which gives

$$\int_{-\infty}^{\infty} f(x) dx = \int_{-\pi/2}^{\pi/2} \frac{f(\tan z)}{\cos^2 z} dz. \quad (5.75)$$

EXAMPLE 5.3: INTEGRATING OVER AN INFINITE RANGE

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

$$I = \int_0^{\infty} e^{-t^2} dt. \quad (5.76)$$

We make the change of variables given in Eq. (5.67) and the integral becomes

$$I = \int_0^1 \frac{e^{-z^2/(1-z)^2}}{(1-z)^2} dz. \quad (5.77)$$

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 gaussxwab
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 = gaussxwab(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

```
0.886226925453
```

In fact, the value of this particular integral is known exactly to be $\frac{1}{2}\sqrt{\pi} = 0.886226925453\dots$. 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 the limits of precision of the computer.

Exercise 5.12: The Stefan–Boltzmann constant

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

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

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 energy per unit area radiated by a black body is

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

- 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 in books or on-line. 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 n th energy level of the one-dimensional quantum harmonic oscillator—i.e., a spinless point particle in a quadratic potential well—is given by

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

for $n = 0 \dots \infty$, where $H_n(x)$ is the n th Hermite polynomial. Hermite polynomials satisfy a relation somewhat similar to that for the 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 $H_1(x) = 2x$.

- 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.
- 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.
- The quantum uncertainty of a particle in the n th level of a quantum 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 on 100 points and 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

$$I = \int_0^1 \int_0^1 f(x, y) dx dy. \quad (5.78)$$

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

$$F(y) = \int_0^1 f(x, y) dx. \quad (5.79)$$

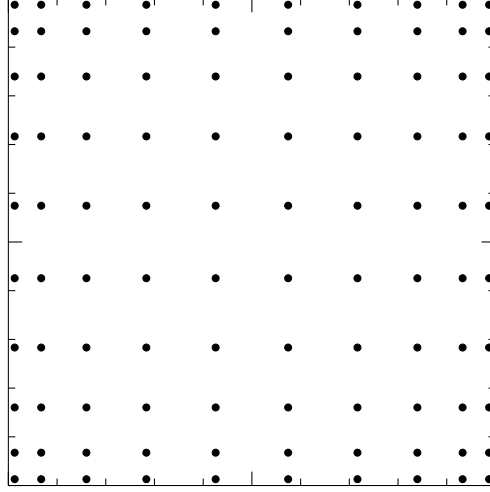


Figure 5.5: Sample points for Gaussian quadrature in two dimensions. If one applies Eq. (5.82) to integrate the function $f(x, y)$ in two dimensions, using Gaussian quadrature with $N = 10$ points along each axis, the resulting set of sample points in the two-dimensional space looks like this.

Then

$$I = \int_0^1 F(y) \, dy. \quad (5.80)$$

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. (5.79), then using those values of $F(y)$ to do the integral in Eq. (5.80). For instance, if we do the integrals by Gaussian quadrature with the same number N of points for both x and y integrals, we have

$$F(y) \simeq \sum_{i=1}^N w_i f(x_i, y) \quad \text{and} \quad I \simeq \sum_{j=1}^N w_j F(y_j). \quad (5.81)$$

An alternative way to look at the calculation is to substitute the first sum into the second to get the *Gauss–Legendre product formula*:

$$I \simeq \sum_{i=1}^N \sum_{j=1}^N w_i w_j f(x_i, y_j). \quad (5.82)$$

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

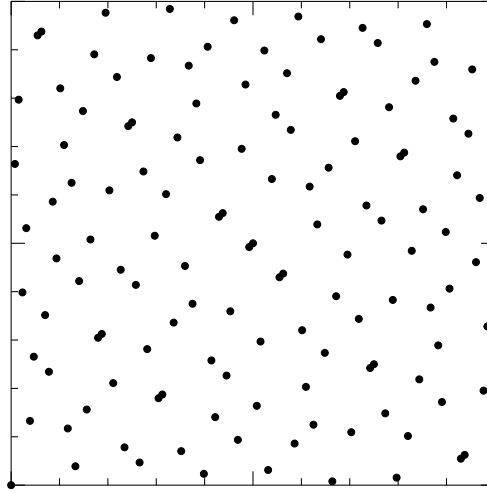


Figure 5.6: 128-point Sobol sequence. The Sobol sequence is one example of a low-discrepancy point set that gives good results for integrals in high dimensions. This figure shows a Sobol sequence of 128 points in two dimensions.

sample points, multiplied by appropriate weights. Equation (5.82) represents a kind of two-dimensional 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's 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 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's nothing random about them). Another common way to

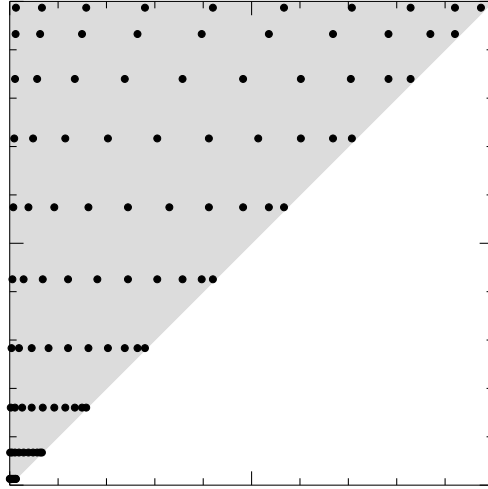


Figure 5.7: Integration over a non-rectangular domain. When the limits of multiple integrals depend on one another they can produce arbitrarily shaped domains of integration. This figure shows the triangular domain that results from the integral in Eq. (5.83). The gray region is the domain of integration. Note how the points become squashed together towards the bottom of the plot.

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 as we will see 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. (5.78) the limits of both integrals were constant, which made the domain of integration rectangular in xy space. It's not uncommon, however, for the limits of one integral to depend on the other, as here:

$$I = \int_0^1 dy \int_0^y dx f(x, y). \quad (5.83)$$

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

$$F(y) = \int_0^y f(x, y) dx, \quad (5.84)$$

so that

$$I = \int_0^1 F(y) dy, \quad (5.85)$$

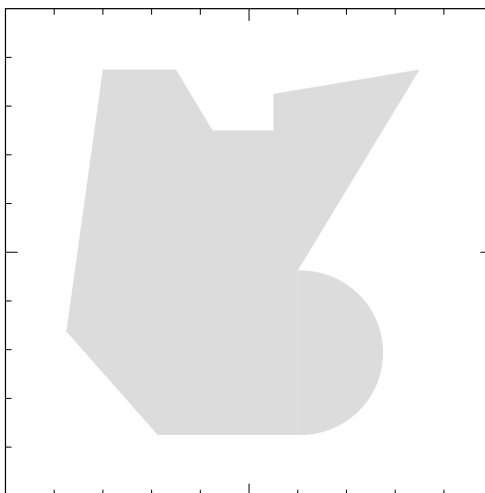


Figure 5.8: A complicated integration domain. Integration domains can be arbitrarily complicated in their shapes. They can even contain holes, or take on complex topologies in higher dimensions such as tori or knotted topologies.

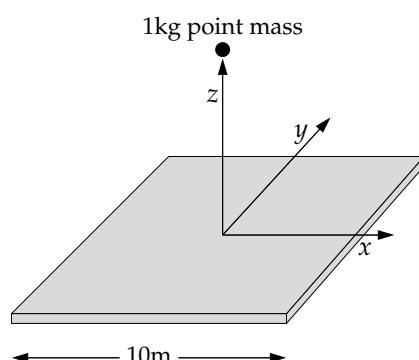
and then do both integrals with any method we choose, such as Gaussian quadrature. The result, again, is a two-dimensional integration rule, but now with the sample points arranged in a triangular space as shown in Fig. 5.7.

This method will work, and will probably give reasonable answers, but it's not ideal. In particular note how the sample points are crammed 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'll 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 worse still. Suppose the domain of integration takes some more complicated 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 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 many 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

$$F_z = G\sigma z \int_{-L/2}^{L/2} \int_{-L/2}^{L/2} \frac{dx dy}{(x^2 + y^2 + z^2)^{3/2}},$$

where $G = 6.674 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$ 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. (5.82), 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 be 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 passing through the galactic center, and the gravitational pull exerted by that plane on bodies outside the galaxy can be calculated by just the methods we have employed here.

5.10 DERIVATIVES

The opposite of a numerical integral is a numerical derivative. You hear a lot less about numerical derivatives than integrals, however, for a number of reasons:

1. The basic techniques for numerical derivatives are quite simple, so they don't take long to explain.
2. Derivatives of known functions can always be calculated analytically, so there's less need to calculate them numerically.
3. There are some significant practical problems with numerical derivatives, which means they are used less often than numerical integrals. (There are, however, some situations in which they are important, particularly in the solution of partial differential equations, which we will look at in Chapter 9.)

For all of these reasons this is a short section—you need to know about numerical derivatives, but we won't spend too much time on them.

5.10.1 FORWARD AND BACKWARD DIFFERENCES

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

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}. \quad (5.86)$$

The basic method for calculating numerical derivatives is precisely an implementation of this formula. We can't take the limit $h \rightarrow 0$ in practice, but we can make h very small and then calculate

$$\frac{df}{dx} \simeq \frac{f(x+h) - f(x)}{h}. \quad (5.87)$$

This approximation to the derivative is called the *forward difference*, because it's 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's simply 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

$$\frac{df}{dx} \simeq \frac{f(x) - f(x-h)}{h}. \quad (5.88)$$

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. There are a few special cases where one is preferred over the other,

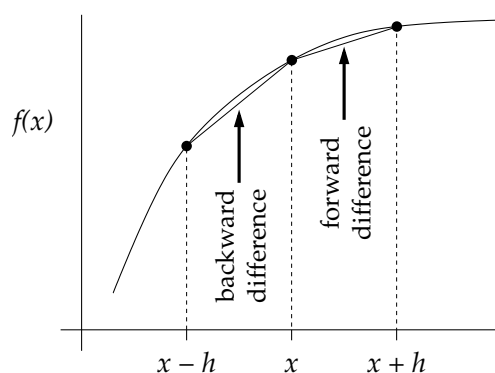


Figure 5.9: Forward and backward differences. The forward and backward differences provide two different approximations to the derivative of a function $f(x)$ at the point x in terms of the slopes of small segments measured in the forward (i.e., positive) direction from x and the backward (negative)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

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 . To work out what the best value is 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 $h \rightarrow 0$, 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 :

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \dots \quad (5.89)$$

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

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2}hf''(x) + \dots \quad (5.90)$$

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. And that's 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. Thus if we make h too small, we will get a large rounding error in our 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 $Cf(x)$, where the value of the error constant C can vary but is typically about $C = 10^{-16}$ 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 $2C|f(x)|$ —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. (5.87), will be $2C|f(x)|/h$.

Meanwhile, the approximation error is, as we have said, about $\frac{1}{2}h|f''(x)|$ from Eq. (5.90), which means that the total error ϵ on our derivative, in the worst case, is

$$\epsilon = \frac{2C|f(x)|}{h} + \frac{1}{2}h|f''(x)|. \quad (5.91)$$

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

$$-\frac{2C|f(x)|}{h^2} + \frac{1}{2}|f''(x)| = 0, \quad (5.92)$$

or equivalently

$$h = \sqrt{4C \left| \frac{f(x)}{f''(x)} \right|}. \quad (5.93)$$

Substituting this value back into Eq. (5.91) we find that the error on our derivative is

$$\epsilon = h |f''(x)| = \sqrt{4C |f(x)f''(x)|}. \quad (5.94)$$

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

In other words, we can get about half of the usual numerical precision on our derivatives but not better. If the 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*:

$$\frac{df}{dx} \simeq \frac{f(x+h/2) - f(x-h/2)}{h}. \quad (5.95)$$

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's 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 expansions:

$$f(x+h/2) = f(x) + \frac{1}{2}hf'(x) + \frac{1}{8}h^2f''(x) + \frac{1}{48}h^3f'''(x) + \dots \quad (5.96)$$

$$f(x-h/2) = f(x) - \frac{1}{2}hf'(x) + \frac{1}{8}h^2f''(x) - \frac{1}{48}h^3f'''(x) + \dots \quad (5.97)$$

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

$$f'(x) = \frac{f(x+h/2) - f(x-h/2)}{h} - \frac{1}{24}h^2f'''(x) + \dots \quad (5.98)$$

To leading order the magnitude of the error is now $\frac{1}{24}h^2 |f'''(x)|$, which is one order in h higher than before. There is also, as before, a rounding error; its size is unchanged from our previous calculation, having magnitude $2C |f(x)|/h$, so the magnitude of the total error on our estimate of the derivative is

$$\epsilon = \frac{2C |f(x)|}{h} + \frac{1}{24}h^2 |f'''(x)|. \quad (5.99)$$

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

$$h = \left(24C \left| \frac{f(x)}{f'''(x)} \right| \right)^{1/3}, \quad (5.100)$$

and substituting this back into Eq. (5.99) we find the optimal error itself to be

$$\epsilon = \frac{1}{8}h^2 |f'''(x)| = \left(\frac{9}{8}C^2 [f(x)]^2 |f'''(x)| \right)^{1/3}. \quad (5.101)$$

Thus, for instance, if $f(x)$ and $f'''(x)$ are of order 1, the ideal value of h is going to be around $h \simeq C^{1/3}$, which is typically about 10^{-5} but the error itself will be around $C^{2/3}$, 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, though we get this accuracy by using a *larger* value of h . This may seem slightly surprising, but it is the correct result.

EXAMPLE 5.4: DERIVATIVE OF A SAMPLED FUNCTION

As an example application 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. Now suppose we want to calculate the derivative of f at one of these 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. (5.95), use points at $x + h/2$ and $x - h/2$ 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

$$\frac{df}{dx} \simeq \frac{f(x+h) - f(x-h)}{2h}. \quad (5.102)$$

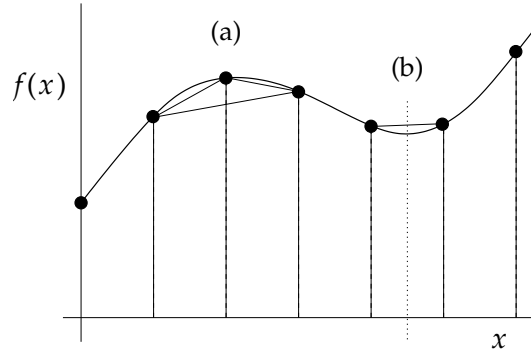


Figure 5.10: Derivative of a sampled function. (a) If we only know the function at a set of sample points spaced a distance h apart then we must choose between calculating the forward or backward difference between adjacent samples, or the central difference between samples $2h$ apart. We cannot calculate a central difference using the standard formula, Eq. (5.95), because we do not know the value of the function at $x \pm \frac{1}{2}h$. (b) We can, however, calculate the value of the derivative at a point half way between two samples (dotted line) using the standard formula.

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's a better approximation or the forward difference because of its smaller interval?

From Eq. (5.94) we see the error on the forward difference is $h |f''(x)|$ and from Eq. (5.101) the error on the central difference—with h replaced by $2h$ —is $h^2 |f'''(x)|$. Which is smaller depends on the value of h . For the central difference to give the more accurate answer, we require $h^2 |f'''(x)| < h |f''(x)|$ or

$$h < \left| \frac{f''(x)}{f'''(x)} \right|. \quad (5.103)$$

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 itself, we want to 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 $x + h/2$ and $x - h/2$, so now we can use the original form of the central difference, Eq. (5.95), 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: 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 $-2 \leq x \leq 2$. 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's called simply `tanh`.)

Exercise 5.16: 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. (5.87) be smaller than on the central difference of Eq. (5.95)?

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, as with the central difference of Section 5.10.3, the best results are obtained by placing the points symmetrically about the point of interest x . 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:

$$ah^2 - bh + c = f(-h), \quad c = f(0), \quad ah^2 + bh + c = f(h), \quad (5.104)$$

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

$$\frac{dy}{dx} = \left[2ax + b \right]_{x=0} = b. \quad (5.105)$$

So we need only the one parameter b , which we can get from Eq. (5.104) 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

$$\frac{df}{dx} \simeq \frac{f(h) - f(-h)}{2h}. \quad (5.106)$$

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

$$\frac{df}{dx} \simeq \frac{f(x+h) - f(x-h)}{2h} \quad (5.107)$$

for general x .

This is the correct result for the quadratic approximation, but it's a disappointing result, since Eq. (5.107) is nothing other than the central difference approximation for sample points $2h$ apart, which we already saw in Eq. (5.102). 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. (5.95). 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

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 $[\frac{1}{24}f(-\frac{3}{2}h) - \frac{27}{24}f(-\frac{1}{2}h) + \frac{27}{24}f(\frac{1}{2}h) - \frac{1}{24}f(\frac{3}{2}h)]/h$. 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.

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 $O(h^2)$, 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.

We will not be using quadratic or higher-order derivative approximations in the remainder of this book—the forward, backward, and central differences will be all we need. But it is worth knowing about them nonetheless; such things come in handy every once in a while.

5.10.5 SECOND DERIVATIVES

We can also derive numerical approximations for the second derivative of a function $f(x)$. 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. (5.95), we can write expressions for the first derivative at $x + h/2$ and $x - h/2$ thus:

$$f'(x + h/2) \simeq \frac{f(x + h) - f(x)}{h}, \quad f'(x - h/2) \simeq \frac{f(x) - f(x - h)}{h}. \quad (5.108)$$

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

$$\begin{aligned} f''(x) &\simeq \frac{f'(x + h/2) - f'(x - h/2)}{h} \\ &= \frac{[f(x + h) - f(x)]/h - [f(x) - f(x - h)]/h}{h} \\ &= \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}. \end{aligned} \quad (5.109)$$

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. (5.109). We perform two Taylor expansions of $f(x)$ thus:

$$f(x + h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \frac{1}{6}h^3f'''(x) + \frac{1}{24}f''''(x) + \dots \quad (5.110)$$

$$f(x - h) = f(x) - hf'(x) + \frac{1}{2}h^2f''(x) - \frac{1}{6}h^3f'''(x) + \frac{1}{24}f''''(x) - \dots \quad (5.111)$$

Adding them together and rearranging, we find that

$$f''(x) = \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} - \frac{1}{12}h^2f''''(x) + \dots \quad (5.112)$$

The first term on the right is our formula for the second derivative, Eq. (5.109), 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 $C|f(x)|$ on each value of $f(x)$ so that, in the worst case, the total rounding error in the numerator of (5.109) is $4C|f(x)|$ and the rounding error on the whole expression is $4C|f(x)|/h^2$. Then the complete error on the derivative is

$$\epsilon = \frac{4C|f(x)|}{h^2} + \frac{1}{12}h^2 |f''''(x)|. \quad (5.113)$$

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

$$h = \left(48C \left| \frac{f(x)}{f''''(x)} \right| \right)^{1/4}. \quad (5.114)$$

Substituting this expression back into Eq. (5.113) gives the size of the optimal error to be

$$\epsilon = \frac{1}{6}h^2 |f''''(x)| = \left(\frac{4}{3}C |f(x)f''''(x)| \right)^{1/2}. \quad (5.115)$$

So if, for instance, $f(x)$ and $f''''(x)$ are of order 1, the error will be roughly of order \sqrt{C} , 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. (5.109) will be good enough.

5.10.6 PARTIAL DERIVATIVES

We will come across a number of situations 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

$$\frac{\partial f}{\partial x} = \frac{f(x + h/2, y) - f(x - h/2, y)}{h}, \quad (5.116)$$

$$\frac{\partial f}{\partial y} = \frac{f(x, y + h/2) - f(x, y - h/2)}{h}. \quad (5.117)$$

By analogy with our approach for the second derivative in Section 5.10.5 we can also calculate second derivatives with respect to either variable, or a mixed second derivative with respect to both, which is given by

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{f(x + h/2, y + h/2) - f(x - h/2, y + h/2) - f(x + h/2, y - h/2) + f(x - h/2, y - h/2)}{h^2}. \quad (5.118)$$

We leave the derivation to the avid reader.

5.10.7 DERIVATIVES OF NOISY DATA

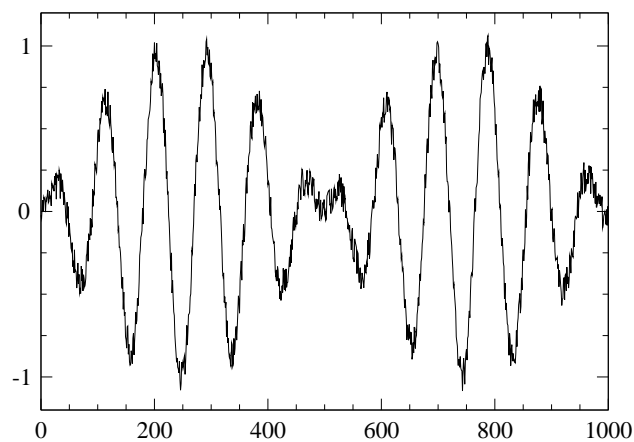
Suppose we have some measurements of a quantity that, when plotted on a graph, look like Fig. 5.11a. Perhaps they come from an experiment in the lab, for instance. The overall shape of the curve is clear from the figure, but there is some noise in the data, so the curve is not completely smooth.

Now suppose we want to calculate the first derivative of this curve. So 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's almost impossible to see the shape of the curve. This is a common problem with numerical derivatives—if there's any noise in the curve you're differentiating, then it can be greatly exaggerated by taking the derivative, perhaps to the point where the results are useless.

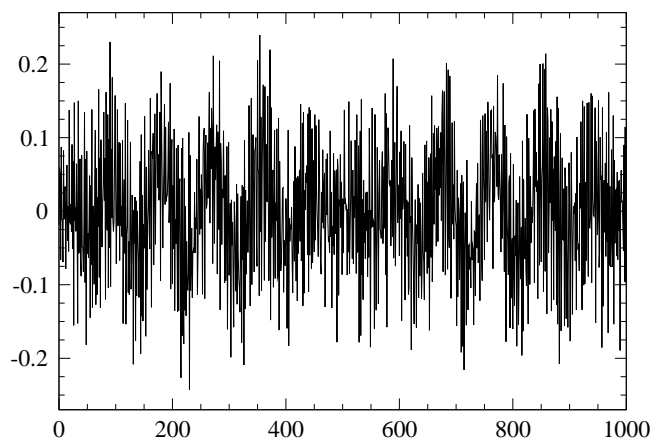
The reason for the problem is easy to see if you zoom in on a small portion of the original data, as shown in Fig. 5.12. In this figure the solid line represents the actual data, and the dotted line is a sketch of what the underlying curve, without the noise, probably looks like. (We don't usually know 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 underlying one reasonably closely, 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 used less than numerical integrals. There are, however, 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



(a)



(b)

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 of the result difficult to discern.

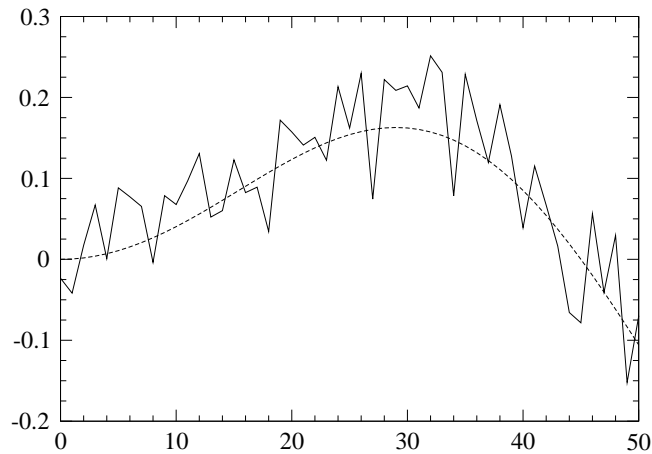


Figure 5.12: An expanded view of the noisy data. The jagged line in this plot is an enlargement of the first portion of the data from Fig. 5.11a, while the dotted line is a guess about the form of the underlying curve, without the noise.

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. (5.93) for the forward difference or Eq. (5.100) for the central difference, but with the error constant C replaced by the fractional error introduced into the data by the noise (which is the inverse of the so-called *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 do a least-squares fit to find the curve that best approximates a larger number of 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.
3. A third approach is to smooth the data in some other fashion before differentiating, which can be done, for instance, using Fourier transforms, which we will study in Chapter 7. (See Exercise 7.4 for an example

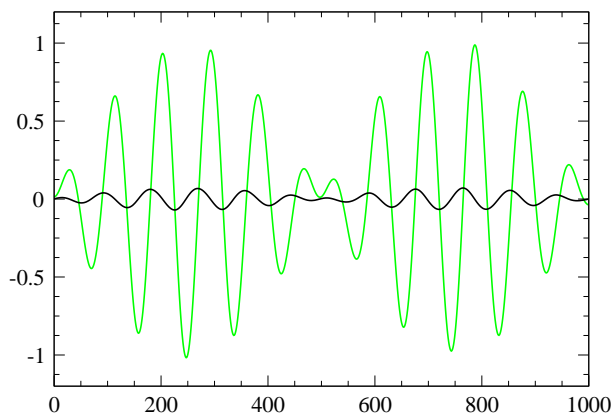


Figure 5.13: Smoothed data and an improved estimate of the derivative. The gray curve in this plot is a version of the data from Fig. 5.11a that has been smoothed to remove noise using a Fourier transform method. The black curve shows the numerical derivative of the smoothed function, which is a significant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

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 INTERPOLATION

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

Suppose you are given the value of a function $f(x)$ at just two points $x = a, b$ and you want to know the value of the function at another point x in between. What do you do? There are a number of possibilities, 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 a curve between the two points, as sketched in the figure. But if we make this assumption then we can calculate $f(x)$ with some elementary geometry.

The slope of the straight-line approximation is

$$m = \frac{f(b) - f(a)}{b - a}, \quad (5.119)$$

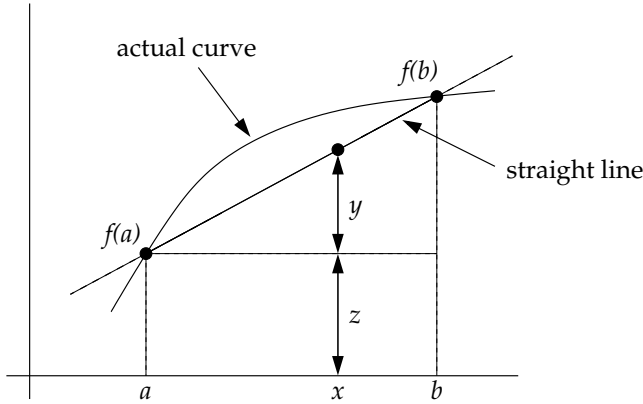


Figure 5.14: Linear interpolation. The value of $f(x)$ in between the two known points at $x = a$ and $x = b$ is estimated by assuming a straight line from $f(a)$ to $f(b)$.

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

$$\begin{aligned} f(x) &\simeq y + z = \frac{f(b) - f(a)}{b - a}(x - a) + f(a) \\ &= \frac{(b - x)f(a) + (x - a)f(b)}{b - a}. \end{aligned} \quad (5.120)$$

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.

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

$$f(a) = f(x) + (a - x)f'(x) + \frac{1}{2}(a - x)^2 f''(x) + \dots \quad (5.121)$$

$$f(b) = f(x) + (b - x)f'(x) + \frac{1}{2}(b - x)^2 f''(x) + \dots \quad (5.122)$$

Substituting these into Eq. (5.120), the terms in $f'(x)$ cancel and, after rearranging a little, we find that

$$f(x) = \frac{(b - x)f(a) + (x - a)f(b)}{b - a} + (a - x)(b - x)f''(x) + \dots \quad (5.123)$$

The first term on the right-hand side is our linear interpolation formula; the remainder 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 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 $O(h^2)$, 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. (5.120), 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 (like calculations of derivatives) that are based on subtractions.

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 using higher-order polynomials. 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. Fitting quadratics or higher polynomials leads to a set of higher-order methods known as *Lagrange interpolation* methods.

When the number of points becomes large, however, this 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 doesn't work because very high order polynomials tend to have a lot of wiggles in them and can deviate from the fitted points badly in the intervals between points. It's 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 won't go into these methods further, however. For our purposes, linear interpolation will be good enough.

FURTHER EXERCISES

5.17 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. (5.67), 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 given in Eq. (5.69), which we repeat here for convenience:

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

- 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 $9! = 362\,880$.

5.18 Rearranging Eq. (5.19) into a slightly more conventional form, we have:

$$\int_a^b f(x) dx = h \left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a + kh) \right] + \frac{1}{12}h^2 [f'(a) - f'(b)] + O(h^4).$$

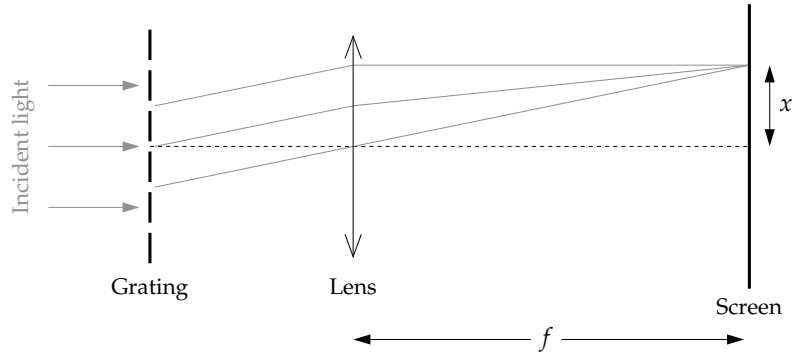
This result gives a value for the integral on the left which has an error of order h^4 —a factor of h^2 better than the error on the trapezoidal rule and as good as Simpson's rule. We can use this formula as a new rule for evaluating integrals, distinct from any of the others we have seen in this chapter.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

- a) Write a program to calculate the value of the integral $\int_0^2 (x^4 - 2x + 1) dx$ 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.

5.19 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, i.e., the fraction of the incident light that the grating lets through.

- 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 $20 \mu\text{m}$.
- 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 $\lambda = 500 \text{ nm}$. 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’ve made your choice you’ll also need to decide the number of sample points you’ll 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 the 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 $10 \mu\text{m}$ wide and the other $20 \mu\text{m}$ wide, with a $60 \mu\text{m}$ gap between the two.

5.20 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 form of the trapezoidal rule 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've already seen, if one knows the right tricks. Here's how the method works.

Suppose we wish to evaluate the integral $I = \int_a^b f(x) dx$ 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 doesn't 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.

a) 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 $\epsilon = 10^{-4}$. 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'll probably have to include this point as a special case in your function using an if statement.

The best way to perform the integration itself is to make use of the technique of recursion, the ability of a Python function to call itself. (If you're not familiar with recursion, you may like to look at Exercise 2.13 on page 83 before doing this exercise.) 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.)

Hint: 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 the method of "local extrapolation," although it's perhaps not obvious what we're extrapolating in this case. We'll discuss local extrapolation again when we study adaptive methods for the solution of differential equations in Section 8.4.)

- b) 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?
- c) Modify your program to make a plot of the integrand with dots added showing where the ends 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.

5.21 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. Calculate the resulting electric potential on a $1\text{ m} \times 1\text{ m}$ 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 `pylab`, 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 `arrow` object from the `visual` package, drawing a grid of arrows with direction and length chosen to represent the field.

- c) Now suppose you have a continuous distribution of charge over an $L \times L$ square. The charge density in Cm^{-2} is

$$\sigma(x, y) = q_0 \sin \frac{2\pi x}{L} \sin \frac{2\pi y}{L}.$$

Calculate and visualize the resulting electric field at 1 cm-spaced points in 1 square meter of the xy plane for the case where $L = 10$ cm, the charge distribution is centered in the middle of the visualized area, and $q_0 = 100 \text{Cm}^{-2}$. You will have to perform a double integral over x and y , then differentiate the potential with respect to position to get the electric field. Choose whatever integration method seems appropriate for the integrals.

5.22 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:

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

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 the positions $z_k = e^{i2\pi k/N}$ for $k = 0 \dots N$. Then the distance between consecutive sample points is

$$h_k = z_{k+1} - z_k = e^{i2\pi(k+1)/N} - e^{i2\pi k/N},$$

and, introducing the shorthand $g(z) = f(z)/z^{m+1}$ for the integrand, the trapezoidal rule approximation to the integral is

$$\begin{aligned} \oint g(z) dz &\simeq \sum_{k=0}^{N-1} \frac{1}{2} [g(z_{k+1}) + g(z_k)] [e^{i2\pi(k+1)/N} - e^{i2\pi k/N}] \\ &= \frac{1}{2} \left[\sum_{k=0}^{N-1} g(z_{k+1}) e^{i2\pi(k+1)/N} - \sum_{k=0}^{N-1} g(z_k) e^{i2\pi k/N} \right. \\ &\quad \left. - \sum_{k=0}^{N-1} g(z_{k+1}) e^{i2\pi k/N} + \sum_{k=0}^{N-1} g(z_k) e^{i2\pi(k+1)/N} \right]. \end{aligned}$$

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

$$\begin{aligned} \oint g(z) dz &\simeq \frac{1}{2} [e^{i2\pi/N} - e^{-i2\pi/N}] \sum_{k=0}^{N-1} g(z_k) e^{i2\pi k/N} \\ &\simeq \frac{2\pi i}{N} \sum_{k=0}^{N-1} f(z_k) e^{-i2\pi km/N}, \end{aligned}$$

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

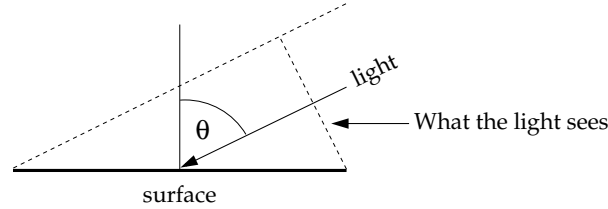
$$\left(\frac{d^m f}{dz^m} \right)_{z=0} \simeq \frac{m!}{N} \sum_{k=0}^{N-1} f(z_k) e^{-i2\pi km/N}.$$

Write a program to calculate the first twenty derivatives of $f(z) = e^{2z}$ at $z = 0$ using this formula with $N = 10000$. 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 m th 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 the better results.

The sum $\sum_k f(z_k) e^{i2\pi km/N}$ 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 detail in Chapter 7.

5.23 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 light makes an angle θ to the normal, it 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 $w(x, y) - z = 0$, and the normal vector \mathbf{v} to the surface is given by the gradient of $w(x, y) - z$ thus:

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

Now suppose we have light coming in represented by a vector \mathbf{a} with magnitude equal to the intensity of the light. Then 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. Thus the intensity of illumination of the surface of the mountains is

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

Let’s take a simple case where the light is shining horizontally with unit intensity, along a line an angle ϕ counter-clockwise from 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 on-line resources you'll 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'll 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'll need to know the value of h , the distance in meters between grid points, which is about 30 000 m in this case.⁷
- 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.)
- c) There is another file in the on-line 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).

⁷It's actually not precisely constant because we are representing the spherical Earth on a flat map, but $h = 30\,000$ m will give reasonable results.