Chapter 13

Numerical Integration I

13.1 A simple example

Integrals cannot always be evaluated analytically. Here's an example. A self–driving car is programmed to travel along a straight road for one hour with velocity v (in kilometers per hour) as a function of time t (in hours) given by

$$v(t) = 100 \sin^2(\pi(t^2 - t)) \ . \tag{13.1}$$

How far does the car travel during this one–hour trip? We know that velocity v is the time derivative of distance s, so the answer is found by integration:

$$s = \int_0^1 \left[100 \sin^2(\pi(t^2 - t))\right] dt \ . \tag{13.2}$$

This integral cannot be expressed analytically in terms of elementary functions. We must rely on numerical methods to find s.

Exercise 13.1

Write a Python code to plot the velocity $v(t)$ as a function of t for $0 \leq t \leq 1$. Note that the car accelerates smoothly from rest to a maximum of 50 km/hr, then decelerates smoothly and comes to a stop after 1 hour.

We will return to this problem later in this chapter, after presenting several techniques for evaluating integrals numerically.

13.2 Left endpoint rule

The integral

$$I = \int_a^b f(x)\,dx \tag{13.3}$$

is the "area under the curve" $f(x)$ from a to b .[1] The area can be approximated by the sum of the areas of rectangles, as shown in Fig. 13.1.

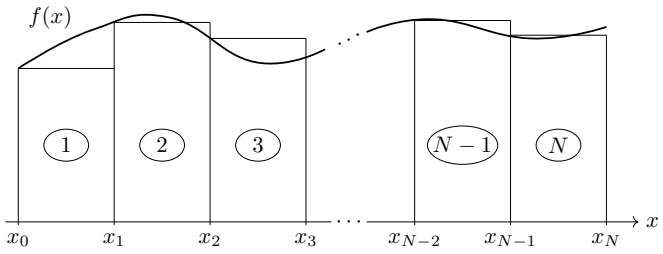

Fig. 13.1: The left endpoint rule (left Riemann sum) approximates the integral (13.3) as the sum of areas of rectangles. The figure shows the first three subintervals and the last two subintervals.

The rectangles are obtained by dividing the interval $a \le x \le b$ into $N+1$ nodes labeled x_0 through x_N. Note that x_0 is just another name for the left endpoint a. Likewise, x_N is another name for the right endpoint b. The $N+1$ nodes span N subintervals, labeled 1 through N. The first subinterval extends from x_0 to x_1. The Nth subinterval extends from x_{N-1} to x_N. In general, the ith subinterval extends from x_{i-1} to x_i.

Figure 13.1 depicts the *left endpoint rule* (also called the left Riemann sum). The height of the rectangle in the ith subinterval is $f(x_{i-1})$, the value of the function at the left–side node. The area of the ith rectangle is the product of its height $f(x_{i-1})$ and its width $x_i - x_{i-1}$. We will take the nodes to be equally spaced, so that $x_i - x_{i-1} = \Delta x$ for each subinterval i. Then the left endpoint rule is

$$I_L = \sum_{i=1}^{N} f(x_{i-1})\Delta x\ . \tag{13.4}$$

[1] This is the case for most functions we encounter in science and engineering. Some exotic functions require a more sophisticated definition of the integral.

The left endpoint rule only approximates the integral (13.3). You can see from Fig. 13.1 that I_L will not equal I exactly, since the rectanges are too tall in some regions and too short in other regions. In the limit as $N \to \infty$, the rectangles become infinitely thin and I_L approaches the exact value for the integral I.

Exercise 13.2a

A farmer plans to build a fence across one side of their property, which is 50 meters long. The fenceposts must be placed 1 meter apart. How many fenceposts does the farmer need?

Remember, when we approximate an integral, the number of nodes (fenceposts) is always one greater than the number of subintervals (space between fenceposts). If the number of subintervals is N, the number of nodes is $N+1$. Also observe that the sum in Eq. (13.4) is a sum over areas of rectangles. The rectanges fill the subintervals. Hence, the sum runs from 1 through N.

It is not difficult to implement the left endpoint rule (13.4) in a computer program. You need to specify the endpoints a, b and the number of subintervals N. You can define the integrand $f(x)$ using a function definition. The width of each subinterval is

```
Deltax = (b - a)/N
```

The node values x_i can be defined as a NumPy array using

```
x = np.linspace(a,b,N+1)
```

The sum can be carried out using a loop such as

```
for i in range(1,N+1):
```

Exercise 13.2b

Why do we divide $b-a$ by N, rather than $N+1$, to obtain Δx? Why do we use $N+1$ rather than N in the linspace command? Why do we use $N+1$ rather than N in the range command?

Exercise 13.2c

Write a code to approximate $I = \int_a^b f(x)\,dx$ using the left endpoint rule. Test your code using $f(x) = \sin x$, $a = 0$ and $b = \pi/2$. What is the exact answer? Compare I_L to the exact answer for $N = 2$, 4,

8, 16 and 32.

13.3 Right endpoint and midpoint rules

For the *right endpoint rule* (also called the right Riemann sum), the height of each rectangle equals the function value at the right–side node. That is, for the ith subinterval, the rectangle height is $f(x_i)$. The right endpoint rule is

$$I_R = \sum_{i=1}^{N} f(x_i)\,\Delta x\ . \tag{13.5}$$

This approximates the integral I. In the limit as $N \to \infty$, the rectangles become infinitely thin and I_R approaches the exact value for I.

Exercise 13.3a

Write a code to approximate $I = \int_a^b f(x)\,dx$ using the right endpoint rule. Test your code using $f(x) = \sin x$, $a = 0$ and $b = \pi/2$. Compare I_R to the exact answer for $N = 2$, 4, 8, 16 and 32.

Consider a region where the function $f(x)$ is increasing. The left endpoint rule I_L underestimates the value of I, whereas the right endpoint rule overestimates the value of I. This is shown in Fig. 13.2. Wouldn't it be better to evaluate the function at the midpoint of the subinterval? This should be more accurate since the extra area from the left half of the subinterval would tend to cancel the missing area from the right half of the subinterval.

This leads us to the *midpoint rule*, depicted in Fig. 13.3. The height of each rectangle is given by the value of the function at the midpoint of the subinterval. For the ith subinterval, the midpoint is $(x_{i-1} + x_i)/2$ and the height of the rectangle is $f((x_{i-1}+x_i)/2)$. The midpoint rule approximation to the integral I is

$$I_M = \sum_{i=1}^{N} f((x_{i-1} + x_i)/2)\,\Delta x\ . \tag{13.6}$$

In the limit as $N \to \infty$, the rectangles become infinitely thin and I_M approaches the exact value for I.

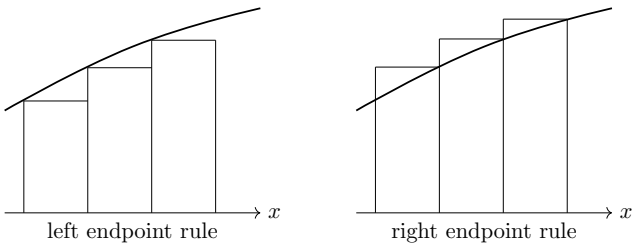

Fig. 13.2: For an increasing function $f(x)$, the left endpoint rule underestimates the area under the curve. The right endpoint rule overestimates the area under the curve.

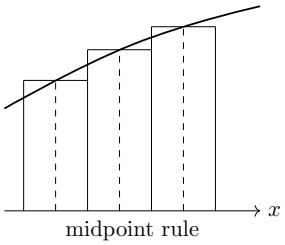

Fig. 13.3: For the midpoint rule, the height of each rectangle is the value of the function $f(x)$ at the midpoint of the subinterval.

Exercise 13.3b

Write a code to approximate $I = \int_a^b f(x)\,dx$ using the midpoint rule. Test your code using $f(x) = \sin x$, $a = 0$ and $b = \pi/2$. Compare I_M to the exact answer for $N = 2$, 4, 8, 16 and 32. Compare the midpoint rule to the left and right endpoint rules.

13.4 Errors and convergence tests

The left endpoint rule I_L, right endpoint rule I_R, and midpoint rule I_M each approximate I with some error. In general the size of the error will decrease if we increase the number of subintervals N. Increasing the number of subintervals N is equivalent to decreasing the size Δx of each subinterval,

since $\Delta x = (b-a)/N$. "Decreasing the error" is also referred to as "increasing the resolution." We increase the resolution (or decrease the error) by increasing N.

A detailed mathematical analysis shows that for smooth integrands, the error in the left and right endpoint rules is (approximately) inversely proportional to the number of subintervals. The error in the midpoint rule is (approximately) inversely proportional to the square of the number of subintervals. To summarize:

$$\begin{aligned} |\text{error for } I_L| &\propto N^{-1} \ , \\ |\text{error for } I_R| &\propto N^{-1} \ , \\ |\text{error for } I_M| &\propto N^{-2} \ . \end{aligned}$$

If we double the resolution for I_L or I_R, we reduce the error by (approximately) 1/2. If we double the resolution for I_M, we reduce the error by (approximately) 1/4.

For most numerical integration methods, the error scales like N^m where m is a *negative* integer. Note that $|\text{error}| \propto N^m$ is equivalent to

$$|\text{error}| = C\, N^m \tag{13.7}$$

for some constant C. Take the base 10 logarithm of this relation to obtain

$$\log|\text{error}| = m \log(N) + \log(C) \ . \tag{13.8}$$

A plot of $\log|\text{error}|$ versus $\log(N)$ should be (approximately) a straight line with slope m.

It is always a good idea to carry out a *convergence test* of your numerical codes. Choose a smooth function whose integral is known analytically. Solve the integral numerically and determine the error as a function of resolution N. The error should scale as in Eq. (13.8); if not, there is a problem with your code.

Remember, in Python, the NumPy command `log` is the base e logarithm. The command for the base 10 logarithm is `log10`.

Exercise 13.4a

As in exercise 13.2c, use the left endpoint rule to approximate the integral $\int_0^{\pi/2} \sin x \, dx$. Since the exact answer is known, you can carry out a convergence test: Compute the error at multiple resolutions. Copy and paste your results into a text file with N in the first column and the error in the second column. Create a code to read the

text file and plot $\log|\text{error}|$ versus $\log(N)$. Does the curve approach a straight line with slope $m = -1$ as the resolution is increased?

Exercise 13.4b

Use the midpoint rule to approximate the integral $\int_0^{\pi/2} \sin x\, dx$ and perform a convergence test. Does the curve of $\log|\text{error}|$ versus $\log(N)$ approach a straight line with slope $m = -2$ as the resolution is increased?

These exercises show that as the resolution increases, the error in the midpoint rule decreases *more rapidly* than the error in the left endpoint rule. This has an important consequence. Let's say we want to know the answer for I to within a certain error, say, ± 0.0001. For most functions $f(x)$, the midpoint rule will require far fewer subintervals than the left or right endpoint rules to achieve a given level of accuracy.

Exercise 13.4c

Consider once again the integral $\int_0^{\pi/2} \sin x\, dx$. For the left endpoint rule, how many subintervals are required to obtain an error (in absolute value) of less than 0.0001? How many for the right endpoint rule? For the midpoint rule?

Exercise 13.4d

Write a code to compute

$$\int_1^{10} x \ln(10/x)\, dx$$

using the left endpoint and midpoint rules, where ln is the base e logarithm. What is the exact answer? With the left endpoint rule, approximately how many subintervals N are required to yield the answer with an error less than 0.01? Less than 0.001? With the midpoint rule, approximately how many subintervals N are required to yield the answer with an error less than 0.01? Less than 0.001?

13.5 Estimating errors and significant figures

In most cases you won't know the exact answer for a given integral, so there is no direct way of determining the error. For example, consider the integral

$$I = \int_0^{10} \sin(\pi \sin x)\, dx \tag{13.9}$$

Using the midpoint rule with $N = 10$ subintervals, Python reports the result as $I_M = 1.3737955$. How accurate is this? Is it correct to 8 significant figures?

Before continuing, let's review the concept of *significant figures*. Roughly speaking, significant figures are numbers that are considered reliable. If a result such as 1.3737955 is only reliable to 3 significant figures, it should be written as 1.37. If it is reliable to 4 significant figures, it should be written as 1.374. (Note, we rounded up.)

Rules for significant figures:

- nonzero numbers are significant
- zeros are significant if
  - they appear between nonzero numbers
  - they are trailing numbers to the right of the decimal point
  - they are trailing numbers in a whole number that includes a decimal point

For example the numbers 4030, 0.0250, and 640. all have 3 significant figures. The same rules apply to numbers written in scientific notation if we ignore the factor of 10 raised to some power. Thus, 3.050×10^7 has 4 significant figures.

With the midpoint rule and $N = 10$ subintervals, our code tells us that the integral (13.9) is $I_M = 1.3737955$. We would like to write this properly, with the correct number of significant figures, by taking into account the accuracy of the result.

Unfortunately we don't know how accurate the result is; equivalently, we don't know what the error is. We do know that the error is (approximately) proportional to N^{-2}, so we can reduce the error by increasing the resolution. If we double the resolution to $N = 20$, we find $I_M = 1.2664347$. This should be more accurate than 1.3737955. Perhaps the answer is around 1.26 or 1.27? To be sure, we need to increase the resolution even further. The following table shows the results obtained by repeatedly doubling the resolution:

N	I_L
10	1.3737955
20	1.2664347
40	1.2433224
80	1.2378149

It appears that the answers are settling down to about 1.23. Perhaps we should round up to 1.24? Notice that the numbers are decreasing as we increase the resolution. Is the correct answer closer to 1.22?

At this point we can be confident that the correct answer to two significant figures is 1.2. If we want a more accurate answer, we need to continue to higher resolution:

N	I_L
160	1.2364534
320	1.2361139
640	1.2360291

With these results, it is fairly clear that the answer is 1.236 to four significant figures. In fact, the answer appears to be converging to 1.2360, or perhaps 1.2359. If we want an answer that is accurate to five significant figures, we need to increase the resolution even further.

This example illustrates an important lesson: *You must run your code at multiple resolutions.* Until you do, you have no idea how accurate the answer is.

Exercise 13.5a

Use the midpoint rule to evaluate the integral of Sec. 13.1 for the distance traveled by the car during its one–hour trip. Show your results for multiple values of N, and use the data to determine the distance to 6 significant figures.

Exercise 13.5b

Use the midpoint rule to evaluate the integral

$$I = \int_{-10}^{10} \ln(2 + \sin(x))\, dx \ .$$

Make a table showing N and I_M for increasing values of N. Use your data to determine the value of the integral to 6 significant figures.