Chapter 14

Numerical Integration II

14.1 Takeaways from before

Numerical computation is all about efficiency. We want to find algorithms that give us the most accuracy with the least amount of computational effort.

In the previous chapter we discussed the left endpoint rule, right endpoint rule, and midpoint rule for numerically evaluating the integral

$$I = \int_a^b f(x)\,dx \ . \tag{14.1}$$

The errors in the left and right endpoint rules are proportional to N^{-1}, where N is the number of subintervals from a to b. To reduce the error by a factor of 100, you must increase N by a factor of 100. This requires 100 times as many evaluations of the integrand $f(x)$. This might be fine for simple problems, with simple integrands. But for complicated integrands that require a lot of computer time to evaluate, this can be a problem.

The midpoint rule is usually better than the left or right endpoint rules. The error for this method is proportional to N^{-2}. With the midpoint rule we can reduce the error by a factor of 100 by increasing N by a factor of 10. This requires only 10 times as many evaluations of the integrand $f(x)$.

Exercise 14.1

Consider the integral

$$I = \int_0^\pi 2x\sin(x^2)\,dx \ ,$$

which has the exact value $I = 1-\cos(\pi^2)$. Approximate this integral using the left endpoint rule I_L and midpoint rule I_M, both with

$N = 10$ subintervals. Which rule gives the more accurate answer? How many subintervals are required to reduce the error in I_L by a factor of 100? How many subintervals are required to reduce the error in I_M by a factor of 100?

14.2 Error analysis

The following analysis shows why the error for the left endpoint rule is proportional to N^{-1}.

As always, divide the interval $a \le x \le b$ into N subintervals. Observe that the integral

$$\int_{x_{i-1}}^{x_i} dx\, f(x) \tag{14.2}$$

is the exact area under the curve $f(x)$ in the ith subinterval. Thus, we can write the integral I of Eq. (14.1) as

$$I = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)\, dx \ , \tag{14.3}$$

which is a sum of integrals in each subinterval.

In the ith subinterval, expand the integrand $f(x)$ in a Taylor series about the left node:

$$f(x) = f(x_{i-1}) + f'(x_{i-1})(x - x_{i-1}) + \frac{1}{2} f''(x_{i-1})(x - x_{i-1})^2 + \cdots \ . \tag{14.4}$$

Here, $f'(x_{i-1})$ and $f''(x_{i-1})$ are the first and second derivatives of $f(x)$ evaluated at the left node x_{i-1}. Integrating over the subinterval, we find

$$\int_{x_{i-1}}^{x_i} f(x)\, dx = f(x_{i-1})\, \Delta x + \frac{1}{2} f'(x_{i-1})\, \Delta x^2 + \cdots \ . \tag{14.5}$$

Now sum this result over subintervals:

$$\sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)\, dx = \sum_{i=1}^{N} f(x_{i-1})\, \Delta x + \sum_{i=1}^{N} \frac{1}{2} f'(x_{i-1})\, \Delta x^2 + \cdots \ . \tag{14.6}$$

The left–hand side is the exact integral I from Eq. (14.3), and the first term on the right–hand side is the left endpoint approximation I_L from Eq. (13.4). Thus Eq. (14.6) can be written as

$$I = I_L + \sum_{i=1}^{N} \left\{ \frac{1}{2} f'(x_{i-1})\, \Delta x^2 + \cdots \right\} \ . \tag{14.7}$$

Therefore the error in the left endpoint rule, $I_L - I$, is

$$(\text{error for } I_L) = -\sum_{i=1}^{N} \left\{ \frac{1}{2} f'(x_{i-1})\, \Delta x^2 + \cdots \right\} . \tag{14.8}$$

The unwritten terms $(\cdots)$ are proportional to Δx^3 and higher powers of Δx.

Assuming f is smooth, so that the derivatives f', f'', *etc* remain finite in the interval from a to b, the error terms will go to zero as Δx goes to zero (as the resolution is increased). The term that goes to zero least quickly is the one proportional to Δx^2. Thus,

$$(\text{error for } I_L) \approx -\frac{\Delta x}{2} \sum_{i=1}^{N} f'(x_{i-1})\, \Delta x \ , \tag{14.9}$$

where $\approx$ means "approximately equal."

The sum in Eq. (14.9) is actually the left endpoint rule approximation to the integral $\int_a^b f'(x)\, dx = f(b) - f(a)$. Therefore, the error becomes

$$(\text{error for } I_L) \approx \left[-\frac{1}{2} \left(f(b) - f(a) \right) \right] \Delta x \ . \tag{14.10}$$

We will use this result in the next section. For now, simply observe that the factor in square brackets is a constant, independent of Δx. Let's call this constant $C/(b-a)$. Since $\Delta x = (b-a)/N$, the error simplifies to

$$(\text{error for } I_L) \approx C\, N^{-1} \ . \tag{14.11}$$

That is, the error for the left endpoint rule is (approximately) proportional to N^{-1}.

A similar analysis shows that the error in the right endpoint rule is (approximately) proportional to N^{-1}.

Exercise 14.2a

The error analysis assumes the integrand is smooth. Compute

$$I = \int_0^9 \frac{1}{\sqrt{x}}\, dx$$

with the left and right endpoint rules. Note that the integrand is not smooth at $x = 0$. What is the exact answer? What result does the left endpoint rule give? Carry out a convergence test using the right endpoint rule. Is the error proportional to N^{-1}?

Exercise 14.2b

Show that the error in the midpoint rule is (approximately) proportional to N^{-2}. Hint: Expand $f(x)$ in a Taylor series about the midpoint $(x_i + x_{i-1})/2$. Integrate from x_{i-1} to x_i then sum over subintervals. You will need to argue that the factor $\sum_{i=1}^{N} f''((x_{i-1}+x_i)/2)\,\Delta x$ is a constant, independent of Δx.

14.3 Trapeziod rule

In the previous section we found an approximate expression, Eq. (14.10), for the error in the left endpoint rule. We can use this result to improve the left endpoint rule. Simply subtract the error from I_L to obtain the trapezoid rule,

$$I_T = I_L + \left[\frac{1}{2}\left(f(b) - f(a)\right)\right]\Delta x \ . \tag{14.12}$$

This numerical integration rule, like the midpoint rule, has errors proportional to N^{-2}. It is generally better than either the left or right endpoint rule.

Recall that $a = x_0$ and $b = x_N$. Using expression (13.4) for the left endpoint rule, we have

$$I_T = \sum_{i=1}^{N} f(x_{i-1})\,\Delta x + \frac{1}{2}f(x_N)\,\Delta x - \frac{1}{2}f(x_0)\,\Delta x \ . \tag{14.13}$$

Expand the sum,

$$\sum_{i=1}^{N} f(x_{i-1})\,\Delta x = \big[f(x_0) + f(x_1) + \cdots + f(x_{N-1})\big]\Delta x \ , \tag{14.14}$$

then rearrange terms to obtain

$$I_T = \left[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right]\Delta x \ . \tag{14.15}$$

This shows that the trapezoid rule can be written as

$$I_T = \frac{1}{2}\big[f(x_0) + f(x_N)\big]\Delta x + \sum_{i=1}^{N-1} f(x_i)\,\Delta x \ . \tag{14.16}$$

Note that the sum extends from 1 to $N-1$.

Why is this numerical integration scheme called the trapezoid rule? Rearrange terms in Eq. (14.15) and write the trapezoid rule as follows:

$$I_T = \frac{1}{2}\left[f(x_0) + f(x_1)\right]\Delta x + \frac{1}{2}\left[f(x_1) + f(x_2)\right]\Delta x + \cdots$$
$$\cdots + \frac{1}{2}\left[f(x_{N-1} + f(x_N)\right]\Delta x \ . \tag{14.17}$$

More compactly, we have

$$I_T = \sum_{i=1}^{N} \frac{1}{2}\left[f(x_{i-1}) + f(x_i)\right]\Delta x \ . \tag{14.18}$$

Written this way the trapezoid rule can be understood as a sum over subintervals. The area in each subinterval is approximated by the area of a trapezoid, as shown in Fig. 14.1. Recall that the area of a trapezoid is given by the average height, multiplied by the width. For the ith subinterval, the average height is the average of the function values at each end node, namely, $\left[f(x_{i-1}) + f(x_i)\right]/2$.

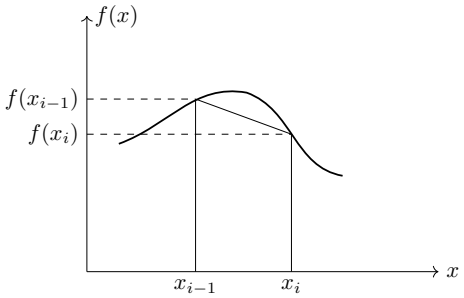

Fig. 14.1: The trapezoid rule. The area under the curve in the ith subinterval is approximated by the area of a trapezoid.

Exercise 14.3a

Write a code that uses the trapezoid rule, Eq. (14.16), to approximate

$$I = \int_1^2 \sin(x + \ln x)\, dx \ .$$

Make a table showing the answer for various numbers of subintervals N. Determine the answer to at least 5 significant figures.

Exercise 14.3b

Test your trapezoid rule code with

$$I = \int_1^{10} x \ln(x)\, dx \ .$$

What is the exact solution? Carry out a convergence test by plotting $\log|\text{error}|$ versus resolution $\log(N)$. Do the errors scale scale like N^{-2}?

14.4 Simpson's rule

For numerical integration, the computer spends most of its time evaluating the integrand $f(x)$ at various values of x. Look closely at the formulas for the left endpoint rule (13.4), right endpoint rule (13.5), midpoint rule (13.6) and trapezoid rule (14.16). For the left and right endpoint rules, the function $f(x)$ must be evaluated N times. Likewise, for the midpoint rule, the integrand $f(x)$ must be evaluated N times. The trapezoid rule is similar, requiring $N+1$ evaluations of the integrand.

Although these four methods require about the same number of function evaluations, the midpoint rule I_M and trapezoid rule I_T are generally much better than the left and right endpoint rules because their errors decrease more rapidly with increasing resolution:

$$|\text{errors for } I_L \text{ and } I_R| \propto N^{-1} \ ,$$
$$|\text{errors for } I_M \text{ and } I_T| \propto N^{-2} \ .$$

Can we do even better? Can we find an algorithm whose errors decrease even more rapidly than N^{-2} without a drastic increase in the number of evaluations of the integrand $f(x)$? The answer is yes—Simpson's rule is such an algorithm.

Simpson's rule is defined by

$$I_S = \frac{\Delta x}{6}\left[f(x_0) + f(x_N) + 2\sum_{i=1}^{N-1} f(x_i) + 4\sum_{i=1}^{N} f((x_{i-1}+x_i)/2)\right] \ . \tag{14.19}$$

Geometrically, Simpson's rule is obtained by approximating the area in each subinterval as the area under a parabola that matches the function $f(x)$ at each endpoint and at the midpoint.

For Simpson's rule the integrand must be evaluated at each node and at the midpoint of each subinterval. For N subintervals, this is $2N+1$

function evaluations. That's roughly twice the number of evaluations as the previous integration rules. However, the error in Simpson's rule decreases very rapidly with increasing resolution:

$$|\text{errors for } I_S| \propto N^{-4} \ .$$

In most cases this decrease more than compensates for the increase in the number of function evaluations.

Exercise 14.4a

Use Simpson's rule to approximate

$$I = -\int_{\pi^2/4}^{4\pi^2} \sin(\sqrt{x})\, dx \ .$$

The exact answer is $I = 2 + 4\pi$. Find the number of subintervals required for Simpson's rule to compute the answer to 8 significant figures. How many evaluations of the integrand does this require? Approximate the same integral using the trapezoid rule, and find the number of subintervals required to reach 8 significant figures. How many evaluations of the integrand are needed for the trapezoid rule?

Exercise 14.4b

Approximate the integral

$$I = \int_0^{\pi} (\sin x)^{3/2}\, dx$$

using Simpson's rule. Find the answer to 6 significant figures.

Exercise 14.4c

Consider the integral

$$I = \int_0^{2\pi} x^2 \cos x\, dx \ .$$

Show that the exact answer is 4π. Compute the error in Simpson's rule for various numbers of subintervals N. Perform a convergence test to show that the error is proportional to N^{-4}.

14.5 Integration with SciPy

The Python library SciPy has a number of built–in functions for integration. For example, the following code uses the SciPy function `quad()` to evaluate the integral $I = \int_0^{\pi/2} \sin x \, dx$:

```
import scipy.integrate as si
import numpy as np

def f(x):             # Define the  integrand
    return np.sin(x)

answer, error = si.quad(f,0,np.pi/2)
print(answer, error)
```

The exact answer is 1. The function `quad()` returns the (approximate) answer 0.9999999999999999 along with an error estimate of $1.1102230246251564e-14$.

Exercise 14.5a

The Fresnel integrals defined by

$$S(u) = \int_0^u \sin(\pi x^2/2) dx \ ,$$
$$C(u) = \int_0^u \cos(\pi x^2/2) dx \ ,$$

play an important role in optics. Write a code to create a plot of $S(u)$ and $C(u)$ versus u for $0 \leq u \leq 5$. Use `quad()` to evaluate the integrals.

The integrand might depend on parameters as well as the integration variable. This code computes $\int_0^{\pi/2} a \sin(bx) \, dx$ with $a = 2.0$ and $b = \pi$:

```
def f(x,a,b):          # Define integrand
     return a*np.sin(b*x)

answer, error = si.quad(f,0,np.pi/2,(2.0,np.pi))
print(answer, error)
```

The function `quad()` passes the values `2.0` and `np.pi` to `f()` as the Python "tuple" `(2.0,np.pi)`. The elements of a tuple are enclosed in parentheses. The function `f()` assigns these values to `a` and `b`.

Exercise 14.5b

The Bessel functions of the first kind, defined by

$$J_n(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin t - nt)\,dt\ ,$$

appear in many contexts such as the vibrations of a drumhead. Write a code that plots $J_n(x)$ versus x in the domain $0 \le x \le 20$, for $n = 0, 1, 2, 3$. Use `quad()` to evaluate the integrals. (Note that the integration variable is t.)

Exercise 14.5c

Euler's constant

$$\gamma = \int_1^\infty \left(\frac{1}{\lfloor x\rfloor} - \frac{1}{x}\right) dx$$

plays an important role in mathematics. Here, $\lfloor x\rfloor$ is the floor function, the integer part of x. NumPy has a built–in floor function called `floor()`. The integral definition of Euler's constant is difficult to evaluate because the integrand is discontinuous at integer values of x, and because the range of integration is infinite. Write a code to evaluate γ using the following strategy. Break the range of integration into subintervals from 1 to 2, from 2 to 3, *etc.* Use `quad()` to evaluate the integral in each subinterval and add the results. Keep adding subintervals until the answer settles down to 4 significant figures.

Exercise 14.5d

The gamma function is defined by

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

Since the integrand approaches zero rapidly as x increases, you can approximate the integral by replacing the upper limit of integration with a large positive constant u and using `quad()`. The gamma function has the amazing property that for positive integer values of its argument, $\Gamma(s) = (s-1)!$. Use this as a check to make sure your choice for u is large enough to give reasonably accurate answers.

Plot a graph of $\Gamma(s)$ for the domain $0.2 \leq s \leq 4$.

14.6 Nonuniform data

The `scipy.integrate` function `simps()` can be used to integrate a function whose values are only known at discrete points. This situation can arise when data are obtained from an experiment. For example, let's assume that only the following function values are known:

$$f(1) = 7$$
$$f(2) = 5$$
$$f(4) = 12$$
$$f(5) = 10$$
$$f(8) = 6$$

This code computes the integral of $f(x)$ from $x = 1$ to $x = 8$:

```
x = np.array([1,2,4,5,8])
y = np.array([7,5,12,10,6])

answer = si.simps(y,x)
print(answer)
```

Exercise 14.6

Write a code that will read the data *NonUniformData.txt*, generated by the code given in the Appendix. The data consist of two columns, x and $f(x)$. Have your code plot the data and compute the integral of $f(x)$.

14.7 Monte Carlo integration

In computational science, the term "Monte Carlo" refers to a host of numerical techniques that rely on random sampling to solve a problem. In particular, Monte Carlo methods can be used to evaluate integrals numerically.

Take another look at the midpoint rule, Eq. (13.6). Since $\Delta x = (b-a)/N$, we can rewrite this as

$$I_M = \left[\frac{1}{N}\sum_{i=1}^{N} f((x_{i-1}+x_i)/2)\right](b-a)\ . \tag{14.20}$$

The factor in square brackets is the average value of the function $f(x)$ in the interval from a to b. Let $\langle f \rangle$ denote this average; that is,

$$\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f((x_{i-1} + x_i)/2) \ . \tag{14.21}$$

In this case the average is obtained by evaluating $f(x)$ at N equally spaced points from a to b. As an alternative, we can compute the average $\langle f \rangle$ using N randomly selected points from a to b. Let $\mathcal{X}$ denote a set of N random numbers selected from the integration region $a \leq x \leq b$; then

$$\langle f \rangle = \frac{1}{N} \sum_{x \in \mathcal{X}} f(x) \ . \tag{14.22}$$

The Monte Carlo approximation to $\int_a^b f(x)\,dx$ is simply $\langle f \rangle (b-a)$, or

$$I_{MC} = \frac{(b-a)}{N} \sum_{x \in \mathcal{X}} f(x) \ . \tag{14.23}$$

In the limit as $N \to \infty$, the Monte Carlo approximation I_{MC} approaches the exact value for the integral.

To implement the Monte Carlo method, use NumPy's default random number generator with the function `uniform(`*low,high*`)` discussed in Sec. 8.3. Each time this function is called it creates a random number between *low* (inclusive) and *high* (exclusive).[1] Alternatively, the command `uniform(`*low,high,N*`)` will produce a NumPy array of N random real numbers between *low* and *high*.

Exercise 14.7

Use the Monte Carlo method to evaluate the Fresnel integral $S(u)$ at $u = 10$. Each time you run your code, the answer will be a bit different. How large must N be so that the result is usually accurate to about 2 significant figures?

14.8 Multidimensional integrals

Monte Carlo is not very practical for one–dimensional integrals. It usually takes a lot of random points N to obtain a modest degree of accuracy.

[1] For real numbers, we don't need to worry that *low* is inclusive but *high* is exclusive. After all, the computer can only approximate most real numbers to machine accuracy, as discussed in Sec. 4.4.

Monte Carlo methods are primarily used to evaluate integrals over higher–dimensional regions with complex shapes. For such problems, Monte Carlo can be relatively simple and efficient.

Here's an example. Evaluate the integral

$$I = \int_{\mathcal{R}} \exp(x+y)\, dx\, dy \ , \tag{14.24}$$

where $\mathcal{R}$ is the interior of the ellipse

$$\frac{x^2}{4} + y^2 = 1 \ . \tag{14.25}$$

The Monte Carlo approximation to I is given by $I_{MC} = A_{ell}\langle f \rangle$, where A_{ell} is the area of the ellipse and $\langle f \rangle$ is the average value of the integrand $f(x,y) = \exp(x+y)$ within the ellipse.

In Fig. 14.2 we see that the ellipse is bounded by the rectangle with $-2 \le x \le 2$ and $-1 \le y \le 1$. We can find the average value of the

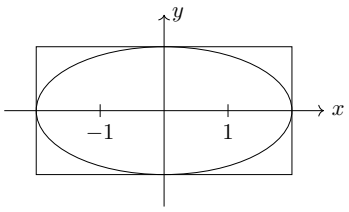

Fig. 14.2: The ellipse $x^2/4 + y^2 = 1$ is contained within the rectangle $-2 \le x \le 2$, $-1 \le y \le 1$.

integrand within the ellipse by choosing random points (x, y) inside the rectangle, and discarding the points that lie outside the ellipse. The area of the ellipse can be found by computing the fraction of random points that lie inside the ellipse, then multplying this fraction by the area of the rectangle. Here is a complete code:

```
import numpy as np

def f(x,y):                    # define the integrand
    return np.exp(x + y)

rnum = np.random.default_rng() # random number generator

Ntot = 200000      # total number of points
sumf = 0.0         # sum of function values
Nin = 0            # number of points inside ellipse
```

```
for n in range(Ntot):           # loop over all points
    x = rnum.uniform(-2,2)      # random x value
    y = rnum.uniform(-1,1)      # random y value
    if x**2/4.0 + y**2 > 1:     # (x,y) is outside ellipse
        continue                # discard that point
    else:                       # (x,y) is inside ellipse
        Nin = Nin + 1           # increment inside points
        sumf = sumf + f(x,y)    # add f to sum

areарec = 8                     # area of rectangle
areaell = (Nin/Ntot)*arearec    # area of ellipse
averagef = sumf/Nin             # average value of f
IMC = areaell*averagef          # answer
print(areaellipse, IMC)
```

Exercise 14.8

Use the Monte Carlo method to evaluate the integral

$$\int_{\mathcal{R}} dx\, dy\, \ln(1+xy)$$

where $\mathcal{R}$ is the region defined by the unit square in the first quadrant with the circle $\sqrt{(x-1/2)^2+(y-1/2)^2} = 1/3$ removed. (See Fig. 14.3.) Use enough points N to deduce the correct answer to three significant figures.

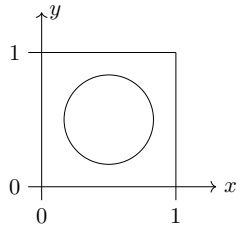

Fig. 14.3: The region of integration for Exercise 14.8 lies between the circle and the square.

14.9 More Exercises

Exercise 14.9a

The *error function* is a special function defined by

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

Write a code to create a plot of $\mathrm{erf}(x)$ versus x for $-3 \leq x \leq 3$. Use the SciPy function `quad()` to evaluate the integral.

Exercise 14.9b

Write a function definition for

$$f(x) = -1 + \int_1^x \frac{1}{t} dt \ ,$$

where Simpson's rule is used to evaluate the integral. Apply the bisection method to find the root of this function. This is one way to determine e, the base of the natural logarithm.