Introduction to Scientific Computing
Lecture 9

Professor Hanno Rein

Last updated: November 13, 2017

1 Integration

In the last two lectures we talked about numerical integration of differential equations. Today we'll talk about something related, solving integrals with a computer. Note that in particular geometric integration methods of differential equations are closely related to solving standard integrals.

We start with the simplest case, an integral of a function of a single variable over a finite range. We can then generalize the concepts to higher dimensions and come up with more accurate algorithms.

1.1 Trapezoidal Rule

Suppose we have the following integral

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

This is the equivalent of calculating the area under the curve. Sometimes, it is possible to perform the calculation exactly, but not in all cases. If it is not possible, we need to use numerical methods to approximate the integral. One way to do that is to approximate the area by dividing it up into multiple rectangles, calculate the area for each, then adding them up. This would be a poor approximation. We can do better without any extra work. The idea is to replace the rectangles with trapezoids. The area under the trapezoids is a better approximation of the area under the curve.

Let us divide the interval from a To b into N slices with width

$$h = \frac{b-a}{N}.$$

Then, the area of the k-th slice is

$$A_k = \frac{1}{2}h\left[f(a+(k-1)\cdot h) + f(a+k\cdot h)\right].$$

We can then calculate the integral as a sum of the individual trapezoids

$$\begin{aligned} I(a,b) \approx \sum_{k=1}^{N} A_k &= \frac{1}{2}h\sum_{k=1}^{N}\left[f(a+(k-1)\cdot h) + f(a+k\cdot h)\right] \\ &= h\cdot\left[\frac{1}{2}f(a) + f(a+h) + f(a+2h) + \cdots + \frac{1}{2}f(b)\right] \\ &= h\left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a+k\cdot h)\right] \end{aligned}$$

Note that we've simplified the trapezoidal rule significantly in these three lines. The last line is calculating the rectangles under the curve. We argued previously that the rectangles are not a good approximation, so why do they appear here again? Note that the boundaries are different! This small change makes all the difference. Of course, for large N this difference will become smaller and smaller and the trapezoidal rule is nothing else than the rectangle rule.

As an example, let us try to integrate

$$\int_0^2 (x^4 - 2x + 1)dx$$

The true value is 4.4. Let us implement the trapezoidal rule using python.

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

N = 10
a = 0.
b = 2.
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)
```

Code 1: Python code of the trapezoidal rule

The output of this code is 4.50656. Only about 2% away from the true value. Obviously, we can increase the accuracy by changing N.

1.2 Simpson's rule

The trapezoidal rule only takes a few lines of code. It uses effectively a linear interpolation between data-points. You might be able to guess what comes next. We will expand the idea to higher order. This is what Simpson's rule does. It uses quadratic curves.

To define a quadratic function, we need three points instead of two. Suppose that as before, we split the interval into N slices with $h = (b-a)/N$. And suppose, for this argument, we have the points at $x = -h$, $x = 0$, and $x = h$. Then we can try to fit the second order polynomial of the form

$$Ax^2 + Bx + C$$

to those point via

$$f(-h) = Ah^2 - Bh + C \qquad f(0) = C \qquad f(h) = Ah^2 + Bx + C$$

Solving these equations gives:

$$\begin{aligned} A &= \frac{1}{h^2}\left[\frac{1}{2}f(-h) - f(0) + \frac{1}{2}f(h)\right] \\ B &= \frac{1}{2h}\left[f(h) - f(-h)\right] \\ C &= f(0) \end{aligned}$$

We can easily integrate the second order polynomial to get

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

This is Simpson's Rule! Note that the final formula only includes h and f evaluated at the grid points. We do not need to perform a quadratic fit every time! It makes Simpson's rule almost as simple as the trapezoidal rule.

Applying Simpson's rule involves slicing up the interval into uniformly spaced points, as before, and then applying the above formula for each interval. At the end, we get the formula for the entire interval:

$$\begin{aligned}I(a,b)\approx\quad & \frac{1}{3}h\left[f(a)+4f(a+h)+f(a+2h)\right]\\ & +\frac{1}{3}h\left[f(a+2h)+4f(a+3h)+f(a+4h)\right]+\ldots\\ & +\frac{1}{3}h\left[f(a+(N-2)h)+4f(a+(N-1)h)+f(b)\right]\end{aligned}$$

We can collect the various term to simplify the algorithm somewhat:

$$\begin{aligned}I(a,b) &\approx \frac{1}{3}h\left[f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+\ldots+f(b)\right]\\ &= \frac{1}{3}h\left[f(a)+f(b)+4\sum_{k\text{ odd}}f(a+kh)+2\sum_{k\text{ even}}f(a+kh)\right]\end{aligned}$$

For the same example as before, with $N=10$ we get 4.00427 as a result. This is 0.01%! Significantly better. Simpson's rule is often the preferred method for evaluating integrals.

1.3 Integrals over infinite ranges

We often encounter integrals over infinite ranges like

$$\int_0^\infty f(x)dx$$

How can we possible integrate those using one of the algorithms we discussed? One way is to use a coordinate transformation such as

$$z=\frac{x}{1+x}$$

or equivalently

$$x=\frac{z}{1-z}$$

Then

$$dx=\frac{dz}{(1-z)^2}$$

and therefore

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

Let's do an example with the Bell Curve

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

Using the above coordinate transformation we get

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

Let's use Simpson's rule to calculate this value.

```
from math import exp
def f(x):
    return 1./(1.-x)**2*exp(-x**2/(1.-x)**2)
N = 100
a = 0
b = 0.9999999999999999
h = (b-a)/N
s = f(a)
s += f(b)
for k in range(1,N,2):
    s += 4.*f(a+k*h)
for k in range(2,N,2):
    s += 2.*f(a+k*h)
s *= h/3.
print(s)
```

Code 2: Python code of Simpson's rule

Note that $\pi = (2I)^2$.

1.4 Multiple integrals

We won't go into details, but consider you want to solve the integral

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

We can rewrite this as

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

where

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

So we can solve the two dimensional integral by solving a series of one dimensional integrals. Note that this now scales as $O(N^2)$ if N is the number of grid points in each dimension. Thus for very large dimensions, this becomes prohibitively large. We will discuss one alternative method to solve integrals in high dimensions.

2 Random numbers

In daily life, random numbers appear all the time. For example when you roll a dice. For a computer, random numbers are difficult to generate because computers are inherently predictable (which is in general a good thing). Thus, coming up with a good random number generator can be tricky. In fact we will call it a *pseudo* random number generator because the numbers are not truly random.

All generators start from a seed value. If you start the generator from the same seed value multiple times, you always get the same answer (because computers are predictable).

Let us first define what we mean by a good random number generator:

- We want long periods for all initial seeds
- Uniformity of distribution for large numbers of generated numbers
- Uncorrelated successive values

Python comes with many random number generators. For example in the `random` package which we can use to generate a random number between 0 and 1:

```
import random
random.seed(1)
random.random()
```

Every time you execute these commands, you will get the same. Note that when you remove the seed line, then python tries to pick a smart seed by itself. This might depend on the time, the current number jobs running on your computer, etc. This gives the illusion that the first number you get is truly random, however, it's not!

Why would we ever care about the details of whether a random number is truly random or predictable. There are many reasons:

- Suppose you are playing a computer game against the computer. You don't want to be able to predict the computer's next move.
- Random numbers are crucial when it comes to cryptography. All your banking information, snapchat messages, etc rely on random numbers!
- Many numerical algorithms require random numbers. We will talk about some of those. For such an algorithm to work well, we need to have *good* random numbers.

We will not discuss algorithms for random number generators in this course. But you should be aware of the issues that they might have (reproducible results, correlations, etc).