Numerical Methods

Math 3338 – Spring 2022

Worksheet 7

Integration Errors

1 Reading

CP	5.2, 5.3, 5.4
NMEP	6.3

Table 1: Sections Covered

2 Error

We know how to calculate integrals for a set number of steps. However, we typically care far more about accuracy. We know that more more steps will produce more accuracy, but we also don't want our computer to run forever.

We're going to focus on the trapezoidal method, there are similar rules for Simpson's method but they aren't really necessary. We're also going to concentrate on a method that doesn't require that we know the explicit function being integrated. This is useful as we typically don't know the function. If we have data given by an experiment, it may just give a sequence of points.

Suppose we use N_1 subintervals to get an approximate area of I_1. The width of each subinterval is $h_1 = \frac{b-a}{N_1}$. The trapezoidal method gives an error $O(h_1^2)$ (the book proves this), if I is the actual area, then

$$I = I_1 + O(h_1^2) = I_1 + ch_1^2$$

where c is some constant. Double the number of steps, $N_2 = 2N_1$ say the area is I_2. Now, our width is $h_2 = \frac{1}{2}h_1$ and the area is

$$I = I_2 + O(h_2^2) = I_2 + ch_2^2$$

Set these two equations equal, the error is $I_2 - I_1$,

$$I_2 - I_1 = ch_1^2 - ch_2^2 = 3ch_2^2$$

In reality, the error is $\epsilon = ch_2^2$, so

$$\epsilon = \frac{1}{3}(I_2 - I_1)$$

This is useful! If we want our area to be accurate up to 4 decimals, we keep doubling the number of subintervals until $\epsilon < .0001$.

The naive thing to do would be to set up a loop and continually double the number of steps until until you get an acceptable error. This is naive because you'll be repeating computations, you'd have to compute $f(a)$ every single time. It's more efficient if each time you double the number of intevals, you only add the

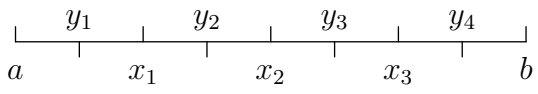

Figure 1: Breaking an interval

new steps. For example, in Figure 1 there are initially 4 subintervals, denoted under the axis. Doubling this to 8 adds the y's above the axis. The trapezoid method for the 8 intervals would be similar to,

$$\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{3} f(x_i) + \sum_{i=1}^{4} f(y_i).$$

Notice the first three terms are the trapezoid rule for 4 intervals. Note that it's not exactly this. Figure it out.

3 Romberg Integration

If our goal is to minimize error, it seems counter intuitive to rely on the trapezoid method. We've already seen Simpson's rule leads to a better approximation. What if there was a way to extend the trapezoid rule to be more precise than Simpson's rule, while simultaneously being easier to code?

First, using Talyor series we could show the error of the trapezoid method is,

$$I = I_i + c_1 h_i^2 + c_2 h_i^4 + c_3 h_i^6 + \cdots \tag{1}$$

Where I_i is the approximation using steps with width h_i. If we want a certain approximation error, we truncate the summation. So $I = I_i + c_1 h_i^2$ is accurate to $O(h_i^4)$. Similar to what we did before, let's say $h_{i-1} = 2h_i$, then the $(i-1)$-approximation is

$$\begin{aligned} I =& I_{i-1} + c_1 h_{i-1}^2 + c_2 h_{i-1}^4 + \cdots \\ =& I_{i-1} + 2^2 c_1 h_i^2 + 2^4 c_2 h_i^4 + \cdots \end{aligned} \tag{2}$$

Equations (1) and (2) are equal. Truncate after the h_i^2 term and subtract to find,

$$0 = (I_i - I_{i-1}) - (2^2 - 1)c_1 h_i^2$$

Solve this for $c_1 h_i^2$ to find,

$$c_1 h_i^2 = \frac{1}{4^1 - 1}(I_i - I_{i-1}).$$

Then

$$\begin{aligned} I =& I_i + \frac{1}{4^1 - 1}(I_i - I_{i-1}) + c_2 h_i^4 + c_3 h_i^6 + \cdots \\ =& R_{i,2} + c_2 h_i^4 + \cdots \end{aligned}$$

is accurate to order 4. Notice we've used the variable $R_{i,2} = I_i + \frac{1}{4^1-1}(I_i - I_{i-1})$. This is essentially a free computation, as we would have calculated them anyway.

This is the core of Romberg integration. The general formula is,

$$R_{i,1} = I_i \qquad\qquad R_{i,m} = R_{i,m-1} + \frac{1}{4^{m-1} - 1}(R_{i,m-1} - R_{i-1,m-1}).$$

Where I_i is the result of trapezoidal integration with 2^i subintervals. The derivation of this is inductive. Suppose we know $R_{i,k}$ for all $i \leq n$ and $1 < k \leq m$, then

$$I = R_{i,m} + c_m h_i^{2m} = R_{i-1,m} + 2^{2m} c_m h_i^{2m}$$

Solve for $c_m h_i^{2m}$ to find,

$$c_m h^{2m} = \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m})$$

which gives the Romberg recursion formula.

Numerical Methods

Math 3338 – Spring 2022

Homework 7 (Due: Tuesday, February 8)

Problem 1 (1 pt) Write two functions that evaluate the trapezoid to a desired degree of accuracy. The inputs to both will be the same, f, a, b, and tol which should default to $1e-6$, or 10^{-6}. This is the tolerance, and your answer should be at least that accurate. Both functions should return an answer of the form `(approximation, number of steps)`.

Call the first function `trap_naive`. This function should use your old `trapezoid` method to compute each approximation.

Call the second `trap_error`. For this function you should be clever and only compute the new steps at each iteration.

Evaluate $\int_0^1 \sin^2(\sqrt{100x})\,dx$ for several small tolerances $tol \in \{10^i | -13 \leq i \leq -1\}$. Make a table that contains the tolerance, the number of steps, and time for both methods. Discuss your observations.

Problem 2 (1 pt) Write a function called `romberg`. The inputs are f, a, b, and tol which should default to $1e-6$, or 10^{-6}. This is the tolerance, and your answer should be at least that accurate. Both functions should return an answer of the form `(approximation, number of steps)`.

Evaluate $\int_0^1 \sin^2(\sqrt{100x})\,dx$ for several small tolerances $tol \in \{10^i | -13 \leq i \leq -1\}$. Make a table that contains the tolerance, number of steps and time for both `romberg` and `trap_error`.

Compare and discuss.

Problem 3 (1 pt) Many of these methods already exist. Let's compare ours to the pre-built ones. Here is a help page `https://docs.scipy.org/doc/scipy-0.14.0/reference/integrate.html`. This is part of the scipy package. Put the following at the top of your file `from scipy import integrate`, now we have access to the integrate package.

Make a table that compares the run time of `romberg`, `integrate.romberg` and `integrate.quadrature`. To set the tolerance you'll need to use the keyword `tol` in the later two. Do this for $tol \in \{10^i | -13 \leq i \leq -1\}$.

Compare and discuss the run times for each. Note that I noticed something strange with the quadrature times, if something similar happens on yours, you should mention it.