Homework 4:

John Wu

13 April, 2020

**Executive Summary:**

In this assignment, we explored methods of integration using computers. In particular, we used two methods of computing integrals: composite trapezoids and Simpson’s 1/3 rule.

**Description of Mathematics:**

Composite Trapezoids: With composite trapezoids, we use the basic fundamentals that we learned in calculus. That is to say that we take a function and place multiple (sometimes equidistant) points on said function and calculate the area between each point using trapezoids. This way, we are able to simplify the integral to a series of trivial computations. Simplified, this can be represented as:

Simpson’s 1/3 Rule: With this method, we are able to find a more accurate estimate of our integral with ease. This method takes advantage of the fact that when we take our points to be equally spaced, we can use three points to simplify our evaluation to simply:

Furthermore, when taking into account that we may want more than three points, we can further represent it as such:

Where x­i represents all odd nodes and xj represents all even nodes.

**Implementation:**

The implementation for the code was pretty simple all-in-all. For the composite trapezoid rule, it was as easy as plugging in the equation shown above. Taking in inputs of: fname, a, b, and n, I first began by computing the step size (h) needed for my evaluation. By taking the end points of my interval, finding the difference, and dividing by n. I then evaluated the function’s values at the endpoints, setting my final output q equal to . Then, I created a for loop, which starts at h (to take into account that I’ve already evaluated the function starting at a, therefore I need to start from 1 step up) and increments by h until it reaches however many points there are multiplied by h. This is so I can easily refer to the value I am wanting to evaluate my function at by calling my function with the parameters “a+i”, adding that value to my final output q each time.

Then, for the composite Simpson’s 1/3 rule, I started out the same way as in the composite trapezoid rule. I computed h, established my starting value of and I added an arbitrarily named counter, more on that later. My for loop was also largely the same with the exception of a few small changes. Firstly, we know that in the composite Simpson’s 1/3 rule, we have 2 summations rather than just one. I’m not sure how much of a difference it would make in the long run, but I wanted to attempt to make this function with just one for loop rather than nesting two into an outer for loop. Thusly, I created an extra counter that would keep track of which iteration I was running on, using this counter to determine whether I was on a positive or negative i value. This was achieved by taking the value of the function at whatever point I was on, splitting it in half and multiplying on of the halves by (-1)i . It’s a very convoluted way to solve for it, and it might not even help with regards to speed, but I wanted to challenge myself.

**Results:**

In the end, the results have yet to betray my expectations, I of course tested this using some of the problems we solved in our books to test the validity of my code. I think the source that most validates the accuracy of my code though is that it always converges to a number, getting more and more accurate as it goes. Not too dissimilar to that of our last programming assignment when we were trying to find roots. I have tested multiple functions with varying amounts of segments allowed, and found that the results converged closer and closer each time to the number that I hand calculated in the written portions of the assignment.

**Conclusions:**

After finishing this assignment, I have come to realize that numerical methods isn’t as hard as I had initially made it out to be. At least, when you start to understand it. Seeing my results and comparing them with my hand calculations, I know that these estimates are somewhat in the ballpark of what the answer should be, which is what I’d expect. However, I do think it is worth noting that there are instances in which these calculations are a bit further off than I’d expect as well. It’s quite evident that these sources of error could be due to rounding error or some other fashion of machine limitation, but I think there’s room to be made for my own errors as well. At the time of writing this, it’s still up in the air as to whether my Simpson’s evaluation is accurate, seeing as my implementation of it may not be foolproof when it comes to the summation of the even and odd intervals.