Corey Schulz
CS 3200 – Spring 2020
February 22, 2020

Homework 2 Report

Question 1

Simpson's rule for integrating a curve is as follows:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$$

…This approximates a definite integral from a to b.

Here are my results from implementing this function in Matlab and trying it out with different variations of x ^ (number) as well as with different values of N. In the following table, the "Actual" column refers to the definite integral value for the function on the range [0, 1].

Function	N = 17	N = 33	N = 65	N = 129	N = 257	N = 513	Actual
x^2	0.3586	0.3449	0.3389	0.3360	0.3347	0.3340	0.3333
x^3	0.2779	0.2623	0.2557	0.2527	0.2513	0.2507	0.2500
x^4	0.2304	0.2131	0.2059	0.2028	0.2013	0.2007	0.2000
x^5	0.1994	0.1804	0.1728	0.1695	0.1680	0.1673	0.1667
x^6	0.1779	0.1573	0.1491	0.1457	0.1442	0.1435	0.1428
x^8	0.1504	0.1268	0.1178	0.1141	0.1125	0.1118	0.1111

…In all cases, the estimated version got increasingly more accurate as N increases, converging on the actual value of the definite integral. However, there *are* diminishing returns, as it does not scale linearly.

Function	Difference (Best approximation to actual)
x^2	.0007
x^3	.0007
x^4	.0004
x^5	.0006

x^6	.0007
x^8	.0007

(Here, each of the numbers are approximated, of course.)
The difference remains roughly constant for each of the functions – this method of approximating the integration did not necessarily perform better on one function over another. However, the approximations in any case with a sufficiently large N required much less computation than evaluating the definite integral's actual value.

The definite integral

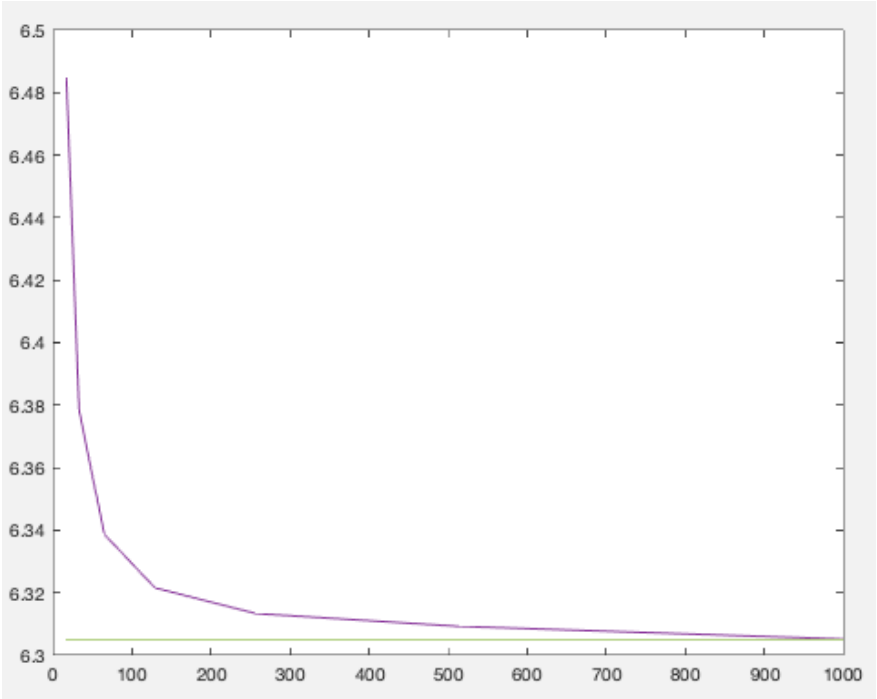
$$\int_0^{2\pi} 1 + \sin(x) \cdot \cos\left(\frac{2x}{3}\right) \cdot \sin(4x)\, dx$$

…Evaluates to:

$$\int_0^{2\pi} \left(1 + \sin(x)\cos\left(\frac{2x}{3}\right)\sin(4x)\right) dx = \frac{216\sqrt{3}}{17017} + 2\pi \approx 6.305171$$

…This potentially requires much computation to compute without approximation. Simpson's rule with different values of N yields the following data:

N = 17	N = 33	N = 65	N = 129	N = 257	N = 513	Actual
6.4849	6.3785	6.3387	6.3216	6.3133	6.3093	6.305171



A convergence plot of the above definite integral and its Simpson's Rule counterpart.

Here, the yellow line is the actual value of the definite integral, and the blue is the convergence of Simpson's Rule. I had to extend N to 1,000 to get the plot to converge, but it *does* eventually get quite close to the actual value of the function! Here especially, getting an approximation of the function proves useful, as it is much less computationally complex than calculating the actual value of the definite integral.

Question 2

The definite integral:

$$\int_0^3 (\cos(x^3))^{200}\, dx$$

…Has a value of:

$$\int_0^3 \cos^{200}(x^3)\, dx = 0.531594$$

This can be approximated, again with potentially much less computation with the QuadTX functions (*and a corresponding tolerance*), as shown below:

Tolerance	Integral approx.	Function Calls	Time
1.0e-7	0.5222	813	0.005547 seconds
1.0e-8	0.5316	1365	0.001805 seconds
1.0e-9	0.5316	2089	0.002560 seconds
1.0e-10	0.5316	3177	0.004124 seconds
1.0e-11	0.5316	5013	0.006866 seconds
1.0e-12	0.5316	7841	0.008282 seconds
1.0e-13	0.5316	12241	0.012433 seconds
1.0e-14	0.5316	19545	0.019248 seconds

In general, as the number of function calls goes up, so too does the time spent on the computation of the approximation. In this case, one could argue that the increase in time – though small – isn't worth it past a tolerance of 1.0e-8 because the approximation remains converged on the actual value. If the tolerance was 1.0e-14, for example, you'd have 10x computation time to converge on the same answer versus if the tolerance was larger to begin with. It's a balancing act between making the tolerance too large such that you get a wrong answer and not making the tolerance too small such that the computation time goes up.

Part 2: Changing tol to tol h/(b-a)

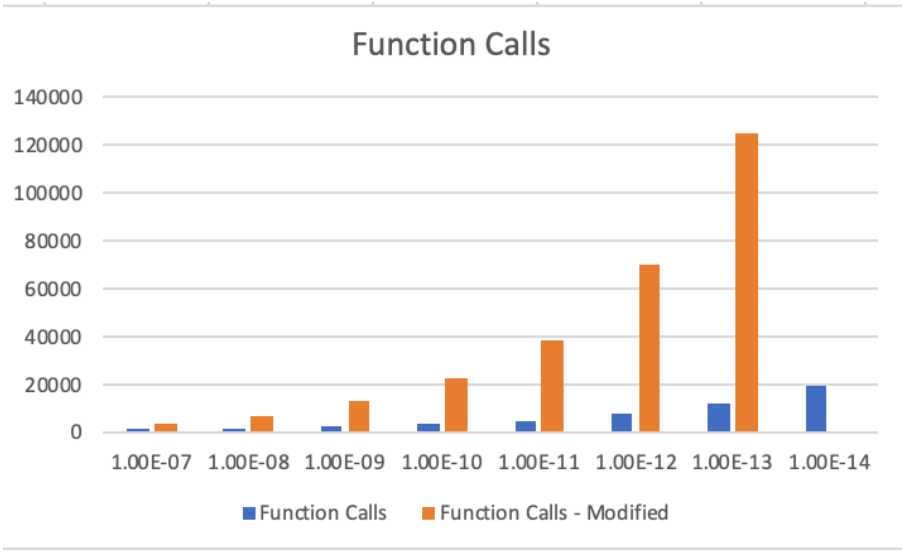
As described in Section 6.3 of the textbook, this problem asks you to cut in half the tolerance (tol / 2) every recursive step.

I did that and re-ran the same calculations as in the first part of this problem, gathering the following data:

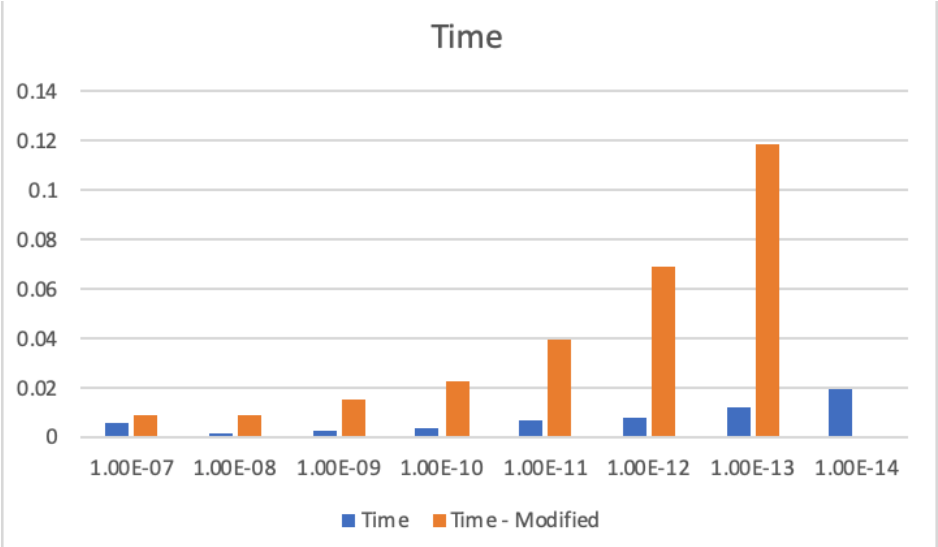
Tolerance	Integral approx.	Function Calls	Time
1.0e-7	0.5316	4101	0.008667
1.0e-8	0.5316	7213	0.008512
1.0e-9	0.5316	12709	0.015234
1.0e-10	0.5316	22285	0.022260
1.0e-11	0.5316	38645	0.039417

1.0e-12	0.5316	69497	0.068903
1.0e-13	0.5316	125205	0.118049
1.0e-14	Stack Overflow	Stack Overflow	Stack Overflow

Here, this data is much less favorable than before.



This chart compares the number of function calls – dividing the tolerance in two every recursive step makes the number blow up and, by the time the original tolerance is as small as 1.0e-14, there are too many function calls and a stack overflow is achieved.



Additionally, the amount of time the function takes to complete grows at roughly the same rate as the function calls increase. Again proving, it's not worth it to change the tolerance bound within the recursive call – it takes longer, more flops are required, and there's a chance you can't recurse far enough to return an answer. Overall, it's not worth it to do this. This is a good example, however, of testing code to make sure it's the most efficient it can be. A similar method is used when picking variables during cross validation in machine learning systems.

Question 3

For this question I changed my code from Question 2 to estimate the overall error – both in the case of QuadTX and Simpson's Rule – and show again how many steps QuadTX takes to converge on a solution.

Firstly, here's how many function calls it took QuadTX to converge, as well as the errors resultant for that data.

Tolerance	Integral approx.	Function Calls (Steps)	Time	Error
1.0e-7	0.5222	813	0.005547 seconds	0.0094
1.0e-8	0.5316	1365	0.001805 seconds	0

1.0e-9	0.5316	2089	0.002560 seconds	0
1.0e-10	0.5316	3177	0.004124 seconds	0
1.0e-11	0.5316	5013	0.006866 seconds	0
1.0e-12	0.5316	7841	0.008282 seconds	0
1.0e-13	0.5316	12241	0.012433 seconds	0
1.0e-14	0.5316	19545	0.019248 seconds	0

And for comparison, here's Simpson's Rule on that same integral (the one from Question 2).

Integral approx.	N (Steps)	Time	Error
0.4866	17	0.002121 seconds	0.0450
0.4711	33	0.000154 seconds	0.0605
0.4818	65	0.000216 seconds	0.0498
0.5376	129	0.000158 seconds	0.0060
0.5432	257	0.001546 seconds	0.0116
0.5317	500	0.000104 seconds	0.0001
0.5316	1000	0.000155 seconds	0
0.5316	2000	0.000106 seconds	0

There's something interesting here. It took around 1,000 steps for both Simpson's Rule and QuadTX to converge on something with an error of zero – close enough to the actual value that the computer can't know the difference, at least. *However,* the time it takes for Simpson's Rule

to converge on this same value is much *less* than that of QuadTX because, simply, it doesn't use recursion. That adds a lot of the computational overhead here in determining the solution to the definite integral.

I tried this on a couple of other fixed grid cases – I'll save you the tables here, though to save space as they roughly all show the same thing: the same answer can be achieved within roughly the same number of steps between QuadTX and Simpson's Rule, but in the case of Simpson's Rule it doesn't use recursion and hence has an inherent advantage in speed over QuadTX. This comes through in speed. QuadTX can have its advantages – it can be graphed much more easily than Simpson's because of its recursive nature. But, an efficient Simpson's Rule code seems to be dominated by QuadTX in terms of time complexity.

It *is* interesting how they appear to converge after roughly the same number of intervals (N, in the case of Simpson's Rule and recursive calls in the case of QuadTX) in both cases.