Dan Ruley
CS 3200
Assignment #2
February 26, 2020

1.)

I used the Composite Simpson's Rule to approximate $\int_0^1 x^p\,dx$ for p $\in$ [2,3,4,5,6,7,8] using mesh sizes of 17, 33, 65, 129, 257, and 513. I calculated the actual values and compared it to the Simpson approximation. See the table below for these results.

Table 1: Simpson's approximation of $\int_0^1 x^p\,dx$

	p = 2	p = 3	p = 4	p = 5	p = 6	p = 7	p = 8
N=17	0.3333333333	0.25	0.2000000998	0.1666669161	0.1428576414	0.125000872	0.1111125051
N=33	0.3333333333	0.25	0.200000007	0.1666666842	0.142857178	0.1250000615	0.1111112094
N=65	0.3333333333	0.25	0.2000000005	0.1666666678	0.1428571452	0.1250000041	0.1111111176
N=129	0.3333333333	0.25	0.2	0.1666666667	0.142857143	0.1250000003	0.1111111115
N=257	0.3333333333	0.25	0.2	0.1666666667	0.1428571429	0.125	0.1111111111
N=513	0.3333333333	0.25	0.2	0.1666666667	0.1428571429	0.125	0.1111111111

Actual Values:

p = 2	p = 3	p = 4	p = 5	p = 6	p = 7	p = 8
0.3333333333	0.25	0.2	0.1666666667	0.1428571429	0.125	0.1111111111

It seems that Simpson's approximation is quite accurate. Below p = 4, the answer is exact even for the small meshes. This makes sense because we discussed in class how the accuracy is influenced by the fourth derivative and for p<4, the derivative will be 0.

I also used the Simpson's Composite Rule to approximate:

$$\int_0^{2\pi} 1 + sin(x) * cos(\tfrac{2x}{3}) * sin(4x)\,dx$$

The mesh sizes were the same as before. This also seemed to be an accurate approximation. I calculated the actual value of the integral using Wolfram Alpha and compared it to my results.

Simpson's converges to 6 decimal places for only N = 33; however, it takes N = 513 to achieve convergence to 9 decimal places. See Figure 1 and Table 2 for the results.

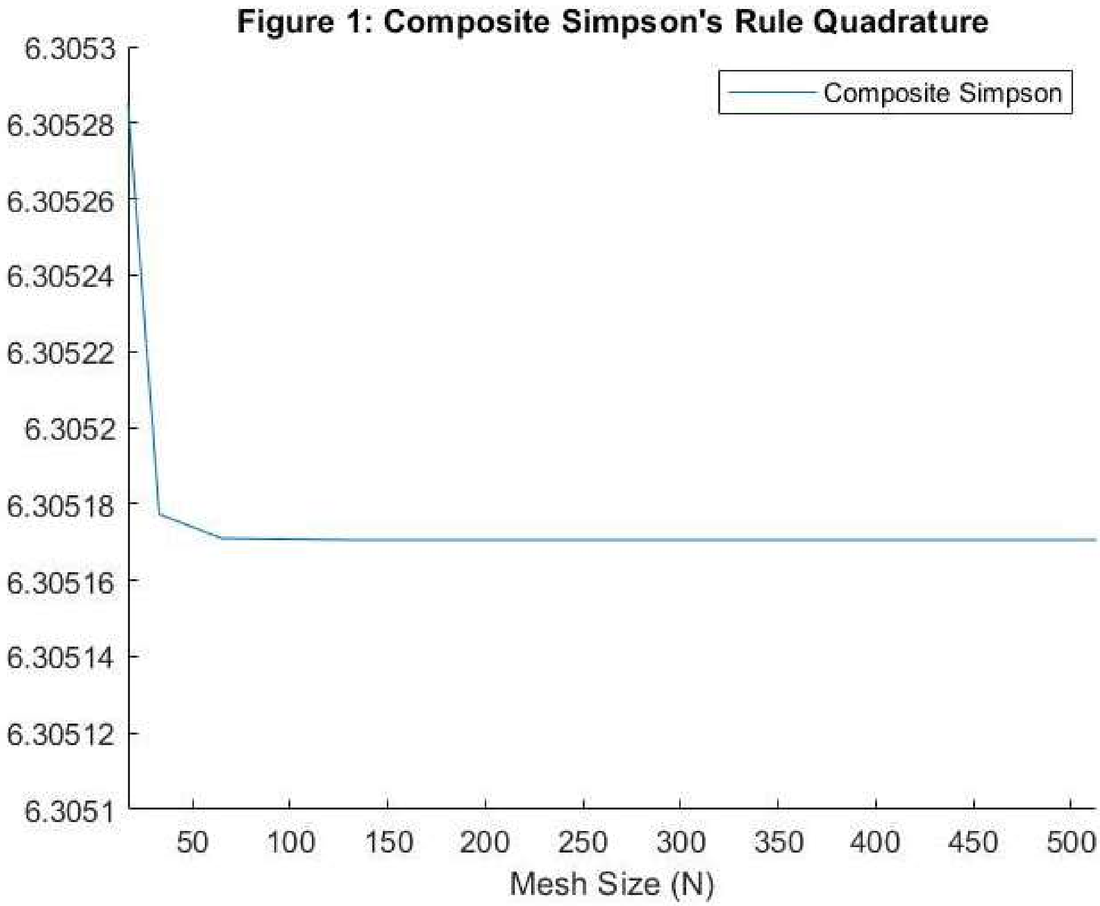

Table 2: Composite Simpson's vs. Actual Value

	Simpson's	Actual	Difference
N=17	6.305285071...	6.305170556...	0.0001145150025
N=33	6.305177291...	6.305170556...	0.00000673544645
N=65	6.305170983...	6.305170556...	0.0000004269332203
N=129	6.305170583...	6.305170556...	0.00000002718959013
N=257	6.305170557...	6.305170556...	0.000000001720660059
N=513	6.305170556...	6.305170556...	0.0000000001083000356

2.)

I used the Quadtx function from the book to approximate $\int_0^3 cos(x^3)^{200}\, dx$, using the tolerance values of 1.0e-6, 1.0e-7, …, 1.0e-14. I recorded the results and compared them to the value calculated by the Matlab integral function as well as the number of function calls quadtx made to achieve the result. See Table 3 below for these results:

Table 3: Quadtx Approximation and Function Calls

Tol	QuadTX	Actual	Function Calls	Difference
1.00E-07	0.5221864391	0.531594452	813	0.009408012881
1.00E-08	0.5315944606	0.531594452	1365	0.000000008591
1.00E-09	0.5315944526	0.531594452	2089	0.000000000616
1.00E-10	0.531594452	0.531594452	3177	0
1.00E-11	0.5315944519	0.531594452	5013	0
1.00E-12	0.5315944519	0.531594452	7841	0
1.00E-13	0.5315944519	0.531594452	12241	0
1.00E-14	0.5315944519	0.531594452	19545	0

It is clear that this is a quite good approximation. Even with the lowest tolerance of 1.0E-7, the quadtx value is quite close to the actual value. The number of function calls required to calculate quadtx seems to increase at a fairly steady rate.

Next, I modified quadtx to make recursive calls on tol * 0.5 instead, as described in section 6.3 of the book. This dramatically increased the number of recursive calls quadtx makes, and actually caused a stack overflow error. I eliminated this by adding a condition that the modified quadtx should only do the recursive call on tol * 0.5 if this value is greater than 1.0E-25. This slightly changes the value for the final tol = 1.0E-14, but still illustrates the effect without causing a runtime error in the program. See Table 4 for these results.

Table 4: Modified Quadtx Function Calls

Tol	Mod. QuadTX	Actual	Function Calls	Difference
1.00E-07	0.5315944523	0.531594452	4101	0.00000000031
1.00E-08	0.5315944522	0.531594452	7213	0.00000000019
1.00E-09	0.5315944519	0.531594452	12709	0
1.00E-10	0.5315944519	0.531594452	22285	0
1.00E-11	0.5315944519	0.531594452	38645	0
1.00E-12	0.5315944519	0.531594452	69497	0
1.00E-13	0.5315944519	0.531594452	125205	0
1.00E-14	0.5315944519	0.531594452	3228673	0

We can see that the modified Quadtx is more accurate. However, it requires significantly more function calls since it recurses much more across the intervals of the functions that are "spiky". The number of function calls increases geometrically as the tolerance decreases.

3.)

I approximated the same function using the Simpson's Composite quadrature, using meshes in the range [50, 1000] and increasing by a step of 50. Then, for every N > 50, I calculated the error of $I_{h/2} - I_h$ by subtracting the values. I then used these values as the tolerance and called the quadtx function with these values. I then recorded how many function calls quadtx made for each error value. See Table 5 for these results.

Table 5: Quadtx Function Calls for Simpson's Error $I_{h/2} - I_h$

N	Error $I_{h/2} - I_h$	Quadtx Calls	N	Error $I_{h/2} - I_h$	Quadtx Calls
50	N/A	N/A	550	1.65E-05	217
100	0.08168001152	17	600	2.15E-05	177
150	0.02421002044	17	650	5.23E-06	357
200	0.00510971364	21	700	1.32E-07	765
250	0.00303803867	21	750	1.08E-06	521
300	0.00132035605	25	800	2.69E-07	677
350	0.00118453078	25	850	5.32E-08	905
400	0.00040518797	29	900	2.40E-08	1185
450	0.00027696817	41	950	1.35E-09	1965
500	0.00015368304	77	1000	7.10E-10	2221