ISC - 3313

Capstone Project

1. Root Finding Methods:

1.b: Compare results:

Iterations to find root with Bisect method: 14.
The root is 142.731323.

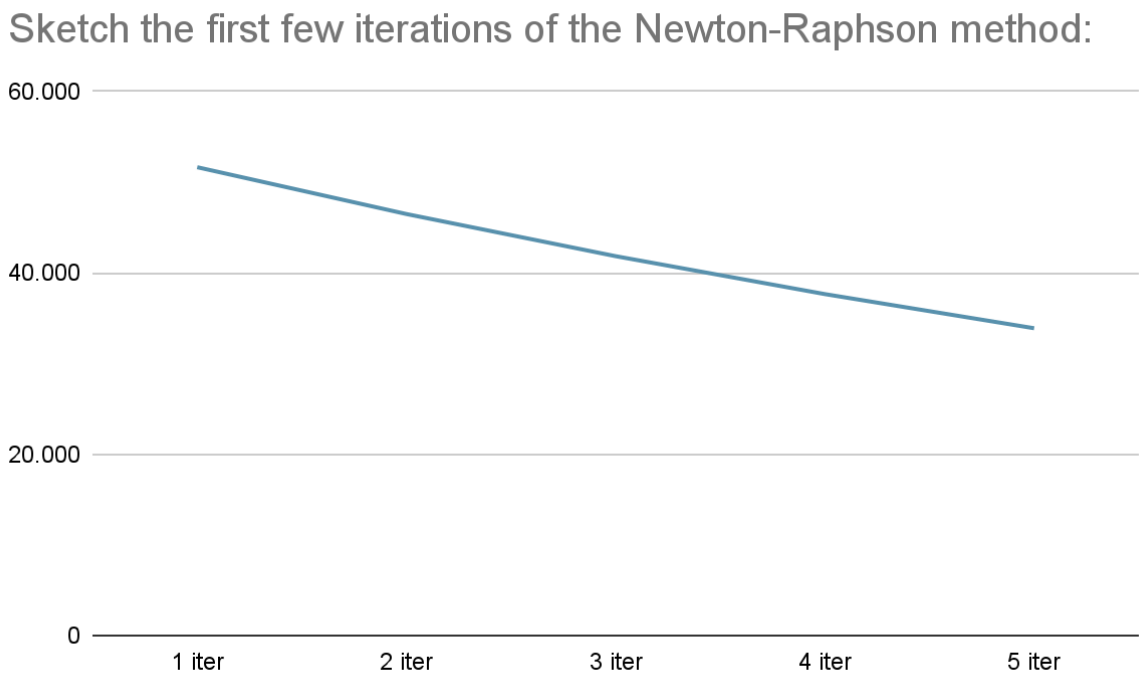
Iterations to find root with Newton Raphson method: 6.
The root is 142.737633.

Iterations to find root with Secant method: 8.
The root is 142.737633.
.

1.c: Case Study:

Iterations to find root with Bisect method: 13.
The root is 1.00002441.

Iterations to find root with Newton Raphson method: 42.
The root is 1.



As the graph shows, with each iteration the program gets closer and closer to the actual value of the root. Starting at 51.650 with the first iteration and reaching the actual value of the root (which is 1) at 42 iterations.
When comparing the Newton-Raphson and Bisection methods, we can see that the Bisection method is able to find the root with fewer iterations but the Newton-Raphson method gives a very slightly more accurate value of the root.

2. Optimization:

2.a: Explain why the current GoldenSectionSearch code only finds the minimum values of a function:

With the current code, we are only able to do GoldenSectionSearch for the lowest point of a function f(x) by reducing the size of an interval towards the minimum.. However, by introducing a coefficient (1 or -1) that multiplies our current points in f respective of the current upper and lower boundaries of the interval (f(x1) and f(x2)), we allow the code to find the opposite of the lowest point when the coefficient is -1 instead of 1, by reducing the size of an interval towards the maximum..

2.c : why does the ParabolicInterpolation method not return the optima for the following function? f(x) = -x^5 + 2x^2 +1

When using this particular function the ParabolicInterpolation method was dividing by zero when getting the value of x4, this will return a value of x4 equal to NaN. This could be fixed by dividing by the lowest possible value in a double instead of 0 when the denominator in x4 is equal to 0.

3. Integration:

3.b: Fill in the table:

n	I	Percent Relative Error
1	1.9648	19.7659298
2	1.82257778	11.0966623
3	0	100
4	1.64053333	6.22605565e-13
5	1.08165689	34.0667534
6	1.60453333	6.63210276e-13

3.c: How might you modify your code to compute the integral to a desired tolerance?

Similar to how I did my Secant method program, I could make a while loop that has “Error > Tol” as one of its conditions that gets the value of “I” .
“Error" would be given a value of 1 when it is initialized/declared, and “Tol” would be taken as an input in the function.