Alex Merino

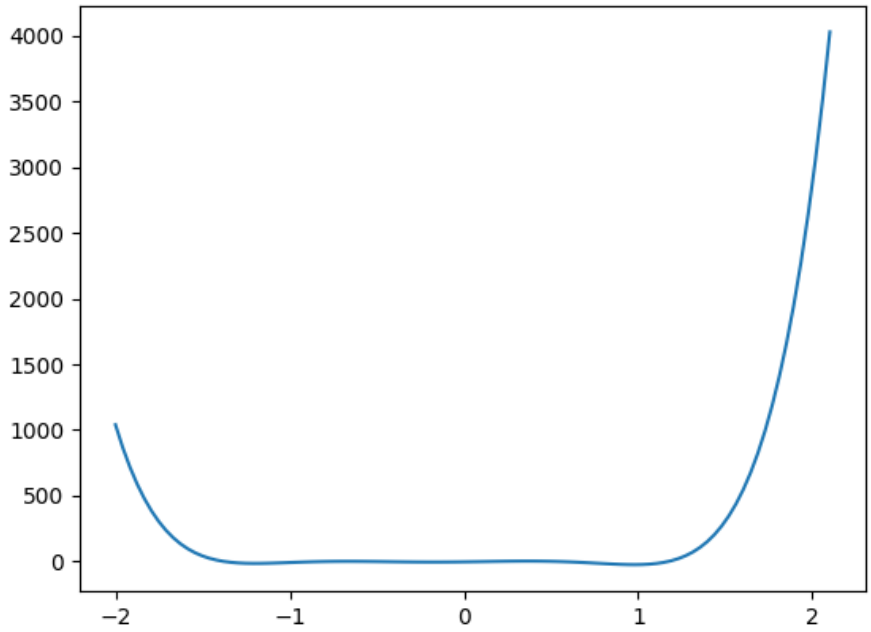
MTH 3312

Due: 9/25/2024

All code documented in the python files as well.

1.

a. In the program q1.py, the function *partA* is called which creates an array between -2 and 2 and applies the array to the function then plots the resulting graph.



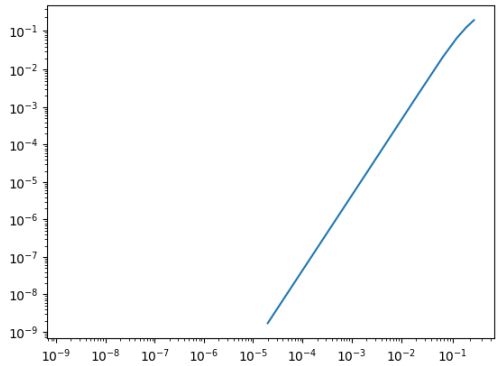
b. The program calls function *partB* which contains 5 programmer defined calls to the *NewtonsMethod* function. The *NewtonsMethod* function is implemented to calculate the zero's of the function within a certain programmer defined tolerance (8 decimal places in our case). After calculating the root, the function outputs a sentence displaying the root,

similarly to the output below.

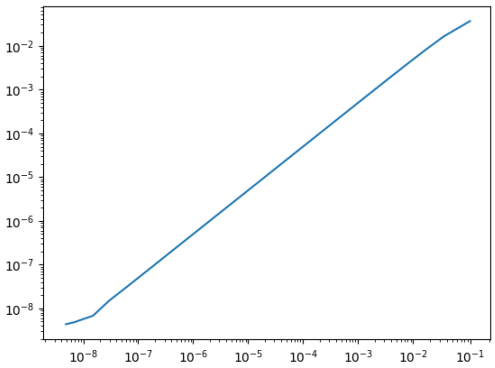
```
Starting at x-value -2, we get the root: -1.381298482
Starting at x-value -0.5, we get the root: -0.666666668
Starting at x-value 0.1, we get the root: 0.205182925
Starting at x-value -0.1, we get the root: 0.500000000
Starting at x-value 1, we get the root: 1.176115557
```

c. Based on the graphs below, the convergence looks to be quadratic, for all of then expect for the second graph, which is the root -2/3 which seems to be linear. The graphs go from right to left since the error is getting smaller, so for all the graphs it starts out slowly then speeds up which can be seen in the difference in the slopes of each part of the graph.

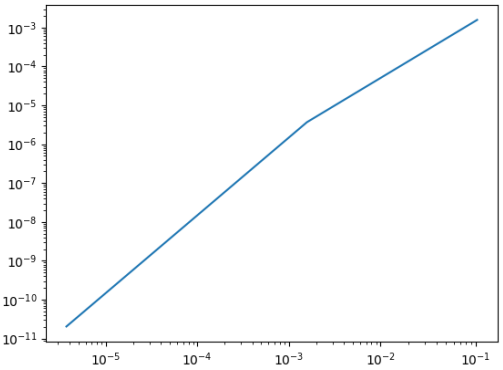
Root: -1.381



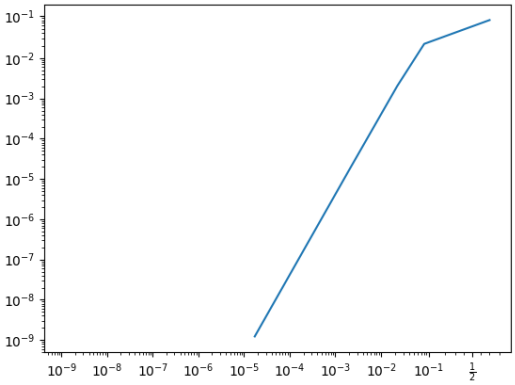
Root: -2/3



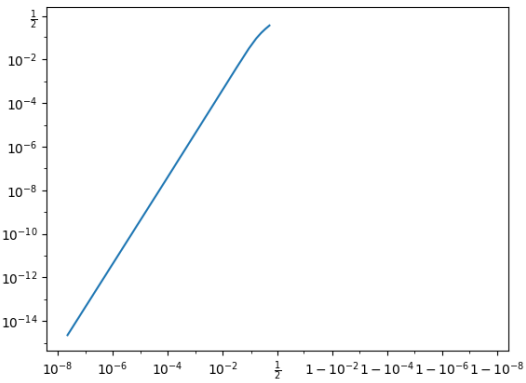
Root: 0.205



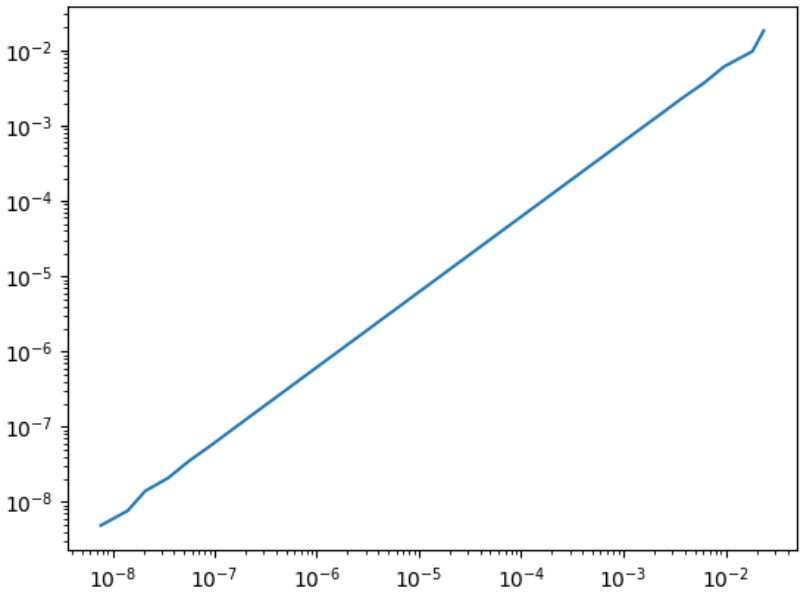
Root: 1/2



Root: 1.1761



d. For this question the function *partD* is called and it contains a call to the secant method, which takes in two parameters. The parameters are the first two numbers in the list. The root is still the same as the Newtons Method, but the error plot looks to be more quadratic with the Secant Method.



2. For question 2, the q2.py program is used. In the program is the *main* function, *GaussElim* function, and *HilbertMat* class. The *HilbertMat* class, instantiates a Hilbert Matrix of size n which is a parameter for the class. The *GaussElim* function does the elimination and back substitution step of gaussian elimination given a matrix A and vector b to solve the equation Ax = b. The main function creates 10 Hilbert Matrices sizes 100, 200, …, 1000. My computer took too long to run larger matrices so I divided the suggested sizes by a factor of 10. The loop then solves for x and saves the time it took to solve for gaussian elimination. The graph below is the time vs size of matrices and it is clear that it is time complexity n^3 because of the way the time increased cubically when increasing the size of the matrix.