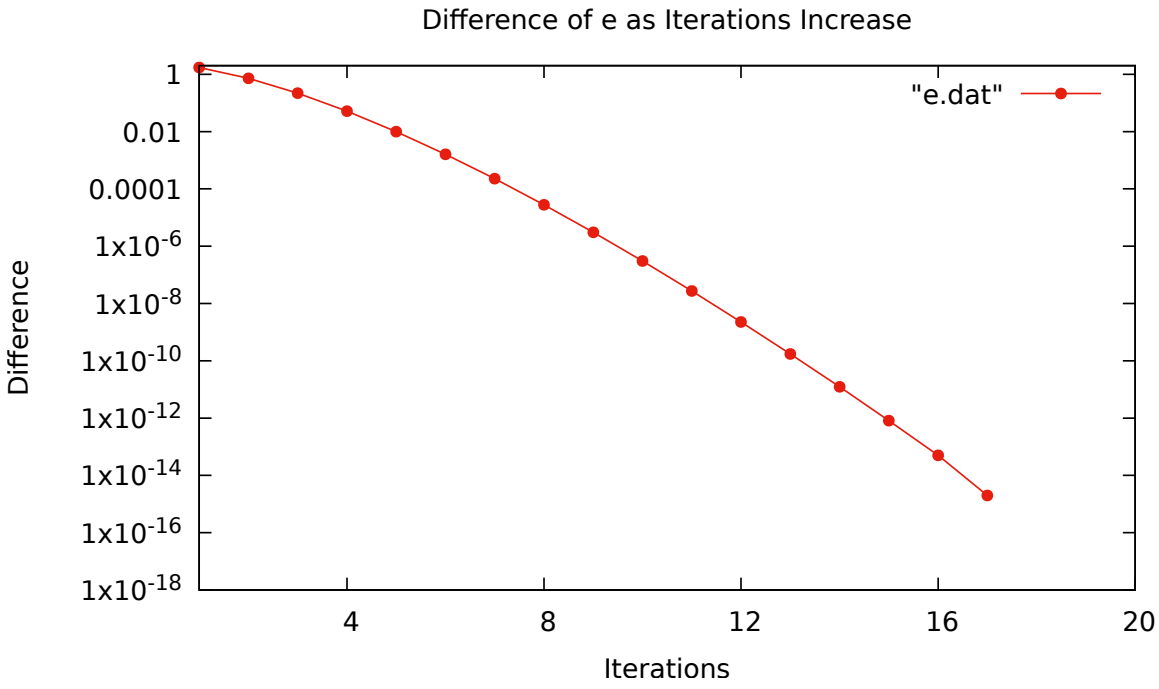
# Differences Between my Approximations and the Math Library

Alex Yeh

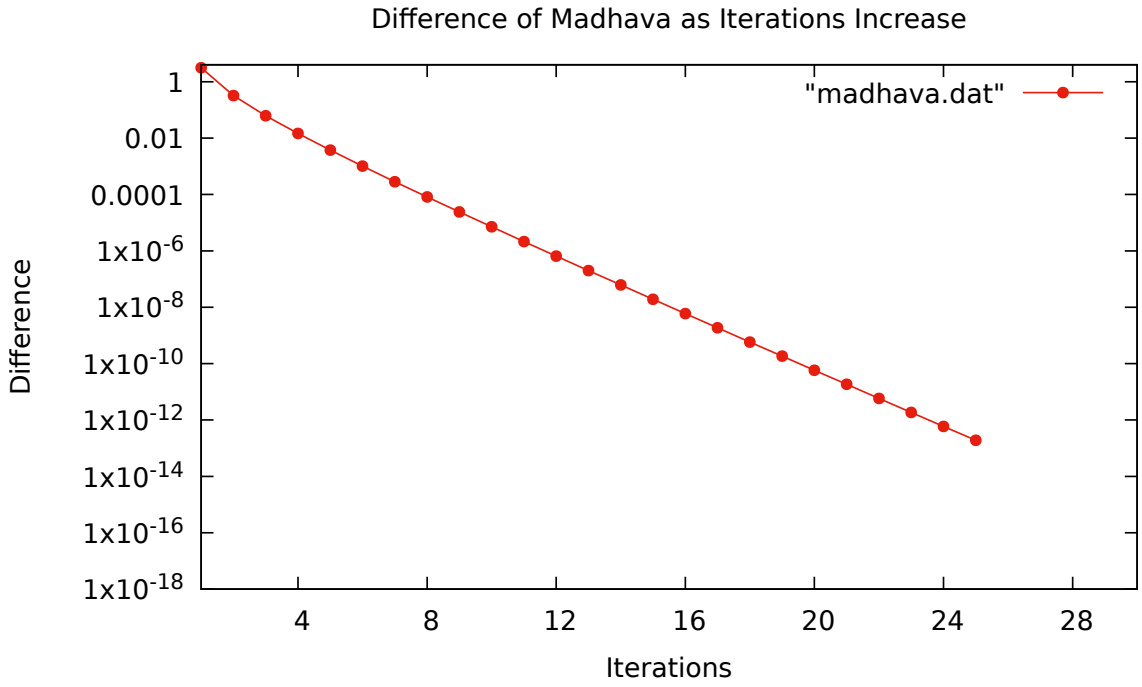
January 29, 2023

## 1 e.c



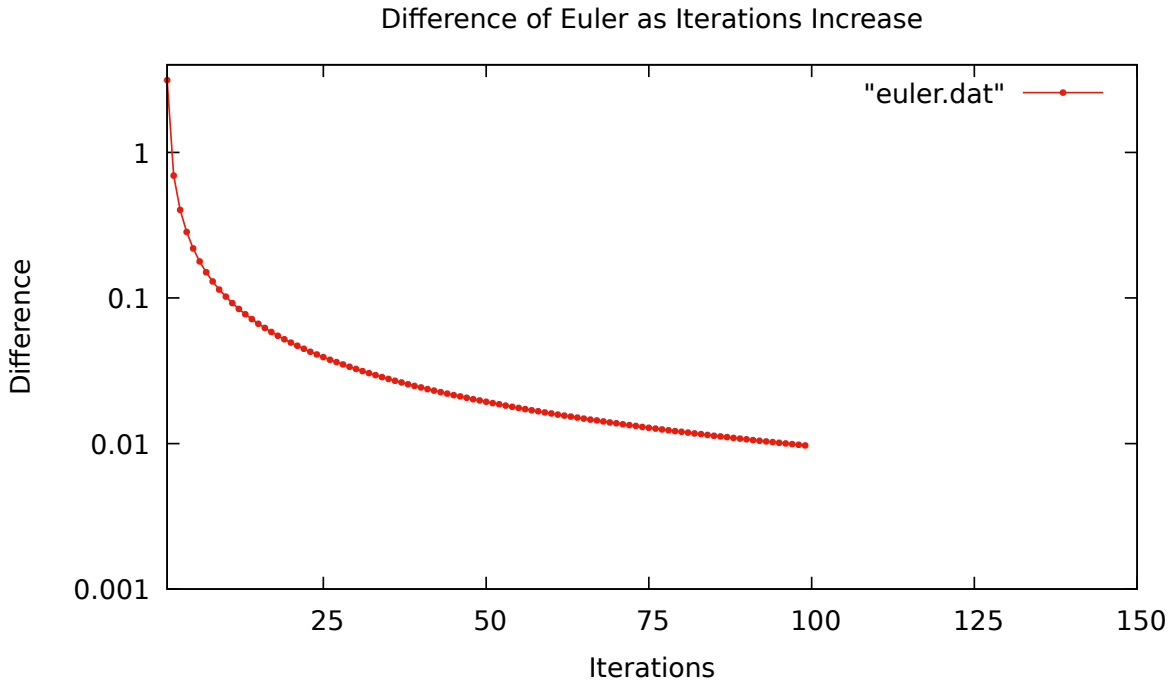

The graph above displays the difference between the values reported by my function “e” in e.c and the constant of euler’s number e in the math library which is M_E in code. For the first term/iteration, the difference is 1.718281828459045 as you can see from where the graph starts at. As the number of iterations increases, the difference between my function’s computed value and the math library’s value of e decreases. We can see this in the graph as the line curves down approaching EPSILON. This is because for each iteration, a smaller term is added to the sum which makes the difference smaller.

## 2 madhava.c



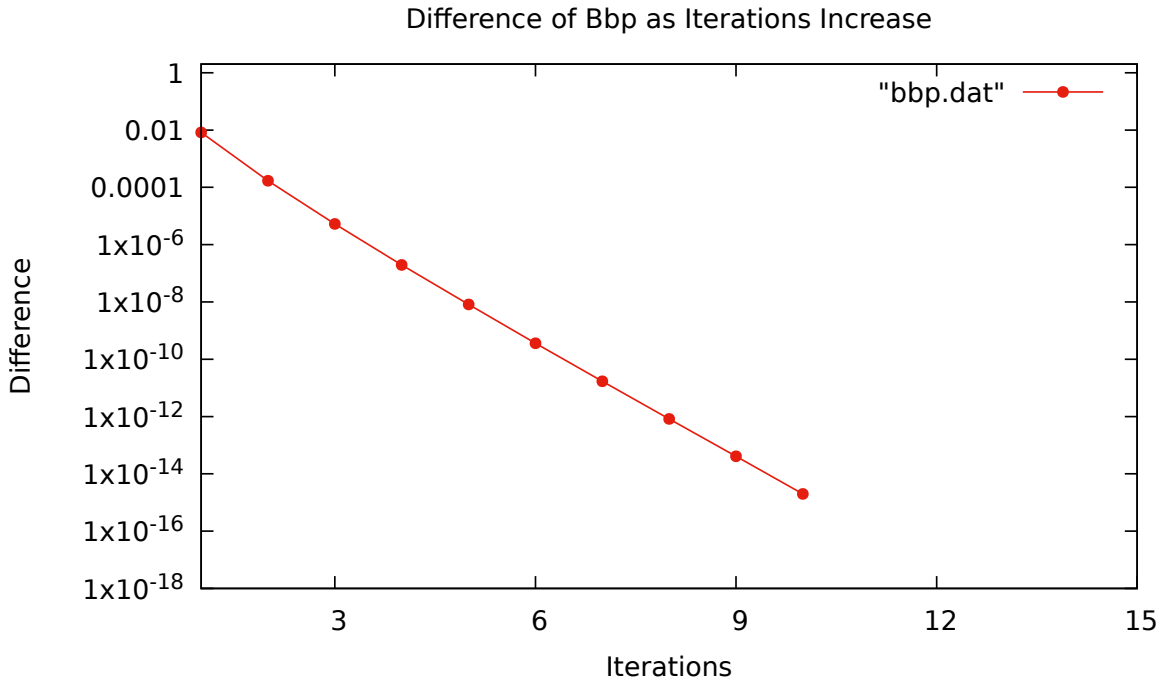

The graph above displays the difference between the values reported by my function "pi_madhava" in madhava.c and the constant of pi in the math library which is M_PI in code. For the first term/iteration, the difference is 3.141592653589793 as you can see from where the graph starts at. As the number of iterations increases, the difference between my function's computed value and the math library's value of pi decreases. We can see this in the graph as the line curves down approaching EPSILON. This is because for each iteration, a smaller term is added to the sum which makes the difference smaller. This is the same concept as e.c which I explained above.

# 3 euler.c



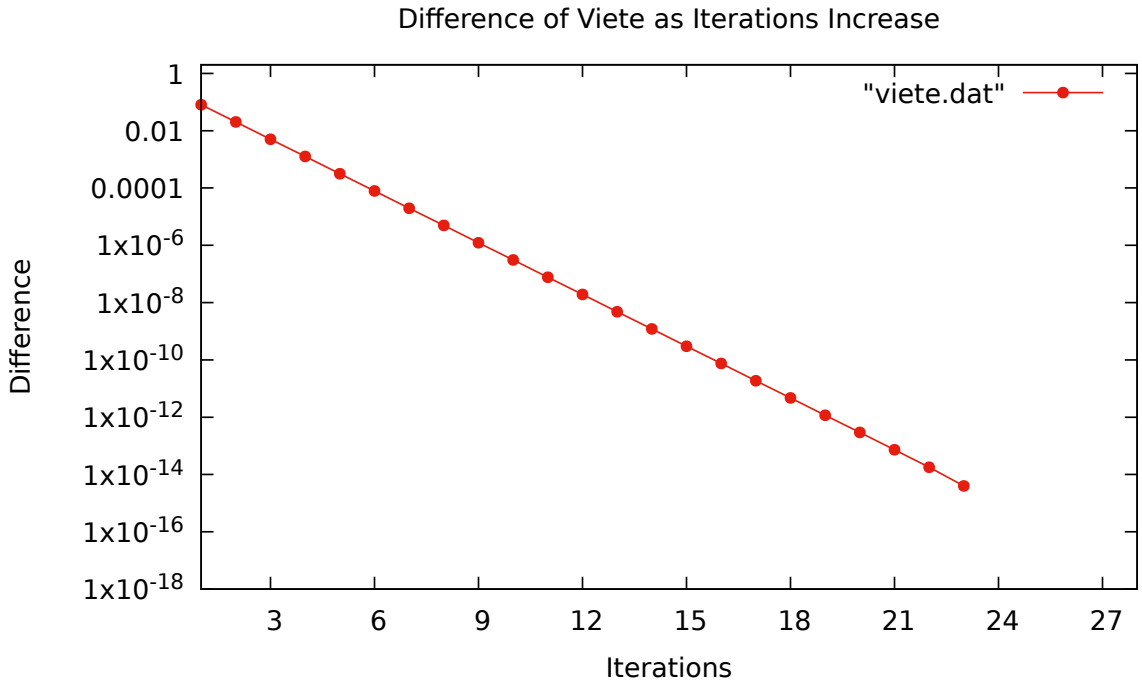

The graph above displays the difference between the values reported by my function “pi_euler” in euler.c and the constant of pi in the math library which is M_PI in code. For the first term/iteration, the difference is 3.141592653589786 as you can see from where the graph starts at. As the number of iterations increases, the difference between my function’s computed value and the math library’s value of pi decreases. This is because for each iteration, a smaller term is added to the sum which makes the difference smaller. My function, “euler.c”, took 10 million iterations to approximate the value of pi when the last term added was less than EPSILON. Because 10 million points was too much to add to a data file and plot, I decided to only plot 100 iterations. Even with less iterations, the concept of the difference decreasing as the amount of iterations increases remains the same.

# 4 bbp.c



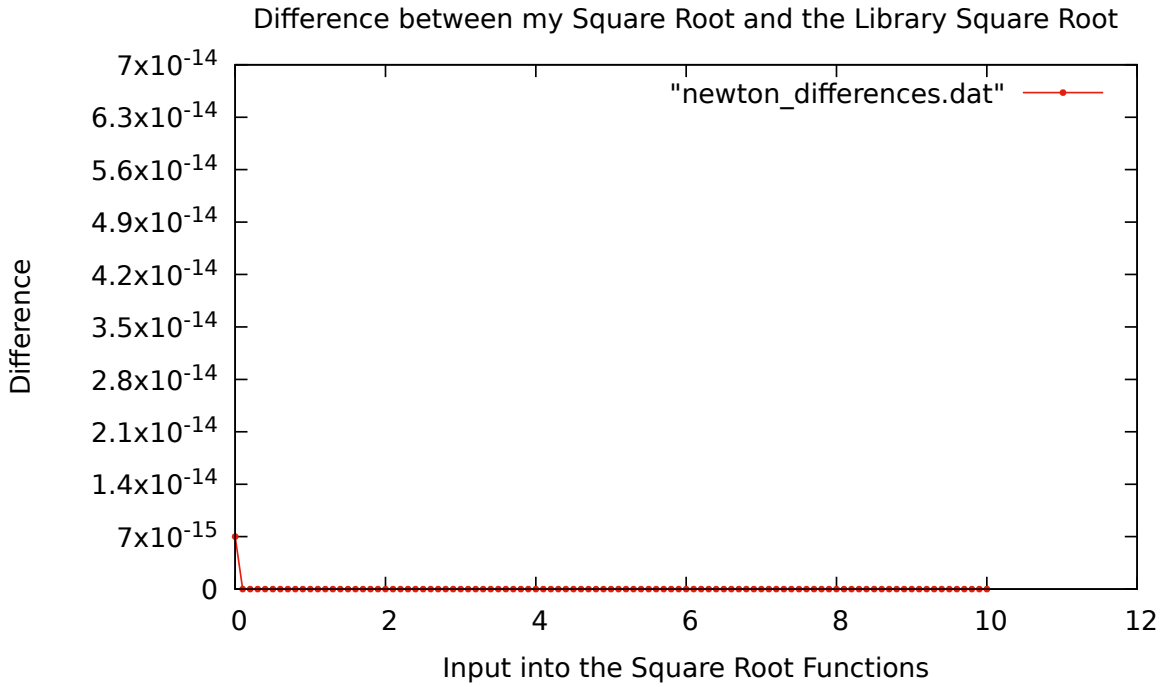

The graph above displays the difference between the values reported by my function “pi_bbp” in bbp.c and the constant of pi in the math library which is M_PI in code. For the first term/iteration, the difference is 0.008259320256460 as you can see from where the graph starts at. As the number of iterations increases, the difference between my function’s computed value and the math library’s value of pi decreases. We can see this in the graph as the line curves down approaching EPSILON. This is because for each iteration, a smaller term is added to the sum which makes the difference smaller. This concept is the same as the other functions I have explained above.

# 5 viete.c

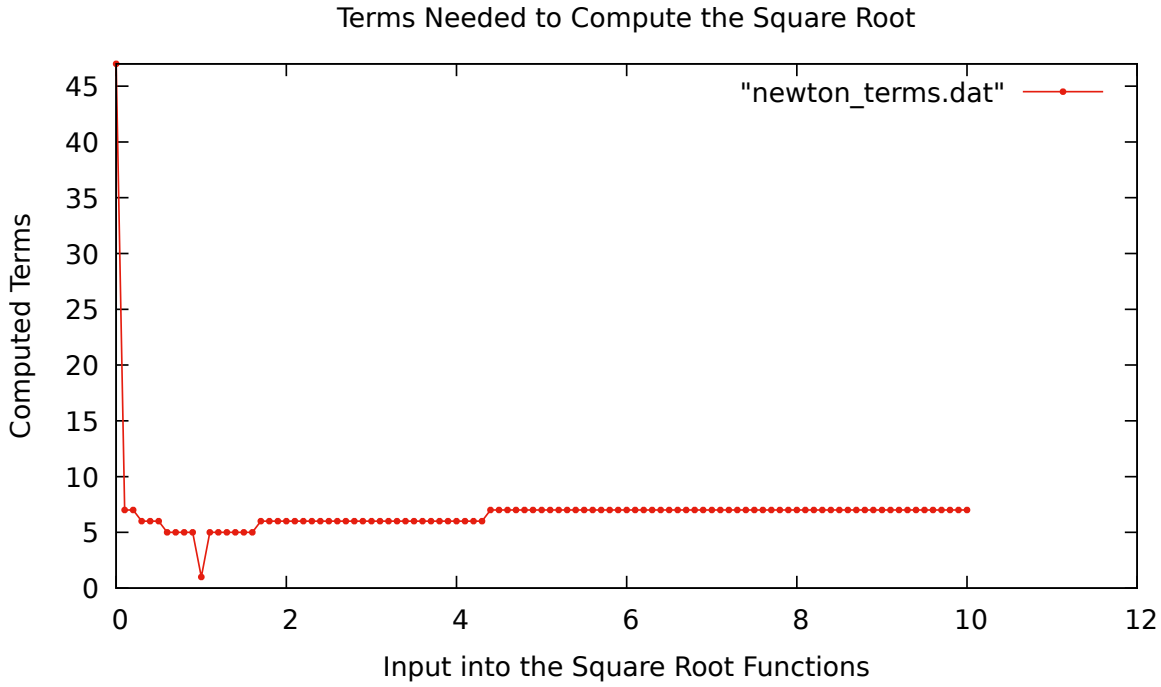



The graph above displays the difference between the values reported by my function “pi_viete” in viete.c and the constant of pi in the math library which is M_PI in code. For the first term/iteration, the difference is 0.080125194669074 as you can see from where the graph starts at. As the number of iterations increases, the difference between my function’s computed value and the math library’s value of pi decreases. We can see this in the graph as the line curves down approaching EPSILON. This is because for each iteration, a smaller term is added to the sum which makes the difference smaller. This concept is the same as the other functions I have explained above.

# 6 newton.c




The graph above displays the difference between the values reported by my function "sqrt_newton" in newton.c and the constant of pi in the math library which is M_PI in code. For the first term/iteration, the difference is 0.000000000000007 as you can see from where the graph starts at. Starting at the second iteration until the last iteration, the difference is 0.000000000000000 as seen from the flat line on the graph. Because this graph doesn't show much because of the flat line, I decided to produce a second graph using "sqrt_newton" which is displayed below.




The graph above displays the number of computed terms required for each input in the square root function from 0 to 10 iterating up by 0.1. As we can see from the graph, my function "sqrt_newton" computed 47 terms for approximating the square root of zero. For calculating the next iteration which was the square root of 0.1, the number of computed terms drastically dropped to 7. For the remaining inputs, the number of computed terms stayed around 5-7 except for only 1 computed terms for calculating the square root of 1. We can see this all from the graph above. This shows that for my function "sqrt_newton" the number of computed terms decreases as the input for my function increases.

# 7 Explanation For Why my Approximation, the Sample Binary, and <math.h>, the C math library are not Identical

Since floating points are not exact and I am doing commands in my functions in different orders than the given binary and the math library, my approximations will not be entirely exact.

# 8 Conclusion

The main takeaway that I had completing this assignment was that we can use summations to approximate the value of euler's number e and the value of pi without using a pow function. I learned that a summation in math was just a simple for-loop in C. I also learned that to replace the pow function, the easiest method was to find the common factor between current and future terms that were being added to the sum in the summation. I never knew that summations

could be used to approximate the value of Euler's number e and the value of pi. It was the first time that I heard of the different formulas used in this assignment. An additional thing that I learned was how to use a git branch, as I needed to utilize this to modify some of my functions to use in a bash script to plot my graphs in this writeup.