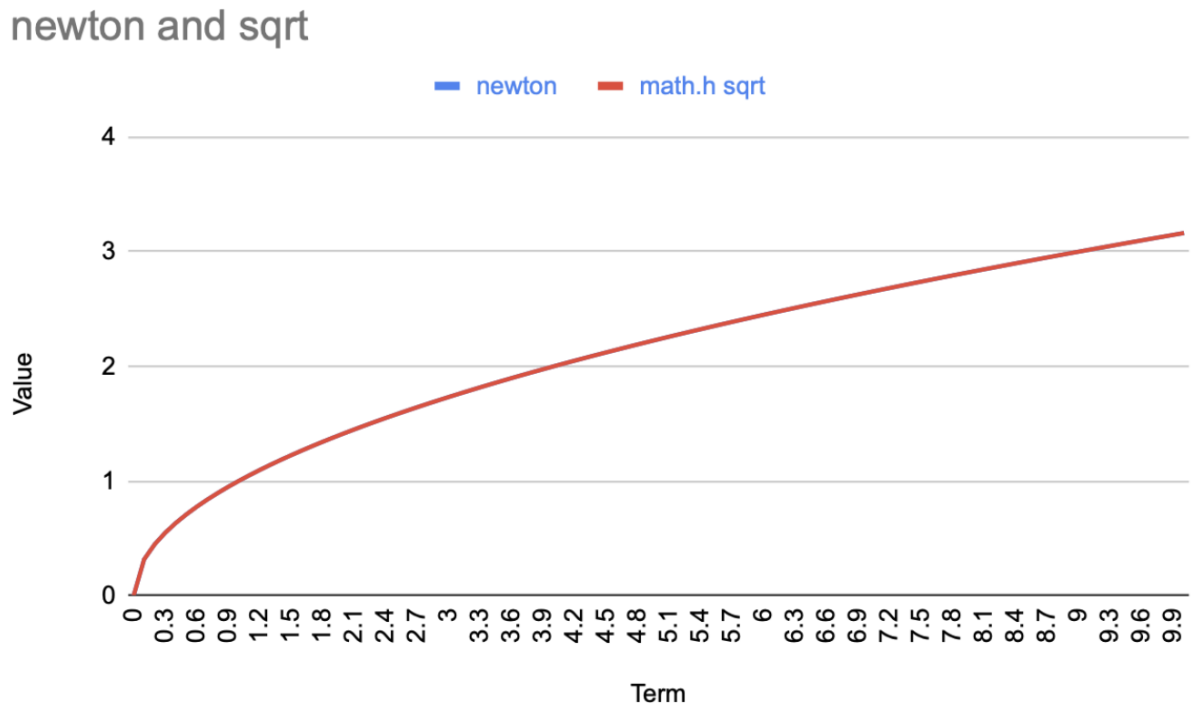
Assignment 2 - A Little Slice of Pi -- *WRITEUP.pdf*

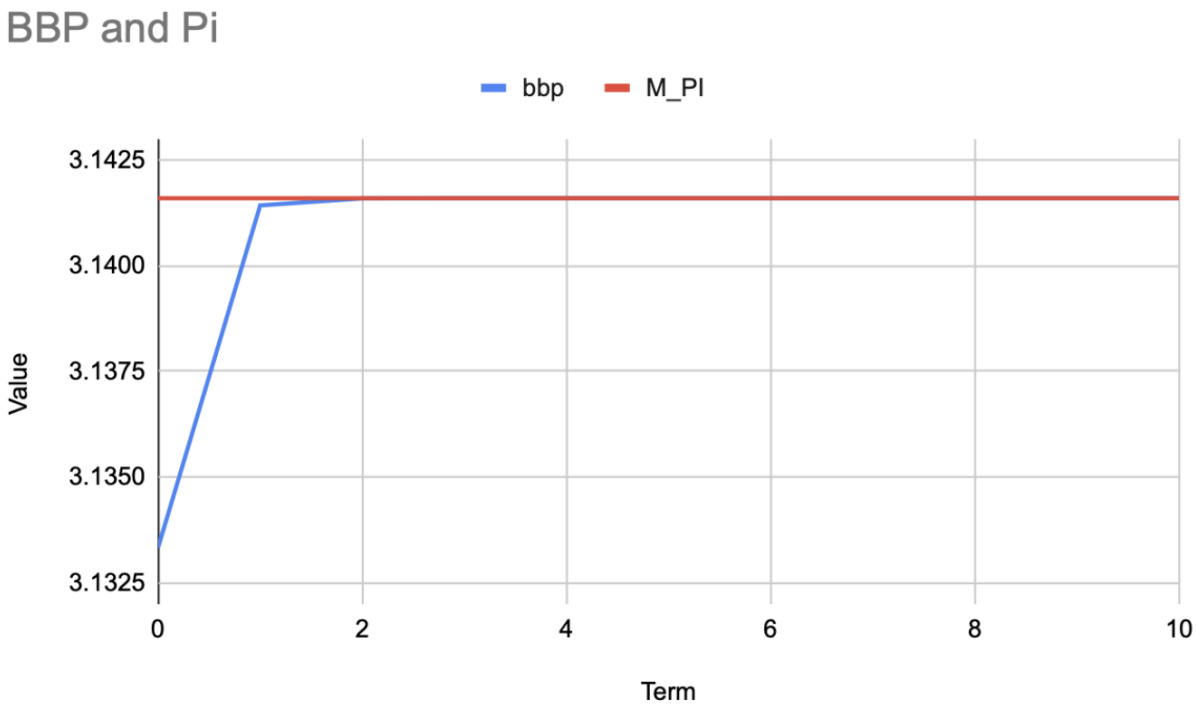
Introduction

This program implements a small number of mathematical functions that mimics <math.h> then using them to compute the constants e and pi. We then compare our outputs to the constants and functions provided by the math library in order to find how long each takes to converge to the value. This assignment allows for better understanding as to how such functions are created and how they work by using simple mathematical operations: addition, subtraction, multiplication, and division.

Graphs and Analysis

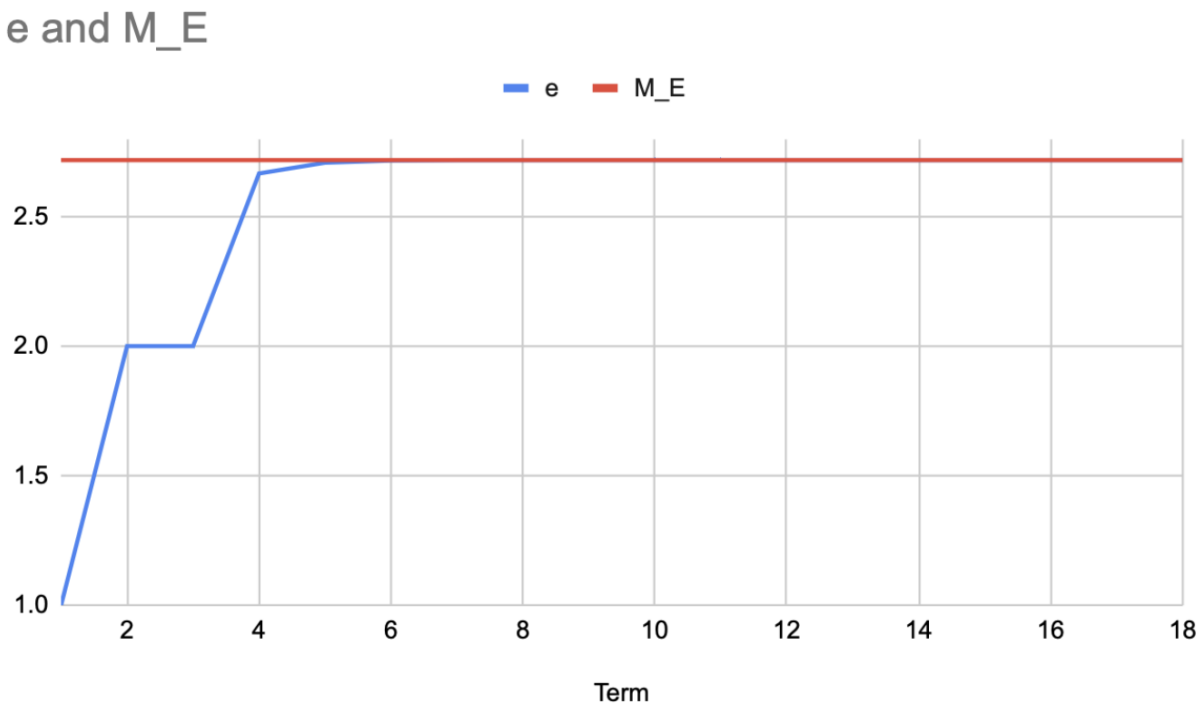


This is a graph of the sqrt_newton() function implemented and the graph of the sqrt() function from the math library. The graph is of each function's square root values from 0 to 10 in increments of 0.1. The function, sqrt_newton(), has virtually no differences in each iteration when compared with the sqrt function from the math library, which can be seen from the same line on the graph. The code for sqrt_newton() was given by Professor Long in the lab document, but in python language which was then directly translated into C.



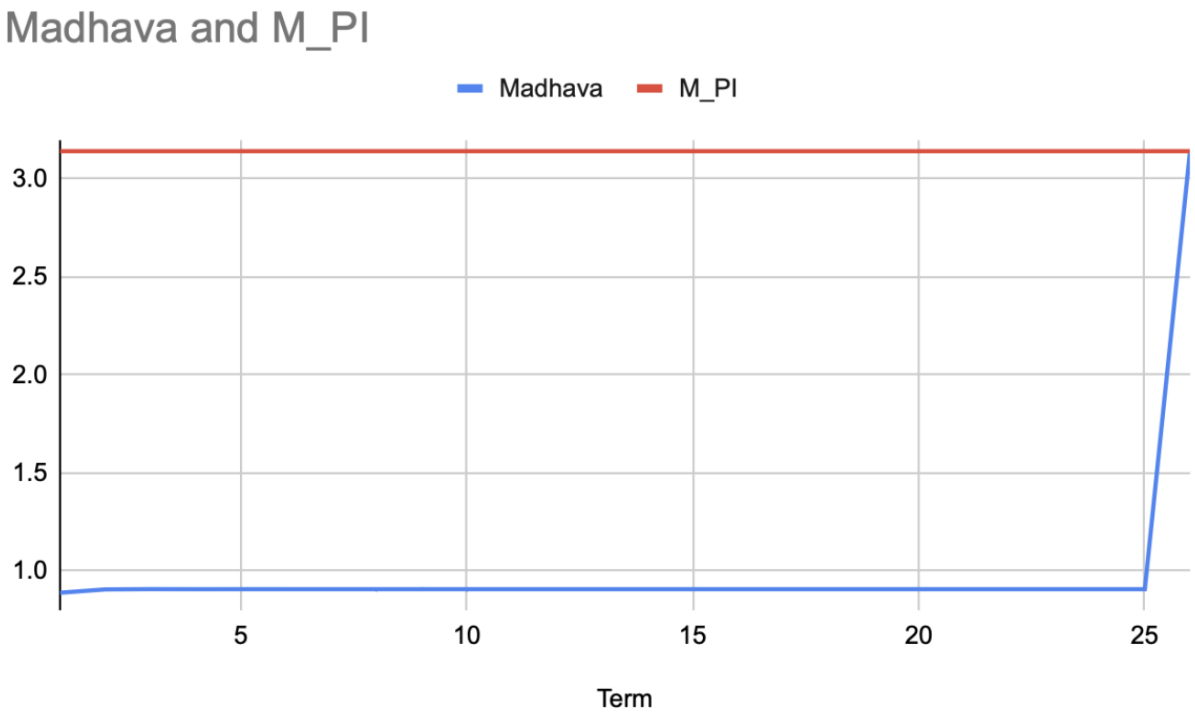
This graph is of the implemented function, pi_bbp(), and the constant M_PI as provided by the math library and shows the comparison of the values of pi_bbp as it approximates pi to that of the constant M_PI. The function, pi_bbp(), is seen to converge with the M_PI (pi from the math.h library) at around 2 terms. This is because in this function, pi's initial value is set to $\frac{47}{15}$ or $3.\overline{13}$, before being iterated upon and adding the summation value from the given formula (adding $\frac{k(120k+151)+47}{16^k(k\,(k\,(k\,(512\,k+1024)+712)+194)+15)}$ as presented by the Bailey-Borwein-Plouffe formula in Horner normal form. Adding this formula to pi in every iteration allows it to converge with the pi given by the math.h library and have similar output.

*Note that M_PI is a constant value which is why its graph is a straight horizontal line, its purpose is to see when the function we wrote (pi_bbp) reaches the constant value.



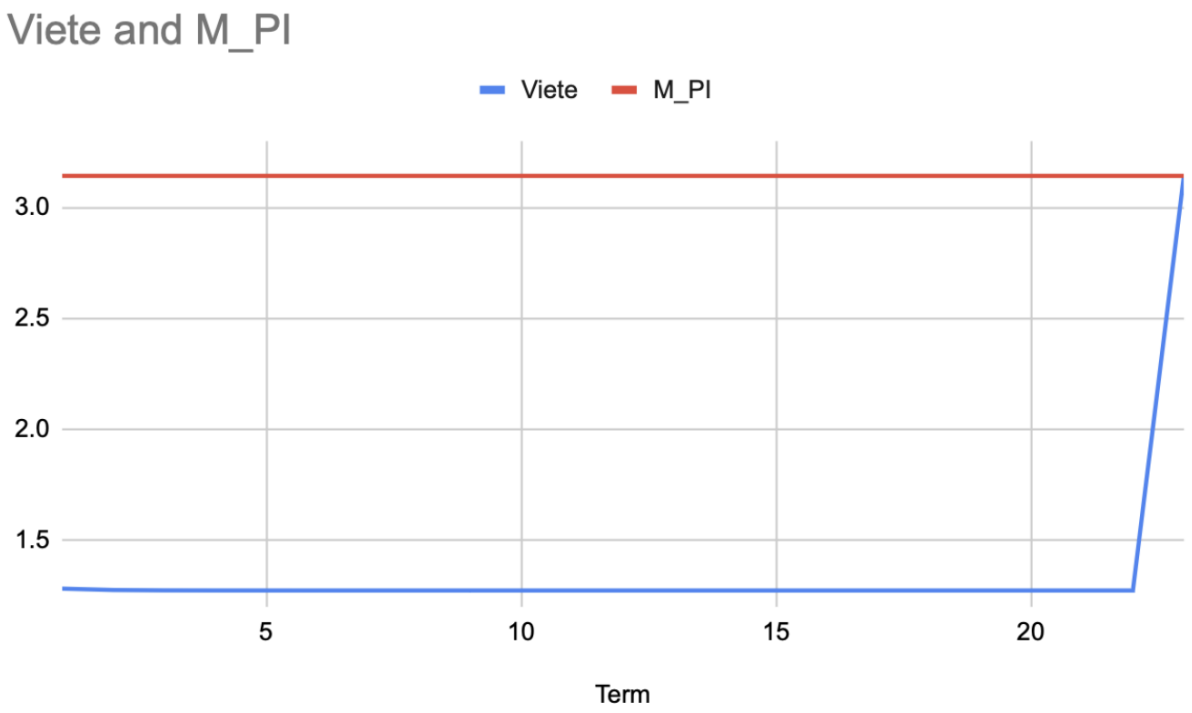
This graph shows the values of the implemented function e() as it approximates the math constant e and compares these values to the math library's M_E. For the e() function in calculating e $(2.718..)$, the code initializes e at 1 before using the given formula to add e by $\frac{1}{k!}$ until the difference between each term is less than EPSILON or $1e^{-14}$. This allows e to converge with the e value from math.h library at around 5 terms, where the difference between the two values becomes too miniscule to see on the graph.

*Note that M_E is a constant value which is why its graph is a straight horizontal line, its purpose is to see when the function we wrote (e) reaches the constant value.



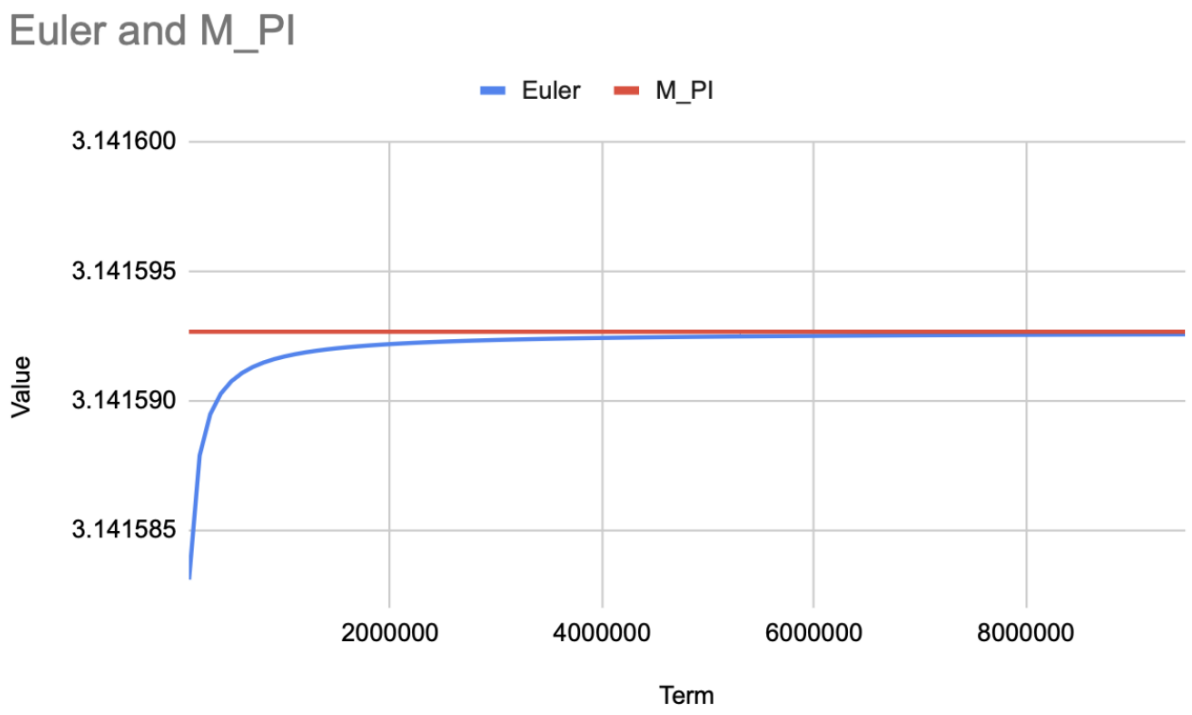
This graph shows the values of the implemented function pi_madhava() as it approximates the math constant pi and compares these values to the math library's M_PI. The values from the madhava function are not seen to converge with the constant pi term until the last iteration. This is due to the formula, where at the last instance $pi = sqrt(12) * pie$ which gives the final value similar to the M_PI constant. Until this point, pi is initialized at 1 before being summed in a while loop by $\frac{1}{-3^k(2k+1)}$. Because this value gets increasingly smaller, there isn't a noticeable difference in its graph until the difference gets smaller than EPSILON - where the loop ends and the result is multiplied by $sqrt(12) * pie$ as mentioned before.

*Note that M_PI is a constant value which is why its graph is a straight horizontal line, its purpose is to see when the function we wrote (pi_madhava) reaches the constant value.



This graph shows the values of the implemented function pi_viete() as it approximates the math constant pi and compares these values to the math library's M_PI. Viete has a similar graph to madhava in that the value does not converge with the constant value until the last term. This is due to the formula and the way Viete is solved, as the the last term of $pi = 2 / (pi / 2)$ as provided by the equation in the lab document. Before this point, pi is initialized to the first term of the formula , a_1, which is $\sqrt{2}$. This allows for the first term to be taken into account before pi is multiplied by $\frac{\sqrt{2+a_n}}{2}$, a small number causing little noticeable difference until the loop (where pi is multiplied) is exited and the final term is calculated.

*Note that M_PI is a constant value which is why its graph is a straight horizontal line, its purpose is to see when the function we wrote (pi_viete) reaches the constant value.



This graph shows the values of the implemented function pi_euler() as it approximates the math constant pi and compares these values to the math library's M_PI. The euler function approximates pi in about 9538440 terms as it converges with the constant pi value slowly. This is because pie is initialized to 1 before being set to $\sqrt{6(\frac{1}{k^2})}$ in each run of the while loop, increasing its value quickly in the beginning but then slowing down as the difference between terms becomes smaller and smaller. There is still a bit of difference seen between M_PI and pi_euler near the end of the graph as the euler function only runs 9538440
times instead of 10000000 as in the example binary. This is likely due to the method of calculating euler, as the loop exits earlier than the example binary's.

*Note that M_PI is a constant value which is why its graph is a straight horizontal line, its purpose is to see when the function we wrote (pi_euler) reaches the constant value.

Conclusion/ Main Learnings

Through this assignment, I learned more about the efficiency of formulas when it comes to approximating a certain term. For example, the euler function approximates pi in almost 10 million iterations while the bbp formula approximates it in about 11 iterations. I also learned more about C data types - formatting these types and to use double to hold the most amount of data possible. I also learned about implementing functions that

converge with those of the C math library, and a better understanding of how these programs are created using mathematical operations.