Noel Ball
CSE13S

mathlib-test.c Writeup Document

mathlib-test.c:

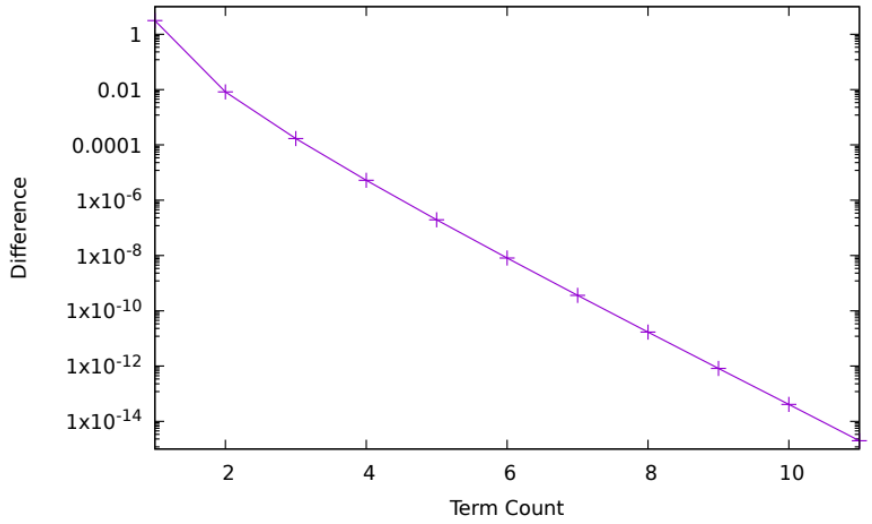
Building the switch-case command line argument with getopt() was straightforward following the documentation. I used the input to set boolean values to true for which files to run and if stats are enabled. If -h was involved, it outputs the synopsis and returns with 1 to end the program. Otherwise if-statements check which files run their functions and terms if needed, then format the outputs to a print statement to display the values.

Makefile:

I followed the unofficial ucsc student guide on Compilation, by Ben Grant, to create my Makefile. With the guide, I learned how to automate the process of assigning the dependent .o files with the same name as their dependencies. I also learned how to automate the clang-format process to all the .c and .h files involved, as well as how to use make all to clean, generate, and format all with one command.

Noel Ball
CSE13S

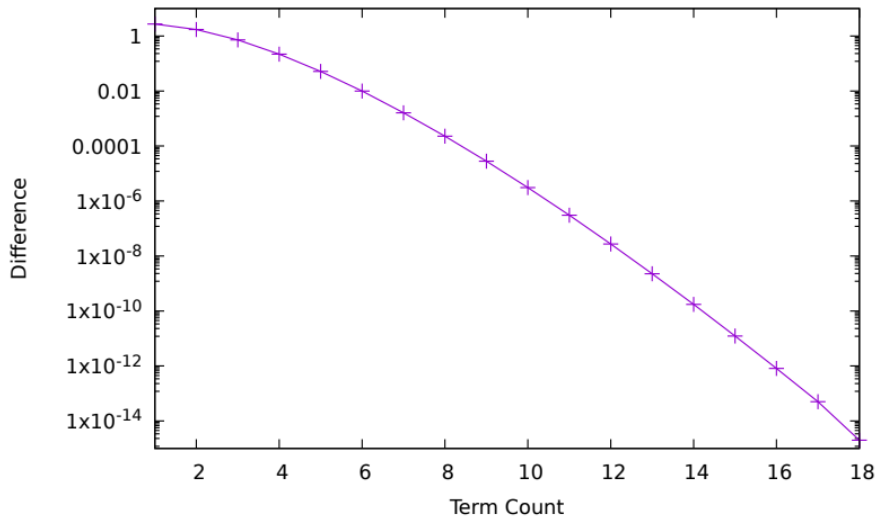
bbp.c:



bbp.error.pdf

Calculating the term of the summation was fairly difficult for me, since I'm not familiar with the amount of sets of parentheses needed in c to get the math to multiply, divide, and add in the correct order. After a few attempts I found the correct syntax. Creating an exponential term with an increasing power just required a multiplication loop.

e.c:

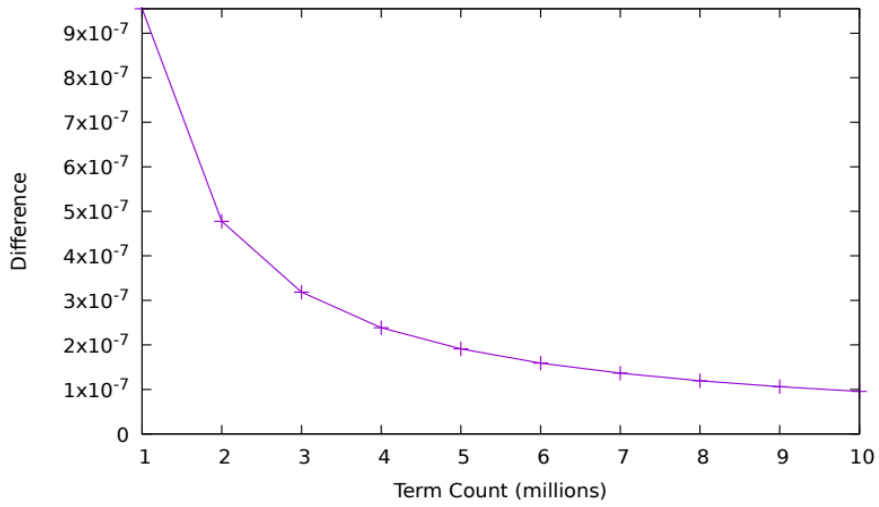


Noel Ball
CSE13S

e.error.pdf

The terms in this summation were straightforward, only having to include a multiplication to loop to act as a factorial for the denominator under 1.

euler.c:



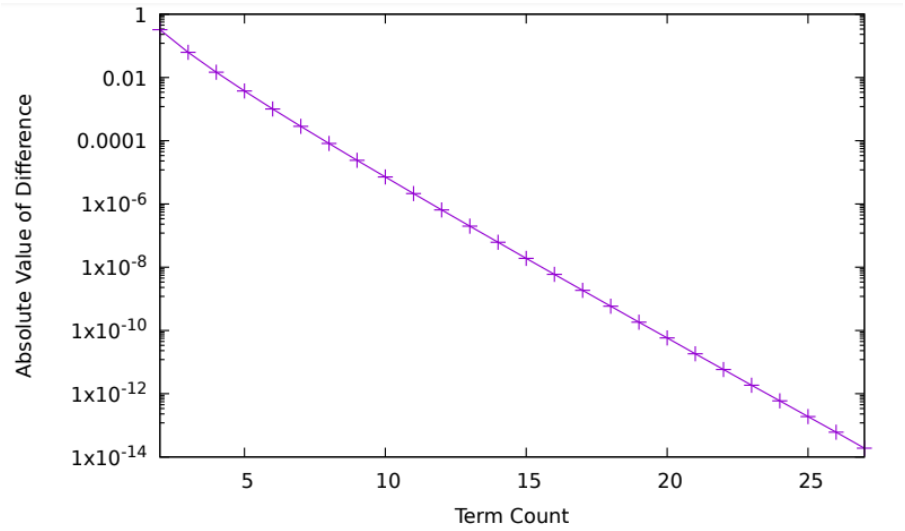
euler.error.pdf

The terms in Euler's sequence were simple as well, setting the denominator under 1 to 2k.

Although at the end of the summation the sum needs to be multiplied by 6 under a square root.

Noel Ball
CSE13S

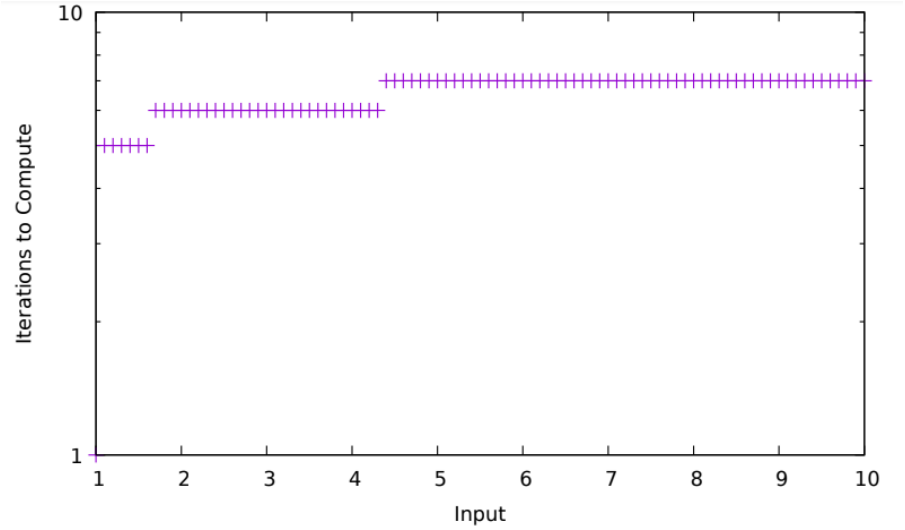
madhava.c:



madhava.error.pdf

My madhava function initially ended the summation just after 2 terms or so, and I eventually found that I needed to check the absolute value of my term against epsilon to correctly adjust when to exit the loop, while still adding the correct term to the total estimation of pi.

newton.c:

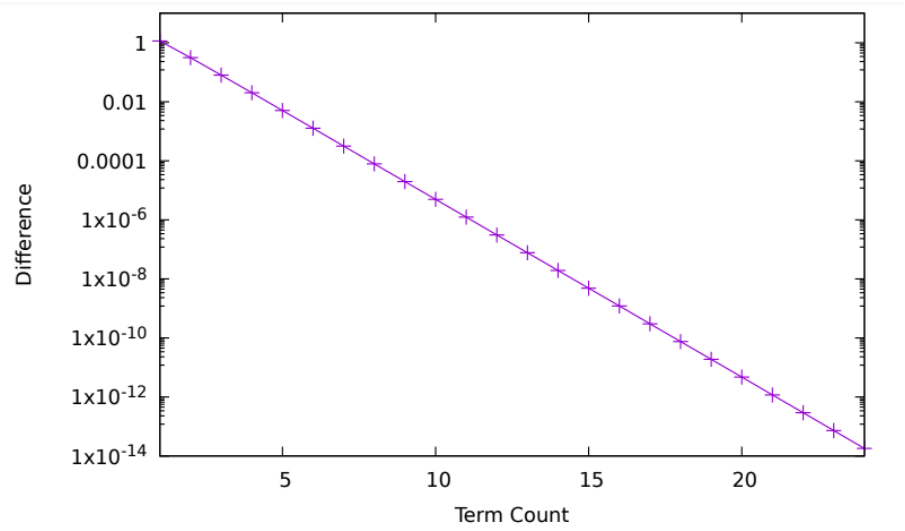


Noel Ball
CSE13S

newton.terms.pdf

For newton.c I chose to represent the iterations taken against the different inputs given that every value essentially had a difference of 0. Following the documentation's python example it was very easy to implement the square root estimation.

viete.c:



viete.error.pdf

For viete's product, my first error was simple in that my previous term was set to 0 like the summations, however that nullified the first term computed and ended the loop. After correctly setting the initial term to 1, I found my next error. Initially I tried to compute too much of the term at once, where I ended up separating the computation of a_k based on the previous a_k, and built the term around that.