Alejandra Sicairos
CSE13s
10/10/2021

Assignment 2 - A Little Slice of π

Implementation of a Small Numerical Library

Introduction:

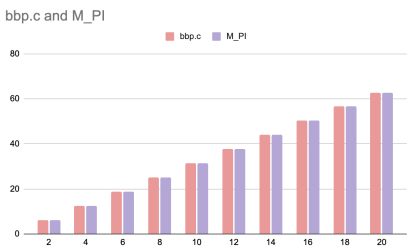
In this program I implemented a series of different formulas to approximate π or e. The formulas were: the Bailey-Borwein-Plouffe, Taylor Series with Euler's number e, Euler's solution, the Madhava series, Newton-Raphson method, and Viète's formula. All these formulas were used to compare the nature of error when self implementation. This error is shown by comparing the output with the <math.h> library in C. The formulas were shown in the Assignment document. The difficulty was properly implementing the formulas without the creation of a factorial function. I soon found that it was not difficult to do the formulas without factorials with minimal error. Most of the formulas came to me at ease for the exception of one. The Bailey-Borwein-Plouffe formula gave me the most trouble. There are multiple reasons as to why. First off, the amount of constants in the Horner normal form confused me when I was making my numerator and denominator calculations. I was adding the parenthesis in the wrong correct spots. Once that was fixed I had the issue that my bbp.c was not entering the while loop. This debugging took me the longest, just to come to the conclusion that it is due to the fact that when writing in my integers I wrote them as integers rather than floating point numbers. Soon after I realized that my output was 0.9878 different than that of <math.h>, this was my own calculation error, when hard-coding my first term of Bailey-Borwein-Plouffe's formula I still multiplied by $\frac{1}{16}$ forgetting that 16^0 is 1, therefore I would be multiplying by 1 rather than $\frac{1}{16}$ and that would properly set up the first term for the formula to properly approximate π.

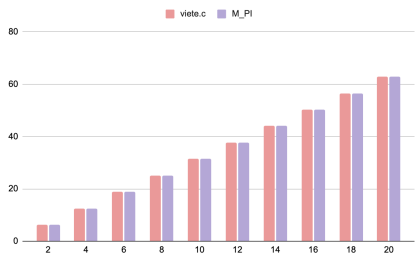
Materials and Methods:

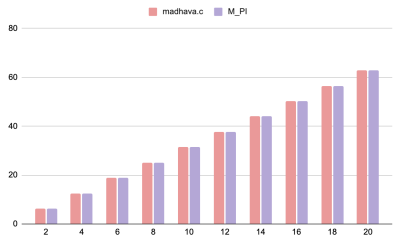
I wrote this program on an Intel processor MacBook. I used my Ubuntu virtual machine to format all my code properly and to assure the results would be cross compatible with other softwares and processors. I did write the code in my terminal using Vim as the text editor. Other materials would be the newton square root function provided to us in the lecture by Dr. Long. For the graphs shown below I used Google Spreadsheets, I unfortunately could not get GNUplot to work and did not understand how to properly use it.

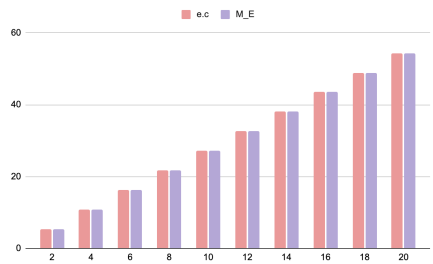
Results:

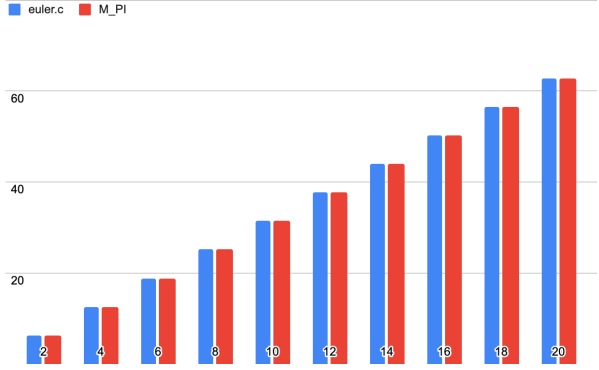
All results are being provided by a M1 processor Mac.











Alejandra Sicairos
CSE13s
10/10/2021

Discussion:

This first graph is comparing my π output using the Bailey-Borwein-Plouffe formula compared to the M_PI the π constant of the math.h library. As you can see the difference is very minimal to the point that it cannot be seen on the graph which is the case for all the graphs with the exception of one. The difference between M_PI and bbp.c as shown by mathlib-test is 0.000000000000000, which is accurately shown in the graph. I will now mention that all the graphs that had no difference between the math library and mathlib-test are the ones that are all the same color and size.

None of the graphs had anything that was a big difference, nor did mathlib-test indicate that there was a difference between the outputs. I will now show the graph of pi_euler. This is the .c program that used Euler's solution with the taylor series to approximate π. This output on mathlib-test gives me a difference of 0.000000095981660. The difference is not super significant to the point where I believe it changes much of the output. It is extremely slight, which is the reason why this chart is so big compared to the others. Therefore I would like to say that Eulers is a fair approximation of π but in terms of exactness any of the other formulas would be a better approximation than this one.

Concluding Remarks:

This assignment was so far the most complex assignment that I have done in a CSE class, and I am actually appreciative of it. I was able to learn so much from it, such as how to take the factorial of a number without having to make a dedicated function or a for loop. As well it demonstrated what is really going on when M_PI is called from the math.h library. It made me think and experiment and overall a great learning experience.