WRITEUP ASGN2

Cruzid: acristea

January 2023 - CSE13S

1 Building Numerical Library

- For every mathematical function, I used asgn2.pdf from our resources git to understand them and then write them in code. I mainly focused on digesting and breaking down the taylor polynomial approximations of each and worked to translating them into coding logic and then into C.

Finding e Using the formula from asgn2.pdf, which was the taylor series approximation of e, I created a for loop and slowly increased by each term towards e. I utilized the iteration variable to replicate the factorial in the equation. As a result, I found that there is 0.000000000000001 difference between my approximation and the Math.h approximation.

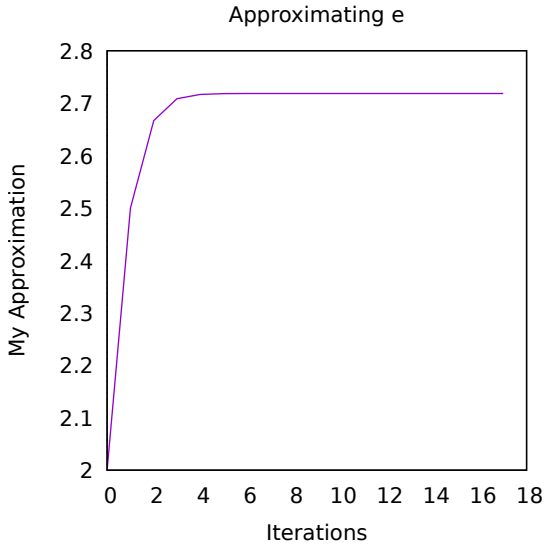
Finding pi - There are multiple different formulas to find pi (as given in the assignment document). All of which I focused on translating the taylor series approximations into C.

- Euler's solution showed that the solution is (pi squared)/6, but his method also requires us to calculate the square root, in this case, an unknown (until we calculate it) number. Using a loop and after exiting multiplying my answer by 6 and then square rooting that I got pi. The difference was 0.000000095493891 from the approximation for pi by Math.h.
- The BBP formula for pi was discovered in 1995 with the formula that they discovered being remarkably simple. Without being able to use pow(), I multiplied the previous term or value by its new value. There was no difference at all, perhaps due to the formula being more simple. However, I would assume that there could be a possibility of the difference being so extremely small that it doesn't appear on the output. In any case, this is extremely successful regardless.
- Viète's formula is an infinite product of nested radicals that can be used for calculations of pi, though it should be noted that methods found before this specific formula are known to produce greater accuracy. Viète's formula gave you pi/2. You then need to do the calculation 2/pi/2 to get pi. The was no difference at all- or again, just too insignificant to even care.

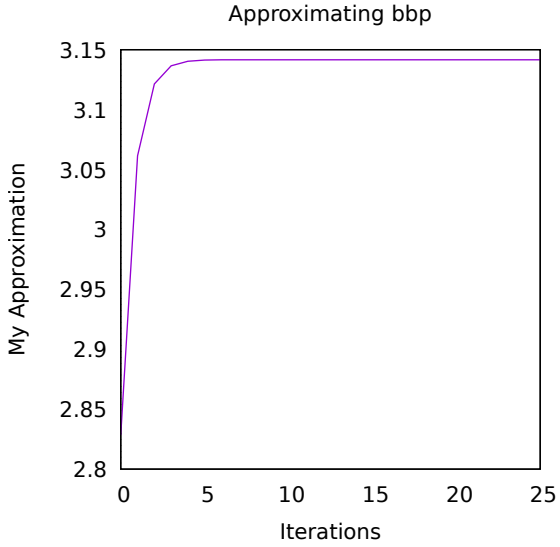
- The Madhava series, similar to BBP uses pow(). I navigated around this issue in a similar manner. The difference was 0.000000000000007 when comparing my approximation to the pi approximation of Math.h.

Square Root Function - Due to some of our formula's using square root and us not being able to use sqrt() we had to make our own square root function. Given a template in python from asgn2.pdf we could create our own- we were tasked to translate the code of the Newton-Raphson method.

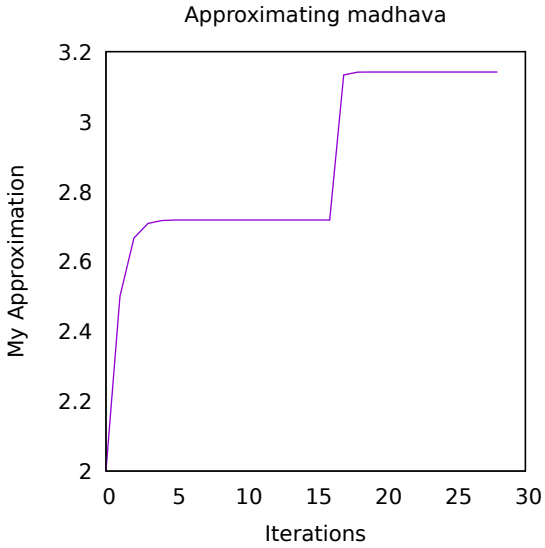
2 Comparison of Functions



This plot shows our formula slowly approaching the value of e. We can observe the accuracy of the function as it reaches the final value of approximately 2.718... which is the actual e value itself. It is interesting to see how the curve represents an almost logarithmic curve and how the slope exponentially increases and slows down toward the e-value.

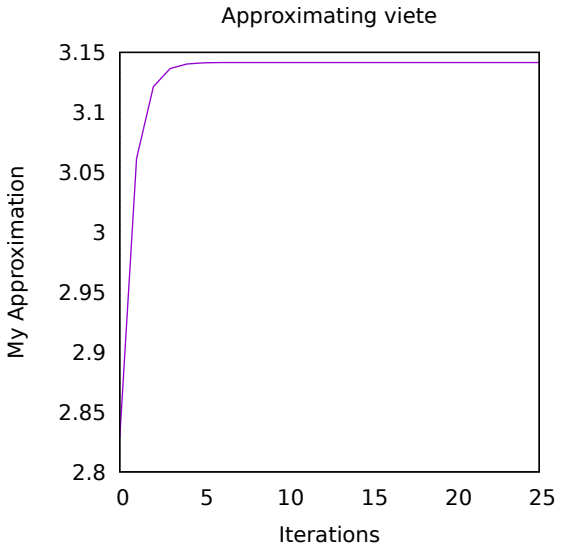


This plot shows the BBP formula approaching the value of PI. Similar shape as the graph of approximating e. I would say that you can easily observe how slowly accurate the values in the bbp function is getting as the amount of iterations increase. Because of the extremely small difference between my implementation and the C math library function, I am not surprised to see this graph output.

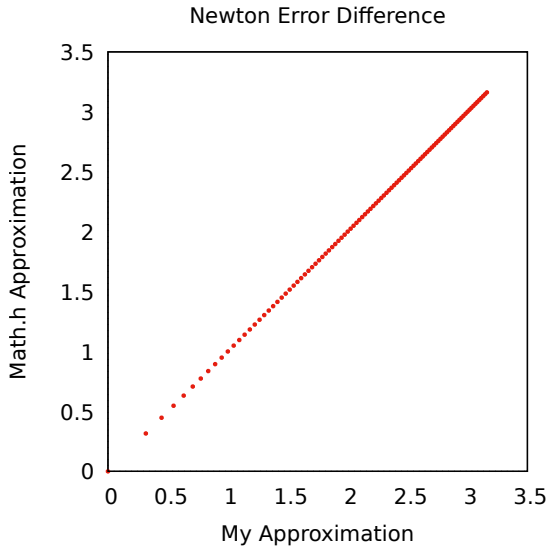


This plot shows the Madhava series approaching the value of PI. There's a sudden spike which I am not entirely sure about. I would like to say that as I was making my bash script to generate these graphs, there were many strange occur-

rences in which the data files that I was pulling were replicating each other, so I ended up generating exactly identical graphs. This, being the only different-looking graph stood out to me. I am not sure what to think of it, but you can still see that the Madhava implemented function still approaches the actual pi value.



This plot shows our viete. Similar to the bbp and e approximation graphs, this graph shows that same curve where the slope is dramatically increasing in the first couple of iterations and slows down as the number of iterations increase.



This scatter plot shows the accuracy of our newton square root formula when

compared to math.h. Because it is linearly scattered and because there are no outliers, you can tell there is no difference between my approximation and math.h's approximation.

3 Conclusion

In conclusion, I learned how to create a test file that referenced multiple other files. I learned how to code math functions and used loops in c to return correct outputs without needing to rely on math.h. I also learned how to make a Makefile to clean and compile my files.