Ali Salehi ASGN2 WRITEUP.PDF

Overall, I feel content with my attempt to mimic the math.h C library for the functions we were given. It took several iterations, and small changes to make my values more accurate, but in the end I felt I received a close approximation.

General Findings

The first finding I had that drastically changed the accuracy of my approximations was that the data type of the variables made a difference in the overall accuracy. Originally, my autopilot implementation was to use floats for all of values since we needed to access decimal places. Then, after seeing the results, which were somewhat accurate, I began tinkering and quickly noticed that changing the data types to doubles gave me a boost in accuracy. This is rather intuitive given that doubles are twice as precise as floats, and our functions go far into the decimal places in long, intensive calculations.

Secondly, I chose to stray away from using pow() from math.h and created a variable for x^2 for my Horner normal form Padé approximations, which I believe could have increased my accuracy and ease of computing.

Sin() and Cos()

With these two functions, I was able to achieve a relatively accurate approximation from only implementing the Horner normal form Padé approximations, but after reading about the benefits of normalization on Piazza I decided to give it a try. First, I remembered that our approximations are most accurate when closest to the center of our functions, which in this case is 0. Secondly, I referenced the graph for Sin and Cos and went half a period length to the right of the center, and half a period length to the left to achieve the most accurate full period of the function, which is [-pi, pi]. From there, I shifted all values outside the most optimal period yet still in the domain of the function into the most optimal period by either adding or subtracting 2pi, or shifting it to the center. This normalization helped my approximations incredibly, making many of the values identical within 8-10 decimal places. I assume the math.h library performs a similar operation to maximize their own algorithm and use values closest to the center.

Tan()

In the domain of Tan given, [-pi/3, pi/3), there is only one period of the function, therefore it was not necessary to perform the same normalization that I did for Sin and Cos. Through using Horner normal form of the Padé approximation, I was able to to attain a difference of all 0's between the math.h tan() function and my own implementation.

Exp()

_____

This function used a Taylor approximation rather than Padé, and I adopted the principal from the lab document and the lecture slides where you keep track of the previous value of the Taylor series, and multiply it by (x/n) every iteration until the “previous term” converges or reaches a value smaller than epsilon. I think the epsilon value given was precise enough to allow this function to exactly mimic the math.h implementation.