Homework 3 Part 1: Trig Functions

Nina Martinez Diers*
Bryn Mawr College Department of Physics
(Dated: March 7, 2024)

Algorithms were written to calculate the sine, cosine, and tangent of a value and the first 5 periods of these functions and the associated error in the values were graphed.

1. INTRODUCTION

Trigonometric functions such as sine and cosine are defined by infinite Taylor series.

$$sin(x) = \lim_{N\to\infty} \sum_{i=0}^{N} \frac{(-1)^i x^{2i+1}}{(2i+1)!} \tag{1}$$

For each evaluation of sin(x), the a finite number of terms in Taylor series in Eq. 1 are summed to achieve a level of accuracy. In this algorithm, I decided to determine the accuracy of the function by the fractional error of the function with itself after adding an additional Taylor series term. The fractional error is calculated using the difference between the value of the function before and after adding an additional term to the taylor series.

$$\sigma = \frac{(-1)^N x^{2N+1}}{sin(x)} \tag{2}$$

The numerator of Eq. 2 is the N^{th} term in the Taylor series, and the denominator is the evaluation of $sin(x)$ with N terms in the Taylor sum. If the value of the sine function does not change by more than a preset value after adding a subsequent term, the value of function is considered acceptable and additional Taylor series terms are not added to the calculation.

This method was also applied to an algorithm for a cosine function. The sine and cosine functions were then used in an algorithm to calculate the tangent function.

2. EXPERIMENT

2.1. Algorithm for Sine

In order to perform the Taylor series, first it was necessary to write an algorithm to calculate factorials. This is done by taking a number, n, and using a while loop to iterate through multiplying by n and subtracting one from n until $n = 0$, when it returns the value.

Because our goal was to build a sine function that is accurate to a preset fractional error, I wanted to use a while loop to calculate the taylor series out to a sufficient number of terms to meet the fractional error criteria.

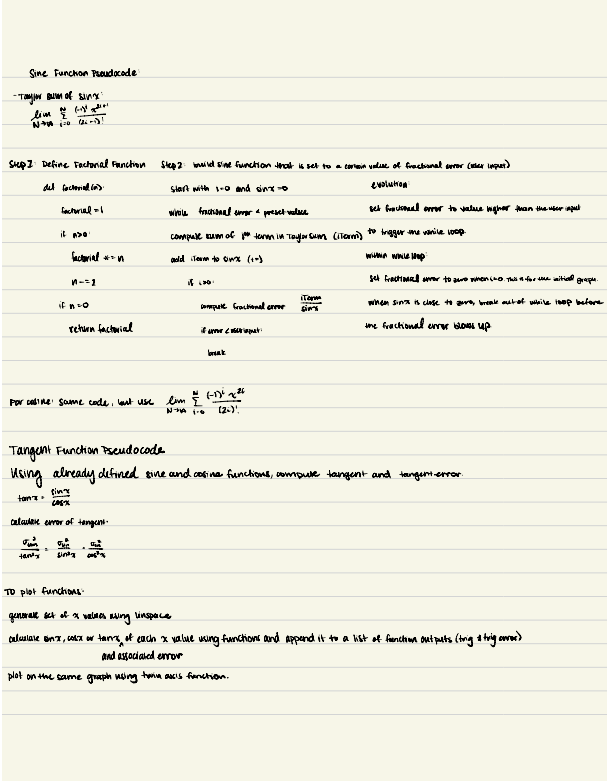

FIG. 1: Pseudocode used to generate algorithms for Sine, Cosine, and Tangent functions.

When the sine function is called, it prompts the user to ask for the desired fractional error and converts the number to a float.

When the while loop is called for the first time, $i = 0$. Every time the while loop runs, the ith term in the Taylor series is calculated, summed to the value of sine, and i increases by one.

Using the current and previous terms in the Taylor series, the fractional error between the terms is calculated. If it is higher than the desired error, then the while loop is called again. If the fractional error is less than or equal to the desired error, the loop is broken and the function returns the value of sine and the fractional error. By evaluating this function over 1000 points between $[0, 10\pi]$, $sin(x)$ and the associated fractional error can be visualized as a function of x.

*Electronic address: nmartinezd@brynmawr.edu; URL: Optionalhomepage

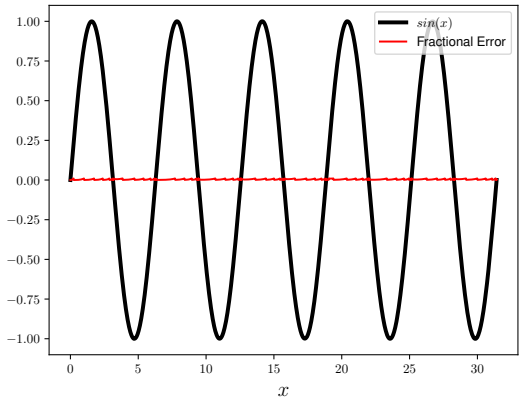

FIG. 2: Five periods of a sine wave. The fractional error produced for these data stays below 0.01, consistent with the set user input value.

2.2. Algorithm for Cosine

The algorithm for cosine is based on the previous algorithm for sine, except it implements the Taylor series for cosine, appropriately.

$$cos(x) = \lim_{N\to\infty} \sum_{i=0}^{N} \frac{(-1)^i x^{2i}}{(2i)!} \tag{3}$$

As with sine, this algorithm was used to produce data to generate a graph of $cos(x)$ with the fractional error.

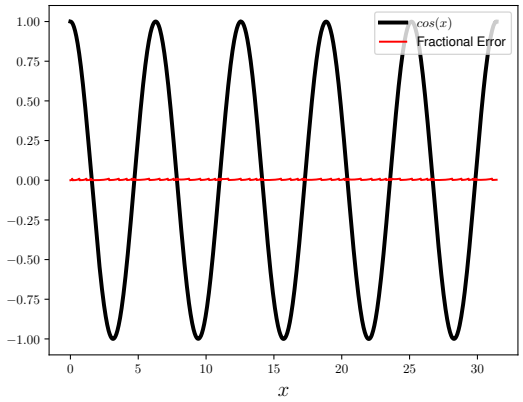

FIG. 3: Five periods of a cosine wave. The fractional error produced for these data stays below 0.01, consistent with the set user input value.

2.3. Algorithm for Tangent

The algorithm for tangent used the algorithms for sine and cosine to evaluate the tangent and calculate the associated error. For each value of x, the sine and cosine functions are evaluated for x and the fractional error is calculated. After these values are retrieved by the tangent function, the value of tangent is evaluated according to Eq. 4.

$$tan(x) = \frac{sin(x)}{cos(x)} \tag{4}$$

The error of this value is found by calculating the deviation of the fractional error of the evaluation of the sine and cosine for each x.[1]

$$\sigma_{tan} = \sqrt{tan^2(x)\left[\frac{\sigma_{sin}^2}{sin^2(x)} + \frac{\sigma_{cos}^2}{cos^2(x)}\right]} \tag{5}$$

As with the sine and cosine functions, this data is used to graph 5 periods of the tangent function.

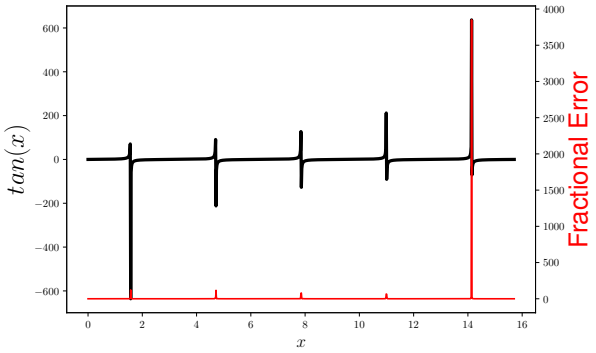

FIG. 4: Five periods of a tangent wave. The desired fractional error input into the function is 0.01. The fractional error produced for these data spikes around multiples of π.

3. RESULTS

To check how similarly my sine function is evaluating other These differences are so small that they are less than the desired accuracy input into my function. However, it is possible that the my function could give more accurate sine value than the numpy sine upon request. The differences in the two functions get more pronounced when x is larger, which could be due to this characteristic of the Taylor sum. When evaluating at high values of x, the Taylor series needs to be calculated out to more terms in order to achieve the same level of accuracy for the sine function. Because my sine function accounts for that in its error evaluation, this problem should be averted. Because the exact algorithm for the numpy sine is unknown, I can't be sure that this is the cause of the more pronounced differences.

Something interesting that happened when I was testing my sine function was that I initially noticed big jumps

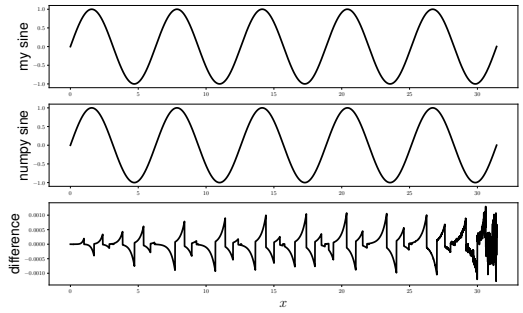

FIG. 5: My sine function and numpy sine. The difference between the evaluations of sine by numpy and my function is negligible. Fractional error input into my sine function is 0.01.

in the error around where $sin(x) = 0$. However, when I increased the number of periods I was plotting from one to 5, the jumps in my fractional error disappeared. This could be due to different sets of x values being evaluated resulting from the different range and spacing of the x values.

The high level of error for the tangent function at multiples of pi is due to either the sine or the cosine function being equal to zero at that point. Because the error calculation for tangent requires dividing by the other two trigonometric functions, when either of those functions equals zero, fundamentally the value of the error for tangent must get infinitely large.

4. CONCLUSIONS

The algorithms for trigonometric functions were successful in producing sine, cosine and tangent functions that could be attenuated to different levels of precision as needed.

[1] M. Newman, *Computational Physics* (2013), revised and expanded ed., ISBN 978-1-4801-4551-1.

Appendix A: Declaration of Collaborators

I collaborated with Woody on this assignment while working on building my sine function.

Appendix B: Survey Question

I spent about 13 hours on this assignment (12 hours coding, 1 hour on the write-up), mostly on the trig functions. This assignment reminded me the importance of sampling: I thought my code was not working as I wanted it to but it was because I had set the user error to be too large, so the fractional error was always within my set condition after expanding the Taylor series out to only one term. The most important thing I learned was from calculating the error while building the trig functions. I have sort of assumed that computers are powerful enough not to have to worry about intrinsic computational error, which I'm realizing was pretty naïve. For this reason, I thought making the tangent function in particular was the most interesting, because I couldn't just divide the sine and cosine to build the function if I wanted to have a reasonable variance. Honestly, I think the problem sets are taking me longer than they should, but I am having a really hard time finishing them. I feel like the coding takes me so long that I don't have the time to write a quality write-up because I need to get something in.