Michael Nguyen
mnguy181@ucsc.edu
1/23/2021

CSE13S Winter 2021
Assignment 2: A Small Numerical Library
Design Document

PURPOSE

With some conditions, a function f(x) can be represented by its Taylor series expansion near a point f(a). In this program, we will utilize the Taylor series expansion method to expand to a finite number of terms to make our computation as accurate as possible in the final approximation.

In this program, I created a small numerical library and a corresponding test harness. I will be writing and developing 5 functions including sin, cos, tan, exponent, and log. While I will use the Taylor series approximation for sin, cos, tan, and exponent, I will use Newton's method instead to approximate log. These functions will then be tested later to compare outputs with the corresponding functions in the standard library <math.h> through test harness and output the results into a table. It is important to note that because sin, cos, tan, exp, and log are functions in the math library, I will instead be creating separate functions Sin, Cos, Tan, Exp, and Log without the use of any of the functions from the math library. In addition, I will not be defining the value of pi myself.

TOP LEVEL

The top level design of my code for mathlib.c is given by the following pseudocode. Note that more in-depth explanations on the implementation of each function follows and that this is just a brief general outline of how I will approach the programming assignment in regards to my math function implementations.
Please do note that the following pseudocode was developed through the help of TA Eugene from his lab section.

```
Abs(x):
    Check if x less than 0
    Multiply by -1 if less than 0
    Return value
Sin(x):
    Normalize the input values to be in the range [-2pi, 2pi]
    Declare variables (sum, term, etc.)
    For loop(start from 3, end when value is < Epsilon)
        Calculate term
        Keep track of the sign of each term (since +, - alternate)
        Add to sum
    Return the approximation
Cos(x):
```

```
    Normalize input values
    Declare variables (sum, term, etc.)
    For loop(start from 2, end when value < Epsilon)
        Calculate term
        Keep track of the sign of each term
        Add to sum
    Return the approximation
Tan(x):
    Normalize input value
    Return sin(x)/cos(x)
Exp(x):
    Declare variables (sum, term, etc.)
    For loop(start k from 1, end when term < Epsilon)
        Calculate term
        Add to sum
    Return the sum
Log(x):
    Declare variables (term, and previous)
    Use Exp() for previous
    While loop(end if previous - x is less than Epsilon)
        Term += (x-previous) / previous
        Previous = Exp(term)
    Return term
```

The file mathlib.c will include the header file “mathlib.h” which has been provided to us by Professor Darrell Long, and we will not be able to modify this file. Furthermore, Epsilon will be defined as 10^-14 at the top. Note that while we will not be able to use the <math.h> functions, on Piazza Note @379, Professor Darrell Long stated that he is granting permission to utilize the fmod() function. Because this will be helpful in normalizing the input values to be in the specified range from the Lab pdf [-2pi, 2pi], this will be the only function from the <math.h> library that will be used.

DESCRIPTION

Function double Abs(double x)

The purpose of this function will be to return the absolute value of the input x. Since I will not be able to use the abs() function included in the <math.h> library, this function is very useful because it will be essential in the other 5 functions when comparing each term to epsilon, checking if the difference between the current approximation and the previous approximation is less than Epsilon. If the value is considered to be smaller, then we can break out of the loop, and return the approximation computed.

The implementation of this function is fairly easy. If the value of x is less than 0, then return -x. Otherwise, return x. Attending Eugene’s section was particularly helpful in implementing this function.

Function double Sin(double x)

This function is one of the four functions that will utilize the Taylor Expansion method. In order to normalize the input value x to be in the range that was specified in the lab document [-2pi, 2pi], I will be utilizing the <math.h> library function fmod() (floating point modulo) in order to set x into the specified range that centers around zero. The reasoning behind this is that <math.h> library sin() function should be able to still work even when the input value x is outside of the specified range. In other words, the function should be able to work with any valid double. Using the fmod() function should look like the following:

x = fmod(x, 2.0 * M_PI);

Next, we will initialize the variables that will be used in the function. The variables that will be needed include the Numerator (set to x), the Denominator (set to 1 since 1! = 1), Term (set to Numerator / Denominator), and the sum (initially set to x). Furthermore, because the Taylor Expansion method switches signs when computing the approximation, we will also initialize a variable sign (set to -1). In the loop that follows, we will alternate signs by multiplying by -1, and multiplying the sign by the term before adding to the sum. In addition, I believe that it will also be helpful to keep track of the previous sum (initially set to 0). The purpose behind keeping track of the previous sum is to break out of the following for() loop when the absolute value of the (current sum - previous sum) is less than Epsilon. In other words, we will not need to iterate through the loop anymore since the value that will be added/subtracted is too small to really matter. The lab document states that Epsilon will be set to 10^-14.

In this function, I will initialize k to start at 3, incrementing k by 2, while also multiplying sign by -1 after each iteration. The loop's end condition will be if the value of sum-previous sum is less than Epsilon, then the loop will end. The purpose behind this loop will be to serve as the implementation for expanding the Taylor series to compute as accurate of an approximation that we can. Based on the lab document, we can see that with the expansion of terms, we can see that the numerator is multiplied by x^2 with each iteration. Thus, with each iteration, numerator = numerator * x * x. The denominator on the other hand will be multiplied by (k-1) * k. Next, we will compute the current term = numerator / denominator followed by a multiplication by the sign. Before adding the term to the total sum, we will set the previous sum to be the current sum.

When the loop has reached its end condition, we will return the approximated value, sum.

Function double Cos(double x)

Along with Sin(x), this function will also be implemented in a similar fashion with the Taylor Series Expansion method. The implementation will be almost exactly the same as the Sin() function. However, there are some differences when initializing and declaring variables. According to the lab document, when the Taylor series is expanded, the first term will be 1 instead of x. To adjust to these changes, variables numerator, denominator, and sum will be initialized to 1 instead of x.

Implementation of the for() loop will be exactly the same except for the starting point of k. As you can see on the lab document, k on the second term begins at 2 instead of the 3 that we saw in Sin(x). Thus, the for() loop will begin with k=2, incrementing k += 2 after each iteration, halting when the final term is less than Epsilon. Again, we will keep track of the sign and previous sum with each iteration, returning the final sum when the loop ends.

Function double Tan(double x)

Tan(x) will also be implemented utilizing the Taylor Series Expansion method. Because expanding this will be difficult, we can instead simplify tan(x) to be sin(x)/cos(x). When using the tan(x) function from <math.h> even if cos(x) == 0, the function still returns a very small value, and thus, I have chosen to not worry about the case when cos(x) is equal to zero. The implementation of this function is fairly easy. We will first normalize x with fmod(x, M_PI / 2). Set a temporary variable i to sin(x). Set another temporary variable j to cos(x), and return i / j.

Function double Exp(double x)

Exp(x) will be implemented utilizing the Taylor Series Expansion method. The variables that will need to be initialized will be numerator (set to 1.0), denominator (set to 1.0), and the term (set to numerator / denominator). In addition, sum will be first initialized to term.

Next, we will implement a for() loop with k starting at 1.0, incrementing by 1 with each iteration of the loop, and ending when the absolute value of the term is less than Epsilon. The logic behind this for() loop, as described in the lab document, is that with each iteration, we can see that to get the term, we simply have to multiply the previous term by (x/k) and add it to the total. Eventually, we will see that k! eventually overwhelms x^k, having the ratio rapidly approach zero. Within the for() loop, we will thus multiply term by x/k and add it to the sum, returning sum when the loop ends.

Function double Log(double x)

Log(x) is the only function that will not be implemented utilizing the Taylor Series Expansion method. Instead, it will utilize Newton's method, ending when the difference between consecutive approximations is less than Epsilon.We will first set x_n equal to 1.0. Then, we will set the previous approximation equal to Exp(x_n). Using a while() loop that ends when the difference between the previous approximation and current approximation is less than Epsilon, we will set x_n += (x - previous) / previous. The previous approximation will then become Exp(x_n) again. We will then return x_n when the loop has ended.

** Please note that this function was heavily inspired by Professor Darrell Long's Log() function shown during lecture. All credit goes to Professor Darrell Long.**