Devan O'Boyle
CSE 13S

Assignment 2: A Small Numerical Library

Purpose: In this lab we are implementing the arcsin, arcos, arctan, and log functions using only addition, subtraction, multiplication, and division since those are the only mathematical functions that a computer can compute. In order to implement these functions, Taylor series and Newton's method will be used to achieve the closest possible values for each of the functions to that of the math.h library in c.

In order to make these functions, first we have to make some functions to help us, like absolute value, square root, and exponential.

Absolute value is simple, take an input x, and if it's less than 0, then return -x.

For the square root function, we take an input x. Then we also have a value that will serve as a guess for the approximation of the square root. That can be set to 1. A while loop will be used to compute the approximation by coming up with a difference between the current guess and the previous guess that is smaller than epsilon. Each guess will be computed using Newton's method for square roots: guess = (previous guess + x/previous guess)/2. The function will return an approximation for the square root of x as accurate as the value of epsilon.

Note: for this lab the value of epsilon is 1*10^-10

For exponent, we have to use the formula:

$$\frac{x^k}{k!}:$$

In order to implement this, the formula can be expanded to look something like:

$$\frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$$

For this formula, we can use a for loop to iterate through each value of k. This will account for the factorial of k on the bottom. A double variable will be used to save a value which will be added to by x/k. The loop will end once the absolute value of the double variable is less than or equal to the value of epsilon.

Now to implement the sin function we implement the taylor series given by the formula:

$$\arcsin(x) = \sum_{k=0}^{\infty} \frac{(2k)!}{2^{2k}(k!)^2} \frac{x^{2k+1}}{2k+1}, \quad |x| \leq 1.$$

Here, the taylor series is centered around 0, because the domain for both arcsin and arccos is [-1,1].
Since the taylor series formula is in summation notation, that means a loop needs to be used to add up all of the terms. In this case, a while loop will be used as we are trying to get the output to be as specific as possible so long as the last term added in the taylor series is greater than the value of epsilon.

This formula is still a bit too complex to compute so let's use a simplified formula:

$$\sum_{n=0}^{\infty} \frac{(2n-1)!!}{(2n)!!} \frac{z^{2n+1}}{2n+1}$$

Here, we have a nice even double factorial on the top and the bottom. This will help to streamline the calculation as we will first compute the terms in the numerator: (2n - 1) and denominator: (2n), divide them, and then add the next iteration in a for loop to account for the factorial. This means that a nested for loop will be used to compute the factorial, but since we are dividing each iteration and then multiplying them to the total factorial, the values will never get too big. The loop should continue while the iterated value, n, is less than or equal to the value of k.

Now the $\frac{(2n-1)!!}{(2n)!!}$ part of the equation has been solved, so now it is time to evaluate $\frac{z^{2n+1}}{2n+1}$

We can't simply compute the power function on top and then divide because then the value will get too large to compute. So instead we first divide the factorial value from before by the (2k+1). Then a for loop will be used to multiply x (in the equation above it says z), by the value we just computed and set it equal to the variable that that value was stored in. The loop should run for as long as the iterated value is less than or equal to 2k + 1.

Once this value has been calculated and initialized to a variable, it can be added to the sum total which will continue to add these values until the final term is smaller than the value of epsilon, in which case we should have a precise enough value and the loop can exit.

Luckily the formula for arccos is:

$$\arccos(x) = \frac{\pi}{2} - \sum_{k=0}^{\infty} \frac{(2k)!}{2^{2k}(k!)^2} \frac{x^{2k+1}}{2k+1}.$$

Which is the same as:

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x).$$

This means that the arccos function can be implemented quite easily by taking the value of pi/2. and then subtracting it by the arcsin value of x by calling the arcsin function with x as the input.

Note: the value of pi will be taken from the math.h library

As for arctan, the formula is:

$$\arctan(x) = \sum_{k=0}^{\infty} \frac{2^{2k}(k!)^2}{(2k+1)!} \frac{x^{2k+1}}{(1+x^2)^{k+1}}.$$

But rather an having to implement another complicated formula, we can use this formula instead that uses arcsin:

$$\arctan(x) = \arcsin\left(\frac{x}{\sqrt{x^2+1}}\right)$$

This means that all that has to be computed here is x/sqrt(x^2+1). First, x^2 will be computed by multiplying x by x. Then 1 will be added to that value. We can call the square root function that we made earlier on that value. Then we can simply divide x by the square rooted value and plug it in to the arcsin function to get arctan.

Let’s go back to arcsin, since while Taylor series is fairly accurate for integers closer to 0, the closer you get to -1 or 1, the more inaccurate the approximation is going to be. Therefore we are going to use the following formula to compute arcsin for when the value of x is between -1 and -0.75 as well as 0.75 and 1:

$$\sin^{-1}(x) = \cos^{-1}(\sqrt{1-x^2}), \;\; 0 \le x \le 1.$$

This formula can be further evaluated out using the other function for arcsin and arccos from before:

$$\frac{\pi}{2} - \arcsin\left(\sqrt{1-x^2}\right) = \arcsin(x)$$

Note: If the initial value of x was negative be sure to change it back to a negative at the end since the squared function will have removed it during computation.

For the log function, rather than using taylor series, we will instead use Newton’s method:

$$x_{k+1} = x_k + \frac{y - e^{x_k}}{e^{x_k}}.$$

Note that in this function y is the input, since we are going by ln(e^y) = x. Therefore, I’m going to rewrite this function with the variables switched for clarity:

$$y_{k+1} = y_k + \frac{x - e^{y_k}}{e^{y_k}}$$

Now this makes more sense given the context of the program. Here, x will serve as the input variable like it has in the other functions, and y is the variable that will be continuously added to. The double variable y will be initialized to 1.0 since we are technically computing ln and ln(0) is undefined, therefore we will start at 1. By using the exponent function, we can plug y into it to get e^y. Then a while loop can be used to compute the sum. The while loop should only end once the difference between e^y and the input x is less than or equal to the value of epsilon, that way we get a value that is as precise as we need it to be. In the while loop the equation will be computed by taking x - e^y and dividing it by e^y. Then we take that value and add it to the current y value to become the new y value. The value of e^y should be reinitialized to account for the new y value. Once the loop breaks the function should return y.

In order to print these functions out, get opt will be used to have a different case to print out each function.

For instance, for arcsin there will be a case s that will print a column with each value of input x, the return value of the function arcSin, the return value of the math.h function asin, and the difference between the return values of arcSin and asin. This will be done for all of the different functions. There will also be a case that prints out the output for every function.

In order to prevent functions being printed out twice, boolean variables for each function will be used to check to make sure that the case for each of the corresponding functions hasn't already been called before. Once a case is called, the boolean variable for that case is set to true, then an if statement will be used to check for when that case is called again so that if the boolean variable has already been set to true, then the case will break and not print anything.