Marian Zlateva
mzlateva@ucsc.edu
1/24/2021

CSE13s Fall 2020
Assignment 2: A Small Numerical Library

Design Document

In this lab, I wrote up several functions that output similar results to those represented in the <math.h> library. I compared them with the functions in the <math.h> library to see how accurate they were. These functions are Sin(x), Cos(x), Tan(x), Exp(x), and Log(x). All of these functions use Taylor series and Newton's formula to reach their answers. The inputs to this program are a series of flags:

-a runs all tests (sin, cos, tan, exp, log)
-s runs sin tests
-c runs cos tests
-t runs tan tests
-e runs exp tests
-l runs log tests

We will have 3 files in this program. One that reads the user's input and calls the corresponding functions. Another that serves to declare the functions (given to us). And finally, one that defines the functions.

Mathlib-test.c

Main:
int opt = 0
bool do_sin, do_cos, do_tan, do_exp, do_log

if there are no arguments, call the terminating function (void inputError)

Read program arguments
set corresponding booleans to true
-a runs all tests (sin, cos, tan, exp, log)
-s runs sin tests
-c runs cos tests
-t runs tan tests
-e runs exp tests
-l runs log tests
if any other argument is passes, call the terminating function

-perform functions of which corresponding booleans are true

-all outputs are formatted with printHeader() and printInFormat() which print the functions in a consistent format

sin:

compares outputs of Sin(x) and sin(x) from [-2pi,2pi]

cos:

compares outputs of Cos(x) and cos(x) from [-2pi,2pi]

tan:

compares outputs of Tan(x) and tan(x) from [-pi/3,pi/3]

exp:

compares outputs of Exp(x) and exp(x) from [1,10]

log:

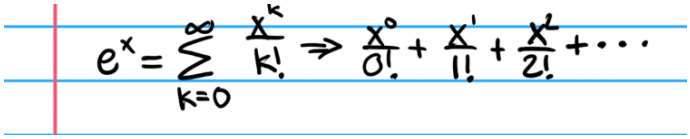
compares outputs of Log(x) and log(x) from [1,10]

Mathlib.c, to be included in Mathlib.h

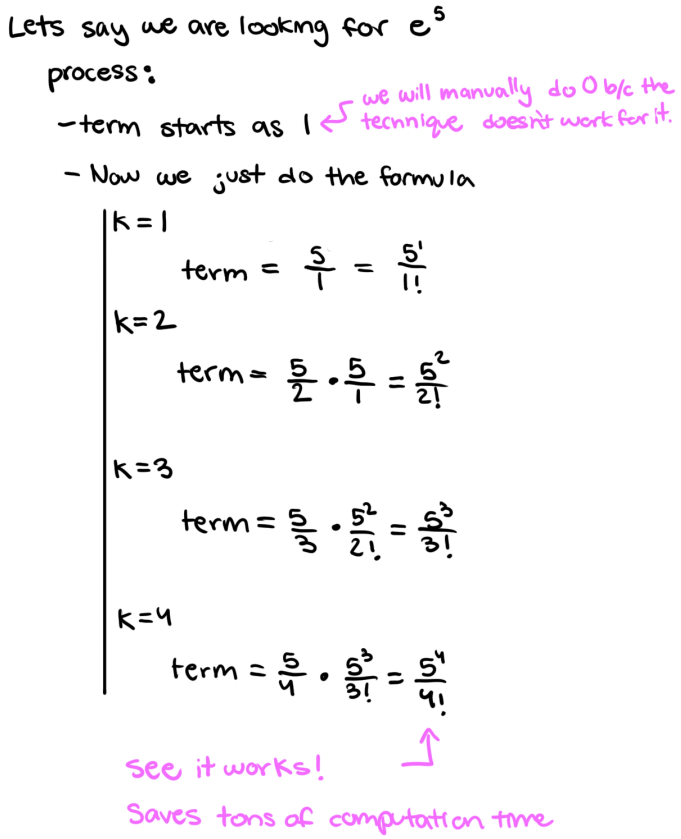
Exp

basically approximates the value of the function f(x) = e^x

the formula says this:



-  is a term
- As the computation proceeds, every new term will be smaller than the last bc k! grows faster than x^k
- basically we need to keep adding a term until each new term is smaller than E
- we don't need to do any for loops or anything when calculating new terms bc we can just use the last term we calculated



- Remember to stop when the term is less than E so that we don't go forever!
- Now we just need to add up all the terms and we have the answer (this will be done inside a loop)
- The professor gave us the code but it basically works like this:

```
Exp(x):
        term = 1
    make sure all values are long doubles so that the number is exact and there are no
    errors when comparing it to E
        sum = term
    this is basically manually doing the k=0 term

    loop a variable k=1 and increment each round by 1 until the new term's value is less
    than E
            term = (x/k)*term
            sum += term

    return sum
```

Log(x):

- basically approximates the value of the function f(x) = log(x)
- Here we will use newtons formula
- We can express the equation y=ln x in terms of 0 by doing the following conversion

  $f(x)=\ln x$

  $y=\ln x$

  $e^y=x$

  $0=x-e^y$
- $g(y)=x-e^y=0$
- Newtons formula will allow us to approximate the value of y such that the equation = 0
- Remember that 'y' is going to be the solution of the equation as shown in y=lnx
- Newton's formula states
  - $y_{n+1}=y_n+\frac{g(y_n)}{g'(y_n)}$
- if we substitute the g(x) equation for the actual equation we get
  - $y_{n+1}=y_n+\frac{x-e^{y_n}}{e^{y_n}}$
    - the derivative of x-e^y is just e^y bc x is a constant in the equation
- the more times we compute this equation, the more accurate of an answer we will get
- e^x taylor series requires that the first guess is 1
- we will stop computing this until e^y gets very close to to x (|e^y - x| < E)
- The professor gave us the code but it basically works like this (remember x is the argument):

Log(x):

var y = first guess (1)
var p = e^y
run a loop of the equation until |e^y - x| < E
    y = y + (x - p) / p
        (p is already e^y so we don't calculate it again until y changes)
    p = e^y (Now that y has changed, we calculate e^y again)
return y

Sin(x):

- basically approximates the value of the function f(x) = sin(x)
- We will use taylor series to do this bc its periodic and always differentiable
- the taylor series formula for sinx is $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$
- This is how the process can be computed

sum = x
num = x
den = 1
loop:

k = 3

used to change the sign every other time

$num = num \cdot -x \cdot x = x \cdot -x \cdot x = -x^3$

$den = den \cdot 3 \cdot (3-1) = 1 \cdot 3 \cdot 2 = 3!$

k(k-1) increases factorial by 2 when multiplied with den

$sum = \frac{num}{den} = x + \frac{x^3}{3!}$

k = 5

$num = num \cdot -x \cdot x = -x^3 \cdot -x \cdot x = -x^5$

$den = den \cdot 5(5-1) = 3! \cdot 5 \cdot 4 = 5!$

$sum = x + \frac{x^3}{3!} + \frac{x^5}{5!}$

- That's all the formula has to it!
- The professor gave us the code but it basically works like this (remember x is the argument):

Sin(x):

- var sum = x (to compute for first term)
- var num = 1 (also the first term's numerator)
- var den = 1 (also the first term's denominator)
- loop with a var k starting at 3 and increment it twice each time the loop passes. Stop loop when numerator/denominator is less than E. that means that the equation is no longer changing that much and it's a good enough estimate
  - num = num * -x * x
  - den = den * k(k-1.0)
  - fraction = num/den
  - sum= sum +fraction
- if the fraction is greater than E, break out of the loop
- return sum

Cos(x):

- basically approximates the value of the function f(x) = cos(x)
- cos is very close to sin, we just need to change our starting value for k from 3 to 2
- here is how the taylor series for cosx looks like

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

- we only need to change 2 variables: the starting number from x to 1 and the starting increment number from 3 to 2
- for the code, its exactly the same as sin but you change these 2 variables

Tan(x):

- basically approximates the value of the function f(x) = tan(x)
- this one is the easiest, all you need to do is use the last 2 functions Sin(x) and Cos(x) bc Tan(x)=Sin(x)/Cos(x)
- The code will behave like this:
  - return Sin(x)/Cos(x)
- That's it!

Design Process

Most of the code in this lab was already given to us. Understanding it was the challenge. It basically took me the entirety of Wednesday and a bit of Thursday to fully grasp the concepts presented in the example code. The most difficult part was Log(x) because you have to inverse e^(x) and then put in Newton's formula. I even ended up watching a video all about Newton's formula to understand it. In the end, I was able to be pretty proud of my work.