Simon Lee
CSE 13S

Write Up

Introduction:

As stated in my DESIGN.pdf, computations in a computer can only be done with the basic operators: addition, subtraction, multiplication, and division. We also should mention that computers are not capable of giving definite answers and that all “answers” given by a math library, and in this case, our library are simply just approximations. That is why in this WRITEUP we will explain how the differences from our mathematical functions and the standard math library functions.

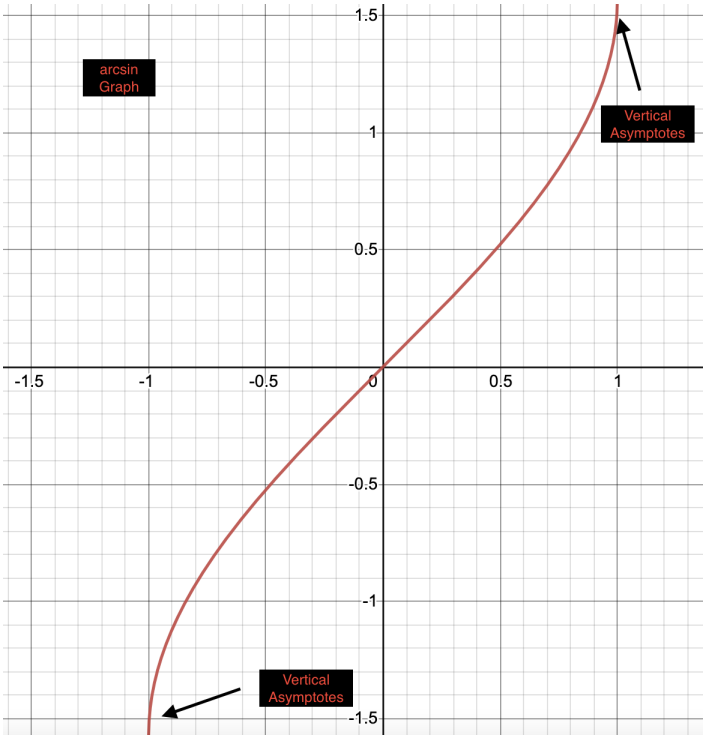
ArcSin:

My output:

```
 x           arcSin          Library         Difference
 -           ------          -------         ----------
iterations 0
iterations 0
iterations 0
iterations 0
-1.0000      -1.57079633     -1.57079633     0.0000000001
iterations 73
iterations 73
-0.9000      -1.11976951     -1.11976951     0.0000000003
iterations 37
iterations 37
-0.8000      -0.92729522     -0.92729522     0.0000000001
iterations 24
iterations 24
-0.7000      -0.77539750     -0.77539750     0.0000000001
iterations 17
iterations 17
-0.6000      -0.64350111     -0.64350111     0.0000000000
iterations 13
iterations 13
-0.5000      -0.52359878     -0.52359878     0.0000000000
iterations 10
iterations 10
-0.4000      -0.41151685     -0.41151685     0.0000000000
iterations 8
iterations 8
-0.3000      -0.30469265     -0.30469265     0.0000000000
iterations 6
iterations 6
-0.2000      -0.20135792     -0.20135792     0.0000000000
iterations 4
iterations 4
-0.1000      -0.10016742     -0.10016742     0.0000000000
iterations 0
iterations 0
-0.0000      -0.00000000     -0.00000000     0.0000000000
```

```
 0.1000       0.10016742      0.10016742     -0.0000000000
iterations 6
iterations 6
 0.2000       0.20135792      0.20135792     -0.0000000000
iterations 8
iterations 8
 0.3000       0.30469265      0.30469265     -0.0000000000
iterations 10
iterations 10
 0.4000       0.41151685      0.41151685     -0.0000000000
iterations 13
iterations 13
 0.5000       0.52359878      0.52359878     -0.0000000000
iterations 17
iterations 17
 0.6000       0.64350111      0.64350111     -0.0000000000
iterations 24
iterations 24
 0.7000       0.77539750      0.77539750     -0.0000000001
iterations 37
iterations 37
 0.8000       0.92729522      0.92729522     -0.0000000001
iterations 73
iterations 73
 0.9000       1.11976951      1.11976951     -0.0000000003
iterations 1996473
iterations 1996473
 1.0000       1.57039703      1.57079631     -0.0003992733
```

Since I had trouble generating graphs on the terminal. I resorted to explaining my differences based on the amount of iterations that were executed. So in my code I made variable iterations that would increment by one every time the loop was executed. When it came to doing approximations for arcSin (1) and arcSin (-1) which are vertical asymptotes as shown below:



When making approximations at these vertical asymptotes you would typically run into an error because the derivatives of the two specific spots are 0. Therefore we can get a closer approximation by using some trigonometric identities. That is why for x = -1.0, I have decided to use the trigonometric identity,

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

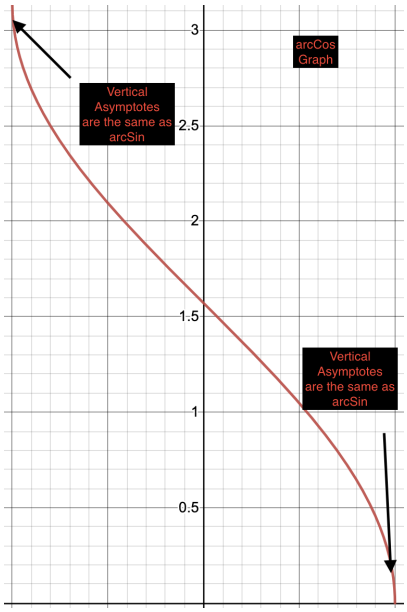
However since we are accounting for -1 we must take into account that it will be a negative value. Therefore we multiply the whole solution by -1. Why do we do this you may be asking? Well if we analyze our approximations and the graph you will notice that everything is in quadrants 1 and 3 in which quadrant 1 only contains all positive values, and quadrant 3 contains only negative values. Therefore, when approximating 1, we put it through the trigonometric identity and get the value 1.57079633. But since -1 is in the third quadrant as previously mentioned, we must multiply the whole answer by -1. That is how we get my answer. Next you will notice that values -0.9 and 0.9 have the highest differences overall and it also undergoes the most iterations in the Taylor Series. After that it slowly becomes more accurate as they approach zero. However they're is a peculiar case that even I cannot explain which happens at x = 1.0. So just like x = -1.0, we have to approximate these values using the trigonometric identity. Building it exactly like x = -1.0, I used an if statement to know when exactly to use the trigonometric identity. However I struggled to ever make it work and have left comments wondering if it might be a computer issue. Therefore the approximations for the value x = 1.0 is really off because it doesn't go through the conditional that I have built it and instead is approximated through the Taylor Series.

arcCos()

My output:

```
 x           arcCos          Library         Difference
 -           ------          -------         ----------
iterations 0
iterations 0
iterations 0
iterations 0
-1.0000      3.14159265      3.14159265     -0.0000000001
iterations 73
iterations 73
-0.9000      2.69056584      2.69056584     -0.0000000003
iterations 37
iterations 37
-0.8000      2.49809154      2.49809154     -0.0000000001
iterations 24
iterations 24
-0.7000      2.34619382      2.34619382     -0.0000000001
iterations 17
iterations 17
-0.6000      2.21429744      2.21429744     -0.0000000000
iterations 13
iterations 13
-0.5000      2.09439510      2.09439510     -0.0000000000
iterations 10
iterations 10
-0.4000      1.98231317      1.98231317     -0.0000000000
iterations 8
iterations 8
-0.3000      1.87548898      1.87548898     -0.0000000000
iterations 6
iterations 6
-0.2000      1.77215425      1.77215425     -0.0000000000
iterations 4
iterations 4
-0.1000      1.67096375      1.67096375     -0.0000000000
iterations 0
iterations 0
-0.0000      1.57079633      1.57079633      0.0000000000
 0.1000      1.47062891      1.47062891      0.0000000000
iterations 6
iterations 6
 0.2000      1.36943841      1.36943841      0.0000000000
iterations 8
iterations 8
 0.3000      1.26610367      1.26610367      0.0000000000
iterations 10
iterations 10
 0.4000      1.15927948      1.15927948      0.0000000000
iterations 13
iterations 13
 0.5000      1.04719755      1.04719755      0.0000000000
iterations 17
iterations 17
 0.6000      0.92729522      0.92729522      0.0000000000
iterations 24
iterations 24
 0.7000      0.79539883      0.79539883      0.0000000001
iterations 37
iterations 37
 0.8000      0.64350111      0.64350111      0.0000000001
iterations 73
iterations 73
 0.9000      0.45102681      0.45102681      0.0000000003
iterations 1996473
iterations 1996473
 1.0000      0.00039929      0.00000002      0.0003992733
```

Very similar to arcSin, we have arcCos which is actually approximated using the same Taylor Series as arcSin. However we subtract $\frac{\pi}{2}$ by the total so the differences from [-0.9, 0.9] are basically identical to that of arcSin. However looking at the graph below we notice that arcCos behavior overall is not like a reflection of itself like arcSin was:



If we look at this graph we can see that we get a unique value for the range [-1.0,1.0]. Since we have vertical asymptotes in the same places we treat these with trigonometric identities once more in which the value at x = 1.0 has once again messed up. However the identity is as follows:

$$cos^{-1}(x) = sin^{-1}(\sqrt{1 - x^2})$$

However that is not all. We should note that unlike arcSin we need to add or subtract $\frac{\pi}{2}$as deemed necessary. $sin^{-1}(0) = 1.57079633$. However we can see on the values of the table that at $cos^{-1}(-1)$ We get the value π. and at $cos^{-1}(1.0)$, we get the value zero. However like previously mentioned I implemented both of these asymptotic values the same way but the if statement at x = 1.0 never seems to go through. I have left comments on my code, have marked it on my README.md, and have referenced it many times. However like the arcSin we can see that the differences are pretty much identical to those of arcSin. However it is important to notice the amount of iterations required as we go up the range [-0.9, 0.9]. Like arcSin the further we stem from 0 the more iterations that are required to get an approximation that is less than ϵ.We can see that the Taylor series is pretty effective overall to compute the approximations of these trigonometric functions.

```
x           arcTan          Library        Difference
-           ------          -------        ----------
iterations: 24.000000
iterations: 24.000000
 1.0000       0.78539816       0.78539816    -0.0000000001
iterations: 28.000000
iterations: 28.000000
 1.1000       0.83298127       0.83298127    -0.0000000001
iterations: 31.000000
iterations: 31.000000
 1.2000       0.87605805       0.87605805    -0.0000000001
iterations: 35.000000
iterations: 35.000000
 1.3000       0.91510070       0.91510070    -0.0000000001
iterations: 39.000000
iterations: 39.000000
 1.4000       0.95054684       0.95054684    -0.0000000002
iterations: 44.000000
iterations: 44.000000
 1.5000       0.98279372       0.98279372    -0.0000000002
iterations: 48.000000
iterations: 48.000000
 1.6000       1.01219701       1.01219701    -0.0000000002
iterations: 53.000000
iterations: 53.000000
 1.7000       1.03907226       1.03907226    -0.0000000002
iterations: 58.000000
iterations: 58.000000
 1.8000       1.06369782       1.06369782    -0.0000000003
iterations: 64.000000
iterations: 64.000000
 1.9000       1.08631840       1.08631840    -0.0000000003
iterations: 69.000000
iterations: 69.000000
 2.0000       1.10714872       1.10714872    -0.0000000003
```

```
iterations: 937.000000
iterations: 937.000000
 9.0000       1.46013910       1.46013911    -0.0000000072
iterations: 955.000000
iterations: 955.000000
 9.1000       1.46134537       1.46134538    -0.0000000074
iterations: 974.000000
iterations: 974.000000
 9.2000       1.46252573       1.46252574    -0.0000000075
iterations: 993.000000
iterations: 993.000000
 9.3000       1.46368100       1.46368100    -0.0000000076
iterations: 1011.000000
iterations: 1011.000000
 9.4000       1.46481196       1.46481197    -0.0000000079
iterations: 1030.000000
iterations: 1030.000000
 9.5000       1.46591938       1.46591939    -0.0000000080
iterations: 1049.000000
iterations: 1049.000000
 9.6000       1.46700398       1.46700399    -0.0000000082
iterations: 1069.000000
iterations: 1069.000000
 9.7000       1.46806645       1.46806646    -0.0000000083
iterations: 1088.000000
iterations: 1088.000000
 9.8000       1.46910747       1.46910748    -0.0000000085
iterations: 1108.000000
iterations: 1108.000000
 9.9000       1.47012767       1.47012767    -0.0000000086
iterations: 1127.000000
iterations: 1127.000000
 10.0000      1.47112767       1.47112767    -0.0000000089
```

arcTan()

My output:

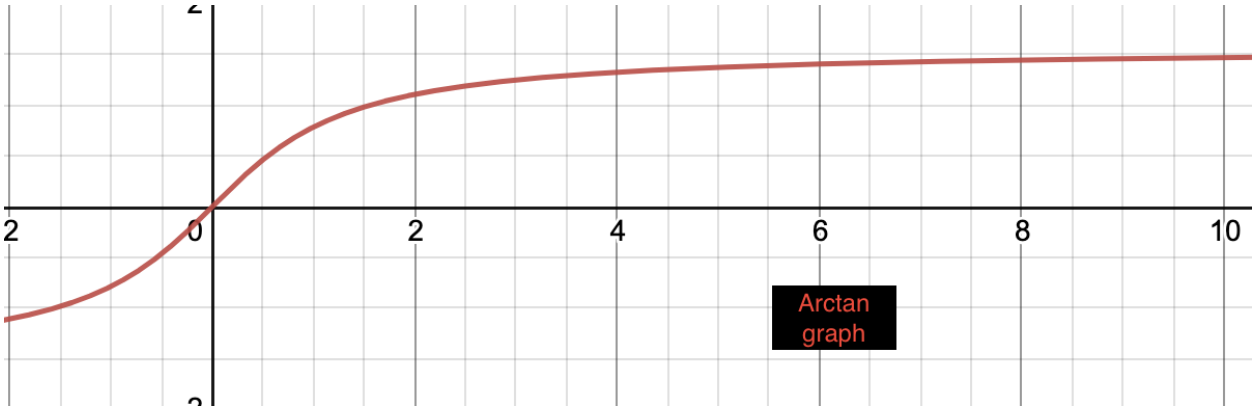
With arctan I decided not to provide all the screenshots simply because the trends are obvious. So I have included the following screenshots with range arctan[1,2] and arctan[9,10]. Arctan was one of two functions where we had to print it out in the range [1,10]. If you ask why it's because the closer it gets to 0 it converges quickly. Reference the graph below:

Though arcTan has a similar behavior to arcSin in the sense that all the positive x values lie in quadrant 1 and all the negative values lie in quadrant 3, here we have a very different situation because our range is different to begin with. Here you notice that the differences slowly ramp up the further it gets away from 0 but that makes sense as these approximations slowly become less accurate the further the taylor series has to stem out. Like arccos, arctan is derived from arcsin and we use the trigonometric identity to compute the approximation:

$$tan^{-1}(x) = sin^{-1}(\frac{x}{\sqrt{x^2+1}})$$

Using this identity we can find the values for arctan through a Taylor Series. Also notice how many more iterations that undergoes the further the solution stems away from 0. This was analogous to that of arcSin and arcCos as well as the approximations became less and less accurate as they stemmed out further. Also if we analyze the graph and the outputs we can see that the answers are inceremenating at a much slower rate and that is why the iterations increase because extremely small values are being added to the taylor series approximation which is ultimately just a summation.

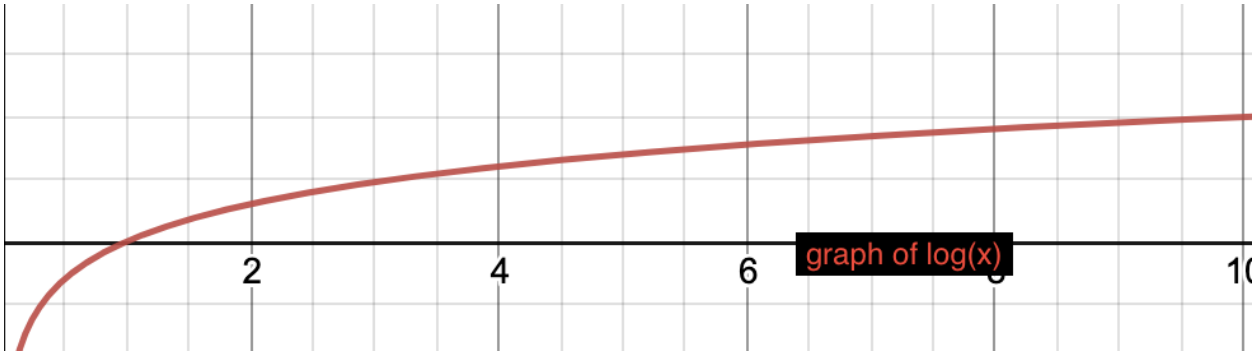
Log()
My output:



Lastly we have Log. Log was also another one of those functions that we had to calculate approximations for the range [1,10]. The reason being is that for any value less than 0 the answer would be undefined. Look at the graph as follows:

```
x           Log             Library         Difference
-           ------          -------         ----------
iterations: 5
iterations: 5
 1.0000      0.00000000      0.00000000      0.0000000000
iterations: 5
iterations: 5
 1.1000      0.09531018      0.09531018      0.0000000000
iterations: 5
iterations: 5
 1.2000      0.18232156      0.18232156      0.0000000000
iterations: 5
iterations: 5
 1.3000      0.26236426      0.26236426      0.0000000000
iterations: 5
iterations: 5
 1.4000      0.33647224      0.33647224      0.0000000000
iterations: 5
iterations: 5
 1.5000      0.40546511      0.40546511      0.0000000000
iterations: 5
iterations: 5
 1.6000      0.47000363      0.47000363      0.0000000000
iterations: 4
iterations: 4
 1.7000      0.53062825      0.53062825      0.0000000000
iterations: 4
iterations: 4
 1.8000      0.58778666      0.58778666      0.0000000000
iterations: 4
iterations: 4
 1.9000      0.64185389      0.64185389      0.0000000000
iterations: 4
iterations: 4
 2.0000      0.69314718      0.69314718      0.0000000000
```

```
 9.0000      2.19722458      2.19722458      0.0000000000
iterations: 7
iterations: 7
 9.1000      2.20827441      2.20827441      0.0000000000
iterations: 7
iterations: 7
 9.2000      2.21920348      2.21920348      0.0000000000
iterations: 7
iterations: 7
 9.3000      2.23001440      2.23001440      0.0000000000
iterations: 7
iterations: 7
 9.4000      2.24070969      2.24070969      0.0000000000
iterations: 7
iterations: 7
 9.5000      2.25129180      2.25129180      0.0000000000
iterations: 7
iterations: 7
 9.6000      2.26176310      2.26176310      0.0000000000
iterations: 7
iterations: 7
 9.7000      2.27212589      2.27212589      0.0000000000
iterations: 7
iterations: 7
 9.8000      2.28238239      2.28238239      0.0000000000
iterations: 7
iterations: 7
 9.9000      2.29253476      2.29253476      0.0000000000
iterations: 7
iterations: 7
 10.0000      2.30258509      2.30258509      0.0000000000
```



The graph actually has a vertical asymptote at x = 0.0 so it wouldn't make sense to compute an approximation for anything less than 0. However though I won't take credit for making this function, Newton's method appears to be far more accurate much quicker. What I mean by that is that it requires less iterations to compute by a longshot versus the Taylor Series Method. The code that I used from Professor Long's lecture slides have even been deemed super accurate because the differences are so small that it can't even be seen. I think it is fair to admit that Professor Long's math functions are far more accurate than those of the math.c library as he mentioned in class. I think it is very impressive that it only takes at most 7 iterations to get an approximation so accurate to that of the math library. If it was not made clear in my DESIGN.pdf, the way you compute using Newton's method is you have a guess, a previous guess, a f(x), and a f'(x). In this case our guess begins at 1.0, $f(x) = x - e^y$ and $f'(x) = e^y$. So by continuously looping through we slowly get a guess that is accurate enough to be an appropriate approximation by taking the absolute value of the guess with the value of $e^{previous\ guess}$.

Final Thoughts

I have learned from this lab that building mathematical functions is no easy task but it is possible through Calculus and other advanced Mathematical tools. Though Newton's method proved to be far more accurate much quicker, I think in my case that the taylor series made more intuitive sense and that the approximations were far closer than when I attempted to make it with Newton's Function. But it was cool to see the amount of iterations it actually took to obtain a value less than epsilon and since my differences were not completely 0.000000 for most of my cases, it shows that if I was able to graph it, that at a minimal scale that my approximations do not match those of the mathematical library. But again, computers do not give definitive answers and instead are just approximations like our mathematical implementations of these functions. So even that of the math library are not accurate to the full extent. I learned a lot overall and enjoyed the lab overall.