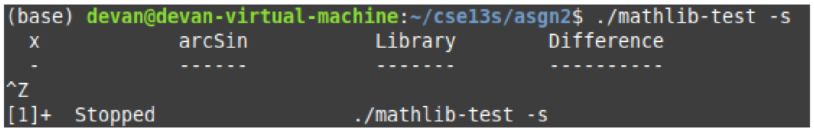
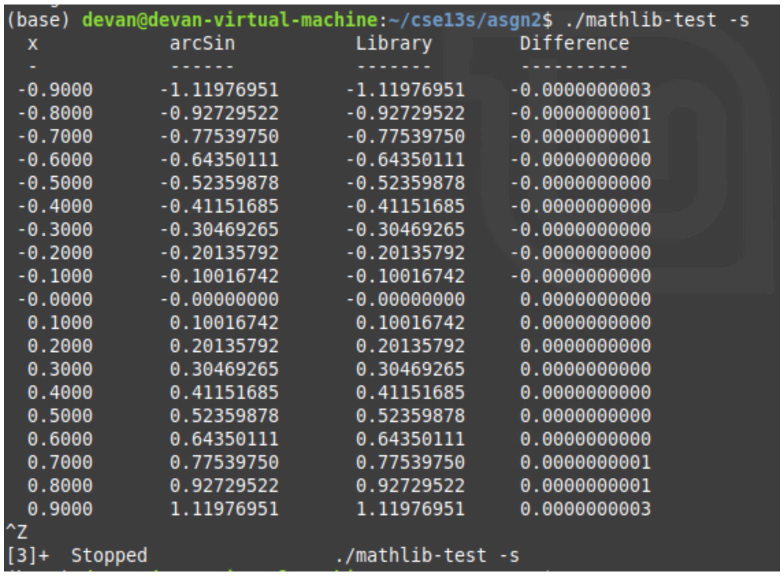
Devan O'Boyle
CSE 13s
Assignment 2: A Simple Numerical Library

Write Up

For this assignment, I used Taylor series to compute my arcSin function. When I first implemented the function, I hit an infinite loop at -1 and 1 as shown here:

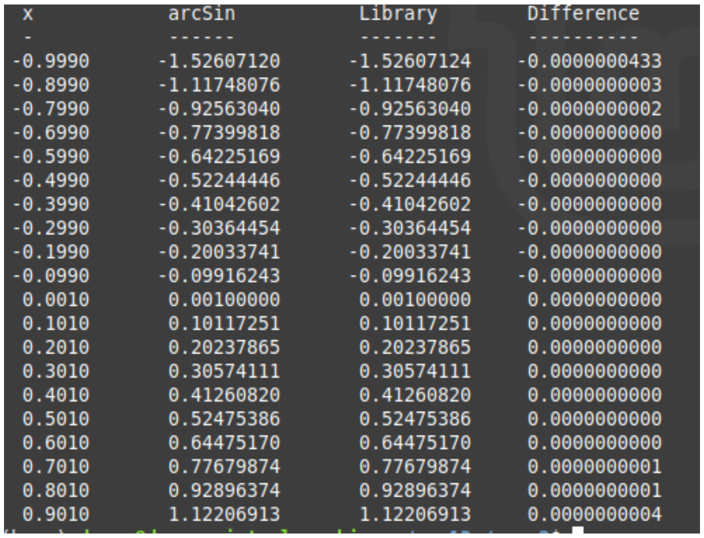


```
(base) devan@devan-virtual-machine:~/cse13s/asgn2$ ./mathlib-test -s
  x            arcSin           Library        Difference
  -            ------           -------        ----------
^Z
[1]+  Stopped                 ./mathlib-test -s
```



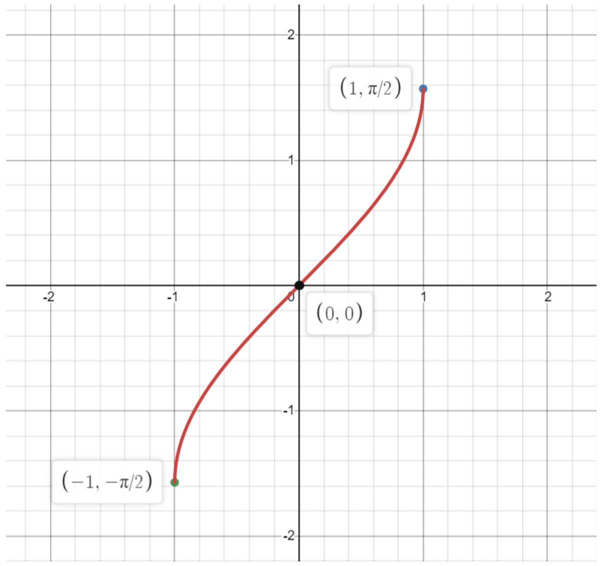
```
(base) devan@devan-virtual-machine:~/cse13s/asgn2$ ./mathlib-test -s
  x            arcSin           Library        Difference
  -            ------           -------        ----------
 -0.9000      -1.11976951      -1.11976951     -0.0000000003
 -0.8000      -0.92729522      -0.92729522     -0.0000000001
 -0.7000      -0.77539750      -0.77539750     -0.0000000001
 -0.6000      -0.64350111      -0.64350111     -0.0000000000
 -0.5000      -0.52359878      -0.52359878     -0.0000000000
 -0.4000      -0.41151685      -0.41151685     -0.0000000000
 -0.3000      -0.30469265      -0.30469265     -0.0000000000
 -0.2000      -0.20135792      -0.20135792     -0.0000000000
 -0.1000      -0.10016742      -0.10016742     -0.0000000000
 -0.0000      -0.00000000      -0.00000000      0.0000000000
  0.1000       0.10016742       0.10016742      0.0000000000
  0.2000       0.20135792       0.20135792      0.0000000000
  0.3000       0.30469265       0.30469265      0.0000000000
  0.4000       0.41151685       0.41151685      0.0000000000
  0.5000       0.52359878       0.52359878      0.0000000000
  0.6000       0.64350111       0.64350111      0.0000000000
  0.7000       0.77539750       0.77539750      0.0000000001
  0.8000       0.92729522       0.92729522      0.0000000001
  0.9000       1.11976951       1.11976951      0.0000000003
^Z
[3]+  Stopped                 ./mathlib-test -s
```

When I first hit this error, I found it rather odd since my values were fairly accurate before they hit 1. So I decided to try some other values that were closer to -1,1 such as -0.9999. However when I tried it, it resulted in an infinite loop just like with -1. Then I tried -0.999, this resulted in the output taking about a quarter of a second to load before the output was printed. Also, the output at -0.999 was still rather inaccurate.

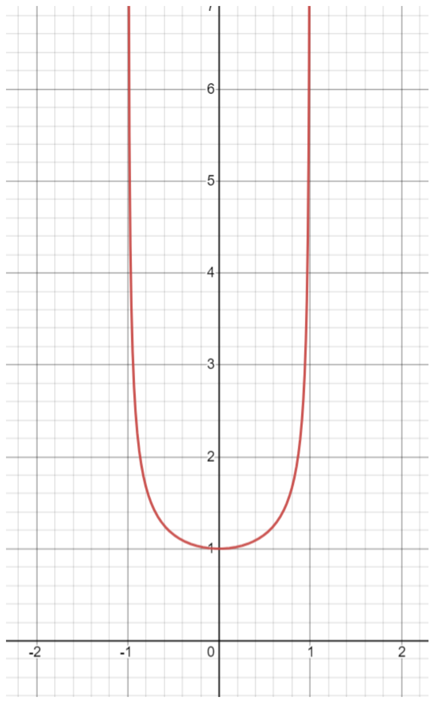


```
  x            arcSin           Library        Difference
  -            ------           -------        ----------
 -0.9990      -1.52607120      -1.52607124     -0.0000000433
 -0.8990      -1.11748076      -1.11748076     -0.0000000003
 -0.7990      -0.92563040      -0.92563040     -0.0000000002
 -0.6990      -0.77399818      -0.77399818     -0.0000000000
 -0.5990      -0.64225169      -0.64225169     -0.0000000000
 -0.4990      -0.52244446      -0.52244446     -0.0000000000
 -0.3990      -0.41042602      -0.41042602     -0.0000000000
 -0.2990      -0.30364454      -0.30364454     -0.0000000000
 -0.1990      -0.20033741      -0.20033741     -0.0000000000
 -0.0990      -0.09916243      -0.09916243     -0.0000000000
  0.0010       0.00100000       0.00100000      0.0000000000
  0.1010       0.10117251       0.10117251      0.0000000000
  0.2010       0.20237865       0.20237865      0.0000000000
  0.3010       0.30574111       0.30574111      0.0000000000
  0.4010       0.41260820       0.41260820      0.0000000000
  0.5010       0.52475386       0.52475386      0.0000000000
  0.6010       0.64475170       0.64475170      0.0000000000
  0.7010       0.77679874       0.77679874      0.0000000001
  0.8010       0.92896374       0.92896374      0.0000000001
  0.9010       1.12206913       1.12206913      0.0000000004
```

The reason for arcSin being so inaccurate at -1 and 1, is because of how long it takes to converge at those values.
ArcSin:



Now on the regular arcsin graph it looks like the value of 1 and -1 map perfectly onto -pi/2 and pi/2 respectively. However, by plotting the derivative of arcsin…
Derivative of ArcSin:

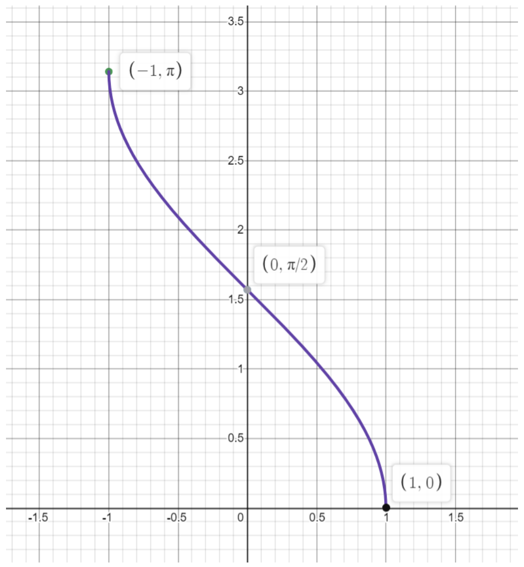


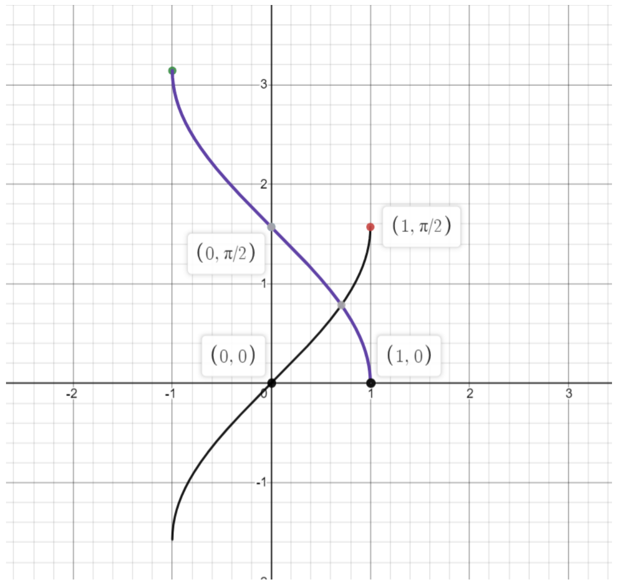
We can see that there are vertical asymptotes at -1 and 1. This means that while arcsin converges at 1, it doesn't converge fast enough to give an accurate approximation using only so many terms greater than epsilon.
Bottom line: Taylor series alone can't be used to accurately calculate the extrema of arcsin or arccos.

However, in a post on Piazza, the professor gave a very useful trig identity which I found extremely helpful in this situation:

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

Looking at the arccos graph:



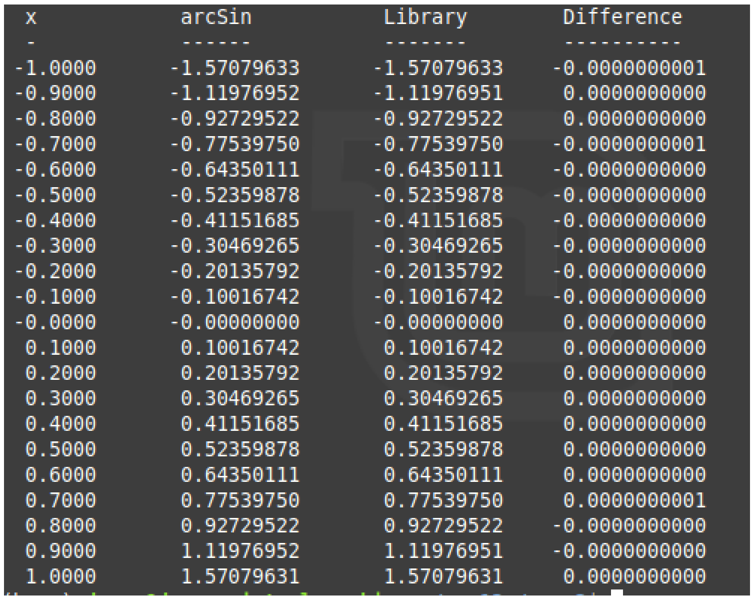


We can see that it’s much easier to solve for pi/2 using arccos since it converges at 0. And, the graph on the right proves this identity since cos is just -arccos added to pi/2. Therefore, by plugging in the arcsin function’s input of x into the arccos function of sqrt(1-x^2), the input for -1 or 1 into the arccos function will be 0. Since arccos(0) converges so well, all we have to do after is convert the arccos output into an arcsin output by using the identity:

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

Ultimately, we can obtain much more accurate numbers at the extrema of arcsin [-1, -0.7] [0.7, 1], because of how quickly those values converge at or near 0 when put into arccos using sqrt(1-x^2).

Once I implemented this into my code, I had much more accurate values.



```
 x            arcSin          Library        Difference
 -            ------          -------        ----------
-1.0000      -1.57079633     -1.57079633    -0.0000000001
-0.9000      -1.11976952     -1.11976951     0.0000000000
-0.8000      -0.92729522     -0.92729522     0.0000000000
-0.7000      -0.77539750     -0.77539750    -0.0000000001
-0.6000      -0.64350111     -0.64350111    -0.0000000000
-0.5000      -0.52359878     -0.52359878    -0.0000000000
-0.4000      -0.41151685     -0.41151685    -0.0000000000
-0.3000      -0.30469265     -0.30469265    -0.0000000000
-0.2000      -0.20135792     -0.20135792    -0.0000000000
-0.1000      -0.10016742     -0.10016742    -0.0000000000
-0.0000      -0.00000000     -0.00000000     0.0000000000
 0.1000       0.10016742      0.10016742     0.0000000000
 0.2000       0.20135792      0.20135792     0.0000000000
 0.3000       0.30469265      0.30469265     0.0000000000
 0.4000       0.41151685      0.41151685     0.0000000000
 0.5000       0.52359878      0.52359878     0.0000000000
 0.6000       0.64350111      0.64350111     0.0000000000
 0.7000       0.77539750      0.77539750     0.0000000001
 0.8000       0.92729522      0.92729522    -0.0000000000
 0.9000       1.11976952      1.11976951    -0.0000000000
 1.0000       1.57079631      1.57079631     0.0000000000
```

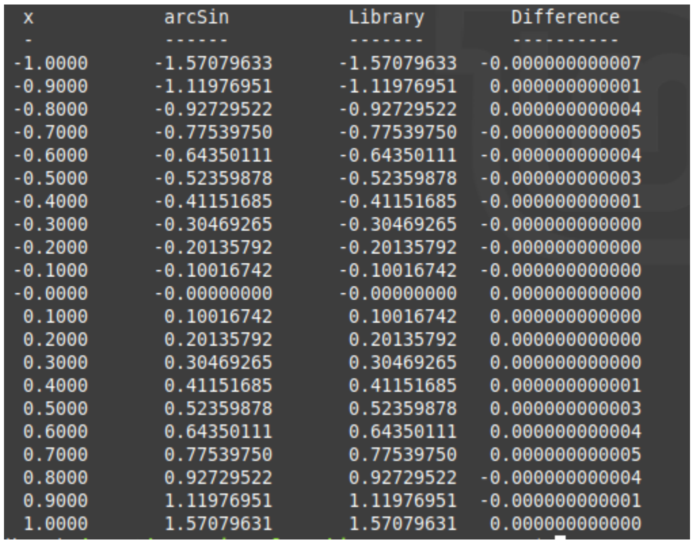
And this was true for arctan and arccos as well since I used the following trig identities to implement those functions:

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

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

Therefore, the accuracy of these functions depends on how accurate my arcsin function is. However, just because my functions are more accurate doesn't mean that they are the same as that of the math.h library.

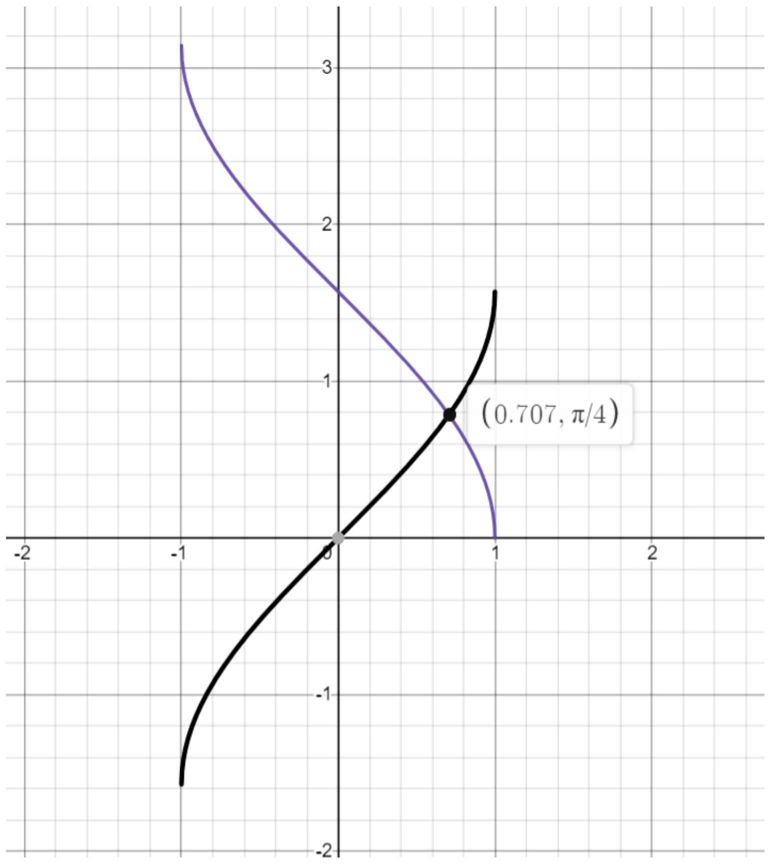
The obvious solution to this problem would be to simply compute more terms by making the value of epsilon smaller:



```
 x            arcSin          Library        Difference
 -            ------          -------        ----------
-1.0000      -1.57079633     -1.57079633   -0.000000000007
-0.9000      -1.11976951     -1.11976951    0.000000000001
-0.8000      -0.92729522     -0.92729522    0.000000000004
-0.7000      -0.77539750     -0.77539750   -0.000000000005
-0.6000      -0.64350111     -0.64350111   -0.000000000004
-0.5000      -0.52359878     -0.52359878   -0.000000000003
-0.4000      -0.41151685     -0.41151685   -0.000000000001
-0.3000      -0.30469265     -0.30469265   -0.000000000000
-0.2000      -0.20135792     -0.20135792   -0.000000000000
-0.1000      -0.10016742     -0.10016742   -0.000000000000
-0.0000      -0.00000000     -0.00000000    0.000000000000
 0.1000       0.10016742      0.10016742    0.000000000000
 0.2000       0.20135792      0.20135792    0.000000000000
 0.3000       0.30469265      0.30469265    0.000000000000
 0.4000       0.41151685      0.41151685    0.000000000001
 0.5000       0.52359878      0.52359878    0.000000000003
 0.6000       0.64350111      0.64350111    0.000000000004
 0.7000       0.77539750      0.77539750    0.000000000005
 0.8000       0.92729522      0.92729522   -0.000000000004
 0.9000       1.11976951      1.11976951   -0.000000000001
 1.0000       1.57079631      1.57079631    0.000000000000
```

Here, I made the value 1*10^-11 instead of 1*10^-10 and the differences got significantly smaller. However this still doesn't address why -0.7 and 0.7 have higher differences.

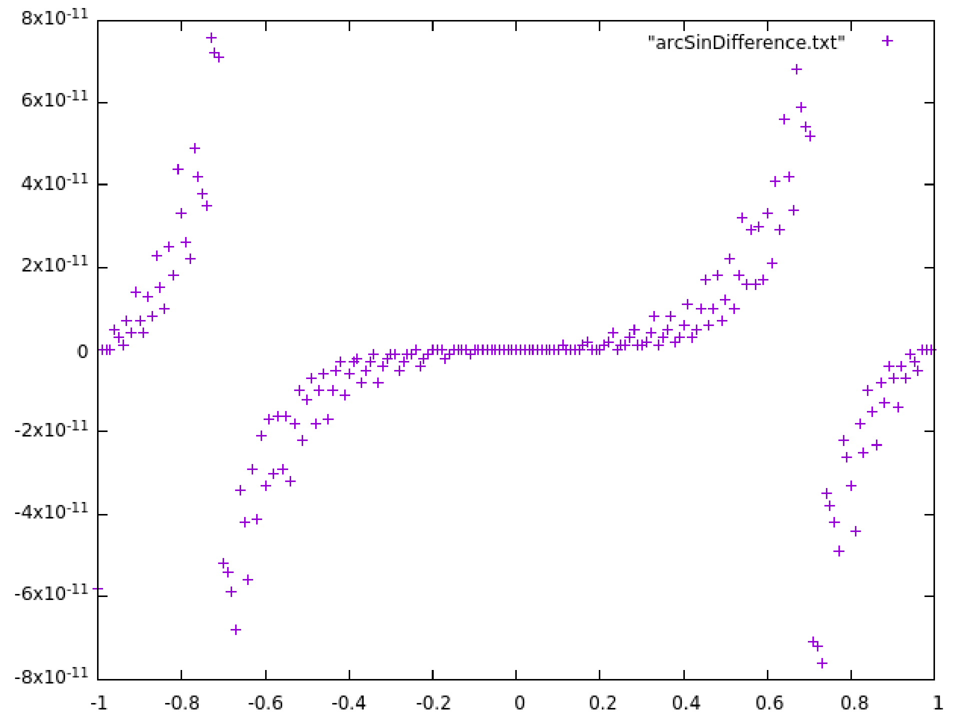
If we go back to the graph of arccos and arcsin we can see that the two graphs intersect at the point (0.707, pi/4):



We know from the arcsin function that the taylor series is most accurate at 0, because of it being centered at 0. Since we use the arccos function to plug in values of arcsin for values from [-1, 0.7] [0.7, 1], which go into arccos as [-0.7, 0.7], it makes sense that 0.7 would be the least accurate point because it is the most distant value from 0 that is calculated for either arcsin or arcsin using the arccos identity:

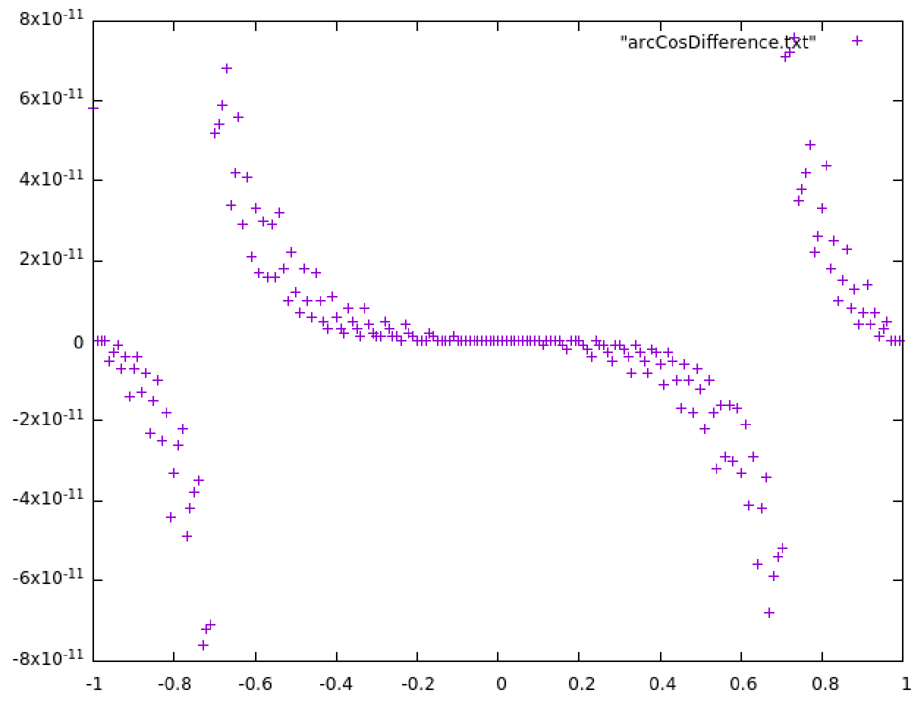
$$\frac{\pi}{2} - \arcsin\left(\sqrt{1-x^2}\right) = \arcsin(x)$$

This can be better portrayed through the graph of the difference of my arcSin function and the asin function from the math.h library:

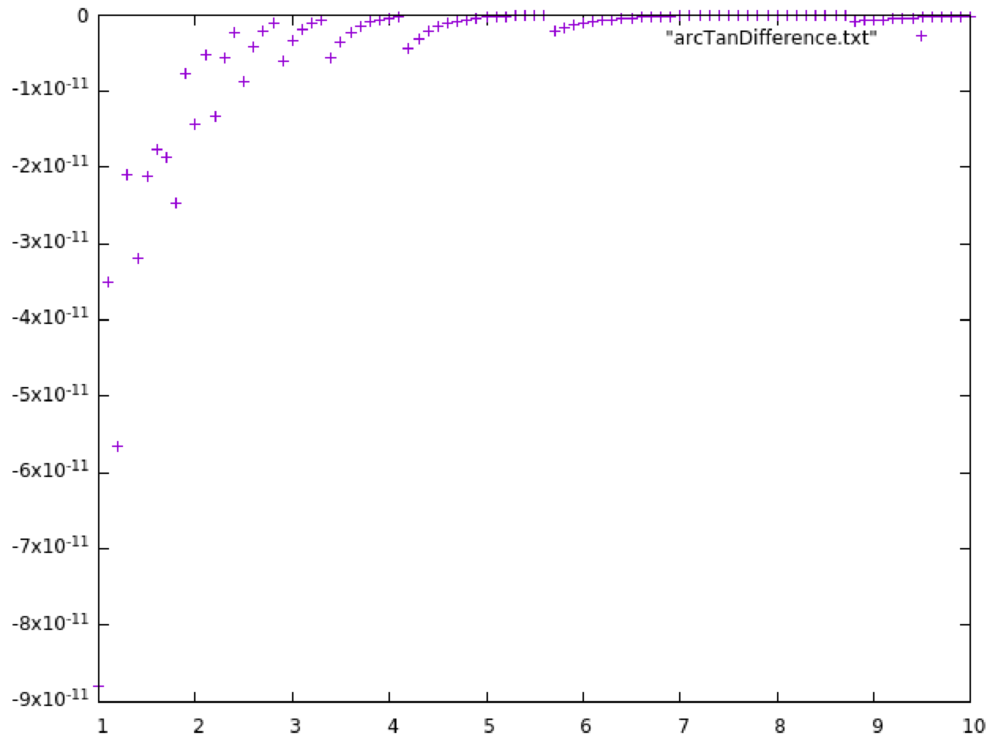


As we can see, the center curve is representative of the arcsin function being used on its own. Whereas the two curves on the sides represent the arccos identity being used for arcsin. The two curves meet at around -0.7 and 0.7, where the graph has the least accuracy.

The same can be seen with the arccos difference graph since my arccos function calls my arcsin function using the identity:

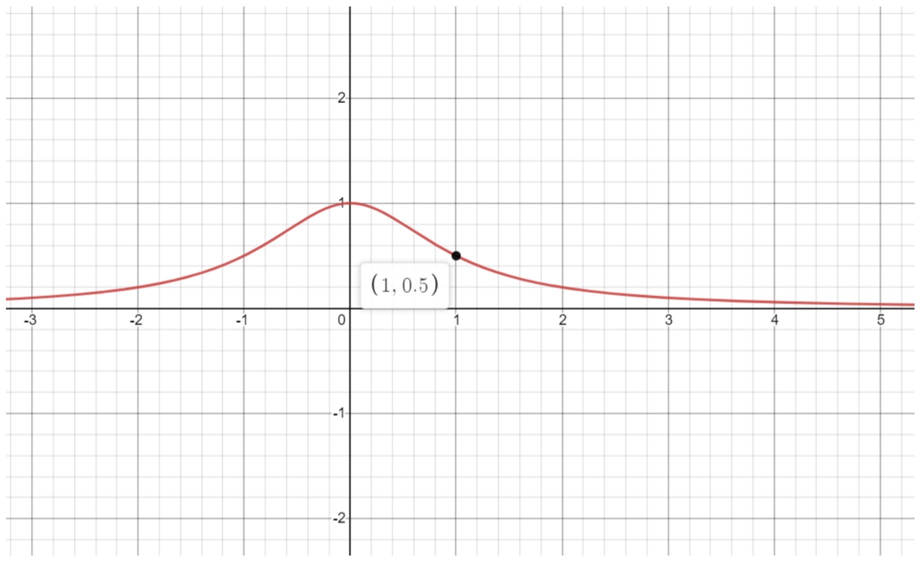


The arctan graph is interesting because the differences are much smaller compared to the arcsin and arccos graphs:

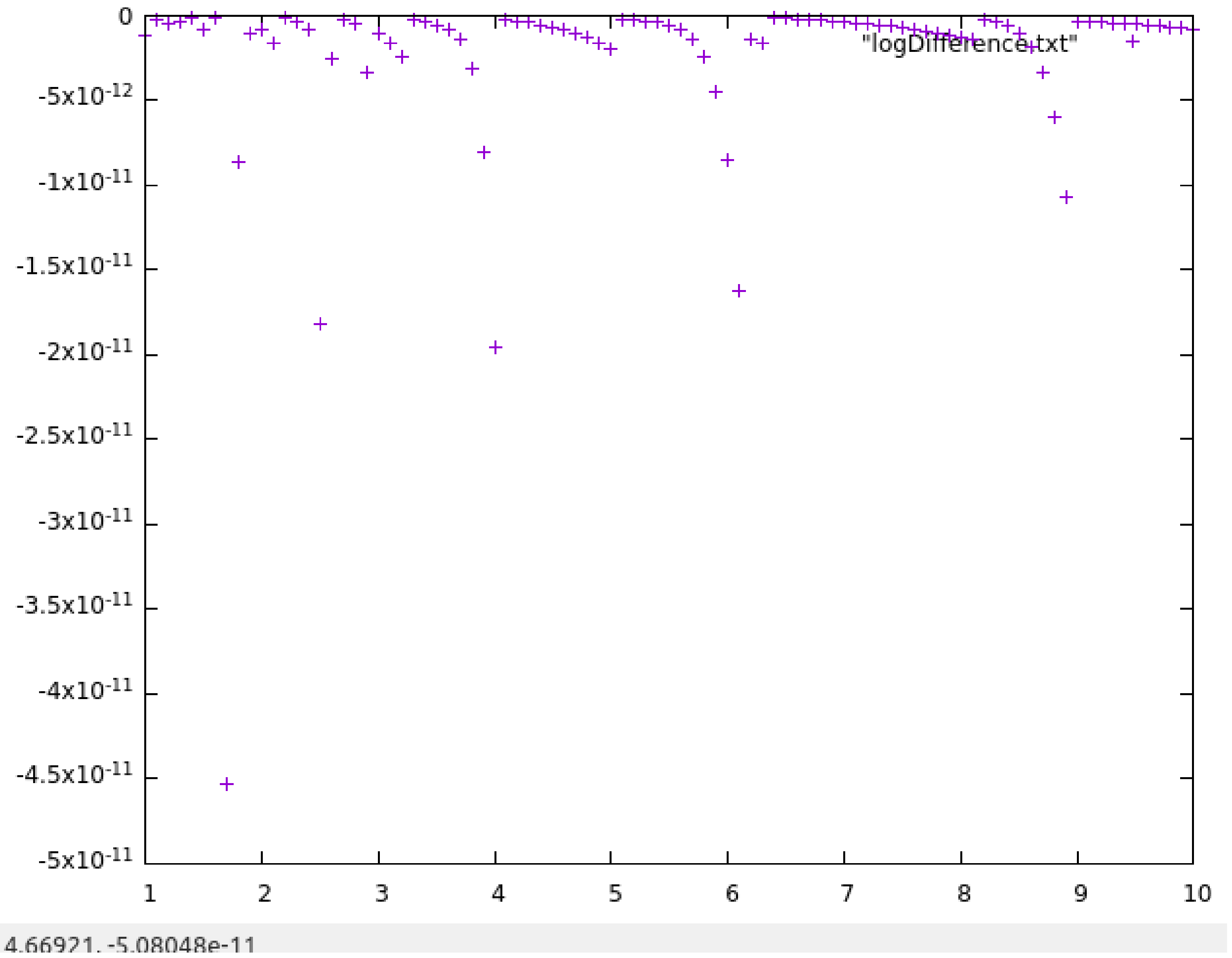


However, we can see that the differences increase in intervals with the integer values being more accurate and the differences increasing as arctan approaches 1. This again has to do with how arctan doesn't converge as fast at x = 1. If we look at the derivative of arctan we can see how the rate of change is much greater at x = 1 than at the other points, so therefore, arctan doesn't converge as quickly there meaning that we would have to compute more values for better accuracy.

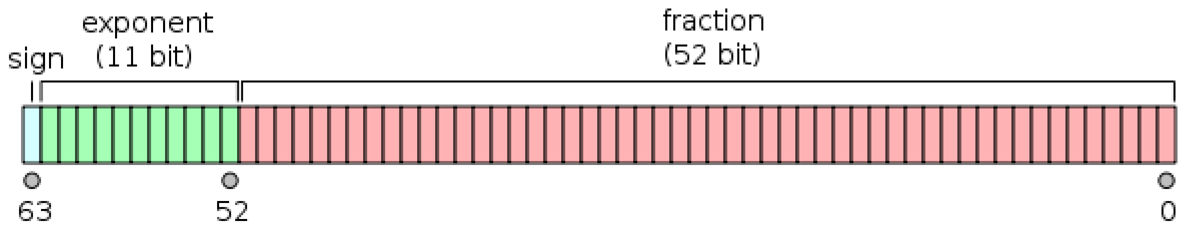
Derivative of arctan:



A somewhat similar case happens in the log function as the differences are relatively low when closer to the whole integer values, and then there is a rapid loss in accuracy as the fractional part gets bigger.



I think this has to do with how computers can't represent floating point numbers exactly because of the binary representation of the numbers. A double in C is a 64-bit IEEE 754 double precision floating point number:



Each decimal place can only represent a decimal of $1/(2^x)$. Therefore, a decimal like 0.1, can't be represented with complete accuracy. The computer can come up with a decimal made up of $1/(2^x)$ that will come close enough in most cases, but when comparing values for accuracy, there will always be some difference since you can calculate an infinite amount of decimal places to make it more accurate but you will never have a value that is exactly accurate unless if it is a multiple of 2^x. Since the values for both arctan and log are within the epsilon value of 1*10^-10, it’s likely that the math.h library computes more terms to make it as accurate as possible which is why there is still a difference. However, not even the math.h library is completely accurate, because, as mentioned before, you can always calculate more terms to get even better accuracy.