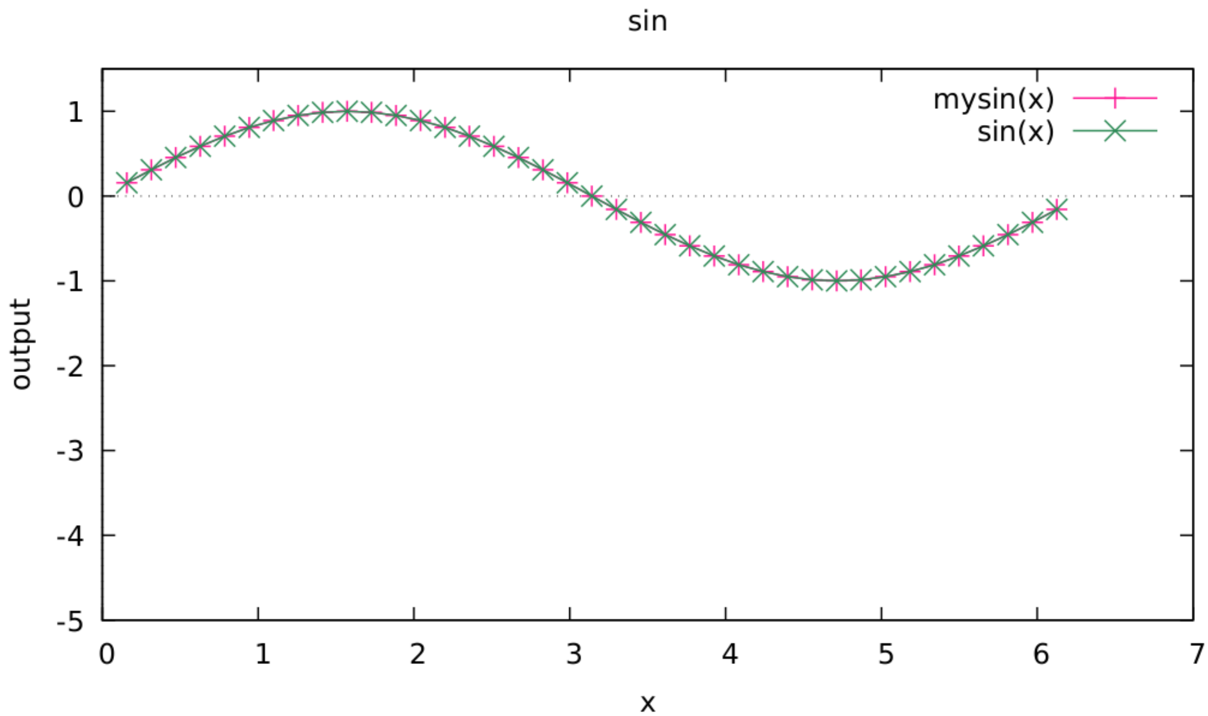
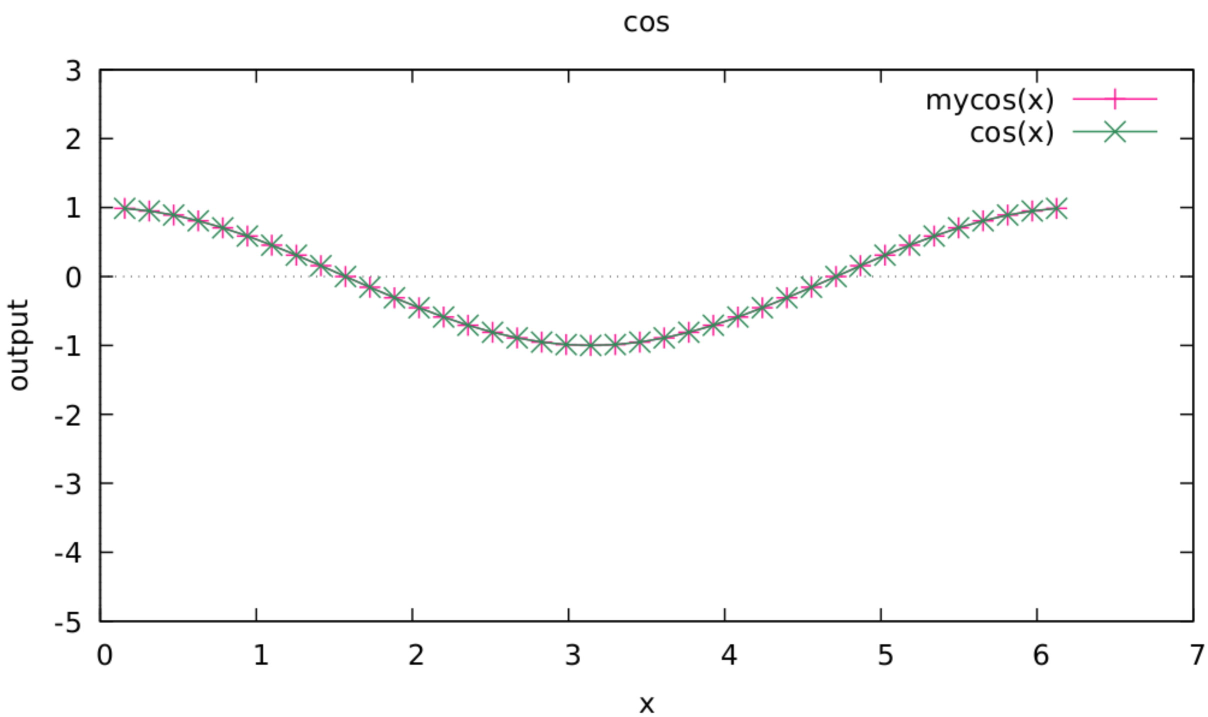


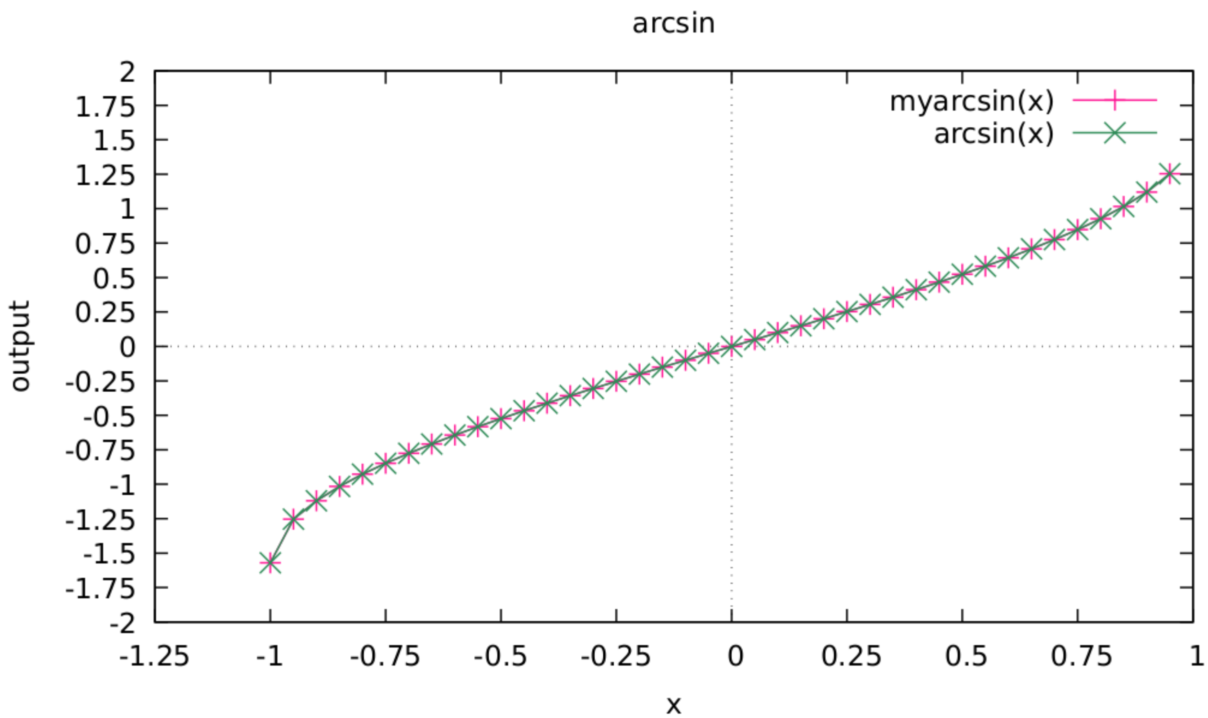
For sin:

As the loop goes through my function and the library's function, the difference seems to either be ±0.000000000000, ±0.000000000001, or ±0.000000000002. This could mean my function is fairly accurate in procuring the sin value of *x*. The possible reason for why it is not fully the same is because there could be round-off errors or truncation errors. Round-off errors are when computers cannot actually represent some numbers exactly. This can happen with floating points. Truncation errors can happen when an approximation is used instead of the exact procedure in mathematics. Since we use a for-loop for sine, and there is a limit to the values we can calculate, we can assume there is some form of truncation happening. It is also possible that the difference could be many more numbers beyond the decimal point since we stop when we get close to Epsilon. Maybe if there were more numbers being expressed on-screen, then we'd see more digits of nonzero numbers. In my graph, mysin(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the sin(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. My difference is so miniscule that the two lines are indistinguishable.



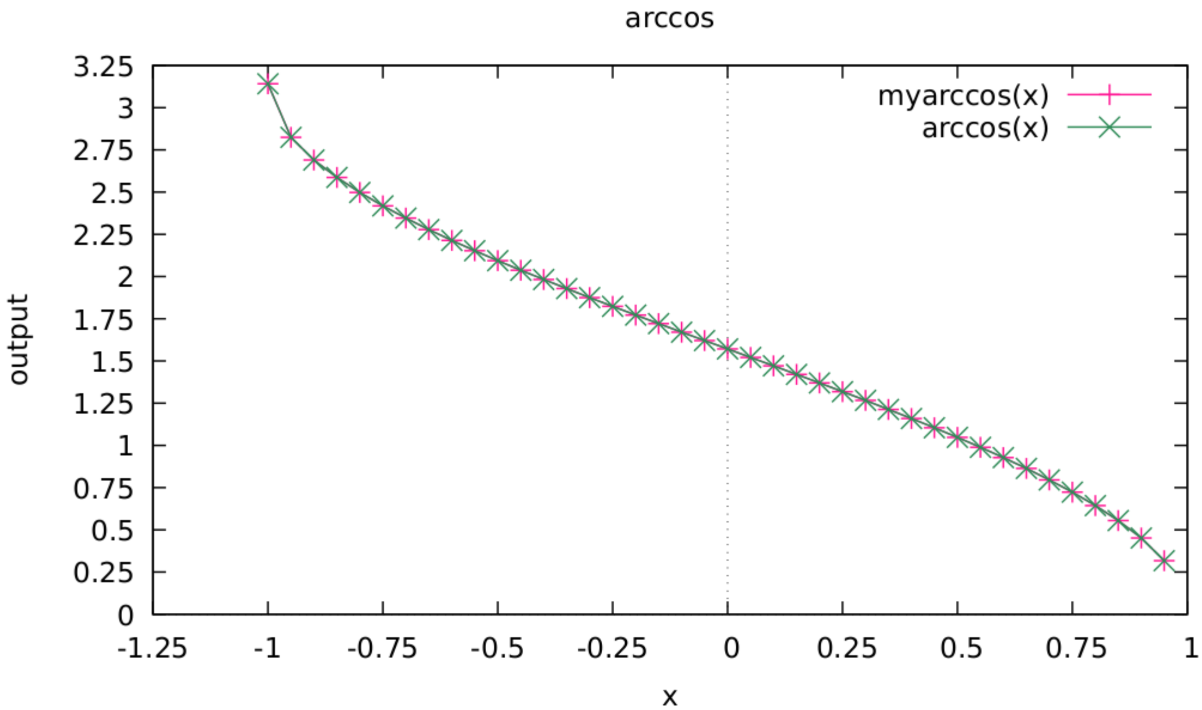
For cosine:

As the loop goes through for my function and the library's function, the difference seems to either be ±0.000000000000, ±0.000000000001, or ±0.000000000002. This could mean my function is also fairly accurate in procuring the cosine value of *x*. The possible reason for why it is not fully the same is because there could be, as I've said above, that there are round-off errors or truncation errors. The same concerns of mine for cosine are the same as above from sine's differences, since they both use a for-loop and the same formula. In my graph, mycos(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the cos(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. Once again, my difference is so miniscule that the two lines are indistinguishable.



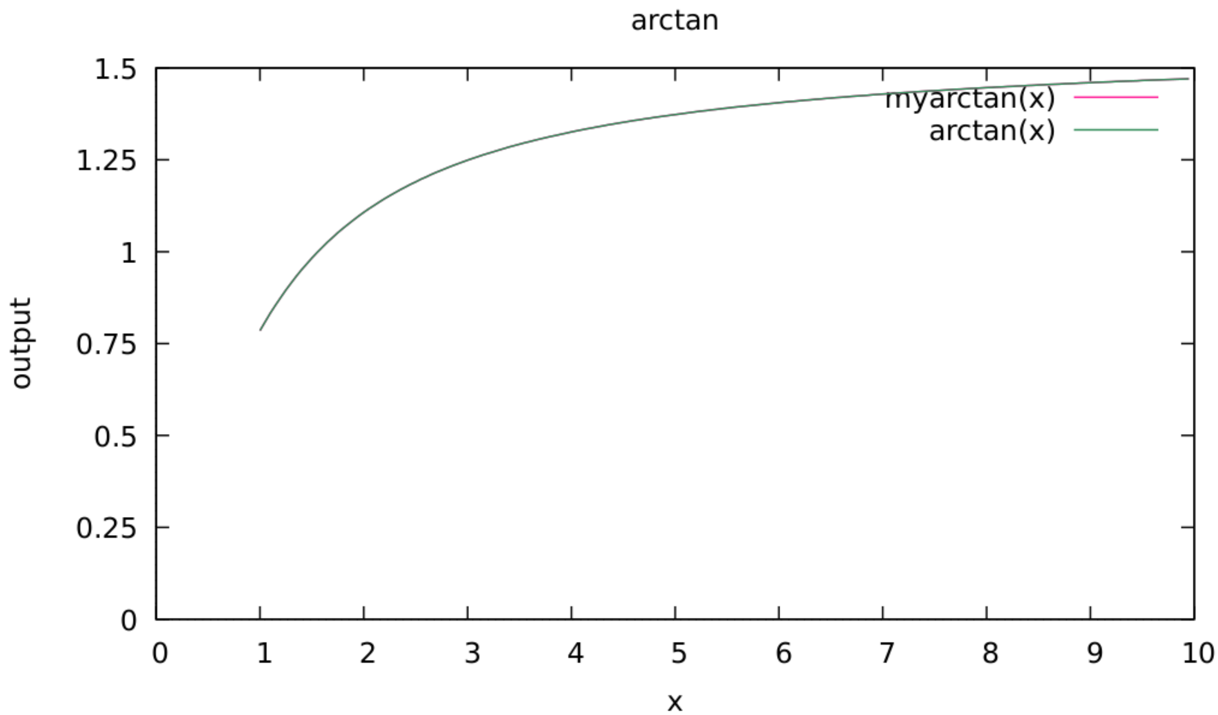
For arcsin:

As the loop goes through for my function and the library's function, the difference seems to be consistent except for the very first time going through the logic. The very first difference is 0.000000294697. After that, it's all ±0.000000000000. I think the reasoning for this is because arcsin starts at negative 1. With a whole negative number, the logic's output is wholly changed. Note that we use our own sine and cosine functions to get the result, so those probably contribute to the difference. I do not use a for-loop, but a do-while loop here. The reasoning is because there is no need for the initialization and step numbers to be implemented into the logic or as the conditional. Since there is a loop in use, we could say that there may be truncation errors. In my graph, myarcsin(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the arcsin(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. Once again, my difference is so miniscule that the two lines are indistinguishable.



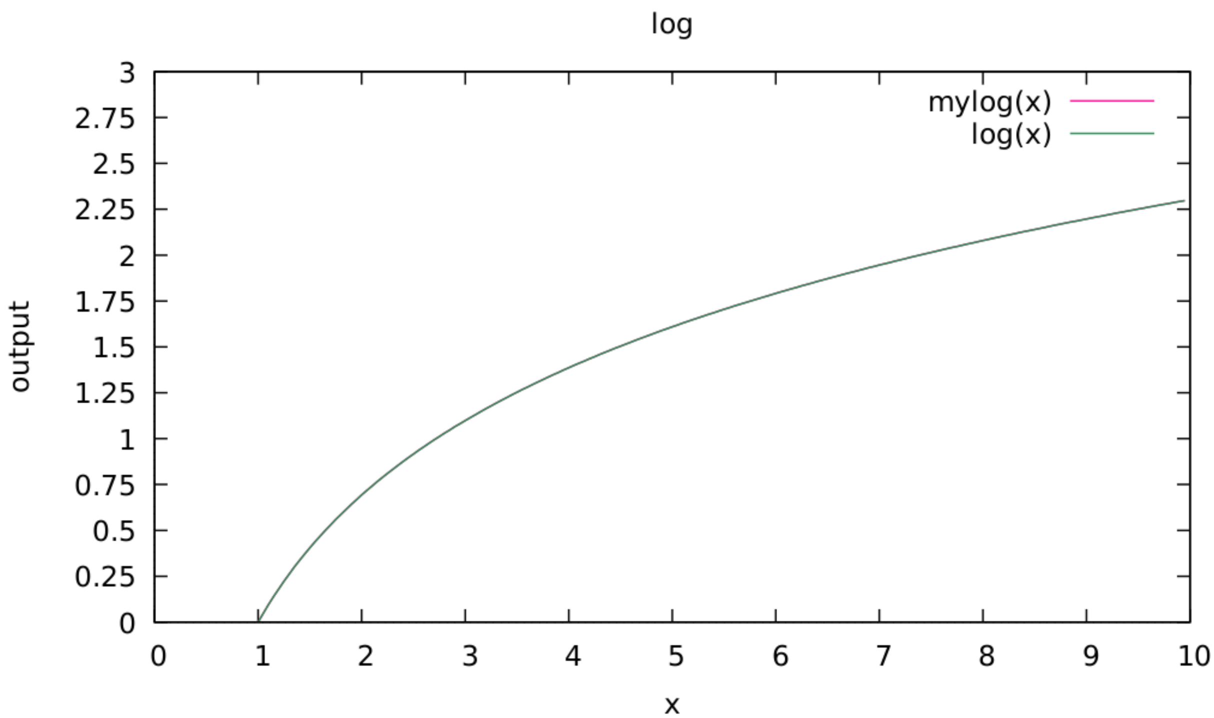
For arccos:

As the loop goes through for my function and the library's function, the difference seems to be consistent except for the very first time going through the logic. The very first difference is -0.000000294697. After that, it's all ±0.000000000000. This is almost the exact same result as my first arcsin difference. Since my arccos function depends on arcsin, that could be the reason why it's the same digits. I also don't use a loop for this function, which would help with other sources of errors. In my graph, myarccos(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the arccos(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. Once again, my difference is so miniscule that the two lines are indistinguishable.



For arctan:

Amazingly, my differences for arctan are only ±0.000000000000. This is quite interesting as a result considering that my function uses my arccos function for the result. However, I do not use a loop, so that limits other sources of error. Perhaps the reasoning for super close exactness is because the *x* value has to go through so many functions, that the remainder of the difference (since only so many numbers are shown on-screen) is very minimal and requires more of the difference to be shown. The idea behind this reasoning is because as it goes through each function, it would decrease the amount of incorrect numbers in the output. In my graph, myarctan(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the arctan(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. Once again, my difference is so miniscule that the two lines are indistinguishable.



For log:
My log function has the most variety in differences. They range, but never past reach 10^{-10} for the differences. There are still 0.000000000000 differences in the column. Perhaps the reason for the variety in difference in my log functions stem from the exponential function that we had to use. There are undeniably some round-off errors when this happens. This plus the logic also being a do-while loop can create more errors, such as truncation errors. In my graph, mylog(x) has its x-coordinates as the numbers being put into my function, and the y-coordinates is the output. For the log(x), it has its x-coordinates as the numbers being put into the math library's function, and the y-coordinates is the output. Once again, my difference is so miniscule that the two lines are indistinguishable.

Resources that I used for this assignment:

I'd like to cite Professor Long's absolute value function that he provided from Piazza. I used this for my assignment.

The square root function that we can use was shared to us in Piazza as well.
https://piazza.com/class/l8ahj4fji3i4om/post/150

I also used this to figure out how to incorporate Boolean in C.
https://www.javatpoint.com/c-boolean

This website was used to help me understand more about errors in my assignment.
https://web.mat.bham.ac.uk/R.W.Kaye/numerics/errors.html