$\sin^{-1}$ arcSin()

domain [-1, 1)
step: 0.1

20 values to calculate

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

k=1 k=2 k=3

$$\arcsin(x) = x + \left(\frac{1}{2}\right)\frac{x^3}{3} + \left(\frac{1}{2}\cdot\frac{3}{4}\right)\frac{x^5}{5} + \left(\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\right)\frac{x^7}{7}$$

k=0 = k=1

$$\frac{(2k)!}{2^{2k}(k!)^2} \rightarrow \cdot 2$$

k=4

$$\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{7}{8}\left(\frac{x^9}{9}\right)$$

$\cos^{-1}$ arcCos()

domain [-1, 1)
step: 0.1

20 values to calculate

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

$\tan^{-1}$ arctan()

domain [1, 10)
step: 0.1

100 values to calculate

$$\arctan() = \arcsin\left(\frac{x}{\sqrt{x^2+1}}\right) = \arccos\left(\frac{1}{\sqrt{x^2+1}}\right), \quad x > 0 \quad \text{or}$$

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

Log()

domain [1, 10)
step: 0.1

y is user input, ie: log(5), y = 5

X_k start with 1, end at

$$X_{k+1} = X_k + \frac{y - e^{X_k}}{e^{X_k}}$$

$\sin^{-1}$ arcSin(x)

Test values from -1 to 1, with step 0.1

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

LEFT

$$\sin^{-1}(x) = x + \left(\frac{1}{2}\right)\frac{x^3}{3} + \left(\frac{1}{2}\cdot\frac{3}{4}\right)\frac{x^5}{5} \ldots$$

RIGHT

If the code uses this style of taylor series, you would need to calculate the factorials from the ground up after each iteration. (INEFFICIENT)

WHERE AS...

This simplified version clearly shows a repeating pattern. The left sides new value's numerator is the current iteration number, and denominator is 2x the previous denominator OR 2x the iteration #. The right side's numerator grows by "x*x" after each iteration. the denominator is plus 2 from previous denominator.

VS

With each iteration, the code needs to recompute all the values such as, $(2k)!$ and (2^{2k})

With each iteration, values do NOT need to be calculated from the ground up. Meaning, more efficient.

Using this simplified version of arcSin is significantly faster.

(I tried the other method, and) NO GOOD

PsuedoCode

arcSin: Declare the changing variables after each iteration

Declare finalans, Declare numerator & denominator for left, and for Right

Declare counter to keep track of iteration,

Declare Testans to remember answer from previous iteration,

and Multiply X by X*X because the loop starts from 1.

Loops

Leftside = leftside * $\left(\frac{\text{Left num}}{\text{Right denom}}\right)$

Finalans = Finalans + (left * (X/Xdenominator))

Counter += 1

X = X*X*X

Left num +2 and Left denom +2

Test Values from $[-1, 1)$, step: 0.1

$\cos^{-1}$ arcCos(x)

$$\text{arcCos}(x) = \frac{\pi}{2} - \text{arcSin}(x)$$

In this function, arcCos(x) would pass x to arcSin(·).
Then return $\left(\frac{\pi}{2} - \text{arcSin}(x)\right)$;

NOTE
π will be defined using <Math.h>

---

arcCos Psuedo Code

As explained above, M_PI = π is defined. SO....

return $\left(\frac{\pi}{2} - \text{arcSin}(x)\right)$;

$\tan^{-1}$ arcTan(x) Test Values from 1 to10, step 0.1

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

Because arcCos calls arcSin.
I will be using $\arcsin\left(\frac{x}{\sqrt{x^2+1}}\right)$ for this function.

Step 1. Pass $\left(\frac{x}{\sqrt{x^2+1}}\right)$ to arcSin. Psuedo

└ Compute Square Roots.
square roots computed through Newton's method.

Loop{ //Remember the previous values

test = finalans

test: The previous answer for the loop

$$finalans = \left(test + \frac{(base)}{test}\right)\frac{1}{2}$$

finalans: The current answer for the loop

Then if the difference from finalans and test is less than epsilon, [redacted]. The square root is computed.

Step2. Return the value from Step 1.

Log( ) Test values from 1 to 10, step 0.1

$$X_{k+1} = X_k + \frac{y - e^{X_k}}{e^{X_k}}$$

The exp() function is provided to us. This function finds answers of (e^x).

y = Value to compute log for

X_k = The previous answer from previous iteration. $\left(X_k + \frac{y - e^{X_k}}{e^{X_k}}\right)$

Log(X){ Psuedo Code

declare "pass" for the "X_k"
declare "epow" for ~~the~~ "e^{X_k}"

Loop{
epow = exp(pass) // Pass is used here because pass is "X_k"

$pass = X_k + \frac{y - e^{X_k}}{e^{X_k}}$ // $pass + \frac{X - epow}{epow}$

Test if the difference ~~between~~ between epow and pass is < epsilon.
if so, return pass.

Test harness

- This function acts as a test harness for mathlib.c

-a : to run all test
-s : to run arcsin test
-c : to run arcCos test
-t : to run arcTan test
-l : to run log test.

If arguments are incorrect, print program usage.

Print arguments in order.

-a, -s should only run -a

Pseudo code

// argc will hold # of inputs
// argv is the array of inputs

```
int main(int argc, char *argv[])
```

chars of the possible inputs

Declare a T/F flag for every test option, set to False

```
while((choice = getopt(argc, argv, "asctl")) != -1){
```

```
switch(choice)
```

for every choice, if that choice is chosen, set the respective flag to True.

```
case '?':
    Print program usage
}
```

argv is the array, and optind is the pointer so, if the pointer points the array as NULL, Print Program Usage.

Finally, make 4 if statements. Each if statement checks if the flag for "every test" = True OR the respective test = True.

If so, print the respective X value, respective test, Test from <math.h>, and the difference