mathlib.c

Include math.h for PI constant

Create Prof. Ethan Miller's square_root code() function so we can use it later

Create Prof. Ethan Millers Exp() function so we can use it later

double my_sin(double x)

    Set limit of N to ~20 to give enough accuracy

    Set input of "x" to be within (0,2pi) so the function stays accurate at high values of x Either subtract 2pi or add 2pi to get x within the range

    Set both approximation and current term to x
    //This is to effectively "solve" the first term

    Loop from 1 to N

        //applying sin formula from lab doc
        Term = term * (x/2n) * (x/2n+1)

        //This is effectively (-1)^n but much faster
        If n is odd, total = total - term
        If n is even, total = total + term

    Return term

double my_cos(double x)
    Return -my_sin(x + 3pi/2);
    //Just returning -sin shifted by 3pi/2 as that is the equivalent of cosine

double my_arcsin(double x)

Create z_n and z_n1 for current guess and next guess respectively

//Endpoints act strange since we are taking the derivative on an endpoint which is undefined.
Check if x is -1 or 1, if it is
Return pi/2 or -pi/2 respectively

//Looping until we are extremely close to the actual value
While z_n - z_n1 > 10e^-10

//Making our previously found next guess our current guess
Z_n1 = z_n;

// Applying arcsin formula from lab doc
Z_n = (z_n1) - ((my_sin(z_n1) - x) / cos(z_n1))

Return z_n

double my_arccos(double x)

//Using shortcut from lab doc
Return (pi/2 - my_arcsin(x))

double my_arctan(double x)
//Using lab docs shortcut to get arctan in terms of arcsin
Return my_arcsin(x/(square_root(x*x +1))

double my_log(double x)

Create a_n and a_n1 for current guess and next guess

While a_n - a_n1 > 10e-10

//setting our new guess to current guess
a_n = a_n1

//Using formula from lab doc and Professors Exp function.
a_n1 = a_n + (x-Exp(a_n)/Exp(a_n)

Return a_n1

mathlib-test.c

```
//Basically just loop through the domain and compare my_function to the math library function.
// Do this for all 6 functions
Void test_sin()
    For (0,2pi) step .05pi
        Error = (my_sin - sin)
        Print ( my_sin, sin, Error)

Void test_cos()
    For (0,2pi) step .05pi
        Error = (my_cos - cos)
        Print ( my_cos, cos, Error)

Void test_arcsin() step.05
    For [-1,1)
        Error = (my_arcsin - asin)
        Print (my_arcsin, asin, error)

Void test_arccos() step.05
    For [-1,1)
```

```
        Error = (my_arccos - acos)
        Print (my_arccos, acos, error)




Void test_arctan() step.05
    For [1,10)
        Error = (my_arctan - atan)
        Print (my_arctan, atan, error)


Void test_log() step.05
    For [1,10)
        Error = (my_log - log)
        Print (my_log, log, error)

//Creating command line options to use
#define OPTIONS “ascISCT”

//Creating a flag for every option so printing multiple of the same table wont happen
Int sin_flag = 1
Int cos_flag  = 1
Int asin_flag = 1
Int acos_flag = 1
Int atan_flag = 1
Int log_flag = 1


For every found option in the command line
    Switch
    Case ‘a’
        Turn all flags to 0

    Case ‘s’
        Turn sin_flag to 0

    Case ‘c’
        Turn cos_flag to 0

    Case ‘S’
        Turn asin_flag to 0

    Case ‘C’
```

```
            Turn acos_flag to 0

        Case 'T'
            Turn atan_flag to 0

        Case 'l'
            Turn log_flag to 0


//For every single flag (sin,cos,asin,acos,atan,log)
If flag == 0
        Print out the xxx_test() of the function
…
…
…


//returning 0 because this is the main function
Return 0
```