Kevin Lee
CSE 13S Winter Quarter

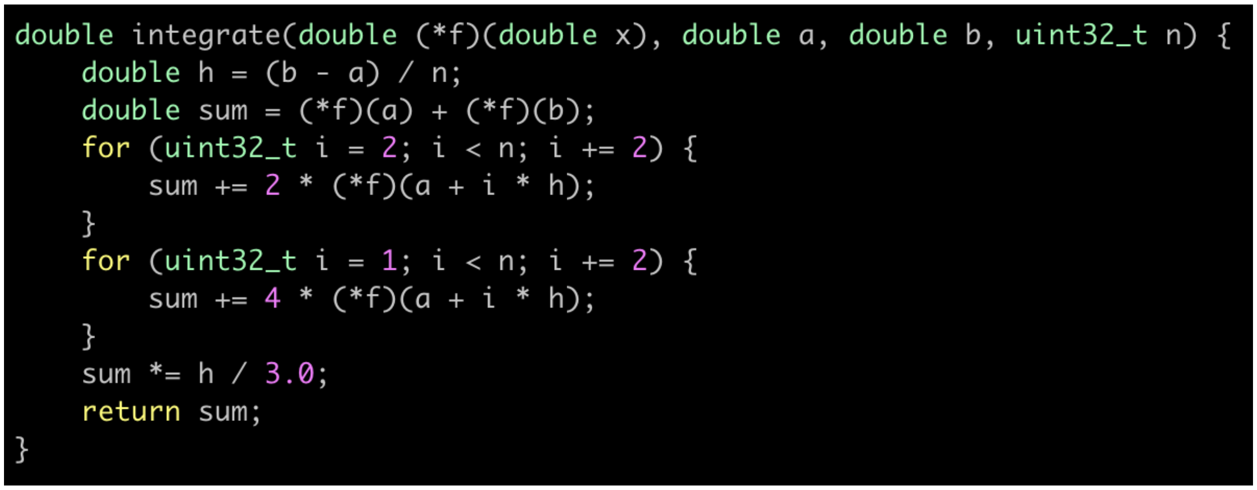
Assignment 2 - Numerical Integration

Introduction:

- This writeup will primarily be focused on numerical integration. It will go over the code I used to produce the numerical integration, along with my own math library. The ten graphs I have produced correspond to the ten functions we were given and go up to 20 partitions.
- A lot of the code for the math library was given in the assignment document. The code for the integration was shown to us by Professor Long at the end of his Friday lecture. An example of the getopt code was provided in the assignment 1 collatz.c, however, Eugene's section did a great job explaining the getopt loop along with header files and the Makefile.

Integration:

- We used the Simpson ⅓ rule for our integration. The formula for this was provided in the assignment 2 document.



```c
double integrate(double (*f)(double x), double a, double b, uint32_t n) {
    double h = (b - a) / n;
    double sum = (*f)(a) + (*f)(b);
    for (uint32_t i = 2; i < n; i += 2) {
        sum += 2 * (*f)(a + i * h);
    }
    for (uint32_t i = 1; i < n; i += 2) {
        sum += 4 * (*f)(a + i * h);
    }
    sum *= h / 3.0;
    return sum;
}
```

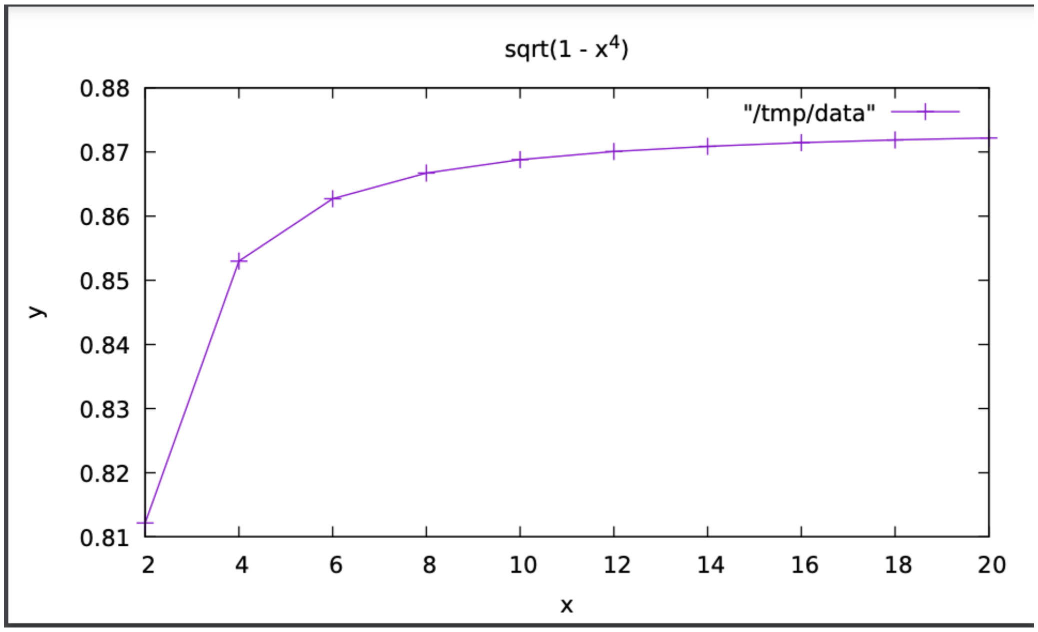
- Here, we see that the integrate function takes four inputs and outputs a double. This is the first time that we had to deal with pointers to functions, as seen with the double (*f)(double x), which we can specify later in our get opt function as to which function to use. The rest of the code mimics the equation we were provided with in the assignment doc for the Simpson ⅓ rule.

Mathlib:

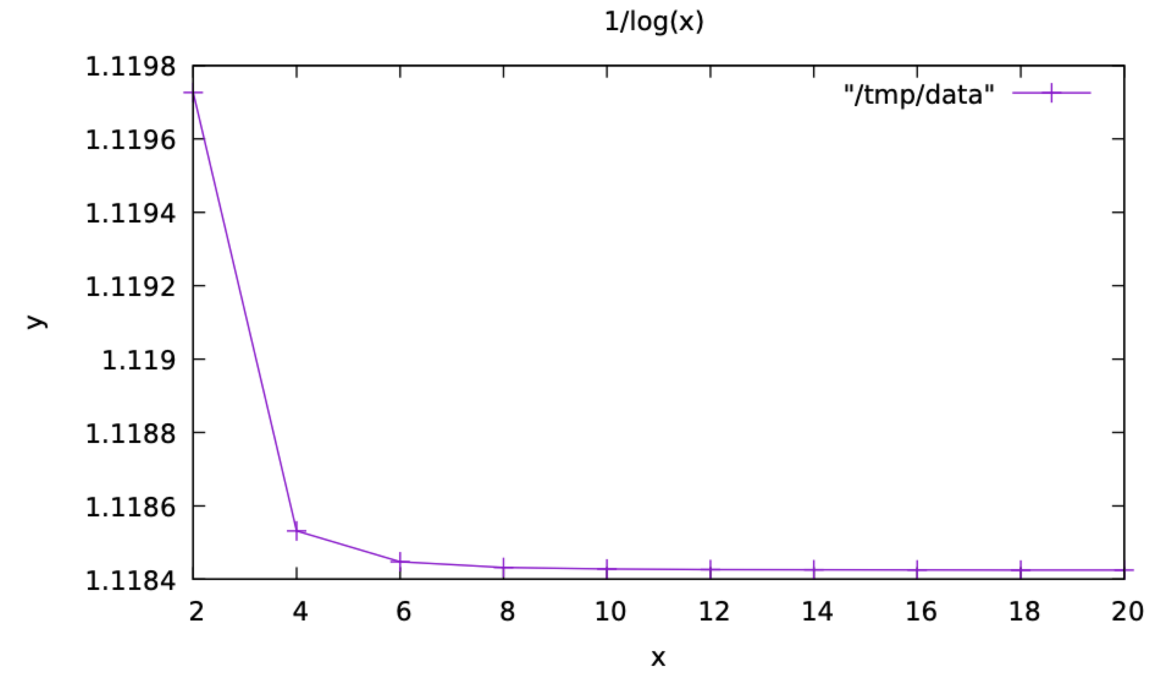
- For the mathlib file, the equations to recreate the mathematical operations were provided in the assignment 2 document. A lot of the equations are predicated on Taylor series approximations.
- One of the trickiest things to wrap my head around was the concept of scaling. For the sin, cos, log, and sqrt require scaling to tackle large inputs.
    - From what I can gather, as the numbers get larger and therefore lose precision, and can't represent all real numbers.
    - Therefore to reduce the error, we can factor out from the function to make the input more manageable.
- For sin and cos, we ended up basically doing a modulo with 2 * π, however, we ended up using while loops, since modulo only works with integers.
- For log, we factored out multiples of e from the input until we were left with an input smaller than e and then added to the final sum the number of times we factored out a multiple of e.
- Lastly, for sqrt, we factored out multiples of 4, and since the square root of 4 is 2, we multiplied to the final sum two to the power of times we factored out a multiple of 4.

Functions:

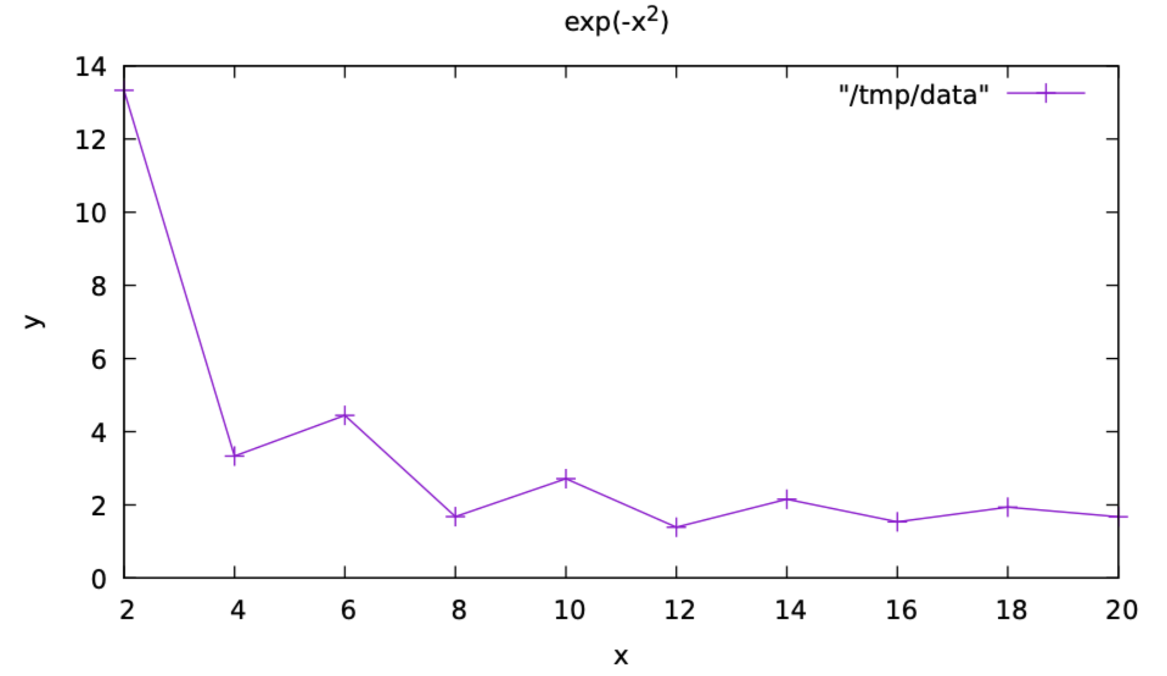
$\sqrt{1 - x^4}$:



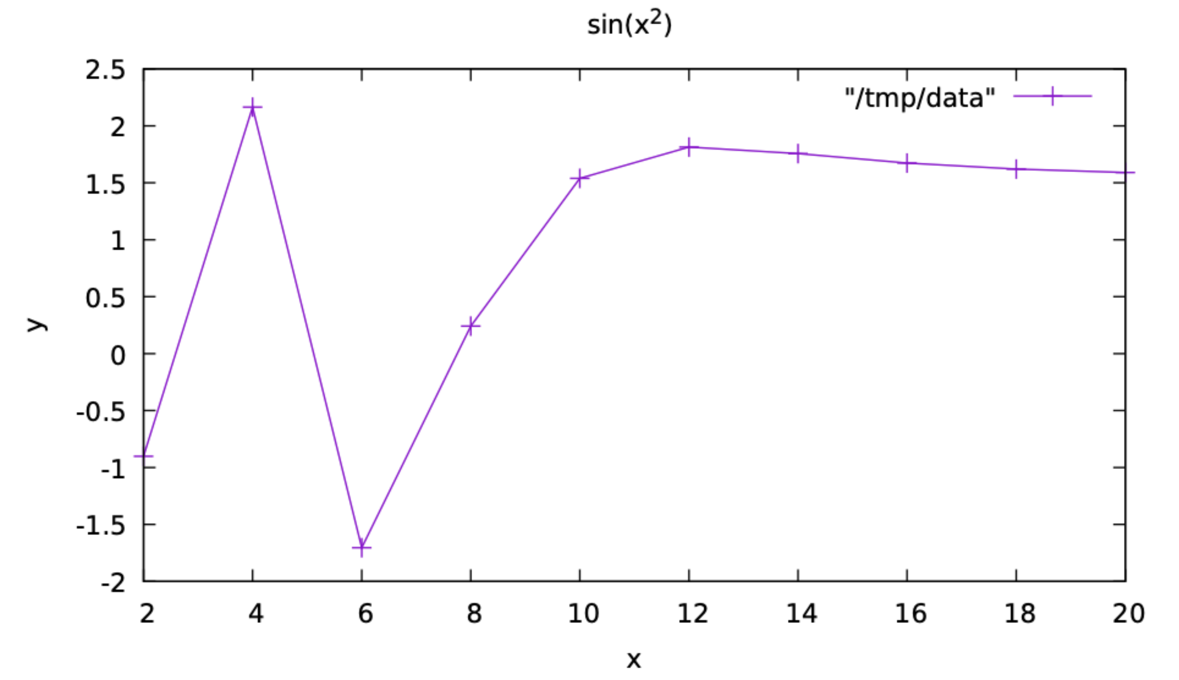
$1/log(x)$:


1/log(x)
"/tmp/data"
y
x
1.1198
1.1196
1.1194
1.1192
1.119
1.1188
1.1186
1.1184
2
4
6
8
10
12
14
16
18
20

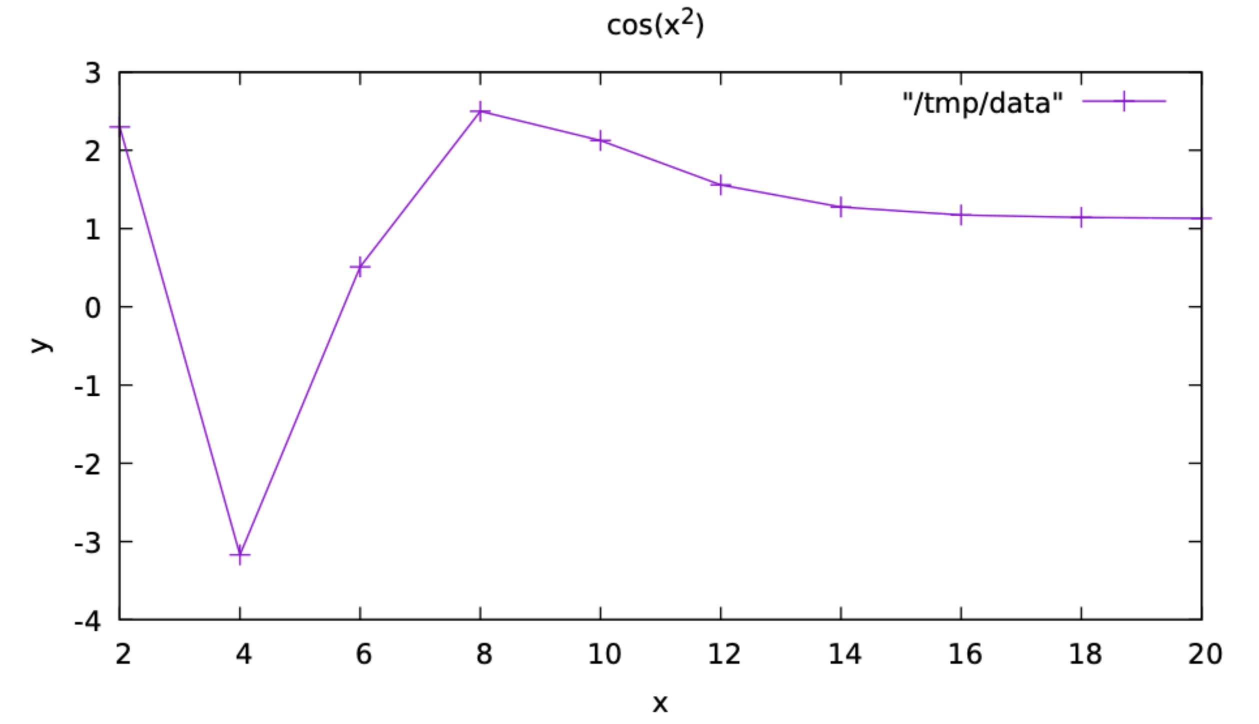
e^{-x^2}:


exp(-x2)
"/tmp/data"
y
x
14
12
10
8
6
4
2
0
2
4
6
8
10
12
14
16
18
20

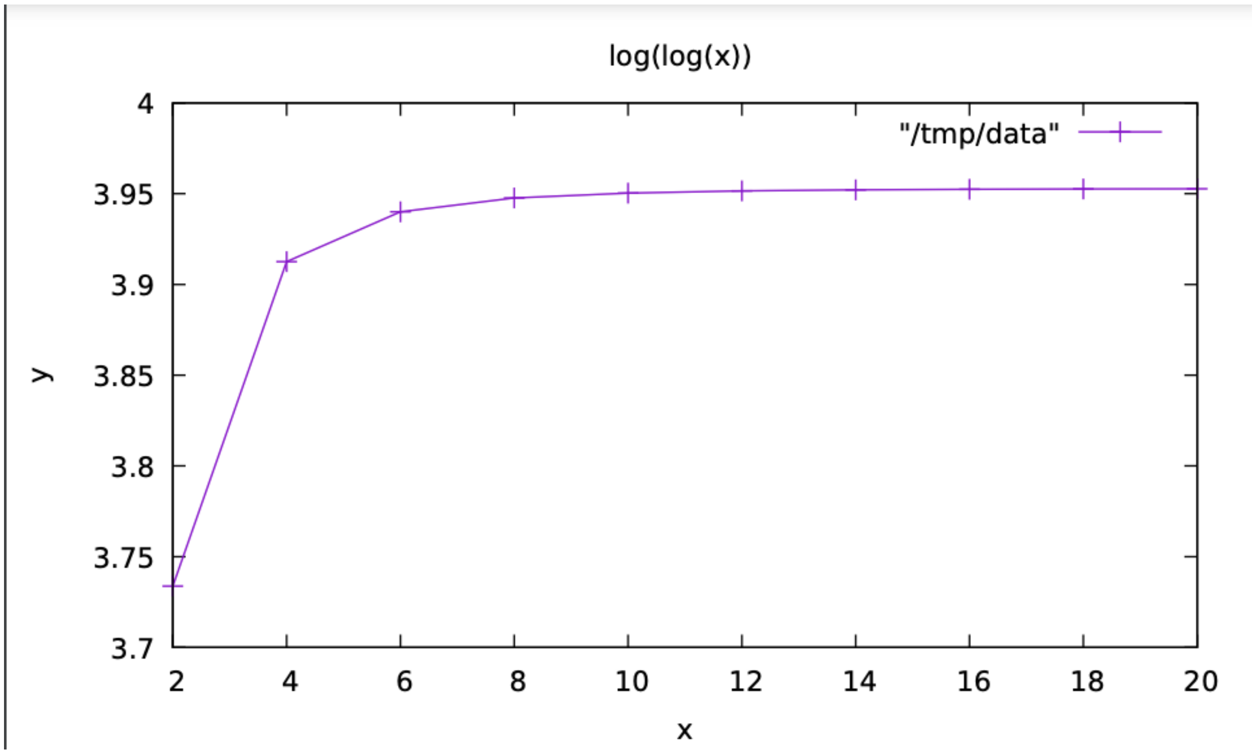
$sin(x^2)$:


sin(x2)
"/tmp/data"
y
x
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
2
4
6
8
10
12
14
16
18
20

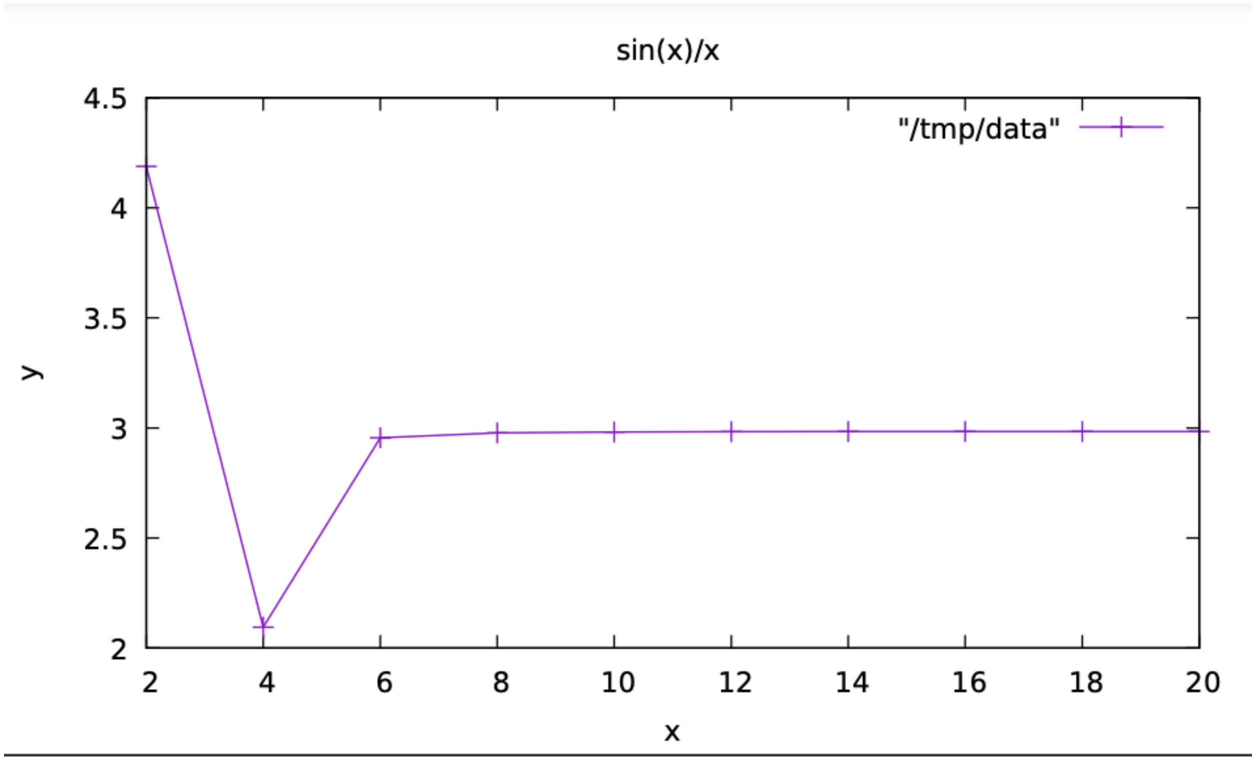
$cos(x^2)$:


cos(x2)
"/tmp/data"
y
x
3
2
1
0
-1
-2
-3
-4
2
4
6
8
10
12
14
16
18
20

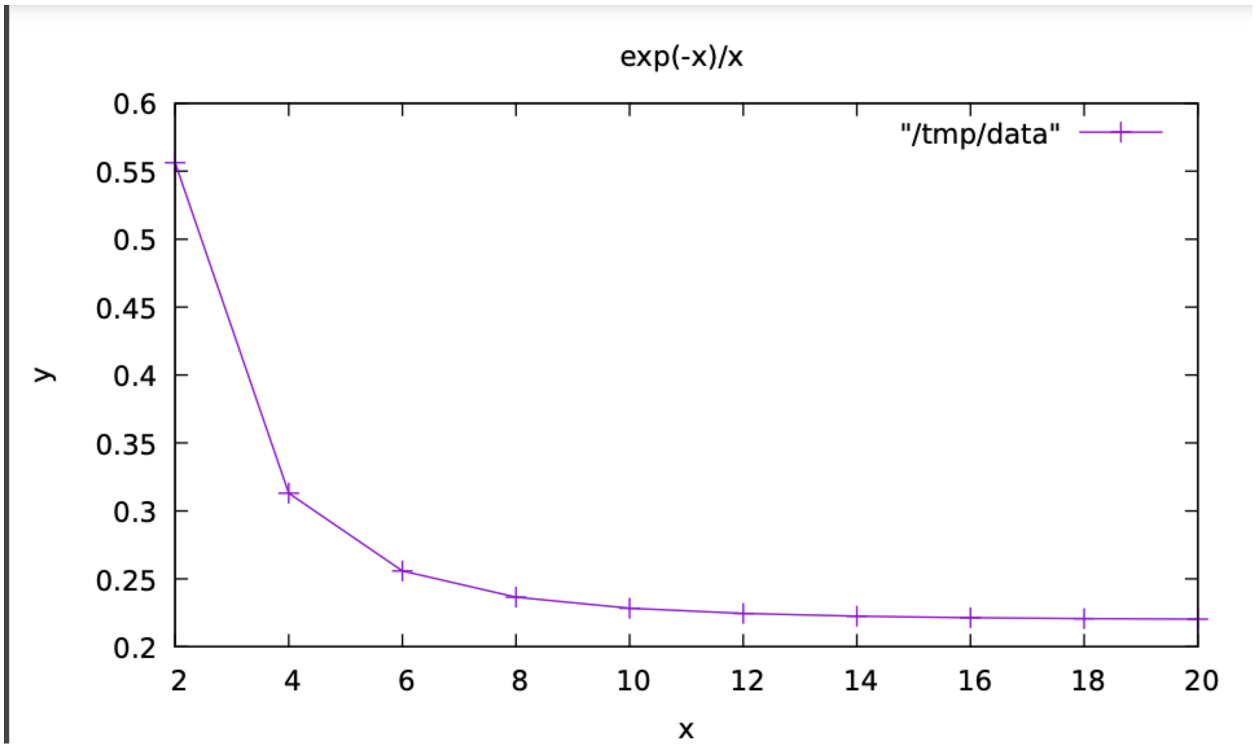
$log(log(x))$:



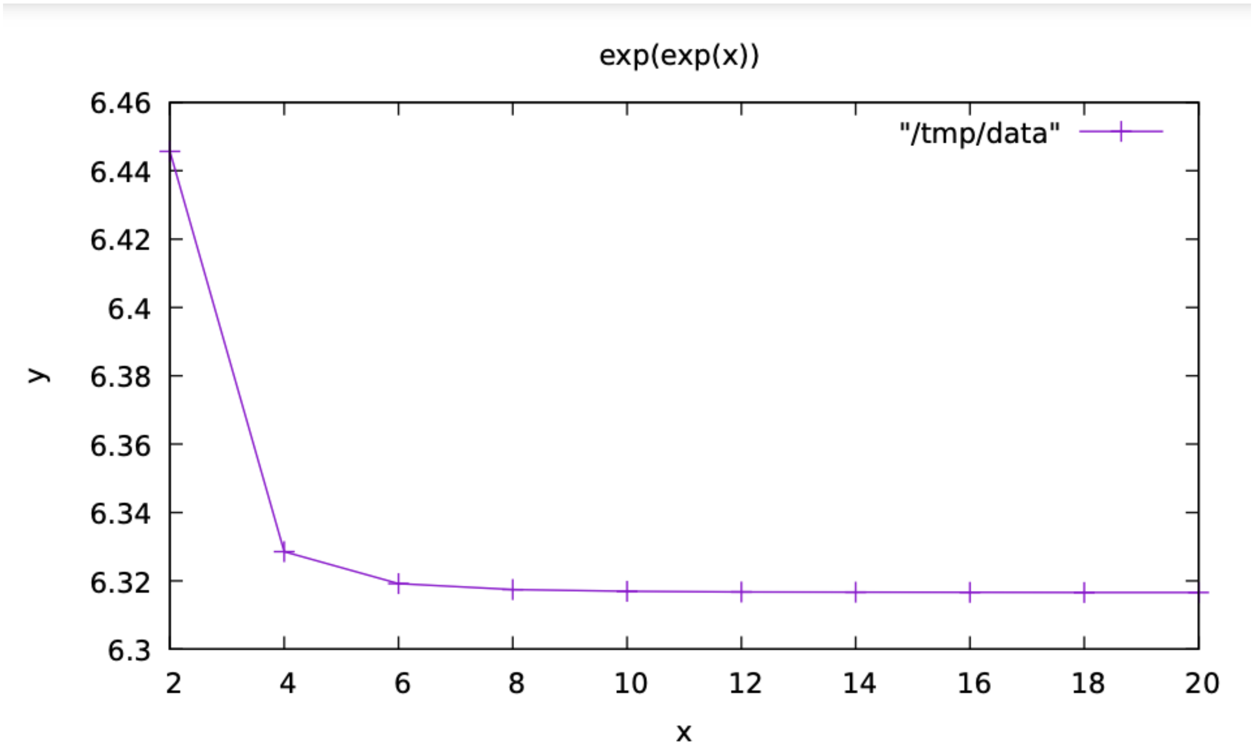
$sin(x)/x$:



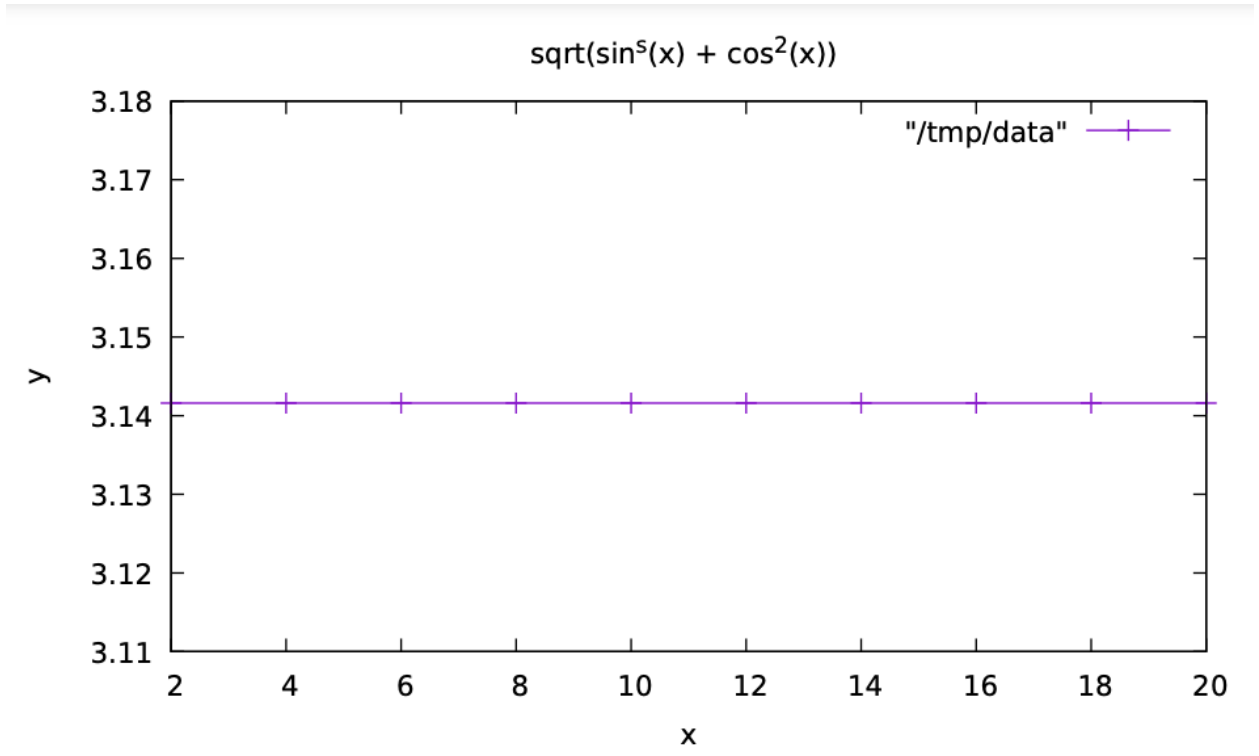
e^{-x}/x:


exp(-x)/x
"/tmp/data"
y
x
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
2
4
6
8
10
12
14
16
18
20

$e^{e^{x}}$:


exp(exp(x))
"/tmp/data"
y
x
6.46
6.44
6.42
6.4
6.38
6.36
6.34
6.32
6.3
2
4
6
8
10
12
14
16
18
20

$\sqrt{sin^2(x) + cos^2(x)}$:



Variance:

- As we can see, the graphs with the most amount of movement in the first 20 partitions, are e^(-x^2), sin(x^2), cos(x^2), and sin(x)/x.
    - The pattern amongst all four of these functions are that go from increasing to decreasing often.
    - The other graphs tend to increase or decrease, and not flip flop often. This makes it so that the 20 partitions estimate the area quite accurately. However, the 4 functions we talked about aren't so accurate as the 20 partitions lack the precision for the amount of movement there is in the graphs.

Floating point numbers:

- Having not done the scaling the scaling the first time I made my math library, I encountered the fact that floating point numbers are in fact not real numbers.
- Once the inputs got to a certain point, they were too large and the functions could no longer accurately produce a result, therefore using scaling, we had to chop the input to a more manageable value, and add onto the sum.