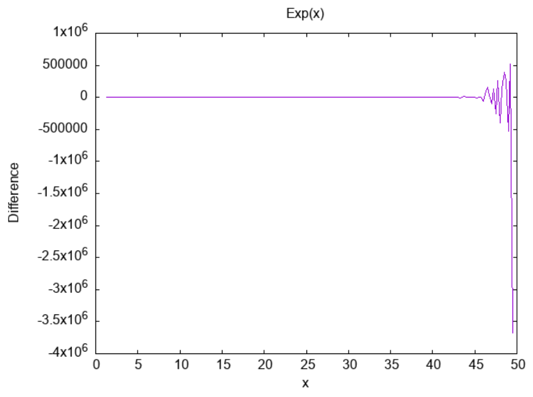
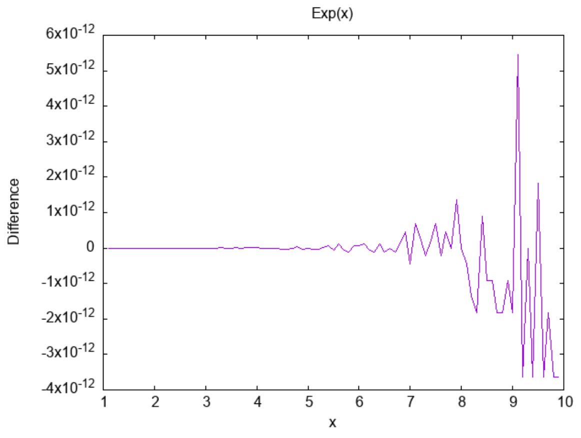
```
./integrate -p 1 -q 25 -n 10 -b> ./tmp/output
```

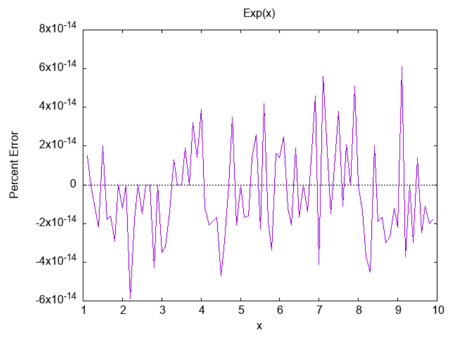
e^x - <math.h> vs “mathlib.h”



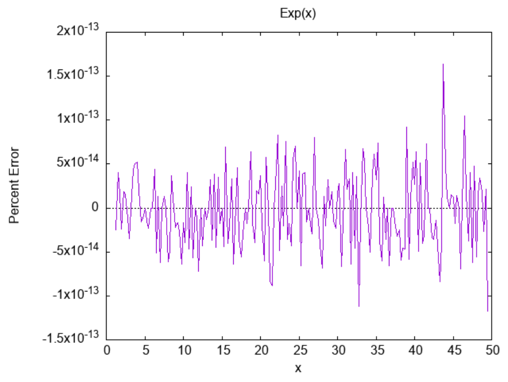
The graph to the left has the range x [0, 50) with increments of 0.25. The trend seems extremely stable and seems to show no sign of error until some x-value between 40 and 45. e^{40+}, however, surpasses a septillion, so the range is much too large to actually determine the error. There is a massive at x almost equals 50, which could be a cause for concern.

The left graph shows x [0, 10) with increments of 0.1. The graph clearly shows that there are signs of difference starting between the x-values of 3 and 4, though it to be noted it is very minor even at this scale of 1x10^12. The difference oscillates but it seems to be more focused toward the negative end of the spectrum therefore it can be inferred that if Exp(x) - exp(x) is mostly negative it is more likely that an answer is likely to have a higher percent error when on the negative side.





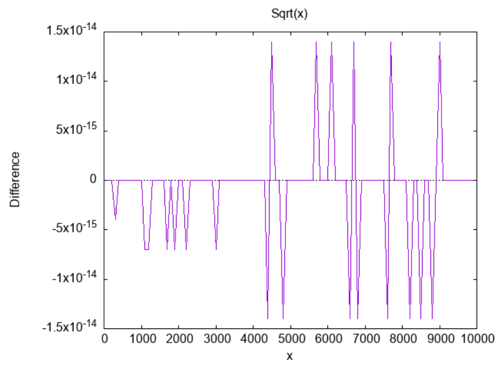
The graph is the %Error graph. There does seem to be a trend that the percent error tends to be higher when the Exp(x) is less than the math.h’s exp(x). However, this graph shows that from the ranges of x [0, 10) the error is relatively small, just barely above the defined EPSILON.

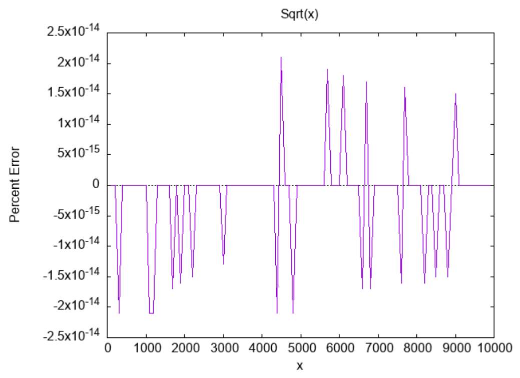


Even when the range is increased to the x [0, 50) the percent error is still contained however the error looks much more balanced compared to before, not staying over the negative.

Overall the Exp(x) function using the Taylor Approximation seems very reliable even when going to massive numbers.

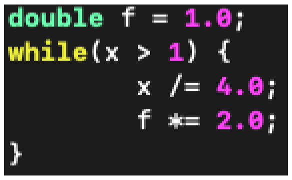
$\sqrt{x}$ - <math.h> vs "mathlib.h"





The top left Difference graph and the top right %Error Graph look very similar with the exact same peaks. In the left just comparing values, the error seems to be getting higher but when actually plotting the %Error graph the percent error is actually getting lower.

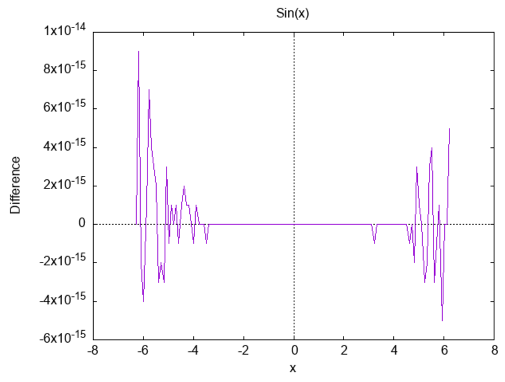
The main takeaway is that the Sqrt(x) function from "mathlib.h" actually gets more accurate overall the larger the x-value is. My guess is that this has something to the with the implementation of the below image before the narrowing down method.

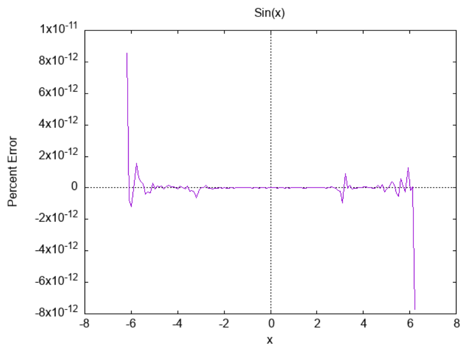


```
double f = 1.0;
while(x > 1) {
        x /= 4.0;
        f *= 2.0;
}
```

The value of 4.0 is much more exact than the method that is essentially guessed and checked. All the error comes from that guess and checks method the sqrt(4) will always equal 2. As the number gets larger the error from the guess of sqrt(1) will become less and less significant.

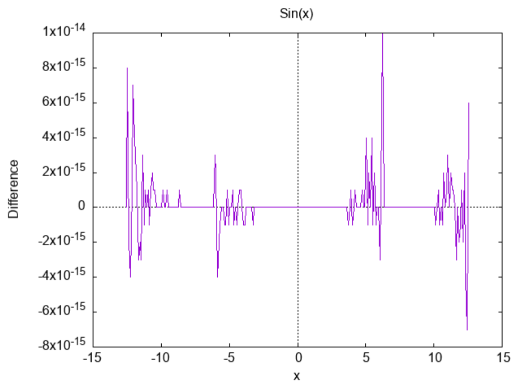
$sin(x)$ - <math.h> vs “mathlib.h”





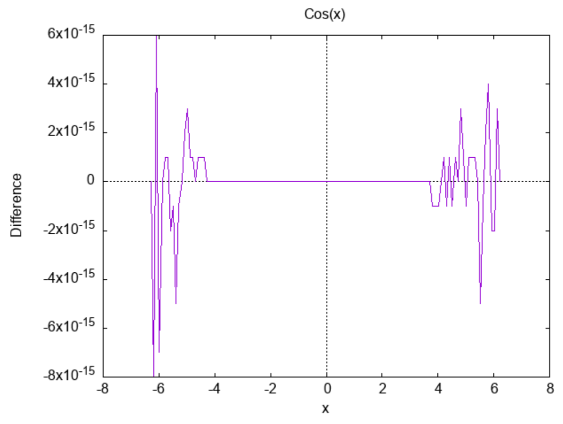
The Difference graph shows that the values of the Sin(x) function is most accurate till what looks from [-π, π] and from ±(π, 2π] the values are volatile, relatively speaking as the difference is still quite small.
The %Error graph also shows the same and reveals that at almost ±2π the values are off by more than 4 times what the regular error% would be. The %Error graph cannot actually reach 2π because sin(2π) is 0 and dividing by 0 will result in inf.

From the graphs, it is best to use Exp(x) between the range of [-π, π] when extreme accuracy is required.



The graph to the left is [-4π, 4π]. Oddly the difference values from [2π,4π] and [0,2π] are not the same despite using Fmod, a function that performs the same function as fmod, to modulus the x input by 2π.

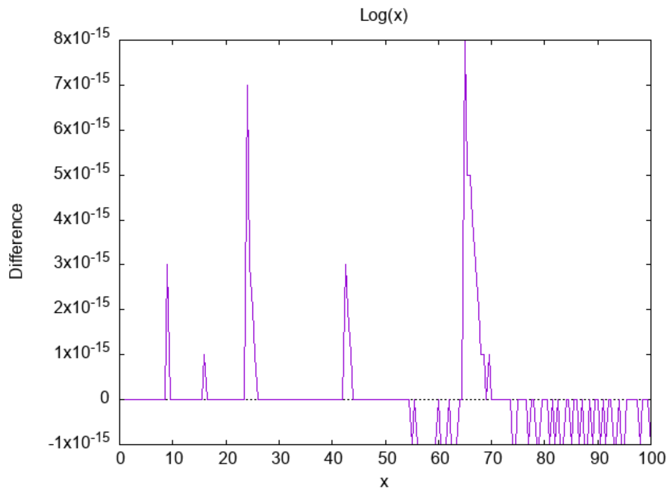
$cos(x)$ - <math.h> vs “mathlib.h”



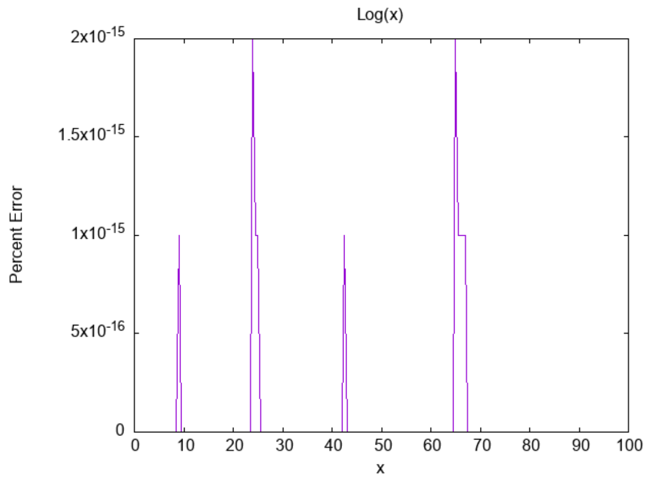
The graph to the left is the Difference graph of Cos(x), unlike Sin(x) where the peak of error was on the positive side.
Similar to Sin(x) the values are the most accurate in the range of [-π, π]

A %Error graph is not required for this since the range of y values is quite small.

lnx - <math.h> vs “mathlib.h”



This graph is the line from x-value[1, 100] with an increment of 0.1. The differences are minimal but spikes in the positive get progressively larger as x increases with wide areas of flat difference. At some x-value greater than 50 there is a very spikey negative difference.

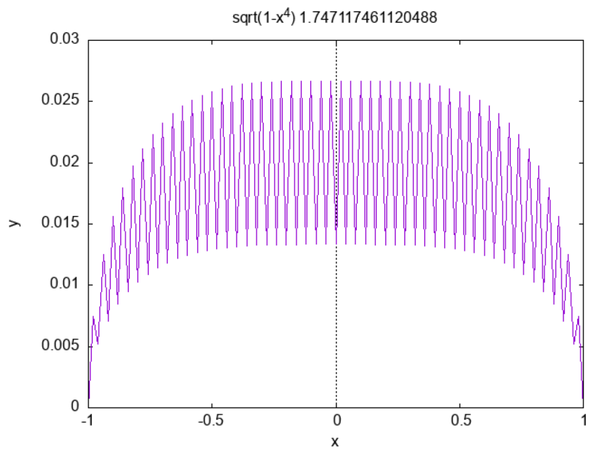


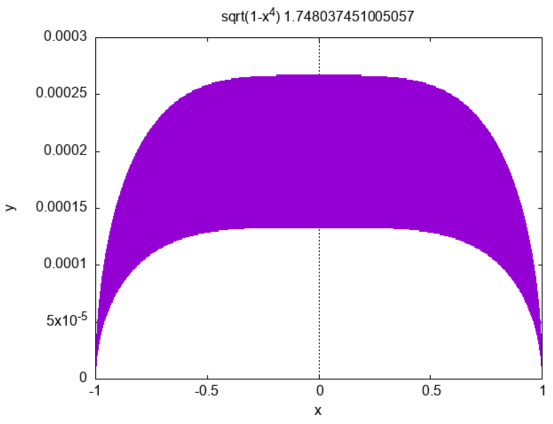
In the $Error graph, only the negative error seen in the Difference graph seems to have vanished. Maybe due to its relatively smaller size.

Log(x) gets more and more accurate as of the x value increases.

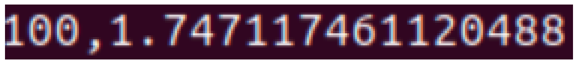
Integration Graphs:

Graph a: $\int \sqrt{1 - x^4} dx$



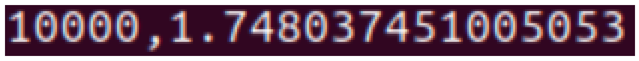


The graphs above use 100 and 10,000 partitions. This particular function has a range is [-1, 1]. If x was beyond it was the square root of a negative number and that would not be possible.



```
100,1.747117461120488
```

100 partitions from the example integrate function is the exact same.



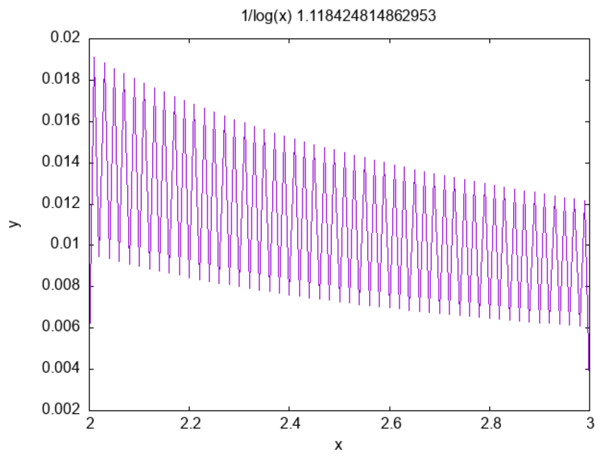
```
10000,1.748037451005053
```

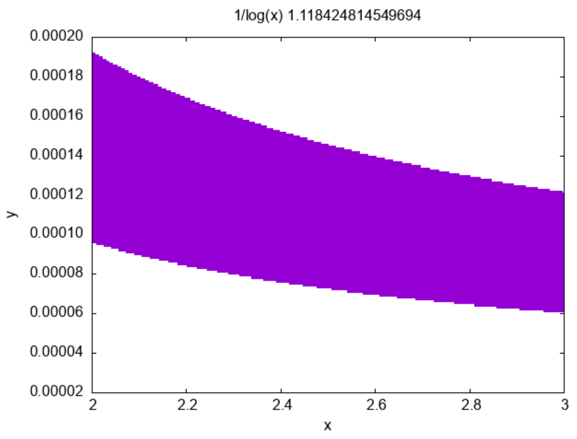
10000 partitions, however, does show a lack of accuracy supporting the idea that too many partitions can ruin accuracy and for the rance of [-1,1] it is too high.

Graph b: $\int \frac{1}{\ln x} dx$

This is an example\expected output

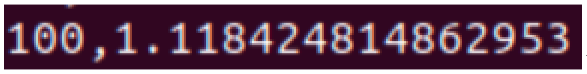
```
1/log(x)                    2        3    1.118424814549702
```





100 vs 10000 partitions.

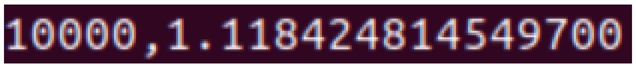
On the 10th decimal point, the value differs from the expected value for 100 partitions.



```
100,1.118424814862953
```

the example is the exact same for 100 partitions.

On the 13th decimal point, the value differs from the expected for the 10,000 partitions.



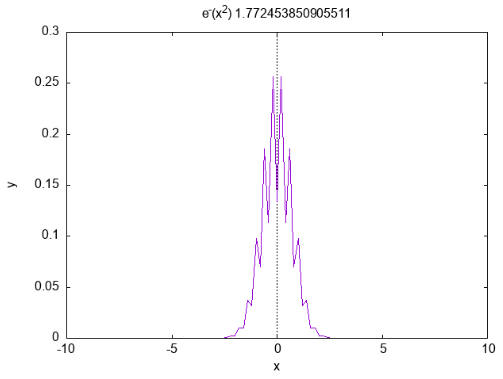
```
10000,1.118424814549700
```

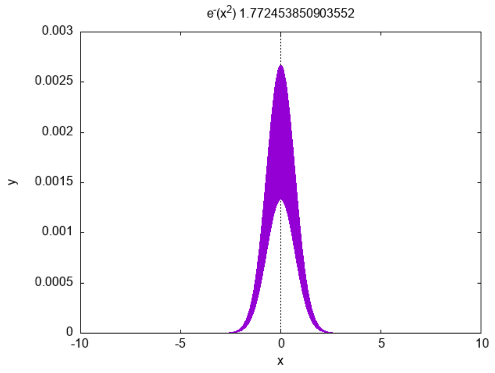
at the 13th decimal point, it deviates slightly.

The final result is more accurate than the more partitions. This makes sense because it is also a range of width [2,3].

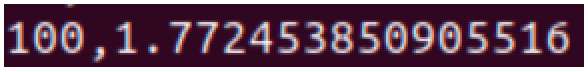
Graph c:

e^{-x^2} -10 10 1.772453850905508





The final for the 10,000 partitions is actually further off than the value of the example. This difference could be from the example itself not having less than 10,000 partitions. This also could be the result of having too many partitions the computing resources being wasted and the loss of accuracy as a result.



```
100,1.772453850905516
```

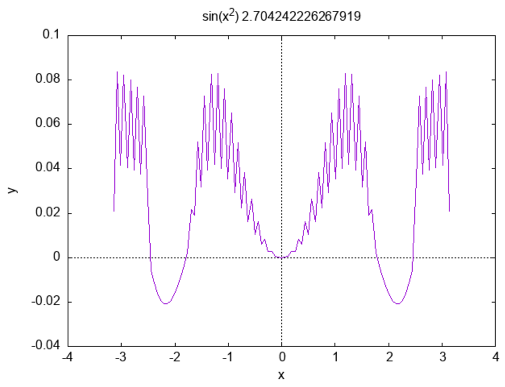
There is an error at the 16th decimal point.

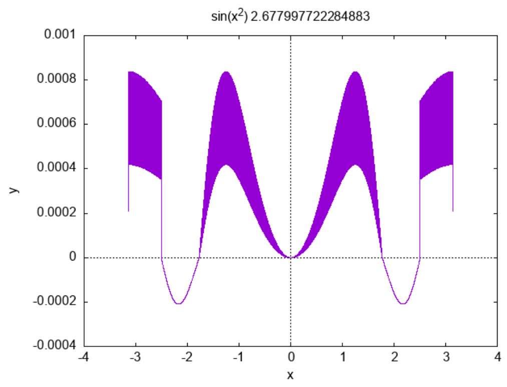
```
10000,1.772453850905516
```
Again for 10,000 partitions, there is an error at the 13th decimal point. Despite the fact that the range has increased from width 1 to width 20 x-values [-10, 10]. This is likely due to the actual range being much less as the real range of the function is around (~-2.5, +2.5).

100 and 1000 partitions for the example integrate function are exactly the same but my implementations of "integrate" are different.

Graph d:

$\sin(x^2)$	$-\pi$	π	1.545303425380133

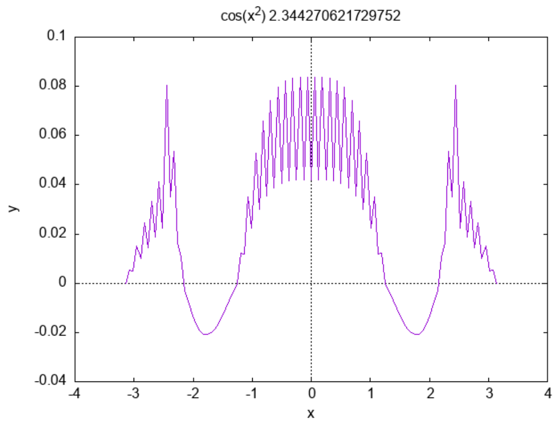


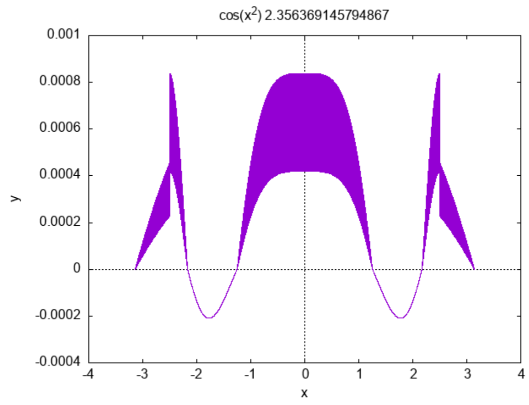


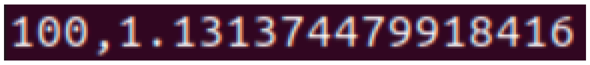
```
100,1.545345889800287
```
Now there is a huge difference between my implementation of integrate and the code. The difference is over 1.1

Graph e:

$\cos(x^2)$	$-\pi$	π	1.131387027213366





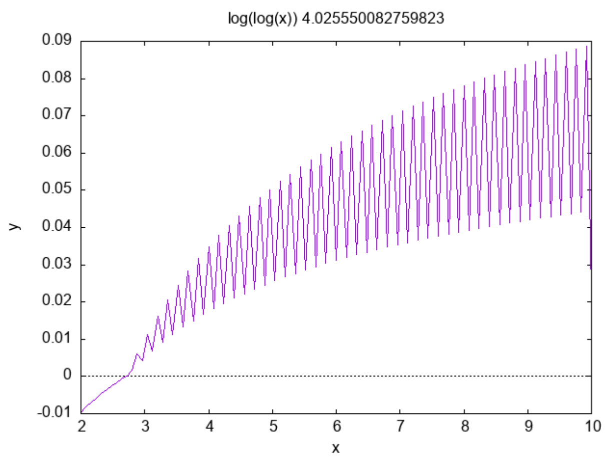


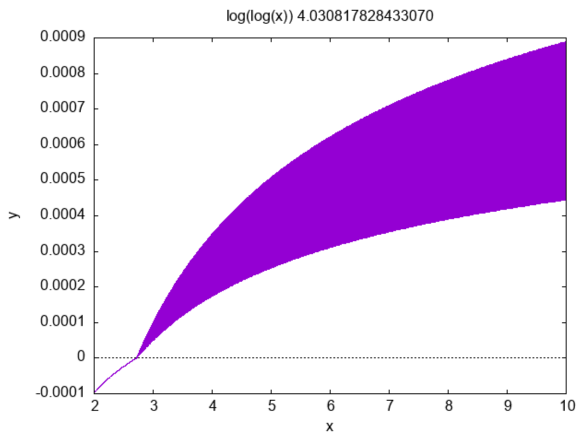
```
100,1.131374479918416
```

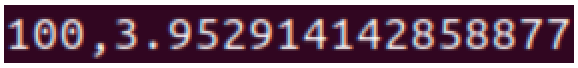
There is a massive difference between the two. The difference between them is over 1.2.

Graph f:

$\log(\log(x))$	2	10	3.95291414285876





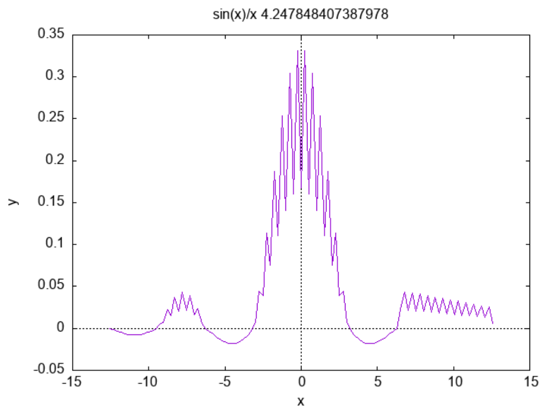


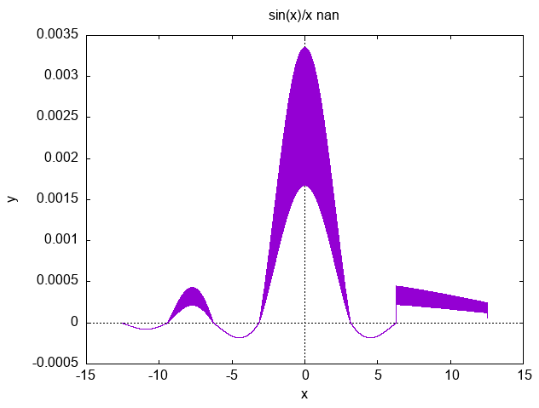
```
100,3.952914142858877
```

Same with this there is an error early at just the first decimal leading to supporting a new conclusion that there is an error in my implementation of integrate. The difference is 0.1.

Graph g:

$\sin(x)/x$	-4π	4π	2.98432245116892



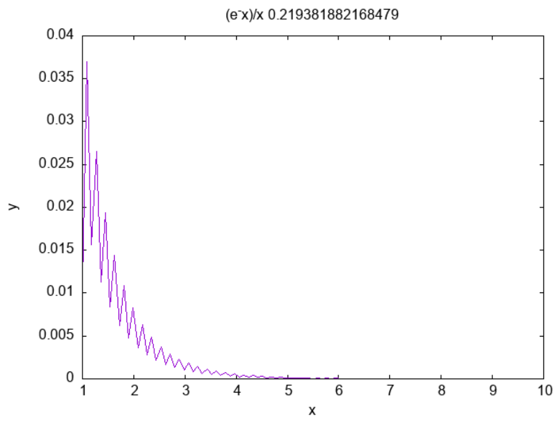


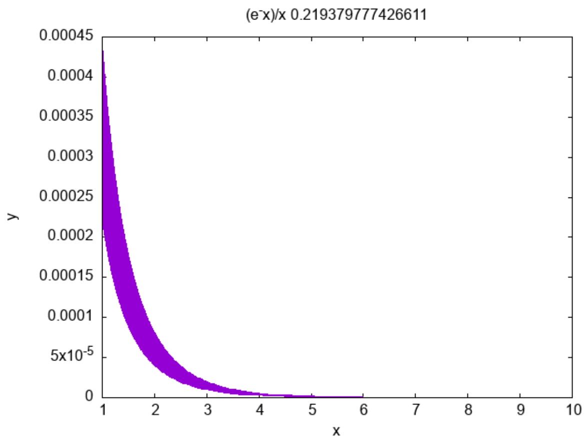
The difference between the example result is 2.9843 which is nowhere near what the 100 partitions were 4.224.

Graphs d, e, f, and g All have massive errors, despite all the parent functions of Exp, Sin, Cos, Log working very closely to math.h's results. This means there is some error in the implementation of integration.

Graph h:

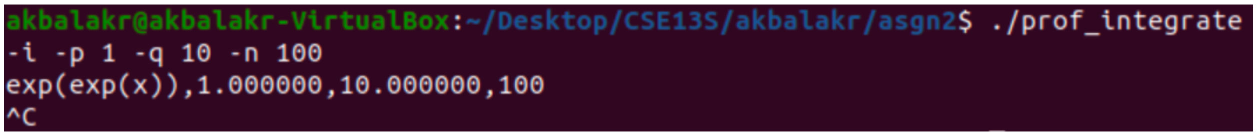
e^{-x}/x	1	10	0.2193797774265986





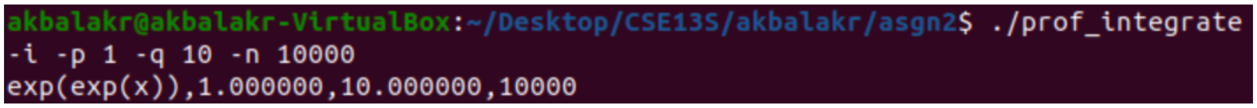
The 100 partitions result deviates at the 5th decimal point from the example above.
The 10000 partitions deviate on the 13th decimal point compared to the example above. In this case, 10000 did result in higher accuracy when the width of the range is 10.

Example integrate breaks at 100



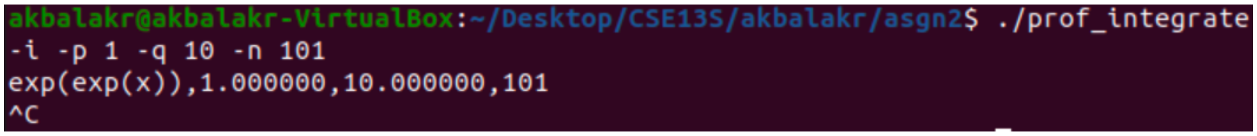
```
akbalakr@akbalakr-VirtualBox:~/Desktop/CSE13S/akbalakr/asgn2$ ./prof_integrate
-i -p 1 -q 10 -n 100
exp(exp(x)),1.000000,10.000000,100
^C
```

Example code breaks at 10,000 partitions



```
akbalakr@akbalakr-VirtualBox:~/Desktop/CSE13S/akbalakr/asgn2$ ./prof_integrate
-i -p 1 -q 10 -n 10000
exp(exp(x)),1.000000,10.000000,10000
```

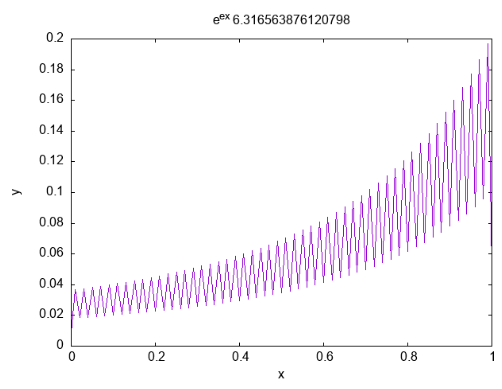
The example also breaks at 101 partitions even if x will not hit 0.0

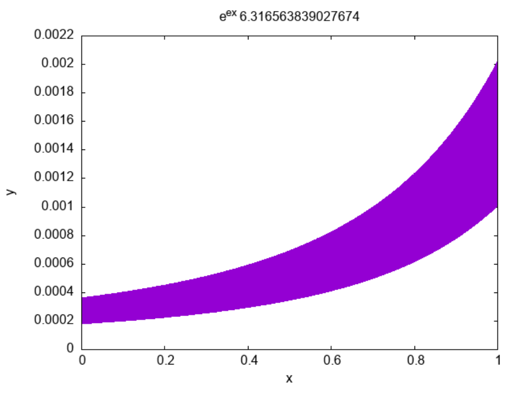


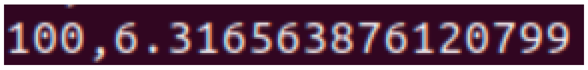
```
akbalakr@akbalakr-VirtualBox:~/Desktop/CSE13S/akbalakr/asgn2$ ./prof_integrate
-i -p 1 -q 10 -n 101
exp(exp(x)),1.000000,10.000000,101
^C
```

Graph i:

e^{e^x}	0	1	6.316563839027766

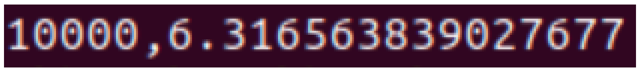






```
100,6.316563876120799
```

For a 100 the error is on the 16th decimal

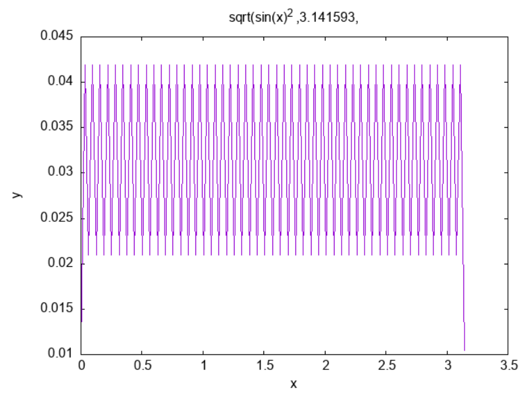


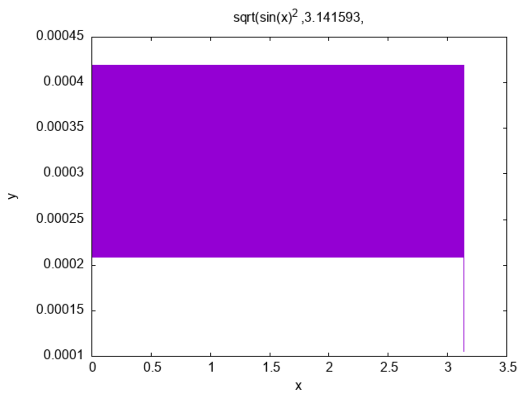
```
10000,6.316563839027677
```

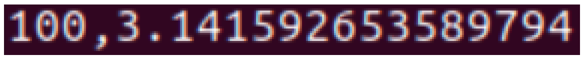
The error for 10000 is also on the 16th decimal point.
For small ranges of x, the error seems to be minimal.

Graph j:

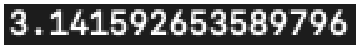
$\sqrt{\sin^2(x)+\cos^2(x)}$ 0 π 3.14159265358979







```
100,3.141592653589794
```



```
3.141592653589796
```

The error is on the 16th decimal point for 100 partitions for function j.