For each thread, the runtimes generally increase as the intervals increase. An example of this is in Reduction CalcPI2, where 1 thread with 1 interval took 0.000098 seconds while 1 thread with 10,000,000,000 intervals took 70 seconds. This pattern is seen as the threads increase. In other words, in each thread size, the runtime increases as the intervals increase. While the increase may be the same, the range of numbers is different. Specifically, the range of runtimes gets smaller as the threads increase. The runtimes are larger with larger amounts of threads at smaller intervals. An example of this is in Reduction CalcPI2, where 1 thread and 1 interval is 0.000098 seconds. With 10 threads and 1 interval, the runtime is 0.00492. However, the runtimes decrease as the threads get large and the intervals get large. With Reduction CalcPI2, 10,000,000,000 intervals with 1 thread takes 70 seconds and 10 threads with the same interval takes 18.6 seconds.

For both programs, the accuracy starts to degrade at 10,000,000,000 intervals. The accuracy stops increasing at 100,000,000 seconds with 16 numbers of precision (including the 3). The maximum amount of precision achieved was 17 decimals after the 3, as we were taught was the maximum precision. Amdahl's law explains that there is a limit of speedup with improved resources. From knowing this rule, I wonder if it also can apply to the limit of precision with improved resources (AKA more threads). The latency in accuracy may be explained by that same principle.

For both programs, 6 threads' runtime does not appear to decrease compared to four threads—overall the runtimes decrease as intervals get larger and threads get smaller. 6 thread runtimes simply do not follow this rule. The change is very slight and not very significant, but it is still a notable trend. An example of this is in Reduction CalcPI2, where 10,000,000,000 intervals takes 19.28 seconds and with 4 threads it is 18.49 seconds.

Let's compare CalcPI's trapezoidal method and calcPI2's arctangent method. Since we run calcPI with PE's and calcPI2 with threads and intervals, it's not very easy to quantitatively compare their precision. However, in general, both methods' precision increases as their resources increase. I noticed a latency in calcPI2's precision as intervals got increasingly large that I did not notice in CalcPI's. There may be a latency as the PE's get large with the trapezoidal method, I may just not have used large enough PE's. However, that is all that I was able to observe.

Lastly, I would like to note the difference between the mutual exclusion and reduction programs' runtimes. There is little to no difference between them. No difference is worth noting. This is because calling the barrier slows the program down significantly. When I called the barrier in my if statement in pthreadReduction.h (by mistake), the program slowed down to be much slower than the mutual exclusion one. Even calling the barrier in the correct places slows down the reduction program to the runtime of the mutual exclusion program. However, there may be an increase in the speed between the mutual exclusion and reduction programs' runtimes as the threads increase largely. In other words, as O(n) threads get increasingly large, the worst-case runtime of the reduction may be much better than mutual exclusion. However, it is not possible to see any speedup with the small amounts of threads that we use.