Ike Kaper, Alan Zhang, Derek Kimball, and Daniel Zhou

The Monte Carlo Experiment

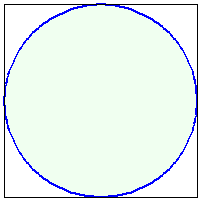
The Effect of the Number of Points Tested in a Monte Carlo Simulation on the Numeral Outcome close to the Value of π

Our group calculated experimental values for π using a Python program and analyzed the data within Microsoft Excel and Google Sheets. Our initial goals were as follows:

* 1. Determine the association, if at all, between the accuracy of a simulation and the length (how many points used to calculate π) of the simulation
  2. Determine the association, if at all, between the spread of resulting values of π and how many coordinates/points were used in the simulation
  3. Determine the association, if at all, between the length of the simulation and the execution time.
  4. Answer for our research question:
     1. **How is the percent difference associated with run length, if at all?**

**The methodology and thought process are as follows:**

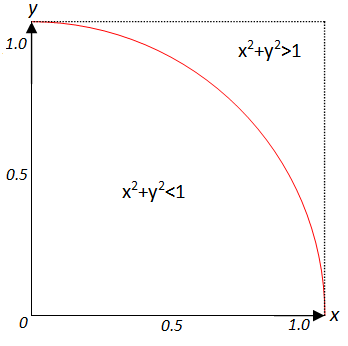
By definition, Monte Carlo simulations use random number sampling to obtain a desired numerical result. To understand the justification for using such a simulation to calculate an experimental value for π, one needs only a rudimentary knowledge of probability, algebra and geometry.



The above is the unit circle inscribed inside a square. While it is technically possible to run the simulation with this figure, We decided not to for several reasons.

1. This is a larger area and the resultant data would be equivalent to data acquired from a smaller segment
2. It would be more difficult, in my view, to understand the visual representation
3. The side length of the square is double the magnitude of the circle's radius. Because of this, the calculations would be more complex

We instead opted to slice the diagram into a fourth as shown below.



The area of the region inside the quarter circle is exactly equal to 1/4 the area of a full unit circle and since in this example we're using the unit circle as our baseline, the radius is equal to 1, the area of the region is as follows



The area of a square in which the unit circle is inscribed is



The area of the square in the above diagram, which is a quarter of the area of the original square is



With the areas determined, we can now determine the probability, if points are "placed" in the quarter square randomly, that they fall within the area of the quarter circle



So, by this chain of logic, we may construct a Monte Carlo simulation to determine a value for π by having the program generate two sets of random numbers (one for the x and y dimensions) between zero and one. If, as the image above states, the sum of the two squares is less than one, the point is counted as falling inside the area of the quarter circle. At the end, the experimental value of π is calculated by

1. Dividing the number of points inside the quarter circle by the total amount of points "thrown".
2. Multiply the ratio from above by 4, since otherwise, as the equation from above demonstrates, the result would be approaching π/4

Our data is randomized thanks to the numpy Python libraries internal random function.

**Issues along the way:**

We had several issues whilst writing the code. The major problem encountered was when the program’s export data to excel function was spitting out nonsensical--smallest integer value, the greatest integer value, 0, and etcetera. The core of the problem was that the program had a special way of formatting the data we wanted to export. We worked around the issue by parsing the data structure to only export the necessary information. In addition, while collecting data, we were surprised to learn that the program’s execution time did not scale linearly with size. For example, a 10x increase in the length of the simulation (from 10,000 to 100,000) yielded a greater than 150x increase in execution time. This told me that there were inefficiencies in my program and this was the primary reason we did not collect data with more than 100,000 points as adding more made the execution time unreasonably large.

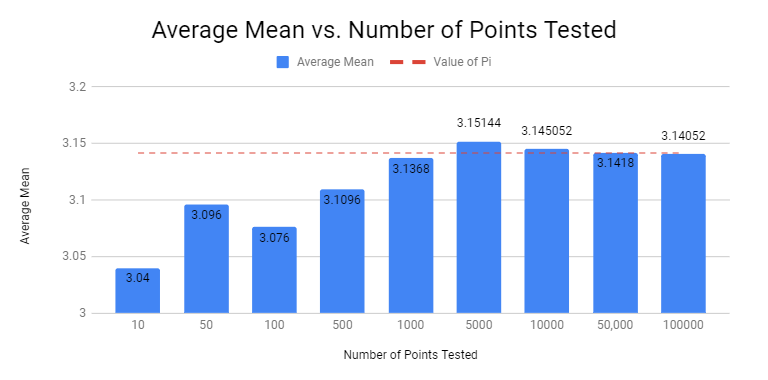
**Improvements:**

**Fixing the inefficiencies currently within the program**: Based on my research, there are complex ways to make the numpy number generation more efficient. If this were fixed, we would be able to see if a run length of one million test points would produce noticeable differences

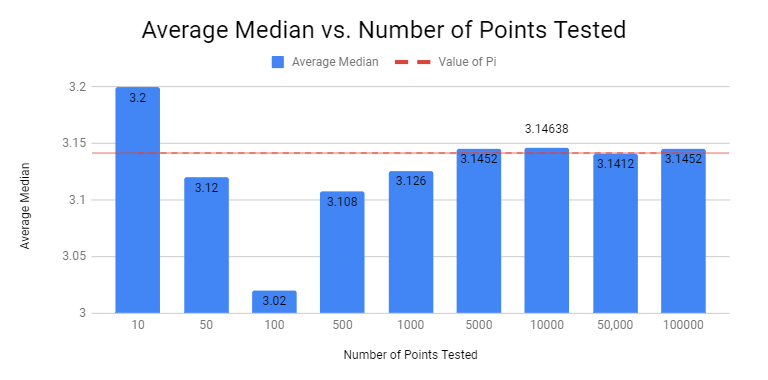
**Automation:** Currently collecting data requires a lot of human input to save and rename both the excel spreadsheet and the screenshots. In addition, after the data is collected and exported, it requires human input to start the next run of data collecting. If the automation was added to the program, we could leave the software running overnight and have it display data.

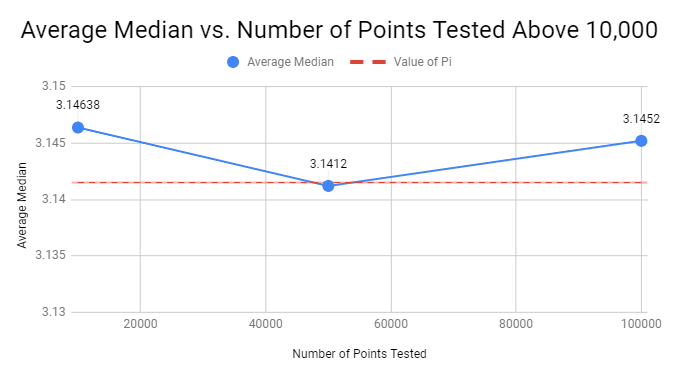
**Analysis of our data:**

On the bar graph of the average mean versus the number of tested numbers, it shows a weak linear positive correlation. Thereafter, from 5,000 to one hundred thousand tested numbers, the averages began to plateau towards π. The average mean graph is skewed to the left. A noticeable difference is that the average mean decreases at 50,000 numbers tested.

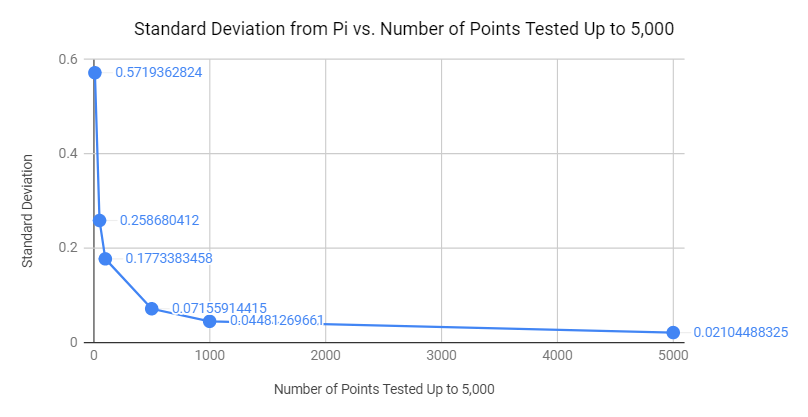


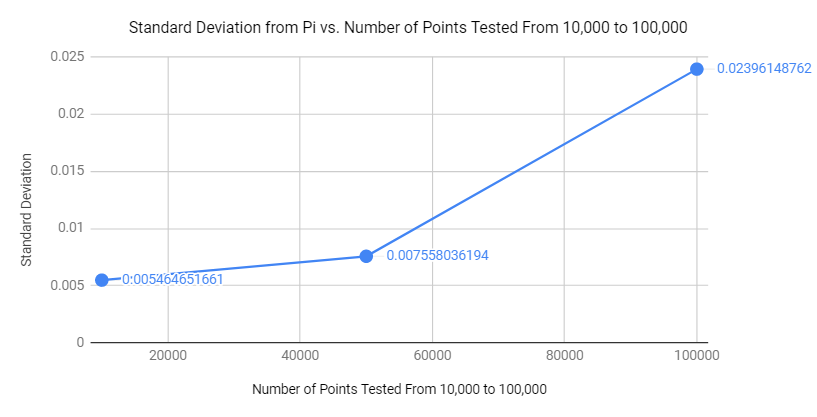
On the bar graph of the average medians of the tested numbers, the average median demonstrated a sharp drop from tested numbers 50 to 100. From 500 to 5,000 tested numbers, the average median showed a strong positive correlation. Thereafter, from 5,000 to one hundred thousand tested numbers, the averages began to plateau towards π. The average median graph is bimodal. Looking at the median averages from 5000 to 100,000, the data shows a major drop at 50,000. Line graphs have been included to illustrate the drop at 100 and 50,000.



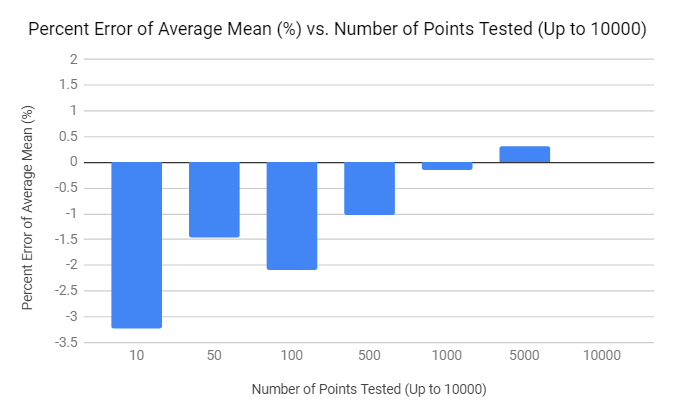


In the line graphs for the standard deviation from π, the standard deviation shows a weak linear negative correlation. Looking at tested numbers from 10,000 to 100,000, the standard deviation from π increased and showed a moderate positive correlation. It is smallest when only 10,000 numbers are tested. Two lines graphs are shown below: the first shows the standard deviation from numbers tested up to 5,000, and the second shows the numbers tested above 10,000.

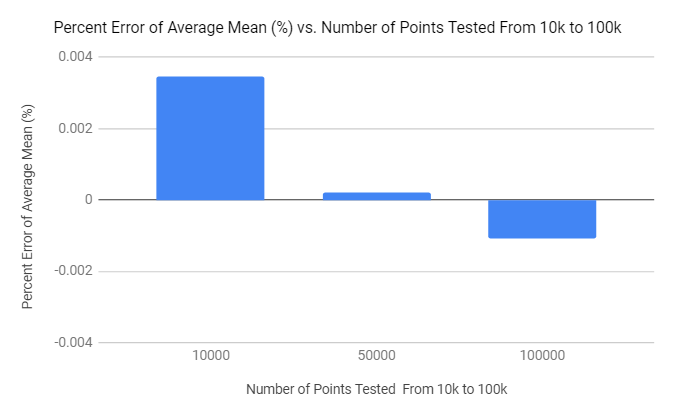




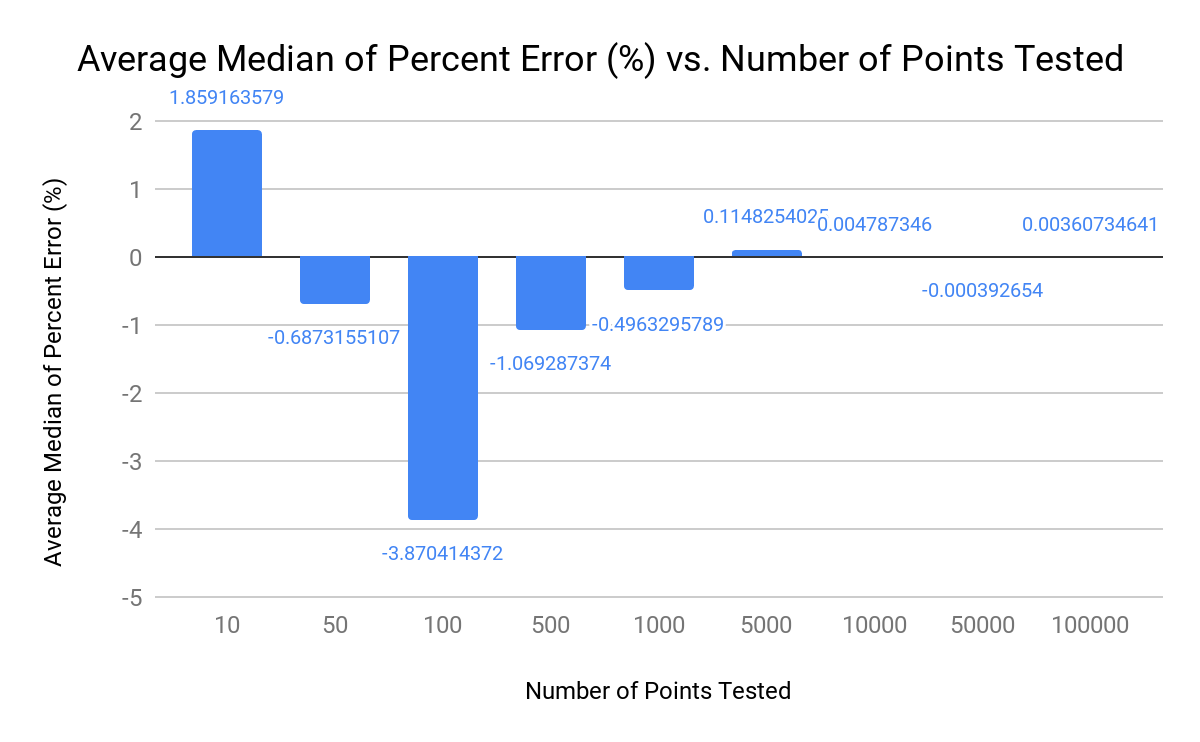
The bar graphs of the percent error of the average mean from π as the number tested numbers up to 10,000 showed a moderate linear positive correlation. Interestingly, from the tested numbers higher than 10,000, the percent error decreased towards zero with 50,000 tested numbers being the closest to zero percent error (0.000207346041% off).



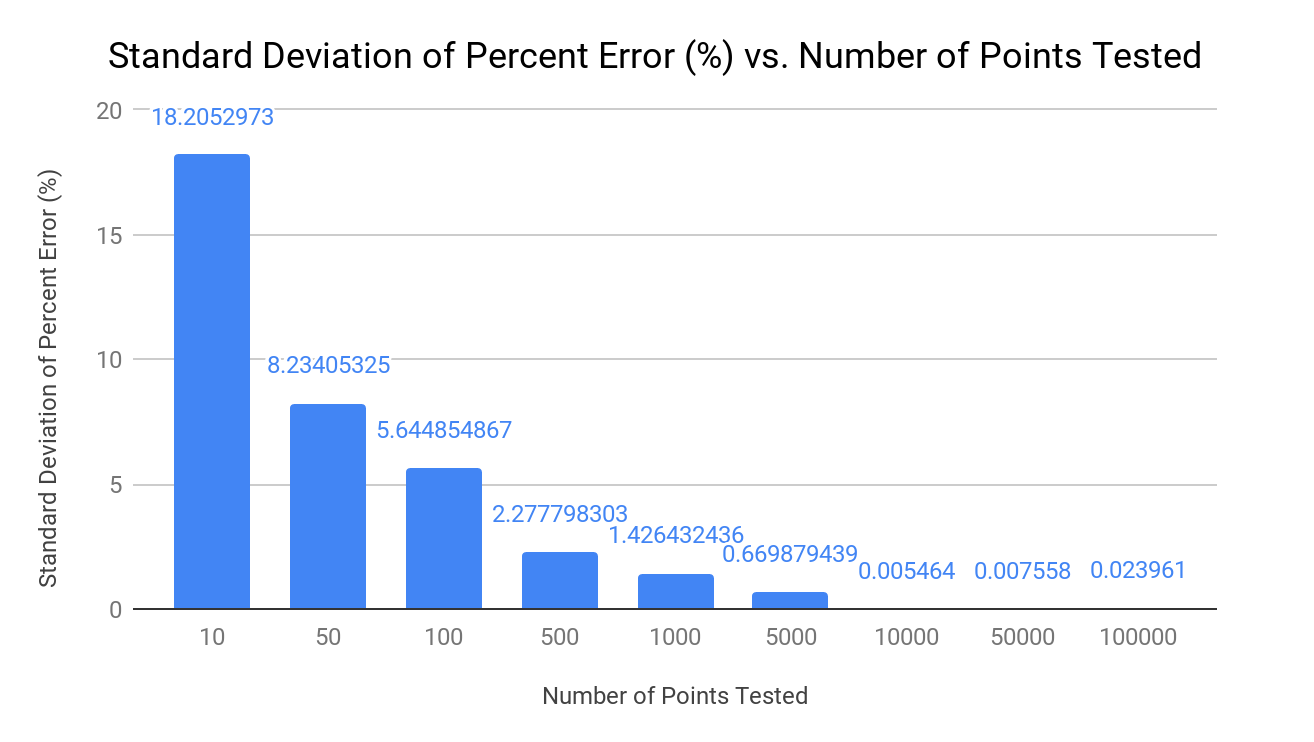
The bar graphs of the percent error of the average mean from π as the number tested numbers above ten thousand showed a moderate linear negative correlation. When 100,000 numbers were tested, the percent error is actually negative. The percent error is lowest at fifty thousand tested numbers.



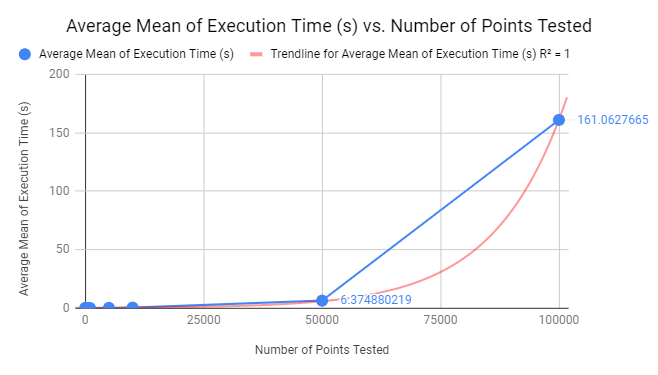
In the bar graph of the average median of percent error, there is no linear correlation. As the number of tested numbers increase, the average percent error approaches zero. From ten to one hundred numbers tested, it shows a strong linear negative correlation (R^2 = 1). From one hundred to five thousand tested numbers, it shows a moderate linear positive correlation (R^2 = 0.846).



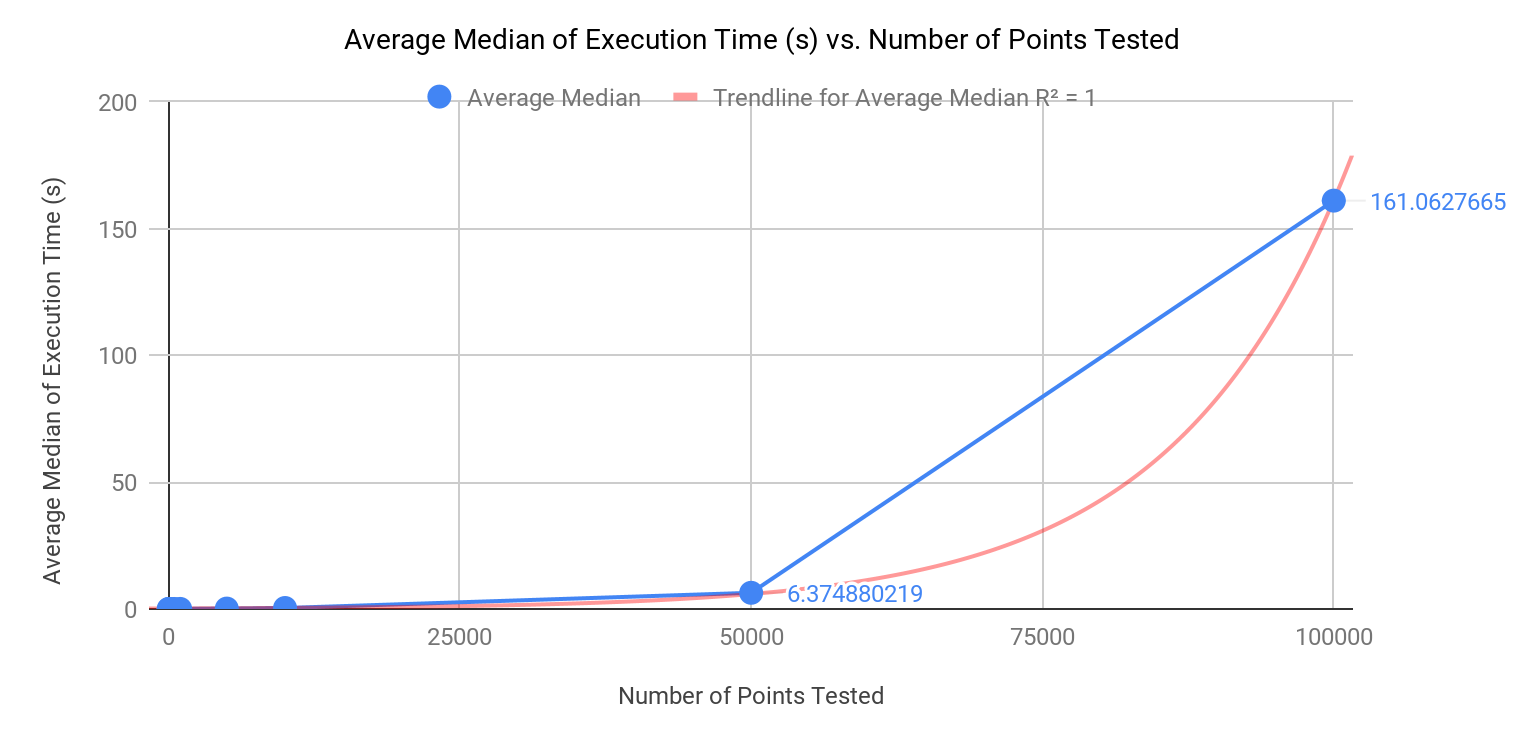
In the bar graph of the standard deviation of percent error, the graph shows a negative correlation and approaches zero as the number of tested numbers increases.



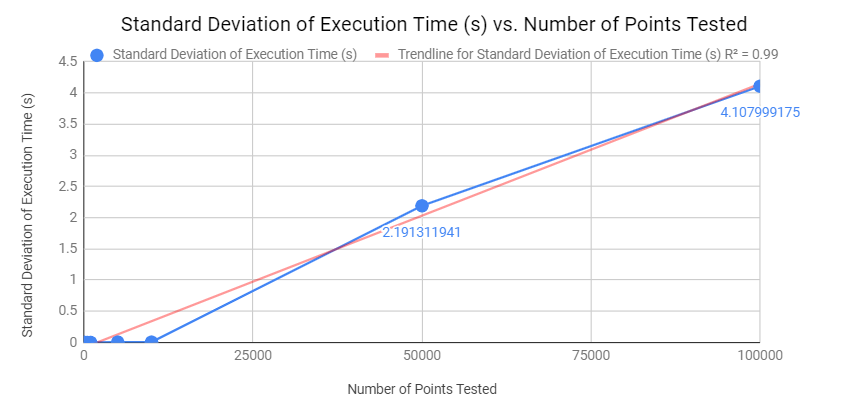
For the line graph of the average mean time of execution for the data as the number of tested points increased, it showed a weak linear positive relationship. The line had a strong exponential positive correlation (R^2 = 1). An exponential equation can be created that models the data: Average Mean = 0.215e^(6.62E-5)x, which shares a similar equation to the exponential relationship of the average median time of execution as the number of points increased.



For the line graph of the average median time of execution for the data versus the number of points tested, it shows a strong exponential positive correlation (R^2 = 1). An exponential equation can be created that models the data: Average Mean = 0.21e^(6.62E-5)x, which shares a similar equation to the exponential relationship of the average median time of execution as the number of points increased.



In the graph of the standard deviation of the execution time, it shows a strong linear positive correlation (R^2 = 0.99). Looking closely, the standard deviation of the execution time at 5,000 tested numbers is greater than the standard deviation for 10,000 tested numbers. The linear equation that models the data: Standard Deviation of Execution time (s) = (4.23 \* 10^ - 5)(Numbers of points tested) - 0.816.



**Conclusion:**

As we increased our run length from the tested numbers higher than 10,000 on the bar graph of the percent error of the average mean from pi, the percent error decreased towards zero, and the accuracy the value of π increased as the run length increased. From the run length of 10,000 and on, there was not that much change in percent error from π. As the run length increased, the spread of the resulting values decreased because the sample size was so big it ignored the outliers. As the run length increased, the time it took to run the python program increased exponentially. There is a hidden outlier which is the code itself.