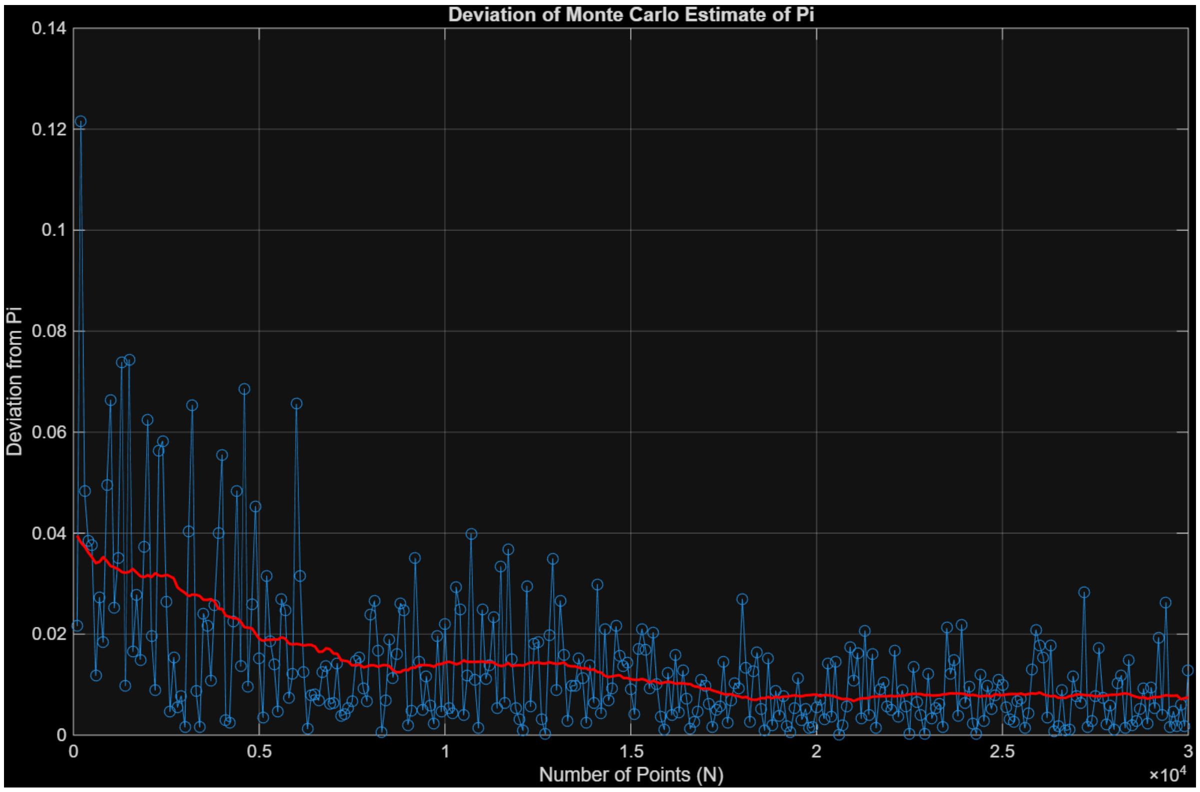
John Sipala – AMS 595 – Project 1

Github Repo - https://github.com/jdsipala/ams595-project1/

The first task was to create a Monte Carlo simulation by using a for loop. Using basic reasoning, we can see that if we create points that are uniformly distributed in [0,1] x [0,1] for both the x and y coordinates, and then count whether or not these points fall inside the (quarter) circle, the ratio of the total number of points within the circle divided by the overall number of points can be used to estimate the value of pi.

You will see a function called 'pi_montecarlo' that shows the specific formula used to calculate this estimated value of pi. The following graph shows how increasing the number of points will make the estimated value of pi closer to the true value of pi as the sample gets larger. The graph below was created by using the function that I created, applying that function to different N values from 100 to 30,000. What you see in this graph is the absolute difference between the estimate of pi for a particular N and the true value of pi. Because the graph can vary a lot from one point to the next, I decided to add a red line that shows a moving average of the points on the graph, to show that the deviation from the true value of pi gradually declines.

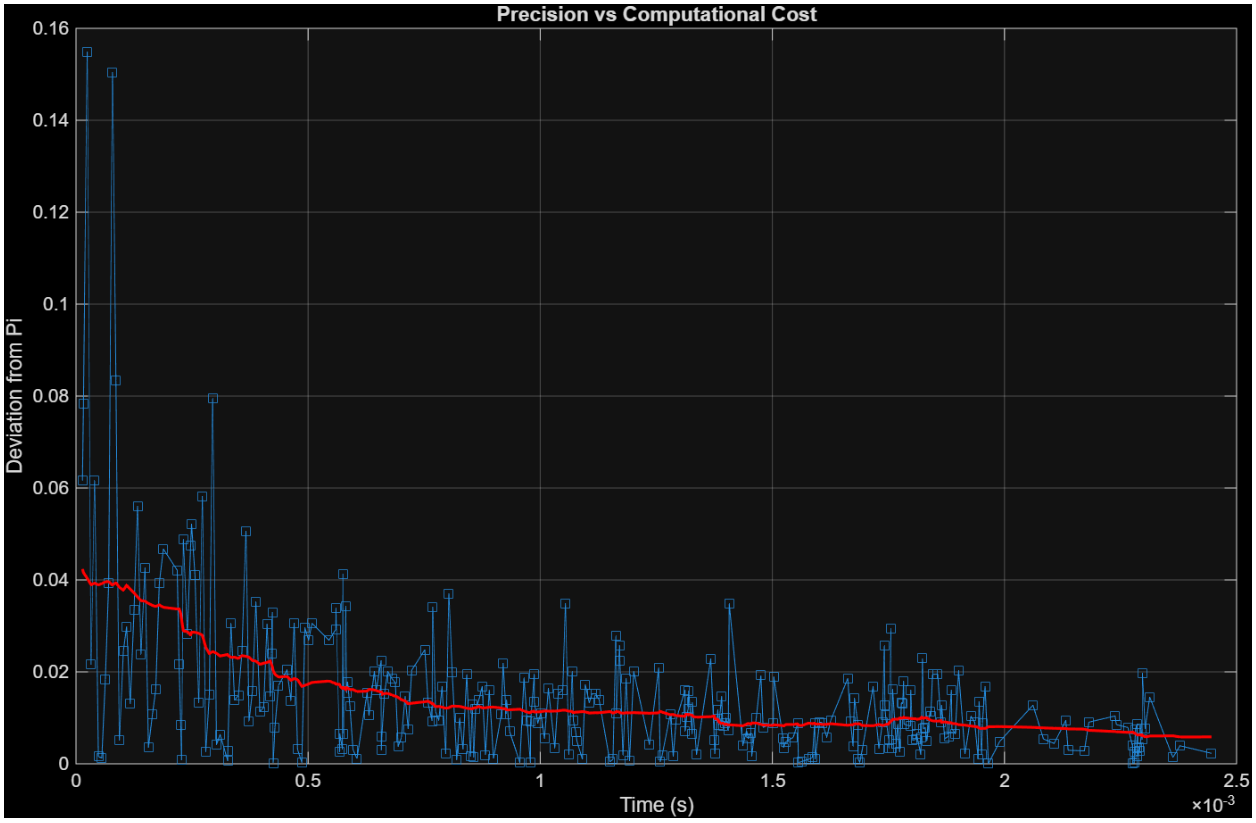


The next graph shows the tradeoff between precision (how close the estimate of pi is to the real value of pi) vs computational cost (how long it takes for the code to run).

https://www.mathworks.com/help/matlab/ref/tic.html

https://www.mathworks.com/help/matlab/ref/toc.html

The matlab functions tic and toc are built-in MATLAB functions that I used to get this data. Once again, I also used a moving average to show a smoother overall trend in the data, which shows that the more accurate samples, in general, took longer to compute.



I also created a function called 'pi_sigfigs' which uses a while loop to compute the estimate of pi. A key part of this function is the use of the normal approximation to the binomial confidence Interval.

$$p \pm z_{1-\alpha/2}\sqrt{\frac{p(1-p)}{n}}$$

The function treats *p* as a the proportion of points that fall inside the circle (and *1-p* as outside). It takes the number of significant figures as an input and continues to add points to the sample until the upper and lower bounds at a 95% confidence interval are rounded to the same value as the requested significant figures (or decimals). The last function I created is called ‘pi_sigfigs_plot’ which shows a graphical representation of the inner workings of our Monte Carlo estimation of pi. This function can show the points being plotted one by one as they are generated, but I set it to plot points in groups of 1,000 just to make the code run faster.

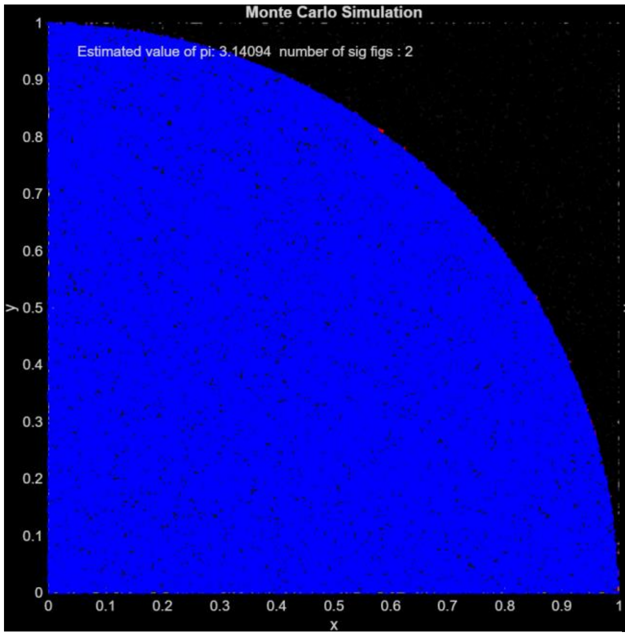
The use of the binomial confidence interval is a strong way to show that the estimate will be statistically close to the desired number of significant figures. However, the drawback of using the binomial confidence interval at 95% is that it can require a sample on the order of billions once you intend to round pi to 5 significant figures or more.

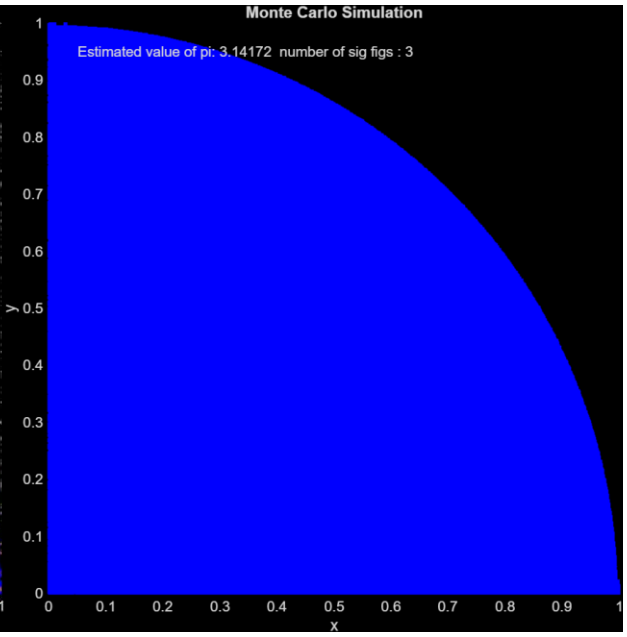
Below are three key MATLAB functions that I used to create the live animation for the graph.

https://www.mathworks.com/help/matlab/ref/animatedline.html

https://www.mathworks.com/help/matlab/ref/animatedline.addpoints.html

https://www.mathworks.com/help/matlab/ref/drawnow.html


Monte Carlo Simulation
Estimated value of pi: 3.14094 number of sig figs : 2
y
x


Monte Carlo Simulation
Estimated value of pi: 3.14172 number of sig figs : 3
y
x