ENGS 107 Problem Set #2: Monte Carlo / Convergence / Seeds / Scripts / Plotting

TASKS (#1-4)

1. Review an example script we discussed in class

I carefully reviewed the following example script discussed in class: coin-example.R

2. Review at least two other example scripts (for example from code replication repositories from papers in your application area)

I carefully reviewed the scripts from the code replication repositories from each of the following research articles published in the metabolic engineering field (my doctoral research field):

1. Backman, T.W.H., Schenk, C., Radivojevic, T., Ando, D., Singh, J., Czajka, J.J., Costello, Z., Keasling, J.D., Tang, Y., Akhmatskaya, E. & Martin, H.G. BayFlux: A Bayesian method to quantify metabolic Fluxes and their uncertainty at the genome scale. *PLoS Comput Biol* 19(**11**), 1-26 (2023).

2. Hosoda, S., Iwata, H., Miura, T., Tanabe, M., Okada, T., Mochizuki, A. & Sato, M. BayesianSSA: a Bayesian statistical model based on structural sensitivity analysis for predicting responses to enzyme perturbations in metabolic networks. *BMC Bioinformatics* 25(**297**) 1-21 (2024).

3. Baltussen, M.G., van de Wiel, J., Regueiro, C.L.F., Jakštaitė, M. & Huck, W.T.S. A Bayesian Approach to Extracting Kinetic Information from Artificial Enzymatic Networks. *Analytical Chemistry* 94(20) 7311-7318 (2022).

3. Review the key sources already assigned as reading with a special focus on:

I carefully reviewed Labs 0 to 3 in: Applegate, P.J., & Keller, K. (Eds.). (2016). Risk analysis in the Earth Sciences: A Lab manual. 2nd edition. Leanpub. Retrieved from https://leanpub.com/raes

Below is a listing of the reviewed Labs:
1. Lab #0: Learning the basics of R (Patrick J. Applegate)
2. Lab #1: Downloading and plotting time series data (Patrick J. Applegate)
3. Lab #2: Normal distributions and the Galton board (Patrick J. Applegate)
4. Lab #3: Other probability distributions and random sampling (Patrick J. Applegate)

4. Use a Monte Carlo simulation method to:

a. Determine the mean and the 95th percentile from a known univariate normal distribution with a mean of zero and a standard deviation of one with your estimated uncertainties

TASK

To address question 4A, this task requires implementing a Monte Carlo simulation method to calculate the mean and the 95th percentile of a known univariate normal distribution with a mean of zero and a standard deviation of one, while quantifying the uncertainties associated with these estimates. Within this task, different seeds should be used to ensure reproducibility and controlled variability across the Monte Carlo simulations. Furthermore, convergence should be assessed and stopping criteria should be established to help assess if the estimates for the mean and 95th percentiles for each seed converge to their expected values as the number of iterations of the Monte Carlo simulation increase up to a maximum threshold of iterations. For each iteration of the Monte Carlo simulation, a random sample should be drawn from the univariate normal distribution and its mean and 95th percentile values should be calculated. The uncertainty of these estimates should be quantified by tracking their variation across the iterations, and a primary goal of this task is to observe how 1) the mean and 95th percentile estimates converge and 2) their associated standard deviations decrease as the number of iterations of the Monte Carlo simulation increase. The deliverables for this task should include recording the calculated average estimated mean and the average estimated 95th percentile along with their associated standard deviations calculations for each seed. In terms of graphically presenting the findings, the results from the normal distribution should be presented as a histogram plot and the estimated mean and 95th percentile values should be plotted for each seed over the number of iterations. In addition, the standard deviations of the estimated mean and 95th percentile values should be plotted for each seed over the number of iterations as well, concluding the analysis.

APPROACH

To accomplish the task described above for question 4A, my approach was broadly aimed at producing a code script in R that followed a sequential and logical flow of phases comprised of smaller steps to set up and execute the Monte Carlo simulations, perform the necessary calculations, and visualize the results using a histogram and a set of convergence plots.

The first phase of my approach was focused on establishing the environment for performing the Monte Carlo simulations. This phase included importing a grid package library to enable enhanced graphical functions in R for use in subsequent plotting and recording of the results, specifying the mean and standard deviation values for the normal distribution from the problem statement, and defining five different random seed values (3, 5, 7, 10, 14) for reproducibility and controlled variability. This phase also included defining the convergence criteria to stop the Monte Carlo simulation iterations if the last 100 estimates for the standard deviation of the estimated mean and 95th percentile estimates fluctuated within the bounds of the threshold (0.005). This phase also included defining the maximum number (upper limit) of Monte Carlo simulation iterations at 10,000 iterations to prevent long computational times or infinite loops. Finally, this phase included initializing data frames to store the estimated means, 95th percentiles, and their standard deviations for each iteration of the Monte Carlo simulation.

The second phase of my approach was focused on setting up the for loop to run the Monte Carlo simulations for each of the five seeds. The for loop that I wrote within the code script is somewhat complicated and a description of each of the primary steps within the for loop is

described below. The first step was to set the seed within the for loop to ensure that each simulation run with that specific seed would produce reproducible results. The second step was to generate an initial random population. To do this, I used the rnorm function to generate 1,000 random values from the normal distribution and stored the values under the random_population variable. The third step was to initialize empty storage vectors to store the estimated means, 95th percentiles, and their rolling standard deviation values. The fourth step was to initialize control variables to 1) track if the stopping criteria had been met and 2) to count the number of Monte Carlo simulations. The fifth step was focused on setting up a while-loop within the for loop to run the Monte Carlo simulation until either the convergence criteria were met, or the maximum number of iterations (10,000 iterations) had been reached. This included setting up sampling with replacement from the generated random population, calculating the estimated mean and estimated 95th percentile values for the sampled data, and calculating a rolling standard deviation for the means and 95th percentile values over the iterations. I also set up an if statement to assess the convergence criteria after each set of 1,000 iterations to determine if stopping the Monte Carlo simulation should occur and I also set up an iteration counter to move to the next iteration if the stopping criteria were not met. The sixth step was to store and save the final set of results for that specific seed once the stopping criteria were met or the maximum number of iterations were reached. The seventh step was to calculate the averaged estimated mean, the average estimated 95th percentile, the final standard deviation of the estimated mean, and the final standard deviation of the estimated 95th percentile for that specific seed. And finally, the eighth step within the for loop was to append the final calculated results for that specific seed to the summary data frame and to close the for loop to allow the for loop to repeat the same process (steps 1-8 of phase 2) for the next seed.

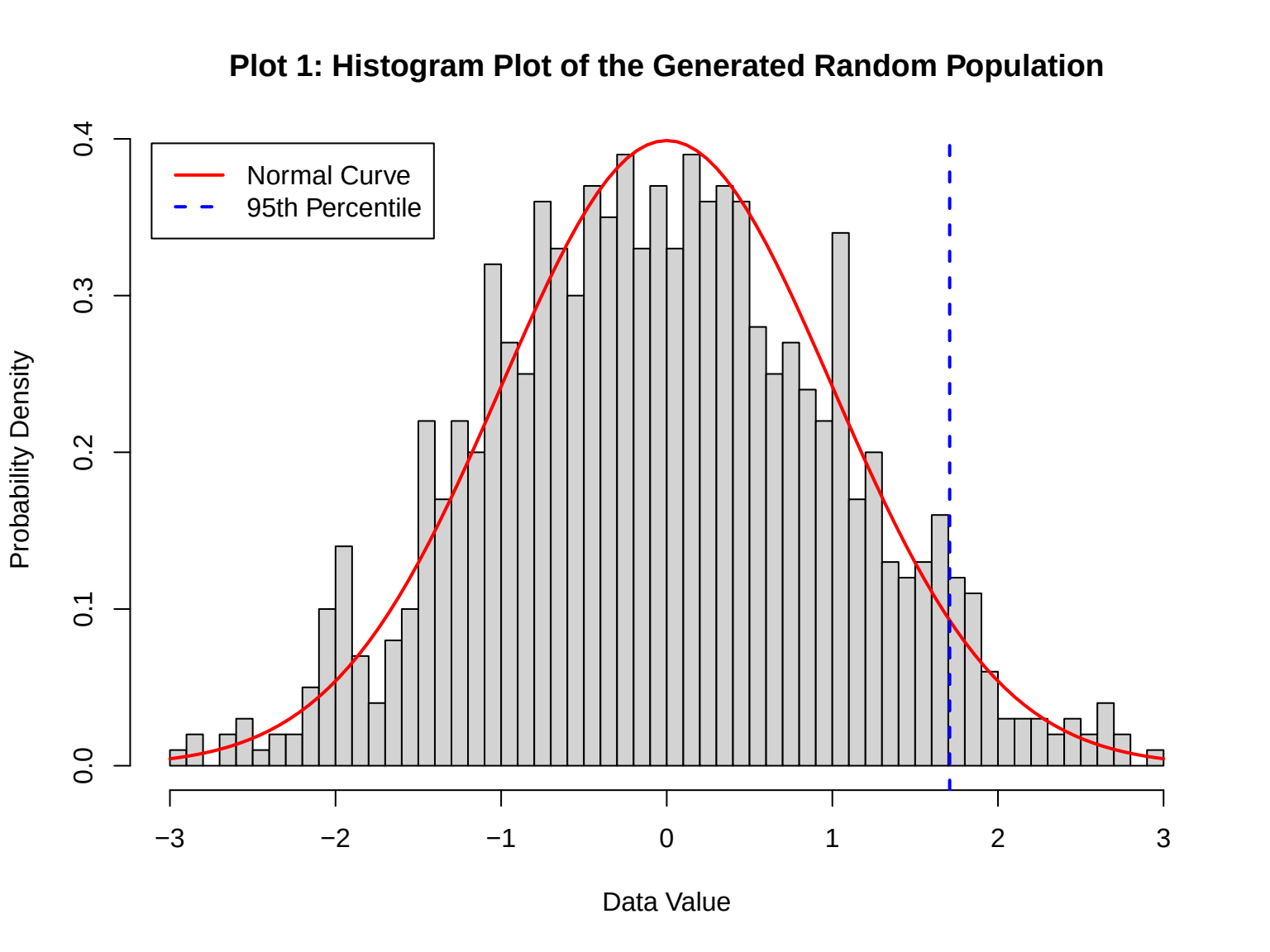
Transitioning from the for loop to perform the Monte Carlo simulation for each seed, the third and final phase of my approach was to visualize my results from the Monte Carlo simulations for each seed through a series of plots exported to a PDF file. Within the third phase, the first step was to create and name the PDF file and to define the colors used for each seed for plotting. The second step was to generate a histogram plot of the normal distribution, with the theoretical normal curve and estimated 95th percentile overlaid on the plot. The third step was to generate a plot of the estimated means for each seed (y-axis) over the number of number of iterations (x-axis). The fourth step was to generate a plot of the standard deviation of the estimated means for each seed (y-axis) over the number of iterations (x-axis). The fifth step was to generate a plot of the estimated 95th percentiles for each seed (y-axis) over the number of iterations (x-axis). The sixth step was to generate a plot of the standard deviation of the estimated 95th percentiles for each seed (y-axis) over the number of iterations (x-axis). The purpose of generating these four convergence plots was to assist in visualizing the uncertainties present in the estimated means and 95th percentiles. The primary goal was to 1) observe how the estimated mean and estimated 95th percentile values for each seed converged toward their expected values as the number of iterations of the Monte Carlo simulation increased, and 2) to observe how the standard deviation values decreased towards zero as the number of Monte Carlo simulations increased. As the seventh and final step of the third phase of my approach, I also printed 1) the average estimated mean, 2) the final standard deviation of the estimated mean, 3) the average estimated 95th percentile, and 4) the final standard deviation of the estimated 95th percentile for each of the five seed values onto the last few pages of the PDF file. This concludes a step-by-step explanation of my code structure and approach for addressing question 4A.

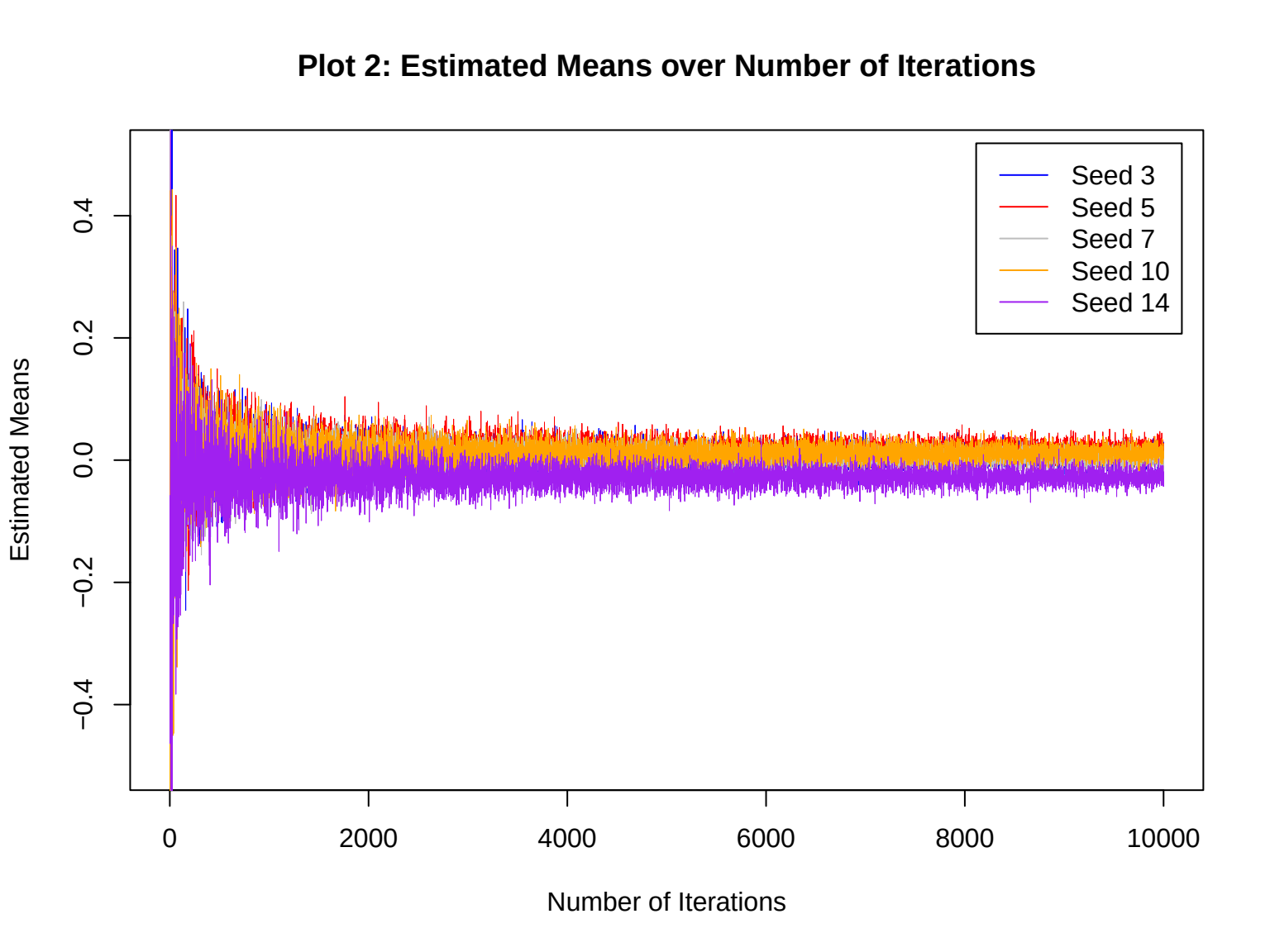
ASSUMPTIONS

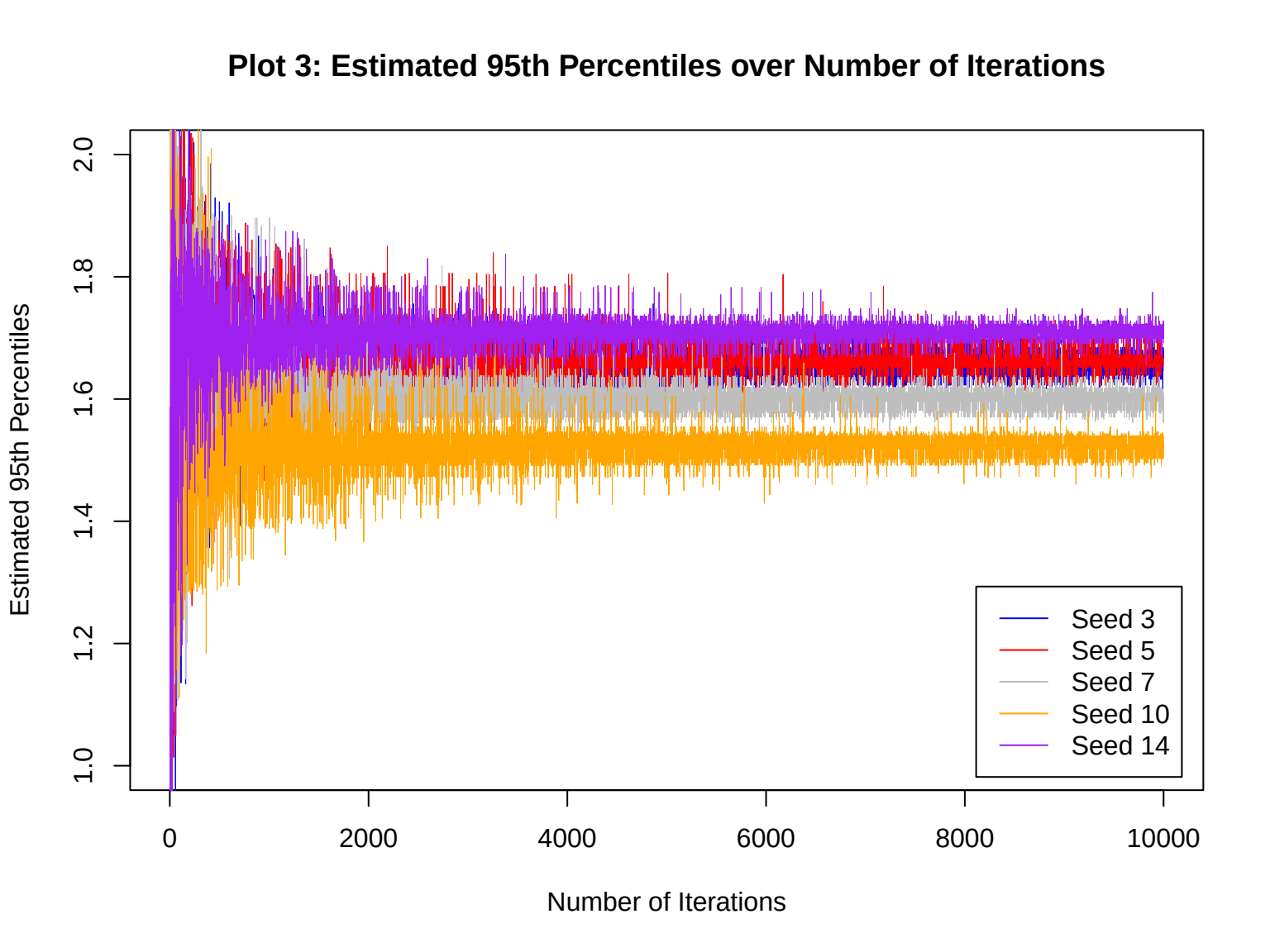
In terms of assumptions, I made several key assumptions in the process of writing the code script in R to solve question 4A. One assumption that I made was that the underlying distribution of the random population was assumed to be normally distributed with a mean = 0 and a standard deviation = 1. In terms of selecting seeds, a second assumption that I made was that selecting only five different seed values (3, 5, 7, 10, 14) was a sufficient number of seeds to examine the impact of reproducibility and controlled randomness on the Monte Carlo simulation results for the estimates of the means and 95th percentile values. In terms of establishing the convergence tolerance as a stopping criterion, a third assumption that I made was that setting the threshold to 0.005 for the standard deviation of the last 100 estimated values for the mean and 95th percentile values was an appropriate threshold value that would enable convergence to occur prior to the maximum number of iterations (10,000) being reached. It also assumes that convergence can be properly determined based upon standard deviation stability, which may not be the best criterion to determine convergence. Extending upon this concept, a fourth assumption that I made was that 10,000 iterations was a large enough upper bound or limit for the number of iterations for the Monte Carlo simulation that would enable convergence to occur based upon the stopping criterion before the maximum number of iterations were reached. In other words, it assumed that 10,000 iterations was a sufficiently large enough number of iterations of the Monte Carlo simulation to produce statistically meaningful estimates of the mean and 95th percentile values.

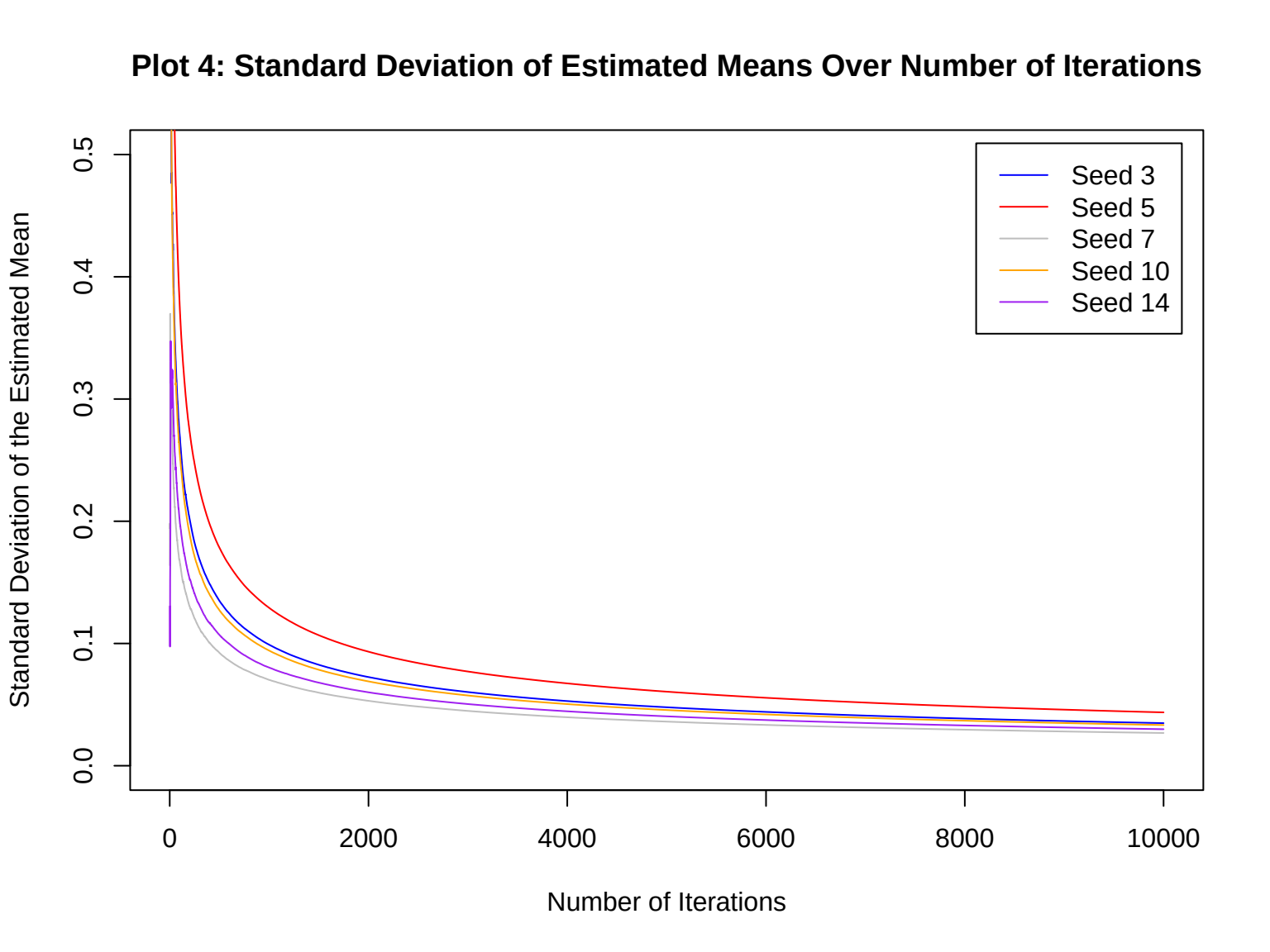
Several assumptions were also made in the process of writing the code to setup the Monte Carlo simulation process. A fifth assumption that I made was that sampling with replacement within the Monte Carlo simulation for loop was the best approach for estimating the mean and 95th percentile values compared to sampling without replacement. Sampling with replacement allows for values from the random population to be selected more than once within a given sample. This is important and can impact estimates for the mean and 95th percentiles compared to sampling without replacement, as it ensures that each iteration is independent and does not reduce the size of the available dataset. A sixth assumption that I made was that calculating the rolling standard deviations of the estimated means and 95th percentiles was an appropriate measure of convergence and an appropriate approach for assessing the uncertainty of these estimates. A seventh assumption that I made was that calculating the average estimated mean, the final standard deviation of the estimated mean, the average estimated 95th percentile, and the final standard deviation of the estimated 95th percentile was a proper and sufficient set of results to extract and analyze from the Monte Carlo simulation results for each seed. An eighth assumption that I made was that the stopping criterion for convergence would not be met within the first 1,000 iterations of the Monte Carlo simulation for each seed, as I wrote the code in a manner such that it would not check for the meeting of convergence criteria prior to the first 1,000 iterations of the simulation being performed.

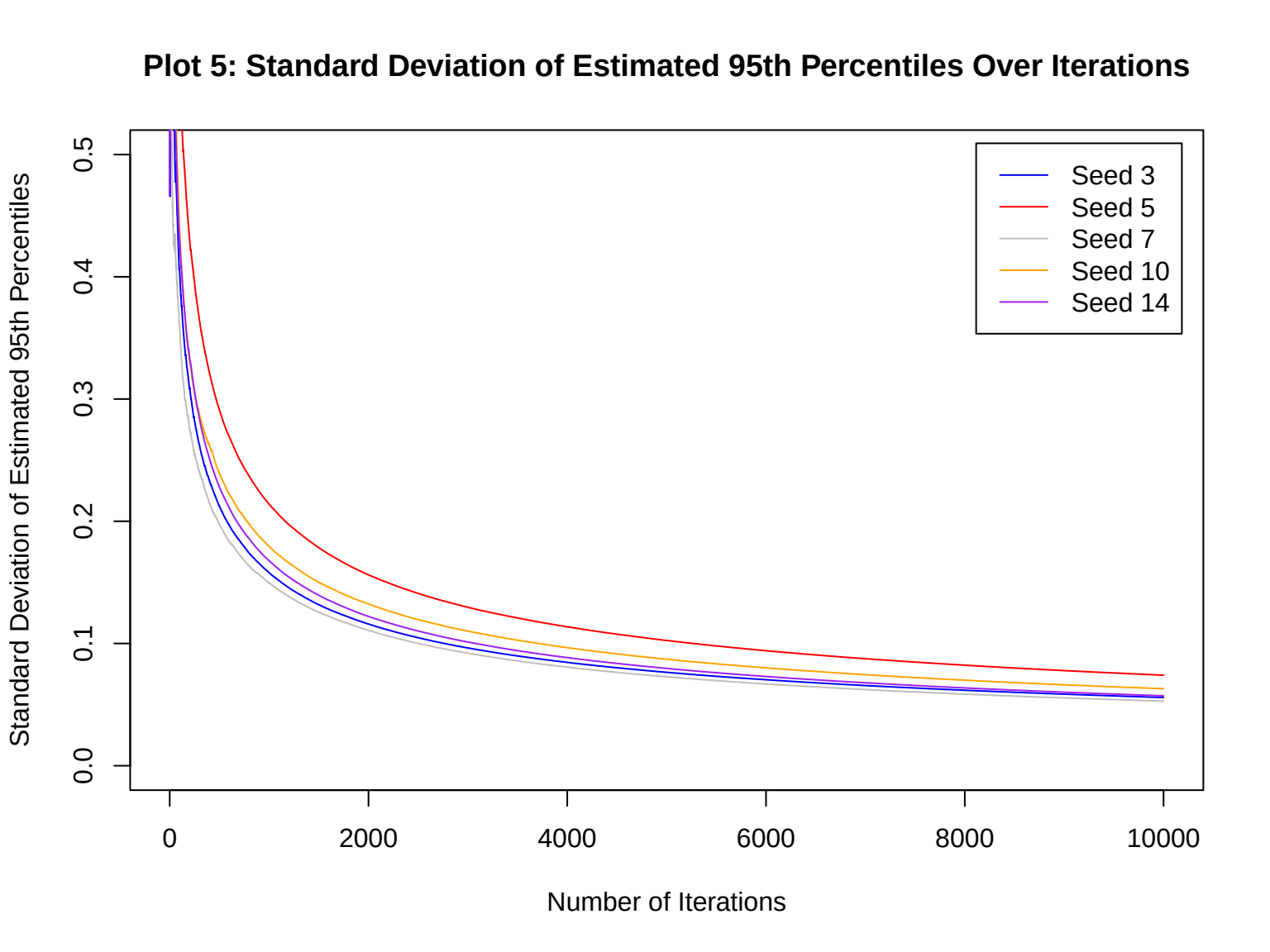
In terms of graphically presenting the results, a ninth assumption that I made was that dividing the histogram plot into 50 bins (using ‘breaks = 50’ within the code script) was sufficient for providing an accurate visualization of the probability density, while not omitting important features of the data, or introducing too much background noise into the plot. A tenth assumption that I made was that the fonts, font sizes, and colors selected for producing the plots were appropriate and facilitated optimal readability and comprehension of the results for all viewers.


Plot 1: Histogram Plot of the Generated Random Population
Normal Curve
95th Percentile
Probability Density
0.0
0.1
0.2
0.3
0.4
−3
−2
−1
0
1
2
3
Data Value


Plot 2: Estimated Means over Number of Iterations
Seed 3
Seed 5
Seed 7
Seed 10
Seed 14
Estimated Means
0.4
0.2
0.0
−0.2
−0.4
0
2000
4000
6000
8000
10000
Number of Iterations


Plot 3: Estimated 95th Percentiles over Number of Iterations
Estimated 95th Percentiles
1.0
1.2
1.4
1.6
1.8
2.0
0
2000
4000
6000
8000
10000
Number of Iterations
Seed 3
Seed 5
Seed 7
Seed 10
Seed 14


Plot 4: Standard Deviation of Estimated Means Over Number of Iterations
Standard Deviation of the Estimated Mean
0.0
0.1
0.2
0.3
0.4
0.5
0
2000
4000
6000
8000
10000
Number of Iterations
Seed 3
Seed 5
Seed 7
Seed 10
Seed 14


Plot 5: Standard Deviation of Estimated 95th Percentiles Over Iterations
Standard Deviation of Estimated 95th Percentiles
0.0
0.1
0.2
0.3
0.4
0.5
0
2000
4000
6000
8000
10000
Number of Iterations
Seed 3
Seed 5
Seed 7
Seed 10
Seed 14

Monte Carlo Simulation Results

Seed: 3
Average Estimated Mean: 0.0071
Final Standard Deviation of the Estimated Mean: 0.0348
Average Estimated 95th Percentile: 1.6556
Final Standard Deviation of the Estimated 95th Percentile: 0.0558

Seed: 5
Average Estimated Mean: 0.0174
Final Standard Deviation of the Estimated Mean: 0.0436
Average Estimated 95th Percentile: 1.6618
Final Standard Deviation of the Estimated 95th Percentile: 0.0741

Monte Carlo Simulation Results

Seed: 7
Average Estimated Mean: 0.0032
Final Standard Deviation of the Estimated Mean: 0.0268
Average Estimated 95th Percentile: 1.601
Final Standard Deviation of the Estimated 95th Percentile: 0.0528

Seed: 10
Average Estimated Mean: 0.0111
Final Standard Deviation of the Estimated Mean: 0.0333
Average Estimated 95th Percentile: 1.5211
Final Standard Deviation of the Estimated 95th Percentile: 0.063

Monte Carlo Simulation Results

Seed: 14
Average Estimated Mean: −0.0283
Final Standard Deviation of the Estimated Mean: 0.0299
Average Estimated 95th Percentile: 1.7068
Final Standard Deviation of the Estimated 95th Percentile: 0.0572

CHOICES

In the process of writing the code script in R to solve question 4A, several important choices were made, and many plausible alternative choices were considered for each decision. One choice that I made was to use the basic plotting functions in R and to import the grid package library into R to add the results summary text for each seed at the end of the PDF file. A plausible alternative choice could have been to import the ggplot2 package that can be used to produce customizable plots. However, I decided to use the grid package library and the basic plotting functions in R because I thought that this option was more straightforward and intuitive as someone who is still becoming familiarized with the coding environment in R.

A second choice that I made was to select and set five different fixed seed values (i.e., 3, 5, 7, 10, 14) to facilitate controlled variability and reproducibility in the Monte Carlo simulation results. In terms of plausible alternatives, a larger set of predefined, fixed seed values (i.e., 10 values instead of 5 values) could have been set to improve the statistical robustness of the analysis. However, setting a larger number of fixed seeds would have increased the computation time and made the analysis unnecessarily complicated. Another plausible choice could have been to set incrementally increasing seed values (i.e., 10, 20, 30, 40, 50) instead of fixed random integer seed values to facilitate a more structured or systematic analysis. Another plausible choice could have been to select a different set of five different fixed seed values than the ones selected. A less plausible choice could have been to set no seed at all. This would result in purely random simulations and no reproducibility in the results could be achieved. Despite all of these alternative choices, I decided to pick a fixed set of five different, random, and small seed values for this problem as I wanted to be able to observe the nuances in results between small random integer seed values and because I wanted to enable reproducibility of my results for anyone running my code script.

A third choice that I made was to set the Monte Carlo simulation stopping criteria such that simulation stops if both the standard deviations of the estimated mean and the 95^{th} percentile values fluctuate within a threshold of 0.005 over the last 100 iterations of the simulation. Another plausible choice could have been to increase the tolerance to a value such as 0.05 to achieve faster, but less precise results for the estimates of the mean and 95^{th} percentile. Another plausible choice could have been to decrease the tolerance to 0.0005 to achieve more precise estimates, but this would result in longer computational times. In light of the choice (discussed next) to set the maximum number of iterations for the Monte Carlo simulation to 10,000, I chose to set the convergence tolerance to 0.005, as I thought that the stopping criterion would be met prior to reaching the maximum or upper limit of iterations set at 10,000.

A fourth choice that I made was to set the maximum number (or upper bound or limit) of Monte Carlo simulations (iterations) to 10,000. Other plausible choices for the maximum number of Monte Carlo simulations could have been to set the number of iterations to 5,000 or 50,000. Increasing the upper limit of Monte Carlo simulations to 50,000 from 10,000 may have led to the stopping criteria being reached before reaching the maximum number of iterations, indicating that convergence may have occurred for the estimates. The problem with this choice is that setting the maximum number of iterations to 50,000 would potentially prolong the computational time or lead to an infinite loop when running the Monte Carlo simulations. Reducing the

maximum number of iterations for the simulation from 10,000 to 5,000 would result in potentially faster computation times, but likely would have not allowed for convergence to occur prior to the maximum number of iterations being reached. Thus, I decided to set the maximum number of Monte Carlo iterations to 10,000, as I thought that selecting this upper limit may allow for convergence to occur prior to reaching the maximum, while also not significantly prolonging the computational time to run the Monte Carlo simulation for each of five seeds.

A fifth choice that I made was to run the Monte Carlo simulation using a for loop. To execute the Monte Carlo simulation, other plausible choices could have been to use the replicate function as a vectorized approach or to use a matrix-based approach. The replicate function executes a function many times and subsequently stores the results in a vector or a matrix. The replicate function avoids looping and internally optimizes execution, which makes it faster than running a loop in the code script. However, the replicate function uses more computational memory than a for loop because it stores intermediate results before extracting them. A matrix based approach is useful for large-scale simulations, is faster than loops, and is optimized for R. A matrix based approach may have been the most plausible choice for running the Monte Carlo simulation for each of the five seeds, but I am not as familiar with writing the code for a matrix operation, so I opted to use a for loop since I was most familiar with setting up a for loop for the simulations.

A sixth choice that I made was to sample with replacement when performing the for loop for the Monte Carlo simulation for each seed. An alternative to this choice could have been to sample without replacement. In sampling without replacement, once a value is selected, it cannot be selected again during that iteration, which can limit the sample size. Thus, I decided to performing sampling with replacement because it enables resampling and ensures that each sample set is always the correct size, which I thought was important for this type of simulation.

A seventh choice that I made was to use the standard deviation of the estimates for the mean and 95th percentiles over the last 100 iterations as my approach for determining convergence (along with the stopping criteria discussed earlier). One plausible alternative choice could have been to use confidence intervals (i.e., using '(t.test())') to determine if the results remained within a predefined bound or threshold. Another potentially plausible choice could have been to use moving averages instead of standard deviations to determine convergence. Since I was already intending to calculate the standard deviation of the estimated means and 95th percentiles to plot over the number of iterations (to observe a decline in the standard deviation values as the number of iterations increased), I decided to determine convergence using the standard deviation of the estimates over the last 100 iterations for two reasons. First, standard deviation is a measure of dispersion. If the variability in the estimated values becomes very small (i.e., within the threshold of 0.005) over a large number of iterations, this is indicative of stability and convergence. Second, I thought using the last 100 iterations of the simulation as the range of iterations for assessing the fluctuation in the standard deviation estimates provided a sufficient number of iterations to assess stability that would prevent premature stopping due to short term fluctuations in the standard deviations (i.e., if the number of iterations was reduced to 50 instead of 100). For these reasons, I decided to determine convergence using the standard deviation of the estimates for the mean and 95th percentiles over the last 100 iterations.

An eighth choice that I made was to set the number of iterations for the convergence check to 1,000 within the for loop for the Monte Carlo simulation for each seed. One plausible alternative choice could have been to perform the convergence check at an earlier stage in the Monte Carlo simulation, such as at 100 or 500 iterations instead of 1,000. This choice may have saved computation time and may have caught convergence earlier on in the simulation. Conversely, I also could have also increased the convergence check to after 2,500 iterations of the simulation. Despite these different choices, I decided to set the number of iterations for the convergence check to 1,000 because I thought that it would be unlikely for the estimates to converge prior to reaching 1,000 iterations and because I thought that extended the number of iterations prior to the convergence check to a larger number such as 2,500 was too long into the simulation and may have resulted in convergence occurring prior to the convergence check.

A ninth choice that I made was to store the results in a data frame using a list of lists format with truncated vectors for estimated values. A plausible alternative to this choice could have been to use matrices or arrays to store the results. However, this may have been the best choice if I had decided to use a matrix operation for performing the Monte Carlo simulations for each seed. I decided to use the data frame approach as I was most familiar with this approach and because I decided to use a for loop to execute the Monte Carlo simulations for each examined seed.

In terms of generating a random population using the rnorm function, a tenth choice that I made was to set the number of random values drawn from the normal distribution to 1000. In terms of plausible alternatives, I could have set the sample size to a fixed value of 500 or 10,000. However, a smaller sample size of 500 may have introduced more randomness that would negatively impact the estimates for the mean and 95th percentile values from the Monte Carlo simulations. Conversely, a larger sample size of 10,000 may have provided more precise estimates for these values due to law of large numbers but may have taken more computational time to perform the simulations for each seed. After doing some experimenting with the size of the random population, I decided to set it to 1000 as I observed that this was producing relatively stable results in a reasonable amount of computation time.

Additional, smaller choices were also made during the process of writing the code in R for producing the histogram and convergence plots. These many small choices included the selection of plot colors, labels, plot titles, axes titles, line width, line style, font etc. Of these many choices, one choice that I wanted to highlight was the choice of selecting the number of bins for the histogram plot. I made the choice of dividing the histogram plot data into 50 bins by specifying in the code: '*breaks = 50'*. Other plausible choices could have included dividing the histogram data into 30 bins or 100 bins. Decreasing the number of bins to 30 would have simplified the histogram plot but could have obscured finer details in the distribution. Conversely, dividing the data into 100 bins could have revealed finer details in the distribution, but could have led to too much noise. In consideration of these alternative choices, I decided to set the number of bins for the histogram plot to 50, as I felt that this was a good balance in presenting the data that revealed both some of the finer details in the distribution, while not obscuring the results with noise. Collectively, documented in the paragraphs above is a summary of the key choices, (and an assessment of their plausible alternatives) that I made while working to solve question 4A.

ASSESSING UNCERTAINTY

Per two class-wide discussions with Dr. Keller during the lecture hour for ENGS 107, it was discussed with the students that the two primary sources of uncertainty present in this analysis were: 1) the small sample size and 2) the choice of the seed. Both features are difficult sources of uncertainty to quantify for question 4A. The small sample size is a source of uncertainty because an initially generated random population with only 1,000 data points (or even a larger number of data points) may not fully representing the true underlying normal distribution and can introduce bias into the subsequent sampling from the generated random population. The choice of the seed value is also a source of uncertainty in this problem because the selection of a specific, fixed seed value, (i.e., a selected seed of 3 compared to a seed of 4 or 8), may impact the estimated mean and 95th percentile values and the convergence behavior over the number of iterations. In other words, selecting different seeds can lead to variations in the results of the simulations.

To account for the uncertainty of the small sample size, I decided to set the initially generated random population to 1000 random samples in an effort to adequately and reasonably represent the true underlying normal distribution. I could have increased the random sample size for the initially generated random population to 5,000 or 10,000, but this may have prolonged the computation time for performing the Monte Carlo simulations for each of the five seeds. Thus, in consideration of the balance of 1) the size of the random population, 2) the uncertainty associated with representing the true underlying normal distribution, and 3) the computation time, I decided to set the initially generated random population to 1000 random samples.

To account for the uncertainty of the choice of seed, I decided to select a relatively small number of fixed seeds (i.e., 5 different seeds) with the intention of introducing independent, controlled randomness across the different Monte Carlo simulation runs. My primary approach in this regard was to experiment with setting different, random fixed seed values and observing the impact of the different seed selections on the estimated mean and 95th percentile values. I ultimately decided to set the seeds to 3, 5, 7, 10, and 14, as these values (along with the influence of the sample size and the number of iterations), resulted in relatively close values of the averaged estimated means and average estimated 95th percentile values to their expected values.

In terms of quantifying uncertainty, my primary approach for question 4A was to iteratively calculate the standard deviations for the estimated means and estimated 95th percentile values over the number of iterations for the Monte Carlo simulation for each of the five seeds (that integrated my choices for the initially generated random population size and the seed choice described above). Uncertainty is often very difficult to estimate and quantify, and this problem is no exception to that case.

REPRODUCIBILITY OF THE ANALYSES

Based upon a few distinct features integrated into the code script for question 4A, the analyses for question 4A are reproducible. One primary feature of the code script that facilitates reproducibility of the Monte Carlo simulation results is setting five different, fixed seed values (i.e., 3, 5, 7, 10, 14). Using fixed seed values results in the Monte Carlo simulation producing the

same results each time that it is run for a specific seed value. Examining the simulation results over five different fixed seed values allows for the observation of controlled randomness.

A second feature integrated into the code script that facilitates the generation of reproducible results is the use of sampling with replacement. This sampling approach guarantees that the distribution characteristics will remain consistent over all of the Monte Carlo simulations (iterations) for a specific seed. A third feature integrated into the code script that facilitates the generation of reproducible results arises from keeping the mean and standard deviation values for the normal distribution fixed at 0 and 1, respectively.

A fourth feature integrated into the code script that facilitates the generation of reproducible results is the defining of specific convergence criteria (i.e., stopping the Monte Carlo simulation when the standard deviations of the estimated means and 95^{th} percentiles are within a threshold of 0.005 for the last 100 iterations) and the specification of an upper limit or bound for the maximum number of Monte Carlo simulation iterations at 10,000. These inclusions into the code script provide an additional layer of control over the simulations for each seed, thus reducing any variability in the results.

b. Determine the value of pi with your estimated uncertainties

TASK

The task outlined in question 4B requires implementing a Monte Carlo simulation method to estimate the value of pi (π) using our estimated uncertainties. The Monte Carlo simulation should aim to estimate the value of pi by simulating random points within a unit square and subsequently determining the proportion of random points that fall inside of a unit circle. Different seed values should be selected and tested to introduce controlled randomness into the analysis, while also enabling reproducibility of the results each time that the Monte Carlo simulation is performed for a specific seed. Within the Monte Carlo simulation, the value of pi and its associated standard deviation value should be calculated for each iteration of the simulation. A convergence tolerance and a set of stopping criteria should be established and monitored within the code script to assess if the estimated value of pi has sufficiently converged to its true value (3.1415…) up to a predefined maximum number (or threshold) of iterations of the Monte Carlo simulation to prevent infinite loops or unnecessarily long computation times. A primary goal of this task should be to observe how 1) the estimated value of pi converges towards its true value and 2) its associated standard deviations decrease as the number of Monte Carlo simulations increase across each seed. The deliverables for this task should include recording the averaged estimated value of pi and the final standard deviation of the estimated value of pi for each of the seeds. In terms of graphically presenting the findings, two different plots should be produced. First, the estimated pi values for each seed should be plotted over the number of iterations. Second, the standard deviations of the estimated pi values for each seed should also be plotted over the number of iterations. Collectively, the results should clearly state the calculated average estimated value of pi (in consideration of the uncertainties) for each seed.

APPROACH

My approach for accomplishing the task described above for question 4B was similar in scope and structure to my approach outlined for question 4A. My focus was to produce a code script in R that followed a logical workflow of phases composed of smaller incremental steps to establish the environment for performing the Monte Carlo simulations, executing the Monte Carlo simulations for each seed, performing the necessary calculations, and graphing the results.

The first phase of my approach in writing the code script was focused on setting up the framework for the Monte Carlo simulations. This phase included importing a grid package library to enable enhanced graphical functions in R for use in subsequent use in recording the results and defining five different random seed values (3, 5, 7, 10, 14 [the same values as 4A]) to introduce controlled randomness (variability) and to ensure reproducibility for each seed across different runs of the Monte Carlo simulation.

The second phase of my approach was focused on establishing the Monte Carlo simulation parameters. This phase included defining the convergence criteria to stop the Monte Carlo simulation if the estimated values of pi fluctuated within a threshold of 0.005 over the last 100 iterations. This phase also included defining the maximum number (upper limit) of Monte Carlo simulation iterations at 10,000 iterations to prevent infinite loops or long computation times. Please note that the convergence criteria threshold and maximum number of iterations for the Monte Carlo simulation were not modified for question 4B from the settings in 4A.

The third phase of my approach was focused on preparing for subsequent plotting of the results from the Monte Carlo simulation. This phase included defining the colors for each of the seed values. Please note that these colors were kept the same from the colors used for each of the seed values in question 4A. I also created a PDF file to save the results from the simulations.

The fourth phase of my approach was focused on setting up a for loop to perform the Monte Carlo simulation for each of the five seed values. In a similar manner to question 4A, the for loop that I wrote within the code script is somewhat complicated and a description of each of the primary steps within the for loop is described below. The first step was to set the seed within the for loop to ensure that each simulation run with that specific seed would produce reproducible results. The second step was to initialize numeric vectors to store the estimated pi values and the associated standard deviation values. A third step was to initialize control variables to track if the stopping criteria had been met and to also count the number of Monte Carlo simulations. These same control variables were set up within the for loop for 4A as well. The fourth step was focused on setting up a while-loop within the for loop to run the Monte Carlo simulation until either the convergence criteria were met, or the maximum number of 10,000 iterations had been reached for each seed. Within the while-loop, a fifth step was focused on randomly generating x and y coordinates with a unit square [-1,1] x [-1,1]. A sixth step was to determine if each of the randomly generated points fell within the unit circle by checking if: $x^2 + y^2 \leq 1$. A subsequent seventh step was to estimate the value of pi using the formula: $\pi \approx (4 *$ [points inside the circle])/(total points generated). Following the estimation of the value of pi, an eighth step was to calculate the standard deviation of the pi estimates for each iteration. A ninth step was to set up an if statement within the while-loop to check for convergence after each set of

1,000 iterations of the Monte Carlo simulation for each seed. If the last 100 estimates of the value of pi fluctuated within the pre-defined threshold of 0.005, the simulation would stop early. If not, the simulation would continue to proceed until the maximum number of iterations (10,000 iterations) were reached for a specific seed. A tenth step was to store the final results of the simulation for each seed in a structured list for subsequent analysis. These stored results included the total number of iterations run, the seed value, the estimated values for pi for each iterations, and the standard deviations of the estimated values of pi for each iteration. The eleventh and final step within the for loop was to close the for loop to allow for the loop to repeat the same process (steps 1-11 of phase 4) again for the next fixed seed value.

Transitioning from the for loop to perform the Monte Carlo simulation for each seed to the analysis, the fifth and final phase of my approach was to produce two plots to visualize the results from the Monte Carlo simulations for each seed and to append the key results to the end of the PDF file. The first step of this phase was to produce a plot of the estimated pi values for each seed (y-axis) over the number of iterations (x-axis). One nuance in this step was to define the y-axis range dynamically based upon the estimated pi values to ensure that the y-axis was large enough to be able to visualize all of the estimated pi values for each of the seeds. I also set up a for loop to iterate through the stored results for each seed and to plot the results. I also created a legend and included a dashed black reference line to denote the true value of pi. The second step of this phase was to produce a plot of the standard deviations of the estimated pi values for each seed (y-axis) over the number of iterations (x-axis). In a similar manner to the first plot, I also set up a for loop within the production of this second plot to loop through the stored results and plot the results for each of the seeds. I also included a legend to specify which seeds correspond to which colors. The third and final step of this phase was to calculate and add the average estimated pi values and the final standard deviation of the estimated pi values for each seed to a page at the end of the PDF file. This included setting up a for loop to loop through the results in each seed to calculate the key results and to append them to the PDF file page. This concludes a step-by-step explanation of my code structure and approach for question 4B.

ASSUMPTIONS

In terms of assumptions, I made several key assumptions in the process of writing the code script in R to solve question 4B. In terms of selecting seeds, one assumption that I made was that selecting only five different seed values (3, 5, 7, 10, 14) was a sufficient number of seeds to examine the impact of both controlled randomness (variability) and reproducibility on the Monte Carlo simulation results for estimating the value of pi. In other words, I am assuming that selecting different fixed seeds will lead to different results, while re-running the Monte Carlo simulation for a specific seed will always produce the same results.

In terms of establishing the convergence tolerance as a stopping criterion, a second assumption that I made was that if the last 100 estimated pi values fluctuate within the defined convergence threshold of 0.005, then the estimates have effectively converged and that the simulation can be stopped prior to reaching the maximum number of iterations of the Monte Carlo simulation. Building upon this same concept, a third assumption that I made was that measuring fluctuations in the estimated pi values within a threshold was an appropriate and reliable metric for determining convergence. A fourth assumption that I made also pertaining to convergence was

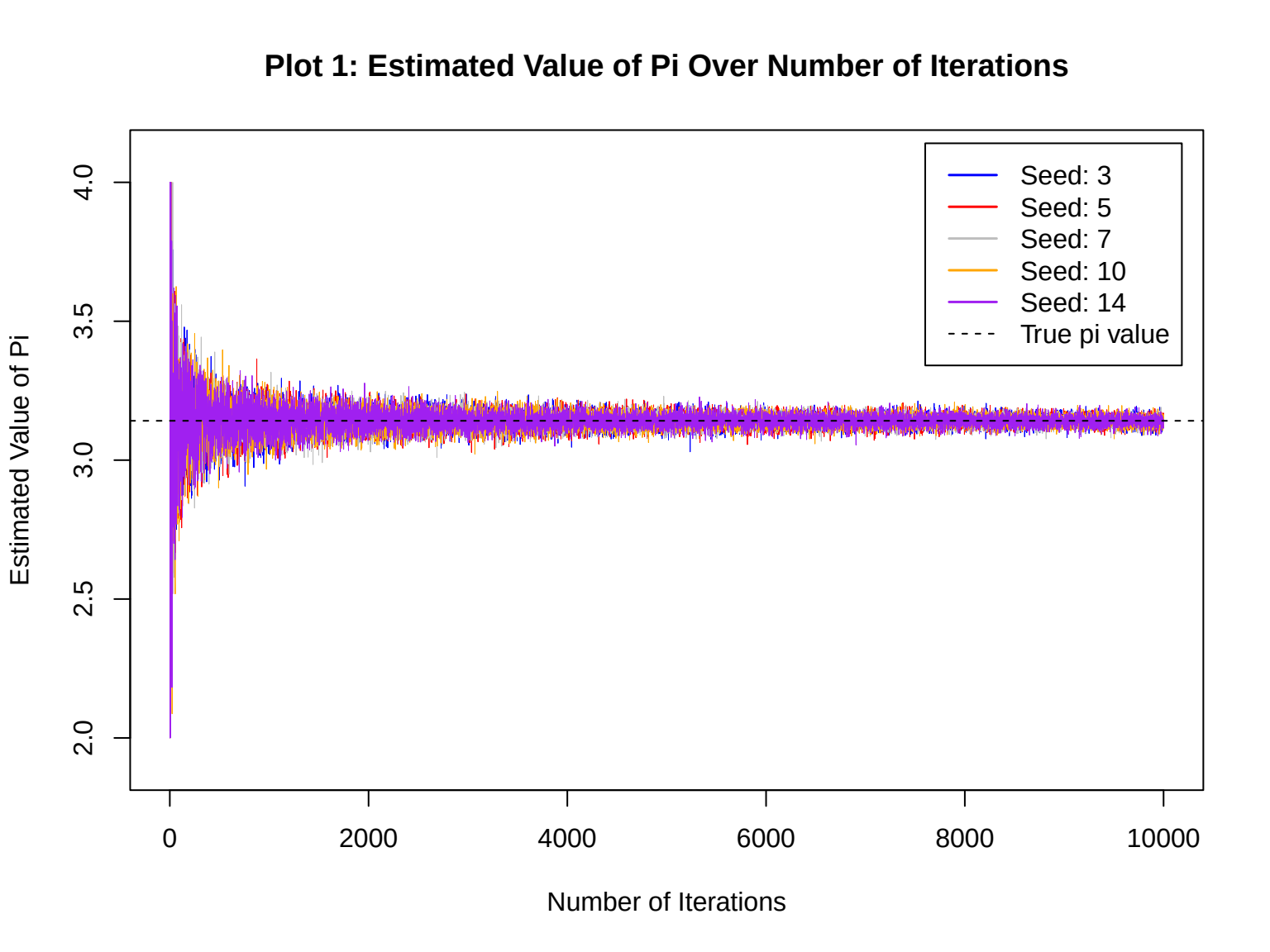
that examining the last 100 iterations was a sufficient number of iterations to consider to determine if convergence had been achieved. A fifth assumption that I made also pertaining to convergence was that 10,000 iterations was a large enough upper bound for the number of iterations for the Monte Carlo simulation that would allow for convergence to occur before reaching this limit. As a sixth assumption, I also assumed that if the stopping criteria were not met, that 10,000 iterations of the simulation was a sufficient number of iterations for an accurate estimate for the value of pi for each seed. In other words, I assumed that 10,000 iterations was a large enough number of iterations to produce statistically meaningful estimates of the value of pi.

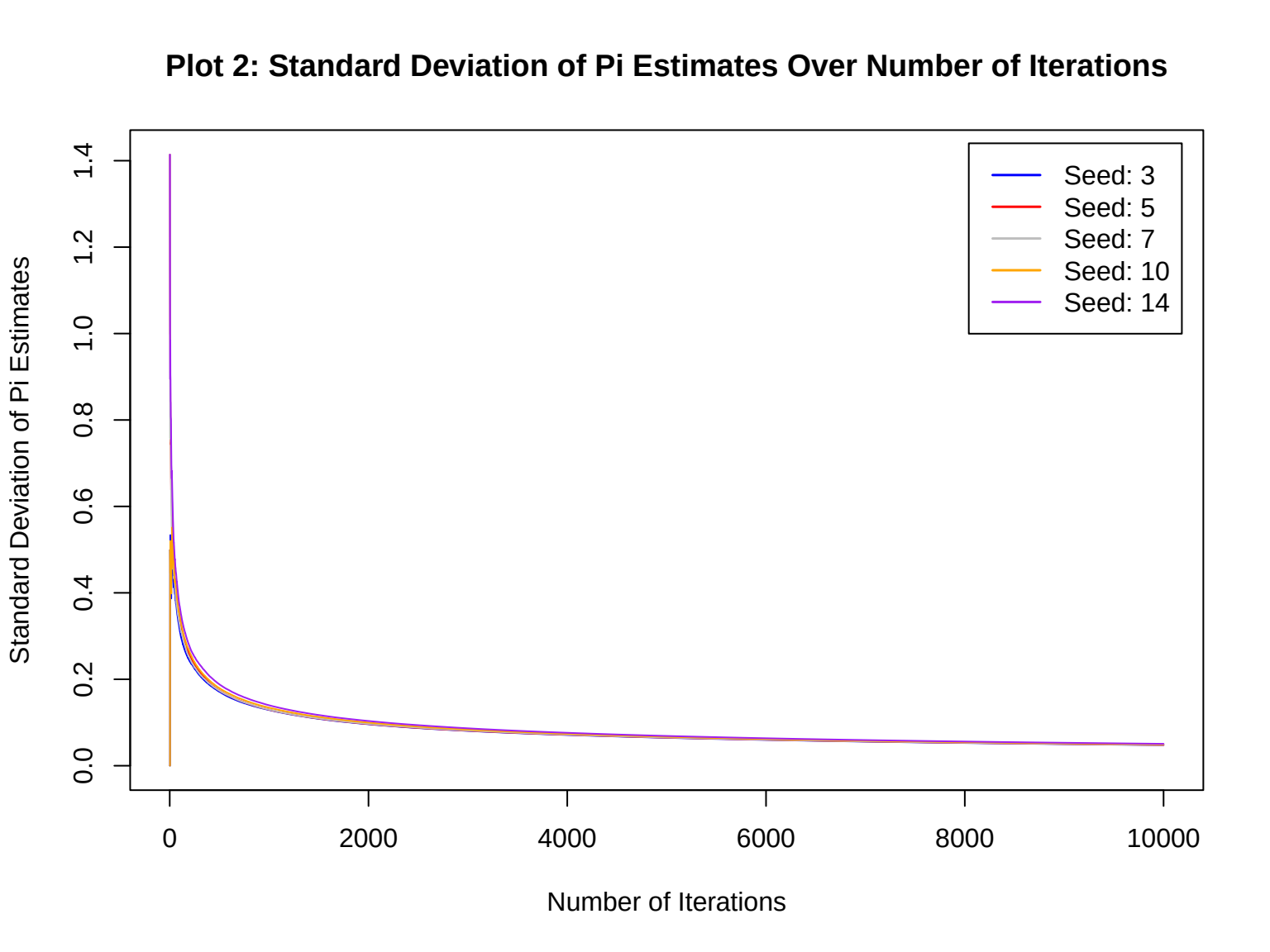
Several assumptions were also made in the process of writing the code to setup the Monte Carlo simulation process. A seventh assumption that I made was that generating random x, y coordinate points using the runif() function produced a truly uniform distribution within the unit square of [-1,1] x [-1,1], that ensured an unbiased estimate for the value of pi. An eighth assumption that I made was that the formula used to estimate the value of pi (i.e., $\pi \approx (4 * [\text{points inside the circle}])/(\text{total points generated})$ was appropriate and sufficiently accurate enough to estimate the value of pi relative to its true value as the number of iterations increased.

Returning to the concept of stopping criteria, a ninth assumption that I made was that the stopping criterion for convergence would not be met within the first 1,000 iterations of the Monte Carlo simulation for each seed, as I wrote the code in a manner such that it would not check for the meeting of convergence criteria prior to the first 1,000 iterations of the simulation.

A tenth assumption that I made was that calculating the average estimated pi value and the final standard deviation of the estimated pi value was a proper and sufficient set of results to extract and analyze from the Monte Carlo simulation for each seed.

In terms of graphically presenting the results, an eleventh assumption that I made in the process of writing the code to produce the first plot (the estimated values of pi for each seed [y-axis] over the number of iterations [x-axis]), was that computing the minimum and maximum values of all estimated pi values enabled proper visualization of the estimated pi values on the plot without manually setting the bounds for the y-axis. Also in terms of plotting, a twelfth assumption that I made was that the fonts, font sizes, and colors selected for producing the plots were appropriate and facilitated optimal readability and comprehension of the results for all viewers.


Plot 1: Estimated Value of Pi Over Number of Iterations
Seed: 3
Seed: 5
Seed: 7
Seed: 10
Seed: 14
True pi value
Estimated Value of Pi
2.0
2.5
3.0
3.5
4.0
0
2000
4000
6000
8000
10000
Number of Iterations


Plot 2: Standard Deviation of Pi Estimates Over Number of Iterations
Standard Deviation of Pi Estimates
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2000
4000
6000
8000
10000
Number of Iterations
Seed: 3
Seed: 5
Seed: 7
Seed: 10
Seed: 14

MONTE CARLO SIMULATION RESULTS

Seed: 3
Average Estimated Value of Pi: 3.1421
Final Standard Deviation of Estimated Pi value: 0.0477

Seed: 5
Average Estimated Value of Pi: 3.1413
Final Standard Deviation of Estimated Pi value: 0.048

Seed: 7
Average Estimated Value of Pi: 3.1416
Final Standard Deviation of Estimated Pi value: 0.0482

Seed: 10
Average Estimated Value of Pi: 3.142
Final Standard Deviation of Estimated Pi value: 0.0493

Seed: 14
Average Estimated Value of Pi: 3.1416
Final Standard Deviation of Estimated Pi value: 0.0507

CHOICES

In the process of writing the code script in R to solve question 4B, several important choices were made, and many plausible alternative choices were considered for each decision. In a similar manner to question 4A, one choice that I made was to use the basic plotting functions in R and to import the grid package library into R to add the results summary text for each seed at the end of the PDF file. A plausible alternative choice could have been to import the ggplot2 package that can be used to produce customizable plots. However, I decided to use the grid package library and the basic plotting functions in R because I experimented with both and found this option to be more straightforward and intuitive as someone who is not super familiar with R.

In a similar manner to question 4A, a second choice that I made was to select and set five different fixed seed values (i.e., 3, 5, 7, 10, 14) to facilitate controlled variability and reproducibility in the Monte Carlo simulation results. In terms of plausible alternatives, a larger set of predefined, fixed seed values (i.e., 10 values instead of 5 values) could have been set to improve the statistical robustness of the analysis. However, setting a larger number of fixed seeds would have increased the computation time and made the analysis more complicated. Another plausible choice could have been to set incrementally increasing seed values (i.e., 10, 20, 30, 40, 50) instead of fixed random integer seed values to facilitate a more systematic analysis. Another plausible choice could have been to select a different set of five different fixed seed values than the ones selected. A less plausible choice could have been to set no seed at all. This would have resulted in purely random simulations and no reproducibility in the results could have been achieved. Despite these many plausible, alternative choices, I decided to pick a fixed set of five different, random, and small seed values for this problem because 1) I wanted to be able to observe the nuances in the results for the estimated pi values between small random integer seed values and 2) because I wanted to produce reproducible results for each of the examined seeds.

Also in a similar manner to question 4A, a third choice that I made was to set the Monte Carlo simulation stopping criteria such that the simulation would stop if the estimated pi values fluctuated within a threshold of 0.005 over the last 100 iterations of the simulation. Another plausible choice could have been to increase the tolerance to a value such as 0.05 to achieve faster, but less precise results for the estimated pi values. Another plausible choice could have been to decrease the tolerance to 0.0005 to achieve more precise estimates, but this may have resulted in longer computational times. In light of the choice (discussed next) to set the maximum number of iterations for the Monte Carlo simulation to 10,000, I chose to set the convergence tolerance to 0.005, as I thought that the stopping criteria would be met prior to reaching the maximum or upper limit of iterations set at 10,000 for each Monte Carlo simulation.

Also in a similar manner to question 4A, a fourth choice that I made was to set the maximum number (or upper bound or limit) of Monte Carlo simulations (iterations) to 10,000. Other plausible choices for the maximum number of Monte Carlo simulations could have been to set the number of iterations to 5,000 or 50,000. Increasing the upper limit of Monte Carlo simulations to 50,000 from 10,000 may have led to the stopping criteria being reached before reaching the maximum number of iterations, indicating that convergence may have occurred for the estimates. The problem with this choice is that setting the maximum number of iterations to 50,000 would potentially prolong the computational time or produce an infinite loop when

running the Monte Carlo simulations. Reducing the maximum number of iterations for the simulation from 10,000 to 5,000 would result in potentially faster computation times, but may not have allowed for convergence to occur prior to the maximum number of iterations being reached. Thus, I decided to set the maximum number of Monte Carlo iterations to 10,000, as I thought that selecting this upper limit may allow for convergence to occur prior to reaching the maximum number of iterations, while also not significantly prolonging the computational time needed to run the Monte Carlo simulation for each of five different seed values.

Also in a similar manner to question 4A, a fifth choice that I made was to run the Monte Carlo simulation using a for loop. To execute the Monte Carlo simulation, other plausible choices could have been to use the replicate function as a vectorized approach or to use a matrix-based approach. The replicate function executes a function many times and subsequently stores the results in a vector or a matrix. The replicate function avoids looping and internally optimizes execution, which makes it faster than running a loop in the code script. However, the replicate function uses more computational memory than a for loop because it stores intermediate results before extracting them. A matrix based approach is useful for large-scale simulations, is faster than loops, and is optimized for R. A matrix based approach may have been the most plausible choice for running the Monte Carlo simulation for each of the five seeds to estimate the value of pi, but I am not as familiar with writing the code for a matrix operation, so I opted to use a for loop since I was most familiar with setting up a for loop for the simulations.

A sixth choice that I made was to determine convergence for this problem by analyzing if the last 100 estimated pi values in the simulation had remained within a threshold range of 0.005 of their mean. One plausible and feasible alternative choice could have been to use confidence intervals to determine if the results remained within a predefined bound or threshold. Since I was already intending to calculate the average estimated pi value for each of the seeds and plot it over the number of iterations, I felt like the most straightforward approach for determining convergence in this instance was to examine the fluctuations in the estimated pi values over the last 100 iterations of the Monte Carlo simulation for a specific seed. Along this same line of thinking, a seventh choice that I made was to choose the value of 100 as a sufficient number of iterations from which convergence could be determined. An alternative plausible choice could have been to increase this number from 100 to 200 or 500. This would have tightened the criteria and required more iterations to occur with negligible fluctuation in the estimated pi values (i.e., within a threshold of 0.005) for convergence to be determined and for the simulation to stop prior to reaching the maximum number of iterations. Another alternative choice, (but a most likely poorer choice), could have been to decrease the number of required iterations for determining convergence from 100 to 80 or 50. This would not have been a good choice, as it may have caused convergence to occur prematurely, leading to an estimated value of pi for a specific seed that may not have been as close to the true value of pi as it could have been if the number of iterations for determining convergence was increased. Thus, in light of these different factors and considerations, I decided to set the number of iterations for determining convergence to 100, as I thought that not observing any fluctuations in the estimated pi values within a range of 0.005 of the mean over 100 iterations was a sufficient number of iterations to determine convergence.

A seventh choice that I made for question 4B was to use the Monte Carlo method to estimate the value of pi by simulating random points within a unit square, and subsequently checking to

determine what proportion of points fell inside of the unit circle. One alternative choice could have been to use a different statistical technique such writing the code to simulate the Buffon Needle Experiment in R. This simulation produces random needle positions and angles, and subsequently calculates if the needle intersects with a line to estimate the value of pi based upon the probability of intersection. Although somewhat interesting, writing the simulation for the Buffon Needle Experiment into the R script within the for loop of the Monte Carlo simulation seemed unnecessarily complicated. Another alternative choice could have been to use a Gaussian probability distribution instead of a uniform random distribution to estimate pi, but this choice also seems unnecessarily complicated, despite potentially leading to more precise results for the estimate of the value of pi. Due to the seemingly complex nature of these alternative choices, I decided to use the Monte Carlo method to estimate the value of pi by simulating random points within a unit square, and subsequently checking to determine what proportion of the points feel inside of the unit circle as this seemed to be the most straightforward and logical approach.

An eighth choice that I made was to set the number of iterations for the convergence check to 1,000 within the for loop for the Monte Carlo simulation for each seed. One plausible alternative choice could have been to perform the convergence check at an earlier stage in the Monte Carlo simulation, such as at 100 or 500 iterations instead of 1,000. This choice may have saved computation time and may have caught convergence earlier on in the simulation. Conversely, I also could have also increased the convergence check to after 2,500 iterations of the simulation. Despite these different choices, I decided to set the number of iterations for the convergence check to 1,000 because I thought that it would be unlikely for the estimates to converge prior to reaching 1,000 iterations and because I thought that extended the number of iterations prior to the convergence check to a larger number such as 2,500 was too long into the simulation and may have resulted in convergence occurring prior to the convergence check.

Also in a similar manner to question 4A, a ninth choice that I made was to use a while loop as the formatting structure within my code for performing convergence checking. An alternative choice could have been to use a repeat loop with an explicit set of break conditions. I also could have used a vectorized approach using built in functions in R (i.e., cumsum() and filter()), which may have expedited the process of convergence checking. However, since I had already structured my for loop for question 4A to contain a while loop for convergence checking, I decided to maintain this same code structure for question 4B for the sake of consistency.

A tenth choice that I made was to use the list() function to store the results from the Monte Carlo simulation for each seed. An alternative choice could have been to use a data frame. A data frame may have provided more structured storage, but I felt that using the list() function was the most straightforward choice for storing the results. Using the list() function also seemed like the most logical approach because it is structured such that each entry corresponds to a different seed.

A eleventh choice that I made was to use the sapply() function to find the maximum and minimum estimated pi values to dynamically define the y-axis range for the plot of estimated pi values for each seed (y-axis) over the number of iterations (x-axis). Another choice could have been to use the range() function as a single pass (and a potentially more efficient approach) for determining the maximum and minimum estimated pi values. Another alternative choice could have been to manually increase the limits of the y-axis. Although my choice may not have been

the most efficient, I felt that my approach was the most logical and straightforward choice for ensuring that the y-axis scale was large enough to visually display all of the estimated pi values.

Also in terms of plotting the results, an eleventh choice that I made was to plot a black, dashed reference line for the true value of pi on the plot of estimated pi values for each seed (y-axis) over the number of iterations (x-axis). An alternative choice could have been to use a shaded confidence interval. However, I decided to only plot the true value of pi as a dashed black line as I thought that this was the most clear and would not distract from the different seed results. Additional, smaller choices were also made during the process of writing the code script for producing the two plots. These many small choices made in the preparation of these two plots included the selection of colors, labels, plot titles, axes titles, line width, line style, and font etc. Collectively, the paragraphs above provide a summary of the many key choices, (and an assessment of their plausible alternatives) that I made while working to solve question 4B.

ASSESSING UNCERTAINTY

In a similar manner to the discussion above for question 4A concerning the assessment of uncertainty, there are also two major sources of uncertainty in question 4B: 1) the random sampling variability and 2) the choice of the seed. Both features are difficult sources of uncertainty to assess and to quantify for the task of estimating the value of pi using a Monte Carlo simulation. Random sampling variability is a source of uncertainty because using the runif() function to produce random x and y coordinates within the unit square [-1,1] x [-1,1] may lead to variability in the estimated pi values because different random samples may produce slightly different approximations for the value of pi. In a similar manner to question 4A, the choice of seed is also a source of uncertainty in question 4B because the selection of a specific, fixed seed value, (i.e., a selected seed of 3 compared to a seed of 4 or 8), may impact the estimated pi values and the convergence behavior over the number of iterations. In other words, initially selecting different seeds will lead to variations in the Monte Carlo simulation results.

Aside from generating random x and y coordinates within the unit square using the most logical simplistic approach within the Monte Carlo simulation for loop for each seed, no further efforts were made to assess the uncertainty of this source. To account for the uncertainty in the choice of seed, I decided to select a relatively small number of fixed seeds (i.e., 5 different seeds) with the intention of introducing independent, controlled randomness across the different Monte Carlo simulation runs. In a similar manner to question 4A, my primary approach was to experiment with setting different fixed seed values and observing the impact of the different seed selections on the estimated pi values. I ultimately decided to set the seeds to 3, 5, 7, 10, and 14, as these were the values I selected for 4A and because the Monte Carlo simulations for each of these seeds produced average estimated values of pi that were relatively close to the true value of pi.

In terms of quantifying uncertainty, my primary approach for question 4B was to iteratively calculate the standard deviations for the estimated pi values over the number of iterations for the Monte Carlo simulation for each of my five selected seeds. These results are presented in the second plot for question 4B (the plot of standard deviation of the estimated pi values [y-axis] over the number of iterations [x-axis]). Other factors that also contribute to the uncertainty of this problem include the decision of what value to select as an upper limit for the maximum number

of iterations for the Monte Carlo simulation and the decision of what sensitivity threshold to select for the convergence stopping criteria. Collectively, the uncertainty present in question 4B is also difficult to assess and quantify accurately in a similar manner to question 4A.

REPRODUCIBILITY OF THE ANALYSES

Based upon a few distinct features integrated into the code script for question 4B, the analyses for question 4B are reproducible. One primary feature of the code script that facilitates reproducibility of the Monte Carlo simulation results is setting five different, fixed seed values (i.e., 3, 5, 7, 10, 14). Using fixed seed values results in the Monte Carlo simulation producing the same results each time that it is run for a specific seed value. Examining the simulation results over five different fixed seed values allows for the observation of controlled randomness.

A second feature integrated into the code script that facilitates the generation of reproducible results is the use of a defined stopping criterion in which the Monte Carlo simulation iterations stop when the estimated pi values fluctuate within a threshold of 0.005 over the last 100 iterations. These predefined values for the threshold and the number of iterations ensure that the Monte Carlo simulations run in a consistent manner across different executions.

A third feature integrated into the code script that facilitates reproducibility of the results is the use of a defined maximum number of iterations (an upper limit) of 10,000 iterations for the Monte Carlo simulation. This predefined value ensures that the simulation stops at 10,000 iterations if the stopping criterion are not met first. This also ensures consistency in the results.

REFERENCES

1. Venables, W.N., Smith, D.M. & the R Core Team. (2024). *An Introduction to R: Notes on R: A Programming Environment for Data Analysis and Graphics Version 4.4.2*. Retrieved from https://cran.r-project.org/doc/manuals/R-intro.pdf.

2. Verzani, J. (2002). *simpleR – Using R for Introductory Statistics*. Retrieved from https://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf.

3. Culhane, A. (2012). Introduction to Programming in R. Retrieved from https://ds.dfci.harvard.edu/~aedin/courses/R/CDC/Intro2R.pdf.

4. Gräbner-Radkowitsch, C. (2022). Fundamental object types in R III: Factors and Data Frames. Retrieved from https://datascience-euf-spring22.netlify.app/post/object-types-adv/pubdir/pdfcontent.pdf.

5. Gräbner-Radkowitsch, C. (2022). Monte Carlo Simulations in R. Retrieved from https://datascience-euf-spring22.netlify.app/post/mcs/pubdir/pdfcontent.pdf.

6. Cotton, R. (2013). *Learning R: A Step-By-Step Function Guide to Data Analysis*. O'Reilly Media.

7. Grolemund, G. (2014). *Hands-On Programming with R: Write Your Own Functions and Simulations*. O'Reilly Media.

8. Dong, H. & Nakayama, M.K. (2020). *A Tutorial on Quantile Estimation via Monte Carlo*. Retrieved from https://web.njit.edu/~marvin/papers/qtut-r2.pdf.

9. Bulut, O. (2020). *Conducting Monte Carlo Simulations in R*. Retrieved from https://era.library.ualberta.ca/items/41513514-cddc-4b35-9117-bd411c19e12e/view/f45bf1b1-0ae4-4720-b78b-6413e70ec0e4/simulations_in_r.pdf.

10. Sriver, R. & Keller, K. (2024). coin-example.R. Retrieved from https://canvas.dartmouth.edu/courses/70487/files/folder/example%20codes?preview=13458702.

11. Applegate, P.J., Sriver, R.L., Garner, G.G., Bakker, A., Alley, R.B. & Keller, K. (2016). *Risk Analysis in the Earth Sciences: A Lab Manual with Exercises in R* (*Version 1.2)*. SCRiM. Retrieved from https://leanpub.com/raes.

12. Robert, C.P. & Casella, G. (2004). *Monte Carlo Statistical Methods* (2nd ed). Springer.

13. Rizzo, M.L. (2019). *Statistical Computing with R* (2nd ed). Chapman & Hall/CRC.

14. Wickham, H. (2015). *Advanced R*. Chapman & Hall/CRC.

15. Hothorn, T. & Everitt, B.S. (2014). *A Handbook of Statistical Analyses Using R* (3rd ed). Chapman & Hall/CRC.

16. R Core Team. (2024). R: A Language and Environment for Statistical Computing. Retrieved from https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf.

17. Murrell, P. (2005). *R Graphics* (2nd ed). Chapman & Hall/CRC.

18. Rosenberg, M. (2021). *Analytics-Using-R*. Retreived from https://pubs.wsb.wisc.edu/academics/analytics-using-r-2019/.

19. Rubinstein, R.Y. & Kroese, D.P. (2016). *Simulation and the Monte Carlo Method* (2nd ed). Wiley.

20. "Estimating Pi using the Monte Carlo Method." *Academo*. Retrieved from https://academo.org/demos/estimating-pi-monte-carlo/. Accessed 20 January 2025.

21. Ross, S.M. (2006). *Simulation* (4th ed). Academic Press.

22. Chang, W. (2013). *R Graphics Cookbook: Practical Recipes for Visualizing Data*. O'Reilly Media.

23. Gillespie, C. & Lovelace, R. (2016). *Efficient R Programming: A Practical Guide to Smarter Programming*. O'Reilly Media.

24. Matloff, N. (2011). *The Art of R Programming*. No Starch Press.

25. Peng, R.D. (2015). *R Programming for Data Science*. Leanpub.

26. "Buffon's Needle: An Analysis and Simulation." *University of Illinois Urbana-Champaign College of Education Office for Mathematics, Science, and Technology Education*. Retrieved from https://mste.illinois.edu/activity/buffon/. Accessed 23 January 2025.

Attached below is the link to access the PS#2 files in my ENGS 107 GitHub Repository:

https://github.com/isaiah-richardson28/Dartmouth-ENGS-107-Winter-2025

APPENDIX

R code script for Question 4A: ***(This code script when copied and pasted into RStudio runs properly and produces the PDF file)***

```
# ENGS 107 Problem Set 2: Monte Carlo/Convergence/Seeds/Scripts/Plotting
# Professor Dr. Klaus Keller
# Due Date: Friday, January 24, 2025 at 11:59 pm
# File name: [isaiah.d.richardson.th@dartmouth.edu].ps#2.R
# Software: R and RStudio
#_____________________________________________________________________
# Author: Isaiah D. Richardson, M.S. (IDR)
# Copyright: copyright by the author
# License: this code is distributed under the GNU GENERAL PUBLIC LICENSE v3.0
# License: for more information regarding GNU GENERAL PUBLIC LICENSE Version 3 visit: https://www.gnu.org/licenses/why-not-lgpl.html
# There is no warranty on this R code script for ENGS 107 Question 4A
#_____________________________________________________________________
# Version 1: last changes made on: January 23, 2025
#_____________________________________________________________________
# Sources:
# - coin-example.R for general formatting and layout of the header notes
# - R help files accessed through R-studio for syntax
# - Grolemund, G. Hands-On Programming with R. For understanding the RStudio interface and for understanding basic functions
# - Cotton, R. Learning R: A Step-by-Step Function Guide to Data Analysis. For understanding how seeds work, the rnorm, nrow, ncol, apply, and sapply functions, how to set up a matrix function, and plotting
# - Venables, W.N., Smith, D.M. & the R Core Team. (2024). An Introduction to R Notes on R: A Programming Environment for Data Analysis and Graphics Version 4.4.2 for functions and plotting
# - Verzani, J. (2002). simpleR - Using R for Introductory Statistics. Used for understanding notation, functions, and formatting
# - Culhane, A. (2012). Introduction to Programming in R. For understanding functions, including the matrix function, and for plotting
# - Grabner-Radkowitsch, C. (2022). Monte Carlo Simulations in R. For understanding the basics of how to perform a Monte Carlo simulation in R
# - Grabner-Radkowitsch, C. (2022). Fundamental object types in R III: Factors and Data Frames. For understanding how to set up a matrix function
```

- Bulut, O. (2020). Conducting Monte Carlo Simulations in R. For understanding how to perform a Monte Carlo Simulation in R
- Dong, H. & Nakayama, M.K. (2020). A Tutorial on Quantile Estimation via Monte Carlo. For understanding what a quantile is in R
- Applegate, P.J., Keller, K. et al.(2016). Risk Analysis in the Earth Sciences: A Lab Manual with Exercises in R. For downloading R and RStudio and understanding the RStudio interface and basic functions
- Robert, C.P. & Casella, G. (2004). Monte Carlo Statistical Methods (2nd edition). For deepening my conceptual understanding of convergence criteria
- Rizzo, M.L. (2019). Statistical Computing with R (2nd edition). To understand basic syntax, functions, background on performing a Monte Carlo simulation in R
- Wickman, H. (2015). Advanced R. To understand how to generate an initial random population and setting up data frames
- Hothorn, T. & Everitt, B.S. (2014). A Handbook of Statistical Analyses Using R (3rd edition). For understanding random number generation in R
- R Core Team. (2024). R: A Language and Environment for Statistical Computing. For understanding the rnorm(), quantile(), sample(), and sd() functions
- Murrell, P. (2005). R Graphics (2nd edition). For understanding the grid package graphics and text capabilities in R for the printing of results onto a PDF file
- Rosenberg, M. (2021) Analytics-Using-R. To understand how to set a seed and to understand how to create a for loop in R.
- Rubinstein, R.Y. & Kroese, D.P. (2016). Simulation and the Monte Carlo Method (2nd ed). For understanding proper convergence criteria when performing a Monte Carlo simulation
- “Estimating Pi using the Monte Carlo Method.” Academo. Retrieved from https://academo.org/demos/estimating-pi-monte-carlo/. Accessed 20 January 2025. For understanding the formulas and conceptual basis for how
- to calculate pi using a Monte Carlo method using a unit square and unit circle.
- Ross, S.M. (2006). Simulation (4th ed). Academic Press. Provided a conceptual basis for understanding convergence criteria for a Monte Carlo simulation
- Chang, W. (2013). R Graphics Cookbook: Practical Recipes for Visualizing Data. O’Reilly Media. For understanding the syntax, functions, formatting, and basics for producing plots and visualizing data in R.
- Gillespie, C. & Lovelace, R. (2016). Efficient R Programming: A Practical Guide to Smarter Programming. O’Reilly Media. For better understanding the RStudio interface, package selection, and optimizing code
- structure and performance.
- Matloff, N. (2011). The Art of R Programming. No Starch Press. For understanding how to set up and structure for loops and while loops in R
- Peng, R.D. (2015). R Programming for Data Science. Leanpub.For understanding how to set up, structure, and write for and while loops in R

```
# - Personal correspondence with Dr. Keller during office hours in-person to conceptually discuss the graphing of the results taking uncertainty and seeds into account (01/17/2025; 01/21/2025)
# - Personal correspondence with TA Siddhi Gothivrekar during office hours over Zoom and in-person to discuss code structure, submitting to Github, ascii file generation, and graphical figure results (01/18/2025;
# - 01/21/2025; 01/23/2025)
#_____________________________________________________________________________
# Collaborators:
# Emma Vejcik - conceptual discussions outside of lecture hours regarding assessing uncertainty (choice of seeds and small sample size)
#_____________________________________________________________________________
# To run this script file:
# 1. Open the ascii file in R
# 2. Use the cursor to highlight the entire script file
# 3. Press either 'Source' or 'Ctrl + Enter' to run the entire script
# 4. Open the resulting PDF file to analyze the results
#_____________________________________________________________________________
# Description of the Problem Statement:
# Use a Monte Carlo simulation method to:
# A) determine the mean and the 95 percentile from a known univariate normal
#    distribution with a mean of zero and a standard deviation of one with
#    your estimated uncertainties.
#_____________________________________________________________________________
# The code script for Problem Set 2 Question 4A is detailed below:
#_____________________________________________________________________________

# Importing a grid package library to enable enhanced functions for subsequent text based plotting in R
library(grid)

# Setting the parameters specified in the problem statement for the normal distribution
mean <- 0  # setting the mean of the normal distribution as 0
standard_deviation <- 1  # setting the standard deviation of the normal distribution as 1

# Setting five different seed values for reproducibility and controlled randomness (variability) in the Monte Carlo simulations
```

```
# Note: Different results are obtained for each seed, re-running the simulation with the same seed produces the same results
seed_values <- c(3, 5, 7, 10, 14)

# Specifying the convergence criteria, specifically the tolerance for the standard deviation of the mean and 95th percentile estimates
# Note: The Monte Carlo simulation iterations stop if the last 100 estimates fluctuate within the bounds of the specified threshold
convergence_tolerance <- 0.005

# Defining the maximum number of iterations for the Monte Carlo simulation (an upper limit or bound that cannot be exceeded)
# Note: This helps to reduce long computation times and prevent infinite loops during the simulation
maximum_iterations <- 10000

# Creating and initializing lists to store the key results for each seed value
results <- list()

# Initializing a data frame to store the key results for each seed value
summary_stats <- data.frame(
  Seed = integer(),
  Average_Estimated_Mean = numeric(),
  Final_Std_Dev_Mean = numeric(),
  Average_Estimated_95th = numeric(),
  Final_Std_Dev_95th = numeric()
)
#_____________________________________________________________________________

# Using a for loop to iterate over different seed values to generate Monte Carlo samples
for (seed in seed_values) {

  # Setting the seed for reproducibility
  set.seed(seed)

  # Generating an initial random population from the normal distribution
  random_population <- rnorm(1000, mean = mean, sd = standard_deviation)
```

```
  # Using the numeric function to initialize vectors to store the estimates over the Monte Carlo simulation iterations
  means <- numeric(maximum_iterations)
  percentiles_95 <- numeric(maximum_iterations)
  std_dev_means <- numeric(maximum_iterations)
  std_dev_percentiles_95 <- numeric(maximum_iterations)

  # Specifying control variables as convergence stopping criteria
  # A Boolean flag to monitor convergence
  converged <- FALSE

  # Starting an iteration counter
  iteration <- 1 #
#_____________________________________________________________________________
  # Running the Monte Carlo simulation for a specific seed until the maximum number of iterations (upper bound = 10,000 iterations)
or convergence criteria are met
  while (iteration <= maximum_iterations && !converged) {

    # Performing sampling with replacement from the generated random population
    # Note: replace = TRUE specifies to sample with replacement
    random_samples <- sample(random_population, iteration, replace = TRUE)

    # Calculating the estimated mean for the sampled data using the mean function
    means[iteration] <- mean(random_samples)
    # Calculating the estimated 95th percentile for the sampled data using the quantile function
    percentiles_95[iteration] <- quantile(random_samples, 0.95)

    # Calculating a rolling standard deviation for the means and 95th percentiles over the iterations for each seed
    if (iteration > 1) {

      # Calculating a rolling standard deviation for the estimated mean
      std_dev_means[iteration] <- sd(means[1:iteration])
      # Calculating a rolling standard deviation for the estimated 95th percentile
      std_dev_percentiles_95[iteration] <- sd(percentiles_95[1:iteration])
```

```
    }

    # Checking if the convergence criteria are met after 1,000 iterations of the Monte Carlo simulation
    # Note: If the last 100 estimates fluctuate within the bounds of the threshold, the simulation stops
    if (iteration > 1000) {
      if (std_dev_means[iteration] < convergence_tolerance && std_dev_percentiles_95[iteration] < convergence_tolerance) {
        converged <- TRUE
      }
    }

    # Counting the number of iterations
    iteration <- iteration + 1
  }
#_____________________________________________________________________________
  # Storing the key results for each seed in a structured list
  results[[as.character(seed)]] <- list(

    # A truncated vector of estimated mean values
    means = means[1:(iteration - 1)],

    # A truncated vector of estimated 95th percentile values
    percentiles_95 = percentiles_95[1:(iteration - 1)],

    # The rolling standard deviation for the estimated means
    std_dev_means = std_dev_means[1:(iteration - 1)],

    # The rolling standard deviation for the estimated 95th percentiles
    std_dev_percentiles_95 = std_dev_percentiles_95[1:(iteration - 1)]
  )
#_____________________________________________________________________________
#### Calculating the summary statistics for the simulation ####

  # Calculating the average estimated mean
```

```
  avg_estimated_mean <- mean(means[1:(iteration - 1)])
  # Calculating the final standard deviation of the estimated means
  final_std_dev_mean <- std_dev_means[iteration - 1]

  # Calculating the average estimated 95th percentile value
  avg_95th_percentile <- mean(percentiles_95[1:(iteration - 1)])
  # Calculating the final standard deviation of the estimated 95th percentiles
  final_std_dev_95th_percentile <- std_dev_percentiles_95[iteration - 1]
#_____________________________________________________________________________

  # Appending the key results to the data frame
  summary_stats <- rbind(summary_stats, data.frame(
    Seed = seed,

    # The average estimated mean
    Average_Estimated_Mean = avg_estimated_mean,

    # The final standard deviation of the estimated means
    Final_Std_Dev_Mean = final_std_dev_mean,

    # The average estimated 95th percentile
    Average_Estimated_95th_Percentile = avg_95th_percentile,

    # The final standard deviation of estimated 95th percentiles
    Final_Std_Dev_95th = final_std_dev_95th_percentile
  ))
}
#_____________________________________________________________________________
# Creating a PDF file to save the Monte Carlo simulation results for all of the seeds
pdf(file = "Monte_Carlo_Simulation_Results_4A.pdf", width = 8, height = 6)

# Defining the colors for each seed value (i.e., 3, 5, 7, 10, 14) for plotting
colors <- c("blue", "red", "gray", "orange", "purple")
```

```
legend_labels <- c()
#_____________________________________________________________________________
#### PLOT 1 ####

# Plotting a histogram of the generated random population
# Note: probability = TRUE converts the histogram plot into a probability density plot; breaks = 50 divides the histogram into 50 bins
hist(random_population, probability = TRUE, breaks = 50, col = "lightgray", border = "black",

    # Specifying the title for the histogram plot
    main = "Plot 1: Histogram Plot of the Generated Random Population",

    # Specifying the x-axis label for the histogram plot
    xlab = "Data Value",

    # Specifying the y-axis label for the histogram plot
    ylab = "Probability Density")

# Adding a legend to the upper left corner of the histogram plot
# Note: lwd specifies the line width; lty = c(1, 2) specifies the lines as 1 (solid) and 2 (dashed); inset offsets the legend from the plot border
legend("topleft", legend = c("Normal Curve", "95th Percentile"), col = c("red", "blue"), lwd = 2, lty = c(1, 2), inset = 0.02)

# Overlaying the histogram plot with the theoretical normal distribution
# Note: dnorm computes the probability density function of the normal distribution
# Note: add = TRUE overlays the normal density curve onto the histogram plot
curve(dnorm(x, mean = mean, sd = standard_deviation), col = "red", lwd = 2, add = TRUE)

# Marking the 95th percentile on the histogram plot
# Note: the quantile function calculates the 95th percentile of the random_population; the abline function adds a dashed vertical line
# Note: lwd specifies the line width; lty = 2 specifies the line as dashed
percentile_95_value <- quantile(random_population, 0.95)
abline(v = percentile_95_value, col = "blue", lwd = 2, lty = 2)
#_____________________________________________________________________________
```

```
#### PLOT 2 ####

# Plot of the Estimated Mean for Each Seed (y-axis) over the Number of Iterations (x-axis)
plot(NULL, xlim = c(1, maximum_iterations), ylim = c(-0.5, 0.5),

     # Specifying the label for the x-axis
     xlab = "Number of Iterations",

     # Specifying the label for the y-axis
     ylab = "Estimated Means",

     # Specifying the label for the plot title
     main = "Plot 2: Estimated Means over Number of Iterations")

# Using a for loop to loop through each seed value and plot the estimated means
for (i in 1:length(seed_values)) {

  # Converting the seed value to a character string to index the list of results
  seed <- as.character(seed_values[i])

  # Plotting the estimated mean values over the number of iterations for a specific seed
  lines(1:length(results[[seed]]$means), results[[seed]]$means, col = colors[i], lwd = 0.05)
}
# Creating a legend to label the results for the five different seeds examined
legend_labels <- paste("Seed", seed_values)

# Formatting the legend; lwd specifies line width, inset offsets the legend from the plot border
legend("topright", legend = legend_labels, col = colors, lwd = 1, inset = 0.02)
#_____________________________________________________________________________
#### PLOT 3 ####

# Plot of the Estimated 95th Percentiles for Each Seed (y-axis) over the Number of Iterations (x-axis)
plot(NULL, xlim = c(1, maximum_iterations), ylim = c(1, 2),
```

```
    # Specifying the x-axis label
    xlab = "Number of Iterations",

    # Specifying the y-axis label
    ylab = "Estimated 95th Percentiles",

    # Specifying the label for the plot title
    main = "Plot 3: Estimated 95th Percentiles over Number of Iterations")

# Using a for loop to loop through each seed value and plot the estimated 95th percentiles
for (i in 1:length(seed_values)) {

  # Converting the seed value to a character string to index the list of results
  seed <- as.character(seed_values[i])

  # Plotting the estimated 95th percentile values over the number of iterations for a specific seed
  lines(1:length(results[[seed]]$percentiles_95), results[[seed]]$percentiles_95, col = colors[i], lwd = 0.05)
}
# Creating a legend to label the results for the five different seeds examined
legend_labels <- paste("Seed", seed_values)

# Formatting the legend for plot 2; lwd specifies line width, inset offsets the legend from the plot border
legend("bottomright", legend = legend_labels, col = colors, lwd = 1, inset = 0.02)
#_____________________________________________________________________________
#### PLOT 4 ####

# Plot of the Standard Deviation of the Estimated Means for Each Seed (y-axis) over the Number of Iterations (x-axis)
# Note: Starting from iteration 2 (not plotting iteration 1 because it makes no sense to plot the standard deviation of a single value)
plot(NULL, xlim = c(2, maximum_iterations), ylim = c(0, 0.5),

    # Specifying the x-axis label
    xlab = "Number of Iterations",
```

```
    # Specifying the y-axis label
    ylab = "Standard Deviation of the Estimated Mean",

    # Specifying the label for the plot title
    main = "Plot 4: Standard Deviation of Estimated Means Over Number of Iterations")

# Using a for loop to loop through each seed value and plot the standard deviations of the estimated means
for (i in 1:length(seed_values)) {

  # Converting the seed value to a character string to index the list of results
  seed <- as.character(seed_values[i])

  # Extracting the standard deviations of the estimated means for a given seed
  std_dev_means <- results[[seed]]$std_dev_means

  # Ensuring that there is more than one value to plot
  if (length(std_dev_means) > 1) {

    # Plotting the standard deviation of the estimated means for a specific seed
    lines(2:length(std_dev_means), std_dev_means[-1], col = colors[i], lwd = 1)
  }
}
# Creating a legend to label the results for the five different seeds examined
legend_labels <- paste("Seed", seed_values)

# Formatting the legend; lwd specifies line width; inset offsets the legend from the plot border
legend("topright", legend = legend_labels, col = colors, lwd = 1, inset = 0.02)
#_____________________________________________________________________________
#### PLOT 5 ####

# Plot of the Standard Deviation of the Estimated 95th Percentiles (y-axis) over the Number of Iterations (x-axis)
```

```
# Note: Starting from iteration 2 (not plotting iteration 1 because it does not make any sense to plot the standard deviation of a single value)
plot(NULL, xlim = c(2, maximum_iterations), ylim = c(0, 0.5),

     # Specifying the x-axis label
     xlab = "Number of Iterations",

     # Specifying the y-axis label
     ylab = "Standard Deviation of Estimated 95th Percentiles",

     # Specifying the label for the plot title
     main = "Plot 5: Standard Deviation of Estimated 95th Percentiles Over Iterations")

# Using a for loop to loop through each seed value and plot the standard deviations of the estimated 95th percentiles
for (i in 1:length(seed_values)) {

  # Converting the seed value to a character string to index the list of results
  seed <- as.character(seed_values[i])

  # Extracting the standard deviations of the estimated 95th percentiles for a given seed
  std_dev_percentiles_95 <- results[[seed]]$std_dev_percentiles_95

  # Ensuring that there is more than one value to plot
  if (length(std_dev_percentiles_95) > 1) {

    # Plotting the standard deviation of the estimated 95th percentiles for a specific seed
    lines(2:length(std_dev_percentiles_95), std_dev_percentiles_95[-1], col = colors[i], lwd = 1)
  }
}
# Creating a legend to label the results for the five different seeds examined
legend_labels <- paste("Seed", seed_values)

# Formatting the legend; lwd specifies line width; inset offsets the legend from the plot border
```

```
legend("topright", legend = legend_labels, col = colors, lwd = 1, inset = 0.02)
#_____________________________________________________________________________
#### ADDING THE KEY RESULTS TO THE END OF THE PDF FILE ####

# Specifying the number of seeds to display per PDF page (so the data does not run off of the page)
number_per_page <- 2

# Calculating the total number of pages needed to print the results for the five different seeds
number_pages <- ceiling(nrow(summary_stats) / number_per_page)

# Using a for loop to loop through each PDF page to add the key results for each seed value
for (page in 1:number_pages) {

  # Creating a new blank plot for each PDF page
  plot.new()

  # Formatting and labeling the title on the top of each PDF page
  # Note: cex allows for adjustment of the character and letter sizes
  text(0.5, 0.9, paste("Monte Carlo Simulation Results"), cex = 1.5, font = 2)

  # Determining the row indices for the PDF page
  # Starting the row index for the PDF page
  start_row <- (page - 1) * number_per_page + 1

  # Ending the row index for each PDF page to ensure that the results to not run off of the PDF page
  end_row <- min(page * number_per_page, nrow(summary_stats))

  # Initializing the vertical position for placement of the results text
  y_pos <- 0.8

  # Using a for loop to loop through the results summary for each seed and to display the results on the PDF page
  # Note: All printed results are rounded to four (4) decimal places
  for (i in start_row:end_row) {
```

```
    text(0.1, y_pos, paste("Seed:", summary_stats$Seed[i]), pos = 4, font = 2)

    # Posting the Average Estimated Mean
    text(0.1, y_pos - 0.05, paste("Average Estimated Mean:", round(summary_stats$Average_Estimated_Mean[i], 4)), pos = 4)

    # Posting the Final Standard Deviation of the Estimated Mean
    text(0.1, y_pos - 0.10, paste("Final Standard Deviation of the Estimated Mean:", round(summary_stats$Final_Std_Dev_Mean[i],
4)), pos = 4)

    # Posting the Averaged Estimated 95th Percentile
    text(0.1, y_pos - 0.15, paste("Average Estimated 95th Percentile:", round(summary_stats$Average_Estimated_95th[i], 4)), pos = 4)

    # Posting the Final Standard Deviation of the Estimated 95th Percentile
    text(0.1, y_pos - 0.20, paste("Final Standard Deviation of the Estimated 95th Percentile:",
round(summary_stats$Final_Std_Dev_95th[i], 4)), pos = 4)

    # Adjusting the vertical position for the set of results
    y_pos <- y_pos - 0.5
  }
}
#_____________________________________________________________________________
# Closing the PDF file
dev.off()
```

R code script for Question 4B: ***(This code script when copied and pasted into RStudio runs properly and produces the PDF file)***

```
# ENGS 107 Problem Set 2: Monte Carlo/Convergence/Seeds/Scripts/Plotting
# Professor Dr. Klaus Keller
# Due Date: Friday, January 24, 2025 at 11:59 pm
# File name: [isaiah.d.richardson.th@dartmouth.edu].ps#2.R
# Software: R and RStudio
#_____________________________________________________________________________
```

Author: Isaiah D. Richardson, M.S. (IDR)
Copyright: copyright by the author
License: this code is distributed under the GNU GENERAL PUBLIC LICENSE v3.0
License: for more information regarding GNU GENERAL PUBLIC LICENSE Version 3 visit: https://www.gnu.org/licenses/why-not-lgpl.html
There is no warranty on this R code script for ENGS 107 Question 4B
#_____
Version 1: last changes made on: January 23, 2025
#_____
Sources (Relevant to both Question 4A and 4B):
- coin-example.R for general formatting and layout of the header notes
- R help files accessed through R-studio for syntax
- Grolemund, G. Hands-On Programming with R. For understanding the RStudio interface and for understanding basic functions
- Cotton, R. Learning R: A Step-by-Step Function Guide to Data Analysis. For understanding how seeds work, the rnorm, nrow, ncol, apply, and sapply functions, how to set up a matrix function, and plotting
- Venables, W.N., Smith, D.M. & the R Core Team. (2024). An Introduction to R Notes on R: A Programming Environment for Data Analysis and Graphics Version 4.4.2 for functions and plotting
- Verzani, J. (2002). simpleR - Using R for Introductory Statistics. Used for understanding notation, functions, and formatting
- Culhane, A. (2012). Introduction to Programming in R. For understanding functions, including the matrix function, and for plotting
- Grabner-Radkowitsch, C. (2022). Monte Carlo Simulations in R. For understanding the basics of how to perform a Monte Carlo simulation in R
- Grabner-Radkowitsch, C. (2022). Fundamental object types in R III: Factors and Data Frames. For understanding how to set up a matrix function
- Bulut, O. (2020). Conducting Monte Carlo Simulations in R. For understanding how to perform a Monte Carlo Simulation in R
- Dong, H. & Nakayama, M.K. (2020). A Tutorial on Quantile Estimation via Monte Carlo. For understanding what a quantile is in R
- Applegate, P.J., Keller, K. et al.(2016). Risk Analysis in the Earth Sciences: A Lab Manual with Exercises in R. For downloading R and RStudio and understanding the RStudio interface and basic functions
- Robert, C.P. & Casella, G. (2004). Monte Carlo Statistical Methods (2nd edition). For deepening my conceptual understanding of convergence criteria
- Rizzo, M.L. (2019). Statistical Computing with R (2nd edition). To understand basic syntax, functions, background on performing a Monte Carlo simulation in R
- Wickman, H. (2015). Advanced R. To understand how to generate an initial random population and setting up data frames

- Hothorn, T. & Everitt, B.S. (2014). A Handbook of Statistical Analyses Using R (3rd edition). For understanding random number generation in R
- R Core Team. (2024). R: A Language and Environment for Statistical Computing. For understanding the rnorm(), quantile(), sample(), and sd() functions
- Murrell, P. (2005). R Graphics (2nd edition). For understanding the grid package graphics and text capabilities in R for the printing of results onto a PDF file
- Rosenberg, M. (2021) Analytics-Using-R. To understand how to set a seed and to understand how to create a for loop in R.
- Rubinstein, R.Y. & Kroese, D.P. (2016). Simulation and the Monte Carlo Method (2nd ed). For understanding proper convergence criteria when performing a Monte Carlo simulation
- "Estimating Pi using the Monte Carlo Method." Academo. Retrieved from https://academo.org/demos/estimating-pi-monte-carlo/. Accessed 20 January 2025. For understanding the formulas and conceptual basis for how to calculate pi using a Monte Carlo method using a unit square and unit circle.
- Ross, S.M. (2006). Simulation (4th ed). Academic Press. Provided a conceptual basis for understanding convergence criteria for a Monte Carlo simulation
- Chang, W. (2013). R Graphics Cookbook: Practical Recipes for Visualizing Data. O'Reilly Media. For understanding the syntax, functions, formatting, and basics for producing plots and visualizing data in R.
- Gillespie, C. & Lovelace, R. (2016). Efficient R Programming: A Practical Guide to Smarter Programming. O'Reilly Media. For better understanding the RStudio interface, package selection, and optimizing code structure and performance.
- Matloff, N. (2011). The Art of R Programming. No Starch Press. For understanding how to set up and structure for loops and while loops in R
- Peng, R.D. (2015). R Programming for Data Science. Leanpub.For understanding how to set up, structure, and write for and while loops in R
- Personal correspondence with Dr. Keller during office hours in-person to conceptually discuss the graphing of the results taking uncertainty and seeds into account (01/17/2025; 01/21/2025)
- Personal correspondence with TA Siddhi Gothivrekar during office hours over Zoom and in-person to discuss code structure, submitting to Github, ascii file generation, and graphical figure results (01/18/2025; 01/21/2025; 01/23/2025)
#_____
Collaborators:
Emma Vejcik - conceptual discussions outside of lecture hours regarding assessing uncertainty (choice of seeds and small sample size)
#_____
To run this script file:
1. Open the ascii file in R

```
# 2. Use the cursor to highlight the entire script file
# 3. Press either 'Source' or 'Ctrl + Enter' to run the entire script
# 4. Open the resulting PDF file to analyze the results
#_____________________________________________________________________
# Description of the Problem Statement:
# Use a Monte Carlo simulation method to:
# B) determine the value of pi with your estimated uncertainties
#_____________________________________________________________________
# The code script for Problem Set 2 Question 4B is detailed below:
#_____________________________________________________________________

# Importing a grid package library to enable enhanced functions for subsequent text based plotting in R
library(grid)

# Setting five different seed values for reproducibility and controlled randomness (variability) in the Monte Carlo simulations
# Note: Different results are obtained for each seed, re-running the simulation with the same seed produces the same results
seed_values <- c(3, 5, 7, 10, 14)

# Specifying the convergence criteria, specifically the tolerance for the estimated pi values
# Note: The Monte Carlo simulation iterations stop if the last 100 estimates fluctuate within the bounds of this threshold
convergence_threshold <- 0.005

# Defining the maximum number of iterations for the Monte Carlo simulation (an upper limit or bound that cannot be exceeded)
# Note: This helps to reduce long computation times and prevent infinite loops during the simulation
max_simulations <- 10000
#_____________________________________________________________________

# Defining the colors for each seed value (3, 5, 7, 10, 14) for subsequent plotting
plot_colors <- c("blue", "red", "gray", "orange", "purple")

# Creating a PDF file to save the Monte Carlo simulation results for all of the seeds
pdf(file = "Monte_Carlo_Simulation_Results_4B.pdf", width = 8, height = 6)
#_____________________________________________________________________
```

```
# Initializing an empty list to store the results from each Monte Carlo simulation
results_list <- list()
#_____________________________________________________________________________

# Using a for loop to run independent Monte Carlo simulations for each seed value
for (seed_index in seq_along(seed_values)) {

  # Extracting the current seed value
  seed <- seed_values[seed_index]

  # Setting the seed to ensure reproducibility of the results
  set.seed(seed)

  # Initializing numeric vectors to store the results
  # Using a numeric vector to store the estimated pi values at each iteration
  pi_estimates <- numeric(max_simulations)
  # Using a numeric vector to store the standard deviations of estimated pi values over the iterations
  standard_deviation_pi <- numeric(max_simulations)

  # Specifying control variables as convergence stopping criteria
  # A Boolean flag to monitor convergence
  has_converged <- FALSE

  # Initializing an iteration counter to track the number of iterations
  iteration <- 1
#_____________________________________________________________________________

  # Running the Monte Carlo simulation for a specific seed until the maximum number of iterations (upper bound = 10,000 iterations)
or the convergence criterion are met
  while (!has_converged && iteration <= max_simulations) {

    # Producing random x and y coordinates that are within the unit square: [-1,1] x [-1,1]
```

```
    # Generating uniformly distributed x coordinate values
    x <- runif(iteration, -1, 1)
    # Generating uniformly distributed y coordinate values
    y <- runif(iteration, -1, 1)

    # Counting the number of x, y coordinate points that lie inside the unit circle
    inside_circle <- sum(x^2 + y^2 <= 1)

    # Estimating the value of pi using a Monte Carlo approximation formula
    # Note: pi is approximately equivalent to: (4 * the number of points inside the circle)/(total number of generated points)
    pi_estimates[iteration] <- (inside_circle / iteration) * 4

    # Calculating the standard deviation of the pi estimates (if more than 1 iteration is finished)
    if (iteration > 1) {
      standard_deviation_pi[iteration] <- sd(pi_estimates[1:iteration])
    }

    # Checking for convergence criteria after 1,000 iterations of the Monte Carlo simulation
    # Note: If the last 100 estimates fluctuate within the bounds of the threshold, the simulation stops
    if (iteration > 1000) {
      # Examining the most recent 100 computed pi estimates
      recent_estimates <- pi_estimates[(iteration - 99):iteration]
      # Calculating the mean of the last 100 pi estimates
      mean_recent <- mean(recent_estimates)
      # Note: if all recent pi estimates fall within the previously defined threshold range from the mean, it is assumed that the simulation
has converged and the loop is terminated
      if (all(abs(recent_estimates - mean_recent) < convergence_threshold)) {
        has_converged <- TRUE
      }
    }

    # Incrementing the iteration counter for the upcoming simulation step
    iteration <- iteration + 1
```

```
  }

  # Storing the final results of the simulation for each seed in a structured list
  results_list[[seed_index]] <- list(

    # The seed value examined in the Monte Carlo simulation run
    seed = seed,

    # The total number of iterations of the Monte Carlo simulation performed
    iterations = iteration - 1,

    # Storing all of the pi estimates up to the final iteration of the Monte Carlo simulation
    pi_estimates = pi_estimates[1:(iteration - 1)],

    # Storing all of the standard deviation of estimated pi values up to the final iteration of the Monte Carlo simulation
    standard_deviation_pi = standard_deviation_pi[1:(iteration - 1)],

    # Storing the last estimated pi value calculated
    final_estimate_pi = pi_estimates[iteration - 1]
  )
}
#_____________________________________________________________________________
#### PLOT 1 ####

# Defining the y-axis range dynamically based on the estimated pi values
# Note: This ensures that the y-axis range is large enough so that all values are visible on the plot
minimum_pi_value <- min(sapply(results_list, function(x) min(x$pi_estimates, na.rm = TRUE)))
maximum_pi_value <- max(sapply(results_list, function(x) max(x$pi_estimates, na.rm = TRUE)))

# Plotting the Estimated pi values for each seed (y-axis) over the number of iterations (x-axis)
plot(NULL, xlim = c(1, max(sapply(results_list, function(x) x$iterations))),

     # Specifying expanded y-axis limits to enable optimal observation of the results
```

```
    ylim = c(minimum_pi_value - 0.1, maximum_pi_value + 0.1), type = "n",

    # Specifying the x-axis label
    xlab = "Number of Iterations",

    # Specifying the y-axis label
    ylab = "Estimated Value of Pi",

    # Specifying the label for the plot title
    main = "Plot 1: Estimated Value of Pi Over Number of Iterations")

# Using a for loop to loop through the stored results for each seed and to plot the results
for (seed_index in seq_along(results_list)) {
  lines(1:results_list[[seed_index]]$iterations, results_list[[seed_index]]$pi_estimates,
        col = plot_colors[seed_index], lwd = 0.1)
}

# Creating and formatting a legend to label the results for the five different seed values examined
legend("topright", legend = c(paste("Seed:", seed_values), "True pi value"),
       col = c(plot_colors, "black"), lwd = c(rep(1.5, length(seed_values)), 1), lty = c(rep(1, length(seed_values)), 2), inset = 0.02)

# Adding a reference line to denote the true value of pi (3.1415...) on the plot for comparison/reference
# Note: h = pi specifies the value for pi; lty = 2 specifies a dashed line; lwd specifies line width
abline(h = pi, col = "black", lty = 2, lwd = 1)
#_____________________________________________________________________________
#### PLOT 2 ####

# Plotting the standard deviation of the pi estimates for each seed (y-axis) over the number of iterations (x-axis)
# Note: Starting from iteration 2 (not plotting iteration 1 because it makes no sense to plot the standard deviation of a single value)
plot(NULL, xlim = c(2, max(sapply(results_list, function(x) x$iterations))),
    ylim = c(0, max(sapply(results_list, function(x) max(x$standard_deviation_pi[-1], na.rm = TRUE)))), type = "n",

    # Specifying the x-axis label
```

```
    xlab = "Number of Iterations",

    # Specifying the y-axis label
    ylab = "Standard Deviation of Pi Estimates",

    # Specifying the label for the plot title
    main = "Plot 2: Standard Deviation of Pi Estimates Over Number of Iterations")

# Using a for loop to loop through the stored results and to plot the results for each seed (starting from the second iteration)
for (seed_index in seq_along(results_list)) {
  lines(2:results_list[[seed_index]]$iterations, results_list[[seed_index]]$standard_deviation_pi[-1],
        col = plot_colors[seed_index], lwd = 1)
}
# Creating and formatting a legend to label the results for the five different seed values examined
# Note: lwd specifies line with; the legend is also offset slightly so it does not overlap with the plot border
legend("topright", legend = paste("Seed:", seed_values), col = plot_colors, lwd = 1.5, inset = 0.02)
#_____________________________________________________________________________
#### ADDING THE RESULTS SUMMARY TO THE PDF FILE ####

# Creating a new blank plot for the final PDF page
plot.new()

summary_text <- paste0(
  "MONTE CARLO SIMULATION RESULTS\n",
  "_____________________________________________\n\n"
)
# Using a for loop to loop through the results for each seed
# Note: purpose is to append the results to the PDF pages after the two plots
# Note: seq_along generates a sequence from 1 to the length of the results list to iterate over the stored results
for (seed_index in seq_along(results_list)) {
  result <- results_list[[seed_index]]

  # Calculating the average estimated pi value
```

```
  # Note: rounding the average estimated pi value result for each seed to four decimal places
  average_estimated_pi_value <- round(mean(result$pi_estimates), 4)

  # Calculating the final standard deviation of the estimated pi values
  if (length(result$pi_estimates) > 1) {
    # Calculating the standard deviation of the estimated values (round the result to 4 decimal places)
    final_std_dev <- round(sd(result$pi_estimates), 4)
  } else {
    # Note: if only one estimate exists, calculating the standard deviation is not meaningful
    final_std_dev <- "N/A"
  }
  # Appending the results to the summary text to paste onto the final PDF page
  summary_text <- paste0(
    summary_text,
    "Seed: ", result$seed, "\n",
    "Average Estimated Value of Pi: ", average_estimated_pi_value, "\n",
    "Final Standard Deviation of Estimated Pi value: ", final_std_dev, "\n\n"
  )
}
# displaying and formatting the key results for each seed on the final PDF page
# Note: just = "left" specifies left justification alignment for the text; x and y values indicate positioning on the page
grid.text(summary_text, x = 0.1, y = 0.5, just = "left", gp = gpar(fontsize = 10, fontface = "bold"))
#_____________________________________________________________________________
# Closing the PDF file
dev.off()
```