Assignment 1: The Monte Carlo Method

Arnav Nepal

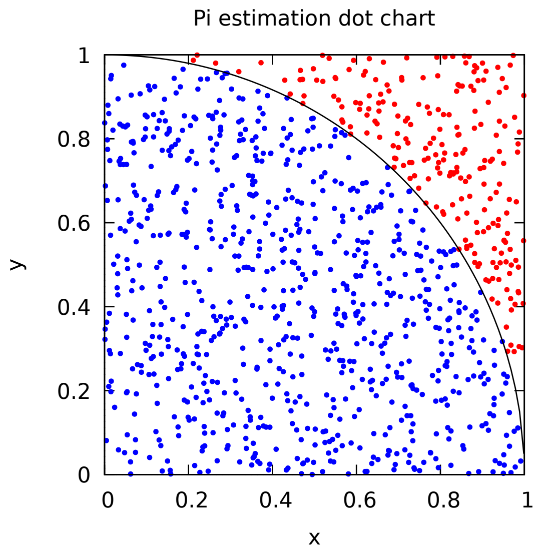
January 23, 2023

1 Introduction

In this assignment, we were tasked with creating graphs based on the Monte Carlo method for estimating π. The Monte Carlo method randomly puts dots in a square with a circle inscribed. After n iterations of Monte Carlo, the total number of dots that fall inside the circle and divided by the total number of dots outside the circle. This results in an estimation for π, with more iterations of Monte Carlo resulting in more accurate estimations. Using this method, we were tasked with creating 2 specific types of graphs - one visualizing the Monte Carlo method and one depicting the error between the estimated π value and actual π value over many iterations.

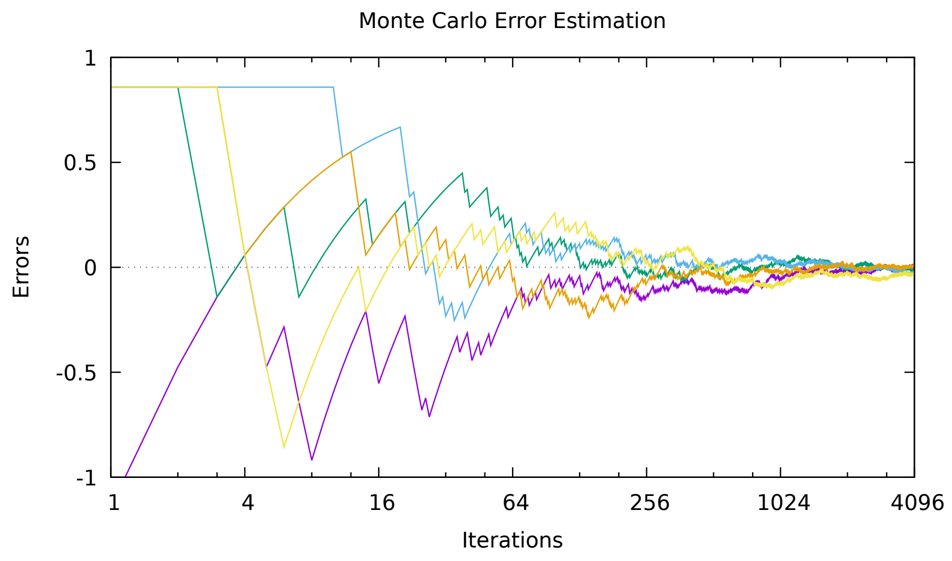
2 Plots

2.1 Monte Carlo Visualization



This graph depicts a visualization of the Monte Carlo method. The function $f(x) = \sqrt{1-x^2}$ represents circle (or a part of it) with blue points falling within the circle and red points falling outside. To make this plot, I first used awk to print the 3rd and 4th columns of the monte_carlo.c output, which contain the x and y coordinates. I used awk because it makes it very simple to get space-separated fields from text, which means it was simple to get the columns I needed. Then I used tail to print that same file starting from line 2 so that the header “x” and “y” would get removed. After I had the file of only x and y coordinates, I used awk again to separate points into 2 separate files based on whether they fell inside or outside the circle. I did this by using the parametric equation $x^2 + y^2 = 1$, which represents the line of a circle of radius 1. All points that fall within the circle will be $x^2 + y^2 <= 1$ while points that fall outside will be $x^2 + y^2 > 1$. Finally, I plotted the graph using gnuplot, by plotting the coordinates using filled points and the curve $y = \sqrt{1-x^2}$ using lines.

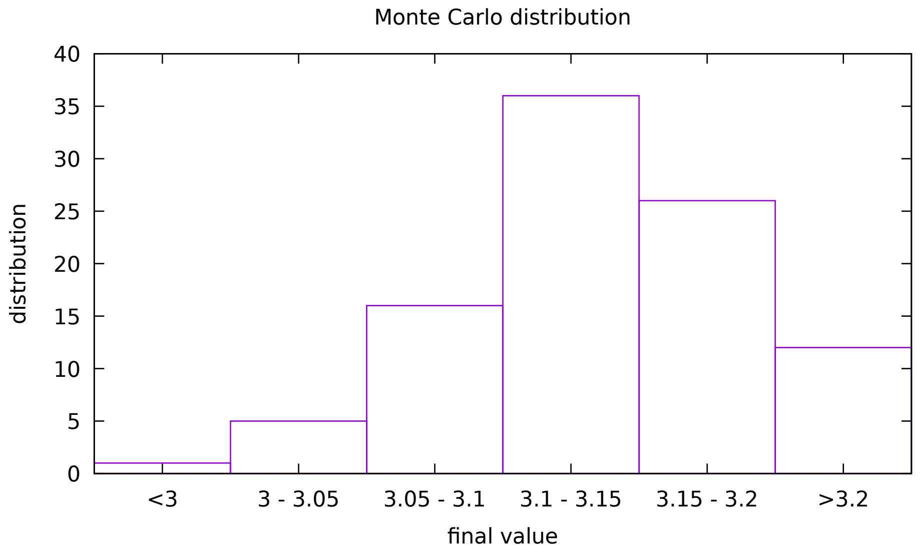
2.2 Monte Carlo Error Estimation



This graph visualizes the difference between the actual value of π and the estimated value of π over a large number of iterations. The different colored lines represent different seeds for the random number generator that monte_carlo.c uses. This graph shows that, regardless of seed, over a large number of iterations the difference between the actual and estimated value of π gets progressively gets lower.

The process for getting the data for this graph was similar to the Monte Carlo visualization graph, but used loops. I first set up a “for loop” to loop over the bash commands 5 times so that at least 5 different seeds would be represented. After running monte_carlo and obtaining the output, I separated the second column containing the π estimation with awk, then removed the header using tail so I would end up with a raw data file containing only the π estimation. I also used awk to subtract the actual value of π from the estimated value so I would get the difference (or “error”). The final output of the loop was put in a file with a numeric variable. This allowed each seed to be placed in its own file which made it easier to graph using gnuplot. I plotted this graph in gnuplot using lines with a x-axis log scale of 4 so that it would be easy to see change in values over a large number of iterations.

2.3 Monte Carlo Final Result Distribution



This box graph/histogram shows the distribution of the estimated values of π after 100 loops and 1000 monte_carlo iterations. Each loop, monte_carlo was randomly seeded using the bash random number generator $RANDOM. After 1000 iterations, the 1000th estimation is taken as the “final estimation” of the seed and the loop restarts. This graph shows the relative accuracy of the Monte Carlo method for estimating π. While getting multiple correct digits of π requires millions of iterations of the monte_carlo program (and a lot of time!), this graph shows that the algorithm generally converges on π accurately with set iterations as low as 1000. The bars representing 3.1 through 3.2 (relatively closest values to π) have the highest number of final results, while the bars representing other values are in the minority of final results.

This graph followed the same looping process as the error estimation graph, with a few key differences. For this graph, the loop was run 100 times instead of 5 and seeds were randomly chosen using $RANDOM. Furthermore, since I only needed the final value, tail only output the last value after a run of monte_carlo instead of everything except the header. The output of tail was appended to a data file containing final estimates so that the data file would gather data through all the loops. After the loop finished, I sorted through the data using a general process for each bar on the histogram: first I used awk with inequalities to separate data based on each range of value on the graph. Then I piped the output of that inequality to wc to count the number of lines so that I would know how many final values fell in that range. I then piped that value into another awk that printed the line count on the second column with the corresponding box position on the first column. The result of this final awk was then appended to a data file so that all the data would be contained within a single file. Finally, I plotted that data file in gnuplot using boxes so the data would be represented as a histogram.

3 Conclusion

This assignment introduced us to working with the UNIX command-line through Bash and some C code. I learned a lot about how to properly use Bash to create, move, and manipulate files and the data within the files. I felt the most important thing I learned about bash scripting is getting some experience with awk. Awk seems to be a very powerful tool, and I think it will be important get used to working with it in the future. While I used awk in this assignment, my usage was somewhat limited and used a fraction of the full range of features awk offers. I also learned how to utilize gnuplot to create interesting and well labeled graphs in this assignment. The 3rd graph ("Monte Carlo Final Result Distribution") was especially helpful towards this since it got me working with fine tuning gnuplot to properly represent data. Finally, this assignment was also useful in refreshing my memory on C through the provided monte_carlo.c file. I was able to understand and interpret the code, which helped when I had to use options and arguments. Ultimately, this was an interesting assignment because it got me working with (and helped me understand) several useful tools that will no doubt become very useful in the future as we start programming in C.