Landon Crowther
U0926601
CS 2420 – Assignment 8

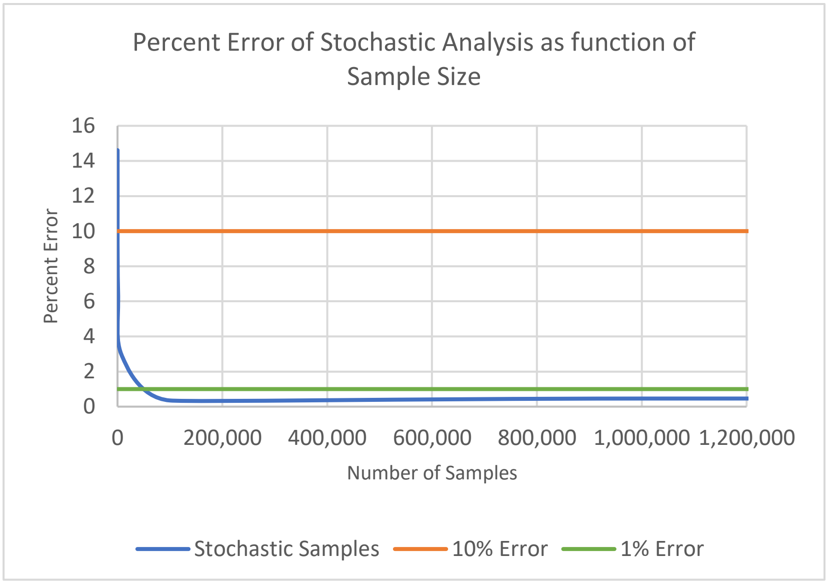
Assignment 8 Analysis:

Project Overview:

Stochastic sampling is a very effective way to determine the odds of something, determine random probability, or make numeric conclusions that may be impossible to determine mathematically. It works by repeating the same random process many times to work around the statistics that would normally be required. It may not be as accurate as checking every single possibility, but when testing many samples, the error is almost negligible. This only works if the random aspect of the sampling is random. If it is not, results will be skewed. An adequate random is one that is not predictable (or cyclic), and can represent all possibilities.

Timing and Accuracy

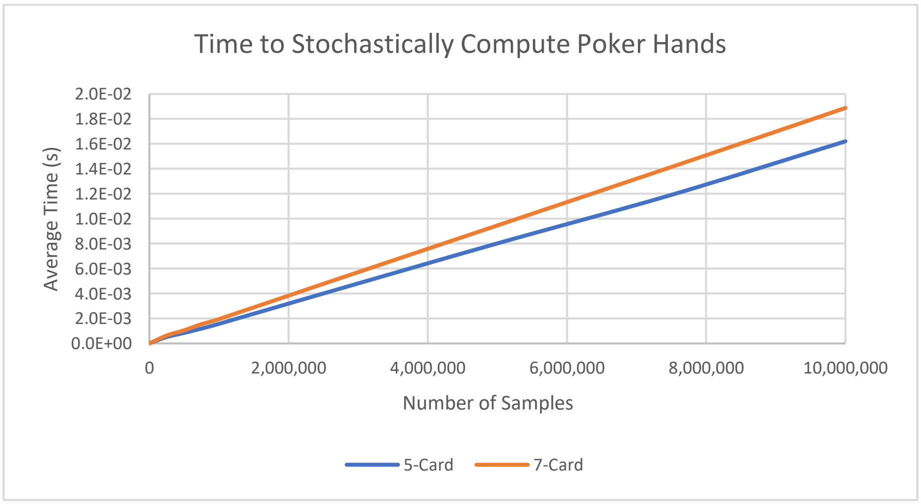
To assess the accuracy of stochastic sampling, calculated the percent error that stochastic sampling gave us. This was noted by the difference between the actual probability for each hand and the "estimated" (stochastic) probability. To avoid making different graphs for each possible hand rank, we simply added the percent difference for each rank. The following figure depicts the results:



As seen in the figure, the jump between 1000 samples and 10,000 samples pretty much minimized the error. After 200,000 samples, there was little to no error.

Stochastic sampling is quite effective; however, in situations where precision is necessary, one might thing the additional computation time for the exhaustive analysis might be helpful.

To address this, we ran another test, where we computed 1000 – 10,000,000 samples stochastically and recorded the time required for these results. We also computed the time required to compute the possibilities of each poker rank exhaustively for comparison. The figure to the right shows that even when computing 10,000,000 samples, the computer could do its analysis in less than 0.02 seconds. In comparison,



to exhaustively compute a 5-card hand, the computer needed 0.36 seconds, and for a 7-card hand, the computer needed 20.46 seconds. For any 5-card hand, exhaustive computation may be reasonable, because the time is relatively close to the max time for stochastic computation, and is slightly more accurate. However, for a 7-card hand, a stochastic comparison would be much more efficient, because the accuracy is just barely below that of exhaustive, yet the computation time is significantly less. Judging by the time for a 5-card and 7-card exhaustive computation, I would estimate that a 9-card exhaustive computation would take over 60 seconds -- very ineffective in comparison to stochastic sampling.

Software Development Log

This project took me more time than I would have liked; however, I was able to get a good head start on the project, and didn't have much last-minute cramming. I don't think that any of the code as particularly difficult, there were just parts that took longer than others. The timing and analysis class was probably the most difficult, as there was a lot going on that we had to keep track of.

Writing tests proved to be extremely useful for debugging purposes. The only tests we wrote were tests that ensured that the evaluate method was assigning proper ranks to card sets. This method was extremely important to test, as it was the basis for the rest of the assignment. The tests showed us that we in fact did have bugs, but we were able to resolve them easily.

Texas Hold 'Em:

It was really cool to see the results of our empirical analysis. Determining the best two hands was extremely difficult. To start, we generated an ArrayList containing Integer[] arrays of all start-card indices. Then, we made a class that was essentially a simpler version of Java's HashMap. We rated each starting hand by calling the odds_to_win method with the starting hand and every other possible starting hand.

We totaled the odds for each comparison, and then divided the odds by the number of comparisons made. This gave us the "average" score for each starting pair of cards when played against all other possible starting cards. We added the index value for the start pair and the associated score to our HashMap object. These HashMap objects were then added to a PriorityQueue, which sorted them by the top score and only kept track of the top 10. After adding all possible start hands to the PriorityQueue, we could retrieve the top 10 scoring cards.

Because aces are highest scoring card, it would make sense that they scored the highest, followed by kings. The following list summarizes the results from the Texas hold 'em analysis method: ace hearts & ace spades, ace diamonds & ace spades, ace clubs & ace spades, ace clubs & ace hearts, ace clubs & ace diamonds, ace diamonds & ace hearts; followed by four different pairs of kings. I would imagine that if we were keeping track of the top 12, all possibilities of a pair of kings would follow the pars of aces.

For each comparison, we used 50 samples. We decided this was adequate, because when going through the scores in debug mode, it was noted that the scores only differed in the ten-thousandths place, and more samples wouldn't have significantly changed anything.

A PDF document titled "odds.pdf" contains a table listing the win to lose ratio of all combinations of spades and hearts. Note, these results may not be absolute, as we only used a sample size of 50.

Landon Crowther
U0926601
CS 2420 – Assignment 8

Though Problems

A random number generator is crucial to successful stochastic analysis. A number generator that doesn’t work very well will not provide a holistic view of the problem. Our number generator worked decently well, however when running it with the files from Monday’s lab, we noticed that the distribution was skewed, so we decided to go with Java’s implementation for the remainder of the assignment, as we felt it was better at being consistent.

For the most part, the odd-odd, odd-even, even-odd, and even-even checks, as well as the distribution charts, are adequate methods of testing the effectiveness of a random number generator.