Random Sampling Algorithm Report

This report examines the provided C code, which implements a random sampling algorithm, and assesses the randomness of the sampling process. The code utilizes probability-based sampling and performs statistical analysis to evaluate the distribution of the sampled elements. The analysis includes comparing the occurrence of each sampled element with an expected distribution and calculating the percentage error.

Algorithm Overview:

'obtainRandomSample': This function simulates random sampling using the inverse of the index and a probability-based approach by comparing it to a pseudo-random decimal number generated by the function ***'drand48()'***. It iterates through a sequence of numbers of no given length and by changing the odds of selection as the sequence grows, each number has the same odds of being returned by the function at the mercy of the random number generated.

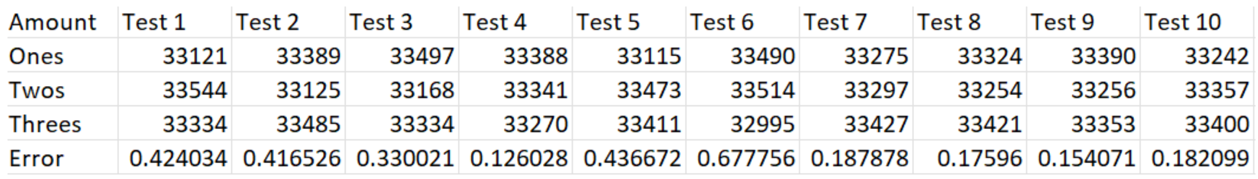
'testRandomSample': This function takes in a txt file created for the purpose of testing the sampling function. For the purpose of the experiment, the file contains exactly 300 1's, 2's, and 3's. This file is then read in and inserted into a malloced array of size 901 (giving room for INT_MAX terminating character). Then, the function runs
'obtainRandomSample' 99,999 times, tallying the occurrence of each possible number outcome (those outcomes being 1, 2, or 3). Those totals are then plugged into the percent error formula listed below.

$$\%\, error \;=\; \left|\frac{\#experimental - \#actual}{\#actual}\right| x100$$

The mean of those % errors is then calculated and compared to the standard error I decided on, which is 1. If the average percent error is below 1%, which it always is, then the function will return 0, representing successful testing of the sampling algorithm. If the file fails to open, malloc fails to allocate memory, or the algorithm doesn't pass the benchmark, then the function will return 1.

Assessment of Randomness:

1. **Probability-Based Sampling**: The use of probability-based mechanism and the inverse of the index introduces a level of randomness to the sampling process.
   $'drand48() \;<=\; \frac{1}{k}'$
2. **Comparison to Expected Distribution**: The code calculates the percentage error between the expected distribution (equal occurrence of each element) and the actual distribution resulting from the sampling process. The error quantifies how closely the sampled distribution matches the theoretical distribution.
3. **Statistical Analysis:**



Amount	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Ones	33121	33389	33497	33388	33115	33490	33275	33324	33390	33242
Twos	33544	33125	33168	33341	33473	33514	33297	33254	33256	33357
Threes	33334	33485	33334	33270	33411	32995	33427	33421	33353	33400
Error	0.424034	0.416526	0.330021	0.126028	0.436672	0.677756	0.187878	0.17596	0.154071	0.182099

As more proof of proof of randomness, here is the data curated from 10 runs of the testing program.

Conclusion:

In conclusion, the algorithm provided for random sampling is sufficient given that keeps a percent error of below 1% consistently. This is proven by 10 tests of 99999 random samples.

GPT-3.5 Credits:

This report was created using inspiration from a report generated by ChatGPT, a language model based on GPT-3.5 architecture, developed by OpenAI. GPT-3.5 is a powerful natural language processing model capable of generating human-like text and providing information, explanations, and reports on various topics.