MINI PROJECT 1

APMA 3100

By Winston Zhang, Luke Mathe

25. March 2022

TABLE OF CONTENTS AND FIGURES

1. OBJECTIVE

The objective of this project is to develop a simple Monte-Carlo simulation, which is a technique of artificially creating very large samples of random variables that can be used to mimic the behavior of random processes. This project is intended for us students to become familiar with designing such a model of experimentation on a computer, so that we as engineers can become skilled enough to apply such a system to more sophisticated problems.

2. SIMULATION SYSTEM

As described by the assignment description, a Monte-Carlo simulation system is comprised of three components:

1. **Random Number Generator**

   An algorithm that outputs a series of real numbers between 0 and 1, also known as pseudo-random numbers. As suggested by the name, a cursory glance at these pseudo-random numbers would appear to be indistinguishable from a sample of independent realizations of a uniform random variable.

2. **Random Variable Generator**

   An algorithm that maps realizations of the uniform random variable into realizations of the random variable with a specific cumulative distribution function.

3. **Mathematical model**

   This model should describe the problem set out to be simulated. This model specifies the CDF of the random variable described previously in list item **(2)**.

3. RANDOM NUMBER GENERATOR

As instructed by the assignment description, the random number generator's implementation will follow that of a commonly used algorithm, known as the linear congruential random number generator. This algorithm is a cyclical recursive function, that generates its outputs through real number division as described below:

$$x_i = (a\, x_{i-1} + c)(\textit{modulo}\ K),$$

$$u_i = x_i / K,$$

where parameters a, c, and K are constants, and the *modulo* refers to the remainder of a number after performing integer division. For this specific implementation, parameters are described in Table 1:

Starting value (seed)	$x_0 = 1000$
Multiplier a	$a = 24\ 693$
Increment c	$c = 3517$
Modulus K	$K = 2^{17}$

Table 1: Parameters for Random Number Generator Algorithm

With these parameters, this specific implementation of our random number generator will yield a cycle of pseudo-random numbers that is 2^{17} values long. This algorithm was implemented in Java as its own class, and took a total of 44 lines to write, including test cases in the main method. The algorithm was run with a few test inputs to ensure correctness, and the corresponding pseudo-random numbers that were expected as outputs, are described in Table 2, with inputs given by the assignment instructions highlighted in boldface:

$\boldsymbol{u_1}$	**0.4195**
$\boldsymbol{u_2}$	**0.0425**
$\boldsymbol{u_3}$	**0.1274**
u_{51}	0.5157
u_{52}	0.4273
u_{53}	0.7682

Table 2: Expected Pseudo-Random Number Outputs for Random Number Generator Algorithm

4. RANDOM VARIABLE GENERATOR

Discrete Random Variable

As covered in APMA 3100 material up until this point, the cumulative distribution function F of random variable X can be defined as the probability of non-exceedance:

$$F(x) = P(X \leq x) = \sum_{y \leq x} p(y),$$

with the previously defined and implemented random number generator, a given random number u_i generates realization x_i of X via the rule:

$$x_i = \min\{x : F(x) >= u_i\},$$

and the above rule may be repeatedly executed over $x = 1, \ldots, k$ until $F(x)$ is found to be greater than u_i.

Continuous Random Variable

Let X be a continuous random variable having a continuous distribution function F whose inverse exists in a closed form such that a given random variable u_i, as input to the inverse CDF, generates realization x_i of X:

$$x_i = F^{-1}(u_i)$$

5. SIMULATION PROBLEM

Model

1. W is a continuous random variable based on X being a continuous random variable when the call is available to be answered. However, a discrete number of seconds are taken when the call cannot be answered for one reason or another, in the instance when the call cannot be answered over the course of 4 attempts, W would be a discrete random variable. The model of the call being answered is a continuous random variable X with an exponential distribution, with $\mu_X = 12$ and thus $\lambda_X = 1/12$. W is a discrete random variable with μ_W being the mean time spent on the phone by the representative.

2. As described in the project assignment, continuous random variable X has an expected value of 12 seconds, with an exponential distribution. Thus X ~ Exponential (1/12) and the cumulative distribution function of X can be described as:

$$F_X(x) = 1 - e^{-(1/12)x}, \text{ for all } x > 0$$

The inverse of the cumulative distribution function of X can then be derived through algebra:

$$1 - e^{-(1/12)x} = u$$

$$-(1/12)x = ln(1 - u)$$

$$x = F(u) = -12ln(1 - u)$$

3. The model can be represented as a flowchart, as shown in Figure 1 (found on next page).

a. It should be noted that the value for X was generated by plugging a randomly generated decimal between 0 and 1 into the inverse CDF function, but indicating this was difficult to do so on the flowchart.

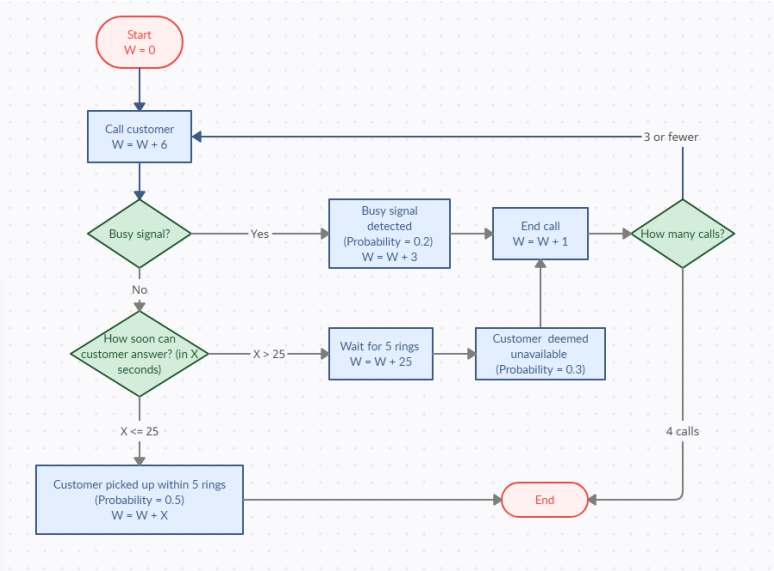

Figure 1: Simulation Problem Modeled as Flowchart

Data Collection

Data was collected through an ad-hoc experiment with actual phones. The time it took to mirror the actions taken by the telemarketer in the assignment description was recorded and repeated for three trials, and the average values were recorded. These values will replace those originally described in the assignment description, as shown in Table 3:

Activity	Described value (seconds)	Trial 1 (seconds)	Trial 2 (seconds)	Trial 3 (seconds)	Average Value (seconds)
Turn on Phone, Dial Number	6	7.16	6.94	7.14	7.08
Detect Busy Signal	5	5.21	5.89	5.10	5.40
Wait for 5 rings	25	32.70	33.09	32.91	32.90
End Call	1	1.27	1.23	1.10	1.20

Table 3: Experimental Time Values

In order to more realistically simulate the described process, numbers being dialed were random and read from a “phone book”, as dialing memorized numbers may skew the data towards smaller time values. Some sources of error may include using a mobile phone instead of a handset that one usually imagines that a telemarketer would use - this may have skewed the results as it takes slightly longer to put a cell phone down and wait for the screen to wake up before the hang up button can be pressed.

Monte-Carlo simulation algorithm

An algorithm was written in Java to simulate this model. The program was implemented in 249 lines and consisted of three components:

1. The Monte-Carlo simulation itself. This method takes in a seed integer so that it can generate a pseudo-random decimal number u, which is used when determining the routes to be taken in the flowchart described on page 6. Routes were structured as nested if-else statements, with the appropriate incrementing of W based on the route taken. If u did not satisfy the situations in which the customer’s line was busy, or the customer was unavailable, pseudo-random value u was plugged into a helper function that acted as the inverse CDF for X, generating the realization of X, which was then added to W, terminating the simulation.
2. A main method - this method made calls to the actual simulation and was capable of running a loop to repeat the trial for a specified amount of times. Every time the simulation was run, the main method would store the output for variable W and in an array/vector, which was used to compute the mean, median, first and third quartile, and several other statistics.
3. Helper methods - this includes several methods, such as the linear congruential random number generator described in Section 3, the inverse cumulative distribution function for X described in Section 4, as well as helped methods needed to calculate the statistics listed in the next section.

Simulation

As instructed in the project assignment description, the simulation was repeated for 1000 trials by setting the loop in the main method to loop 1000 times.

Statistical Estimation

Values of *W* were saved in a vector, as previously described, and said vector was passed into several helper functions also previously described in order to determine certain statistics, listed in Table 4:

Statistic	Value (in seconds)
Mean	30.114
First Quartile	9.735
Median	14.144
Third Quartile	43.387

Table 4: Mean, Median, and Quartiles of Generated Sample Data

Additionally, helper methods were used to find the probabilities of certain events of non/exceedance. This was done by iterating through the vector of saved values, counting every value that is a member of the event, and then performing simple division to find the probability. These values are reflected in Table 5:

Event	Probability
P[W <= 15]	0.511
P[W <= 20]	0.571
P[W <= 30]	0.616
P[W > 40]	0.318
P[W > 70]	0.128
P[W > 100]	0.038
P[W > 120]	0.008

Table 5: Probabilities Corresponding to Certain Events of (Non)exceedance

These values all make sense when comparing the respective events and their probabilities against the quartile and median values enumerated in Table 4.

Statistical Analysis

1. As shown in Section 5, the median value (14) is smaller than the mean (29.662). This means that values of W are more concentrated toward the lower end of the sample space of W, with a few outliers skewing the mean to the right. This is to be expected, as W is partially dependent on X, which is an exponential distribution, which has a higher concentration of realizations on the lower end of its sample space.
2. The sample space of W is an interval of integers ranging from 6 seconds to 128 seconds. This was determined by identifying the shortest and longest paths on the flowchart respectively, with the shortest path being the representative dialing a number for 6 seconds and the customer picking up immediately, and the longest path being the representative repeating the 32 second process of dialing and waiting for five rings four times, for a total of 128 seconds. The data confirms these bounds for the sample space of W, as no values generated by the simulation algorithm were found to be less than 6 or greater than 128.
3. An approximate CDF can be graphed using the probabilities listed in Table 5, as well as some interpolated values where needed. This confirms that our data at least maintains the expected distribution that closely resembles that of X, which is exponential. The graph of $F_W(w)$ can be found in Figure 2, created by inputting the data in Table 5 into a Google spreadsheet and generating a chart (found on next page):

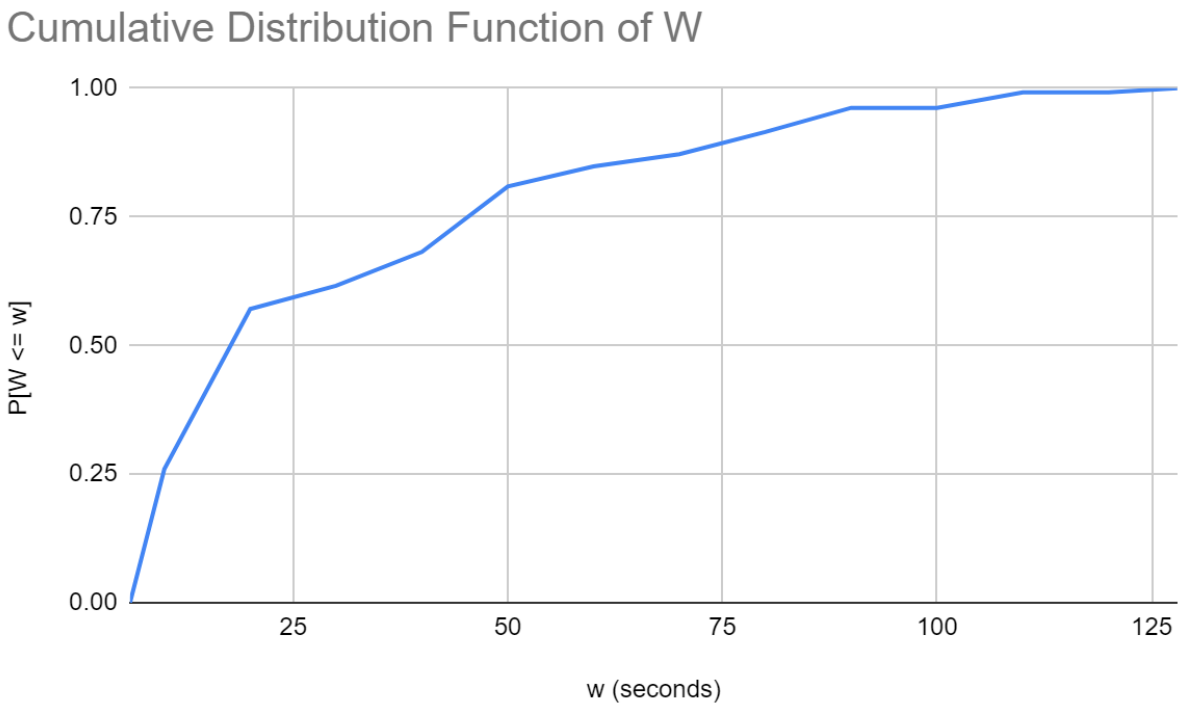

Figure 2: Cumulative Distribution Function of Continuous Random Variable *W*

Comments

1. The most challenging step was deciphering the paragraph description of the simulation problem. It was slightly difficult trying to figure out that the simulation problem was a combination of both discrete and continuous random variables, as well as how the inverse CDF of *X* came into play. Once a good model was formulated, implementing the algorithm in the form of code was the easiest part of the project.
2. Similar to above, the most time consuming part of the project was figuring out what exactly was supposed to be simulated, and what to put on our report, which took several days of working on. Inversely, the coding implementation took a single evening.
3. We are moderately confident in the accuracy of our findings, as the distribution of data closely resembles the distributions of the component random variables. Data points generated by the algorithm all fell within the sample space and no extraneous or outrageous data points were

generated. An example of such an outrageous data point would be a call time of 5 minutes (300 seconds), as the telemarketer described in the problem statement should've given up calling the customer by the time a total of 128 seconds were spent calling and waiting for a response. The pseudo-random values were confirmed to be in line with the values given as examples by the project assignment, and the seeding system used in the algorithm was also tested to make sure that every input integer was unique so that every corresponding pseudo-random value was also unique. Moreover, an extensive use of commenting was used in the coding process, with a system of Doxygen documentation in place to make sure that all methods took in the correct parameters, all methods were properly described, and all variables were kept track of.

HONOR PLEDGE

On our honor as students, we have neither given nor received unauthorized aid on this assignment.

Winston Zhang (wyz5rge) and Luke Mathe (lkm6eka)