Project 2 — Exploring the Fundamentals of Monte-Carlo Simulation

On My Honor as a student, I have neither given nor accepted unauthorized aid on this assignment

Caden Kowalski
Ted Nyberg

Introduction

"A representative of a high-speed Internet provider calls customers to assess their satisfaction with the service." This representative will call customers up to 4 times before giving up. We define the continuous random variable W as the total number of seconds the representative spends waiting for a customer to answer the phone over the course of the 4 potential calls. W has a minimum of 6 seconds (assuming the customer picks up immediately on the first call) and a maximum of 128 (assuming the customer does not pick up after 4 calls). We are interested in several statistics of W (ie. we want to know how W is distributed, purportedly to learn about the answering behavior of customers). In order to visualize the distribution of W, we use monte-carlo simulation to mimic trials of this experiment.

Part 1 — Formulating a Model: Notation, Equations, and Diagrams

Events

- **D:** probability the customer is dialed | P[D] = 1
- **A:** probability the customer is available | P[A] = 0.5
- **B:** probability the line is busy | P[B] = 0.2
- **O:** probability the line is open | P[O] = 0.3
- **M:** probability the customer is available but misses the call | P[M] = P[A] * P[X > 25]
- **R:** probability the customer is available and receives the call | P[R] = P[A] * P[X ≤ 25]

Random Variables

- **N:** number of calls (out of 4) until the customer picks up
    - N ~Geometric (p = 0.5)
    - $$p_N(n) = \begin{cases} 0.5\,(0.5)^{n-1} & n = 1,2,3,4, \\ 0 & \text{otherwise} \end{cases}$$
    - $$F_N(n) = \begin{cases} 1-(0.5)^n & n = 1,2,3,4, \\ 0 & \text{otherwise} \end{cases}$$
- **X:** number of seconds until an available customer picks up
    - E[X] = 12
    - X ~Exponential ($\lambda = 1/12$)

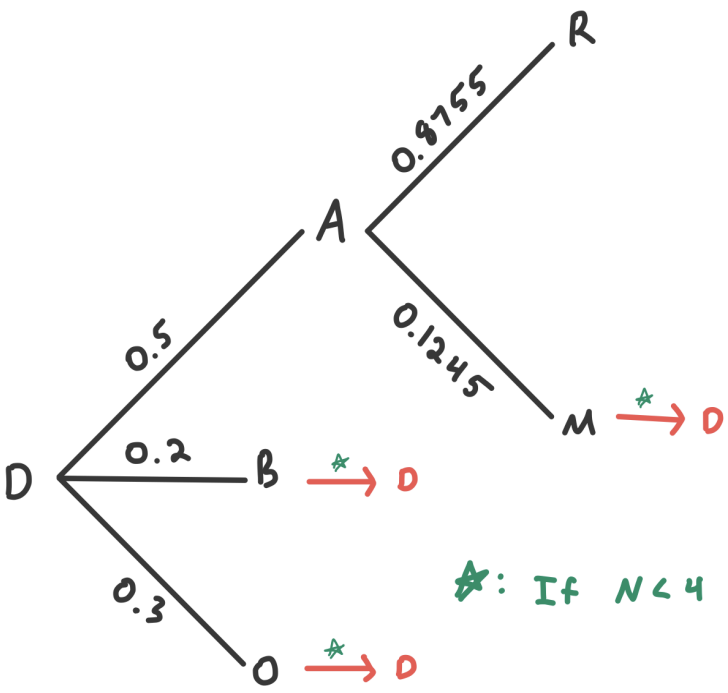
$$f_X(x) = \begin{cases} \frac{1}{12} e^{-x/12} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$F_X(x) = \begin{cases} 1 - e^{-x/12} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- $x = -12 \ln[1 - F_X(x)]$
- $P[X \leq 25] = 0.8755 \rightarrow P[R] = 0.4378$
- $P[X > 25] = 0.1245 \rightarrow P[M] = 0.0623$

- **W:** random variable W is defined as the number of seconds it takes, over the course of 4 potential calls, for a customer to pick up
  - W is a function of the two other random variables: N and X | W = f(N, X)

Tree Diagram for Calling Process



Part 2 — Designing a Monte-Carlo Simulation Algorithm

Random Number Generator

We implemented the *linear congruential random number generator* algorithm in our code defined by the following equations

- $x_i = (ax_{i-1} + c)(modulo\ K)$
- $u_i = decimal\ representation\ of\ \frac{x_i}{K}$

such that x_i is an intermediary value used as the seed for the next iteration, and u_i is the actual i^{th} random number in (0,1). We define the other quantities as follows:

- $x_0 = 1,000$
- $a = 24,693$
- $c = 3,517$
- $K = 2^{17}$

The first three random numbers of the cycle created by these initial conditions are 0.4195, 0.0425, and 0.1274. As a demonstration of the accuracy of our implementation, the values for u_{51}, u_{52}, and u_{53} are 0.5157, 0.4273, and 0.7682 respectively. We implemented this algorithm in python by defining two functions. The first is generate_random_number(int:int:int:int) → float that takes the initial conditions as parameters and returns the first random number in the cycle defined by those conditions. The second is generate_nth_random_number(int:int:int:int:int) → float, which takes the same initial conditions, as well as the n^{th} number to generate. This function iterates n times, calling generate_random_number() each time and passing the value of x_i as the seed value of each successive call. During testing, we realized that this was certainly not the most efficient way to calculate a series of random numbers, since each successive generation would effectively have to calculate each previous random number first. This fact makes our algorithm $O(n^2)$, and while not an issue for 1000 simulations, becomes a significant bottleneck at just one order of magnitude larger. Perhaps we could define a new function generate_sequence_of_random_numbers() that returns an array of floats which would operate in O(n).

Random Variable Generator

Next, we have to map our random number to a realization of a random variable. We defined X as a continuous exponential random variable representing the number of seconds it takes for the customer to pick up. We calculated the cumulative distribution function and inverse CDF of X as defined in part 1. We then defined a function get_realization_of_X(float) → float that accepted a random number and used the inverse CDF to map the random number to a realized value of X. That is to say, if our random number was 0.75, our function returns the value of X such that there is a 75% chance that the customer takes less than X seconds to pick up the phone. With that random value of X, we can derive a realization of W by simulating phone calls. We define a

function get_realizations_of_w(int) → [float] that simulates a given number of trials and returns an array of realizations of W. For every iteration of this function, we call the function get_realization_of_w(int) → float that accepts an integer representing the n^{th} position that specific realization occupies in a sequence of trials. This n^{th} position parameter is used in generate_nth_random_number() to offset each generation by the trial number. Inside this function, we call our third and final function simulateCall(int:int) → (float, bool) that simulates one single phone call. This function accepts the n^{th} position passed to get_realization_of_w() as well as an integer representing the call number (from 1 → 4) which is used to further offset the random number generation so as not to repeat realizations of X for one realization of W. This function uses our offset random number to traverse the first dimension of our tree diagram (deciding whether the call is received or missed (line is busy or open)), and calculate a realization of x that is used to traverse the second dimension (whether the customer picks up within 25 seconds). Our function adds up the seconds spent on each task, and returns a tuple containing the total length of the simulated call and a boolean representing whether a recall is needed. Our get_realization_of_w() function uses this boolean to decide whether to simulate another call, or return the total seconds spent across the N number of calls already simulated. Ultimately, we are left with an array of realizations of W.

Part 3 — Simulate the Calling Process

This was simply accomplished by running generate_realizations_of_w(1000). Our result is an array of realizations of W, which we can plot in a few ways to reveal key insights. To start, we'll plot a histogram showing the frequency of various realizations of W:

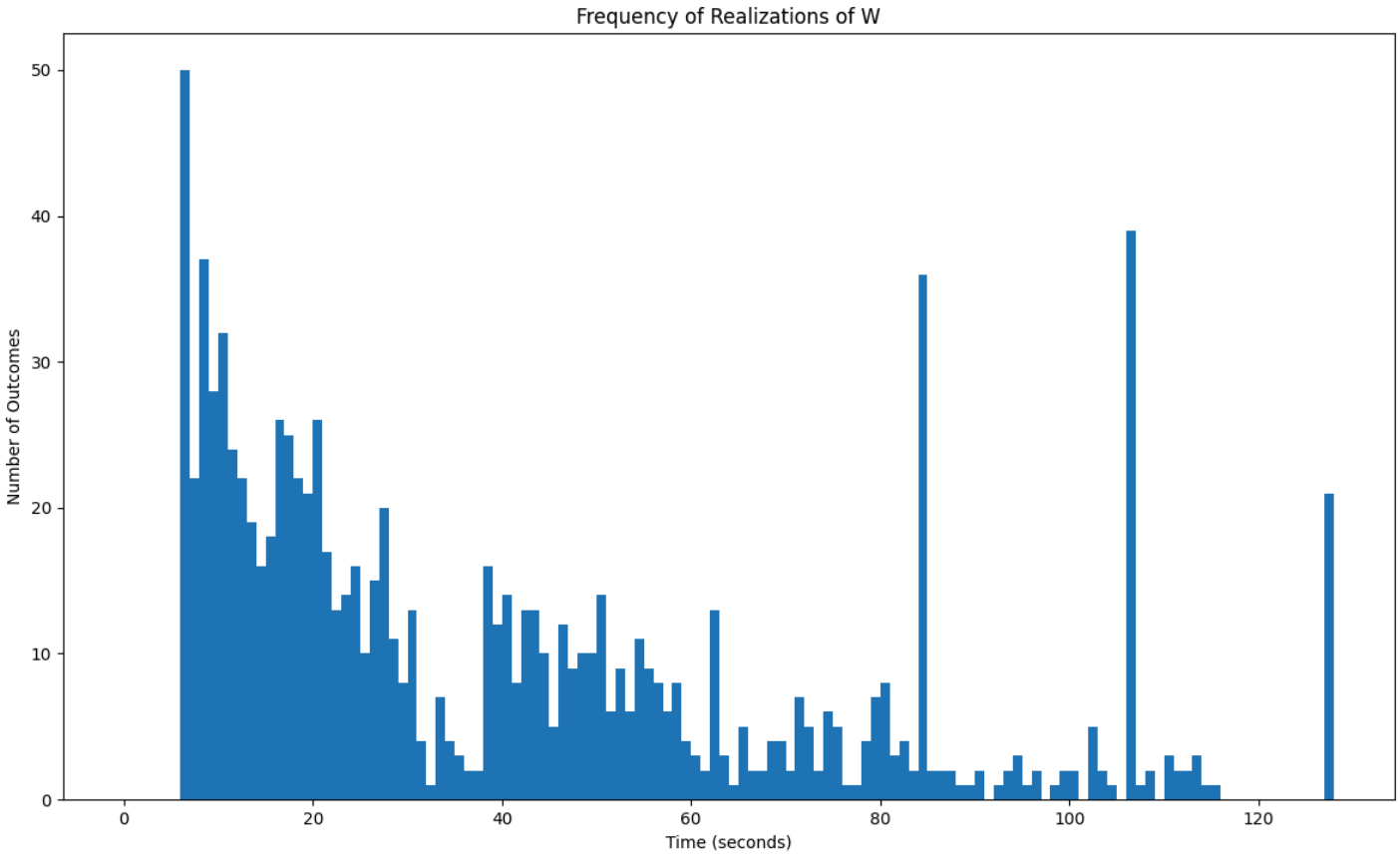
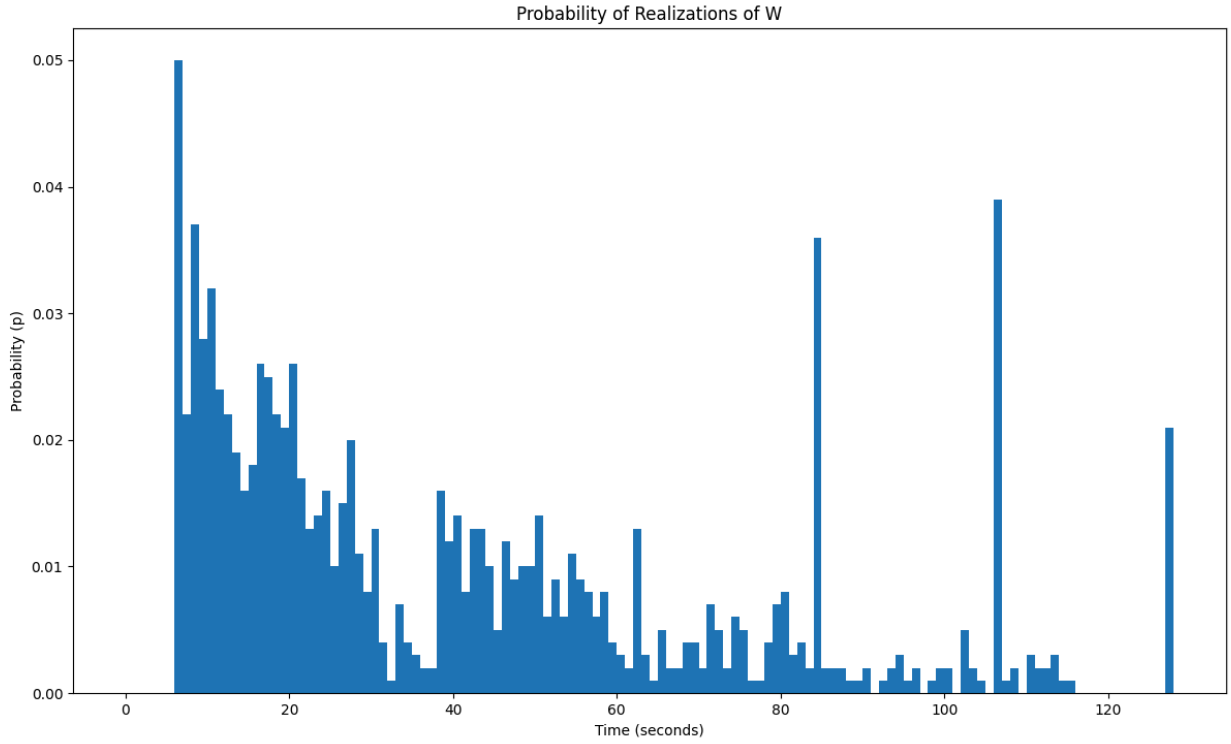


Figure 1: frequency distribution of 1000 random trials of the calling process

Next, we'll similarly plot a histogram, but scale the y-axis based on the total number of simulations so as to to get the probability of each realization of W within the context of our simulation:

Figure 2: frequency distribution of 1000 random trials of the calling process scaled by the



number of trials to display probability of each realization of W

We can take these plots one step further by noticing a correlation between random variables X and W. Since X denotes the time for a customer to pick up after being dialed, it follows that W should increase roughly at the same rate as X. We can see this exact relationship by overlaying an estimated PMF function for X on our graph of frequency distribution of W. While we realize that there is no concept of a PMF for a continuous random variable (since $P[X=x] = 0$) for all x, we can calculate the probability $p[t_l \leq X \leq t_h]$ where t_l and t_h refer to the lower and upper bounds of the bins used to plot the histogram of W. Effectively, this gives us an estimated graph of the "probability that X takes on some value between t_l and t_h, smoothed out to model a continuous function":

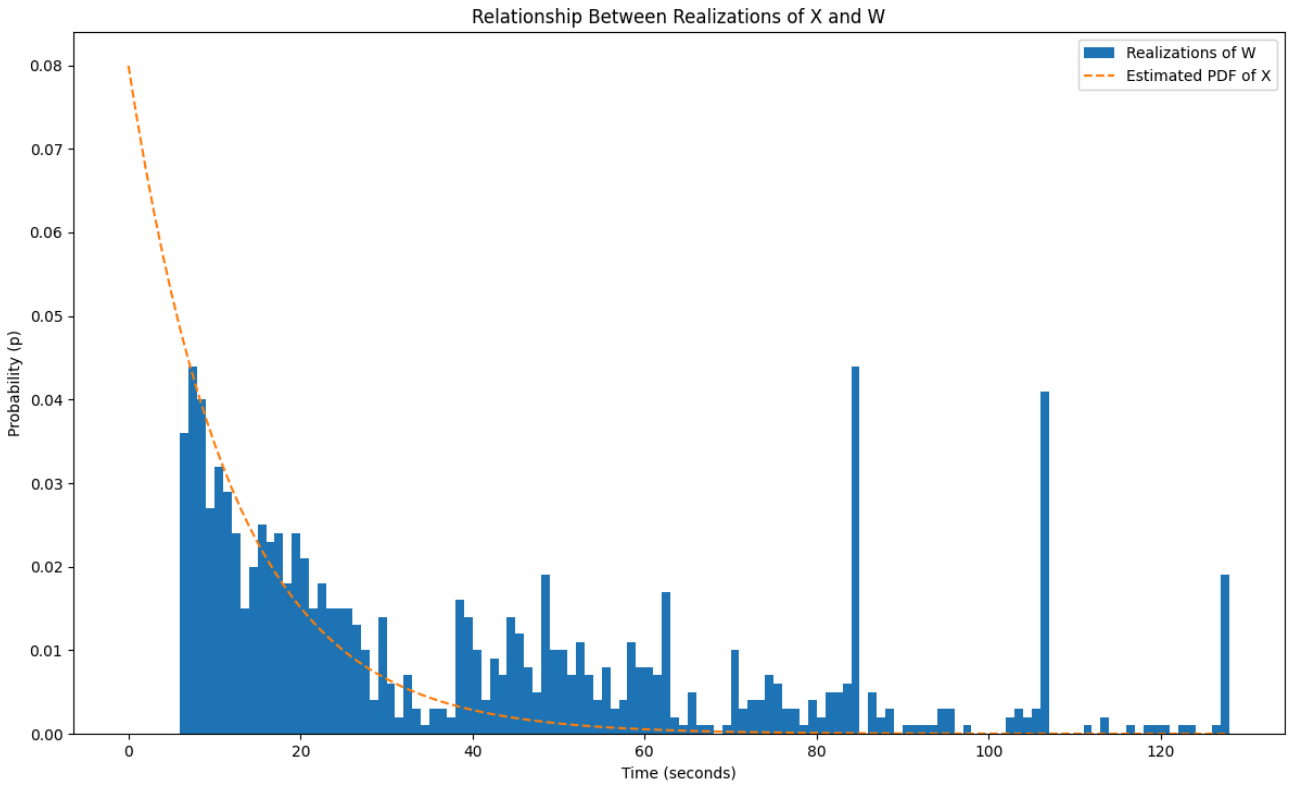


Figure 3: relationship between growth of W and X over 1000 trials

While outside the scope of this assignment, we can simulate more trials to get a closer approximation of the distribution of W. Right now, it can be hard to see a pattern in the distribution of W, but if we run our simulation where n = 1,000,000 instead of 1000, it becomes clear. Thus, simply to help explain the relationship between X and W, we simulate n = 1,000,000:

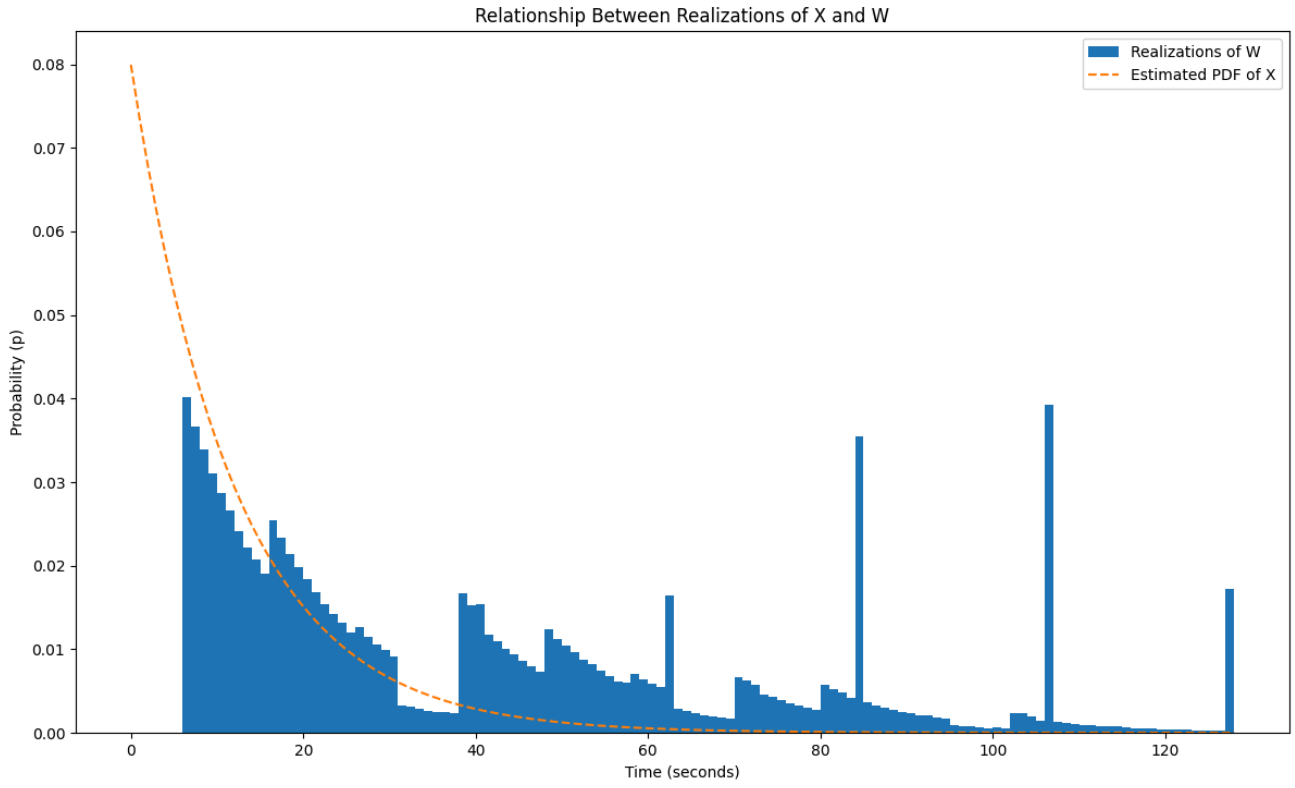


Figure 4: relationship between growth of W and X over 1,000,000 trials

As we can see, not only is the relationship between W and X much clearer now, but there is a distinct pattern in the probability for certain values of W. These repeated sections correspond to the 4 potential calls made. The probability of each corresponding value for each section of the pattern is lower as time increases, since it gets less and less likely that each successive phone call is made. Moreover, you see the jump up at the beginning of each section since random variable X resets every call and follows the same exponential decrease as time increases for each segment. Now to address the elephants in the room, w = 62s, w = 84s, w = 106s, and w = 128s. These realizations of W correspond to key traversals through the tree diagram. While most other values are dependent on the value of X, these 4 values correspond to the traversals that do not ever rely on X, and thus do not fit the X curve. Starting with w = 128s, this corresponds to all 4 calls taking 32 seconds (6 seconds to dial, 25 seconds to wait for 5 rings, 1 second to hang up). Each successively smaller value of W represents an additional one of the 4 calls being detected as busy, causing it to only take 10 seconds (6 seconds to dial, 3 seconds to detect busy signal, 1 second to hang up). That is to say, w = 84s is achieved by 2 full calls of 32 seconds, and 2 calls that are detected as busy. Since the range of X is (0, 25), customers who have an available line but miss the call add the same amount of time as customers who are not available. That is why these are the only 4 values that do not fit into the pattern, because every other traversal through the tree relies on X. Finally, given our expected value of X, E[X] = 12, we can see that in each repeated segment, there is a jump halfway through. This corresponds to the 6 seconds it takes to dial plus the 12 expected seconds it takes for the customer to pick up. That is to say, the jumps occur at w = 18s, w = 50s, w = 82s, w = 114s (all separated by 32 seconds, which is one additional phone call that was missed).

Part 4 — Estimate Statistics for W

We ran getRealizationsOfW(10000), i.e. 10000 individual calls.

Mean

The mean of our samples of W was ~ 40.668 seconds.

First Quartile

The first quartile realization of W for our samples was 14.366 seconds

Median

The median realization of samples of W was 28.845 seconds

Third Quartile

The third quartile realization was 59.58 seconds

Probabilities of [W ≤ w]

P[W ≤ 15] = 0.263
P[W ≤ 20] = 0.376
P[W ≤ 30] = 0.512
P[W > 40] = 0.430

Custom Values of w

To depict the right tail of the CDF of W, we chose realizations $w_5 = 50$, $w_6 = 115$, $w_7 = 125$. These values were chosen to avoid the jumps at w = 84, 106, and 128
P[W > 80] = 0.163
P[W > 111] = 0.026
P[W > 126] = 0.017

Part 5 — Analysis of Statistics for W

The mean is much greater than the median. This indicates a strong right skew in the PDF of W. This is because the outliers on the right side of the graph pull the mean higher, but they do not affect the median.
The sample space of W is [6, 128]. 6 seconds is achieved if the customer picks up immediately on the first call. 128 seconds occurs if the phone rings but is not answered each of the 4 calls. W can be represented as an exponential random variable with $\lambda \approx 0.03$. The CDF does have many steps and discontinuities, as seen in figure 5.

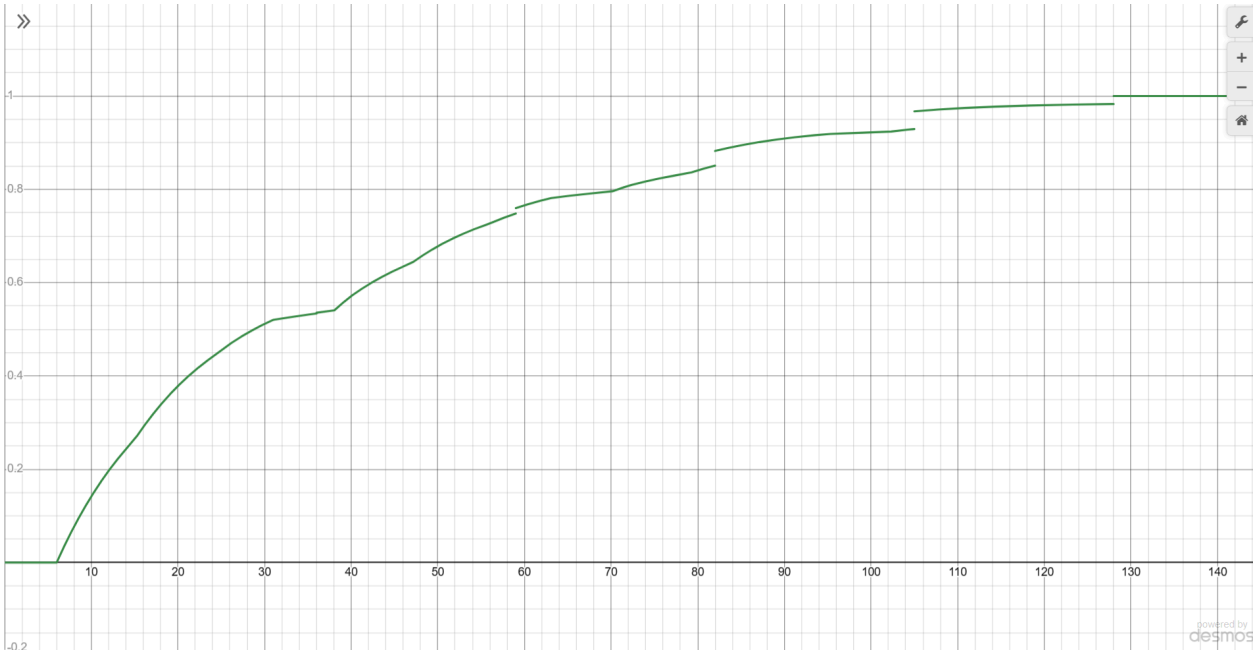


Figure 5: CDF of W

However, despite these jumps, the graph resembles the form of an exponential random variable's CDF, which would fit the form 1 - e ^ (-λx). Figure 6 displays the CDF overlaid with the function y = 1 - e ^ (-0.03(x-6)).

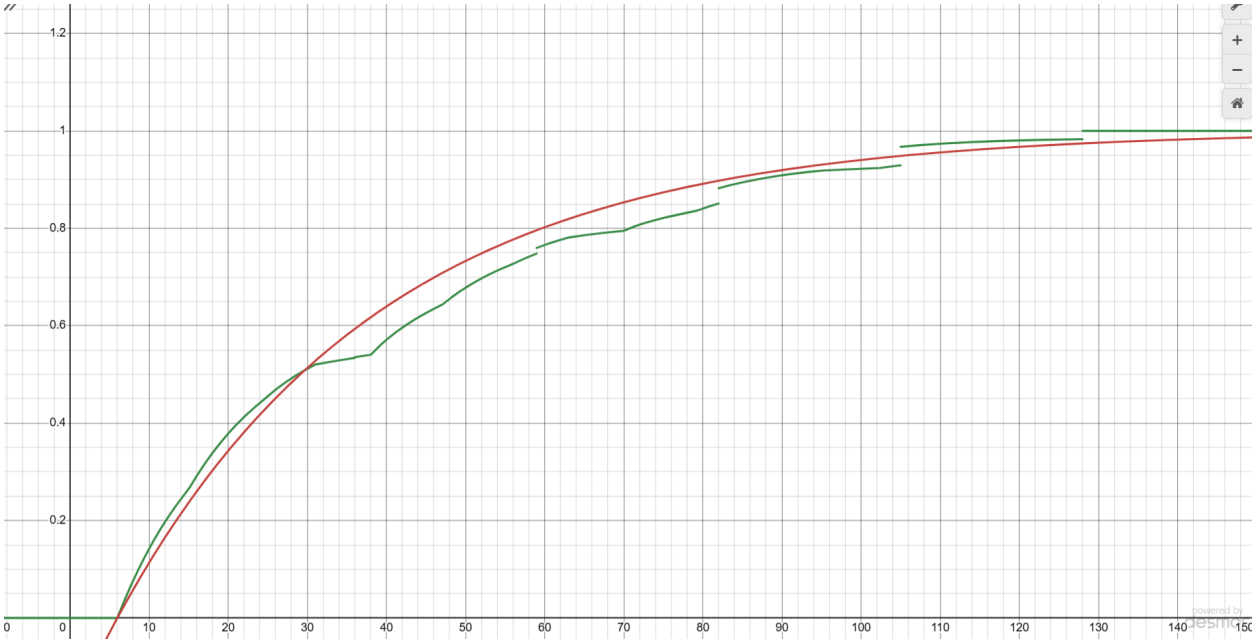


Figure 6: CDF of W with Exponential Random Variable CDF λ = 0.03

We chose w_1, w_2, and w_3, avoiding the large jumps in the graph. We see P[40<W<80] = P[W>40]-P[W>80] = 0.267. P[0<W<40] = 1 - P[W>40]. We see P[40<W<80] < P[0<W<40], which shows that within a range of 40 about the mean, the probability is higher for W to be less than the mean. Similarly, we see P[111<W<126] = 0.009 < P[W<15]. This shows that the highest 15 seconds (excluding the jump at 128) is less likely than the lowest 15 seconds. This shows a right skew in the PDF.

This skew is also seen in the 1st and 3rd quartiles. The 1st quartile is ~14 seconds below the median. The 3rd quartile is ~30 seconds above the median. This shows The 3rd quartile must encompass a greater range because W is less likely to be there.

Part 6 — Discussion

Caden Kowalski

I personally think that this project was really cool. It's fascinating to see real world results that match your intuitive thinking about the results of an experiment. For instance, it makes sense in your head that the value of W should be related to the value of X, and the graphs show precisely that relationship. In general, Monte Carlo simulation seems like a very practical approach for modeling real world situations for which you don't have a precise probability model, but do have information about various outcomes. That being said, the most challenging part of this project for me was writing the code that would simulate the calling process. I probably made it more difficult on myself by trying to make it perfect (ie. trying to use proper code design principles) rather than writing something that would be ugly but functional. Nevertheless, even with a tree diagram and well defined variables and events, I found it challenging to translate that into an algorithmic simulation. The least challenging part was probably the majority of part 1 (bar creating the tree diagram). It was relatively easy to define the events and the equations for the probability models of X and N. The most time consuming part of the project, on the other hand, was probably making sense of the graphs. Once I was at least sure that my code was working and giving me correct results, it took me quite a while to make sense of them. I knew intuitively that there should be a relationship between X and W, but I just couldn't quite grasp it right off the bat. It also took me a long time to understand why there were huge jumps at w = 62s, 84s, 106s, and 128s. I thought for a long time that I had messed something up in the code. Now that I do understand it though, I feel like I have a much deeper understanding of this specific calling process.

Ted Nyberg

I thought determining the CDF was the most challenging part. I tried to solve it, by making a recursive piecewise function, but it was too complicated. Eventually, I brute forced it with a long tree diagram, then I plotted everything on Desmos and looked at it from there. For these same

reasons, this was the most time consuming step. I also solved for the CDF during the 1st part of the project when we were supposed to solve for X's CDF, not W's. It was also difficult to understand how the various components of the simulation worked together, i.e. how the random number (0,1) could turn into a discrete number or a trial that follows a given CDF. The easiest step was solving for X's CDF when I realized that was what I had to do. Seeing the distribution graph of W was the most fun part of the project. Being able to see how each scenario affected the distribution and seeing how it fell into shape with more trials was very interesting.