Finding the Distribution of a Simple, Probability/Dice Based Game

Ina Holtschmit

Georgia Tech Institute of Technology

25 April, 2023

Abstract

Using R Studio, a simple game was programmed that is dependent on two players (Player A and Player B) tossing a die that results in them losing or winning coins. A cycle of the game is defined as both players A and B having a turn. From a Monte Carlo simulation (repeated sampling), the amount of cycles that it takes for one of the players to not be able to complete a turn, which results in one full game played, is estimated. From a simulation of 100,000 games, the amount of cycles it takes for a game to be completed follows an exponential distribution, with a mean of 17 and rate/lambda of 0.0587.

Introduction

Dice have long been used to explain mathematical distributions. Since a die only has 6 sides (numbers 1 through 6), rolling it results in the outcome of a discrete random variable, and follows a uniform distribution since each of the six numbers have an equal chance of being rolled. However, since the game is dependent on other rules, it ends up following a continuous distribution.

In the methods below, a game made up of two players, a die, and several rounds, is described. The objective of the write up below is to find the mean number of rounds, or cycles, that a game lasts. Because we are waiting for the first time a failure event happens, cannot have negative values, and not bound by an upper limit, an initial educated guess of possible distributions the results would follow are the following: exponential, gamma, log-normal, and Weibull.

To get an accurate estimate of how many cycles are played in one full game by the two players, a Monte Carlo simulation can be run. This is a standard mathematical technique that predicts an outcome that is based on random variables (such as a 6-sided die). One simulation contains many samples of the same process, repeated over and over, to get a large sample size from which results can be extracted.

Methods

The premise of the game, as outlined in the PDF supplied by Professor Dave Goldsman, is as follows:

- There are two players in the game, Player A and Player B, and one pot
- The players start the game with 4 coins each, and the pot contains two coins
    - There are a total of 10 coins in the game.
- The players roll the die in sequential order, where one cycle is defined as both Player A and Player B taking their turn rolling the die, the exception being the last round, where a Player is unable to complete their turn—this still counts as one cycle.

- Using a 6-sided dye, if the player rolls a:
    - 1: the player does nothing
    - 2: the player takes all the coins in the pot
    - 3: the player takes half of the pot (rounded down)
    - 4, 5, or 6: the player puts a coin in the pot
- Once a player is unable to complete their turn, meaning they roll a 4, 5, or 6 and have 0 coins, meaning they cannot place a coin of their own in the pot, the game is finished.
- The following assumptions are being made:
    - If there are 0 coins in the pot but a player rolls a "2", 0 is added to their personal stash of coins, and the game continues.
    - If a there a 0 coins in the pot but a player rolls a "3", this was programmed as 0 being divided by 2, resulting in 0 being added to their personal stash of coins, and the game continues.
    - Even if a player has 0 coins, they are still in the game as long as they roll a 1, 2, or 3 during their next turn.

The programming software, R Studio, was used to program the rules of the game described above (the main code can be seen in the appendix below, while the full code is attached in the zip file) using a series of loops, functions, and conditional statements. Additionally, the libraries "ggplot2", and "MASS," were used to plot the results and extract specific parameters relevant to a gamma distribution. Four loops were used to run the Monte Carlo simulations: each one for the respective number of runs/samples: 100, 1000, 1,000, 10,000.

Lastly, alpha and beta was found for each simulation using the "fitdistr()" function and the "exponential" parameter. The mean, standard deviation, and variance, was calculated for each of the four simulations and stored in a data frame. The following equations were used:

Mean:

$$\mu = \frac{1}{\lambda}$$

Standard Deviation:

$$\sigma = \sqrt{\frac{1}{\lambda^2}}$$

Results

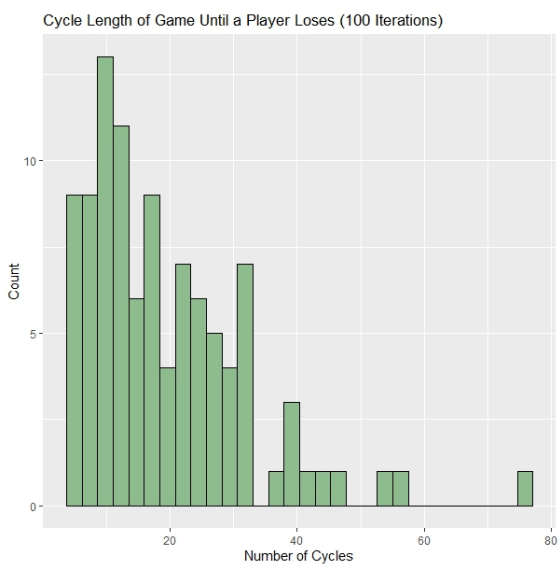
From Table I, we can see that aside from the mean, lambda and standard deviation did not change very much from the simulation of 100 samples to 100,000 samples. By the time we 10,000 samples were obtained, all parameters were fairly accurate. From the simulation of 10,000 games, to the simulation of 100,000 games, lambda only increased by 0.0004, mean decreased by 0.012, and the standard deviation decreased by 0.005.

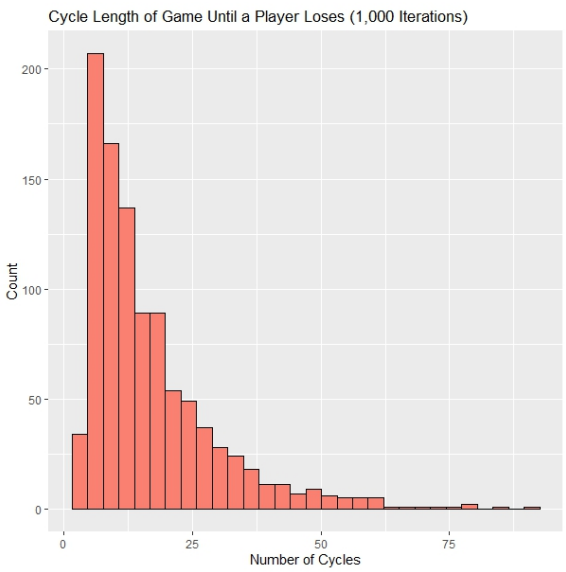
Figures I through V, show the distribution of the number of cycles a game takes for each of the four simulations. The simulation of 100 does not exhibit an obvious distribution when looking at its histogram. The next histograms look to follow an exponential distribution more and more.

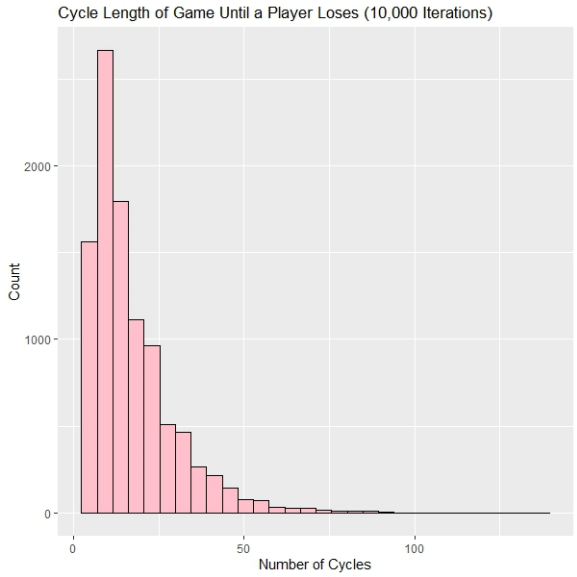
Table I. Final Parameters of Simulations

Number of Runs in Simulation	Lambda	Mean	Standard Deviation
100	0.0513	19.49	12.68
1,000	0.0604	16.55	12.74
10,000	0.0583	17.16	12.83
100,000	0.0587	17.04	12.77

Figures I to V. Histograms of Each Simulation (Left to Right, Top to Bottom)







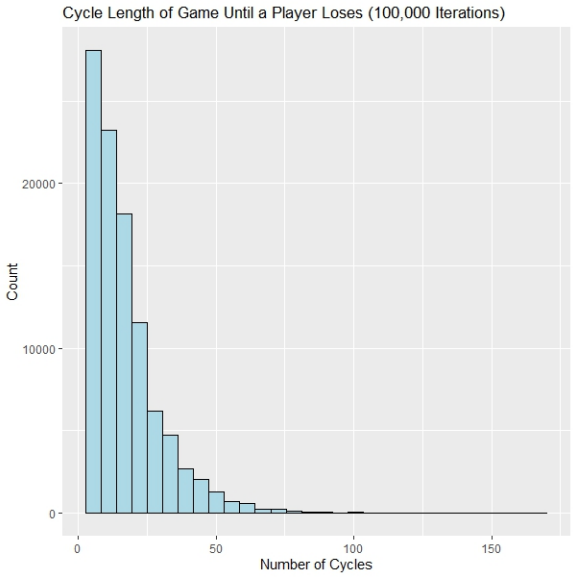
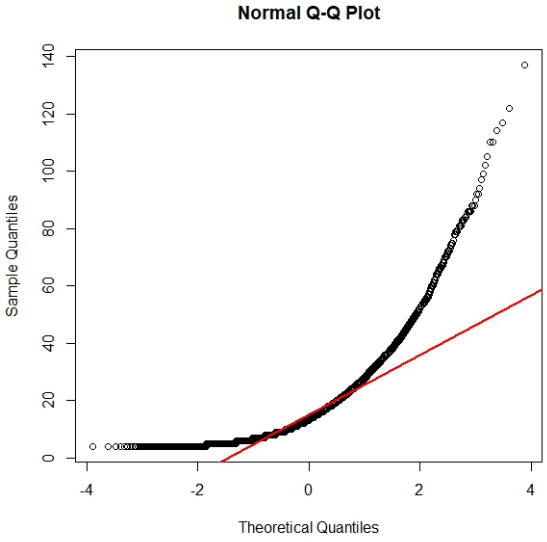
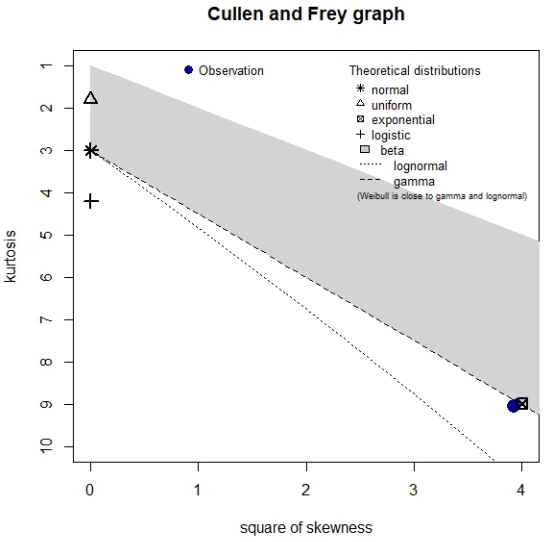


Figure VI. Quantile Plot of 10,000 Run Simulation

Figure VII. Cullen and Frey Plot





To reinforce the claim that the number of rounds a game lasts follows an exponential distribution, a quantile graph was plotted (Figure VI). In the figure, we can see that the sample quantiles follow those of the exponential distribution. Additionally, a Cullen and Frey graph was plotted (Figure VII), which shows that the observed values of the 100,000 simulation follow an exponential distribution. The point for the observed values can be seen in the bottom right corner in blue, nearly overlapping the those for the control exponential distribution, as their square of skewness if very similar. Looking at the

histograms of the 1,000 and 10,000 simulations, they seem to follow a gamma distribution. From Figure VII, we can see that it nearly overlaps the line for the gamma distribution as well.

Conclusion

Something that I learned from this project is how easy it is to assume that your results follow a certain distribution from histograms of their probability density functions, but the more you look into them, the less certain you become. I learned that you cannot always trust your first instinct, and even if the results are intuitive, you must look for various methods to back up your claim and not just rely on one.

Appendix

PLayers A and B starting with 4 coins, pot with 2

```
player_A <- c(4)
player_B <- c(4)
players <- c(player_A, player_B)
pot <- c(2)
```

Function for a single game

```
playing_one_game <- function(player, pot) {
 i <- sample(1:6)[1]
 if (i == 4 | i == 5 | i == 6) {
  if (player >= 1){
   player <- player - 1
   pot <- pot + 1
  } else {
   return(c(0, player, pot))
  }
 } else if (i == 2) {
  player <- player + pot
  pot <- 0
 } else if (i == 3) {
  amount <- floor(pot / 2 )
  player <- player + amount
  pot <- pot - amount
 } else if (i == 1) {
  player <- player
  pot <- pot
 }
 return(c(1, player, pot))
```

```
}
```

How many games until a player no longer has coins

```
cycle_length <- function(players, pot){
  game_list <- 0
  while (result[1] != 0) {
    result <- playing_one_game(players[1], pot)
    players[1] <- result[2]
    pot <- result[3]
    if (result[1] == 0){
      break
    }
    result <- playing_one_game(players[2], pot)
    players[2] <- result[2]
    pot <- result[3]
    game_list <- game_list + 1
  }
  return(game_list)
}
```

First run to see whether the loop works

```
final_results_1000 <- c()
for (i in 1:1000){
  length_result <- cycle_length(players, pot)
  final_results_1000[i] <- length_result
}
```