On The Monty Hall Problem: An Abstraction Through Monte Carlo

Jeffrey Cheng

06-03-19

1 Introduction

This paper provides various methods of looking at the Monty Hall Problem. We will first look at the base case, in which there are 3 doors, intuitively. Then, we will go over the Monte Carlo Simulation for the base case. After that, we will look at cases where the number of doors increases to an arbitrary n number of doors. Then, we will go over the Monte Carlo Simulation for these cases.

For those unfamiliar with the problem, it is stated here: Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

2 Base Case: $n = 3$

2.1 Initial Explanation

The choice that Monty is giving you is actually a choice between your original door and the 2 remaining doors, as if he had not opened any door at all. The two doors that were not initially chosen either contain I : a goat and a car or II : two goats. Monty opens a door and shows you a goat. It was already known that one of the other two doors contained a goat (regardless of which door you first chose) so Monty effectively has not given you any "new" information. What he has done, however, is created a situation in which picking the last door is effectively the same as picking both doors that were not chosen initially. This means that you are effectively making a choice between one random door and two random doors, which give probabilities of 1/3 and 2/3 respectively. So the right answer is that you should switch.

2.2 Axiomatically

It may also be helpful to look at this through the axioms of probability.

§1.1

I $P(A) \geq 0 \; \forall \; A \in S$

II $P(S) = 1$

III $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Let A_1, A_2, A_3 be the event that the car is behind either door 1, 2, or 3 respectively. It is trivial to see that $P(A_1) = P(A_2) = P(A_3) = 1/3$. Let A_p be the event that the door the player chose is the door the car is behind, with p being either 1, 2, or 3. $P(A_p) = 1/3$: $P(A_p)$ is either $P(A_1)$, $P(A_2)$, or $P(A_3)$, which all have probabilities of 1/3. In the initial state:

$$\sum_{x=1}^{3} P(A_x) = 1$$

Now, lets see how this changes when the host opens one of the doors.

Let A_{np} be the event that the car is behind the door that is closed that was not chosen by the player. In our new, "secondary" state, $A_p + A_{np} = S$. By Axiom II, $P(S) = 1$. $P(A_p) = 1/3$: this is not changed from the initial state. So:

$$\begin{aligned} 1/3 + P(A_{np}) &= 1 \\ P(A_{np}) &= 1 - (1/3) \\ P(A_{np}) &= 2/3 \end{aligned}$$

2.3 Monte Carlo Simulation

I created a Monte Carlo Simulation for this situation.

~~~
```{r}
sample_size <- 10000
count <- 0

varSamp <- function(x) {
  if (length(x) <= 1) {
    return(x)
  } else {
    return(sample(x, 1))
  }
}
~~~

In this part, a sample size for the simulation is created and a function to sample a list regardless of length is created. The preset "sample" function has issues sampling from a list of size 1, which may occur in this simulation.

```
vals <- 1:3
for(i in 1:sample_size) {
  doors <- array(0, dim = c(1, 3))
  #set a random door to have the car behind it (the number 1)
  car_door <- sample(vals, 1, replace = TRUE)
  doors[car_door] <- 1
  #player picks a random door
  init_door <- sample(vals, 1, replace = TRUE)
```

The structure of the simulation is created here. Three doors are represented as an array of three elements, all initiated to 0. The door with the car behind it is randomly chosen and the respective door is incremented in value by one. The same is done for the door that the player chooses.

```
  can_open <- 1:3
  indexes <- c(init_door, car_door)
  indexes <- sort(indexes, decreasing = TRUE)
  if(indexes[1] != indexes[2]) {
    for(i in 1:2) {
    can_open <- can_open[-indexes[i]]
    }
  } else ]
    can_open <- can_open[-indexes[1]]
  }
```

This code processes which doors can be opened by the host (sorts out which doors are not already chosen by the user and do not have the car behind them).

```
opened_door <- varSamp(can_open)
  new_door <- 1:3
  indexes2 <- c(init_door, opened_door)
  indexes2 <- sort(indexes2, decreasing = TRUE)
  for(i in 1:2) {
    new_door <- new_door[-indexes2[i]]
    }
```

The door to be opened is chosen.

```
doors[new_door] <- doors[new_door] + 1
  if(doors[car_door] == 2) {
    count <- count + 1
  }
}
print(count/sample_size)
```

The switch is made. If any door in the array has a value of 2, then the car door and the player's door are the same and a win is represented, causing the program to increment the count of wins by 1. If the seed is set to 100 (set.seed(100)) and a sample size of 10000 is used, the result is 0.6647.

3 n doors

3.1 Axiomatically

We can extend the argument for 3 doors that used the axioms of probability (See: §1.1) to prove that for n doors, the probability of success after switching is $n - 1/n$. Let $n \in \mathbb{Z}$ be the number of doors available to choose and let D be the set of all n. Let $A_1, A_2, A_3, \ldots, A_n$ be the event that the car is behind either door 1, 2, 3, …, n respectively. It is trivial to see that $P(A_1) = P(A_2) = P(A_3) = \ldots = P(A_n) = 1/n$. Let A_p be the event that the door the player chose is the door the car is behind, where p $\in D$. $P(A_p) = 1/n$. In the initial state,

$$\sum_{x=1}^{n} P(A_x) = 1$$

Let A_{np} be the event that the car is behind the door that is closed that was not chosen by the player. In our new, "secondary" state, $A_p + A_{np} = S$. By Axiom II, $P(S) = 1$. $P(A_p) = 1/n$: this is not changed from the initial state. So:

$$\begin{aligned} 1/n + P(A_{np}) &= 1 \\ P(A_{np}) &= 1 - (1/n) \\ P(A_{np}) &= (n-1)/n \end{aligned}$$

3.2 Monte Carlo Simulation

I created a Monte Carlo Simulation for this situation as well.

~~~
```{r}
sample_size <- 10000
count <- 0
n <- 5

#function to sample regardless of length of list of available doors
varSamp <- function(x) {
  if (length(x) <= 1) {
    return(x)
  } else {
    return(sample(x, 1, replace = FALSE))
  }
}
~~~

This part is largely the same. The only new addition is that a variable for n, the number of doors, is initialized.

```
vals <- 1:n
for(i in 1:sample_size) {
  doors <- array(0, dim = c(1, n))
  #set a random door to have the car behind it (the number 1)
  car_door <- sample(vals, 1, replace = TRUE)
  doors[car_door] <- 1
  #player picks a random door
  init_door <- sample(vals, 1, replace = TRUE)

  can_open <- 1:n
  indexes <- c(init_door, car_door)
  indexes <- sort(indexes, decreasing = TRUE)
  if(indexes[1] != indexes[2]) {
    for(i in 1:2) {
    can_open <- can_open[-indexes[i]]
    }
  } else {
    can_open <- can_open[-indexes[1]]
  }
```

This section is the same, with the arrays and sampling methods adjusted appropriately for having n doors.

```
opened_doors <- list()
  for(i in 1:(n-2)) {
    door <- varSamp(can_open)
    opened_doors <- c(opened_doors, door)
    can_open <- can_open[which(can_open != door)]
  }
  new_door <- 1:n
  indexes2 <- c(init_door, opened_doors)
  indexes2 <- as.numeric(as.character(unlist(indexes2)))
  indexes2 <- sort(indexes2, decreasing = TRUE)
  for(i in 1:(n-1)) {
    new_door <- new_door[-indexes2[i]]
    }
```

The door to be opened is chosen. This section required the most revision. This number of doors are opened is more than one, and therefore must be created in list format. Furthermore, the list of doors that can be opened must constantly be revised as doors are chosen so that a door is not repeatedly opened.

```
doors[new_door] <- doors[new_door] + 1
  if(doors[car_door] == 2) {
    count <- count + 1
  }
}
print(count/sample_size)
```

This section is the same as well. The switch and deterministic calculation are made.

4 Conclusion

In this paper, I showed that it is advantageous to switch doors in two ways. After that, I showed my Monte Carlo Simulation for the case where $n = 3$. I then used the same axiomatic argument for $n = 3$ to show that a more general result holds for any n. I then showed my Monte Carlo Simulation for this more general case.