title: “November Post - Post 2”
author: Jessica Cherny
output: html_document
keep_md: true

Post 2: A Deeper Dive into the Subject of Monte Carlo Simulations

Introduction:

Did anyone else in the class read the syllabus from a couple weeks back and see the topic of “Monte Carlo Simulations”? Did you have your interest peaked at the connotation of bringing something as glamorous as Monte Carlo to the world of statistical programming? Me too! However, we only scratched the surface of Monte Carlo simulations in class, and I was left unsatisfied not being able to fully and formally explain what a Monte Carlo simulation even was. As I was looking over the syllabus for Data100 just for fun, I noticed a familiar topic being taught– Monte Carlo simulations! However, I only remembered seeing it in my Stat133 syllabus. I thought to myself that this must be an important topic in the world of Data Science, so I decided to dive deeper into the topic and write a post about it in order to educate myself and my classmates on this cool topic! Here we go.

Background

So what is a Monte Carlo simulation and what are they used for?
The name “Monte Carlo simulation” is inspired by the games of chance popular in Monte Carlo, like roulette, slot machines, and dice. Monte Carlo simulations are used to “model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables” (Investopedia). Monte Carlo simulations are used in optimization and generating draws from a random probability distribution. In theory, Monte Carlo simulations can be used to solve any problem with a blueprint determined by probabilistic outcomes.

Examples:

Monte Carlo simulations can be applied to many fields like finance, business, astronomy, oil and gas, environmental economics, physics, meteorology, computer graphics, computational biology, applied statistics, and artificial intelligence. We can model the movement of the price of an asset using Monte Carlo simulations and historical data, model probabilities of a certain event happening (like the probability of getting a certain number of heads in a certain amount of coin flips), and can model the best move to make in a game using Monte Carlo simulations and Monte Carlo tree searches.

How it Works:

A Monte Carlo simulation works by building models from a probability distribution given by a situaion that has inherent uncertainty. Common probability distributions include: Normal (bell curve), uniform, lognormal (where values are positively skewed), discrete, and triangular (max, min, and most likely are specified). The simulation then calculates whatever results the researcher is interested in over and over again from the specified probability distribution.

Code:

Suppose we have three dies, each with 6 sides. If we want to see the probability that their sum is at least 10, we can run a simulation 10000 times of rolling three dies and finding the probability that their sum is at least 10. Let’s do this now.

This code is taken from Stanford’s Bios 221 Lab 3.

```
isEvent = function(numDice, numSides, targetValue, numTrials){
  apply(matrix(sample(1:numSides, numDice*numTrials, replace=TRUE),
               nrow=numDice), 2, sum) >= targetValue
}

set.seed(0)
outcomes = isEvent(3, 6, 10, 10000) #three dies, 6 faces, at least 10 sum, 10000 times
mean(outcomes)
```

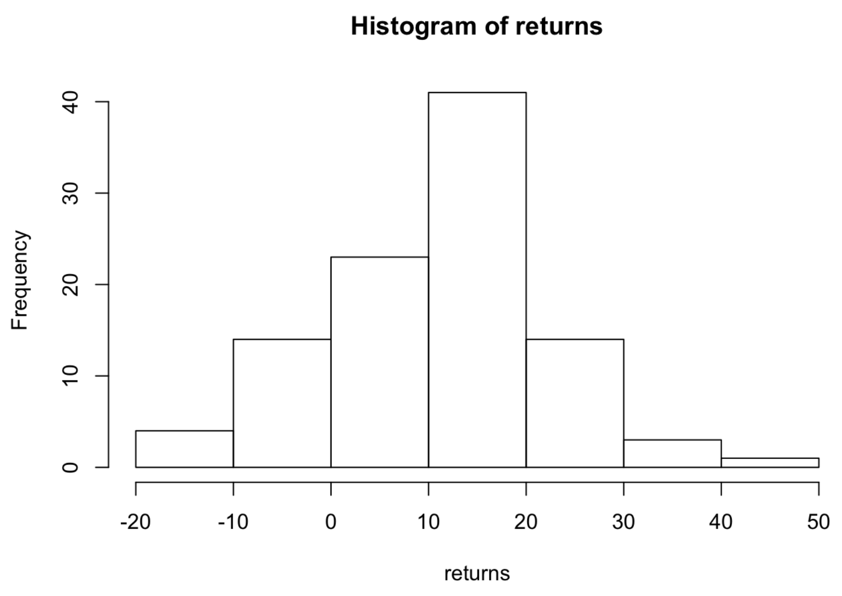
```
## [1] 0.6248
```

Say we have a stock with a normal distribution on the rate of return. The mean rate of return is 11%, and the standard deviation is 13%.

```
set.seed(1729)

# Generate 100 values
returns = rnorm(1:100, mean = 11, sd = 13)

 # plot a histogram
hist(returns)
```



We can assume that our return is our return per year. If we want to see our return in 5 years, we can increase the size parameter to 5.

```
sample(returns,size = 5)
```

```
## [1]  2.666618 12.616386 -3.560977 12.246473 10.073030
```

We can do this 10000 times and get a more accurate distribution of the mean rate of return for 5 years.

```
return_mat = matrix(ncol = 5, nrow = 10000)
for(year in 1:ncol(return_mat)) {
# for each year sample the return 10000 times
 for(i in 1:10000){
  return_mat[i,year] = sample(returns,1)
 }
}
head(return_mat)
```

```
##            [,1]       [,2]      [,3]      [,4]      [,5]
## [1,]  9.634693   7.124639 23.614901 29.738314  4.830532
## [2,] 14.324362  12.246473 12.246473 -6.999543 13.233281
## [3,] 26.958906 -13.592686 13.593467 36.511362  8.061349
## [4,]  3.769540 -18.194901  9.634693 24.854026  3.769540
## [5,]  3.569901 -10.346209 23.892870 23.254393 26.958906
## [6,] 12.616386  -9.183669 14.746880  8.345495 23.254393
```

I'll examine all the means of the columns.

```
apply(return_mat,2,mean)
```

```
## [1] 11.04399 10.94296 10.83140 10.82787 10.82143
```

We can see here that they are all pretty close to 11, following the specified normal distribution from above after sampling many times due to the Law of Large Numbers. According to Whatis.tecttarget.com, the Law of Large Numbers can be explained as: "a principle of probability according to which the frequencies of events with the same likelihood of occurrence even out, given enough trials or instances. As the number of experiments increases, the actual ratio of outcomes will converge on the theoretical, or expected, ratio of outcomes."

Another use for Monte Carlo Simuluations is to estimate the probability of getting more than 8 heads in 15 coin flips.

```
#number of trials
n = 100000

#heads is 1 and tails is 0
#returns true is the number of heads is greater than or equal to 8
flip_coin <- function(){
  sum(sample(c(0,1),15,replace=T)) > 8
}

#repeat this coin tossing n times and average out the probabilities
mc_binom <- sum(replicate(n,flip_coin()))/n
mc_binom
```

```
## [1] 0.30419
```

We can compare to R's builtin Binomial distribution function to see how our answer compares to the theoretical outcome.

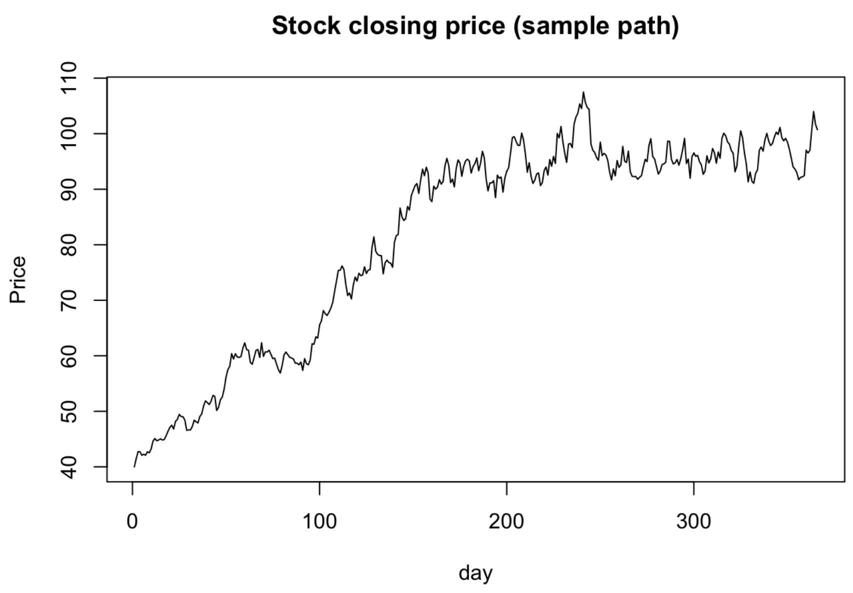
```
pbinom(8,15,0.5,lower.tail=FALSE)
```

```
## [1] 0.3036194
```

You see how close they are! Cool, right?

Lastly, we can model more stock performance in this mroe detailed example. Say a company has just IPO'ed. We can model its forecasted performance over a certain amount of days. Let's look at predicted performance over a year (365 days). This stock gains 1.003 times its opening price day every day during trading on average, with standard deviation 0.02 (this could represent the volatility of the market). We can use Monte Carlo simulations to model a stock's performance that opens up at $40 per share by taking a cumulative product from a Normal distribution.

```
days = 365
changes <- rnorm(days,mean=1.003,sd=0.02)
plot(cumprod(c(40,changes)),type='l',ylab="Price",xlab="day",main="Stock closing price (sample path)")
```



Now, we need to use Monte Carlo simulations to get accurate bounds on the closing price of a stock at the end of a year in order to assess risk. We can simulate 10000 times to get a distribution of closing prices.

```
runs <- 10000
#simulates future movements and returns the closing price on day 365
gen_stock_path <- function(){
  days <- 365
  changes <- rnorm(days,mean=1.003,sd=0.02)
  path <- cumprod(c(40,changes))
  close_price <- path[days+1] #+1 because we add the opening price
  return(close_price)
}

closing <- replicate(runs,gen_stock_path())
print(median(closing))
```

```
## [1] 111.5633
```

```
print(quantile(closing,0.05))
```

```
##       5%
## 59.47313
```

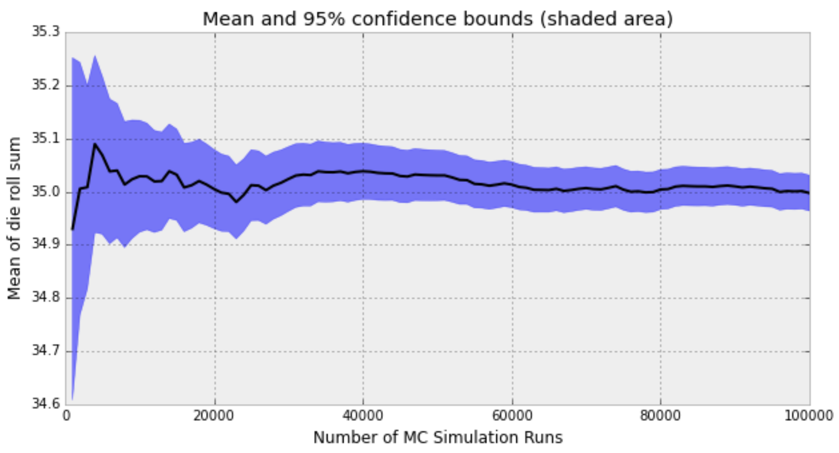
```
print(quantile(closing,0.95))
```

```
##      95%
## 207.2785
```

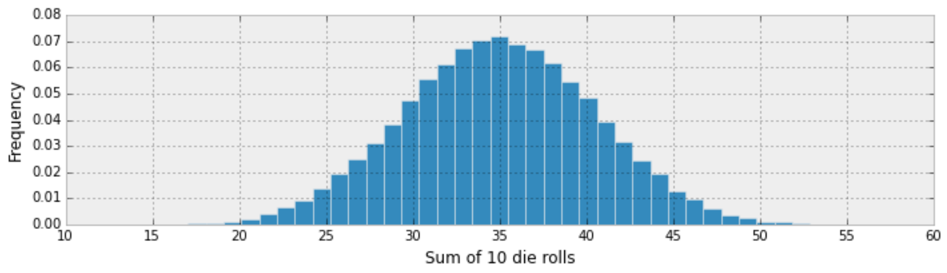
We can see that the median closing price of this fictitious stock is $111.12, with lower and upper bounds of $58.64 and $207.04.

Images:

Here's an image of the convergence of 10 dice rolls and the mean sum of that after 100,000 simulation runs. The theoretical mean sum is 35.



Here’s a histogram display of that mean sum.

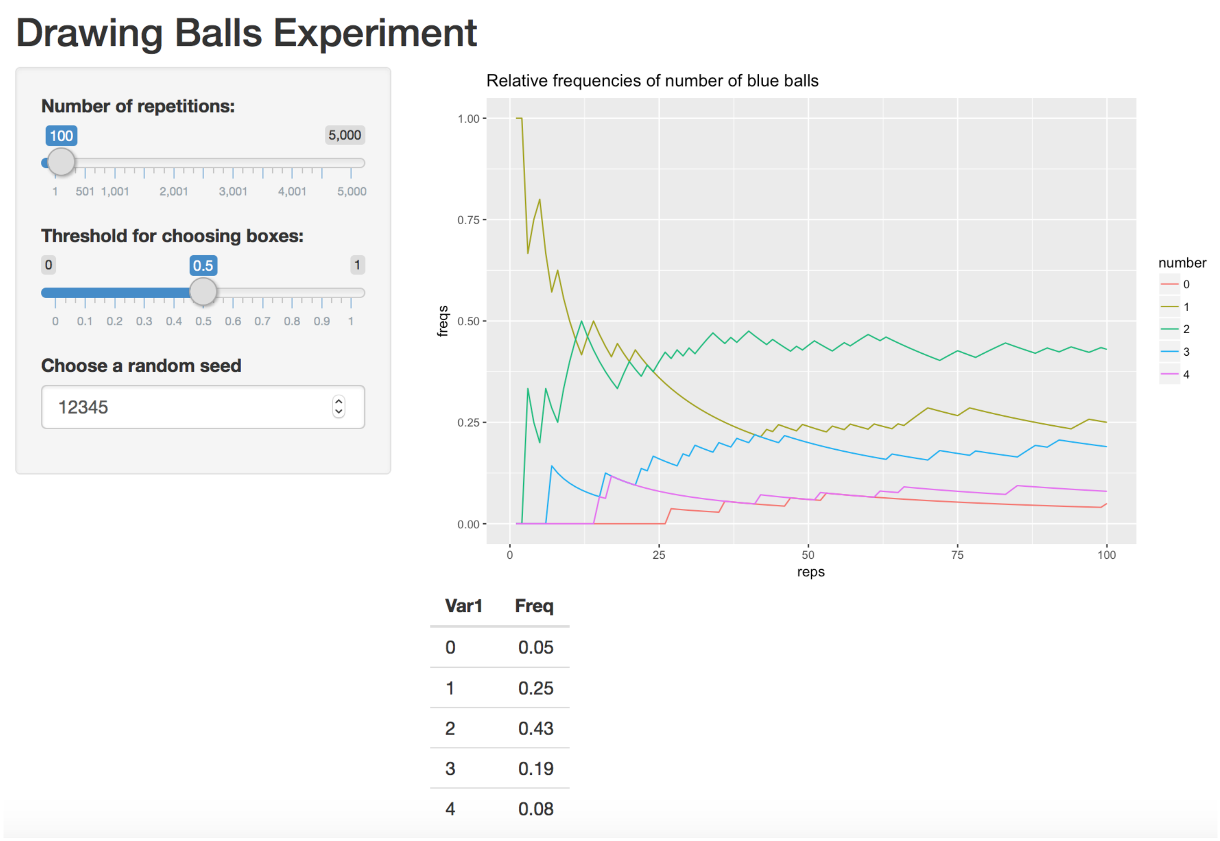


See how the distributions are centered around 35 (our theoretical sum)? Our Monte Carlo simulation relied on large numbers of sampling iterations to produce this amazing result!

Discussion:

You can play around with a Monte Carlo Simulation using the Shiny App available in the github repository along with this post (it’s from Stat 133 Fall 2017 Lab 9, available to the public to view). Check it out some time. The UI output looks like this. You can see that as you increase the number of trials, the frequencies stabilize off to the expected frequencies found for each number in the table below the app.

Here’s a screenshot below for those who can’t access it right now.



Drawing Balls Experiment

Number of repetitions: 100

Threshold for choosing boxes: 0.5

Choose a random seed 12345

Var1	Freq
0	0.05
1	0.25
2	0.43
3	0.19
4	0.08

.

Take-home message:

If anyone were to take away a major message from this tutorial, it would be to internalize the importance of having a large number of repetitions conducted on a random sampling method in order to yield the average of a certain desired statistic. For example, if we toss a coin 10,000,000 times, the percentage of heads that appear will converge to 50%. In essence, Monte Carlo simulations can depict accurate bounds for an uncertain process given a probability distribution by resampling randomly many many times.

Dirk P. Kroese states that “modern statistics and data science increasingly relies on computational tools such as resampling and Monte Carlo simulations in order to analyze very large and/or high dimensional data sets” (University of New South Wales). The Monte Carlo method is easy and efficient, as it relies on the strength of randomness to execute complex models in a simple fashion. It’ll be interesting to see how Monte Carlo simulations will be applied in the future, especially in the development of new technology innovcations and scientific discoveries!

References:

I used these references to construct the coding tutorial portion of this assignment as well as to paraphrase the conceptual definitions of Monte Carlo Simulations.

1. https://www.r-bloggers.com/introducing-the-montecarlo-package/
2. https://www.investopedia.com/terms/m/montecarlosimulation.asp
3. http://www.palisade.com/risk/monte_carlo_simulation.asp
4. https://web.stanford.edu/class/bios221/labs/simulation/Lab_3_simulation.html
5. https://rpubs.com/Koba/Monte-Carlo-Basic-Example
6. https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations
7. https://www.r-bloggers.com/probability-and-monte-carlo-methods/
8. https://phaethonprime.wordpress.com/2015/05/10/computational-methods-for-tabletop-games-1-zombie-dice-and-monte-carlo-simulation/
9. http://whatis.techtarget.com/definition/law-of-large-numbers
10. https://people.smp.uq.edu.au/DirkKroese/ps/whyMCM_final.pdf