# Lab3: Simulations

Matthew Varkony

1/26/2020

```
library(tidyverse)
set.seed(4200)
```

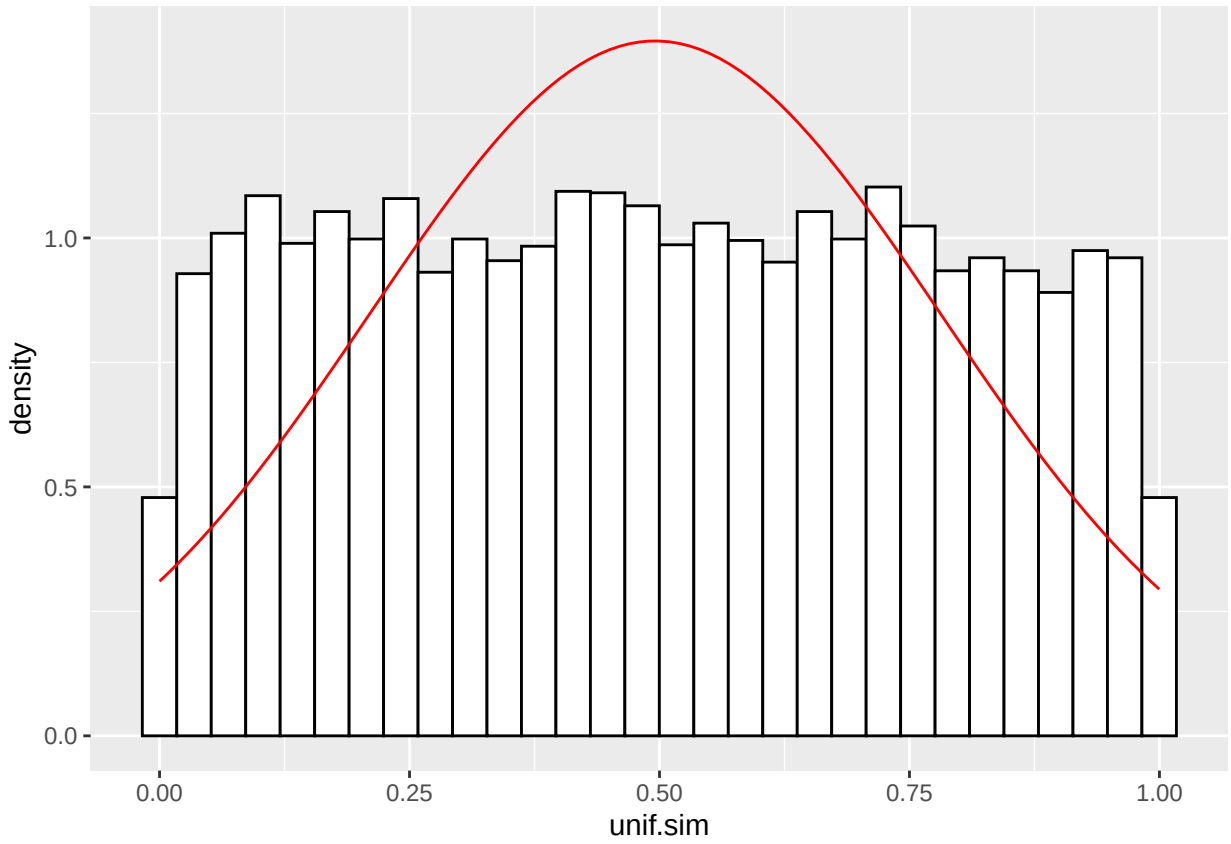
## Simulating Data

Simulating data is important for a number of reasons. Most importantly, simulations allows us the ability to test assumptions of models used in our analysis. R enables simulations by providing functions that generate random numbers based on certain restrictions we impose on the models we are working with. We will start out by simulating using the Uniform distribution, and build upon this to show how in reality all other simulations and distributions can be derived from this specific uniform call.

##Uniform Distribution

The uniform distribution applies an equal weighted probability to each outcome within the distributions outcome space. The distribution is typically bounded by an upper and lower limit. In the following example I chose a range of [0,1] and overlayed a normal distribution on the graph. We can see that for 10,000 draws from the uniform distribution are relatively evenly divided over the set.

```
unif.sim = runif(10000, min = 0, max = 1)
ggplot() +
  aes(unif.sim) +
  geom_histogram(aes(y = ..density..),
                 bins= 30,
                 color = "black", fill = "white") +
  stat_function(fun = dnorm, color = "red" , args =
                 list(mean = mean(unif.sim),
                      sd = sd(unif.sim)))
```




Now that we have simulated some data and created a graph, let's see how the uniform distribution can be coerced to display the properties of the binomial distribution. This will also allow us to implement a for loop that will demonstrate how the law of large numbers is applied to a coin flipping process described by the Bernoulli distribution.

**Example: Coin Flip Bernoulli**

Bernoulli distribution is used for discrete variables that have a set dependent on two outcomes. The most common example of a Bernoulli distribution is the flipping of a coin. We can describe the Bernoulli distribution with a uniform distribution in the following way...

Let X be our Bernoulli random variable with probability of heads $p$.

$$X = \begin{cases} 1 & U < p \\ 0 & U \geq p \end{cases}$$

So the probability of obtaining a heads can be defined as

$$P(H) = P(X = 1) = P(U < p) = p$$

Now lets assume that this is a fair head flip, so we can assign $p = 0.5$. Then the code for this would look as follows.

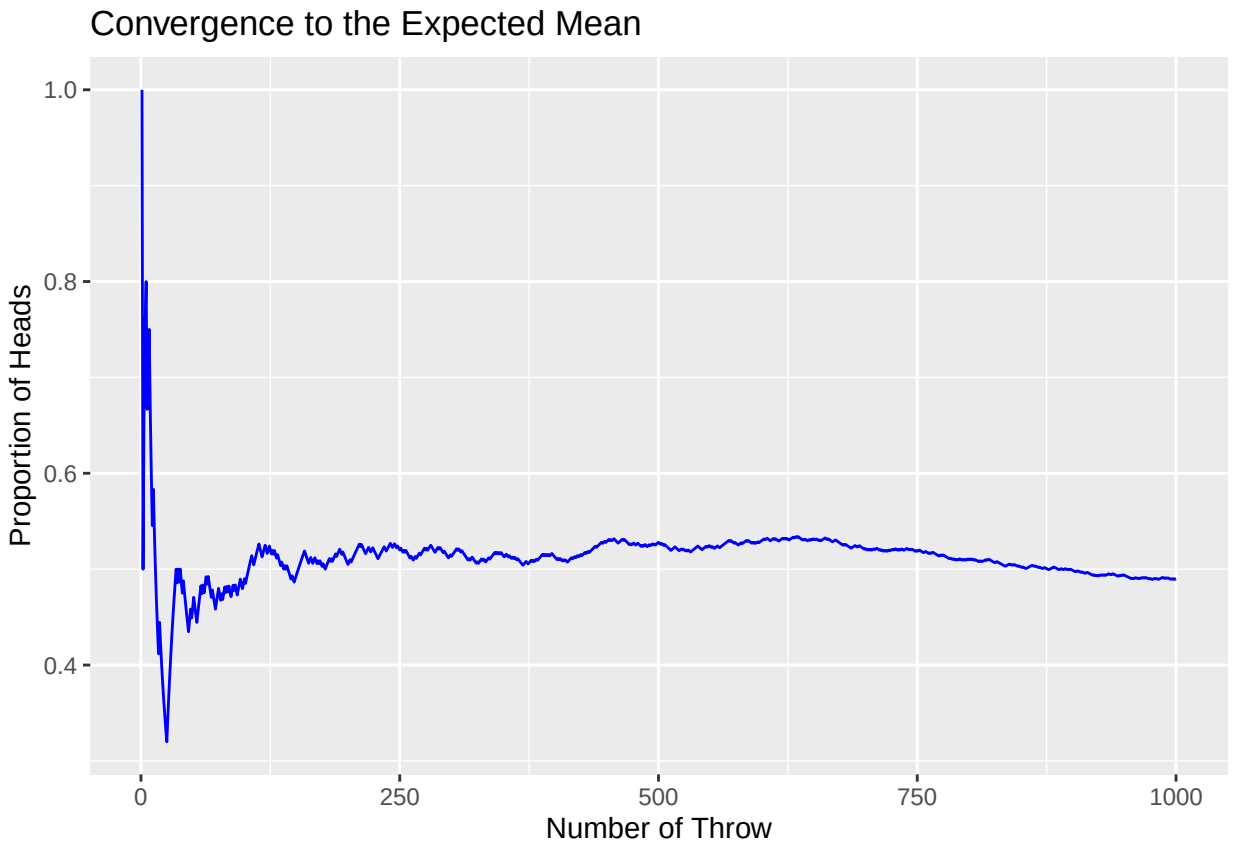
```
U = runif(n = 1, min = 0, max = 1)
U
```

```
## [1] 0.568409
```

Now, that we have demonstrated how to generate a Bernoulli distribution using a uniform simulator, we can implement the for loop. This will allow us to simulate a coin flip multiple times. We will also get a chance to visualize the law of large numbers.

Typically when we run simulations they are not for to look at different distributions. Therefore we have a number of other variables to keep track of in our for loop, and we often want to change the number of samples that we create. This is why it is good practice to generate a variable outside of the loop environment that represents how many samples you want to pull from a distribution.

```
n = 1000
Uni = runif(n, min = 0, max = 1)
toss = Uni < 0.5
a = vector("numeric", n+1)
avg = vector("numeric", n)
for(i in 2:length(a)){
  a[i] = a[i-1] + toss[i-1]
  avg[i-1] = a[i]/(i-1)
}
ggplot() +
  geom_line(aes(x = 1:n, y = avg), col = "blue") +
  labs(x = "Number of Throw", y = "Proportion of Heads", title = "Convergence to the Expected Mean")
```

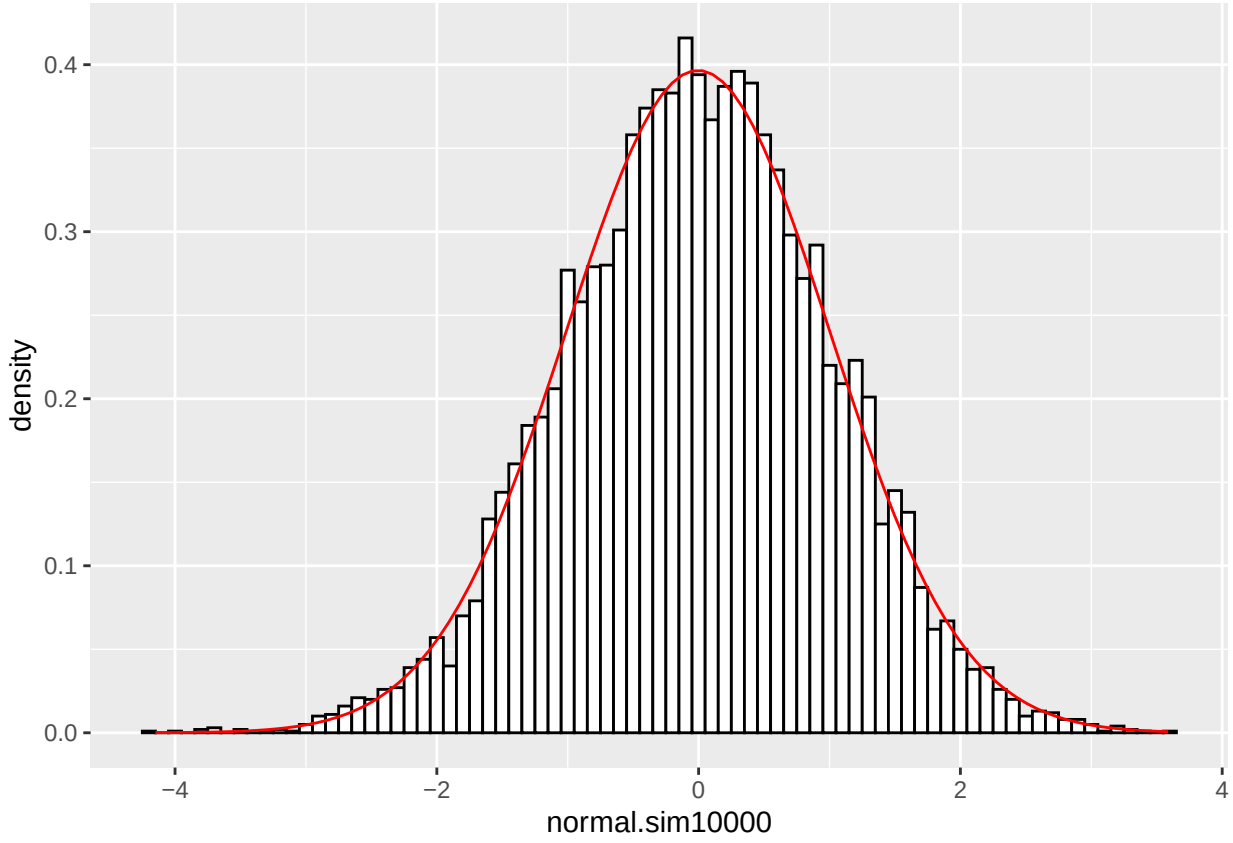



Here we are setting our number of samples to `n = 1000`. The `Uni` variable represents the results form the simulated Uniform Simulation. In this case you will see the Uni will have 1000 values [0,1]. The toss variable will return a `logical` string of `TRUE` and `FALSE` depending on the values from the simulated data. This command `Uni < 0.5` checks if each of the values simulated is less than 0.5 and than assigns the appropriate logical values.

## Normal Distribution

The most well known distribution throughout all fields is the **normal distribution**. The normal distribution has important properties that enable inference in statistical analysis. Based on proofs from the central limit theorem, it can be shown that the summation of standardized independent random variables, converge towards the standard normal distribution. The significance of this result resides in the fact that random variables generated from non-normal distributions can be aggregated and analyzed under assumptions inherent to the normal distribution.

```
normal.sim10000 = rnorm(10000, mean = 0, sd = 1)
ggplot() +
  aes(x=normal.sim10000)+
  geom_histogram(aes(y = ..density..),
                 binwidth = .1,
                 color = "black", fill = "white") +
    stat_function(fun = dnorm, color = "red",
                  args = list(mean = mean(normal.sim10000), sd = sd(normal.sim10000)))
```




We can see that the shape of this histogram looks similar to that of a normal distribution that is created using the mean and standard deviation of our sampled points.

### Note

`R` can simulate many other distributions as well. You just need to look them up online to obtain the wording for these functions. In general, the following prefixes are applied to these function calls for different reasons.

- `d` evalutes the probability density function (ie. `dnorm`)
- `r` draws random variables from the chosen distribution (ie. `rnorm`)

- `p` evaluates the cumulative distribution function for the chosen distribution (ie. `pnorm`)
- `q` evalues the inverse cdf of the chosen distribution (ie. `qnorm`)

## Monte Carlo Simulations

Let's pretend that we are throwing two six sided die. We want to know what the probability that their sum is less than or equal to 5? We can incorporate iterations (loops or apply function) and also function creation to solve this problem. So let's try that.

First we create our function.

```
mc.func = function(numDice, numSides, targetValue, numTrials){ ##variables
  apply(matrix(sample(1:numSides, numDice*numTrials, replace = TRUE), nrow = numDice), 2, sum) <= targe
}
```

```
outcomes = mc.func(2,6,5,5)
mean(outcomes)
```

```
## [1] 0
```

Obviously this is not close to the theoretical value, since we know that there are a number of combinations that satisfy the sum of the two die values being less than 5. So we can run the simulation again.

```
outcomes2 = mc.func(2,6,5,10000)
mean(outcomes2)
```

```
## [1] 0.2695
```

## Exercise

Now let's try an application of the methods we have learned in the past few weeks and apply it to problems similar to your workout.

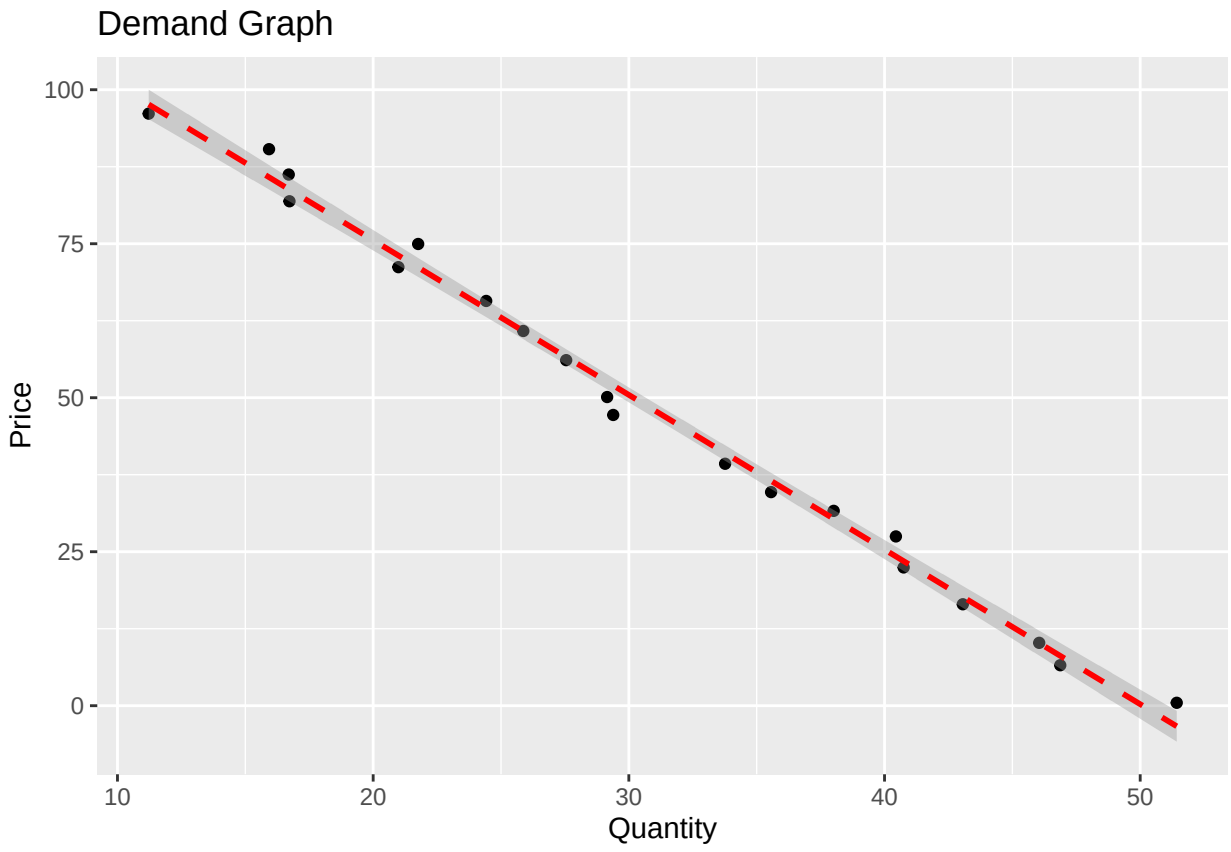
1. Generate a demand curve with an error around the price and final quantity. Graph the resulting demand curve using the following demand curve and label the graph.

$$Q = -0.4P + 50$$

2. Next randomly sample a price from the generated curve, determine the quantity demanded for each price and label this points on the graph.

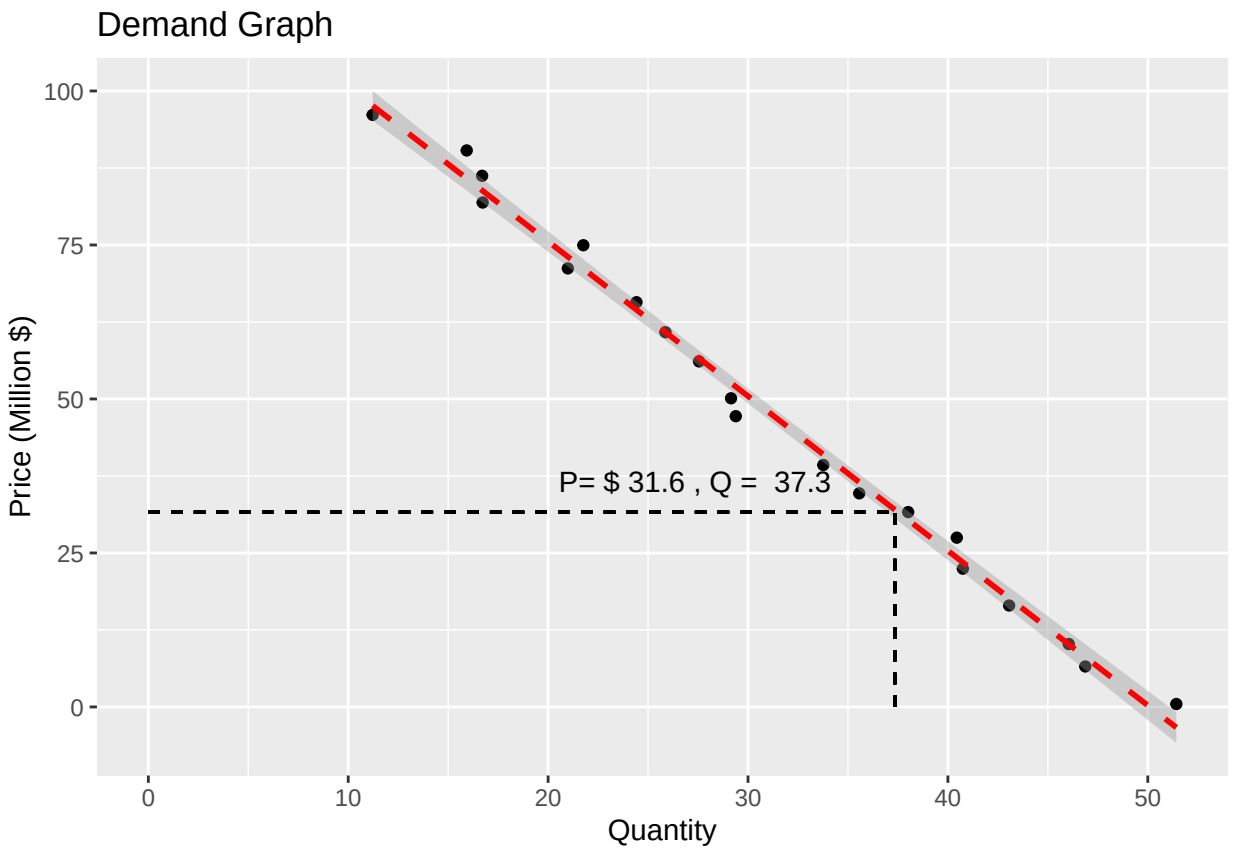
```
prices = seq(1,100, 5)
p_n = length(prices)
er.prices = rnorm(p_n, mean = prices, sd = 1)

##function
dem.fun = function(slope, intercept, price){
  q = slope*price + intercept
  return(q)
}

##quantity
quan = rnorm(p_n, dem.fun(-.4, 50, er.prices), 1)

##demand
dem = cbind(er.prices, quan) %>%
```

```
  data.frame()

##plot
dem.curve =
  ggplot(dem) +
  geom_point(aes(x = quan, y = er.prices)) +
  stat_smooth(method = 'lm', aes(x= quan, y = er.prices), col = "red", lty = 2) +
  labs(x = "Quantity", y = "Price", title = "Demand Graph")
dem.curve
```




```
##sample
samples = sample(er.prices, 1)

# Create a function to determine the quantity demanded for each price,
# and label the points on individual graphs
graph.samp.fun = function(prices.samp)
{
  dem.curve + # the original demand curve
    geom_linerange(aes(x = dem.fun(-.4, 50, prices.samp),
                       y= NULL, ymin=0, ymax=prices.samp),
                   linetype="dashed")+ # add a vertical line from zero to the quantity
    geom_segment(aes(x=0, xend= dem.fun(-.4, 50, prices.samp),
                     y=prices.samp, yend=prices.samp),
                 linetype="dashed" )+ # add a horizontal line from zero to the price
    annotate(geom="text", x=dem.fun(-.4, 50, prices.samp)-10,
             y=prices.samp + 5,
             label=paste("P= $",signif(prices.samp,3),", Q = ", signif(dem.fun(-.4, 50, prices.samp),3)
```

```
            size = 4 , color="black") + # label the point on the graph, use paste to make a nice label
    xlab("Quantity") + # add an accurate x label
    ylab("Price (Million $)") # add an accurate y label
}

graph.samp.fun(samples[1])
```




## Bibliography

H. Pishro-Nik, "Introduction to probability, statistics, and random processes", available at https://www.probabilitycourse.com, Kappa Research LLC, 2014.