Simulating Discrete Distributions in R

Warren

Dec. 3, 2017

setup for ggplot

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.2
```

Introduction

In lecture, we learned basics about random numbers and used that to simulate tossing coins. With those skills in hand, we can now simulate four famous discrete distributions. If you have taken probability courses such as LD stats, stat 134, Math 55, CS 70, you will find those distributions familiar. If you don't, don't worry! I will explain them in details.

About random numbers

As discussed in class, we can set seeds in R to "force" outcomes for a random event. This is because R, like most programs nowadays, uses pseudo-random. As its name suggests, this kind of random looks like random, but is actually not random. For example, in R, the default pseudo-random number generator is called Mersenne Twister. How it is run is not important for us, but the result is a seemingly random (long) list of (32-bit) integers. This list of integers is what we call "seeds". When we need to do something random in R, we pick a number from this list as our current "seed" and use it to calculate the outcome of the random event. This is why we can use the set.seed() function to force outcomes: it bypasses the "pick number from list" part and set the seed directly.

In this post, I will set seeds for some functions to make the results reproduceable. However, feel free to delete the set.seed() and rerun the functions if you want to get a different outcome!

Useful functions

Toss coins

Let's use the toss coin function from class. The following code is directly copied and pasted from the tutorial (link in reference).

```
# x: coin object (a vector)
# times: number of tosses
toss <- function(x, times = 1) {
  sample(x, size = times, replace = TRUE)
}
```

However, instead of using the c("head", "tail") coin, let's use the c(1, 0) coin instead. This is because we will be counting number of heads in the distributions, so the c(1, 0) coin will ease our computation (head means +1, tail means +0).

```
# Our coin
coin = c(1, 0)
```

Poker hands

We will also simulate drawing cards from a standard deck.(That is, 52 cards, 13 of each(4) kind, without the 2 jokers).

```
#x: a standard deck of cards
#number: number of cards to draw
draw = function(x, number = 5) {
  sample(x, size = number, replace = FALSE)
}
```

And, we will only be interested in whether a card is Ace or not. So our deck of card should be abstracted and it looks like this:

```
#48 non-aces
nonA = rep(0,48)
#4 aces
A = rep(1, 4)
#put them together
deck = c(nonA, A)
```

1. Binomial distribution

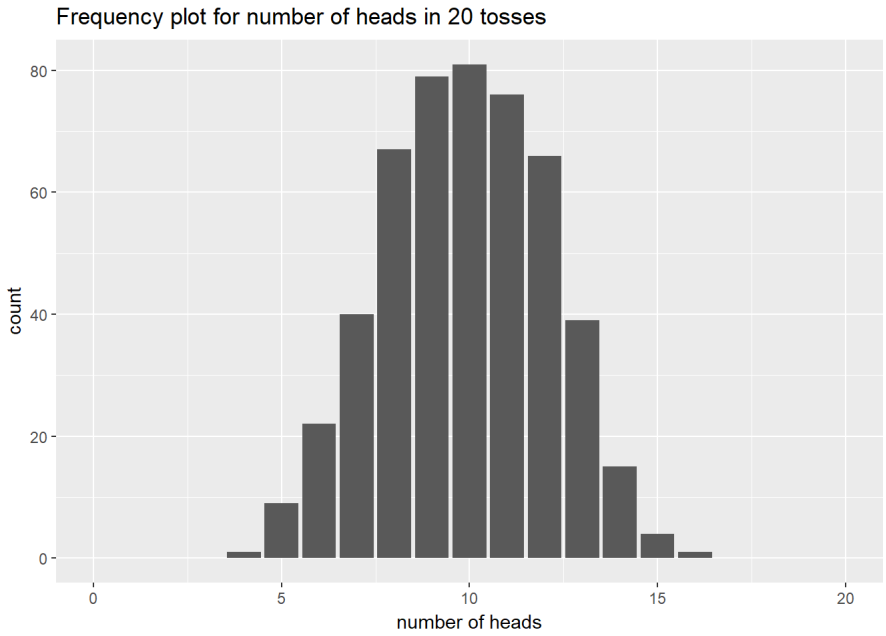
Q: Imagine tossing a fair coin n (a fixed number of) times. How many heads will we get?
Say n = 20. Let's do a simulation!

```
set.seed(1)
sum(toss(coin, 20))
```

```
## [1] 9
```

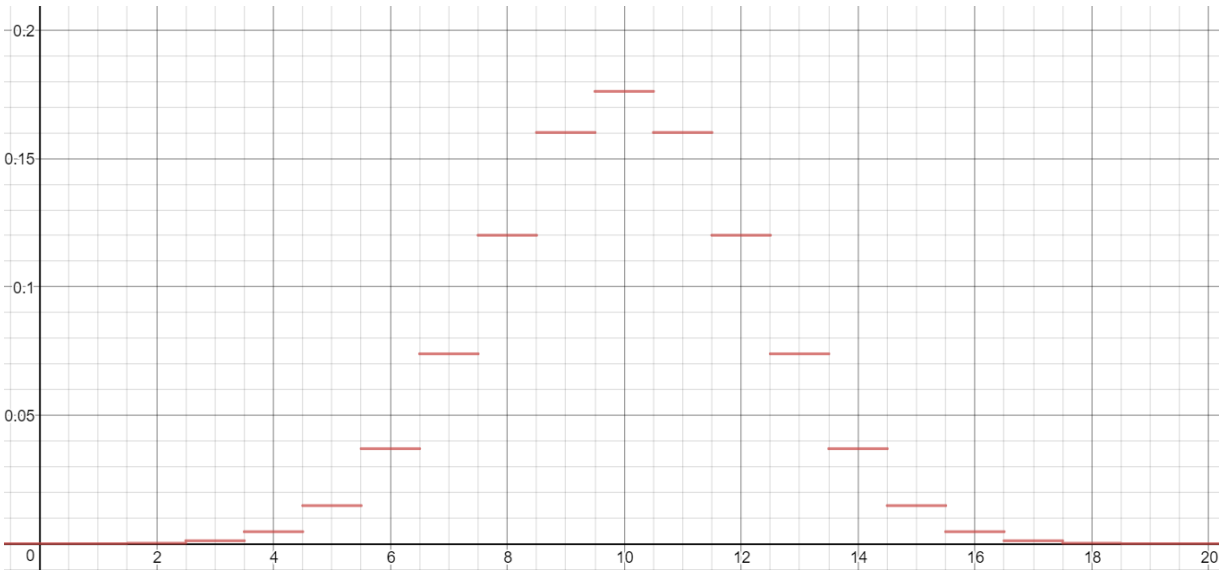
OK, 9. Hmmm… This number is not useful thought, right? Because if we do the same thing again (with another seed), we might get another result. So let's repeat this for some times and see what we get.

```
#initialize a list of number of heads.
num_heads = c()
#repeat the event (toss 100 coins) 500 times
set.seed(2)
for(i in 1:500){
  num_heads = c(num_heads, sum(toss(coin, 20)))
}
#data frame for frequency of heads
heads_freq = data.frame(num_heads)
#Visulizing the result
ggplot(heads_freq, aes(x = num_heads)) + geom_bar() +
  xlab("number of heads") + xlim(0, 20) +
  ggtitle("Frequency plot for number of heads in 20 tosses")
```



From our probability courses, or according to almighty Google, this kind of distribution (counting the number of successes in n independent trials with probability p) is called binomial. The probability of getting k successes is:
{{n}\choose{k}}p^kq^{n-k} In our example, the formular becomes:
{20 \choose {x}}0.5^x0.5^{20-x} = {20\choose {x}}0.5^{20} Here is a graph for this function:

Graph for the probability function binomial(n = 20, p = 0.5)



binom(20, 0.5)

Comparing to our barplot, the shape of them looks similar. (They should not look exactly the same, because of chance variations.) So we have verified that our coin-tossing event do follow the binomial distribution!

2. Geometric distributions

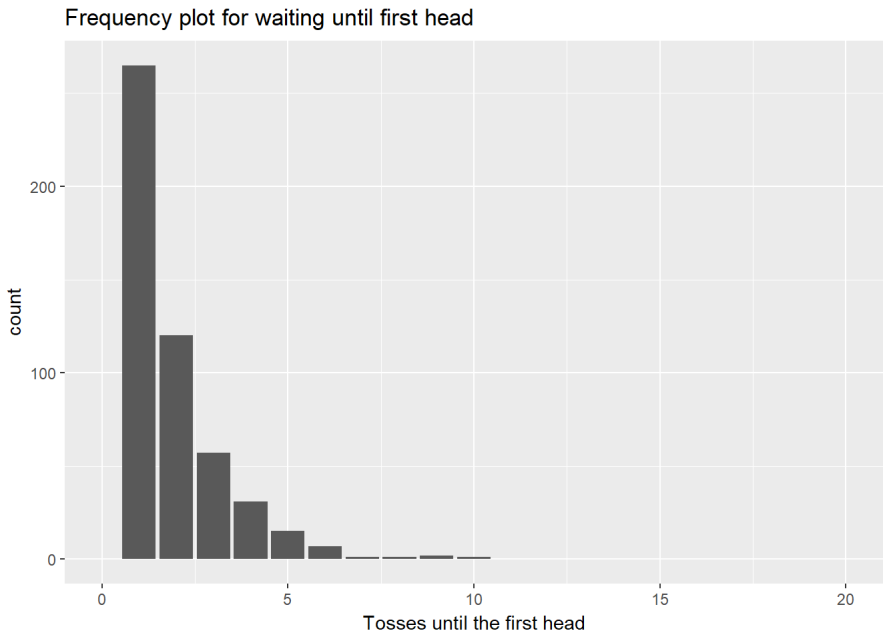
We are still interested in coin tosses. Now, We want to know when we will see the first success. So we keep tossing coins until we see the first head. Simulation time!

```
#initialize counter. We need to toss at least 1 time, so we start with 1
times_geom = 1
set.seed(3)
#toss until we see a head
while(toss(coin)!=1) times_geom = times_geom + 1
times_geom
```

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

So with this seed (3) we get a head on the very first toss. Again, one result is not representitive. Let's repeat it!

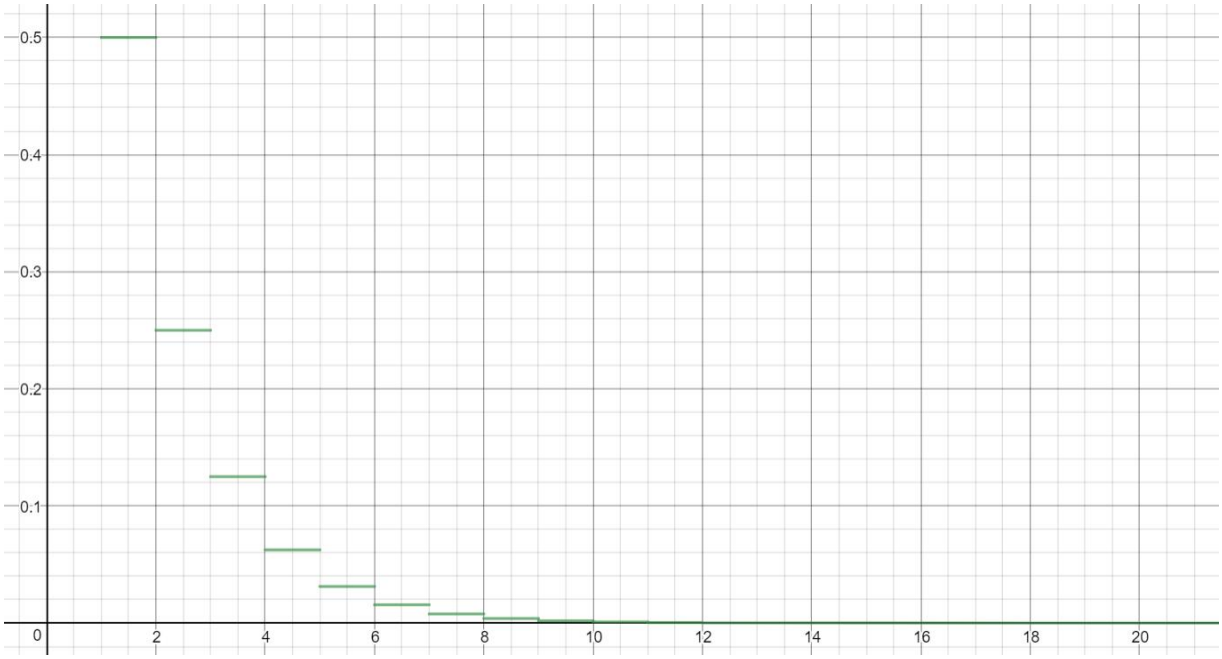
```
#initialize a list of waiting time (aka, time until first head).
waittime_geom = c()
#repeat the event (wait for first head) 500 times
set.seed(4)
for(i in 1:500){
  times_geom = 1
  while(toss(coin)!=1) times_geom = times_geom + 1
  waittime_geom = c(waittime_geom, times_geom)
}
#data frame for waiting time
wt_geom = data.frame(waittime_geom)
#Visulizing the result
ggplot(heads_freq, aes(x = wt_geom)) + geom_bar() +
  xlab("Tosses until the first head") + xlim(0, 20) +
  ggtitle("Frequency plot for waiting until first head")
```



Again, from stats or Google, we have our probability function for this kind of distribution - geometric. Probability for getting the first success on the kth trial (each trial is independent and has probability p) is:
(1-p)^{k-1}p In our example, p is 0.5, so we get:
0.5^{k-1}0.5 = 0.5^k

and graphically:

Graph for the probability function geom(p = 0.5)



geom(0.5)

So our results look correct!

3. Negative Binomial distribution

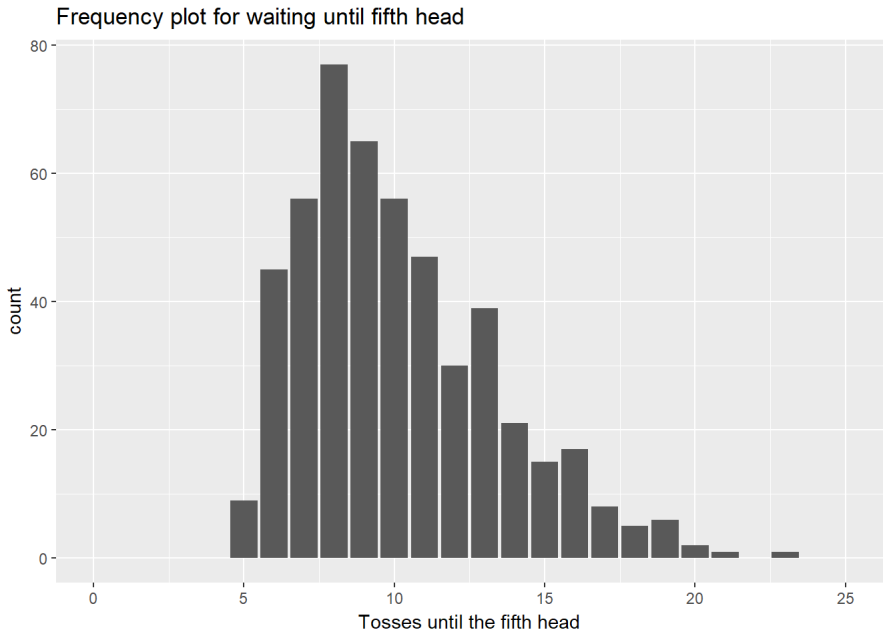
In part 2 we only waited for one success. What if we want more? Say, how many times do we need to toss a coin until we see the 5th head? Let's still do one run first.

```
#initialize trials counter.
#This time we start with 0, because now we have a success counter so need to +1 every trial
times_nb = 0
#initialize success counter
successes_nb = 0
set.seed(5)
#toss until we see a head
while(successes_nb!=5){
  if(toss(coin)==1) successes_nb = successes_nb + 1
  times_nb = times_nb + 1
}
times_nb
```

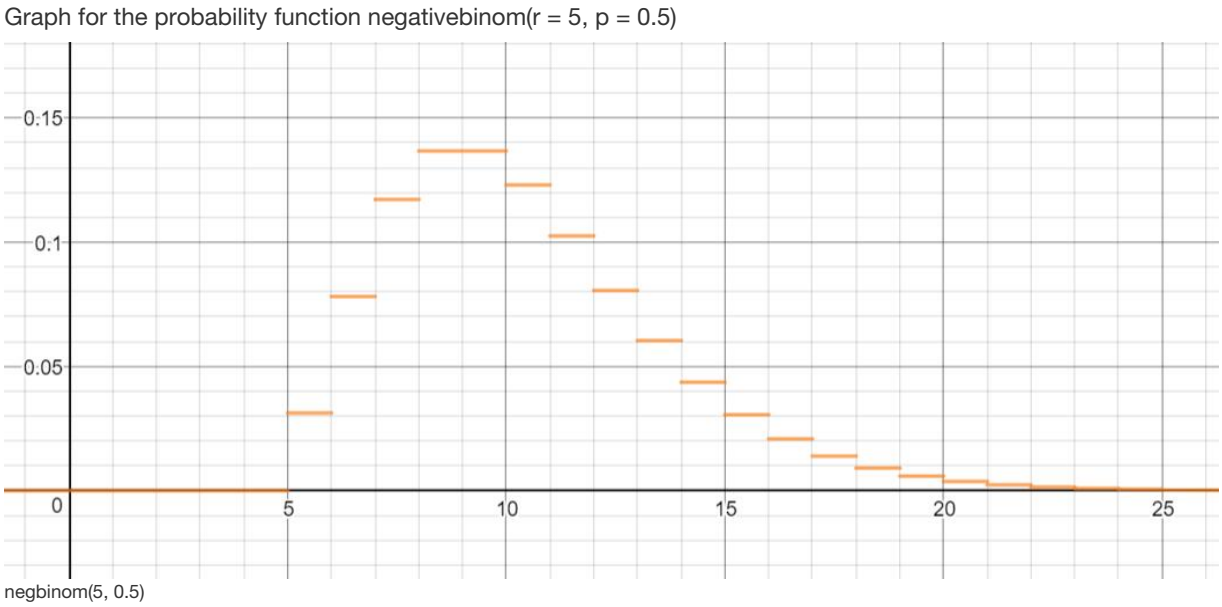
```
## [1] 11
```

So we tossed the coin 11 times until we see the 5th head. Now its repetition time!

```
#initialize a list of waiting time (aka, time until fifth head).
waittime_nb = c()
#repeat the event (wait for fifth head) 500 times
set.seed(6)
for(i in 1:500){
  #initialize trials counter
  times_nb = 0
  #initialize successes counter
  successes_nb = 0
  #toss until we see a head
  while(successes_nb!=5){
    if(toss(coin)==1) successes_nb = successes_nb + 1
    times_nb = times_nb + 1
  }
  waittime_nb = c(waittime_nb, times_nb)
}
#data frame for waiting time
wt_nb = data.frame(waittime_nb)
#Visulizing the result
ggplot(heads_freq, aes(x = wt_nb)) + geom_bar() +
  xlab("Tosses until the fifth head") + xlim(0, 25) +
  ggtitle("Frequency plot for waiting until fifth head")
```



Formula time! This kind of distribution is called negative binomial, defined as: getting the rth success on the kth trial (each trial is independent and has probability p). Probability of it is: {{k-1} \choose {r-1}}p^{r-1}(1-p)^{k-r}p In our example: {{x-1} \choose 4}0.5^{4}0.5^{x-5}0.5 = {{x-1} \choose 4}0.5^{x} and the graph is:



negbinom(5, 0.5)

So we have done another verification!

4. ??? distribution

OK I know a lot of you have taken stat courses and have taken google as a joke. And you probably expect me to put Hypergeometric here. But no, the point of this post is not to prove you that some event indeed follow a distribution. The point is that when we do not know a certain distribution, we can use simulations to try to approximate it!

So let's consider this question: you continue dealing cards from a deck until you draw two Aces, of any kind. How many cards will you draw? Notice that this question is different from the previous (negative binomial) one. This time, the probability changes as you deal cards: first one is 4/52, second 4/51, third 4/50 …… So this is not any distribution you know!

Also notice that this question is equivalent to: deal every card from the 52 card deck. Count from the first card until the second Ace. This will make our simulation easier.

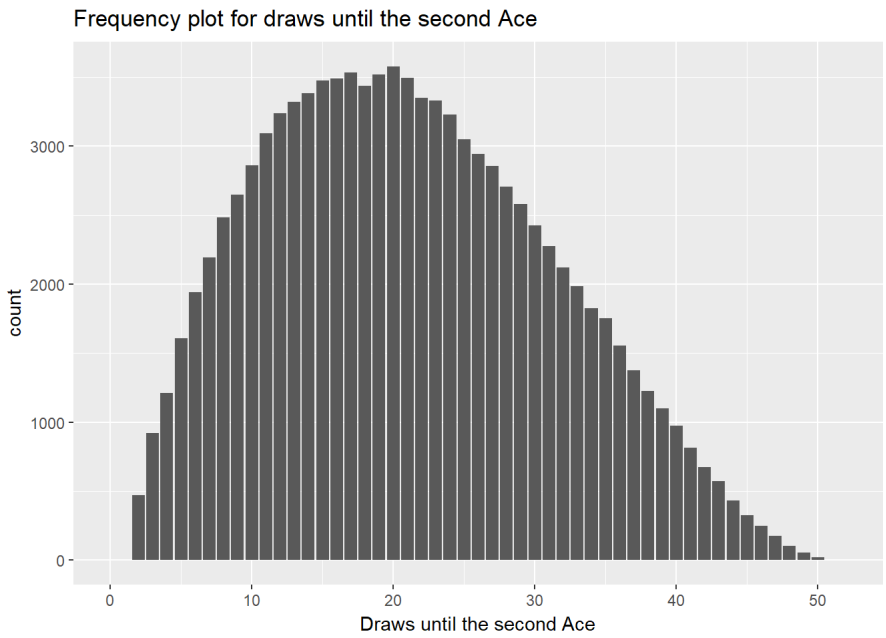
Stuck on probability theories? Let's do a simulation!

```
set.seed(7)
#draw all the cards from the deck
drew_cards = draw(deck, 52)
#initialize counter
counter = 0
#initialize success counter
successes_nh = 0
for(i in drew_cards){
  counter = counter + 1
  if(i == 1) successes_nh = successes_nh + 1
  #stop when we've got 2
  if(successes_nh == 2) break
}
counter
```

```
## [1] 14
```

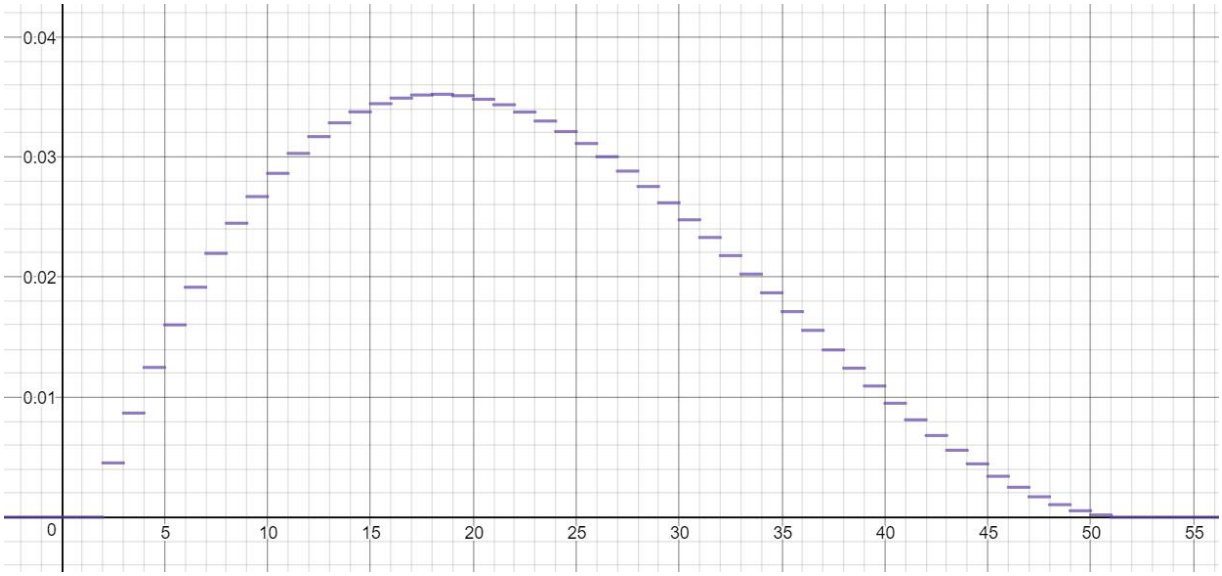
So we got the second ace on the 14th draw.
You know what we need to do now: repeat it! As our goal is to approximate probability, we will follow a rule in statistics (or more generally, in science): in order for your results to be more accurate, do the same thing over and over (and over and over and over and over…) again!

```
#initialize a list of draw number.
draws_nh = c()
#repeat the event (wait for first Ace) 100000 times
#takes a while
set.seed(8)
for(i in 1:100000){
  #draw all the cards from the deck
  drew_cards = draw(deck, 52)
  #initialize counter
  counter = 0
  #initialize success counter
  successes_nh = 0
  for(i in drew_cards){
    counter = counter + 1
    if(i == 1) successes_nh = successes_nh + 1
    #stop when we've got 2
    if(successes_nh == 2) break
  }
  draws_nh = c(draws_nh, counter)
}
#data frame for waiting time
drawnum_nh = data.frame(draws_nh)
#Visulizing the result
ggplot(heads_freq, aes(x = drawnum_nh)) + geom_bar() +
  xlab("Draws until the second Ace") + xlim(0, 52) +
  ggtitle("Frequency plot for draws until the second Ace")
```



If you change this frequency plot to a porpotion plot, you get an approximate probability curve!
Although you did not learn it, this is also a named distribution, called negative hypergeometric. I will just provide the actual probability function graph with our parameters to convince you that we are correct.

Graph for the probability function NHG(N = 52, K = 48, r = 2)



nhg(52, 48, 2)

Very, very close, right? So maybe we can just use the upper graph if we just need an approximate probability!

Summary

So we have verified for the three coin-tossing examples that they indeed follow the famous distribution models. From there, we also tried to approximate an unknown distribution by running the simulation 100,000 times. Hence we have our take home message below:

Take home message

If you ever encounter a distribution (known or unknown), you can use R to simulate it. In fact, if you run the simulation a huge number of times, you will be able to get a (pretty accurate) approximation for the probability density function.

Reference:

1. https://stat.ethz.ch/R-manual/R-devel/library/base/html/Random.html
2. https://en.wikipedia.org/wiki/Mersenne_Twister
3. https://github.com/ucb-stat133/stat133-fall-2017/blob/master/tutorials/10-intro-to-random-numbers.Rmd
4. https://www.stat.berkeley.edu/~stark/SticiGui/Text/montyHallTest.htm
5. https://www.stat.berkeley.edu/~stark/SticiGui/Text/randomVariables.htm
6. https://en.wikipedia.org/wiki/Binomial_distribution
7. https://en.wikipedia.org/wiki/Geometric_distribution
8. https://en.wikipedia.org/wiki/Negative_binomial_distribution
9. https://en.wikipedia.org/wiki/Negative_hypergeometric_distribution
10. All the probability graphs are drawn on: https://www.desmos.com/calculator