Law of large numbers, Poisson distribution, Monte Carlo simulation

STA 032: Gateway to data science Lecture 16

Jingwei Xiong

May 8, 2023

Today

- Common probability distributions
    - Binomial
        - Sampling and law of large numbers; effect of changing parameters
    - Poisson
    - Monte Carlo Simulation

Sampling from a binomial distribution in R

- Recall draws from the Bernoulli distribution:
  - For any value of $0 \leq p \leq 1$ (including .5), we can use the `rbinom()` function
  - `rbinom()` has the arguments `n, size, prob`, where `n` is the number of draws, and `prob` is the probability of success.
  - `size` is the "number of trials (zero or more)" -- for the Bernoulli distribution, we use `size = 1`.

Sampling from a binomial distribution in R

- For the binomial distribution, `size` is the number of Bernoulli trials; in the e-cigarette example, `size = 3`
- **NOTE**: we use n to denote the number of Bernoulli trials; in R this is the `size` argument

Description	R
Number of draws/samples from Binomial	`n`
Number of Bernoulli trials, size n	`size`
Probability of success, p	`prob`

- It should now make sense that for draws from the Bernoulli distribution, we use the argument `size = 1`.

Sampling from a binomial distribution in R

Example: 100 draws from the binomial distribution; each draw is made of 3 Bernoulli trials (size 3); probability of success = .2

```r
set.seed(0) # so results are reproducible
inputP <- .2
binomDraws <- rbinom(n = 100, size = 3, prob = inputP)
binomDraws
```

```
  [1] 2 0 0 1 2 0 2 2 1 1 0 0 0 1 0 1 0 1 2 0 1 2 0 1 0 0 0 0 0 1 0 0 1 0 0 1
 [38] 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0 2 0 0 0 1 0 0 1 0 1 0 1
 [75] 0 0 1 1 0 1 2 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0 1
```

Sample mean

- Recall: for a binomial random variable with number of trials n and probability of success p, $E(X) = np$ and $Var(X) = n(p)(1-p)$
- In our example, $p = .2$, $n = 3$, so $E(X) = .6$ and $Var(X) = 3 * .2 * .8 = .48$
- We can calculate the sample mean from our sample of 100:

```
mean(binomDraws)
```

```
[1] 0.56
```

Using sapply in R

- sapply() is a function in R that is used to apply a function to each element of a vector or list and returns a vector or matrix of the results.

Syntax:

sapply(X, FUN, ..., simplify = TRUE)

- X: vector or list to apply the function to.
- FUN: function to apply to each element of X.
- simplify: logical indicating if the resulting output should be simplified.

```
numbers <- c(1, 2, 3, 4, 5)
squared_numbers <- sapply(numbers, function(x) x^2)
print(squared_numbers)
```

```
[1]  1  4  9 16 25
```

Law of Large Numbers

- The Law of Large Numbers is a statistical theory that states that as the number of samples increases, the mean of the samples will approach the expected value of the population.
- In simpler terms, as the sample size increases, the sample mean will get closer to the true population mean.
- Assumes that the samples are drawn randomly and independently from the population.

```r
set.seed(0) # so results are reproducible
binomDraws <- rbinom(n = 5000, size = 3, prob = .2)
myMeans <- data.frame(sampleSize = 1:5000, myMean = NA)
meanFun <- function(inputSampSize, outcomes) {
  return(mean(outcomes[1:inputSampSize]))
}
myMeans$myMean <- sapply(myMeans$sampleSize, meanFun, binomDraws)
head(myMeans,3)
```

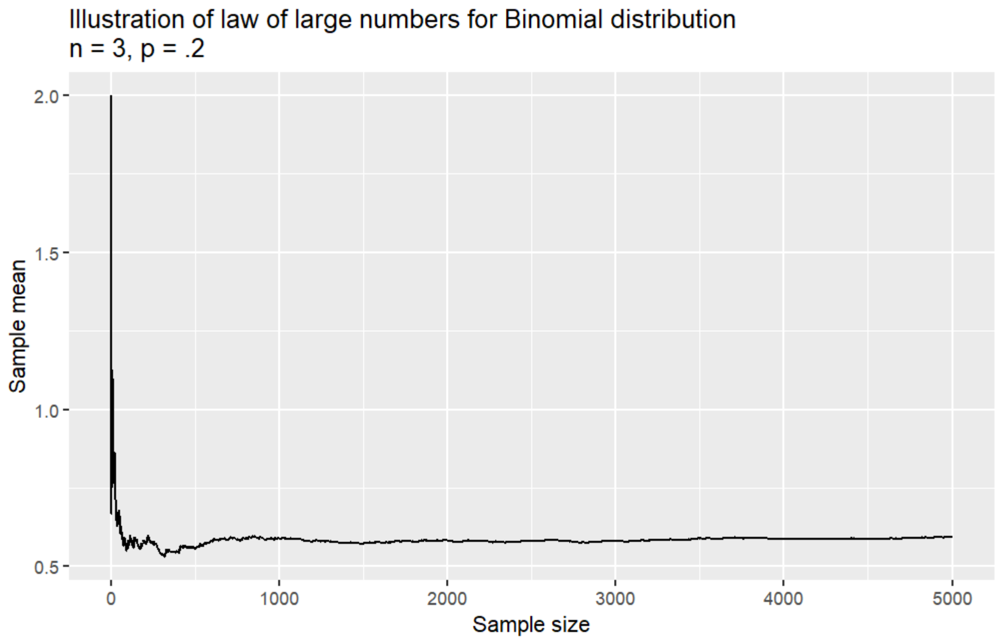
```
  sampleSize    myMean
1          1 2.0000000
2          2 1.0000000
3          3 0.6666667
```

```r
tail(myMeans,3)
```

```
     sampleSize    myMean
4998       4998 0.5948379
4999       4999 0.5947189
5000       5000 0.5946000
```

Law of Large Numbers

```
myMeans %>%
  ggplot(aes(x = sampleSize, y = myMean)) +
  geom_line() +
  labs(x = "Sample size",
       y = "Sample mean",
       title = "Illustration of law of large numbers for Binomial distributio
```



Frequency distribution of e-cigarette smokers (from sampling)

- The frequency distribution from sampling from a binomial distribution should resemble the probability distribution of the binomial distribution
- Use `rbinom()` to get 5000 draws from the population
- In R:

```r
set.seed(0) # so results are reproducible
binomDraws <- rbinom(n = 5000, size = 3, prob = .2)
table(binomDraws)/5000
```

```
binomDraws
 0      1      2      3 
0.5246 0.3638 0.1040 0.0076 
```

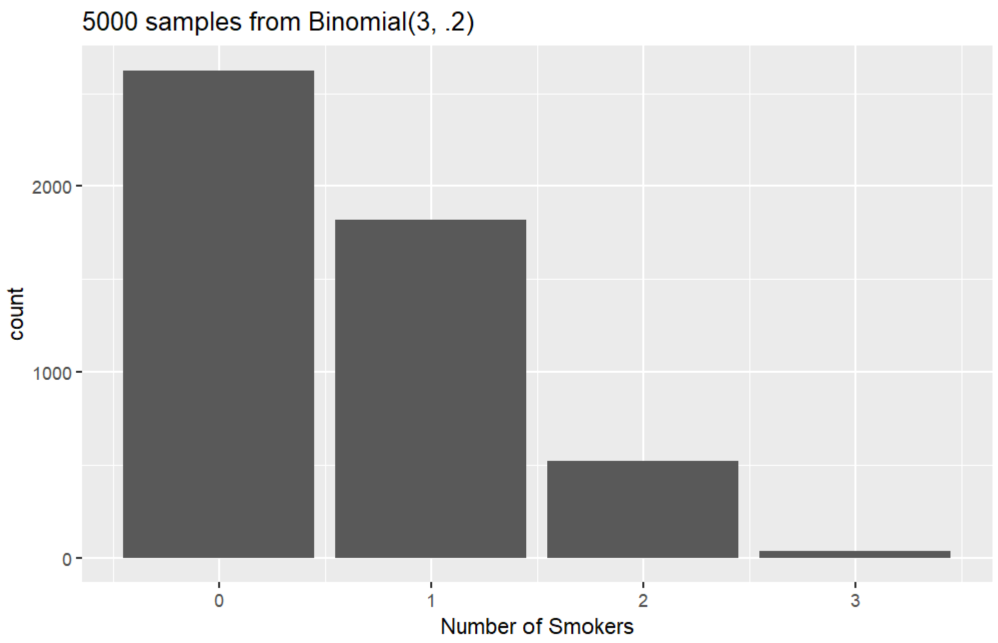
- Compare with the theoretical probabilities:

```r
dbinom(x = 0:3, size = 3, prob = .2)
```

```
[1] 0.512 0.384 0.096 0.008
```

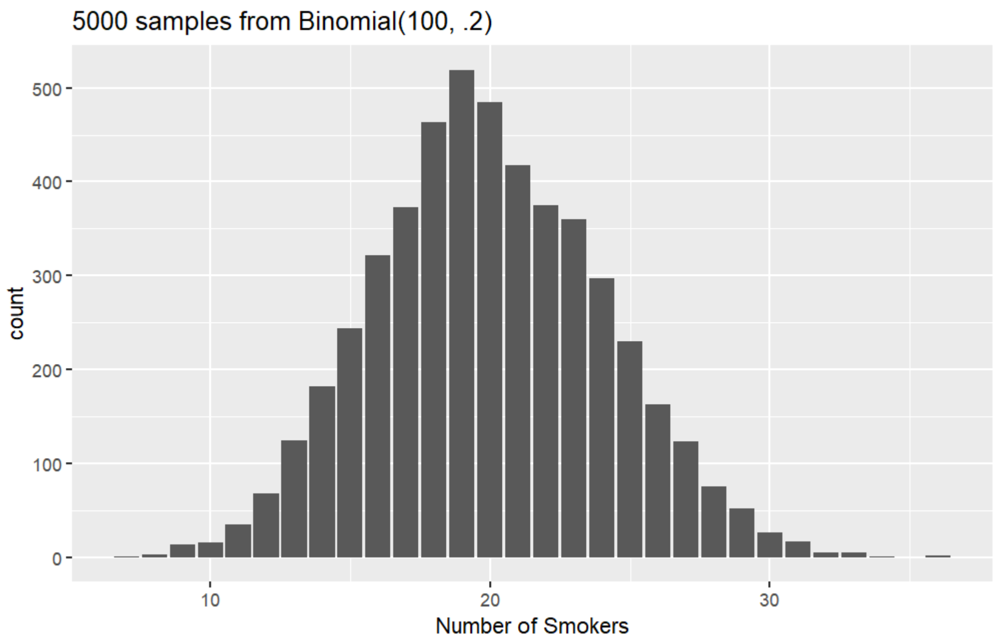
Frequency distribution of e-cigarette smokers

```
data.frame(binomDraws) %>%
  ggplot(aes(x = binomDraws)) +
    geom_bar() +
    labs(x = "Number of Smokers",
         title = "5000 samples from Binomial(3, .2)")
```



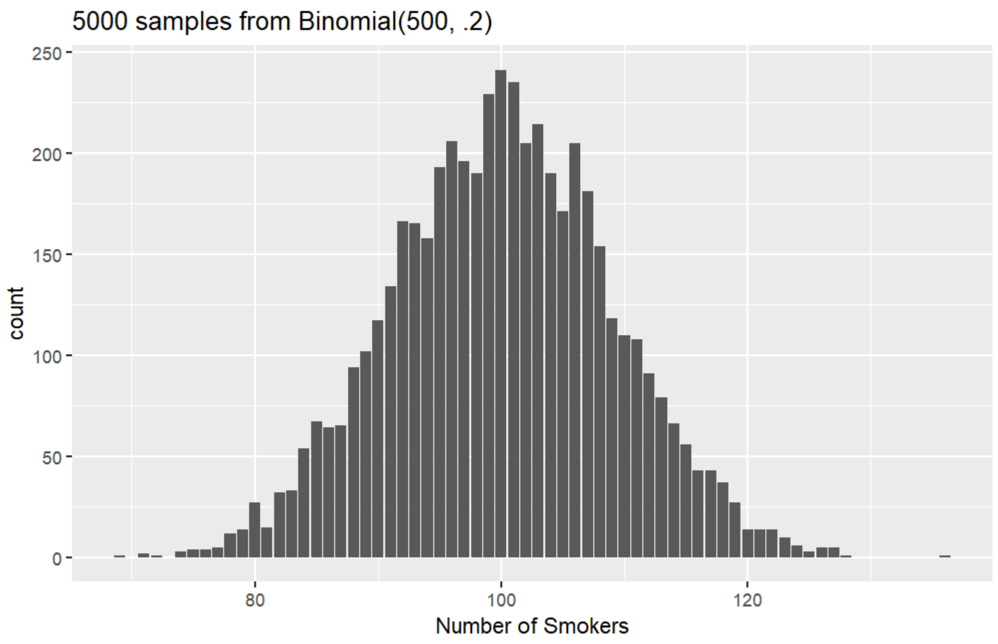
Varying the number of Bernoulli trials: 100 trials

```r
set.seed(0) # so results are reproducible
binomDraws100 <- rbinom(n = 5000, size = 100, prob = .2)
data.frame(binomDraws100) %>%
  ggplot(aes(x = binomDraws100)) +
    geom_bar() +
    labs(x = "Number of Smokers",
         title = "5000 samples from Binomial(100, .2)")
```



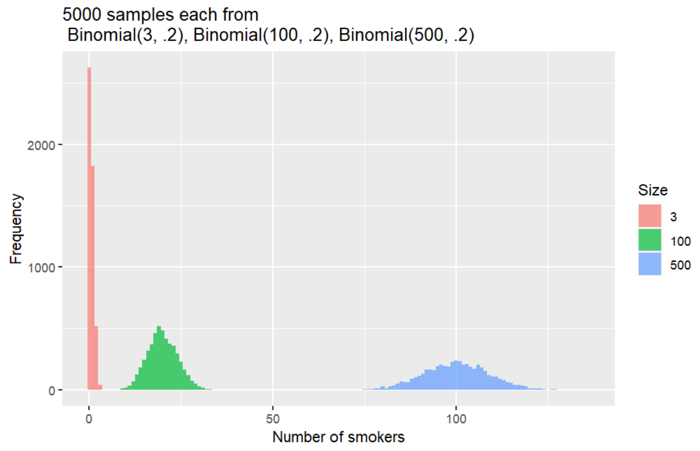
Varying the number of Bernoulli trials: 500 trials

```r
set.seed(0) # so results are reproducible
binomDraws500 <- rbinom(n = 5000, size = 500, prob = .2)
data.frame(binomDraws500) %>%
  ggplot(aes(x = binomDraws500)) +
    geom_bar() +
    labs(x = "Number of Smokers",
         title = "5000 samples from Binomial(500, .2)")
```



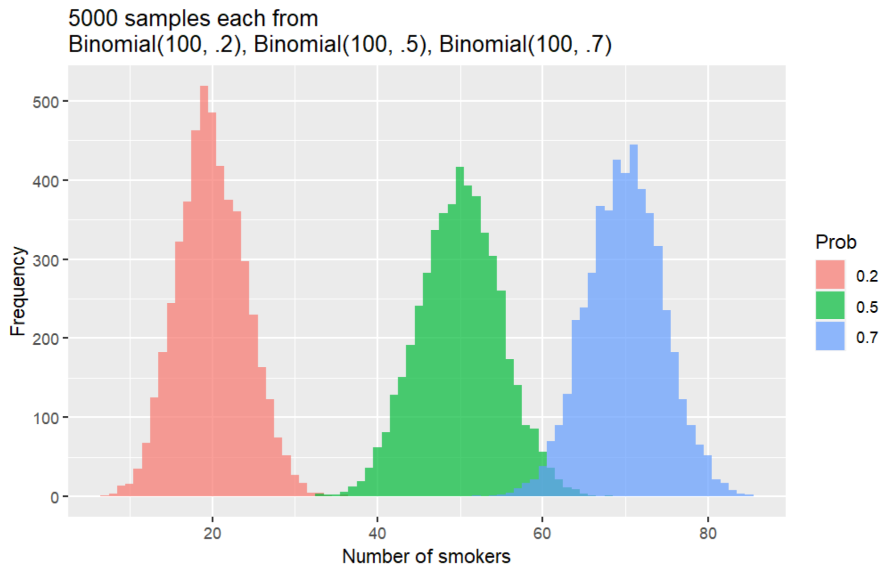
Frequency distribution of e-cigarette smokers varying number of Bernoulli trials

```r
data.frame(binomDraws) %>%
  bind_cols(size = 3) %>%
  bind_rows(
    data.frame(binomDraws100) %>%
      rename(binomDraws = binomDraws
      bind_cols(size = 100)
  ) %>%
  bind_rows(
    data.frame(binomDraws500) %>%
      rename(binomDraws = binomDraws
      bind_cols(size = 500)
  ) %>%
  ggplot(aes(x = binomDraws,
             fill = as.factor(size))
    geom_histogram(binwidth = 1, pos
    labs(
      x = "Number of smokers",
      y = "Frequency",
      title = "5000 samples each fro
      fill = "Size"
    )
```



Frequency distribution of e-cigarette smokers varying probability of success

```r
set.seed(0) # so results are reproducible
binomP.2 <- rbinom(n = 5000, size = 100, prob = .2)
binomP.5 <- rbinom(n = 5000, size = 100, prob = .5)
binomP.7 <- rbinom(n = 5000, size = 100, prob = .7)
```



Poisson distribution

- The Poisson distribution is often useful for estimating the **number of events in a large population over a unit of time**.
- For example:
    - The number of people having heart attacks in New York City every year
    - The number of accidents occurring at an intersection per hour
    - The number of typos in every 100 pages of a book
- It is named after French mathematician Siméon Denis Poisson

Poisson distribution\

- Examples: The number of people having heart attacks in New York City every year
- This distribution involves a few key ingredients
  - There must be a **fixed interval** of time or space
  - Events happen with a **known average rate**, independently of time since the last event ("memoryless" property)
    - One person having a heart attack does not change the probability of another person having a heart attack, hence the timing of the next heart attack
- The parameter that defines a Poisson distributed random variable is the **rate**, often denoted by λ, where $\lambda > 0$
- The rate is the **average number of occurrences per unit of time**
- Often used to model rare events

Probability mass function, mean and variance

- The probability mass function for a Poisson distributed random variable is $P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$, defined over non-negative integer values of x
- Recall: n! (pronounced "n factorial") is shorthand for the recursive multiplication $n! = n(n-1)(n-2)\cdots(1)$.
- The distribution has no upper limit, i.e., x can take very large non-negative integer values
- For a Poisson random variable, the mean, $E(X) = \lambda$, and the variance, $Var(X) = \lambda$

Poisson probabilities

- Like we saw before, we can use the probability mean function, $P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$, to calculate probabilities of taking a certain value
- For $x = 2$ and $\lambda = 3$, we have

$$P(X = 2) = \frac{3^2 e^{-3}}{2!} = \frac{9(e^{-3})}{2(1)} = 0.2240418$$

- In R:

```r
dpois(x = 2, lambda = 3)
```

```
[1] 0.2240418
```

- For large values of x, the probability is very small because of the large denominator

```r
dpois(x = 10, lambda = 3)
```

```
[1] 0.0008101512
```

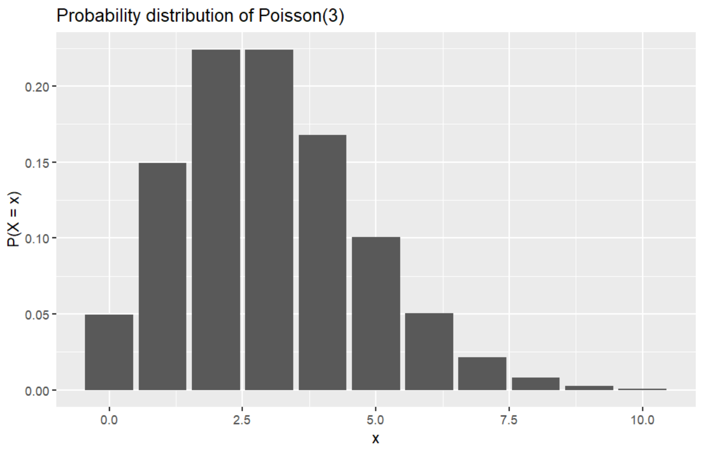
Probability distribution

- In the same manner, we can derive the entire probability distribution

```
dpois(x = 0:10, lambda = 3)
```

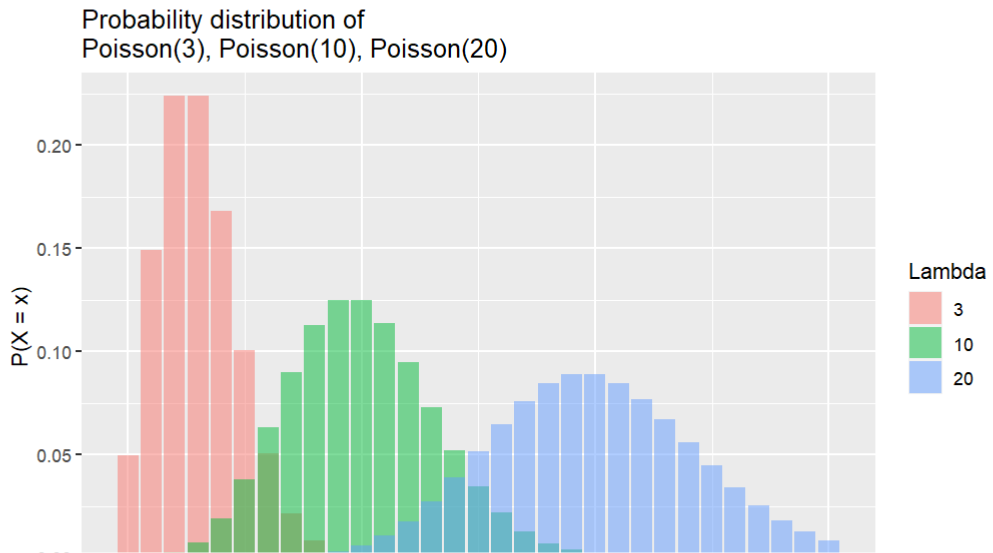
```
 [1] 0.0497870684 0.1493612051 0.2240418077 0.224041807
 [6] 0.1008188134 0.0504094067 0.0216040315 0.008101511
[11] 0.0008101512
```

```
data.frame(x = 0:10, y = dpois(0:10, lambda = 3)
  ggplot(aes(x = x, y = y)) +
    geom_bar(stat = "identity") +
  labs(title = "Probability distribution of Pois
       y = "P(X = x)")
```



Probability distribution varying lambda

```
data.frame(x = 0:30, y = dpois(0:30, lambda = 3), lambda = 3) %>%
  bind_rows(data.frame(x = 0:30, y = dpois(0:30, lambda = 10), lambda = 10)) 
  bind_rows(data.frame(x = 0:30, y = dpois(0:30, lambda = 20), lambda = 20)) 
    ggplot(aes(x = x, y = y, fill = as.factor(lambda))) +
      geom_bar(stat = "identity",
               position = "identity",
               alpha = .5) +
    labs(title = "Probability distribution of \nPoisson(3), Poisson(10), Pois
         y = "P(X = x)",
         fill = "Lambda")
```



Sampling from Poisson distribution in R

- For any non-negative value of λ, we can simulate random draws using the `rpois()` function
- `rpois()` has the arguments `n, lambda`, where `n` is the number of draws from the distribution, and `lambda` is the mean.

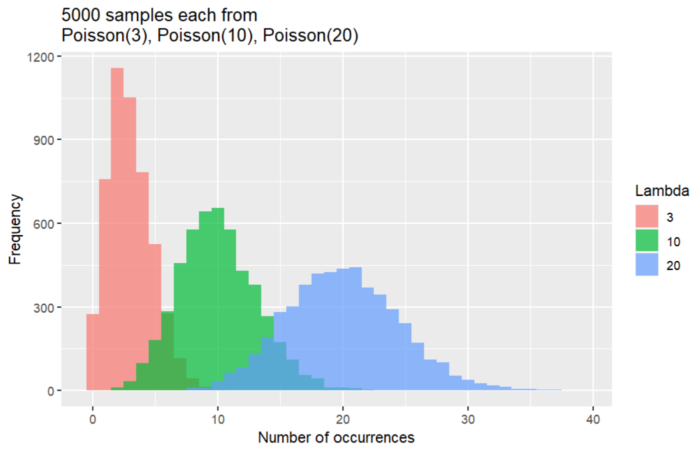
```r
set.seed(0) # so results are reproducible
inputLambda <- 3
poissonDraws <- rpois(n = 100, lambda = inputLambda)
poissonDraws
```

```
  [1] 5 2 2 3 5 2 5 6 4 3 1 2 1 4 2 4 3 4 8 2 4 6 2 4 1 2 2 0 2 5 2 3 3 3 1 5
 [38] 4 1 4 2 5 3 4 3 3 4 0 3 4 4 3 5 3 2 1 1 2 3 4 2 5 2 3 2 4 2 3 4 1 5 2 5
 [75] 2 3 5 5 2 4 6 3 4 2 2 4 2 4 1 2 1 2 1 3 5 4 4 3 2 4
```

Frequency distribution varying lambda

```
set.seed(0) # so results are reproducible
poissonL3 <- rpois(n = 5000, lambda = 3)
poissonL10 <- rpois(n = 5000, lambda = 10)
poissonL20 <- rpois(n = 5000, lambda = 20)
```

```
data.frame(poissonL3) %>%
  rename(outcome = poissonL3) %>%
  bind_cols(lambda = 3) %>%
  bind_rows(
    data.frame(poissonL10) %>%
      rename(outcome = poissonL10) %>%
      bind_cols(lambda = 10)
  ) %>%
  bind_rows(
    data.frame(poissonL20) %>%
      rename(outcome = poissonL20) %>%
      bind_cols(lambda = 20)
  ) %>%
  ggplot(aes(x = outcome,
                 fill = as.factor(lambda))) +
    geom_histogram(binwidth = 1, position = "ide
    labs(
      x = "Number of occurrences",
      y = "Frequency",
      title = "5000 samples each from \nPoisson(
      fill = "Lambda"
    )
```



Monte Carlo simulations

- Monte Carlo simulation is a computational technique used to estimate the probability distribution of an event by running simulations using random sampling.
- It is used when the analytical solution is not available or when a complex system is difficult to model using deterministic methods.
- The simulation is performed by generating a large number of random samples from a probability distribution and calculating the outcome for each sample.
- By repeating the simulation many times, the results converge to the true probability distribution due to the Law of Large Numbers.

sample function in R

The sample function in R is a powerful tool for randomly selecting items from a vector or data frame. It can be used for a variety of tasks, such as creating random samples for statistical analysis.

The basic syntax of the sample function is as follows:

```
sample(x, size, replace = FALSE, prob = NULL)
```

where `x` is the vector or data frame to sample from, `size` is the number of samples to draw, `replace` is a logical value indicating whether sampling should be done with replacement, and `prob` is a vector of probabilities for each element in `x`. If no `prob` is given, all elements in x will be sampled with equal probabilities.

Case study 1: blackjack

First, let's construct a deck of cards. For this, we will use the `expand.grid` and `paste` functions. We use `paste` to create strings by joining smaller strings. The function `expand.grid` gives us all the **combinations** of entries of two vectors.

Here is how we generate a deck of cards:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
             "Eight", "Nine", "Ten", "Jack", "Queen", "King")
deck <- expand.grid(number=numbers, suit=suits)
head(deck)
```

```
  number     suit
1    Ace Diamonds
2  Deuce Diamonds
3  Three Diamonds
4   Four Diamonds
5   Five Diamonds
6    Six Diamonds
```

```
deck <- paste(deck$number, deck$suit)
head(deck)
```

```
[1] "Ace Diamonds"   "Deuce Diamonds" "Three Diamonds" "Four Diamonds"
[5] "Five Diamonds"  "Six Diamonds"
```

With the deck constructed, we can double check that the probability of a King in the first card is 1/13 by computing the proportion of possible outcomes that satisfy our condition:

```
kings <- paste("King", suits)
mean(deck %in% kings)
```

```
[1] 0.07692308
```

Now compute the probability of a Natural 21 in Blackjack (assume all combinations are equal probability), we can do this:

```
library(gtools)
aces <- paste("Ace", suits)
facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid(number = facecard, suit = suits)
facecard <- paste(facecard$number, facecard$suit)
# gives you all possible combinations of a hand for 2 different card
#(combinations in package gtools)
hands <- combinations(52, 2, v = deck)
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
```

```
[1] 0.04826546
```

In the last line, we assume the `Ace` comes first. This is only because we know the way combination enumerates possibilities and it will list this case first.

Monte Carlo simulation of blackjack

Instead of using combinations to deduce the exact probability of a Natural 21, we can use a Monte Carlo to estimate this probability. In this case, we draw two cards over and over and keep track of how many 21s we get. We can use the function sample to draw two cards without replacements:

```
hand <- sample(deck, 2)
hand
```

```
[1] "Five Diamonds" "Ace Clubs"
```

```
# Then check if one card is an Ace and the other a face card or a 10
(hands[1] %in% aces & hands[2] %in% facecard) |
  (hands[2] %in% aces & hands[1] %in% facecard)
```

```
[1] FALSE
```

If we repeat this 10,000 times, we get a very good approximation of the probability of a Natural 21.

If we repeat this 10,000 times, we get a very good approximation of the probability of a Natural 21.

Let's start by writing a function that draws a hand and returns TRUE if we get a 21. The function does not need any arguments because it uses objects defined in the global environment. The function returns TRUE if we get a 21 and FALSE otherwise:

```
blackjack <- function(){
   hand <- sample(deck, 2)
  (hand[1] %in% aces & hand[2] %in% facecard) |
    (hand[2] %in% aces & hand[1] %in% facecard)
}
```

Now we can play this game, say, 100,000 times:

```
B <- 100000
results <- replicate(B, blackjack())
mean(results)
```

```
[1] 0.04911
```

```
# True probability: 0.04826546
```

Case 2: Monty Hall problem

In the 1970s, there was a game show called “Let’s Make a Deal” and Monty Hall was the host. At some point in the game, contestants were asked to pick one of three doors. Behind one door there was a prize. The other doors had a goat behind them to show the contestant they had lost. After the contestant picked a door, before revealing whether the chosen door contained a prize, Monty Hall would open one of the two remaining doors and show the contestant there was no prize behind that door. Then he would ask “Do you want to switch doors?” What would you do?

The result is quite counterintuitive. If you stick with the original door choice, your chances of winning a prize remain 1 in 3. However, if you switch to the other door, your chances of winning double to 2 in 3!

Here we will use a Monte Carlo simulation to see which strategy is better.

```r
B <- 10000
monty_hall <- function(strategy){
  doors <- as.character(1:3)
  prize <- sample(c("car", "goat", "goat"))
  prize_door <- doors[prize == "car"]
  my_pick  <- sample(doors, 1)
  show <- sample(doors[!doors %in% c(my_pick, prize_door)],1)
  stick <- my_pick
  stick == prize_door
  switch <- doors[!doors%in%c(my_pick, show)]
  choice <- ifelse(strategy == "stick", stick, switch)
  choice == prize_door
}
stick <- replicate(B, monty_hall("stick"))
mean(stick)
```

```
[1] 0.3296
```

```r
#> [1] 0.342
switch <- replicate(B, monty_hall("switch"))
mean(switch)
```

```
[1] 0.6722
```

As we write the code, we note that the lines starting with my_pick and show have no influence on the last logical operation when we stick to our original choice anyway.

Summary

- Common probability distributions: Binomial and Poisson
  - Theoretical properties: probability mass function, parameters, mean and variance, effect of varying parameters
  - Sampling and law of large numbers; effect of changing parameters
  - R functions:
    - `d____()`, e.g., `dbinom()`: for densities (more accurately, for discrete random variables these are probability mass functions, $P(X = x)$)
    - `p____()`, e.g., `pbinom()`: for $P(X \leq x)$
    - `r____()`, e.g., `rbinom()`: for random sample
  - Monte Carlo simulation