Simulation

Lecture 8

Quiz 2 Information

- 10 to 12 questions.
  - Lecture 5 to 8 but you may see some discrete distributions (in regards to simulation).
  - Multiple choice, multiple answer, calculation, and coding in R.
- We will release the practice quiz by Thursday.
- No multivariate calculus will be evaluated (i.e., partial derivatives and multiple integrals).

A quick overview before our last lecture!

How to deal with uncertainty?

- So far, we have seen many quantities that help us communicate an uncertain outcome:
  - PDF/PMF
  - Odds
  - Mode/Mean/Median
  - Entropy/Variance/standard deviation

However…

- Sometimes, it is difficult to compute them.
- In these situations, we can use simulation.

Outline

1. Review on Random Samples
2. Random seeds
3. Generating Random Samples
4. Running Simulations
5. Multi-Step Simulations

1. Review on Random Samples

- A random sample is a collection of random variables.
- A random sample of size n: $X_1, \ldots, X_n$.
- Examples:
  - The outcomes of ten dice rolls ($n = 10$).

Assumption about Random Samples

- Unless we make additional sampling assumptions, a random sample is assumed to be independent and identically distributed (iid).

2. Seeds

- We use algorithms to generate pseudorandom numbers, which is not truly random.
- These numbers appear random, but are actually generated by a deterministic process.

Stochastic vs Deterministic

- The word stochastic refers to having some uncertain outcome.
- The opposite of stochastic is deterministic: an outcome that will be known with 100% certainty (0 entropy).

Example of a pseudorandom number generator

- Starting with a number $x_0 \in (0, 1)$, we generate the next number by:

$$x_{i+1} = 4x_i(1 - x_i).$$

- The generator produces pseudorandom numbers between 0 and 1 ($X \sim \text{Uniform}(0, 1)$).

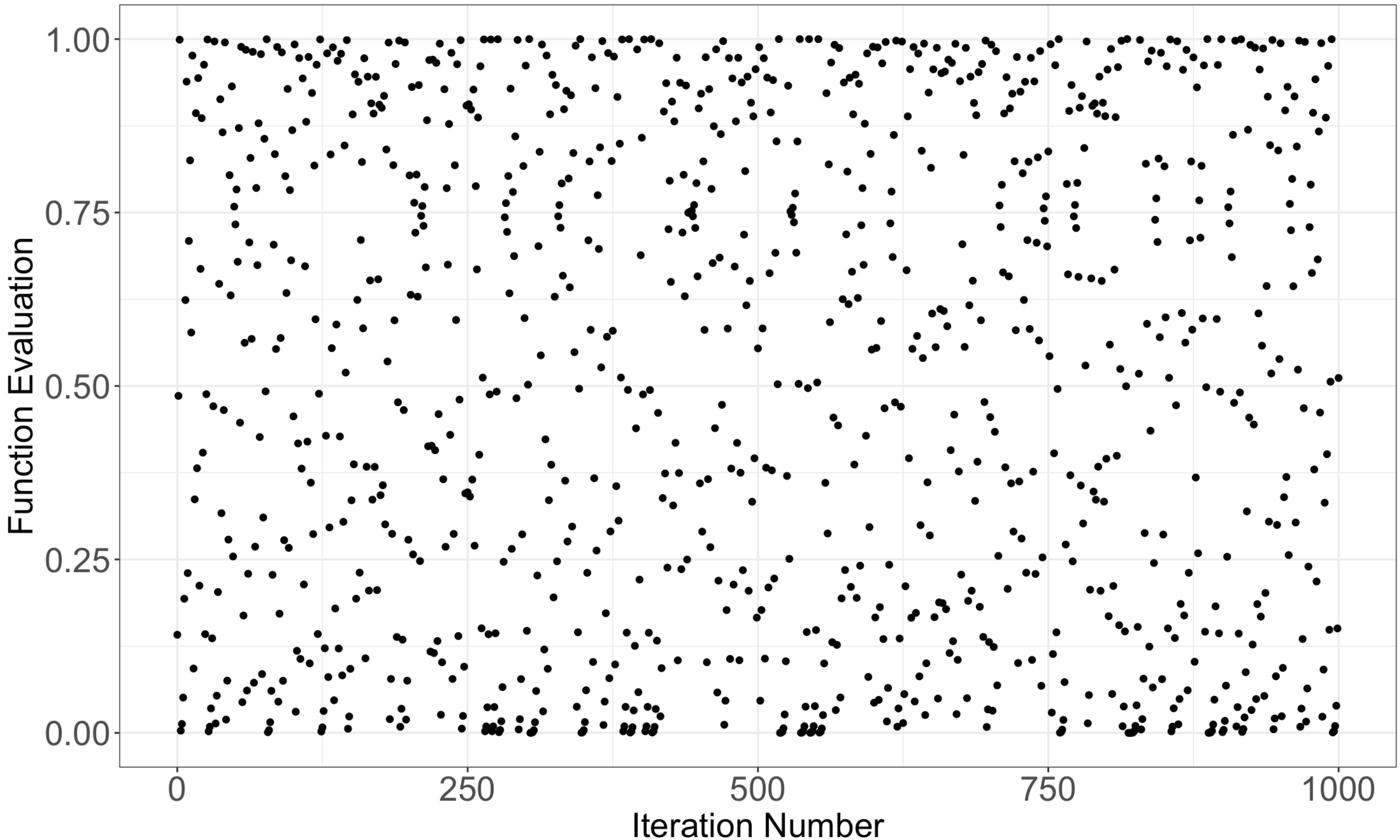
The output

- Here is the resulting sequence when we start with $x_0 = 0.3$ and iterate $n = 1000$ times:

R Code | Start Over | Run Code

```
n <- 1000
x <- 0.3
for (i in 1:n) x[i + 1] <- 4 * x[i] * (1 - x[i])
head(x, 10)
```

Plotting values from vector x



All pseudorandom number generators have some pitfalls

- The generated sequence is deterministic, but it looks like a random sample.
- Moreover, in this example, neighbouring pairs are not independent of each other.
- Some sophisticated algorithms can produce outcomes that more closely resemble a random sample.

The seed or random state

- The seed (or random state) in a pseudorandom number generator is some initial values that determines the generated sequence.
  - In this example, x_0 is the initial value.
- In R, we can use `set.seed()`.
- In `Python`, we can use `numpy.random.seed()`.
- It is important to set seed to make sure our result is reproducible.

3. Generating Random Samples: Code

- We will look at some `R` and `Python` functions.
- Note we will focus on discrete distributions here.

3.1. Sampling from Finite Number of Outcomes

- `R`: `sample()`.
- `Python`: `numpy.random.choice()`.
- Both `R` and `Python` uses their own pseudorandom number generators.

`sample()` in R

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

- `x` is the vector of outcome.
- `size` is the desired sample size.
- `replace = TRUE` for sampling with replacement.
- `prob` is the vector of the probabilities of the outcomes respective to `x`.

- If `prob` does not add up to 1, R will automatically adjusts the probabilities so that they add up to 1.

R Example

- Here is an example of generating $n = 10$ items using the Mario Kart item distribution from `lecture1`.

R Code Start Over Run Code

```r
set.seed(1)
outcomes <- c("banana", "bob-omb", "coin", "horn", "shell")
probs <- c(0.12, 0.05, 0.75, 0.03, 0.05)
n <- 10
sample(outcomes, size = n, replace = TRUE, prob = probs)
```

`numpy.random.choice()` in `Python`

> random.choice(a, size=None, replace=True, p=None)

- `a` is the array of outcomes.
- `size` is the desired sample size.
- `p` is the array of the probabilities respective to `x`.

- `p` needs to add up to 1.

Python Example

- Using the Mario Kart example again, we have the following:

Python Code | Start Over | Run Code

```python
import numpy
numpy.random.seed(551)
outcomes = ["banana", "bob-omb", "coin", "horn", "shell"]
probs = [0.12, 0.05, 0.75, 0.03, 0.05]
n = 10
numpy.random.choice(a = outcomes, size = n, p = probs)
```

3.2. Sampling from a Distribution Family

- R: `r<dist>()` function, where `<dist>` is replaced with a short form of the distribution family's name.
- Python: `scipy.stats` from the `scipy` library.

The table below summarizes the functions for discrete distribution families

Family	R Function	Python Function
Binomial	`rbinom()`	`scipy.stats.binom.rvs()`
Geometric	`rgeom()`	`scipy.stats.geom.rvs()`
Negative Binomial	`rnbinom()`	`scipy.stats.nbinom.rvs()`
Poisson	`rpois()`	`scipy.stats.poisson.rvs()`

How to use these functions?

> rbinom(n, …)

> scipy.stats.binom.rvs(…, size=1)

- Sample size:
  - For R: the argument `n`, which comes first.
  - For `Python`: the argument `size`, which comes last.

How to use these functions?

> rbinom(n, size, prob)

> scipy.stats.binom.rvs(n, p, size=1)

- In both languages, each parameter has its own argument.
- Sometimes, we have the option for different parameterizations.
- Be sure to specify the exact number of parameters required to identify the distribution!

Generate $n = 10$ Binomial random numbers with $p = 0.6$ and 5 trial

R Code Start Over Run Code

```r
set.seed(551)
rbinom_output <- rbinom(n = 10, size = 5, prob = 0.6)
rbinom_output
```

Python Code Start Over Run Code

```python
import scipy
import numpy
numpy.random.seed(551)  # scipy.stats uses numpy.random to generate its random numbers
scipy.stats.binom.rvs(n = 5, p = 0.6, size = 10)
```

Negative Binomial Example

- Suppose the following:

$$X \sim \text{Negative Binomial}(k, p).$$

- X refers to the number of failures in independent Bernoulli trials before experiencing k successes with probability p.

R Function `rnbinom()`

- We can sample `n` Negative Binomial-distributed random numbers with `size` as $k = 5$ and `prob` as $p = 0.6$:

R Code Start Over Run Code

```
set.seed(551)
rnbinom(n = 10, size = 5, prob = 0.6)
```

Specifying too few parameters would result in an error

R Code Start Over Run Code

```r
set.seed(551)
rnbinom(n = 10, size = 5)
```

- R would indicate that `prob` is missing.

Another way to use `rnbinom()` in `R`!

- Recall the expected value of a Negative Binomial random variable is

$$\mu = \mathbb{E}(X) = \frac{k(1-p)}{p} = \frac{5(1-0.6)}{0.6} = 3.33.$$

- A Negative Binomial distribution can also be parameterized with k and its mean.

Therefore...

- We can use the the argument `mu` in the random number generator `rnbinom()`. Note `mu` refers to $\mu = \mathbb{E}(X)$:

R Code | Start Over | Run Code

```r
set.seed(551)
rnbinom(n = 10, size = 5, mu = 3.33)
```

R Code | Start Over | Run Code

```r
set.seed(551)
rnbinom(n = 10, size = 5, prob = 0.6)
```

iClicker Question

Suppose you want to simulate hourly bank branch queues of customers. **Historically**, hourly queues show an average of 10 people.

What R random number generator will you use to simulate 20 random numbers?

A. `rpois(n = 20, lambda = 1 / 10)`

B. `rbinom(n = 20, size = 10, prob = 1 / 10)`

C. `rpois(n = 20, lambda = 10)`

D. `rgeom(n = 20, prob = 1 / 10)`

Answer

- We need to simulate numbers of people in a time interval (i.e., hourly). These numbers are **integers** and non-negative.
- Therefore, we can go ahead with Poisson.
- Now, the parameter to use is the rate $\lambda = 10$ people per hour.

How do we read the function's output below?

- These 20 numbers indicate the simulated number of people per hour: 12 people, 10 people, 11 people, …

R Code Start Over Run Code

```r
set.seed(551) # Reproducibility
rpois(n = 20, lambda = 10)
```

iClicker Question

Suppose you want to simulate the number of non-authentic bubble tea shops you will try before encountering your very first authentic one in Vancouver. Overall, **it is known that 70% of bubbl tea shops in Vancouver are considered non-authentic**.

What R random number generator will you use to simulate 15 random numbers?

A. `rbinom(n = 15, size = 15, prob = 0.7)`

B. `rgeom(n = 15, prob = 0.3)`

C. `rbinom(n = 15, size = 15, prob = 0.3)`

B. `rgeom(n = 15, prob = 0.7)`

UBC Master of Data Science

Answer

- We need to simulate the number of failures (i.e., non-authentic) before encountering the first success (i.e., first authentic).
- Therefore, a Geometric distribution with $p = 0.3$ is our way to go! Note that p is the probability of success (i.e., authentic).

How do we read the function's output below?

- We have 15 random numbers indicating the simulated number of non-authentic places before encountering the first authentic one: 0 places, 0 places, 0 places, 5 places, …

R Code Start Over Run Code

```r
set.seed(551) # Reproducibility
rgeom(n = 15, prob = 0.3)
```

About Student Experience of Instruction (SEI) Surveys

- They help us improve our teaching!
- UBC uses these in evaluating professors for hiring, and tenure promotion.

What to comment on:

- Things that worked well for you!
- Things that could be improved!

How to write your comments:

- Be constructive.
- Be respectful.
- Be concrete.

SEI Surveys (10-15 mins break)

> You voices can make a difference!

- Please go to this website: https://seoi.ubc.ca/surveys or scan the QR code, and fill out the survey for
  - Vincent Liu (**Instructor**)
  - Mahdi Asmae (**TA**)
  - Anne-Sophie Fratzscher (**TA**)
  - Cindy Zhang (**TA**)

4. Running Simulations

- There are two ways to calculate probabilistic quantities (e.g., means and probabilities):

1. The distribution-based approach (using the distribution), resulting in true values.
2. The empirical approach (using data), resulting in approximate values that improve as the sample size increases.

Example: The Mean

- The mean of a discrete random variable X can be calculated as

$$\mathbb{E}(X) = \sum_x x \cdot P(X = x).$$

- Or can be approximated using the empirical approach from a random sample $X_1, \ldots, X_n$ by

$$\mathbb{E}(X) \approx \frac{1}{n} \sum_{i=1}^{n} X_i.$$

4.1. R functions for Calculating Empirical Quantities

- `mean()` calculates the sample average

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

- `var()` calculates the sample variance

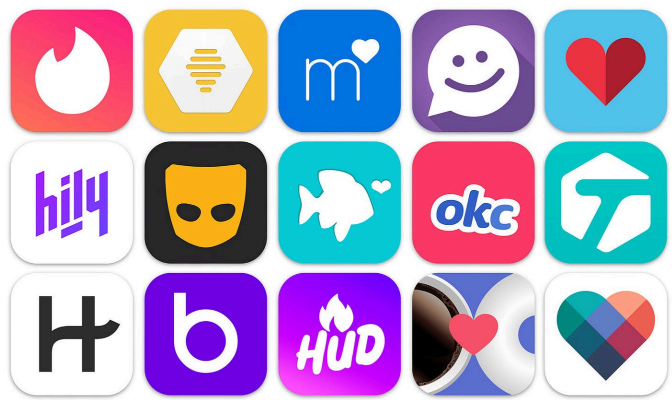
$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2.$$

Furthermore...

- `sd()` calculates the sample standard deviation.
- `quantile()` calculates the empirical p-quantile: the np'th largest (rounded up) observation in a sample of size n.
- To get the entire PMF, use the `table()` function, or more conveniently, the `janitor::tabyl()` function.
- To get the mode, either get it manually using the `table()` or `janitor::tabyl()` function, or you can use `DescTools::Mode()`.

4.2. Basic Simulation

- Consider a random person dating via a dating app with a probability of having a successful date of 0.7.



Also...

- Suppose we want to evaluate the number of failed dates before they experience 5 successful dates.

- Define X to be the number of failed dates before experiencing 5 successful ones, then

$$X \sim \text{Negative Binomial}(k = 5, p = 0.7).$$

Comparing distribution-based and empirical approach

- First, generate our random sample ($n = 10000$ observations).

R Code Start Over Run Code

```r
set.seed(551)
k <- 5
p <- 0.7
n <- 10000
random_sample <- rnbinom(n, size = k, prob = p)
head(random_sample, 30) # Showing the first 30 random numbers in vector random_sample
```

4.2.1. Mean

- Theoretically, the mean of X is $\mathbb{E}(X) = \frac{k(1-p)}{p}$.

R Code | Start Over | Run Code

```r
(k * (1 - p)) / p
```

- Empirically, we can approximate $\mathbb{E}(X)$ with the sample average in `random_sample`:

R Code | Start Over | Run Code

```r
mean(random_sample)
```

4.2.2. Variance

- Theoretically, the variance of X is $\mathrm{Var}(X) = \frac{k(1-p)}{p^2}$.

R Code Start Over Run Code

```
(k * (1 - p)) / p^2
```

- Empirically, we can approximate $\mathrm{Var}(X)$ with the sample variance in `random_sample`:

R Code Start Over Run Code

```
var(random_sample)
```

4.2.3. Standard deviation

- Theoretically, the standard deviation of X is

$$\operatorname{sd}(X) = \sqrt{\frac{k(1-p)}{p^2}}.$$

R Code Start Over Run Code

```r
sqrt((1 - p) * k / p^2)
```

- Empirically, we can approximate $\operatorname{sd}(X)$ with the sample standard deviation in `random_sample`:

R Code Start Over Run Code

```r
sd(random_sample)
```

4.2.4. Probability of Seeing 0 Failures (i.e., 0 failed dates!)

- Theoretically, this probability can be computed as

$$P(X = 0) = \binom{k-1}{0} p^k (1-p)^x.$$

- Using R, we can compute this probability directly:

R Code | Start Over | Run Code

```
dnbinom(x = 0, size = k, prob = p)
```

Empirically...

- We can approximate $P(X = 0)$ by counting the number of random numbers equal to 0 in `random_sample` and dividing this count by $n = 10000$.

R Code Start Over Run Code

```r
head(random_sample) # First 6 random numbers out of 10000
head(random_sample == 0) # First 6 logical values out of 10000
mean(random_sample == 0) # Using function mean()
```

4.2.5. Probability Mass Function

- We can also do it for $P(X = i)$ with $i = 1, 2, \ldots$ (i.e., the whole PMF!).

R Code Start Over Run Code

```r
library(tidyverse)
library(janitor)

PMF <- tabyl(random_sample) %>%
  select(x = random_sample, Empirical = percent) %>%
  mutate(Theoretical = dnbinom(x, size = k, prob = p))

PMF <- PMF %>%
  mutate(
    Theoretical = round(Theoretical, 4),
    Empirical = round(Empirical, 4)
  )
```

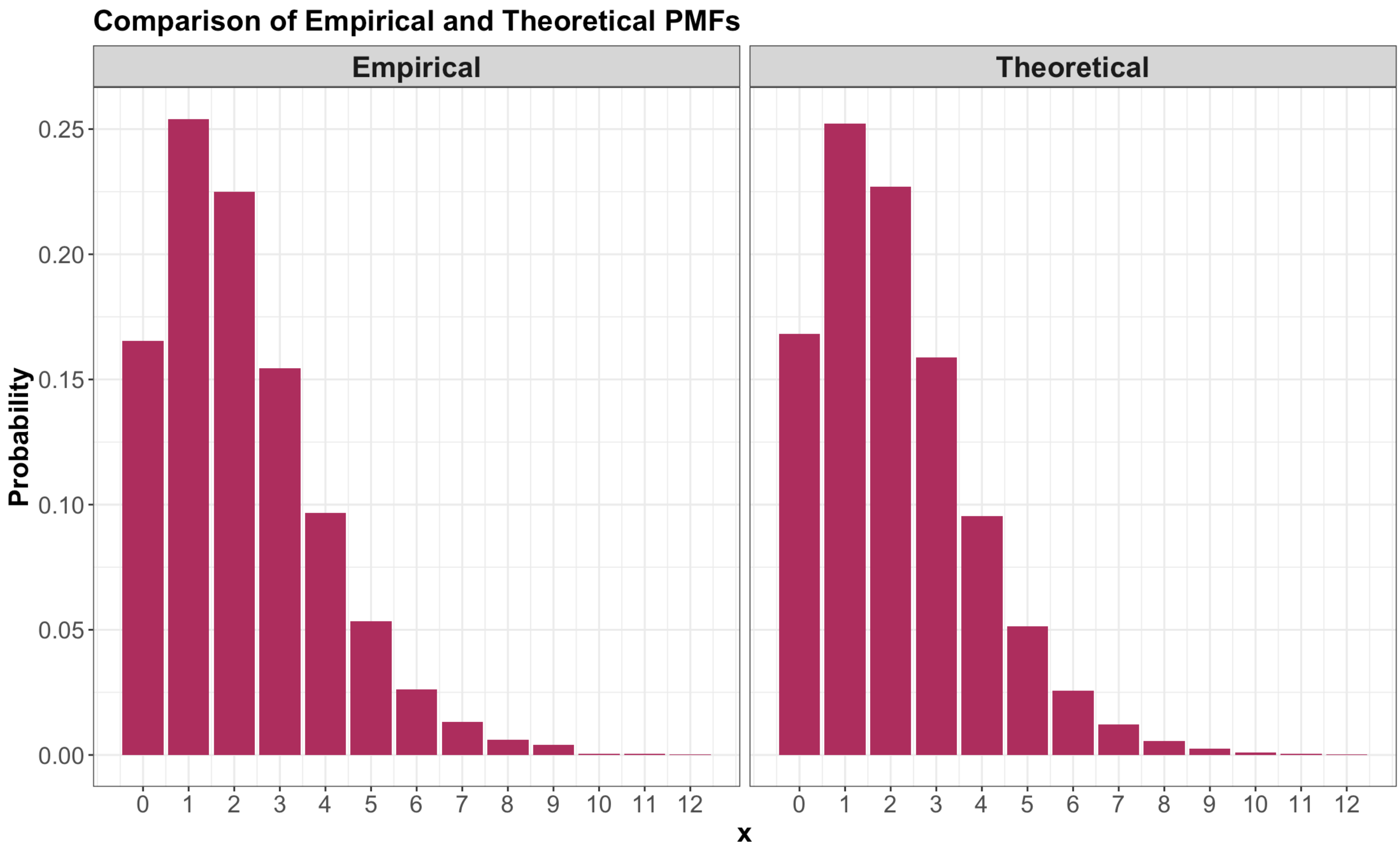
Showing the output `PMF`

R Code Start Over Run Code

```r
PMF
```

- Note that the probabilities are similar!

Plotting both PMFs



4.2.6. Mode

- The mode is the outcome with the largest probability.
- From our previous plots, we can see that the mode is 1.

R Code Start Over Run Code

```r
## Theoretical
PMF %>% filter(Theoretical == max(Theoretical)) %>%
  pull(x)
```

R Code Start Over Run Code

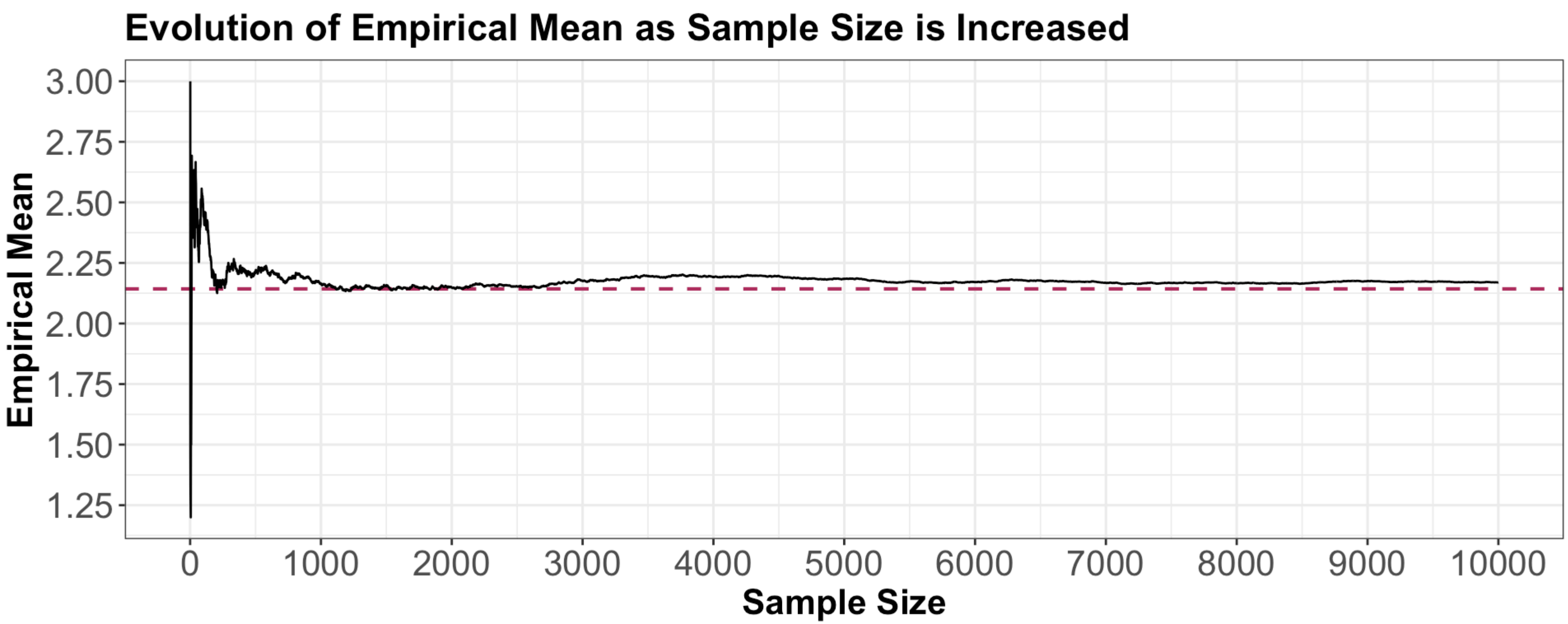
```r
## Empirical
PMF %>% filter(Empirical == max(Empirical)) %>%
  pull(x)
```

Why is our simulation so accurate?

- The Law of Large of Numbers states that, as we increase our sample size n, our empirical mean converges to the true mean.
- That is, $\bar{X} \to \mu$ as $n \to \infty$.

Increasing the sample size n in our example

$$X \sim \text{Negative Binomial}(k = 5, p = 0.7) \text{ with } \mathbb{E}(X) = 2.14.$$



5. Multi-Step Simulations

- Simulation gets more interesting when we want to calculate things for a random variable that transforms and/or combines multiple random variables.

For example...

- Consider a random variable T that we can obtain as follows:

$$X \sim \text{Poisson}(\lambda = 5)$$
$$T = \sum_{i=1}^{X} D_i.$$

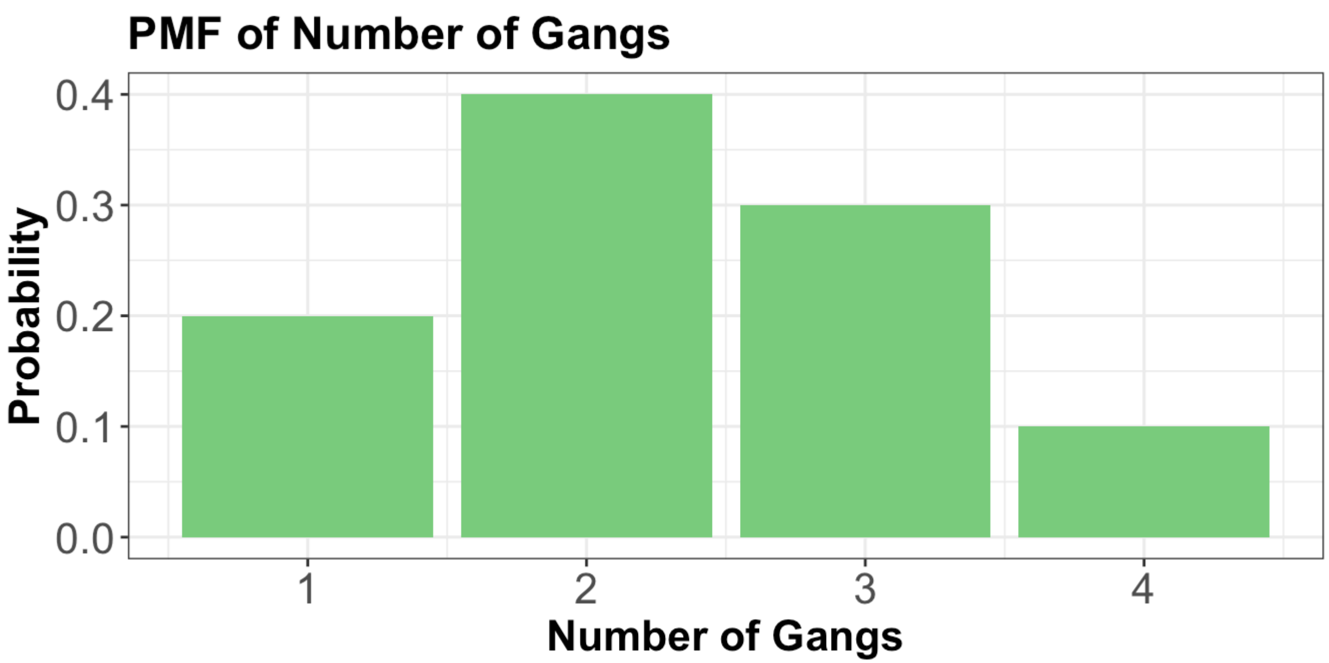
  where each D_i are *iid* with some specified distribution.

- We would first generate X, then generate X values of D_i, then sum those up to get T.

The Port of Vancouver Example

- Whenever a ship arrives, they request a certain number of gangs (groups of people) to help unload the ship.

- Let D_i denotes the random variable for the i-th ship with the following PMF:



Simulation function

- Given a number of ships s, the function simulates the number of gangs requested, and sums up the gangs demand.
- That is, it simulates $T = \sum_{i=1}^{s} D_i$.

R Code Start Over Run Code

```r
gang <- 1:4
p <- c(0.2, 0.4, 0.3, 0.1)

#' Generate gang demand
#'
#' Simulates the GRAND TOTAL number of gangs requested, if each ship
#' requests a random number of gangs.
#'
#' @param n_ships Number of ships that are making demands.
#' @param gangs Possible gang demands made by a ship.
#' @param prob Probabilities of gang demand corresponding to "gangs."
#'
```

```r
#' @return Number representing the total gang demand
demand_gangs <- function(n_ships, gangs = gang, prob = p) {
  if (length(gangs) == 1) {
    gangs <- c(gangs, gangs)
    prob <- c(1, 1)
  }
  requests <- sample(
    gangs,
    size = n_ships,
    replace = TRUE,
    prob = prob
  )
  sum(requests)
}
```

Using the simulation function

- As an example, we can simulate the total gang request for $s = 10$ ships:

R Code | Start Over | Run Code

```r
set.seed(551) # Reproducibility
demand_gangs(10)
```

If the number of ships arriving is random

- Suppose S, the number of ships arriving on a given day, follows a Poisson distribution with a mean of $\lambda = 5 \text{ ships}$.
- What is the distribution of total gang requests on a given day?

Two-step simulation

1. Generate the number of ships $S \sim \text{Poisson}(\lambda = 5 \text{ ships})$ for n days .
2. For each day, simulate the total gang request $T = \sum_{i=1}^{S} D_i$ for the simulated number of ships.
3. We now have a random sample of size n.

The Code

- Let us try this, obtaining a sample of $n = 10000$ days.

R Code | Start Over | Run Code

```r
n_days <- 10000

# Setting global seed!
set.seed(551)

## Step 1: generate a bunch of ships arriving each day.
arrivals <- rpois(n_days, lambda = 5)
head(arrivals)
```

R Code | Start Over | Run Code

```r
## Step 2: Simulate the grand total gang request on each day.
library(purrr)
total_requests <- map_int(arrivals, demand_gangs)
head(total_requests)
```

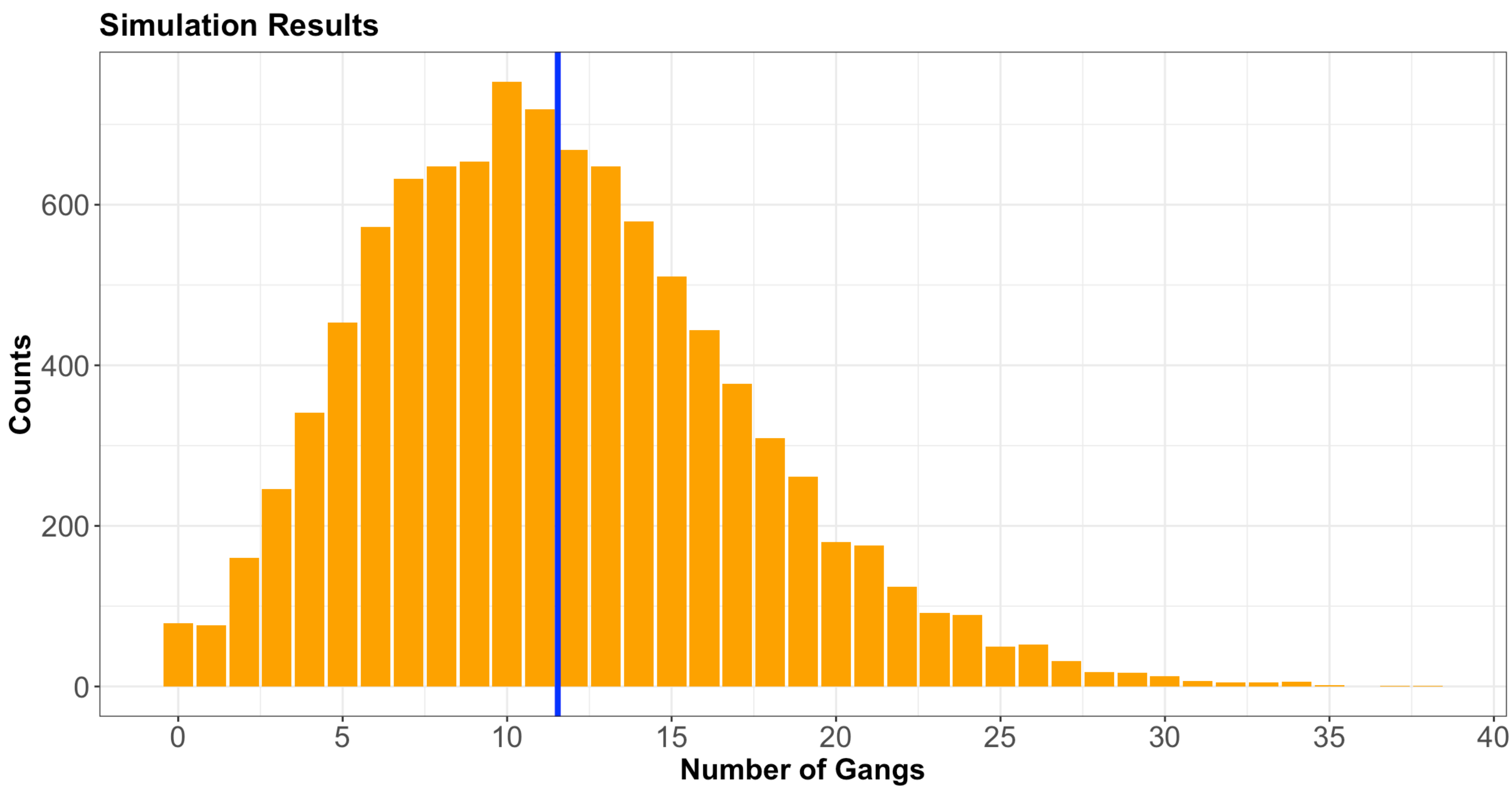
Then…

R Code

```r
## Step 3: Compute mean and variance.
simulation_outputs <- tibble(
  mean = mean(total_requests),
  variance = var(total_requests)
)
simulation_outputs
```

- For $n = 10000$, the summary statistics indicate that we expect a demand of 11.5 gangs on a given day in the whole port.

Checking the empirical distribution of total gang requests!



Today's Learning Goals

- Generate a random sample from a distribution in R.
- Reproduce the same random sample each time you re-run your code in R by setting the seed or random state.
- Use simulation to approximate distribution properties (e.g., mean and variance), especially for random variables involving multiple other random variables.
- Understand why simulations can approximate true properties of a distribution.

Final Wrap Up

- Probability is everywhere: finance, computer science, physics, engineering, and data science!
- You will learn more advanced topics that build on what we’ve covered, such as regression, Bayesian inference, machine learning models, and LLMs.