Day 5: Function, Loops, and Monte Carlo Simulations

Qingyin Cai

Department of Applied Economics
University of Minnesota

WebR Status

🟢 Ready!

Introduction

- In the previous lecture, we learned how to do regression analysis using R, which is a fundamental skill for econometric analysis.
- Today, we will learn how to code Monte Carlo simulations in R. Monte Carlo simulations are a very important tool in learning econometrics and statistics. With Monte Carlo simulations, you can test any kind of statistical theory or property, which is very useful!!

Note

Before we start the Monte Carlo simulation, let's review a few key R concepts:

- `for loop` function: how they work and when to use them.
- Writing your own functions: we won't need this for today's simulation, but it's an important skill to practice for future coding tasks.

🎯 Learning Objectives

- to be able to write code for your own R functions.
- to be able to write code for a Monte Carlo simulation using the loop function.

✳ Reference

- Section 19 Functions
- Section 20 Iteration in R for Data Science

Today's Outline

- User-Defined Functions
- Loops
  - Basics
  - How to Save the Loop Output
  - Multiple Outputs
- Introduction to Monte Carlo Simulations
  - Demonstration
- Overall Review

User-Defined Functions

Introduction to User-Defined Functions

Intro

You can define your own functions. The beauty of creating your own functions is that you can reuse the same code over and over again without having to rewrite it. A function is more useful when the task is longer and more complicated.

Example Situations

- When you want to automate the process of data cleaning.
- When you do a complicated simulation or resampling methods, such as bootstrapping or Monte Carlo simulations.

Introduction to User-Defined Functions

Basics

You can define your own functions using the `function()` function.

General Syntax

```
function_name <- function(arg1, arg2, ...){
  code to be executed

  return(output)
}
```

Note

- You need to define the function name (`function_name`), what kind of inputs the function takes (`arg1`, `arg2`, etc.), and how the function processes using the given input objects.
- The `return()` function is used to return the output of the function. By default, the output defined in the last line of the function is returned.

Introduction to User-Defined Functions

Examples

```r
function_name <- function(arg1, arg2, ...){
  code to be executed

  return(output)
}
```

1. A simple function

Run Code

```r
# --- Define a function --- #
sum_of_two_numbers <- function(x, y){
  x + y
}

# --- Use the function --- #
sum_of_two_numbers(x = 3, y = 4)
```

2. A function with multiple outputs

Run Code

```r
# --- Define a function --- #
mean_sd <- function(vec_x){
  c(mean = mean(vec_x), sd = sd(vec_x))
}

# --- Use the function --- #
```

```
7  vec_test <- c(3, 5, 7, 8)
8  mean_sd(vec_x = vec_test)
```

Introduction to User-Defined Functions

Default Value

You can set default values for function arguments by `argument = value`.

Example:

Run Code

```r
# --- Function to convert a Acre of Hectare --- #
acre_to_ha <- function(x = 1){
  x * 0.404686
}

# --- Use the function --- #
acre_to_ha() # because x = 1 by default

acre_to_ha(x = 1)
```

Load Functions from a File

If you have multiple functions or a long function, you might want to save the function in a separate file and load it when you need it.

For example:

- Save the function code file in a `.R` file (`.Rmd`, etc.).
  - Click `File` -> `Save As...`
  - Choose a name, e.g., `my_function.R`.
- Load the function using the `source()` function.

Let's practice this on your Rstudio!

In-class Exercise

Exercise 1

Questions

1. Write a function (you can name it whatever you want) to calculate the area of a circle with a given radius. The function should return the area of the circle. (Hint1: `Area = π × r^2`; Hint2: Use `pi`, which is a built-in constant for the value of in R.)
2. Write a function to count the number of NA values in a given vector. (Hint: use the `is.na()` function.)
3. Write a function called `mean_no_na` that calculates the mean (average) of a numeric vector, but ignores any NA values.

Run Code

```
1  # You can write your code here.
```

In-class Exercise

Exercise 1

Answers

1. Write a function (you can name it whatever you want) to calculate the area of a circle with a given radius. The function should return the area of the circle. (Hint1: `Area = π × r^2`; Hint2: Use `pi`, which is a built-in constant for the value of π in R.)
2. Write a function to count the number of NA values in a given vector. (Hint: use the `is.na()` function.)
3. Write a function called `mean_no_na` that calculates the mean (average) of a numeric vector, but ignores any NA values.

Run Code

```r
# Question 1
circle_area <- function(radius) {
  area <- pi * radius^2
  return(area)
}

circle_area(3)

# Question 2
count_na <- function(x) {
  na_count <- sum(is.na(x))
  return(na_count)
}
```

```r

count_na(c(1, 2, NA, 4, NA, 6))

# Question 3
mean_no_na <- function(x) {
  avg <- mean(x, na.rm = TRUE)
  return(avg)
}


mean_no_na(c(1, 2, 3, NA, 5))
```

In-class Exercise

Exercise 2 (optional)

Questions

You're a data expert at a store chain. The company needs to study its monthly sales growth to plan better. They expect sales to grow by a fixed percentage each month. Your job is to create an R function that shows sales growth over a year.

For sales growth, use the following formula:

$$S_t = S_0 \times (1 + g)^{t-1}$$

, where S_t is the sales in month t, S_0 is the starting sales, and g is the growth rate.

Create a function called `monthly_sales_growth` with the following three inputs:

- `initial_sales`: Starting sales (in thousands of dollars).
- `growth_rate`: Monthly growth rate (as a decimal, like 0.03 for 3% growth).
- `months`: How many months to predict (usually 12 for a year).

The function should give back a vector of numbers (or it would be even better if you could show in a `data.frame` or `data.table` in which two columns, e.g., month and sales, show the expected sales for each month.)

Run Code

```
1  # You can write your code here.
```

In-class Exercise

Exercise 2 (optional)

Answers

You're a data expert at a store chain. The company needs to study its monthly sales growth to plan better. They expect sales to grow by a fixed percentage each month. Your job is to create an R function that shows sales growth over a year.

For sales growth, use the following formula:

$$S_t = S_0 \times (1 + g)^{t-1}$$

, where S_t is the sales in month t, S_0 is the starting sales, and g is the growth rate.

Create a function called `monthly_sales_growth` with the following three inputs:

- `initial_sales`: Starting sales (in thousands of dollars).
- `growth_rate`: Monthly growth rate (as a decimal, like 0.03 for 3% growth).
- `months`: How many months to predict (usually 12 for a year).

The function should give back a vector of numbers (or it would be even better if you could show in a `data.frame` or `data.table` in which two columns, e.g., month and sales, show the expected sales for each month.)

Run Code

```r
monthly_sales_growth <- function(initial_sales, growth_rate, months = 12) {
  # months: positive integer (e.g., 12 for a year)
  t <- seq_len(months) # a vector, or t <- 1: months
  sales <- initial_sales * (1 + growth_rate)^(t - 1)
```

```r
  return(sales)
}

# or using data.table/data.frame

monthly_sales_growth <- function(initial_sales, growth_rate, months = 12) {
  # months: positive integer (e.g., 12 for a year)
  t <- seq_len(months) # a vector, or t <- 1: months
  sales <- initial_sales * (1 + growth_rate)^(t - 1)
  data.frame(month = t, sales = sales) # or data.table(month = t, sales = sales)
}


# Example: $200k starting sales, 3% monthly growth, 12 months
monthly_sales_growth(initial_sales = 200, growth_rate = 0.03, months = 12)
```

Loops

Why loop?

Using Loop is useful when you want to repeat the same task (but with a slight change in parameters) over and over again.

Examples

- Downloading the data from the web iteratively.
  - When you want to download the ag-production data from USDA-NASS, you are limited to download 50,000 records per query. You need to repeatedly download the data until you get all the data you need.
  - USDA crop scale data, NOAA weather data, etc.
- Loading multiple data files in a folder.
- Running the same regression analysis for multiple datasets.
- Running simulations or resampling methods, such as bootstrapping or Monte Carlo simulations.

While there are several looping commands in R (e.g., `foreach`, `lapply`, etc.), we will use the `for loop` function, as it is the most basic and widely used looping function in R.

For loops

Basics

The `for loop` function is a built-in R function. The syntax of the `for loop` is very simple.

Syntax:

```
for (name in collection) {
  # code to run for each element
}
```

Components

- `name` (loop variable): a placeholder that takes on one value from the collection each time the loop runs.
- `collection`: the set of values you want to loop over (usually a vector or list).
- `code block`: the instructions you want R to execute at each step.

Note

- On each iteration, the loop variable (`name`) takes the **next value** from the collection.
- The collection can be:
  - A numeric sequence (e.g., `1:5`)
  - A character vector (e.g., `c("apple", "banana")`)
  - A list of objects (e.g., datasets)

For loops

Examples

1. Print the numbers from 1 to 5.

Loop variable `i` takes each number in the sequence `1:5` in order and print the value of `i`.

Run Code

```
for (i in 1:5){
  print(i)
}
```

2. Print characters in a list.

Loop variable `x` takes each character in the list `list("I", "like", "cats")` in order and print the value of `x`.

Run Code

```
for (x in list("I", "like", "cats")){
  print(x)
}
```

3. Calculate the mean of each element in a list.

Can you tell me what's going on in the following code?

Run Code

```
ls_seq_num <- list(1:3, 4:6, 7:9)

for (seq_num in ls_seq_num){
  print(mean(seq_num))
```

```
5  }
```

In-class Exercise

Exercise 1

Questions

In econometric class, we use the `rnorm()` function a lot! It is a function that generates random numbers from a normal distribution. See `?rnorm` for more details.

The basic syntax is `rnorm(n, mean = 0, sd = 1)`, where `n` is the number of random numbers you want to generate, `mean` is the mean of the normal distribution, and `sd` is the standard deviation of the normal distribution. So `rnorm(n, mean =0, sd = 1)` generates `n` random numbers from a standard normal distribution.

Generate 1000 random numbers from a standard normal distribution and calculate the mean the numbers (use `mean()` function), and print the results. Repeat this process 10 times using the `for loop`.

Run Code

```
for (i in 1:10){
  # write your code here

}
```

In-class Exercise

Exercise 1

Answers

In econometric class, we use the `rnorm()` function a lot! It is a function that generates random numbers from a normal distribution. See `?rnorm` for more details.

The basic syntax is `rnorm(n, mean = 0, sd = 1)`, where `n` is the number of random numbers you want to generate, `mean` is the mean of the normal distribution, and `sd` is the standard deviation of the normal distribution. So `rnorm(n, mean = 0, sd = 1)` generates `n` random numbers from a standard normal distribution.

Generate 1000 random numbers from a standard normal distribution and calculate the mean the numbers (use `mean()` function), and print the results. Repeat this process 10 times using the `for loop`.

Run Code

```r
# Set seed for reproducibility
set.seed(123)

# Repeat the process 10 times
for (i in 1:10) {
  # Step 1: Generate 1000 random numbers from standard normal
  x <- rnorm(1000, mean = 0, sd = 1)

  # Step 2: Calculate the sample mean
  m <- mean(x)
```

```
11
12    # Step 3: Print the result
13    print(m)
14  }
```

In-class Exercise

Exercise 2 (nested loop)

Questions

You can nest the `for loop` inside another `for loop`. For example,

Run Code

```
# Outer loop
for (i in 1:3) {
  # Inner loop
  for (j in 1:2) {
    print(paste("i =", i, "j =", j))
  }
}
```

Using the above code as a reference, fill in the following empty 3 x 3 matrix with the sum of the row and column indices.

The output should look like this:

```
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    3    4    5
[3,]    4    5    6
```

Run Code

```
# Here, is the empty 3 x 3 matrix.
empty_matrix <- matrix(NA, nrow = 3, ncol = 3)
```

In-class Exercise

Exercise 2 (nested loop)

Answers

You can nest the `for loop` inside another `for loop`. For example,

Run Code

```
# Outer loop
for (i in 1:3) {
  # Inner loop
  for (j in 1:2) {
    print(paste("i =", i, "j =", j))
  }
}
```

Using the above code as a reference, fill in the following empty 3 x 3 matrix with the sum of the row and column indices.

The output should look like this:

```
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    3    4    5
[3,]    4    5    6
```

Run Code

```
# Here, is the empty 3 x 3 matrix.
empty_matrix <- matrix(NA, nrow = 3, ncol = 3)
for (i in 1:3){
```

```
  for (j in 1:3){
    empty_matrix[i, j] <- i + j
  }
}
print(empty_matrix)
```

For loops: How to Save the loop output?

Introduction

Unlike R functions we have seen so far, `for loop` does not have a return value. It just iterates the process we defined in the loop.

Let's do some experiments:

Experiment 1

Run Code

```
ls_seq_num <- list(1:3, 4:6, 7:9)

for (seq_num in ls_seq_num){
  # save the results in x
  x <- mean(seq_num)
}

# What is the value of x?
x
```

Every round of the loop, the variables defined inside the loop are updated.

For loops: How to Save the loop output?

Introduction

Unlike R functions we have seen so far, `for loop` does not have a return value. It just iterates the process we defined in the loop.

Let's do some experiments:

Experiment 2

Run Code

```
ls_seq_num <- list(1:3, 4:6, 7:9)

x <- for (seq_num in ls_seq_num){
  # save the results in x
  mean(seq_num)
}

# What is the value of x?
x
```

- You cannot assign loop to a variable directly (e.g `x <- for (i in 1:3){print(i)}` does not work).

For loops: How to Save the loop output?

Basics

- To save the results of the loop, you need to create an empty object before the loop and save the output in the object in each iteration. (You did this in the exercise 2!)
  - The object can be a vector, a list, a matrix, or a data frame (or data.table), depending on the type of the output you want to save.

Example

Suppose you want to cube each number in the sequence `1:5`.

Run Code

```
# --- Create an empty vector --- #
output_storage <- rep(0, 5)

# --- for loop --- #
for (i in 1:5){
  output_storage[i] <- i^3
}

output_storage
```

Note

- Since the output of each iteration is a number, vector is a good choice for the storage object. (Alternatively you can use a list object.)

Multiple Outputs

What if we want to have multiple outputs from the loop and combine them into a single dataset?

Example

Let's generate 100 random numbers from a standard normal distribution and calculate the mean and the standard deviation of numbers. Repeat this process 10 times using the `for loop` and save the results in a dataset.

Run Code

```
# --- Number of iterations --- #
B <- 10

# --- Create an empty data.table (or data.frame for matrix) --- #
output_storage <- data.table(mean = rep(0, B), sd = rep(0, B))

# --- for loop --- #
for(i in 1:B){
  # --- Generate random numbers --- #
  random_numbers <- rnorm(100, mean = 0, sd = 1)

  # --- Calculate the mean and the standard deviation --- #
  output_storage[i, mean := mean(random_numbers)]
  output_storage[i, sd := sd(random_numbers)]
}

#or output_storage[i, `:=`(
#  mean = mean(random_numbers),
#  sd   = sd(random_numbers)
```

```
20  #)]
21
22  output_storage
```

In-class Exercise

Exercise 1

Questions

1. Using the `for loop`, calculate the sum of the first `n` numbers for `n = 1, 2, ..., 10`. For example, the sum of the first 3 numbers (n=3) is 1 + 2 + 3 = 6. Save the results in a vector object.
2. Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones (e.g. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …). Write a function that generates the first `n` numbers in the Fibonacci sequence. (You use the `for loop` function inside the function.) For example, when `n = 5`, the function should return `c(0, 1, 1, 2, 3)`. For simplicity, let's assume that $n \geq 3$ (You don't need to consider the case where n = 1 or 2).

Run Code

```
1  # You can write your code here
```

In-class Exercise

Exercise 1

Answers

1. Using the `for loop`, calculate the sum of the first `n` numbers for `n = 1, 2, ..., 10`. For example, the sum of the first 3 numbers (n=3) is 1 + 2 + 3 = 6. Save the results in a vector object.
2. Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones (e.g. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …). Write a function that generates the first `n` numbers in the Fibonacci sequence. (You use the `for loop` function inside the function.) For example, when `n = 5`, the function should return `c(0, 1, 1, 2, 3)`. For simplicity, let's assume that $n \geq 3$ (You don't need to consider the case where n = 1 or 2).

Run Code

```
#--------Part1--------#
sums <- rep(0, 10) # or sums <- numeric(10)

# Loop over n = 1 to 10
for (n in 1:10) {
  sums[n] <- sum(1:n)
}

# Print results
sums

```

```r
#========Part2========#
fibonacci <- function(n) {
  # Create a vector to store the sequence
  fib <- rep(0, n)

  # First two numbers of Fibonacci sequence
  fib[1] <- 0
  fib[2] <- 1

  # Generate the rest using a for loop
  for (i in 3:n) {
    fib[i] <- fib[i - 1] + fib[i - 2]
  }

  return(fib)
}

# Example: first 5 Fibonacci numbers
fibonacci(5)
```

In-class Exercise

Exercise 2 (optional)

Questions

NOTE: It's okay if you cannot solve this problem! I will show two approaches to solve this problem, and do speed comparison between the two approaches.

Pythagorean triples are sets of three **positive integers** that satisfy the equation , who are named after the ancient Greek mathematician Pythagoras. (a, b, c) $a^2 + b^2 = c^2$

Let's take this concept further. Suppose Pythagoras challenges you to find all possible Pythagorean triples where and are less than or equal to a given number . To address this problem, let's create an R function that will produce all such triples. a b n

1. Write a function that takes one argument n, an integer, representing the maximum value for a and b. The function should return a data frame with columns a, b, and c, containing all Pythagorean triples where and . $b \leq a \leq n$ $a^2 + b^2 = c^2$

Hints:

- Consider using nested loops to iterate through all possible values of a and b up to n.
- Use the sqrt() function to calculate the potential value of c, and check if it's an integer.
- Use the floor() function to round down the value of c.

Run Code

```
1  # You can write your code here.
```

Reference: Pythagorean Triples

In-class Exercise

Exercise 2 (optional)

Answers

Run Code

```
#/*-----------------------------*/
#' ## Approach 1 (loops)
#/*-----------------------------*/
find_pythagorean_triples <- function(n){
  # --- prepare an empty storages --- #
  triples_storage_dt <-
    data.frame(a = integer(), b = integer(), c = integer())

  for(a in 1:n){
    # for example a = 4
    for(b in 1:a){
      # for example b = 3
      c <- sqrt(a^2 + b^2)
      if(c == floor(c)){
        res_triples_dt <- data.frame(a = a, b = b, c = c)
        triples_storage_dt <- rbind(triples_storage_dt, res_triples_dt)
      }
    }
  }
  return(triples_storage_dt)
```

```r
}

# --- test --- #
find_pythagorean_triples(n = 5)


#/*-----------------------------*/
#' ## Approach 2 (Vectorized)
#/*-----------------------------*/
library(data.table)
library(dplyr)

find_pythagorean_triples_vec <- function(n){
  # for example, n = 5
  # --- create all combinations of (a, b) --- #
  abc_all_dt <-
    CJ(a = 1:n, b = 1:n) %>%
    .[, c := sqrt(a^2 + b^2)] %>%
    # find a row where c is an integer
    .[c == floor(c), ]

  return(abc_all_dt)
}

# NOTE: #CJ() is a function from data.table package that creates all combinations of two vectors. (It is similar to expand.grid() function in base R.)

# --- test --- #
```

```r
find_pythagorean_triples_vec(n = 5)

#/*--------------------------------*/
#' ## Speed Competition
#/*--------------------------------*/
system.time(find_pythagorean_triples(n = 10^4))
system.time(find_pythagorean_triples_vec(n = 10^4))

# Lesson: Vectorized operations are much faster than loops in R!
```

Check Point

Up to this point, as long as you understand the following points, you are good to go!

- You know how to use `function()` to define a simple function yourself.
- You know how to use `for loop` (i.e., syntax, which argument you need to define).
- You know that you need to prepare an empty object to save the output of the loop.

Introduction to Monte Carlo Simulations

Introduction to Monte Carlo Simulations

Analogy

- Imagine you want to know the chance of winning a dice game. Instead of trying to calculate the exact probability with formulas, you could just roll the dice thousands of times, record the outcomes, and then see how often you win.
- That’s basically what Monte Carlo Simulation does: it uses repeated random experiments to approximate answers to problems that are hard to solve analytically.
- Key ideas:
  - We model uncertainty with randomness.
  - We simulate many scenarios (sometimes thousands or millions).
  - We average the results to estimate probabilities, risks, or outcomes.

Introduction to Monte Carlo Simulations

What is it?

Monte Carlo simulation is a technique to approximate a likelihood of possible outcomes (e.g., predictions, estimates) from a model by iteratively running the model on artificially created datasets. In every iteration, the datasets are randomly sampled from the assumed data generating process, so it varies every iteration.

- The incorporation of randomness in the simulation is the key feature of the Monte Carlo simulation. *It is mimicking the randomness of real-world phenomena.*

Introduction to Monte Carlo Simulations

Monte Carlo simulation in Econometrics

In econometrics, the Monte Carlo simulation is used to evaluate the performance of a statistical procedure or the validity of theories in a realistic setting.

For example

Suppose that a researcher came up with a new estimator to estimate the coefficients of a regression model.

- An estimator (e.g, sample mean, standard error, OLS) is a function of a random variable, therefore it is also a random variable.
- A random variable has its own probability distribution.
- So, to understand the performance of the estimator (e.g., unbiasedness and efficiency), we need to examine the properties of the probability distribution of the estimator.
- We use Monte Carlo simulation to approximate the probability distribution of the estimator!

Example: Binomial Distribution

Think about the following example.

Example

- Suppose that we flip a coin $n = 10$ times and count the number of heads. Let's denote the number of heads X.
- The coin is not fair, however. The probability of getting a head is $p = Pr[heads] = 1/3$.
- Suppose that you repeat this experiment 1000 times. What is the mean and the variance of X?

This kind of experiment is modeled by the binomial distribution. According to the theory, it is predicted that

- Mean of X is $E[X] = np = 10 \times 1/3 = 3.33$
- Variance of X is $Var[X] = np(1-p) = 10 \times 1/3 \times 2/3 = 2.22$

Is it true? Let's check this using a Monte Carlo simulation!

Monte Carlo Simulation: Steps

step 1: Specify the data generating process.

- You need to pick a specific probability distribution to generate a random number.

step 2: Repeat:

- step 2.1: generate a (pseudo) random sample data based on the data generating process.
- step 2.2: get an outcome you are interested in based on the generated data.

step 3: compare your estimates with the true parameter

Demonstration: Binomial Distribution

A Single Iteration

Let's start writing code for a single iteration to get an idea of the Monte Carlo simulation process in R.

- We want to repeat this 1000 times.

Run Code

```r
# For the reproducibility purpose, set the seed.
set.seed(1843)

# === Step 1: Specify the Data Generating Process === #
# parameters
p <- 1/3 # the probability of getting a head
n <- 10 # the number of trials

# === Step 2.1: Generate Data  === #
# Here I randomly picked 1 or 0 for n = 10 times, with the probability of p = 1/3 for 1.
seq_x <- sample(c(1,0), size=n, prob=c(p, 1-p), replace=TRUE)  #replace = TRUE means sampling with replacement

# === Step 2.2: Get an Outcome You are Interested in === #
# count the number of heads
sum(seq_x)
```

Demonstration: Binomial Distribution

Multiple Iterations

Run Code

```r
# For the reproducibility purpose, set the seed.
set.seed(1843)

# === Step 1: Specify the Data Generating process === #
# parameters
p <- 1/3 # the probability of getting a head
n <- 10 # the number of trials
B <- 1000 # the number of iterations

# Don't forget to create an empty object to save the output!
X <- rep(0, B)

# === Step 2: Iteration === #
for (i in 1:B){
  # --- Step 2.1: Generate Data --- #
  # Here I randomly picked 1 or 0 for n = 10 times, with the probability of p = 1/3 for 1.
  seq_x <- sample(c(1,0), size=n, prob=c(p, 1-p), replace=TRUE)

  # --- Step 2.2: Get an Outcome You are Interested in --- #
  # count the number of heads and save the result
  X[i] <- sum(seq_x)
}
```

```
23
24  # === Step 3 === #
25  # Compute the mean of X
26  mean(X)
27
28  # Compute the variance of X
29  var(X)
```

In-class Exercise Problem

Exercise Problem: (Weak) Low of Large Number

Questions

Weak law of large number states that the sample mean converges (in probability) to the population mean as the sample size increases. In other words, the sample mean more accurately estimates the population mean as the sample size increases.

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i E[X] \xrightarrow{p} E[X]$$

Lets check this using Monte Carlo simulation! We compare the distribution of sample mean with different sample size. Let's compare two sample sizes: $n = 100$ and $n = 1000$.

Process

Repeat 1 and 2 for $B = 1000$ times.

1. Using a normal distribution with mean $\mu = 5$ and $sd = 10$, generate random numbers for $n = 100$ and $n = 1000$. e.g. `rnorm(n = 10, mean = 5, sd = 10)`.
2. Compute sample mean for each sample data, and save them.

Finally,

3. Plot histograms of the sample means obtained from the two samples.

Exercise Problem: (Weak) Low of Large Number

Answers

Run Code

```r
set.seed(1843)

# --- Parameters --- #
mu <- 5
sd <- 10

B <- 1000

# --- Create an empty object to save the output --- #
sample_mean_100 <- rep(0, B)
sample_mean_1000 <- rep(0, B)

# --- for loop --- #
for (i in 1:B){
  # --- Generate random numbers --- #
  sample_100 <- rnorm(n = 100, mean = mu, sd = sd)
  sample_1000 <- rnorm(n = 1000, mean = mu, sd = sd)

  # --- Compute sample mean --- #
  sample_mean_100[i] <- mean(sample_100)
  sample_mean_1000[i] <- mean(sample_1000)
}
```

```
23
24
25  # --- Plot --- #
26  ggplot() +
27    geom_histogram(
28      aes(x = sample_mean_100, fill = "Sample size: 100"),
29       alpha = 0.7
30    ) +
31    geom_histogram(
32      aes(x = sample_mean_1000, fill = "Sample size: 1000"),
33      alpha = 0.7
34    ) +
35    geom_vline(
36      xintercept = mu,
37      linetype = "dashed",
38    ) +
39    theme_bw()
```

Exercise Problem: Two estimators to estimate the population mean? (optional)

Questions

Suppose you're interested in estimating the unknown population mean of men's heights (i.e.,) in the US. We have randomly sampled data with the size of . Let denote the individual 's height in the sample. **How should we use the sample data to estimate the population mean?**

$n = 1000$ i

Your friends suggested two different estimators:

Estimator 1. Use the sample mean: $\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$

Estimator 2. Use only the first observation: X_1

Theoretically, both estimators are unbiased estimators (i.e., if we repeat the estimation process many times on a different sample data every time, the average of the estimates will be equal to the population mean):

$$E[\bar{X}_n] = E\left[\frac{1}{n}\sum_{i=1}^{n}\right] = \frac{1}{n}E\left[\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n} E[X_i] = \frac{1}{n} \cdot n \cdot \mu =$$

$$E[X_1] = \mu$$

Questions:

- Is it true that both estimators are correctly estimating the population mean, on average?
- Which one is more accurate in estimating the population mean?

Using Monte Carlo simulation, let's examine these questions!

Exercise Problem: Two estimators to estimate the population mean? (optional)

Hint

1. Repeat the following processes 1000 times:

step 1. Draw $n = 1000$ random numbers with known mean μ and standard deviation σ. This will be the sample data.
step 2. Get (1) the mean of the sample and (2) the value of the first observation, and save the results.

1. The previous iterations produce 1000 estimates of the population mean for estimator 1 and estimator 2, respectively. Compute the means for each estimator. Are they both close to the true population mean? Compute the variance of the estimates. Which one has a smaller variance?

If you could also visually compare the distribution of estimates from the two estimators, that would be great!

Run Code

```
1
```

Exercise Problem: Two estimators to estimate the population mean? (optional)

Answers

Run Code

```r
# --- Parameter setting --- #
B = 1000
n = 1000
mu = 170 #(cm)

# --- Prepare a storage object --- #
res_data <-
  data.table(
    x_bar = rep(0, B),
    x_single = rep(0, B)
    )

# --- Monte Carlo simulation --- #
for(i in 1:B){

  # --- Create sample data --- #
  data <- rchisq(n, df=mu)   #rchisq() is a random number generator for the chi-squared distribution.

  # --- Compute the sample mean --- #
  x_bar <- mean(data)
  # --- Get the 1st value --- #
  x_single <- data[1]
```

```r

  # --- store the results --- #
  res_data[i, "x_bar"] <- x_bar
  res_data[i, "x_single"] <- x_single
}

# === Visualization === #
# Just for convenience, transform the data to long-format  (you don't need to do this though)
vis_sampling <-
  ggplot(res_data) +
    geom_histogram(aes(x=x_bar, fill="x_bar"), alpha = 0.5)+
    geom_histogram(aes(x=x_single, fill="x_single",), alpha = 0.5)+
    labs(title="Sample size 1000")+
    theme_bw() +
    # modify x-label and legend name
    labs(x = expression (~hat(beta)), fill = "Estimator")

vis_sampling
```

Appendix

`foreach` Function

Basics

The `foreach` function is a function of the `foreach` package. It is used to iterate the same process over and over again, similar to the `for loop` function.

Basic Syntax

While there are some differences, the basic syntax of the `foreach` function is pretty much similar to the `for loop` function.

```
foreach(variable = collection_of_objects) %do% {
  the code to be executed in each iteration

  return(output)
}
```

Note

- Differences between `for loop` and `foreach` function:
  - use `=` instead of `in`.
  - You need to use `%do%` operator.
  - `foreach` function has a return value, while `for loop` does not. By default, the output is returned as a list.
- `foreach` function also supports parallel processing. (we will not cover this in this class.)

`foreach` Function

Example

Using the `for loop` and `foreach` function, let's calculate the square of the numbers from 1 to 10, respectively.

foreach

Run Code

```
# --- Load the package --- #
library(foreach)

results <- foreach(i = 1:10) %do% {
  i_square <- i^2
  return(i_square)
}

# see the output
results
```

for loop

Run Code

```
# --- Create an empty object  --- #
results <- rep(0, 10)

# --- for loop --- #
for (i in 1:10){
  results[i] <- i^2
}

# see the output
results
```

`foreach` Function

Change the Output Format

By default, `foreach` function returns each iteration's result as a list. But you can choose the format of the output by using the `.combine` argument.

- `.combine = c` combines each iteration's result as as a vector (like `c()` to create a vector).
- `.combine = rbind` combines each iteration's result by row.
- `.combine = cbind` combines each iteration's result by column.

The last two options are used when the output is a `matrix` or a `data.frame` (or `data.table`).

Example

- Try different `.combine` options in the following code.

Run Code

```
# --- Load the package --- #
library(foreach)

results <- foreach(i = 1:10, .combine = c) %do% {
  i_square <- i^2
  return(i_square)
}

# see the output
```

```
10  results
```

Overall Review

Review

Basics of R

- Coding rules & projects: file paths, working directories, using `.Rproj`.
- Data types: numeric, character, logical; handling `NA`.
- Core structures: vectors, matrices, data frames — creation, indexing, subsetting.
- Base math & objects: assignment, arithmetic, functions; saving/loading data.

Review

Data Manipulation with `data.table`

- `DT[i, j, by]` pattern and chaining.
- Row filtering and column selection; create/modify columns with `:=`.
- Grouped summaries with `by`; ordering and efficient aggregation.
- Reshaping with `melt()` and `dcast()` (wide ↔ long).

Review

Visualization with `ggplot2`

- Grammar of Graphics: `data` + `aes()` + `geom_*`.
- Common plots: scatter, line, bar, histogram, box, density; faceting.
- Aesthetics: color/size/shape, grouping, collective geoms.
- Labels/themes: axes, titles, legends; layering multiple datasets.

Review

Regression & Reporting

- Linear models with `lm()`.
- `modelsummary` for publish-ready regression tables; `datasummary` for descriptives.
- Customization: model labels, stats formatting, notes.
- Quarto reporting: render to HTML/PDF for reproducible outputs.

Review

Functions, Loops, & Monte Carlo

- User-defined functions `function()`: arguments, returns, defaults; save in .R and `source()`.
- Loops function `for()`: basics, saving outputs, multiple outputs.
- Randomness & reproducibility: `set.seed()`, `rnorm()`, `sample(..., replace=TRUE)`.

Review Exercise

Instructions

This exercise integrates all major concepts from our R course. We will use the built-in R datasets `mtcars` provided.

Setup (Run this first!)

Run Code

```
# Load required packages
library(data.table)
library(ggplot2)
library(modelsummary)

# Load and convert datasets
data(mtcars)
data(airquality)
mtcars_dt <- as.data.table(mtcars)
air_dt <- as.data.table(airquality)

# Add car names to mtcars
mtcars_dt[, car_name := rownames(mtcars)]
```

Review Exercise

Questions

Data Structures and Basic Operations

1. Show the structure of mtcars_dt
2. Calculate the mean mpg (miles per gallon) for all cars
3. Create a logical vector showing which cars have both mpg > 20 AND hp < 150
4. Extract the car names of cars with the highest mpg

Run Code

```
1  #Write your code here
```

Review Exercise

Questions

data.table Operations

1. Create a new column ‘power_ratio’ = hp/wt (horsepower per weight) and add it to mtcars_dt
2. Replace all mpg values less than 15 with NA
3. Filter mtcars_dt to show only cars with automatic transmission (am == 0)
4. Sort mtcars_dt by mpg in descending order and show top 5 cars
5. Calculate average mpg by number of cylinders (cyl)
6. For each transmission type (am), calculate: count of cars, mean mpg, mean hp

Run Code

```
1  #Write your code here
```

Review Exercise

Questions

Data Visualization

- Create a scatter plot of wt vs mpg, with:
  - Different colors for transmission type (am).
  - Different shapes for number of cylinders (cyl).
  - A smooth trend line for each transmission type.

Run Code

```
1  #Write your code here
```

Review Exercise

Questions

Regression Analysis

1. Run a regression: mpg ~ hp + wt + factor(cyl)
2. Extract the coefficient for hp from your model
3. Run a second model: mpg ~ hp + wt + factor(am), and create a comparison table using modelsummary() for both models

Run Code

```
1  #Write your code here
```

Review Exercise

Questions

Programming

1. Write a function that calculates mpg per cylinder (mpg/cyl)

Run Code

```
calc_mpg_per_cyl <- function(mpg_val, cyl_val) {
  # Your code here

}

# Test: calc_mpg_per_cyl(21, 6) should return 3.5
```

2. Use a for loop to calculate the mean mpg for cars with 4, 6, and 8 cylinders. Store results in a vector called ‘cyl_means’

Run Code

```
cyl_values <- c(4, 6, 8)
cyl_means <- rep(0, 3)

# write your code here
```

Review Exercise

Answers

Data Structures and Basic Operations

1. Show the structure of mtcars_dt
2. Calculate the mean mpg (miles per gallon) for all cars
3. Find how many cars have mpg > 20

Run Code

```
#--------part 1---------#
str(mtcars_dt) # or ?mrcars_dt

#--------part 2---------#
mean(mtcars_dt$mpg)

#--------part 3---------#
sum(mtcars_dt$mpg > 20)
```

Review Exercise

Answers

data.table Operations

1. Create a new column ‘power_ratio’ = hp/wt (horsepower per weight) and add it to mtcars_dt
2. Filter mtcars_dt to show only cars with automatic transmission (am == 0)
3. Sort mtcars_dt by mpg in descending order and show top 5 cars
4. For each transmission type (am), calculate: count of cars, mean mpg, mean hp

Run Code

```
#--------part 1---------#
mtcars_dt[, power_ratio := hp/wt]

#--------part 2---------#
mtcars_dt[am == 0]

#--------part 3---------#
mtcars_dt[order(-mpg)][1:5]

#--------part 4---------#
mtcars_dt[, .(count = .N, avg_mpg = mean(mpg, na.rm = TRUE), avg_hp = mean(hp)), by = am]
```

Review Exercise

Answers

Data Visualization

- Create a scatter plot of wt vs mpg, with:
  - Different colors for transmission type (am).
  - A smooth trend line for each transmission type.

Run Code

```
ggplot(mtcars_dt, aes(x = wt, y = mpg, color = factor(am))) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Weight vs MPG by Transmission and Cylinders",
       x = "Weight (1000 lbs)",
       y = "Miles per Gallon",
       color = "Transmission") +
  theme_bw() +
  theme(legend.position = "right")

```

Review Exercise

Answers

Regression Analysis

1. Run a regression: mpg ~ hp + wt + factor(cyl)
2. Extract the coefficient for hp from your model
3. Run a second model: mpg ~ hp + wt + factor(am), and create a comparison table using modelsummary() for both models

Run Code

```
#--------part 1---------#
model1 <- lm(mpg ~ hp + wt + factor(cyl), data = mtcars_dt)

#--------part 2---------#
coef(model1)["hp"]

#--------part 3---------#
model2 <- lm(mpg ~ hp + wt + factor(am), data = mtcars_dt)
modelsummary(list("Model 1" = model1, "Model 2" = model2))
```

Review Exercise

Answers

Programming

1. Write a function that calculates mpg per cylinder (mpg/cyl)

Run Code

```
calc_mpg_per_cyl <- function(mpg_val, cyl_val) {
  return(mpg_val / cyl_val)
}

# Test: calc_mpg_per_cyl(21, 6) should return 3.5
```

2. Use a for loop to calculate the mean mpg for cars with 4, 6, and 8 cylinders. Store results in a vector called ‘cyl_means’

Run Code

```
cyl_values <- c(4, 6, 8)
cyl_means <- rep(0, 3)

for (i in 1:3) {
  cyl_means[i] <- mean(mtcars_dt[cyl == cyl_values[i], mpg], na.rm = TRUE)
}
```