End Sem

Statistics (Besics)

- Worksheet 1
- Worksheet 2
- Worksheet 3 *(Simulating Experiments)*

Image processing

- Worksheet 4

Coding efficiently

- Worksheet 5
- Worksheet 14
- Worksheet 15
- Worksheet 16

Web Scraping

- Worksheet 6 *(rvest + tidyverse)*
- Worksheet 7 *(rvest + tidyverse)*
- Worksheet 8 *(rvest + dylyr)*

Statistics - Visualizations

- Worksheet 9 *(Data Collection - Sampling)*
- Worksheet 10 *(Descriptive Measures of Statistics)*
- Worksheet 11 *(Visualizations)*

In R, the function runif() generates random numbers from a uniform distribution.
default - min : 0, max : 1

```r
runif(5)
# Out : 0.348 0.829 0.091 0.610 0.720

runif(5, min = 1, max = 5)
# Out : 3.423793 3.498854 4.991008 1.114341 1.747155
```

Create matrix $A_{m \times n}$

```r
m <- 3
n <- 3
```

```r
A <- matrix(runif(m*n), nrow = m, ncol = n)
```

In R, an anonymous function is a function that is defined without a name. These are often used when you need a function temporarily, such as passing it as an argument to another function (like *apply, sapply, lapply, map*, etc.) without formally creating a named function.

```r
lapply()
```

Stands for: “list apply”
Input: A list (or vector)
Output: Always a list
Use case: When you want to keep the result as a list, even if each element has length 1.

```r
# List of numbers
numbers <- list(1, 2, 3, 4)
# Add 10 to each element using an anonymous function
result <- lapply(numbers, function(n) { n + 10 })
print(result)
```

```r
sapply()
```

Stands for: “simplified apply”
Input: A list (or vector)
Output: Simplifies the result:
Returns a vector if possible
Returns a matrix if the result is multi-dimensional
Falls back to a list if it cannot simplify

```r
x <- 1:5
# Apply an anonymous function to square each element
squares <- sapply(x, function(y) { y^2 })
print(squares)
```

	lapply	*sapply*
Output type	Always a list	Simplified (vector, matrix, or list)

Simplification	No	Yes
Use case	Keep result as list	Convenient for quick vector/matrix output

In R, gsub is a very useful function for pattern matching and replacement in strings. It stands for "global substitution", meaning it replaces all occurrences of a pattern in a string.

```
gsub

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
fixed = FALSE)
```

$pattern$: The text or regular expression you want to match.
$replacement$: The text you want to replace the matched pattern with.
x: The string or vector of strings to perform the replacement on.
$ignore.case$: If TRUE, ignores case while matching.
$perl$: If TRUE, uses Perl-style regular expressions.
$fixed$: If TRUE, treats pattern as a fixed string, not a regex.

```
text <- "I like apples. Apples are sweet."
gsub("apples", "oranges", text, ignore.case = TRUE)
# Output: "I like oranges. Oranges are sweet."

text <- "My phone number is 12345."
gsub("[0-9]", "", text)
# Output: "My phone number is ."
```

---

2. (b) Consider the following matrix A and let $A_1,\ A_2,\ A_3$ denote the columns of A

$$A = \begin{pmatrix} 3 & 4 & -1 \\ 1 & 5 & 2 \\ -2 & 3 & -2 \end{pmatrix} \qquad p_i = \frac{\|\ A_i\ \|}{\sum_{j=1}^{3}\ \|\ A_j\ \|}$$

Here, for $x = (x_1,\ x_2,\ \dots, x_k)$, $\|\ x\ \| = \sqrt{x_1^2 + x_2^2 + \ \cdots\ + x_k^2}$ denotes Euclidean norm.

Define a Marix :

$$A = \begin{pmatrix} 3 & 4 & -1 \\ 1 & 5 & 2 \\ -2 & 3 & -2 \end{pmatrix}$$

```r
A <- matrix(c(3, 1, -2, 4, 5, 3, -1, 2, -2), nrow = 3, ncol = 3)
```

Compute Euclidean norms :

```r
column_norms1 <- apply(A, 2, function(col) sqrt(sum(col^2)))
```

$$A_1 = \begin{pmatrix} 3 \\ 1 \\ -2 \end{pmatrix} \implies \parallel A_1 \parallel = \sqrt{3^2 + 1^2 + (-2)^2} = \sqrt{14} \approx 3.742$$

$$A_2 = \begin{pmatrix} 4 \\ 5 \\ 3 \end{pmatrix} \implies \parallel A_2 \parallel = \sqrt{4^2 + 5^2 + 3^2} = \sqrt{50} \approx 7.071$$

$$A_3 = \begin{pmatrix} -1 \\ 2 \\ -2 \end{pmatrix} \implies \parallel A_3 \parallel = \sqrt{(-1)^2 + 2^2 + (-2)^2} = \sqrt{9} \approx 3.000$$

Compute total norm sum :

$$\sum_{j=1}^{3} \parallel A_j \parallel \sqrt{14} + \sqrt{50} + \sqrt{9} \approx 3.742 + 7.071 + 3.000 = 13.813$$

Compute probabilities :

```r
probabilities <- column_norms1 / sum(column_norms1)
print(probabilities)
```

$$p_1 = \frac{\sqrt{14}}{13.813} \approx \frac{3.742}{13.813} \approx 0.271$$
$$p_2 = \frac{\sqrt{50}}{13.813} \approx \frac{7.071}{13.813} \approx 0.512$$
$$p_3 = \frac{\sqrt{9}}{13.813} = \frac{3.000}{13.813} \approx 0.217$$

```r
selected_column <- sample(1:ncol(A), size = 1, prob =
probabilities)
```

```r
print(selected_column)
```

This means column 2 has the highest Euclidean norm and therefore the highest probability of being chosen. You're more likely to pick the "larger" (in magnitude) column.

Coding efficiently

- Worksheet 5
- Worksheet 14
- Worksheet 15
- Worksheet 16

Efficient R by Selina Baldauf
→ Good for introductory learning benchmarking and profiling. ()
Benchmarking by DataCamp
→ Good for Introduction, it uses microbenchmark and my coursework has rbenchmark but overl idea is same.

What to Search ? (*this is important because i don't get anything to seach*)
Vectorization of Function in R

Worksheet 5

```r
library(rbenchmark)

benchmark(
  vec_sum = sum(1:1e6),
  loop_sum = {
    s <- 0
    for (i in 1:1e6) s <- s + i
    s
  },
  replications = 10,
  columns = c("test", "replications", "elapsed", "relative")
)
```

→ This shows the vectorized $sum()$ is ~210× faster than the loop.

Toeplitz matrix - Wikipedia
Solve 766. Toeplitz Matrix (LeetCode) if you want to play with Toeplitz (Read Diary)

```
rho_mat_loop <- function(n, rho) {
  mat <- matrix(0, nrow = n, ncol = n)  # Initialize an n x n
matrix of 0s
  for (i in 1:n) {
    for (j in 1:n) {
      mat[i,j] <- rho^(abs(i-j))  # Fill in each element with rho
to the power of |i - j|
    }
  }
  return(mat)  # Return the resulting matrix
}
```

```

```

Stirling's Theorem says that

Stirling's Theorem says that

$$\lim_{n \to \infty} \frac{n!}{\left(\frac{n^n}{e^n}\right)\sqrt{2\pi n}} = 1 \text{ ``Verify'' the above by plotting the fraction versus } n \text{ for}$$

$n = 10, \ldots 10^6$

Consider a vector $x = (x_1, \ldots, x_n)$, *and suppose we want to calculate:*

$$\frac{\log(x_i)}{\sum_{k=1}^{n} \log(x_k)} \text{ The function func below calculates the above for a given vector vec.}$$

$$\frac{\log(x_i)}{\sum_{k=1}^{n} \log(x_k)}$$

$$p_i = \frac{f(x_i)}{f(x_i) + f(y_i)} \text{ for } i = 1, 2, 3, \ldots n$$

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

Mid Semester Exam

Note : I did not performed well but I perform something as Comprated to Quiz-1,2,3.

Q1. Simulation

Story. It's 2036 and you've moved to Hyderabad. You have two social circles there:

• Circle A: 120 former classmates

• Circle B: 40 ex-colleagues

Assumptions (all encounters uniformly random from the locality):

• Locality size: 200,000 people

• Daily encounters: 500 people (independent from day to day)

• Meeting anyone from either Circle A or Circle B ends the simulation

Task. Write an R function $cityDejaVu()$ with no inputs that simulates day by day and until you first meet someone from Circle A or B and returns:

1. p_day, the probability that, on a single day, you meet at least one person from Circle A or B.

2. days until the first "hit" (each day is a Bernoulli trial with success prob p_day).

Note :

Initialize Parameters

$N = 200000$ (total population)

$A = 120$ (size of Circle A)

$B = 40$ (size of Circle B)

$E = 500$ (number of encounters per day)

```
cityDejaVu <- function() {
  N <- 200000L
  A <- 120L
  B <- 40L
  E <- 500L
}
```

1. Compute probability that one encounter is a hit:

$$p_{one} = \frac{n(A) + n(B)}{n(S)}$$

```r
p_one <- (A + B) / N
```

2. Compute probability that at least one hit occurs in E encounters using the complement rule:

$$p_{day} = 1 - (1 - p_{one})^E$$

P(atleast 1 hit in a day) = 1-P(no hit in 500 encounters)

The probability that one encounter is not a hit: $P(no\ hit\ in\ one\ encounter\ = (1 - p_1)^1$

The probability that 500 encounter is not a hit: $P(no\ hit\ in\ 500\ encounter) = (1 - p_1)^{500}$

```r
p_day <- 1 - (1 - p_one)^E
```

Simulate day-by-day until first hit

```r
days <- 0L

# effective in terms of speed and memory
repeat {
  days <- days + 1L
  if (runif(1) < p_day) break #stop (hit occurs)
}

# Slower : Using Sample
repeat {
  days <- days + 1L
  hit <- any(sample(N, E, replace = TRUE) <= A+B)
  if (hit) break
}

# Moderate : While Loop
hit  <- FALSE
while (!hit) {
  days <- days + 1L
  hit <- any(sample(N, E, replace = TRUE) <= A+B)
}
```

Return results

```r
list(days = as.integer(days), p_day = as.numeric(p_day))
```

run the simulation multiple times

```r
# Print p_day
print(cityDejaVu()$p_day)

set.seed(12345)
# run cityDejaVu 200 times to compute the mean of "days";
# save it in mean_days and print it
n_runs <- 200L
days <- replicate(n_runs, cityDejaVu()$days)
print(mean(days))
```

```

```

End Semester Exam

End Sem 2024
[Monte Carlo integration - Wikipedia]
→ Read this, you'll get basic Idea about problem.

By [Garg University Guy]
→ Must watch, He did all this in excel. means you'll understand what going here.

[Monte Carlo Simulation - Explained]
→ Should be Good

By [Ritvik]
→ I didn't watched yet but i think its good for building intuition.

[Calculating Circle Area the Monte Carlo Way]
[Monte Carlo Estimation of π]
→ Read it if you want. I didn't.

Search Guide :
area of a circle using Monte Carlo