Homework 1

Zelin Chen

Table of contents

| | ~ | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|-----|--|--|--|--|---|--|--|--|--|--|--|--|----|--|--|--|--|---|--|---|--|---|--|---|---|--|--|---|
| Appendix | | | | | | | | | | | | | | 10 | | | | | | | | | | | | | | | |
| Question | n 3 | | | | | • | | | | | | | | | | | | | - | | • | | • | | • | • | | | 4 |
| Question | n 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 |
| Question | n 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 |

Link to the Github repository

Due: Fri, Jan 26, 2024 @ 11:59pm

Please read the instructions carefully before submitting your assignment.

- 1. This assignment requires you to:
 - Upload your Quarto markdown files to a git repository
 - Upload a PDF file on Canvas
- 2. Don't collapse any code cells before submitting.
- 3. Remember to make sure all your code output is rendered properly before uploading your submission.

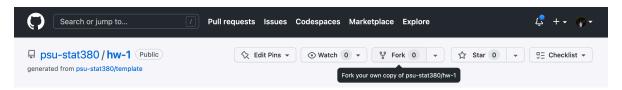
Please add your name to the the author information in the frontmatter before submitting your assignment.

Question 1



In this question, we will walk through the process of *forking* a git repository and submitting a *pull request*.

1. Navigate to the Github repository here and fork it by clicking on the icon in the top right



Provide a sensible name for your forked repository when prompted.

2. Clone your Github repository on your local machine

```
$ git clone <<insert your repository url here>>
$ cd hw-1
```

3. In order to activate the R environment for the homework, make sure you have renv installed beforehand. To activate the renv environment for this assignment, open an instance of the R console from within the directory and type

```
renv::activate()
```

Follow the instrutions in order to make sure that renv is configured correctly.

- 4. Work on the reminaing part of this assignment as a .gmd file.
 - Create a PDF and HTML file for your output by modifying the YAML frontmatter for the Quarto .qmd document
- 5. When you're done working on your assignment, push the changes to your github repository.
- 6. Navigate to the original Github repository here and submit a pull request linking to your repository.

Remember to include your name in the pull request information!

If you're stuck at any step along the way, you can refer to the official Github docs here

Question 2



9 30 points

Consider the following vector

```
my_vec <- c(
    "+0.07",
    "-0.07",
    "+0.25",
    "-0.84",
    "+0.32",
    "-0.24",
    "-0.97",
    "-0.36",
    "+1.76",
    "-0.36"
)
```

For the following questions, provide your answers in a code cell.

- 1. What data type does the vector contain?
- 2. Create two new vectors called my_vec_double and my_vec_int which converts my_vec to Double & Integer types, respectively,
- 3. Create a new vector my_vec_bool which comprises of:
 - TRUE if an element in my_vec_double is ≤ 0
 - FALSE if an element in my_vec_double is ≥ 0

How many elements of my_vec_double are greater than zero?

4. Sort the values of my_vec_double in ascending order.

```
my_vec_double <- as.double(my_vec)</pre>
my_vec_int <- as.integer(my_vec)</pre>
my_vec_bool <- my_vec_double <= 0</pre>
```

```
my_vec_sorted <- sort(my_vec_double)</pre>
 my_vec_double
[1] 0.07 -0.07 0.25 -0.84 0.32 -0.24 -0.97 -0.36 1.76 -0.36
 my_vec_int
```

[1] 0 0 0 0 0 0 0 0 1 0

my_vec_bool

[1] FALSE TRUE FALSE TRUE FALSE TRUE TRUE TRUE FALSE TRUE

my_vec_sorted

[1] -0.97 -0.84 -0.36 -0.36 -0.24 -0.07 0.07 0.25 0.32 1.76

Question 3



9 50 points

In this question we will get a better understanding of how R handles large data structures in memory.

1. Provide R code to construct the following matrices:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & \dots & 100 \\ 1 & 4 & 9 & 16 & 25 & \dots & 10000 \end{bmatrix}$$

```
matrix_1 <- matrix(1:9, nrow=3, ncol=3, byrow = T)</pre>
  matrix_1
     [,1] [,2] [,3]
[1,]
        1
             2
[2,]
        4
             5
                  6
[3,]
        7
             8
                  9
  matrix_2 < -matrix(c(1:100,(1:100)^2), nrow=2, ncol=100, byrow = T)
  matrix_2
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]
             2
                  3
                       4
                                      7
        1
                            5
                                 6
                                            8
                                                 9
                                                      10
                                                            11
                                                                  12
                                                                        13
[2,]
                  9
                      16
                           25
                                36
                                     49
                                           64
                                                81
                                                     100
                                                           121
                                                                 144
                                                                       169
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
                                      20
                                            21
                                                   22
                                                         23
                                                                     25
[1,]
        15
              16
                    17
                          18
                                19
                                                               24
                                                                           26
[2,]
       225
             256
                   289
                         324
                               361
                                     400
                                            441
                                                  484
                                                        529
                                                              576
                                                                    625
                                                                           676
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
                    29
[1,]
        27
              28
                          30
                                31
                                      32
                                             33
                                                   34
                                                         35
                                                               36
                                                                     37
                                    1024 1089 1156
                                                      1225
[2,]
       729
             784
                   841
                         900
                               961
                                                            1296 1369
                                                                        1444
     [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
[1,]
              40
                    41
                          42
                                43
                                      44
                                            45
                                                   46
                                                         47
                                                               48
                                                                     49
[2,] 1521
          1600 1681 1764 1849
                                    1936 2025 2116 2209
                                                             2304
                                                                  2401 2500
     [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
[1,]
        51
              52
                    53
                          54
                                55
                                      56
                                            57
                                                   58
                                                         59
                                                               60
                                                                     61
                                   3136
[2,] 2601
           2704 2809 2916 3025
                                         3249
                                                3364
                                                      3481
                                                            3600
                                                                  3721
                                                                        3844
     [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
[1,]
        63
              64
                    65
                          66
                                67
                                      68
                                             69
                                                   70
                                                         71
                                                               72
                                                                     73
           4096 4225 4356 4489
[2,]
    3969
                                    4624 4761 4900 5041
                                                            5184
                                                                  5329
                                                                         5476
     [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
              76
[1,]
        75
                    77
                          78
                                79
                                      80
                                             81
                                                   82
                                                         83
                                                               84
                                                                     85
[2,]
    5625
           5776 5929 6084 6241
                                    6400 6561 6724
                                                      6889
                                                             7056 7225
                                                                         7396
     [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
[1,]
                                      92
                                                                     97
              88
                    89
                          90
                                91
                                             93
                                                   94
                                                         95
                                                               96
[2,]
     7569
           7744 7921 8100 8281 8464 8649
                                                 8836
                                                      9025
                                                             9216 9409
                                                                         9604
     [,99] [,100]
[1,]
        99
              100
[2,]
     9801
           10000
```

Recall the discussion in class on how R fills in matrices

In the next part, we will discover how knowledge of the way in which a matrix is stored in memory can inform better code choices. To this end, the following function takes an input nand creates an $n \times n$ matrix with random entries.

```
generate_matrix <- function(n){</pre>
    return(
         matrix(
             rnorm(n^2),
             nrow=n
    )
}
```

For example:

```
generate_matrix(4)
```

```
[,1]
                      [,2]
                                 [,3]
                                             [,4]
     0.9961008 1.1763881 0.2201596
[1,]
                                      0.32938052
[2,] -1.0252302 -0.6891301 -0.3862098 -0.58465137
[3,] -0.8971821 -0.1995638 -2.5931500 -0.09730648
[4,] -1.0462294 0.9362134 -0.9067132 -1.42501691
```

Let M be a fixed 50×50 matrix

```
M <- generate_matrix(50)</pre>
mean(M)
```

[1] -0.005699159

2. Write a function row_wise_scan which scans the entries of M one row after another and outputs the number of elements whose value is ≥ 0 . You can use the following starter code

```
row_wise_scan <- function(x){</pre>
     n \leftarrow nrow(x)
     m \leftarrow ncol(x)
```

```
# Insert your code here
count <- 0
for(i in 1:n){
    for(j in 1:m){
        if(x[i,j] >= 0){
            count <- count + 1
        }
    }
    return(count)
}</pre>
```

3. Similarly, write a function col_wise_scan which does exactly the same thing but scans the entries of M one column after another

```
col_wise_scan <- function(x){
    n <- nrow(x)
    m <- ncol(x)

# Insert your code here
    count <- 0

for(i in 1:m){
        for(j in 1:n){
            if(x[j,i] >= 0){
                count <- count + 1
            }
        }
    }
    return(count)
}</pre>
```

You can check if your code is doing what it's supposed to using the function here¹

```
sapply(1:100, function(i) {
    x <- generate_matrix(100)
    row_wise_scan(x) == col_wise_scan(x)
})</pre>
```

¹If your code is right, the following code should evaluate to be TRUE

4. Between col_wise_scan and row_wise_scan, which function do you expect to take shorter to run? Why?

col_wise_scan, since R search the values in matrix column by column.

5. Write a function time_scan which takes in a method f and a matrix M and outputs the amount of time taken to run f(M)

```
time_scan <- function(f, M){
   initial_time <- Sys.time() # Write your code here
   f(M)
   final_time <- Sys.time() # Write your code here

   total_time_taken <- final_time - initial_time
   return(total_time_taken)
}</pre>
```

Provide your output to

```
list(
    row_wise_time = time_scan(row_wise_scan, M),
    col_wise_time = time_scan(row_wise_scan, M)
)
```

```
$row_wise_time
Time difference of 0.0001249313 secs
```

```
$col_wise_time
Time difference of 0.0001199245 secs
```

Which took longer to run? Looks same

- 6. Repeat this experiment now when:
 - M is a 100×100 matrix
 - M is a 1000×1000 matrix
 - M is a 5000×5000 matrix

```
M_100 <- generate_matrix(100)</pre>
  M_1000 <- generate_matrix(1000)</pre>
  M_5000 <- generate_matrix(5000)
  result_100 <- list(
      row_wise_time_100 = time_scan(row_wise_scan, M_100),
      col_wise_time_100 = time_scan(col_wise_scan, M_100)
  result_1000 <- list(
      row_wise_time_1000 = time_scan(row_wise_scan, M_1000),
      col_wise_time_1000 = time_scan(col_wise_scan, M_1000)
  )
  result_5000 <- list(</pre>
      row_wise_time_5000 = time_scan(row_wise_scan, M_5000),
      col_wise_time_5000 = time_scan(col_wise_scan, M_5000)
  )
  result_100
$row_wise_time_100
Time difference of 0.0006520748 secs
$col_wise_time_100
Time difference of 0.0006010532 secs
  result_1000
$row_wise_time_1000
Time difference of 0.07779813 secs
$col_wise_time_1000
Time difference of 0.06186914 secs
  result_5000
$row_wise_time_5000
Time difference of 2.336965 secs
```

```
$col_wise_time_5000
Time difference of 1.119328 secs
```

What can you conclude? Column wise scan is faster than row wise scan.

Appendix

Print your R session information using the following command

```
sessionInfo()
```

R version 4.3.1 (2023-06-16 ucrt)

Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 11 x64 (build 22621)

Matrix products: default

locale:

- [1] LC_COLLATE=Chinese (Simplified)_China.utf8
- [2] LC_CTYPE=Chinese (Simplified)_China.utf8
- [3] LC_MONETARY=Chinese (Simplified)_China.utf8
- [4] LC_NUMERIC=C
- [5] LC_TIME=Chinese (Simplified)_China.utf8

time zone: America/New_York
tzcode source: internal

attached base packages:

[1] stats graphics grDevices datasets utils methods base

loaded via a namespace (and not attached):

| [1] | compiler_4.3.1 | fastmap_1.1.1 | cli_3.6.2 | htmltools_0.5.7 |
|-----|----------------|-------------------|------------|-----------------|
| [5] | tools_4.3.1 | rstudioapi_0.15.0 | yaml_2.3.8 | rmarkdown_2.25 |
| [9] | knitr_1.45 | jsonlite_1.8.8 | xfun_0.41 | digest_0.6.34 |

[13] rlang_1.1.3 renv_1.0.3 evaluate_0.23