Homework 1

Victoria Ritorto

Table of contents

Link to the Github repository

> ❗ Due: Sun, Jan 29, 2023 @ 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

> 💡 20 points

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
```

    Alternatively, you can use Github codespaces to get started from your repository directly.

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 `.qmd` 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

> 💡 30 points

Consider the following vector

```r
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?

```r
# The vector contains double and integer.
```

2. Create two new vectors called `my_vec_double` and `my_vec_int` which converts `my_vec` to Double & Integer types, respectively,

```r
my_vec_double <- as.numeric(my_vec)
my_vec_double
```

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

```r
typeof(my_vec_double)
```

```
[1] "double"
```

```
my_vec_int <- as.integer(my_vec)
my_vec_int
```

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

```
typeof(my_vec_int)
```

```
[1] "integer"
```

3. Create a new vector `my_vec_bool` which comprises of:

   - TRUEif 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?

```
my_vec_bool <- c()
count = 0

for (i in my_vec_double) {
    if (i > 0) {
        my_vec_bool <- append(my_vec_bool, "TRUE", after = length(my_vec_bool))
        count <- count + 1
    }
    else{
        my_vec_bool <- append(my_vec_bool, "FALSE", after = length(my_vec_bool))
    }
}
my_vec_bool
```

```
 [1] "TRUE"  "FALSE" "TRUE"  "FALSE" "TRUE"  "FALSE" "FALSE" "FALSE" "TRUE"
[10] "FALSE"
```

```
# 4 elements are greater than 0.
```

4. Sort the values of `my_vec_double` in ascending order.

```
sort(my_vec_double, decreasing = FALSE)
```

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

Question 3

💡 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:

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

> ⚠️ Tip
>
> Recall the discussion in class on how R fills in matrices

```
x <- matrix(
    c(1, 2, 3, 4, 5, 6, 7, 8, 9),
    nrow=3,
    byrow = TRUE
)
x
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
y <- matrix(seq(1,100),nrow=1)
z <- matrix(seq(1,100),nrow=1)
# 1 to 1000, xn=n^2
```

```r
a = rbind(y, z)
a
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    2    3    4    5    6    7    8    9    10    11    12    13    14
[2,]    1    2    3    4    5    6    7    8    9    10    11    12    13    14
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]    15    16    17    18    19    20    21    22    23    24    25    26
[2,]    15    16    17    18    19    20    21    22    23    24    25    26
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
[1,]    27    28    29    30    31    32    33    34    35    36    37    38
[2,]    27    28    29    30    31    32    33    34    35    36    37    38
     [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
[1,]    39    40    41    42    43    44    45    46    47    48    49    50
[2,]    39    40    41    42    43    44    45    46    47    48    49    50
     [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
[1,]    51    52    53    54    55    56    57    58    59    60    61    62
[2,]    51    52    53    54    55    56    57    58    59    60    61    62
     [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
[1,]    63    64    65    66    67    68    69    70    71    72    73    74
[2,]    63    64    65    66    67    68    69    70    71    72    73    74
     [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
[1,]    75    76    77    78    79    80    81    82    83    84    85    86
[2,]    75    76    77    78    79    80    81    82    83    84    85    86
     [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
[1,]    87    88    89    90    91    92    93    94    95    96    97    98
[2,]    87    88    89    90    91    92    93    94    95    96    97    98
     [,99] [,100]
[1,]    99    100
[2,]    99    100
```

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 n and creates an $n \times n$ matrix with random entries.

```r
generate_matrix <- function(n){
    return(
        matrix(
            rnorm(n^2),
            nrow=n
        )
```

```
    )
}
```

For example:

```
generate_matrix(4)
```

```
            [,1]        [,2]        [,3]        [,4]
[1,]   0.1256626 -0.04372229 -2.0123214 -0.3648940
[2,]   0.8497608  2.08678901  0.6629387 -1.9677012
[3,] -0.8239795  0.22389987 -0.3522940  0.6865467
[4,] -0.6125858 -1.54080721  0.7332954  1.4638881
```

Let M be a fixed 50×50 matrix

```
M <- generate_matrix(50)
mean(M)
```

```
[1] 0.01928925
```

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){
    n <- nrow(x)
    m <- ncol(x)

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

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){
    count <- 0

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

You can check if your code is doing what it's supposed to using the function here[1]

4. Between `col_wise_scan` and `row_wise_scan`, which function do you expect to take shorter to run? Why? ::: {.cell}

```
# I expect col_wise_scan to take shorter to run because the data for each column is sorted
```

:::

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
    f(M)
    final_time <- Sys.time

    total_time_taken <- final_time - initial_time
    return(total_time_taken)
}
```

Provide your output to

---

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

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

Which took longer to run?

6. Repeat this experiment now when:

- `M` is a 100×100 matrix
- `M` is a 1000×1000 matrix
- `M` is a 5000×5000 matrix

What can you conclude? ::: {.cell}

```
# I can conclude the larger matrix takes longer to run.
```

:::

Appendix

Print your `R` session information using the following command

```
sessionInfo()
```

```
R version 4.2.2 Patched (2022-11-10 r83330)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.5 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
```

[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats   graphics grDevices datasets utils   methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.2 fastmap_1.1.0   cli_3.6.0   htmltools_0.5.4
 [5] tools_4.2.2   yaml_2.3.7   rmarkdown_2.20 knitr_1.42
 [9] xfun_0.36   digest_0.6.31   jsonlite_1.8.4 rlang_1.0.6
[13] renv_0.16.0-53 evaluate_0.20

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