Homework 1

Robert Chappell{style='background-color: yellow;'}

Table of contents

Question 2 1
Question 3 2

Question 2

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

1.

```
class(my_vec)
```

```
[1] "character"
```

2.

```r
#Makes my-vec into double
my_vec_double <- as.double(my_vec)

#Makes my_vec into a int
my_vec_int <- as.integer(my_vec)
```

3.

```r
#Will use ifelse to make Bool vector
my_vec_bool <- ifelse(my_vec_double > 0, FALSE, TRUE)
greaterthanzero = length(my_vec_bool) - sum(my_vec_bool)
greaterthanzero
```

```
[1] 4
```

4.

```r
sort(my_vec_double)
```

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

Question 3

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

```r
#Test of function
M <- generate_matrix(50)
mean(M)
```

```
[1] -0.006370336
```

2.

```
row_wise_scan <- function(x){
    n <- nrow(x)
    m <- 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)
}
```

```
row_wise_scan(M)
```

```
[1] 1240
```

3.

```
col_wise_scan <- function(x){
    count <- 0
    n <- nrow(x)
    m <- ncol(x)
    for(i in 1:m){
      for(j in 1:n){
        if(x[i,j] >= 0){
                count <- count + 1
            }
        }
    }

    return(count)
}
```

```r
col_wise_scan(M)
```

```
[1] 1240
```

```r
#Check to see if functions work
library(tidyr)
sapply(1:100, function(i) {
    x <- generate_matrix(100)
    row_wise_scan(x) == col_wise_scan(x)
}) %>% sum == 100
```

```
[1] TRUE
```

4.

I would expect the functions to take the same amount of time. With both the columns and rows equal in length. If one was longer than the other I would expect the function that looks at the longer option first to run longer.

5.

```r
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)
}
```

```r
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 0.0004479885 secs

$col_wise_time
Time difference of 0.0002920628 secs
```

Row_wise_time took slightly longer to run.

6.

```
#M is 100x100
M <- generate_matrix(100)
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 0.0008230209 secs

$col_wise_time
Time difference of 0.0008819103 secs
```

```
#M is 1000x1000
M <- generate_matrix(1000)
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 0.08374906 secs

$col_wise_time
Time difference of 0.08512306 secs
```

```
#M is 5000x5000
M <- generate_matrix(5000)
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 2.815115 secs
```

```
$col_wise_time
Time difference of 2.77276 secs
```

After running the experiment on different sizes of M. I can conclude that checking the row first will take more time than checking the column first.