HW2

2023-09-10

1.

(a)

```
X_50 <- seq(0, 10, length.out = 50)
X_100 <- seq(0, 10, length.out = 100)
X_500 <- seq(0, 10, length.out = 500)
```

(b)

```
dataset_size <- 10^5


dataset_matrix50 <-  matrix(nrow = 10^5, ncol = 50)
set.seed(1)
for (n in 1:10^5){
  lambda <- exp(1 + 0.25 * X_50)
  Y <- rpois(length(X_50), lambda)
  dataset_matrix50[n, ] <- Y


}

norm(dataset_matrix50)
```

```
## [1] 3312427
```

```
dataset_size <- 10^5


dataset_matrix100 <-  matrix(nrow = 10^5, ncol = 100)
set.seed(2)
for (n in 1:10^5){
  lambda <- exp(1 + 0.25 * X_100)
  Y <- rpois(length(X_100), lambda)
  dataset_matrix100[n, ] <- Y


}

norm(dataset_matrix100)
```

```
## [1] 3308410
```

```
dataset_size <- 10^5


dataset_matrix500 <-  matrix(nrow = 10^5, ncol = 500)
set.seed(3)
for (n in 1:10^5){
  lambda <- exp(1 + 0.25 * X_500)
  Y <- rpois(length(X_500), lambda)
  dataset_matrix500[n, ] <- Y


}

norm(dataset_matrix500)
```

```
## [1] 3312357
```

```
sample_list <- list(dataset_matrix50,dataset_matrix100,dataset_matrix500)
```

(c). Here is the for loop and lapply comparison of n=50:

```
system.time(for (i in 1:10^5) model <- glm(sample_list[[1]][i,] ~ X_50, family = "poisson"))
```

```
##    user  system elapsed
##   76.90    0.38   77.28
```

```
data_frame <- as.data.frame(sample_list[[1]])
data_frame$X_50 <- X_50
```

```
mycolname <- colnames(data_frame)
mycolname <- mycolname[-51]
```

```
system.time(invisible(lapply(mycolname,
       function(x) glm(data_frame[[x]] ~ data_frame[["X_50"]], family="poisson" ))))
```

```
##    user  system elapsed
##    7.67    1.98    9.66
```

Here is the for loop and lapply comparison of n=100:

```
system.time(for (i in 1:10^5) model <- glm(sample_list[[2]][i,] ~ X_100, family = "poisson"))
```

```
##    user  system elapsed
##   83.13    0.41   83.53
```

```
data_frame_100 <- as.data.frame(sample_list[[2]])
data_frame_100$X_100 <- X_100
mycolname_100<- colnames(data_frame_100)
mycolname_100 <- mycolname_100[-101]
```

```
system.time(invisible(lapply(mycolname_100,
       function(x) glm(data_frame_100[[x]] ~ data_frame_100[["X_100"]], family="poisson" ))))
```

```
##    user  system elapsed
##   15.14    1.65   16.80
```

Here is the for loop and lapply comparison of n=500:

```
system.time(for (i in 1:10^5) model <- glm(sample_list[[3]][i,] ~ X_500, family = "poisson"))
```

```
##    user  system elapsed
##  133.22    0.57  133.81
```

```
data_frame_500 <- as.data.frame(sample_list[[3]])
data_frame_500$X_500<- X_500
mycolname_500<- colnames(data_frame_500)
mycolname_500 <- mycolname_500[-501]
```

```
system.time(invisible(lapply(mycolname_500,
       function(x) glm(data_frame_500[[x]] ~ data_frame_500[["X_500"]], family="poisson" ))))
```

```
##    user  system elapsed
##   78.58    8.25   86.83
```

Here I can notice that the lapply function is always efficient than the for loop. In my experiment, the lapply always show a better result.

2.

(a)

```
whether_in_mandelbrot   <- function(c){
    z<- 0


   for (i in 1:100) {
    z <- z^2 + c
    if (Mod(z) >= 10^6) {
      #print(z/10^6)
      return(FALSE)

    }

  }
  #print(z/10^6)
  return(TRUE)


}
```

(b)

```r
v <- seq(-1, 1, length.out = 2001)
```

(c)

```r
tf_matrix <- matrix(NA,2001,2001)
for (i in 1:2001) {
  for (j in 1:2001) {
    c <- complex(real = v[i], imaginary = v[j])
    tf_matrix[i, j] <- whether_in_mandelbrot(c)
  }
}
```

(d)

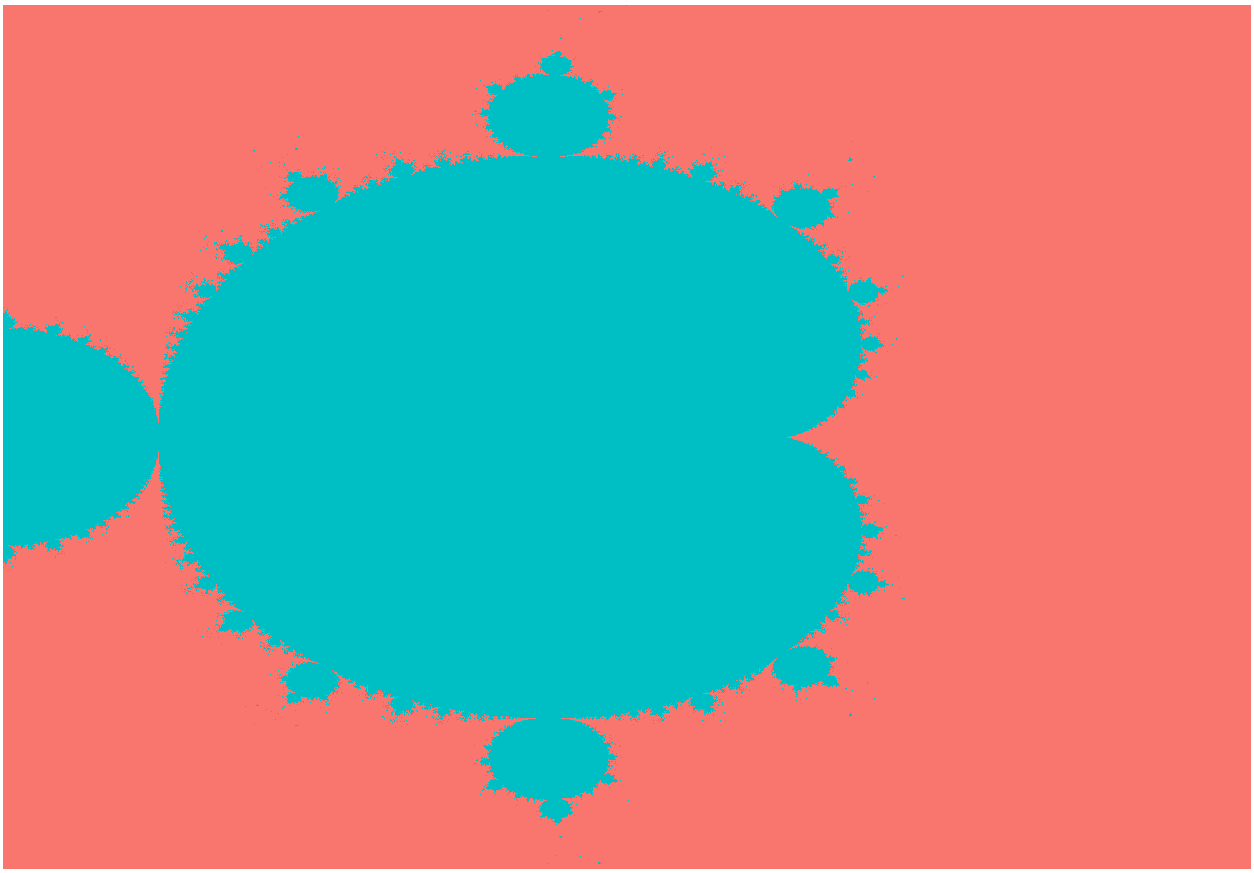
```r
library(ggplot2)
```

```r
row_names <- v
col_names <- v
```

```r
dimnames(tf_matrix) <- list(row_names, col_names)
```

```r
density_df <- as.data.frame(as.table(tf_matrix))
```

```r
ggplot(density_df, aes(x = Var1, y = Var2, fill = Freq)) +
  geom_tile() +  labs(x = NULL, y = NULL) +   theme_void()+ theme(legend.position = "none")    # Remove
```



3.

(a)

```
A <- list(-pi, 0, 1 + 1i, "statistics", TRUE, NA, -Inf)
```

```
myfunction_3 <- function(x) {

  value <- (1 - (1 - sin(x))^2) / factorial(x)

  return(value)
}
```

```
#for (i in 1:7){
  #print(myfunction_3(A[[i]]))

#}
```

The above code will gibes the result:

[1] 1.391734e-16

[1] 0

Error in gamma(x + 1) : unimplemented complex function

At the complex value, the calculations terminate.

(b)

```
for (i in 1:7){
  print(try(myfunction_3(A[[i]])))

  print(paste("This is the",i,"'s loop"))

}
```

```
## [1] 1.391734e-16
## [1] "This is the 1 's loop"
## [1] 0
## [1] "This is the 2 's loop"
## Error in gamma(x + 1) : unimplemented complex function
## [1] "Error in gamma(x + 1) : unimplemented complex function\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in gamma(x + 1): unimplemented complex function>
## [1] "This is the 3 's loop"
## Error in sin(x) : non-numeric argument to mathematical function
## [1] "Error in sin(x) : non-numeric argument to mathematical function\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in sin(x): non-numeric argument to mathematical function>
```

```
## [1] "This is the 4 's loop"
## [1] 0.9748686
## [1] "This is the 5 's loop"
## [1] NA
## [1] "This is the 6 's loop"

## Warning in sin(x): NaNs produced

## Warning in gamma(x + 1): NaNs produced

## [1] NaN
## [1] "This is the 7 's loop"
```

Based on the result it shows, The following element will t return an error message { 1+1, "statistics", NA, -∞ }.

(c)

```
out <- rep(0,7)
for (i in 1:7){
    out[i] <- tryCatch(
        myfunction_3(A[[i]])
        ,
        error = function(e) {
    NaN
  })

}
```

```
## Warning in sin(x): NaNs produced

## Warning in gamma(x + 1): NaNs produced
```

```
out
```

```
## [1] 1.391734e-16 0.000000e+00          NaN          NaN 9.748686e-01
## [6]           NA          NaN
```

4.

(a)

```
my_cv <- function(x){
  my_mean <- mean(x)
  my_sd <- sd(x)

  return(my_sd/my_mean)
}
```

(b)

```r
library(Rcpp)
```

```r
#meanC
sourceCpp("D:/Desktop/705/HW2/meanC.cpp")
```

```r
myCV(c(1,2,3,4))
```

```
## [1] 0.5163978
```

```r
sd(c(1,2,3,4))
```

```
## [1] 1.290994
```

```r
my_cv(c(1,2,3,4))
```

```
## [1] 0.5163978
```

```r
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.3.1
```

```r
set.seed(1)
x <- rnorm(5*10^4)

microbenchmark(
  myCV(x),
  my_cv(x),
  times = 1000

)
```

```
## Unit: microseconds
##       expr   min    lq     mean median    uq    max neval
##   myCV(x) 131.4 131.9 135.2657  134.5 134.9  710.5  1000
##  my_cv(x) 200.6 203.8 210.6611  207.7 208.7 1908.2  1000
```

```r
set.seed(2)
x <- rnorm(5*10^5)

microbenchmark(
  myCV(x),
  my_cv(x),
  times = 1000

)
```

```
## Unit: milliseconds
##       expr    min      lq     mean median      uq    max neval
##   myCV(x) 1.3093 1.31560 1.346871 1.3395 1.35185 1.9730  1000
##  my_cv(x) 1.9314 1.95285 1.992334 1.9780 2.00510 2.3759  1000
```

```r
set.seed(3)
x <- rnorm(5*10^6)

microbenchmark(
  myCV(x),
  my_cv(x),
  times = 1000

)
```

```
## Unit: milliseconds
##       expr     min       lq     mean   median       uq     max neval
##    myCV(x) 13.5680 13.84305 14.07768 13.99855 14.20915 15.8310  1000
##   my_cv(x) 21.5325 21.91730 22.27787 22.12555 22.49930 25.0327  1000
```

The time is shown above. I notice that, when the sample size is 5×10^4, the C++ function use only 2/3 of the total time of R. In the sample size of 5×10^5 and 5×10^6, it also show the same performance.