STAT 545: HW 2 Report

Problem 1

We have 2 files. They are mainly based on the provided code. I just had to seperate the code into two seperate files. The code is here:

file1.R

```
source("file2.R")

# Read data
score <- read.csv("scores.csv", header = TRUE)

# Just calculate the grades
final_results <- calculate_letter_grades(score)

# Check our work
# print(final_results)

# Write to a CSV
write.csv(final_results, "final_letter_grades.csv", row.names = FALSE)
```

file2.R

```
# We create our function, basically copied and pasted

calculate_letter_grades <- function(score) {
  #unsubmitted hw/quiz/exam/project counts as 0 score
  score[is.na(score)] <- 0

  #Calculate component-wise scores according to the syllabus
  hw.component = (score$HW_maxscore_100/100)*30
  midterm.component = (score$Midterm_maxscore_50/50)*25
  final.component = (score$Final_maxscore_50/50)*25
  project.component=(score$Project_maxscore_100/100)*20

  #Calculate total score
  total = cbind(hw.component, midterm.component, final.component, project.component)
  totalscore = apply(total,1,sum)

  #Assign grades
  grades = rep(NA,nrow(score))

  for ( i in 1:nrow(score)){
    if (totalscore[i]< 59) grades[i] = "F"
    else if ((totalscore[i]>= 60) & (totalscore [i] <70)) grades[i] = "D"
```

```
    else if ((totalscore[i]>= 70) & (totalscore [i] < 80)) grades[i] = "C"
    else if ((totalscore[i]>= 80) & (totalscore [i] < 90)) grades[i] = "B"
    else if ((totalscore[i]>= 90) & (totalscore [i] <100)) grades[i] = "A"
    else grades[i] = "ERROR"
  }

  finalscore = cbind(as.character(score[,1]), totalscore, grades)
  finalscore
}
```

We can test our code and look at the results:

```
# Read data
score <- read.csv("scores.csv", header = TRUE)

# Just calculate the grades
final_results <- calculate_letter_grades(score)

# Check our work
print(final_results)
```

```
##                totalscore grades
## [1,] "Alice"   "85"       "B"
## [2,] "Bob"     "90.5"     "A"
## [3,] "Charlie" "47.6"     "F"
## [4,] "Dave"    "80.4"     "B"
## [5,] "Erin"    "91.8"     "A"
```

Problem 2

Professor's code

profexample.R

```
## Author: Anindya Bhadra
## STAT 545
## Example of code profiling and timing: for loop vs. apply()
##Clear workspace and load required libraries
rm(list=ls())
library(MASS)

pow=function(x,lambda){x^lambda}

## Write a function called "rowstat" using for loop to compute row mean and row sd of a matrix
rowstat=function(x){
    rr=c(NA, length=nrow(x))
    rrsq=c(NA, length=nrow(x))
    for (i in 1:nrow(x))
        {rr[i] = sum(x[i,])
         rrsq[i]=sum(pow(x[i,],2))
    }
    list(mean=rr/ncol(x),sd=sqrt((rrsq - ncol(x)*(pow(rr/ncol(x),2)))/(ncol(x)-1)))
```

```
}

Sigma <- matrix(c(10,3,3,2),2,2)

## Generate 10,000 samples from a bivariate Normal distribution with mean 0 and covariance matrix Sigma
xx=mvrnorm(n=10000,c(0,0), Sigma)

## Call the function "rowstat"
pp=rowstat(xx)

## Do the same operations using apply
llm=apply(xx,1,mean)
lls=apply(xx,1,sd)

## Check that they give the same result
head (pp$mean)
```

```
##                length
## -5.0026673  1.9820776 -0.3263347 -0.9978646  0.6112884 -4.3061106
```

```
head(llm)
```

```
## [1] -5.0026673  1.9820776 -0.3263347 -0.9978646  0.6112884 -4.3061106
```

```
head(pp$sd)
```

```
##              length
## 1.4653096 0.4145808 0.3992531 0.3845635 0.1655412 3.0288760
```

```
head(lls)
```

```
## [1] 1.4653096 0.4145808 0.3992531 0.3845635 0.1655412 3.0288760
```

```
## Check the differences in time
system.time(rowstat(xx))
```

```
##    user  system elapsed
##    0.02    0.00    0.01
```

```
system.time(apply(xx,1,mean)) +  system.time(apply(xx,1,sd))
```

```
##    user  system elapsed
##    0.10    0.00    0.08
```

```
## Use Rprof() to profile the code. Is Rprof() useful here? Where is the maximum time being spent?
Rprof(interval=0.002)
pp=rowstat(xx)
Rprof(NULL)
summaryRprof()
```

```
## $by.self
##         self.time self.pct total.time total.pct
## "pow"       0.002       50      0.002        50
## "Rprof"     0.002       50      0.002        50
##
## $by.total
##                       total.time total.pct self.time self.pct
## "block_exec"               0.004       100     0.000        0
## "call_block"               0.004       100     0.000        0
## "doWithOneRestart"         0.004       100     0.000        0
## "eng_r"                    0.004       100     0.000        0
## "eval"                     0.004       100     0.000        0
## "evaluate"                 0.004       100     0.000        0
## "evaluate::evaluate"       0.004       100     0.000        0
## "in_dir"                   0.004       100     0.000        0
## "in_input_dir"             0.004       100     0.000        0
## "knitr::knit"              0.004       100     0.000        0
## "process_file"             0.004       100     0.000        0
## "process_group"            0.004       100     0.000        0
## "rmarkdown::render"        0.004       100     0.000        0
## "with_handlers"            0.004       100     0.000        0
## "with_options"             0.004       100     0.000        0
## "withCallingHandlers"      0.004       100     0.000        0
## "withOneRestart"           0.004       100     0.000        0
## "withRestartList"          0.004       100     0.000        0
## "withRestarts"             0.004       100     0.000        0
## "withVisible"              0.004       100     0.000        0
## "xfun:::handle_error"      0.004       100     0.000        0
## "pow"                      0.002        50     0.002       50
## "Rprof"                    0.002        50     0.002       50
## "rowstat"                  0.002        50     0.000        0
##
## $sample.interval
## [1] 0.002
##
## $sampling.time
## [1] 0.004
```

Part 1

We can update his function rowstat to compute row-wise medians instead of row mean and sd like it currently does in his code.

We use the apply function row-wise (MARGIN=1) and we apply the median to each row.

```
rowstat <- function(x){
    apply(Sigma, 1, median)
}
```

A more manual approach would require sorting each row and then computing the median either as the average of the two middle values or the singular middle value. Alternatively, just use median in each row like the example below.

```r
rowstat <- function(x){
    r_md <- numeric(length=nrow(x))
    for (i in 1:nrow(x))
        {r_md[i] <- median(x[i,])
    }
    list(median = r_md)
}
```

We can even test it out:

```r
Sigma <- matrix(c(10,3,3,2),2,2)
medians <- rowstat(Sigma)
medians
```

```
## $median
## [1] 6.5 2.5
```

Part 2

We update the code and we can run our function against the median function already in R. All code is also in the R script file. WE update it to 15,000 samples and test how fast they each run.

profexample_updated_Problem2_Part2.pdf

```r
## Author: Anindya Bhadra
## STAT 545
## Example of code profiling and timing: for loop vs. apply()
##Clear workspace and load required libraries
rm(list=ls())
library(MASS)

pow=function(x,lambda){x^lambda}

## Write a function called "rowstat" using for loop to compute row medians
rowstat <- function(x){
  r_md <- numeric(length=nrow(x))
  for (i in 1:nrow(x))
  {r_md[i] <- median(x[i,])
  }
  list(median = r_md)
}

Sigma <- matrix(c(10,3,3,2),2,2)

## Generate 15,000 samples from a bivariate Normal distribution with mean 0 and covariance matrix Sigma
xx=mvrnorm(n=15000,c(0,0), Sigma)

## Call the function "rowstat"
pp=rowstat(xx)

## Do the same operations using apply
llm=apply(xx,1,median)
```

```r
## Check that they give the same result
head (pp$median)
```

```
## [1]  1.1030823 -0.1005155  0.9240484 -1.3238688  0.6407471 -4.0549843
```

```r
head(llm)
```

```
## [1]  1.1030823 -0.1005155  0.9240484 -1.3238688  0.6407471 -4.0549843
```

```r
## Check the differences in time
system.time(rowstat(xx))
```

```
##    user  system elapsed
##    0.23    0.03    0.27
```

```r
system.time(apply(xx,1,median))
```

```
##    user  system elapsed
##    0.27    0.02    0.32
```

```r
## Use Rprof() to profile the code. Is Rprof() useful here? Where is the maximum time being spent?
Rprof(interval=0.002)
pp=rowstat(xx)
Rprof(NULL)
summaryRprof()
```

```
## $by.self
##                 self.time self.pct total.time total.pct
## "median"            0.010    29.41      0.032     94.12
## "sort"              0.006    17.65      0.016     47.06
## "eval"              0.002     5.88      0.034    100.00
## "mean"              0.002     5.88      0.022     64.71
## "sort.default"      0.002     5.88      0.010     29.41
## "sort.int"          0.002     5.88      0.008     23.53
## "match.arg"         0.002     5.88      0.004     11.76
## "mean.default"      0.002     5.88      0.004     11.76
## "is.factor"         0.002     5.88      0.002      5.88
## "is.na"             0.002     5.88      0.002      5.88
## "Rprof"             0.002     5.88      0.002      5.88
##
## $by.total
##                         total.time total.pct self.time self.pct
## "eval"                       0.034    100.00     0.002     5.88
## "block_exec"                 0.034    100.00     0.000     0.00
## "call_block"                 0.034    100.00     0.000     0.00
## "doWithOneRestart"           0.034    100.00     0.000     0.00
## "eng_r"                      0.034    100.00     0.000     0.00
## "evaluate"                   0.034    100.00     0.000     0.00
## "evaluate::evaluate"         0.034    100.00     0.000     0.00
## "in_dir"                     0.034    100.00     0.000     0.00
```

```
## "in_input_dir"             0.034    100.00     0.000     0.00
## "knitr::knit"              0.034    100.00     0.000     0.00
## "process_file"             0.034    100.00     0.000     0.00
## "process_group"            0.034    100.00     0.000     0.00
## "rmarkdown::render"        0.034    100.00     0.000     0.00
## "with_handlers"            0.034    100.00     0.000     0.00
## "with_options"             0.034    100.00     0.000     0.00
## "withCallingHandlers"      0.034    100.00     0.000     0.00
## "withOneRestart"           0.034    100.00     0.000     0.00
## "withRestartList"          0.034    100.00     0.000     0.00
## "withRestarts"             0.034    100.00     0.000     0.00
## "withVisible"              0.034    100.00     0.000     0.00
## "xfun:::handle_error"      0.034    100.00     0.000     0.00
## "median"                   0.032     94.12     0.010    29.41
## "rowstat"                  0.032     94.12     0.000     0.00
## "mean"                     0.022     64.71     0.002     5.88
## "median.default"           0.022     64.71     0.000     0.00
## "sort"                     0.016     47.06     0.006    17.65
## "sort.default"             0.010     29.41     0.002     5.88
## "sort.int"                 0.008     23.53     0.002     5.88
## "match.arg"                0.004     11.76     0.002     5.88
## "mean.default"             0.004     11.76     0.002     5.88
## "is.factor"                0.002      5.88     0.002     5.88
## "is.na"                    0.002      5.88     0.002     5.88
## "Rprof"                    0.002      5.88     0.002     5.88
## "isTRUE"                   0.002      5.88     0.000     0.00
##
## $sample.interval
## [1] 0.002
##
## $sampling.time
## [1] 0.034
```

We can see that the heads do indeed match up. Unfortunately, it seems that using apply is faster than our method of applying median using a for loop (0.25 seconds for rowstat() to 0.24 for apply() in my best run). However, it is not too much faster, only by 0.01 seconds. Most of the time spent is actually using the function median, which makes sense. Median probably requires some sort of sorting algorithm that might take some time.

Interestingly enough, every time I knit this into a pdf, it takes a vastly different amount of time to run the codes, from 0.24 seconds to over 0.5. I will report the times I saw on my last run but it seems quite inconsistent and it is hard to clearly say which method is truly faster given such variability. The true times may be inaccurate in the final knit file. The code is provided as documentation within this notebook.