HW4_2_linal20

Lina Lee

10/16/2020

problem 2

```
set.seed(1256)
theta <- as.matrix(c(1,2),nrow=2)
X <- cbind(1,rep(1:10,10))
h <- X%*%theta+rnorm(100,0,0.2)
one<-rep(1,100)

lm(h~0+X)
```

```
##
## Call:
## lm(formula = h ~ 0 + X)
##
## Coefficients:
##     X1      X2
## 0.9696  2.0016
```

problem 3

Part a.

Using a step size of 1e−7 and tolerance of 1e−9, try 10000 different combinations of start values for 0 and 1 across the range of possible 's +/-1 from true determined in Problem 2, making sure to take advantages of parallel computing opportunities. In my try at this, I found starting close to true took 1.1M iterations, so set a stopping rule for 5M. Report the min and max number of iterations along with the starting values for those cases. Also report the average and stdev obtained across all 10000 's.

```
# b0range<-seq(1-1,1+1,by=0.02)
# b0range<-b0range[-1]
# length(b0range)
# b1range<-seq(2-1,2+1,by=0.02)
# b1range<-b1range[-1]
# length(b1range)
#
# input_grid<-expand.grid(b0range,b1range)
# m=length(X[,1])
```

```
#
# library(parallel)
# library(doParallel)
# library(tidyverse)
# cores<-detectCores()-1
# cl<-makeCluster(cores)
# registerDoParallel(cl)
#
# dim(input_grid)
#
# a=10^(-7)
# t=10^(-9)
# result<-foreach(i = 1:10000, .combine='c')%dopar%{
# theta0_new=input_grid$Var1[i]
# theta1_new=input_grid$Var2[i]
# while(abs(theta0_new -theta0)>t & abs(theta1_new-theta1 &j)>t){
#
# theta0_new <-theta0-a*(1/m)*t(one)%*%(X%*%theta-h)
# theta1_new <-theta1-a*(1/m)*t(X[,2])%*%(X%*%theta-h)
# }
# }
# stopCluster(cl)


load("result.Rdata")
result<-as.vector(result)
result.mat<-as.data.frame(matrix(result,nrow=10000,ncol=3, byrow=TRUE))
names(result.mat)<-c("iterations","beta0","beta1")
head(result.mat)
```

```
##   iterations     beta0    beta1
## 1    1825760 0.2070143 2.110240
## 2    1829545 0.2259341 2.107537
## 3    1833406 0.2448521 2.104835
## 4    1837347 0.2637683 2.102132
## 5    1841371 0.2826825 2.099430
## 6    1845480 0.3015947 2.096728
```

Report the min and max number of iterations along with the starting values for those cases.

```
result.mat[which.max(result.mat$iterations),]
```

```
##      iterations    beta0    beta1
## 4099      5e+06 1.883911 1.870227
```

```
result.mat[which.min(result.mat$iterations),]
```

```
##      iterations beta0 beta1
## 4098          1  1.96  1.82
```

Also report the average and stdev obtained across all 10000 's

```
library(tidyverse)
result.mat%>%
  summarise(avgb0=mean(beta0),stdevb0=sd(beta0) ,avgb1=mean(beta1),stdevb1=sd(beta1))
```

```
##      avgb0    stdevb0    avgb1     stdevb1
## 1 1.006954 0.5566088 1.995641 0.09277835
```

Average and stdev of b0 is 1.007 and 0.557, Average and stdev of b1 is 1.996 and 0.093.

Problem 4: Inverting matrices

so John Cook makes some good points, but if you want to do: $\hat{\beta} = (X^T X)^{-1} X^T y$ what are you to do?? Can you explain what is going on?

```
X <- cbind(1,rep(1:10,10))
y <- X%*%theta+rnorm(100,0,0.2)
solve(t(X)%*%X)%*%t(X)%*%y
```

```
##           [,1]
## [1,] 0.9920806
## [2,] 1.9995305
```

$\hat{\beta} = (X^T X)^{-1} X^T y$ is the least square estimator of β. This estimates the betas which minimize the squared error term.

Problem 5: In this problem, we are looking to compute the following:

$y = p + AB^{-1} * (q - r)$ Where A, B, p, q and r are formed by:

```
set.seed(12456)
G <- matrix(sample(c(0,0.5,1),size=16000,replace=T),ncol=10)
R <- cor(G) # R: 10 * 10 correlation matrix of G
C <- kronecker(R, diag(1600)) # C is a 16000 * 16000 block diagonal matrix
id <- sample(1:16000,size=932,replace=F)
q <- sample(c(0,0.5,1),size=15068,replace=T) # vector of length 15068
A <- C[id, -id] # matrix of dimension 932 * 15068
B <- C[-id, -id] # matrix of dimension 15068 * 15068
p <- runif(932,0,1)
r <- runif(15068,0,1)
C<-NULL #save some memory space
```

Part a.

How large (bytes) are A and B? Without any optimization tricks, how long does the it take to calculate y?

```
object.size(A)
```

```
## 112347224 bytes
```

```
object.size(B)
```

```
## 1816357208 bytes
```

```
Prob5part_a<-system.time({y = p + A%*%solve(B)%*%(q - r)})
names(Prob5part_a)<-"timing without cholesky decomposition"
Prob5part_a
```

```
##    user  system elapsed
##  877.75   12.11  893.32
```

size of A is 112347224 bytes and the size of B is 1816357208 bytes. It takes to calculate y without any optimization tricks.

Part b.

How would you break apart this compute, i.e., what order of operations would make sense? Are there any mathmatical simplifications you can make? Is there anything about the vectors or matrices we might take advantage of?

we can compute B inverse first and then product A and B inverse. and compute q-r. Next, product the AB^{-1} and q-r. To get y we can add p to the $AB^{-1} * (q - r)$ Because the matrix A and B is so large, we need a way to simplify the matrix. Since the matrix is positive definite matrix, we can apply Cholesky decomposition and Cholesky inverse to simplify the the special matrice and speed up the computing.

Part c.

Use ANY means (ANY package, ANY trick, etc) necessary to compute the above, fast. Wrap your code in "system.time({})", everything you do past assignment "C <- NULL".

```
Prob5part_b<-system.time({y = p + A%*%chol2inv(B, size = NCOL(B), LINPACK = FALSE)%*%(q - r)})
names(Prob5part_b)<-"timing with cholesky decomposition"
```

timing without cholesky decomposition user system elapsed 781.67 14.56 799.17 timing with cholesky decomposition user system elapsed 545.28 0.97 546.55

The time using cholesky inverse took shorter time than the original way.

Problem 6

a.

Create a function that computes the proportion of successes in a vector. Use good programming practices.

```
successProp<-function(vec){
  successProp<-mean(vec)
  return(successProp)
}
```

b.

Create a matrix to simulate 10 flips of a coin with varying degrees of "fairness" (columns = probability)

```
set.seed(12345)
P4b_data <- matrix(rbinom(10, 1, prob = (31:40)/100), nrow = 10, ncol = 10, byrow = FALSE)
```

c.

Use your function in conjunction with apply to compute the proportion of success in P4b_data by column and then by row. What do you observe? What is going on?

```
# by row
successPropList1<-apply(P4b_data,MARGIN = 1,FUN = successProp)
successPropList1
```

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

```
# by col
successPropList2<-apply(P4b_data,MARGIN = 2,FUN = successProp)
successPropList2
```

```
##  [1] 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
```

the value computed by row is 0.6 times the value computed by column. It looks like it is calculated as if sets of 10 flips are not random.

d. You are to fix the above matrix by creating a function whose input is a probability and output is a vector whose elements are the outcomes of 10 flips of a coin. Now create a vector of the desired probabilities. Using the appropriate apply family function, create the matrix we really wanted above.

Prove this has worked by using the function created in part a to compute and tabulate the appropriate marginal successes.

```
 prob <- (31:40)/100

binom<-function(p){
  binomVec<-rbinom(10, 1, prob = p)
 return(binomVec)
```

```
}


binomMat<-sapply(prob,FUN = binom)

binomMat
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    0    1    1    1    1    1    1    1     0
##  [2,]    0    0    0    0    1    0    0    0    1     1
##  [3,]    1    1    0    1    0    1    0    0    0     1
##  [4,]    0    1    1    1    0    1    0    0    0     1
##  [5,]    0    0    0    0    1    1    0    0    1     0
##  [6,]    0    0    0    0    0    0    0    0    0     1
##  [7,]    0    1    1    0    1    1    1    1    1     1
##  [8,]    0    0    1    0    0    0    1    0    1     1
##  [9,]    0    0    0    0    0    0    0    1    0     0
## [10,]    1    0    0    0    0    1    0    0    0     0
```

The matrix is combined vectors whose elements are the outcomes of 10 flips of a coin. 1 is success(or head), 0 is fail(or tail).

```
apply(binomMat,MARGIN = 1,FUN = successProp)
```

```
##  [1] 0.7 0.3 0.5 0.5 0.3 0.1 0.8 0.4 0.1 0.2
```

each value of the vector is the proportion of successes(head or tail) of 10 flips.

Problem 7

In Homework 4, we had a dataset we were to compute some summary statistics from. The description of the data was given as "a dataset which has multiple repeated measurements from two devices by thirteen Observers". Where the device measurements were in columns "dev1" and "dev2". Reimport that dataset, change the names of "dev1" and "dev2" to x and y and do the following:

```
df <- readRDS(file="C:/Users/linal/Desktop/VT2020Fall/stat programming/HW3_data.rds")
library(ggplot2)
library(tidyverse)
```

1.

create a function that accepts a dataframe of values, title, and x/y labels and creates a scatter plot

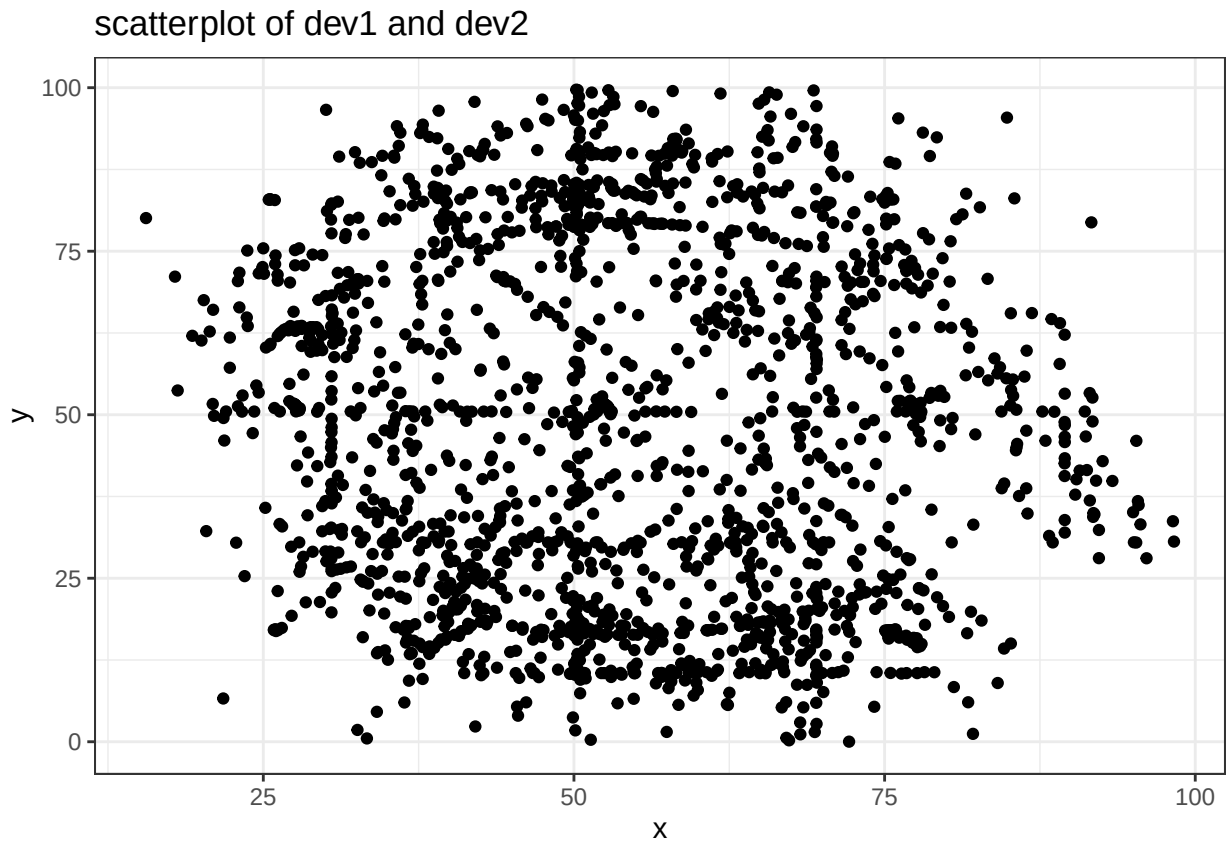
```
names(df)<-c("obs","x","y")
scatterplot<-function(data,title,xlab,ylab){

ggplot()+
    geom_point(aes(x=x,y=y),data=data)+
    labs(title=title, x=xlab,y=ylab)+
    theme_bw()
}
```

2. use this function to create:

(a) a single scatter plot of the entire dataset

```
scatterplot(data=df,title="scatterplot of dev1 and dev2",xlab="x",ylab="y")
```



(b) a seperate scatter plot for each observer (using the apply function)

```
str(df)
```

```
## 'data.frame':    1846 obs. of  3 variables:
##  $ obs: num  4 4 4 4 4 4 4 4 4 4 ...
##  $ x  : num  55.4 51.5 46.2 42.8 40.8 ...
##  $ y  : num  97.2 96 94.5 91.4 88.3 ...
```

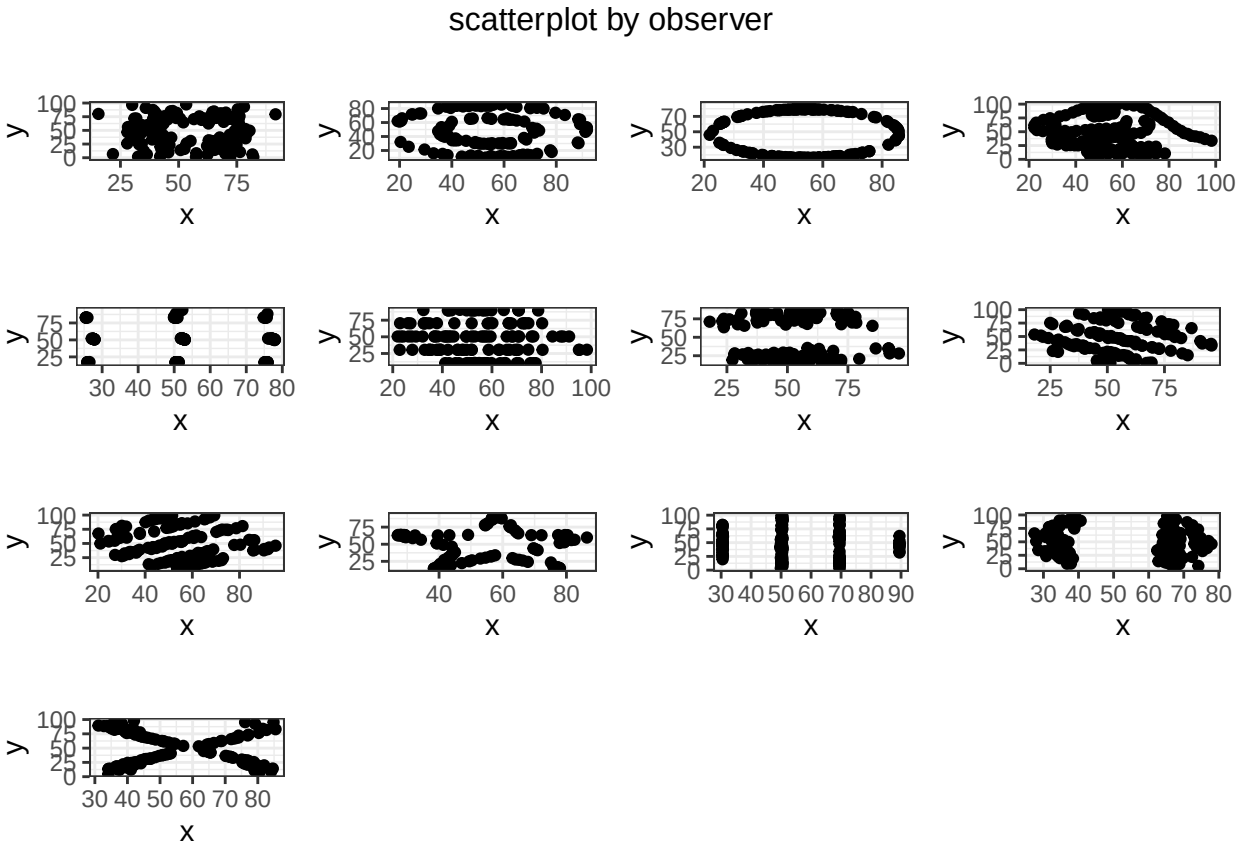
```
subset<-function(i){
subdf<-df%>%filter(obs==i)
return(subdf)
}

obsVec<-1:13
subdf1<-subset(1)
subcollect<-lapply(obsVec, FUN = subset)
```

```
pl<-list()

for(i in 1:13){
pl[[i]]<-scatterplot(data=subcollect[i][[1]], title="",xlab = "x" ,ylab = "y")
}


gridExtra::grid.arrange(pl[[1]],pl[[2]],pl[[3]],pl[[4]],pl[[5]],
                        pl[[6]],pl[[7]],pl[[8]],pl[[9]],pl[[10]],
                        pl[[11]],pl[[12]],pl[[13]],top="scatterplot by observer")
```



Problem 8

Part a.

Get and import a database of US cities and states. Here is some R code to help:

```
#we are grabbing a SQL set from here
# http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip
#download the files, looks like it is a .zip

library(downloader)
download("http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip",dest="us_cities_states
```

```
unzip("us_cities_states.zip")
#read in data, looks like sql dump, blah
library(data.table)
states <- fread(input = "./us_cities_and_states/states.sql",skip = 23,sep = "'", sep2 = ",", header = F
### YOU do the CITIES
### I suggest the cities_extended.sql may have everything you need
### can you figure out how to limit this to the 50?
cities <- fread(input = "./us_cities_and_states/cities_extended.sql",skip = 23,sep = "'", sep2 = ",", he
```

Part b.

Create a summary table of the number of cities included by state.

```
library(tidyverse)
cityN<-cities %>%
  group_by(V4)%>%
    count(V4)

cityNtbl<-cityN
names(cityNtbl)<-c("State","Number of Cities")

knitr::opts_chunk$set(fig.align = 'center')
kable(cityNtbl,caption = "A summary table of the number of cities by state")
```

Part c.

Create a function that counts the number of occurances of a letter in a string. The input to the function should be "letter" and "state_name". The output should be a scalar with the count for that letter.

```
letter_count <- data.frame(matrix(NA,nrow=50, ncol=26))
string<-tolower(states$V2)
string<-as.array(string)

letterL<-letters[1:26]
countMat<-array(rep(NA,26*50),dim=c(20,26))

# str(string)

getCount <- function(letter,state_name){
# state_name<-string[[1]]
  countVec<-c()
temp <- strsplit(state_name,"")

for(i in 1:length(temp[[1]])){
countVec[i] <- ifelse(temp[[1]][i]==letter,1,0)
}
count<-sum(countVec)
# countMat[is.na(countMat)] <- 0
# count<-apply(countMat,2,sum)
return(count)
}
```

Table 1: A summary table of the number of cities by state

State	Number of Cities
AK	273
AL	838
AR	709
AZ	532
CA	2651
CO	659
CT	438
DC	284
DE	98
FL	1487
GA	972
HI	139
IA	1060
ID	325
IL	1587
IN	989
KS	756
KY	961
LA	725
MA	703
MD	619
ME	489
MI	1170
MN	1031
MO	1170
MS	533
MT	405
NC	1090
ND	407
NE	620
NH	284
NJ	733
NM	426
NV	253
NY	2207
OH	1446
OK	774
OR	484
PA	2208
PR	176
RI	91
SC	539
SD	394
TN	795
TX	2650
UT	344
VA	1238
VT	309
WA	732
WI	898
WV	859
WY	195

Create a for loop to loop through the state names imported in part a. Inside the for loop, use an apply family function to iterate across a vector of letters and collect the occurance count as a vector.

```
letter_count<-array(rep(NA,26*52),dim=c(52,26))
for(i in 1:length(string)){

letter_count[i,] <- unlist(lapply(letterL,function(x) getCount(x,string[[i]])))
}

c(3,4,5,6) %in% letter_count[1,]
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```
# createt a table including variable which indicates state highlighted or not
states_highlighted<-c()
for(i in 1:length(string)){
states_highlighted[i]<-(TRUE %in% (letter_count[i,]>2))
}
states_highlighted
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [25] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
## [49] FALSE  TRUE FALSE
```

```
states<-cbind(states,states_highlighted)
states_sub<-states%>%filter(states_highlighted==TRUE)
```

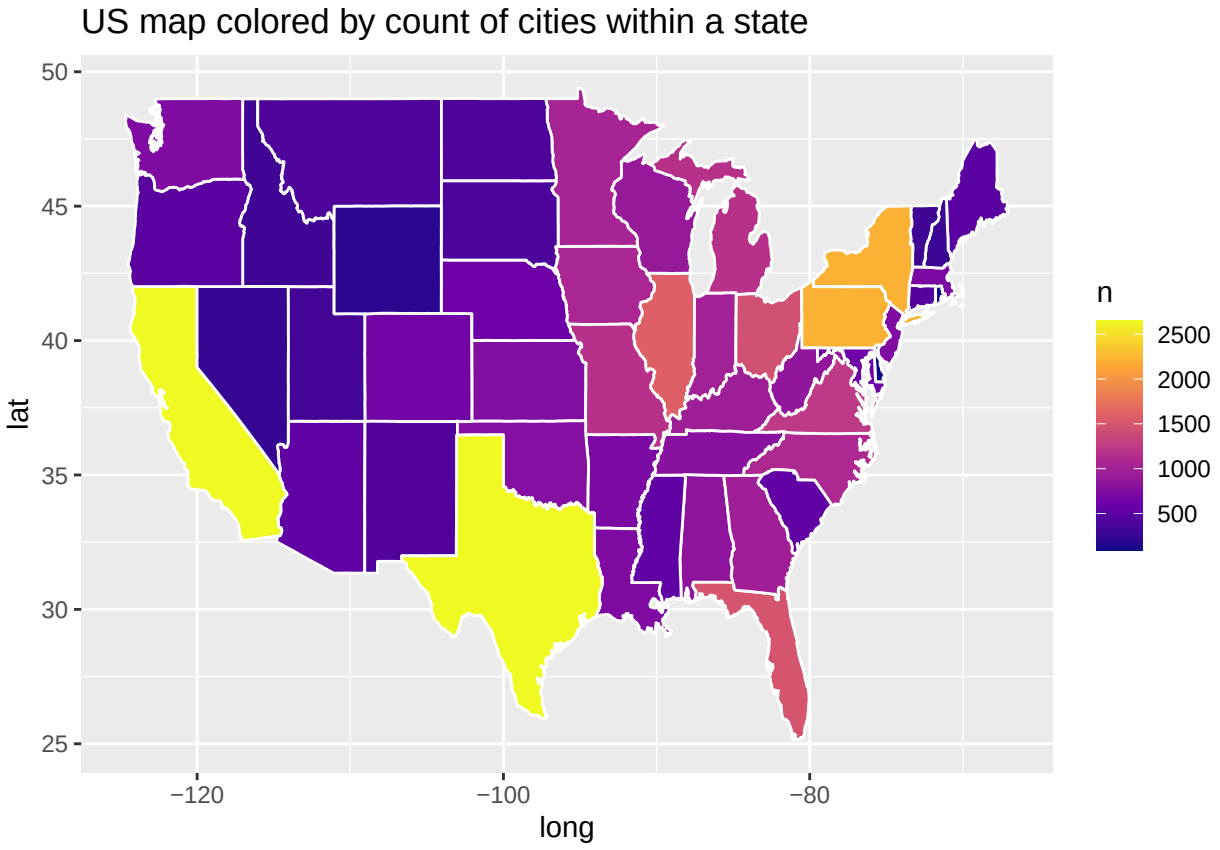
Part d.

Create 2 maps to finalize this. Map 1 should be colored by count of cities on our list within the state. Map 2 should highlight only those states that have more than 3 occurances of ANY letter in thier name. Quick and not so dirty map: #https://cran.r-project.org/web/packages/fiftystater/vignettes/fiftystater.html

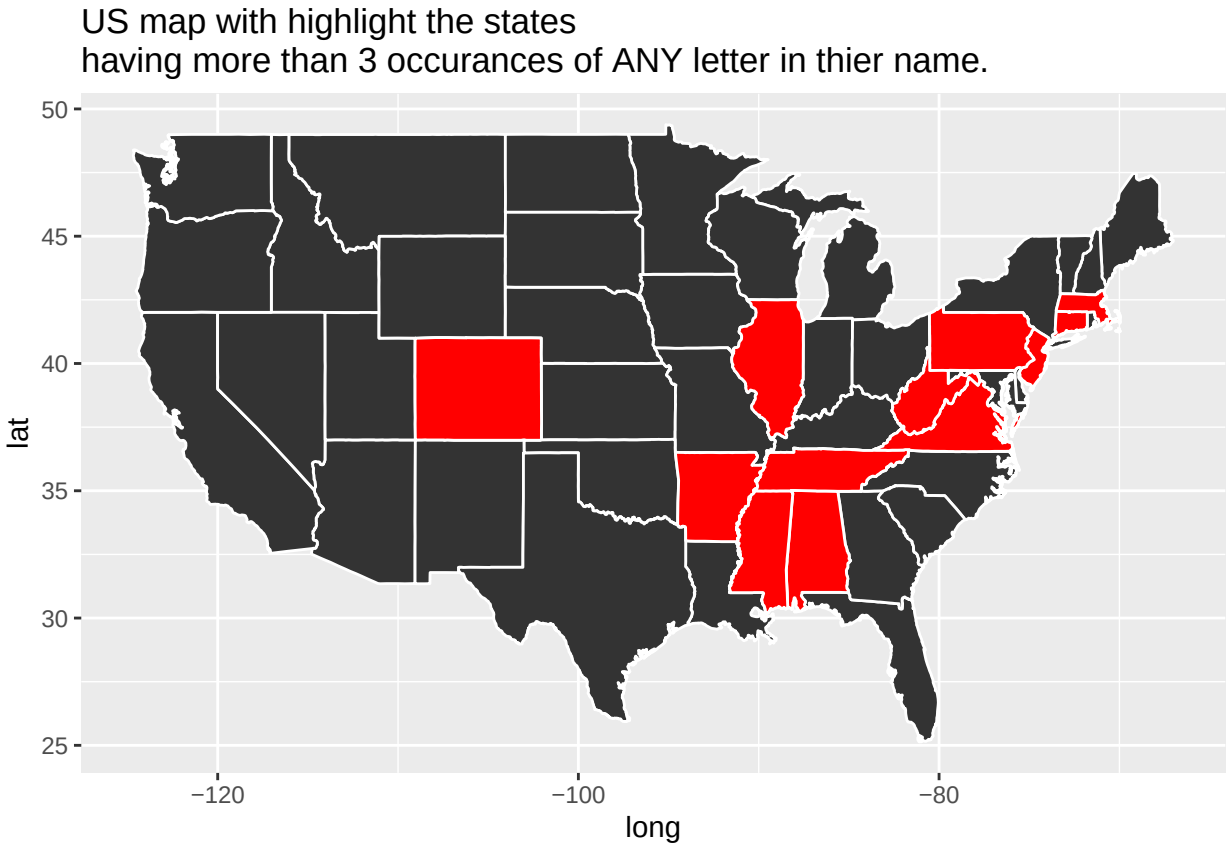
```
states_map <- map_data("state")
cityN_new<-left_join(states, cityN, by = "V4")
names(cityN_new)<-c("region","Abbre","highlight","n")
cityN_new$region<-tolower(cityN_new$region)


names(states_sub)<-c("region","Abbre","highlight")
states_sub$region<-tolower(states_sub$region)

map <- left_join(states_map,cityN_new , by = "region")
map_high<-left_join(states_sub,states_map , by = "region")
# Create the map
ggplot(map, aes(long, lat, group = group))+
  geom_polygon(aes(fill = n), color = "white")+
  scale_fill_viridis_c(option = "C",direction = 1)+
  labs(title="US map colored by count of cities within a state")
```




```
ggplot(map, aes(long, lat, group = group))+
  geom_polygon(color = "white")+
  geom_polygon(data = map_high, fill = "red", color = "white") +
  scale_fill_viridis_c(option = "C",direction = 1) +
  labs(title="US map with highlight the states \nhaving more than 3 occurances of ANY letter in thier na
```




```
#ggsave(plot = p, file = "HW6_Problem6_Plot_Settlage.pdf")
```

Problem 9

Part a.

First, the question asked in the stackexchange was why is the supplied code not working. This question was actually never answered. What is the problem with the code? If you want to duplicate the code to test it, use the quantreg package to get the data.

That is because the person used set.seed(). set.seed with the same starting number produce same results he should not use set.seed() to produce different resamples of original data.

Part b.

Bootstrap the analysis to get the parameter estimates using 100 bootstrapped samples. Make sure to use system.time to get total time for the analysis. You should probably make sure the samples are balanced across operators, ie each sample draws for each operator.

```
# get data

library(RCurl)
# read data into R
```

```r
mydat <- as.vector(read.delim("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat",skip=
# transform data into vector
mydat<-as.vector(t(data.matrix(mydat)))
# remove NA elements
mydat<-mydat[!is.na(mydat)]
# transform vector into the matrix form
# now each row is for a item
newdat<-as.data.frame(t(matrix(mydat,ncol = 10)))
# transform data so that each columns represent item or an operator
data_old<-matrix(NA,ncol=6)
for(i in 1:10){
data_new<-cbind(rep(i,3),t(matrix(newdat[i,][-1],5)))
data_old<-rbind(data_old,data_new)
}
# remove initialized empty row
updateDat<-as.data.frame(data_old[-1,])
#change the columns names
names(updateDat)<-c("item","1","2","3","4","5")
# change the columns types
updateDat$item<-as.factor(unlist(updateDat$item))
updateDat$"1"<-unlist(updateDat$"1")
updateDat$"2"<-unlist(updateDat$"2")
updateDat$"3"<-unlist(updateDat$"3")
updateDat$"4"<-unlist(updateDat$"4")
updateDat$"5"<-unlist(updateDat$"5")
# stack data so that we can have one columns for operator
stackDat<-stack(updateDat[,-1])
# combined data with the vector for item
finaldf<-cbind(unlist(rep(updateDat[,1],5)),stackDat)
#change the columns' positions
sensoryDat <- finaldf[, c(1, 3, 2)]
# change the columns' names
names(sensoryDat)<-c("item","operator","values")
```

```r
library(tidyverse)


B<-100
boot_coef<-array(rep(NA, B*5),dim = c(B,5))

resamples<-list()
time_a<-system.time(for (b in 1:B){
  for(i in 1:5){
  data<-sensoryDat[sensoryDat$operator==i,]
  resamples[[i]]<- data[sample(1:nrow(data), replace=T), ]

  }
  bootsamp<-do.call("rbind", resamples)
  boot_coef[b,]<-lm(values~operator,data=bootsamp)$coef
})
```

```r
coef_final<-apply(boot_coef,2,mean)
names(coef_final)<-c("intercept","operater2","operater3","operater4","operater5")
```

Table 2: The bootstrapped coefficient of regression

	x
intercept	4.5724333
operater2	0.4640000
operater3	-0.4058000
operater4	0.6671000
operater5	-0.3206333

```
knitr::opts_chunk$set(fig.align = 'center')

kable(coef_final,caption = "The bootstrapped coefficient of regression")
```

Part c. Redo the last problem but run the bootstraps in parallel (cl <- makeCluster(8)), don't forget to stopCluster(cl)). Why can you do this? Make sure to use system.time to get total time for the analysis.

```
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators
```

```
library(tidyverse)
cores<-detectCores()-1
cl<-makeCluster(cores)
registerDoParallel(cl)
clusterExport
```

```
## function (cl = NULL, varlist, envir = .GlobalEnv)
## {
##     for (name in varlist) {
##         clusterCall(cl, gets, name, get(name, envir = envir))
##     }
## }
## <bytecode: 0x000000001ced1460>
## <environment: 0x000000001ced3c78>
```

Table 3: timing from part a and b

	user.self	sys.self	elapsed
time_a	0.31	0.00	0.31
time_b	0.05	0.01	0.19

```
time_b<-system.time(bootcoefd<-foreach(b = 1:B, .combine='c')%dopar%{
  for(i in 1:5){
  data<-sensoryDat[sensoryDat$operator==i,]
  resamples[[i]]<- data[sample(1:nrow(data), replace=T), ]

  }
  bootsamp<-do.call("rbind", resamples)
  boot_coef[b,]<-lm(values~operator,data=bootsamp)$coef
})



stopCluster(cl)
```

Since we use several clusters to run the bootstrap, it takes shorter time than the one without using parallel. we can do parallel since we can allocate the bootstrappings in each cluster.

Create a single table summarizing the results and timing from part a and b. What are your thoughts?

```
remove(time)
```

```
## Warning in remove(time): object 'time' not found
```

```
time<-rbind(time_a,time_b)
time<-time[,c(1,2,3)]

knitr::opts_chunk$set(fig.align = 'center')

kable(time,caption = "timing from part a and b")
```

Problem 10

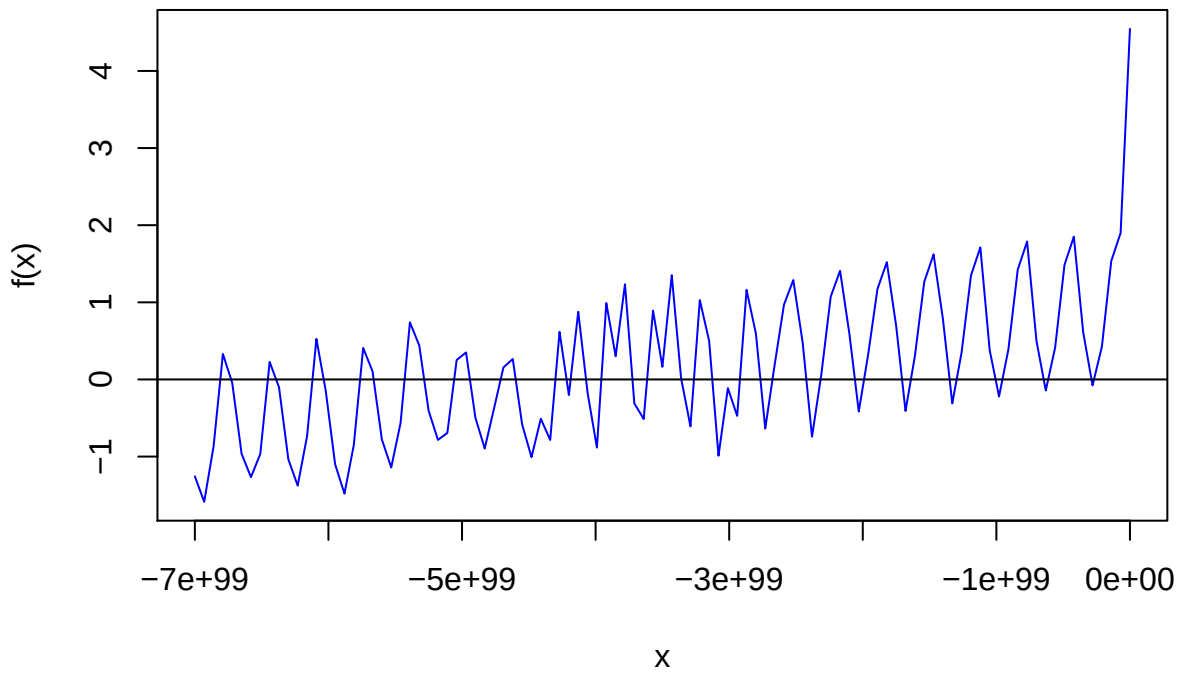
Newton's method gives an answer for a root. To find multiple roots, you need to try different starting values. There is no guarantee for what start will give a specific root, so you simply need to try multiple. From the plot of the function in HW4, problem 8, how many roots are there? Create a vector (length.out=1000) as a "grid" covering all the roots and extending +/-1 to either end.

Part a. Using one of the apply functions, find the roots noting the time it takes to run the apply function.

```r
# create a function of given formula
f <- function(x){return(3^x-sin(x)+cos(5*x))}

# plot the function

curve(expr=f,from=-7*10^99,to=1.5,col="blue")
abline(h=0)
```



there are total 41

```r
# create a function of newton algorithm
newton <- function(x0) {

  f=f
  tol=1E-12
  N=50
  h <- 0.001
  i <- 1; x1 <- x0
  p <- numeric(N)

  while (i<=N) {
    df.dx <- (f(x0+h)-f(x0))/h
    x1 <- (x0 - (f(x0)/df.dx))
    p[i] <- x1
```

Table 4: A summary table of timing from both parts a and b

	user.self	sys.self	elapsed
time_aa	0.08	0	0.08
time_bb	0.00	0	0.02

```
    i <- i + 1
  if (abs(x1-x0) < tol) break
    x0 <- x1
  }
  return(p[(i-1)])
}
#
# p<-newton(0.5)
# p
# create a input (x) data
input.grid<-as.data.frame(x=seq(-10^3,0,by=1))
names(input.grid)<-c("x")

# system.time({root<-sapply(input.grid, function(x) newton(x))})

time_aa<-system.time({root<-apply(input.grid,1, function(x) newton(x))})
root<-unique(root)
head(root)
```

```
## [1] -999.8119 -999.4192 -997.8484 -996.6703 -996.6703 -994.7068
```

Part b. Repeat the apply command using the equivelant parApply command. Use 8 workers. cl <- makeCluster(8) Create a table summarizing the roots and timing from both parts a and b. What are your thoughts?

```
library(parallel)

cl<-makeCluster(8)
clusterExport(cl,c('newton','f'))
time_bb<-system.time({parLapply(cl,input.grid,function(x) newton(x) )})
stopCluster(cl)
```

```
timetbl2<-rbind(time_aa,time_bb)
timetbl2<-timetbl2[,c(1,2,3)]
knitr::opts_chunk$set(fig.align = 'center')
kable(timetbl2,caption = "A summary table of timing from both parts a and b")
```

The way with parApply command takes a shorter time than the one without using parellel way.