Homework 5

Radhakrishna Adhikari

11/14/2021

Problem 2

Part 1. The problem with the code is inside the for loop. Even though the code samples the data with replacement every time, while performing regression the code is not using this sampled data. He is using old data. I have fixed it in the following code.

```
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
apple08 <- getSymbols('AAPL', auto.assign = FALSE, from = '2008-1-1', to =
                        "2008-12-31",warnings = FALSE)[,6]
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
#market proxy
rm08<-getSymbols('^ixic', auto.assign = FALSE, from = '2008-1-1', to =
                   "2008-12-31",warnings = FALSE)[,6]

#log returns of AAPL and market
logapple08<- na.omit(ROC(apple08)*100)
logrm08<-na.omit(ROC(rm08)*100)

#OLS for beta estimation
beta_AAPL_08<-summary(lm(logapple08~logrm08))$coefficients[2,1]


df08<-cbind(logapple08,logrm08)
set.seed(666)
Boot=1000
sd.boot=rep(0,Boot)
for(i in 1:Boot){
  # nonparametric bootstrap
  bootdata=df08[sample(nrow(df08), size = 251, replace = TRUE),]
  sd.boot[i]= coef(summary(lm(bootdata$AAPL.Adjusted~bootdata$IXIC.Adjusted)))[2,2]
}
##checking the first 10 values
sd.boot[1:10]
```

```
##  [1] 0.06861689 0.05623265 0.05940474 0.07227112 0.04946581 0.06534474
##  [7] 0.05475725 0.06822139 0.05293367 0.04875212
```

Notice that first 10 values are not same. It means that the code is working.

Part b. I have used bootstrap to calculate the betas in the following code.

```
set.seed(1)
X<-seq(from=0.5,to=10,by=0.5)
x<-c(X,X,X)
y <- 1+2*x+rnorm(60)

#print the regression coefficients
summary(lm(y~x))$coef
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 0.9782697 0.23050980  4.24394 8.019604e-05
## x           2.0246375 0.03848517 52.60825 1.201670e-50
```

```
#Bootstrap

n<-length(y)
boot=100


bbsest1=NULL

#start the time from here
start<-Sys.time()
```

```
for(i in 1: boot){
  sampleind<-sample(c(1:n), n, replace=TRUE)
  samx<-x[sampleind]
  samy<-y[sampleind]
  model<-lm(samy~samx)
  bbsest1<-rbind(bbsest1,coef(model))
}

end<-Sys.time()

#time to run above chunk of code
end-start
```

```
## Time difference of 0.183876 secs
```

```
#calculated b0
b0<-mean(bbsest1[,1])
b0
```

```
## [1] 0.9569727
```

```
#calculated b1
b1<-mean(bbsest1[,2])
b1
```

```
## [1] 2.02379
```

Problem 3

Part a.
From the plot, there are 4 roots of this equation. I have written the following code in order to calculate root. Since, there are 4 roots and following algorithm returns only one root at a time, I use different initial points to find roots.

```
f<-function(x){
  (3^x)-sin(x)+cos(5*x)+(x^2)-1.5
}


fprime <- function(x) {
 ( x*(3^(x-1)) ) + cos(x) - 5*sin(5*x) + 2*x
}

#newton algorithm to find the root
newton_root <- function( init,f1=f, f2=fprime, tolerance=.00001,max_iter=10000) {
  conv=FALSE
  iter=0
  while (conv==FALSE) {
    init = init - f(init)/fprime(init)
  if(abs(f(init)) <= tolerance){
```

```
    conv=TRUE
  return(init)
  }
    iter=iter+1
    if(iter>max_iter){
      return(NA)
    }
  }
  }

#root 1
newton_root(f, fprime, init=-1)
```

```
## [1] -0.8622336
```

```
#root 2
newton_root(f, fprime, init=1)
```

```
## [1] 0.7748524
```

```
#root 3
newton_root(f, fprime, init=0.5)
```

```
## [1] -0.221411
```

```
#root 4
newton_root(f, fprime, init=0.15)
```

```
## [1] 0.2356729
```

In the program above, if my function fails to converge, it will return NA value.

Part b.
Here, I created a grid of points which contains all roots and used each point as a initial value to run the newton method algorithm.

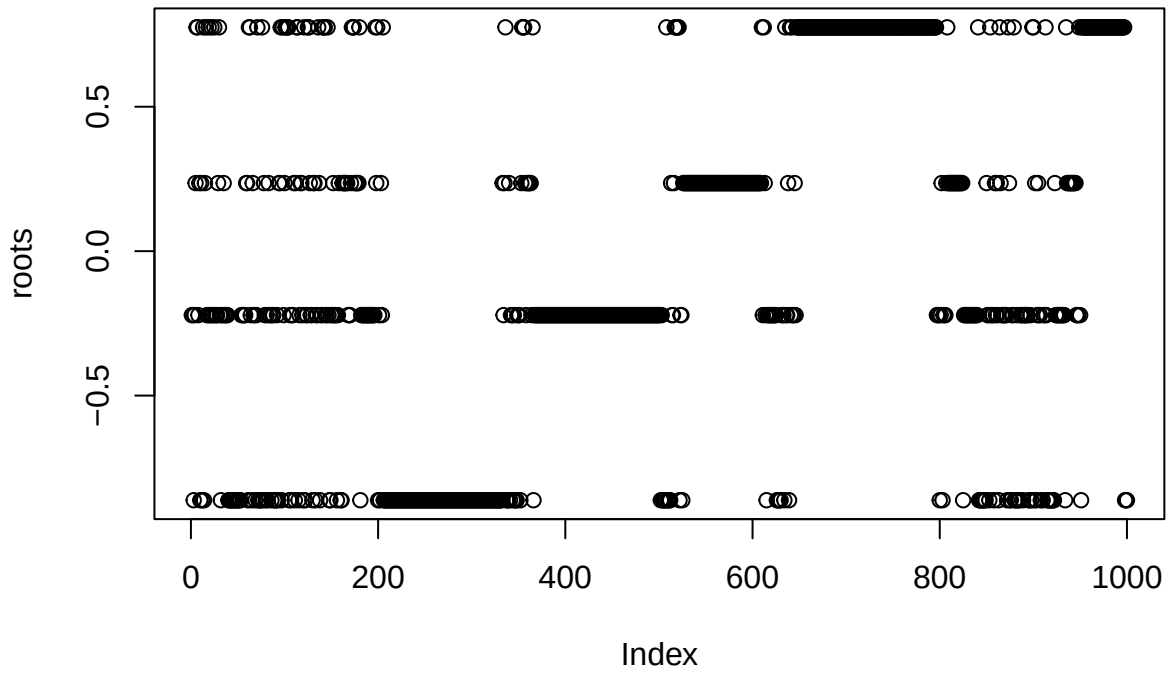
```
xrange=seq(-2,2,length.out=1000)
start=Sys.time()
roots=sapply(xrange, FUN = newton_root)
end=Sys.time()

#time taken
print(end-start)
```

```
## Time difference of 0.1431751 secs
```

```
summary(roots)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -0.8622 -0.8622 -0.2214 -0.0458  0.7749  0.7749       2
```

```
plot(roots)
```



Notice in the graph that all the points from grid returns one of the 4 roots.

Problem 4

In this problem, I have used gradient descent method to calculate regression coefficients of linear regression model. **Part a.**
Program to calculate the regression coefficients using gradient descent method is as following.

```
#data
x<-seq(from=0.5,to=10,by=0.5)
y1 <- 1+2*x+rnorm(60)
x1=c(x,x,x)

gradientDesc <- function(beta0,beta1, x=x1, y=y1,step_size=1e-5,tolerance=1e-7, max_iter=1000000) {
  n=length(x)
  yhat = beta0+beta1*x
  MSE=sum((y - yhat) ^ 2) / (n-2)
  conv = F
  iter = 0
  while(conv == F) {
    ## Implement the gradient descent algorithm
    beta1_new <- beta1 - step_size * ((1 / (n-2)) * (sum((yhat - y) * x)))
    beta0_new <- beta0 - step_size * ((1 / (n-2)) * (sum(yhat - y)))
```

```
    beta1 <- beta1_new
    beta0<- beta0_new
    yhat <-beta0+beta1*x
    MSE_new <- sum((y - yhat) ^ 2) / (n-2)
    if(MSE - MSE_new <= tolerance) {
      a=c(beta0,beta1)
      conv= T
      return(a)
    }
    MSE=MSE_new
    zz=c(beta0,beta1)
    iter = iter + 1
    if(iter > max_iter) {

      conv = T
      return(NA)
    }
  }
}
#Calculate betas using given initial values
gradientDesc(beta0 = 1,beta1 = 0)
```

```
## [1] 1.149982 2.008414
```

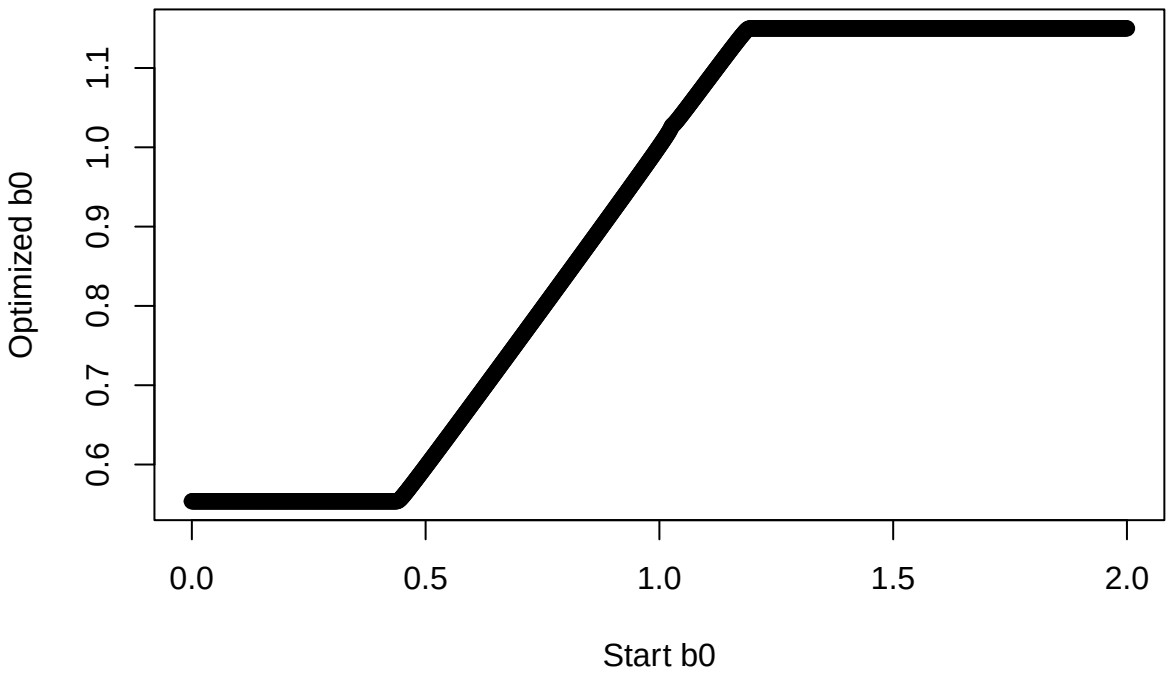
Part b.
Stopping rule for the above code is based on Mean Squared Error(MSE). When MSE in two consecutive iteration are very close to each other i.e smaller than tolerance value, then we stop the loop and return the calculated β_1 and β_0. I have used $\beta_0 = 1 \text{and} \beta_1 = 0$ as an initial value arbitrarily.

Part c.
I created a grid of β_1 and β_0 and used apply function to apply gradient descent function to a pair of β_1 and β_0. I wrote following program.

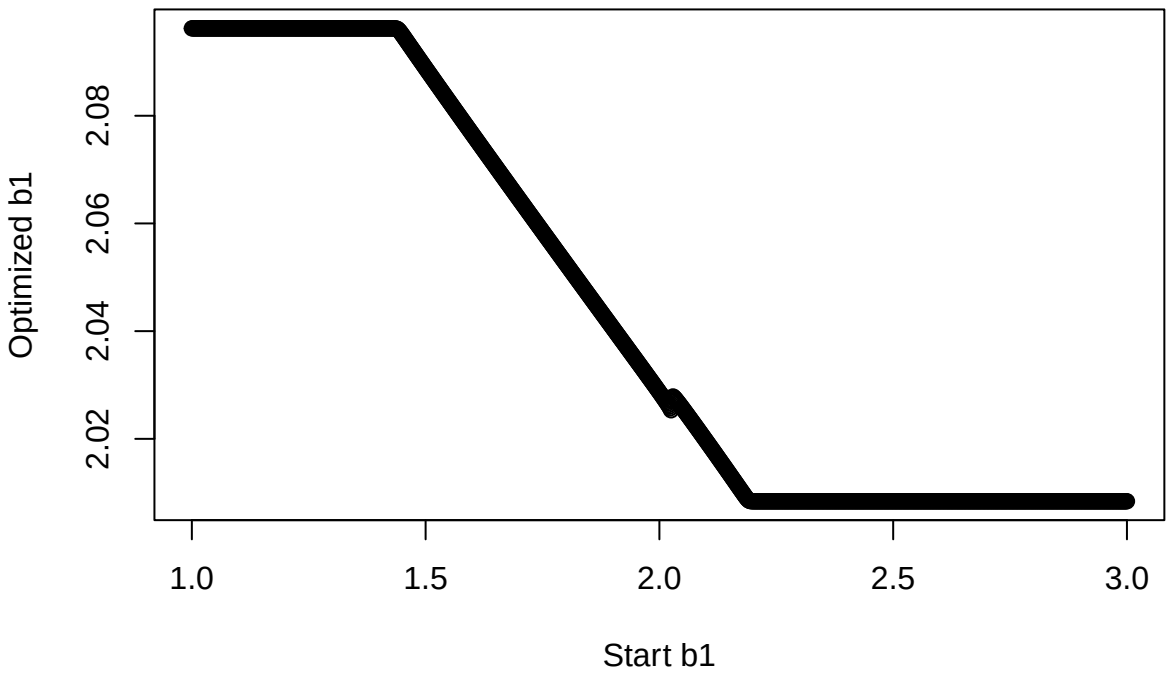
```
beta0range=seq(0,2,length.out=1000)
beta1range=seq(1,3,length.out=1000)

c=t(mapply(gradientDesc,beta0range,beta1range))
```

Part d.
Based on the calculations from c, following is the graphs of start β vs optimized β.

```
optimizedbeta0=c[,1]
optimizedbeta1=c[,2]
plot(beta0range,optimizedbeta0,xlab="Start b0",ylab="Optimized b0")
```




```
plot(beta1range,optimizedbeta1,xlab="Start b1",ylab="Optimized b1")
```


Optimized b1
2.02
2.04
2.06
2.08
1.0
1.5
2.0
2.5
3.0
Start b1