HW5_kexinx

Kexin Xie

11/10/2021

Problem 2

Part a

On the one hand, he did not set different value of seed for each loop, because 'set.seed(666)' makes sampling function generate the same random numbers in every single loop. Thus, we should change the 'set.seed(666)' to 'set.seed(666+i)', or, we can use 'clusterSetRNGStream'.

On the other hand, he set a Boot_time but use Boot instead of Boot_time in his remain code.

Last but not least, the data 'df08' he used in the bootstrap function did not contain the variables which he called to do reression. The colnames name should be changed into 'logapple08' and 'logrm08'.

```
#1)fetch data from Yahoo
#AAPL prices
apple08 <- getSymbols('AAPL', auto.assign = FALSE, from = '2008-1-1', to =
"2008-12-31")[,6]
#market proxy
rm08<-getSymbols('^ixic', auto.assign = FALSE, from = '2008-1-1', to =
"2008-12-31")[,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]

#bootstrap
df08<-cbind(logapple08,logrm08)
colnames(df08)<-c('logapple08','logrm08')  #add this line to change the colname name
Boot_times=1000
sd.boot=rep(0,Boot_times)
for(i in 1:Boot_times){
set.seed(i+666)  #change set.seed(666) to set.seed(i+666)
bootsample=sample(nrow(df08), size = 251, replace = TRUE)
# nonparametric bootstrap
bootdata=df08[bootsample,]
sd.boot[i]= coef(summary(lm(logapple08~logrm08, data = bootdata)))[2,2]
}

## check bootstrap result
knitr::kable(head(sd.boot),format = "markdown", caption="Bootstrap result of estimated SD")
```

Table 1: Bootstrap result of estimated SD

x
0.0642324
0.0500119
0.0579390
0.0577434
0.0510747
0.0598188

Part 3

First we should deal with the data wrangling.

```
url<-'http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat'
sensory_raw<-read.table(url, header=F, skip=2, fill=T, stringsAsFactors = F)

sensory_tidy1<-sensory_raw%>%t()%>%c()%>% matrix(nrow=10,byrow=TRUE)%>%
                 as.data.frame()%>%select_if(~ !anyNA(.))

sensory_tidy<- sensory_tidy1 %>%
  rename('Item'=V1,'1.'=c(V2,V3,V4),'2.'=c(V5,V6,V7),'3.'=c(V8,V9,V10),'4.'=c(V11,V13,V14),'5.'=c(V15,V
  gather(key=Operator,value=y,-Item) %>% separate(Operator, c("Operator", "Times"))

knitr::kable(head(sensory_tidy),format = "markdown", caption="Tidy Sensory Data")
```

Table 2: Tidy Sensory Data

Item	Operator	Times	y
1	1	1	4.3
2	1	1	6.0
3	1	1	2.4
4	1	1	7.4
5	1	1	5.7
6	1	1	2.2

Then, we can create the bootstrap of regression.

```
x<-as.numeric(sensory_tidy$Operator)
y<-as.numeric(sensory_tidy$y)
df<-as.data.frame(cbind(x,y))
  #desired number of bootstrap samples to take
bootstrapfun<-function(B){
boot.coef.intercept<-NULL
boot.coef.slope<-NULL
for (i in 1:B){
  sample<-c()
  for (k in 1:5){
  set.seed(i+k+555)
  sample[(1+30*(k-1)):(30*k)]<-sample((1+30*(k-1)):(30*k),30,replace = TRUE)
  }
  boot.sample = as.data.frame(df[sample, ])
```

```
  boot.model2<-lm(y~x,data = boot.sample)
  boot.coef.intercept<-c(boot.coef.intercept, boot.model2$coefficients[1])
  boot.coef.slope<-c(boot.coef.slope, boot.model2$coefficients[2])
}
#par(mfcol=c(1,2))
#hist(boot.coef.intercept,cex=0.8)
#hist(boot.coef.slope,cex=0.8)
beta0<-mean(boot.coef.intercept)
beta1<-mean(boot.coef.slope)
return(c(beta0,beta1))
}

result<-data.frame(beta0=bootstrapfun(100)[1],beta1=bootstrapfun(100)[2],time=system.time({bootstrapfun

knitr::kable(result,format = "markdown", caption="Bootstrap Result")
```

Table 3: Bootstrap Result

beta0	beta1	time
4.596583	0.01845	0.08

Problem 3

Part a

As shown in the plot, there are 4 roots for this function.

```
fun<-function(x){
  x=x
  y=3^x-sin(x)+cos(5*x)+x^2-1.5
  dy=3^x*log(3)-cos(x)-5*sin(5*x)+2*x
  J=eval(y)/eval(dy)
  list(x=x,y=y,dy=dy,J=J)
}

ep=1e-5
it_max<-1000

Newtonfun<-function(x0){
  index=0;k=0;x=x0
  norm=abs(fun(x)$J)
  while(norm>=ep && k<=it_max){
    k=k+1
    if(eval(fun(x)$dy)==0) {
      index=1
      warning('slope is zero: no futher improvement possible')
      break}
     x=x-fun(x)$J
  norm=abs(fun(x)$J)
  }
  #list(root=x,it=k,index=index)
  return(x)
```

```
}

print(paste0('One of the roots of this function is: ',Newtonfun(1)))
```

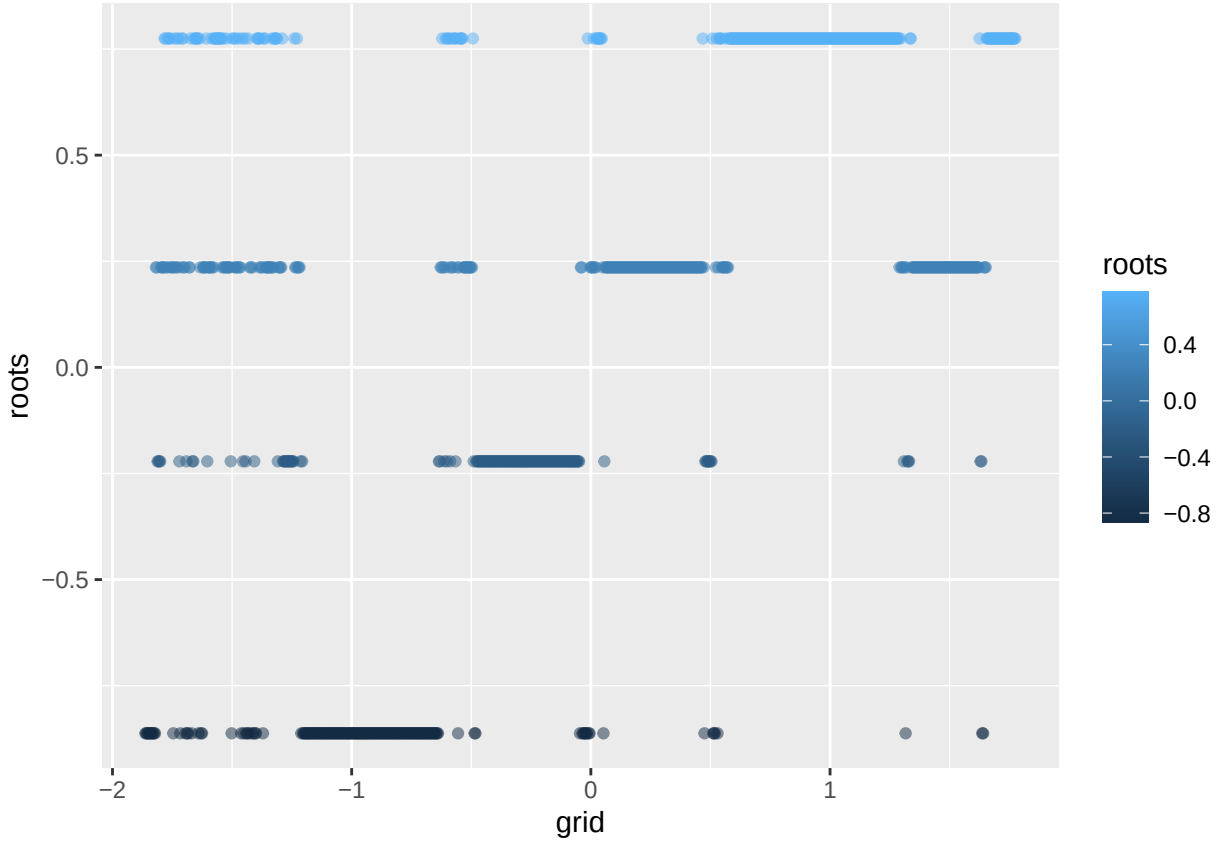
```
## [1] "One of the roots of this function is: 0.774853959051899"
```

Part b

```
grid<-seq(-1+Newtonfun(-1),1+Newtonfun(1),length.out=1000)
roots<-sapply(grid,Newtonfun)
system.time({roots<-sapply(grid,Newtonfun)})
```

```
##    user  system elapsed
##    0.11    0.00    0.11
```

```
ggplot(mapping = aes(x=grid,y=roots,color=roots))+geom_point(alpha = 0.5)
```



Problem 4

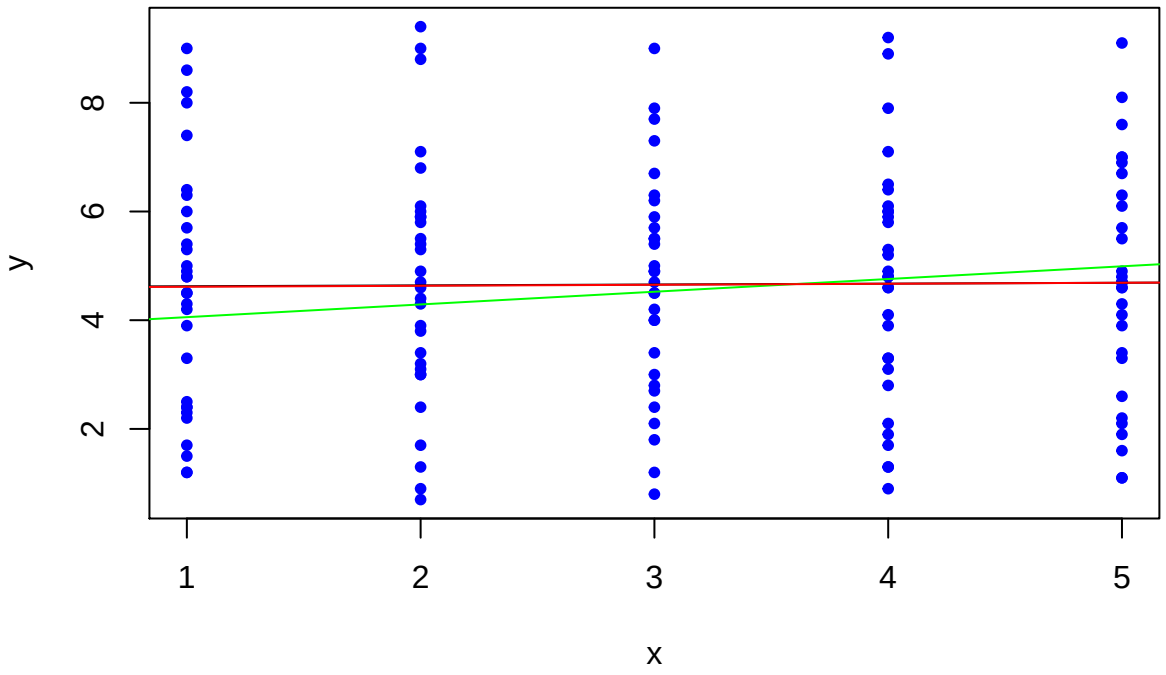
Part a

The basic idea of fitting a simple linear model is the least squared estimation, which means we should calculate the optimized parameters to minimize the sum of squared error. Thus, we can consider using the gradient descent method to solve this optimization problem.

```r
x<-as.numeric(sensory_tidy$Operator)
y<-as.numeric(sensory_tidy$y)
df<-as.data.frame(cbind(x,y))
X<-cbind(1,matrix(x))

msefun<-function(x,y,beta){
  sum((y-x%*%beta)^2)/(2*length(y))
}

learn_rate<-0.01
it_max<-1000
ep<-1e-5

gradientDesc<-function(beta0){
  k=0;
  beta=matrix(beta0,nrow=2);
  while(k<=it_max){
  k=k+1
  delta<-t(X) %*% (X %*% beta - y) / length(y)
  mse<-msefun(X,y,beta)
  beta<-beta-learn_rate*delta
  mse_new<-msefun(X,y,beta)
  norm=abs(mse-mse_new)
   if(!isTRUE(norm) && norm < ep){
      break
    }
  }
  #return(paste("Optimal intercept:", beta[1,1], "Optimal slope:", beta[2,1]))
  return(beta)
}

gradbeta<-gradientDesc(c(0,0))
paste("Optimal intercept:", gradbeta[1,1], "Optimal slope:", gradbeta[2,1])
```

```
## [1] "Optimal intercept: 3.82246851307868 Optimal slope: 0.233764281815939"
```

```r
mod<-lm(y~x,data = df)
b0 <- mod$coefficients[1]
b1 <- mod$coefficients[2]
# check answer
plot(x,y,col='blue',pch=20)
abline(b0,b1,col='black')
abline(gradbeta[1,1],gradbeta[2,1],col='green')
abline(bootstrapfun(100)[1],bootstrapfun(100)[2],col='red')
```



Part b

My stopping rule is that if the tolerance threshold is met or the number of iterations exceeds a certain number, the algorithm will return the latest estimate regardless of proximity.

If the true value of the parameter is known, then the algorithm will stop when it finds that the values of β_0 and β_1 are close enough to the stated values. A potential problem may be that the algorithm has found a local minimum instead of a global minimum.

For the guess of the initial value, I will use the true value of the parameter.

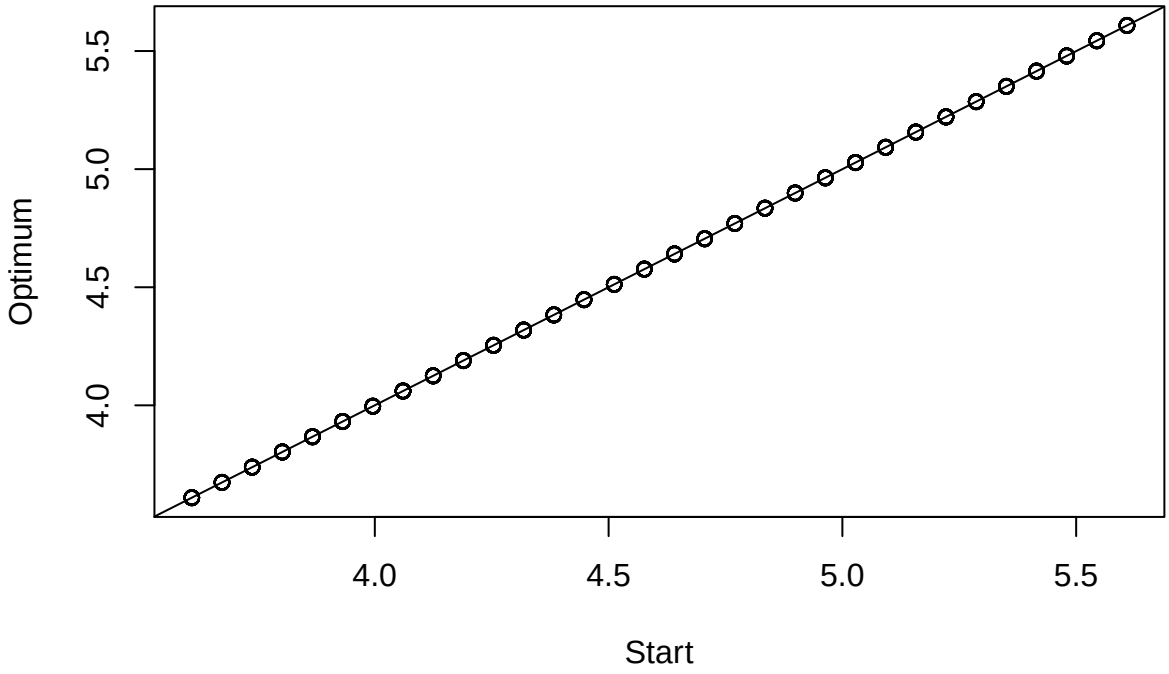
Part c

```
learn_rate<-1e-7
it_max<-5000
ep<-1e-9

grid0 <- seq(b0  - 1, b0  + 1, length.out=32)
grid1 <- seq(b1  - 1, b1  + 1, length.out=32)

grid_new<-expand.grid(grid0, grid1)

result4c<-apply(grid_new,1,gradientDesc)
```

Part d

From the figure below, we find that the predicted value and the true value have a strong linear relationship with the slope close to 1, which means that the algorithm can make the predicted value smoothly close to the true value. This method seems to be suitable for approximating the real parameters using observed samples, although it seems to require considerable computational time.

```r
betahat0<-result4c[1,]
betahat1<-result4c[2,]
beta0_new<-as.numeric(grid_new[,1])
beta1_new<-as.numeric(grid_new[,2])
plot(beta0_new,betahat0, xlab="Start", ylab="Optimum")
abline(lm(betahat0~beta0_new)$coefficient[1],lm(betahat0~beta0_new)$coefficient[2])
```




```r
plot(beta1_new,betahat1, xlab="Start", ylab="Optimum")
abline(lm(betahat1~beta1_new)$coefficient[1],lm(betahat1~beta1_new)$coefficient[2])
```