HW5_ssoon

Samuel Soon

11/8/2021

```
#
# library(quantreg)
# library(quantmod)
# # #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]
#
# #create df from AAPL returns and market returns
# 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(AAPL.Adjusted~IXIC.Adjusted, data = bootdata)))[2,2]
# }
# sd.boot
```

Problem 2

a

The code uses the wrong variable name while bootstrapping; the linear model should be comparing AAPL.Adjusted~IXIC.Adjusted.

b

```r
sensory  <- read.delim("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat",
                      header = TRUE, sep="\t")
sensory <- sensory[2:nrow(sensory),]

sensory <- separate(sensory, X, into = c("1", "2", "3", "4", "5", "F"), sep = " ", convert=TRUE)

sensory[!is.na(sensory$F),][1:5] <-sensory[!is.na(sensory$F),][2:6]
sensory  <- sensory[1:5]
sensory <- melt(sensory)
```

```
## No id variables; using all as measure variables
```

```r
sensory$variable <- as.numeric(sensory$variable)

sensory_ops <- split(sensory, sensory$variable)

boot <- c()


system.time(
  {
c<-1

for(op in sensory_ops){

  bootdata <- c()
  for(i in 1:100){

    bootdata <- c(bootdata, sample(op$value,nrow(op), replace=TRUE))


    boot <- c(boot, mean(bootdata))

  }
  print(paste("Estimate for operator", c, ":", mean(boot)))
  c<-c+1
}
}
)
```

```
## [1] "Estimate for operator 1 : 4.55609867956575"
## [1] "Estimate for operator 2 : 4.79179715277645"
## [1] "Estimate for operator 3 : 4.62056940035552"
## [1] "Estimate for operator 4 : 4.77543841712621"
## [1] "Estimate for operator 5 : 4.67538736650596"
```

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

```r
# print("Bootstrap set:")
# boot
```

Problem 3

a

There are 4 roots in the given graph.

My function will return the latex calculated x_1 value if it fails to converge.

```
tol = 1e-5

f <- function(x){
  if(x > 100){
  x <- as.brob(x)
  }
  return((3^x- sin(x) +  cos(5*x) + x^2 - 1.5)/(3^x*log(3)-cos(x)-sin(5*x)*5+2*x))
}

df  <- function(x){



}

newton <- function(x0, iter = 0){
  if(iter >= 100){
   # print(paste("Max number of iterations reached. Current value:", x0))
    return(x0)
  }else{
    x1  <- x0- f(x0)
    #if(is.na(x1)){print(paste(x0, iter))}
    if(abs(x1-x0) <= 1e-5){
      return(x1)
    }
    else{
      return(newton(x1, iter+1))
    }
  }
}

newton(0)
```

```
## [1] -0.862233
```

b

```
vec <- seq(-3,2.5,by=5.5/999)



system.time(sapply(vec, newton, iter=0))
```

```
##    user  system elapsed
##    5.22    0.00    5.38
```

Problem 4

#a

```r
mse <- function(y,yhat, n){
      return(sum((y - yhat)^2)/n)
  }



grad <- function(dat, start1, start2, step, tol, it, n, b) {
  b1  <- start1
  b0 <- start2
  # mse_prev <- 100
  diff <- 100
  i <- 0
  x<- dat[1]
  y <- dat[2]
  while( i < it) {
    yhat <- b1*x + b0
    #mse1 <- mse(y,yhat,n)

    b1 <- b1 - step * sum((yhat - y) * x)/n
    b0 <- b0 - step * sum(yhat - y)/n
    yhat2 <- b1 * x + b0
    #mse2 <- mse(y,yhat2,n)

    #print(paste(mse1, mse2))

    diff <- abs(sum((y - yhat)^2)/n - sum((y - yhat2)^2)/n)

    if(!isTRUE(diff) && diff < tol){
      break
    }
    #print(is.na(diff))
    i<-i+1
  }

  if(b== 0){
    return(b0)
  }
  else{
    return(b1)
  }
}

#grad(sensory, 0.05, 4, 1e-7, 1e-9, 1, nrow(sensory))
```

b

My stopping rule is that the algorithm returns the latest estimates regardless of proximity if either the tolerance threshold is met, or the number of iterations exceed a certain number. If the true values of the parameters were known, then I would stop when the algorithm finds values of b1 and b0 close enough to said values. A potential problem could be that variance within data means that some samples will not fit the true values well, or that the algorithm finds a local minimum instead of a global minimum. For a guess of inital value, I would use the true values of parameters.

C

Using larger step/tolerance to reduce runtime on my laptop

```r
mod <- lm(value ~., sensory)

summary(mod)
```

```
##
## Call:
## lm(formula = value ~ ., data = sensory)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.861 -1.684  0.048  1.335  4.796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.81367    0.41170  11.692   <2e-16 ***
## variable    -0.05233    0.12413  -0.422    0.674
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.15 on 148 degrees of freedom
## Multiple R-squared:  0.001199,   Adjusted R-squared:  -0.005549
## F-statistic: 0.1777 on 1 and 148 DF,  p-value: 0.6739
```

```r
s<- 1e-7
t <- 1e-9

b0 <- mod$coefficients[1]
b1 <- mod$coefficients[2]

range0 <- seq(b0  - 1, b0  + 1, length.out=1000)
range1 <- seq(b1  - 1, b1  + 1, length.out=1000)

g <- expand.grid(range0, range1)

cores <- detectCores() - 1
cores <- max(1, detectCores() - 1)

cl <- makeCluster(cores)


map0 <- clusterMap(cl, grad, dat=sensory, range1, range0, step=1e-7, tol=1e-9, it=5e+4, n=nrow(sensory)

map1 <- clusterMap(cl, grad, dat=sensory, range1, range0, step=1e-7, tol=1e-9, it=5e+4, n=nrow(sensory)


stopCluster(cl)
```

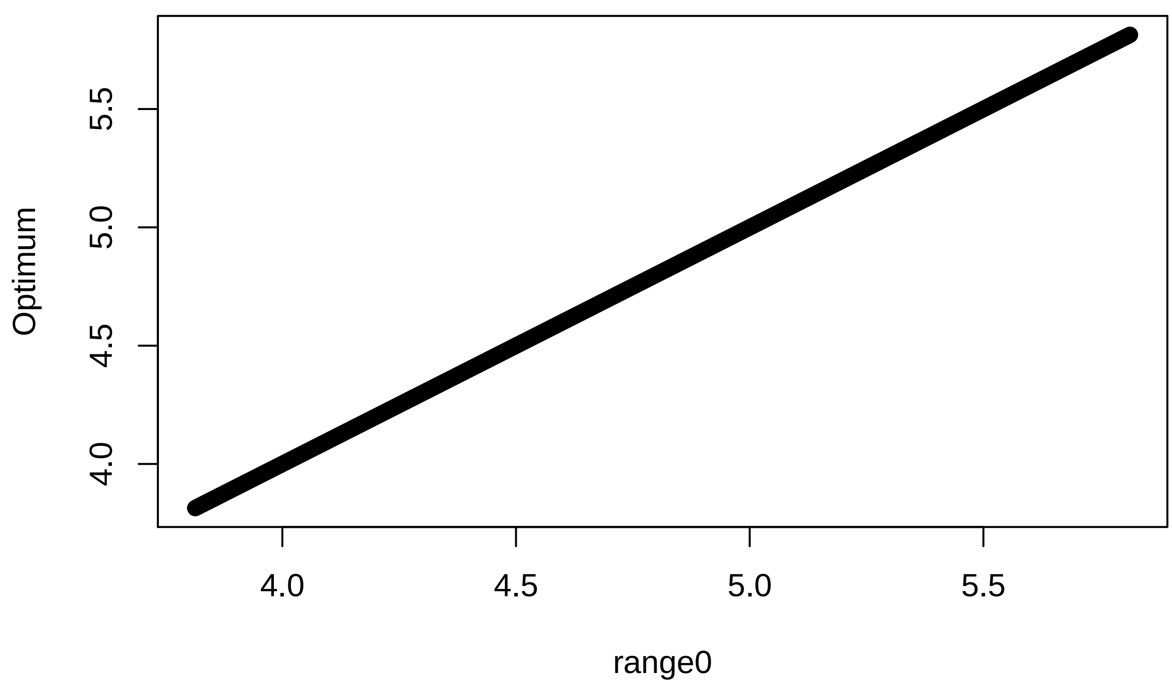
```r
hat0 <- unlist(map0)
hat1 <- unlist(map1)
```

d

From the given plots, it seems that the algorithm did succeed in smoothing the predicted β values to be closer to the true value. This method looks good for approximating the true parameters using an observed sample, though it seems to require quite a bit of computation time.

```r
plot(hat0~ range0, x_lab="Start", ylab="Optimum")
```

```
## Warning in plot.window(...): "x_lab" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "x_lab" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "x_lab" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "x_lab" is not a
## graphical parameter

## Warning in box(...): "x_lab" is not a graphical parameter

## Warning in title(...): "x_lab" is not a graphical parameter
```




```r
plot(hat1~ range1, x_lab="Start", ylab="Optimum")
```

```
## Warning in plot.window(...): "x_lab" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "x_lab" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "x_lab" is not a
## graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "x_lab" is not a
## graphical parameter
## Warning in box(...): "x_lab" is not a graphical parameter
## Warning in title(...): "x_lab" is not a graphical parameter
```