Homework 5

Mengkun Chen (mengkun21@vt.edu)

```
library(data.table)
```

Problem 2

Part a

```
sd.boot[i]= coef(summary(lm(logapple08~logrm08, data = bootdata)))[2,2]
```

The variable names are not `logapple08` and `logrm08` in `df08`. The above line should be changed as

```
sd.boot[i]<-coef(summary(lm(AAPL.Adjusted ~ IXIC.Adjusted, data = bootdata)))[2,2]
```

Part b

- Data wrangling

```
sensory<-read.table("http://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat",
                    fill = TRUE, skip = 2, header = FALSE)
colnames(sensory)<-c("Item", 1:5)
sensory[-seq(1,30, by = 3),2:6]<-sensory[-seq(1,30, by = 3),1:5]
sensory[-seq(1,30,by = 3),1]<-rep(1:10, each = 2)
opt<-data.frame("operator" = rep(1:5, each = nrow(sensory)),
                "y" = c(sensory[,2], sensory[,3], sensory[,4], sensory[,5], sensory[,6]))
```

- Bootstrap

```
B<-100
beta<-matrix(0, nrow = B, ncol = 2)

system.time(
  for (i in 1:B) {
    s1<-sample(1:30, size = 30, replace = TRUE)
    s2<-sample(31:60, size = 30, replace = TRUE)
    s3<-sample(61:90, size = 30, replace = TRUE)
    s4<-sample(91:120, size = 30, replace = TRUE)
    s5<-sample(121:150, size = 30, replace = TRUE)
    opt.boot<-opt[c(s1,s2,s3,s4,s5),]
    m<-lm(y ~ operator, data = opt.boot)
    beta[i,]<-coefficients(m)
})
```

```
##    user  system elapsed
##    0.08    0.02    0.10
```

```
apply(beta, 2, mean)
```

```
## [1]  4.778437 -0.043330
```

Problem 3

Part a

From the plot of the function, $f(x) = 0$ has 4 roots.

```
newton.root<-function(x, tol = 1e-5, max.iter){
  iter<-0; c<-x
  f<-3^x-sin(x)+cos(5*x)+x^2-1.5
  while (abs(f) >= tol & iter < max.iter) {
    df<-log(3)*3^c-cos(c)-5*sin(5*c)+2*c
    if (df == 0) {
      c<-c+0.01
    } else{
      c<-c-f/df
    }
    f<-3^c-sin(c)+cos(5*c)+c^2-1.5
    iter<-iter+1
  }
  if (iter < max.iter) {
    return(c)
  } else{
    if (abs(f) < tol) {
      return(c)
    } else{
      return(Inf)
    }
  }
}
```

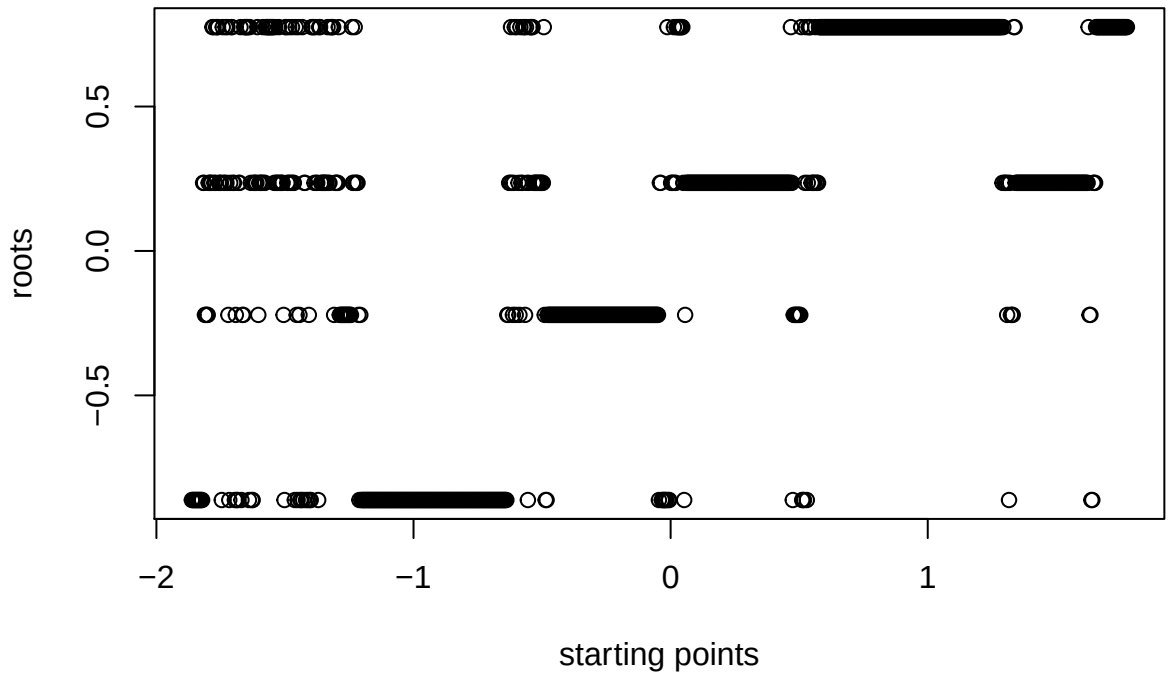
The function will retrun ∞ when it fails to converge.

Part b

```
lb<-newton.root(-1, max.iter = 100)-1
ub<-newton.root(1, max.iter = 100)+1
x.vec<-seq(lb, ub, length.out = 1000)
r.vec<-sapply(x.vec, newton.root, max.iter = 100)
system.time(sapply(x.vec, newton.root, max.iter = 100))
```

```
##    user  system elapsed
##       0       0       0
```

```r
plot(x.vec, r.vec, xlab = "starting points", ylab = "roots")
```



Problem 4

```r
x<-opt$operator
y<-opt$y
gradiant.descent<-function(beta, tol = 1e-5, step, max.iter){
  df1<--2*sum(y-beta[1]-beta[2]*x)
  df2<--2*sum(x*(y-beta[1]-beta[2]*x))
  f<-sum((y-beta[1]-beta[2]*x)^2)
  c<-beta
  d<-Inf
  iter<-0
  while (abs(d) >= tol & iter < max.iter) {
    f<-sum((y-c[1]-c[2]*x)^2)
    c[1]<-c[1]-step*df1
    c[2]<-c[2]-step*df2
    df1<--2*sum(y-c[1]-c[2]*x)
    df2<--2*sum(x*(y-c[1]-c[2]*x))
    d<-f-sum((y-c[1]-c[2]*x)^2)
    iter<-iter+1
  }
  return(c)
```

```
}

gradiant.descent(c(4,-0.05), tol = 1e-5, step = 0.00001, max.iter = 1e+6)
```

```
## [1]  4.79467307 -0.04707241
```

Part b

Stopping rule is the difference between two iteration is smaller than tolerance or the number of iteration is greater than maximum iteration.

If the stopping rule include knowledge of the true value, the function will find overall minimum.

It's not a good way to run this algorithm. The result depends on initial guess of β and may find local minimum.

A best guess at an initial value is close to true minimum.

Part c

```
beta0<-seq(2,6, length.out = 50)
beta1<-seq(-1,0, length.out = 20)
init<-expand.grid(beta0, beta1)
beta<-apply(init, MARGIN = 1, gradiant.descent, tol = 1e-5, step = 1e-4, max.iter = 5e+6)
```

PS. My computer will be so hot that I can't set tol and step too small.

Part d

```
par(mfrow = c(1,2))
plot(init[,1], beta[1,], xlab = "start", ylab = "optimum", main = expression(beta[0]))
plot(init[,2], beta[2,], xlab = "start", ylab = "optimum", main = expression(beta[1]))
```