HW5_kmg0122

Mingang Kim

2021 11 8

```
library(dplyr)
```

```
## Warning:    'dplyr'  R   4.1.1

##
##         : 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## Warning:    'tidyverse'  R   4.1.1

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v stringr 1.4.0
## v tidyr   1.1.3     v forcats 0.5.1
## v readr   2.0.1

## Warning:    'ggplot2'  R   4.1.1

## Warning:    'tibble'  R   4.1.1

## Warning:    'tidyr'  R   4.1.1

## Warning:    'readr'  R   4.1.1

## Warning:    'purrr'  R   4.1.1

## Warning:    'forcats'  R   4.1.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

2.

(a)

```
library(quantmod)
```

```
## Warning:    'quantmod'  R   4.1.1

##          : xts

## Warning:    'xts'  R   4.1.1

##          : zoo

## Warning:    'zoo'  R   4.1.1

##
##         : 'zoo'

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

##
##         : 'xts'

## The following objects are masked from 'package:dplyr':
##
##     first, last

##          : TTR

## Warning:    'TTR'  R   4.1.1

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

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

```r
#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_times=1000
sd.boot=rep(0,Boot_times)
for(i in 1:Boot_times){
# nonparametric bootstrap
bootdata=df08[sample(nrow(df08), size = 251, replace = TRUE),]
sd.boot[i]= coef(summary(lm(logapple08~logrm08, data = bootdata)))[2,2]
}
```

The problem of the given code is that it did not use the right variable name. So when the data is made by "cbind", the variable names are "AAPL.Adjusted" and "IXIC.Adjusted". However, the bootstrap code used logapple08~logrm08 for regression and it results in using original data instead of bootstrapped data. To correct the code, this should be modified like below. In addition, it used Boot in the loop which was not mentioned, it was supposed to be Boot_times, so I modified that part too.

```r
bootdata=df08[sample(nrow(df08), size = 251, replace = TRUE),]
sd.boot[i]= coef(summary(lm(AAPL.Adjusted~IXIC.Adjusted, data = bootdata)))[2,2]
```

(b)

```r
data.2<-read.csv("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat", header=F, skip=2

data.2.m <- data.2

for(i in 1: (length(data.2[,1])/3)){
  for(j in 2:6){
    data.2.m[3*i-2,j-1]<-data.2.m[3*i-2,j]
  }
}

data.2.m <- data.2.m[,1:5]

colnames(data.2.m) <- c('1','2','3','4','5')
data.2.m <- gather(data.2.m, key = "Operator", value = "y", 1:5)
data.2.m$y <- as.numeric(data.2.m$y)
data.2.m$Operator <- as.numeric(data.2.m$Operator)



data.2.m
```

```
##    Operator   y
## 1         1 4.3
## 2         1 4.3
## 3         1 4.1
## 4         1 6.0
## 5         1 4.9
## 6         1 6.0
## 7         1 2.4
## 8         1 3.9
## 9         1 1.9
## 10        1 7.4
## 11        1 7.1
## 12        1 6.4
## 13        1 5.7
## 14        1 5.8
## 15        1 5.8
## 16        1 2.2
## 17        1 3.0
## 18        1 2.1
## 19        1 1.2
## 20        1 1.3
## 21        1 0.9
## 22        1 4.2
## 23        1 3.0
## 24        1 4.8
## 25        1 8.0
## 26        1 9.0
## 27        1 8.9
## 28        1 5.0
## 29        1 5.4
## 30        1 2.8
## 31        2 4.9
## 32        2 4.5
## 33        2 5.3
## 34        2 5.3
## 35        2 6.3
## 36        2 5.9
## 37        2 2.5
## 38        2 3.0
## 39        2 3.9
## 40        2 8.2
## 41        2 7.9
## 42        2 7.1
## 43        2 6.3
## 44        2 5.7
## 45        2 6.0
## 46        2 2.4
## 47        2 1.8
## 48        2 3.3
## 49        2 1.5
## 50        2 2.4
## 51        2 3.1
## 52        2 4.8
## 53        2 4.5
## 54          2 4.8
## 55          2 8.6
## 56          2 7.7
## 57          2 9.2
## 58          2 4.8
## 59          2 5.0
## 60          2 5.2
## 61          3 3.3
## 62          3 4.0
## 63          3 3.4
## 64          3 4.5
## 65          3 4.2
## 66          3 4.7
## 67          3 2.3
## 68          3 2.8
## 69          3 2.6
## 70          3 6.4
## 71          3 5.9
## 72          3 6.9
## 73          3 5.4
## 74          3 5.4
## 75          3 6.1
## 76          3 1.7
## 77          3 2.1
## 78          3 1.1
## 79          3 1.2
## 80          3 0.8
## 81          3 1.1
## 82          3 4.5
## 83          3 4.7
## 84          3 4.7
## 85          3 9.0
## 86          3 6.7
## 87          3 8.1
## 88          3 3.9
## 89          3 3.4
## 90          3 4.1
## 91          4 5.3
## 92          4 5.5
## 93          4 5.7
## 94          4 5.9
## 95          4 5.5
## 96          4 6.3
## 97          4 3.1
## 98          4 2.7
## 99          4 4.6
## 100         4 6.8
## 101         4 7.3
## 102         4 7.0
## 103         4 6.1
## 104         4 6.2
## 105         4 7.0
## 106         4 3.4
## 107         4 4.0
## 108        4 3.3
## 109        4 0.9
## 110        4 1.2
## 111        4 1.9
## 112        4 4.6
## 113        4 4.9
## 114        4 4.8
## 115        4 9.4
## 116        4 9.0
## 117        4 9.1
## 118        4 5.5
## 119        4 4.9
## 120        4 3.9
## 121        5 4.4
## 122        5 3.3
## 123        5 4.7
## 124        5 4.7
## 125        5 4.9
## 126        5 4.6
## 127        5 2.4
## 128        5 1.3
## 129        5 2.2
## 130        5 6.0
## 131        5 6.1
## 132        5 6.7
## 133        5 5.9
## 134        5 6.5
## 135        5 4.9
## 136        5 1.7
## 137        5 1.7
## 138        5 2.1
## 139        5 0.7
## 140        5 1.3
## 141        5 1.6
## 142        5 3.2
## 143        5 4.6
## 144        5 4.3
## 145        5 8.8
## 146        5 7.9
## 147        5 7.6
## 148        5 3.8
## 149        5 4.6
## 150        5 5.5
```

```r
# bootstrap

set.seed(82)
boot<-function(data, boot.num=100){
  boot.beta <- matrix(0, boot.num, 2)
  for(i in 1:boot.num){
    index1 <- sample(c(1:30), 30, replace = TRUE)
    index2 <- sample(c(31:60), 30, replace = TRUE)
    index3 <- sample(c(61:90), 30, replace = TRUE)
    index4 <- sample(c(91:120), 30, replace = TRUE)
    index5 <- sample(c(121:150), 30, replace = TRUE)
    boot.data <- data[c(index1,index2,index3,index4,index5),]
    fit <- lm(y ~ Operator, boot.data)
    boot.beta[i,] <- fit$coefficients
  }
  boot.beta0 <- mean(boot.beta[,1])
  boot.beta1 <- mean(boot.beta[,2])

  beta<-cbind(boot.beta0, boot.beta1)
  return(beta)
}

system.time(boot(data.2.m))
```

```
##        elapsed
##    0.07    0.00    0.07
```

```
boot(data.2.m)
```

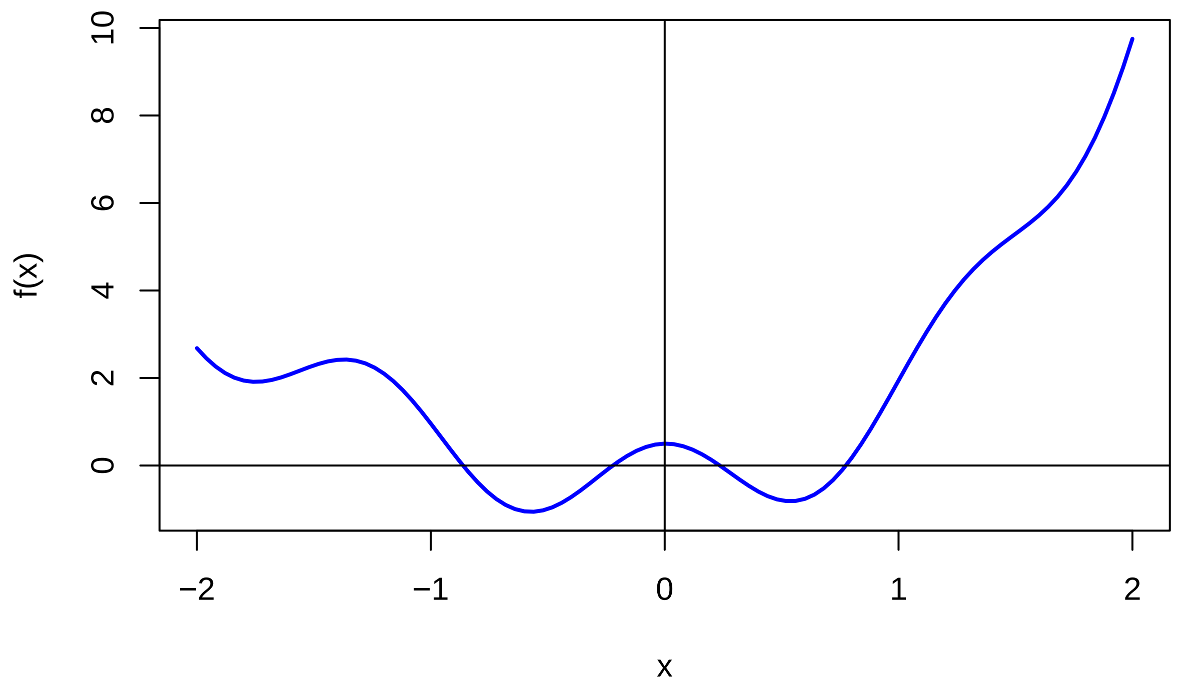
```
##      boot.beta0  boot.beta1
## [1,]    4.82638 -0.06107333
```

3.

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

a.

```
#graph
curve(f, xlim=c(-2,2), col='blue', lwd=2, lty=1, ylab='f(x)')
abline(h=0)
abline(v=0)
```



There are 4 roots.

```
#finding root

library(numDeriv)
```

```
## Warning:    'numDeriv'  R   4.1.1
```

```
newton.raphson <- function(f, a, tol = 1e-5, iter.n = 1000) {

  x0 <- a

  fa <- f(a)
  if (fa == 0.0) {
    return(a)
  }


  for (i in 1:iter.n) {
    dx <- genD(func = f, x = x0)$D[1]
    x1 <- x0 - (f(x0) / dx)

    if (abs(x1 - x0) < tol) {
      root <- x1
      return(root)
    }
    x0 <- x1
  }
  print('It failed to converge')
}

newton.raphson(f, -2)
```

```
## [1] 0.774853
```

(b)

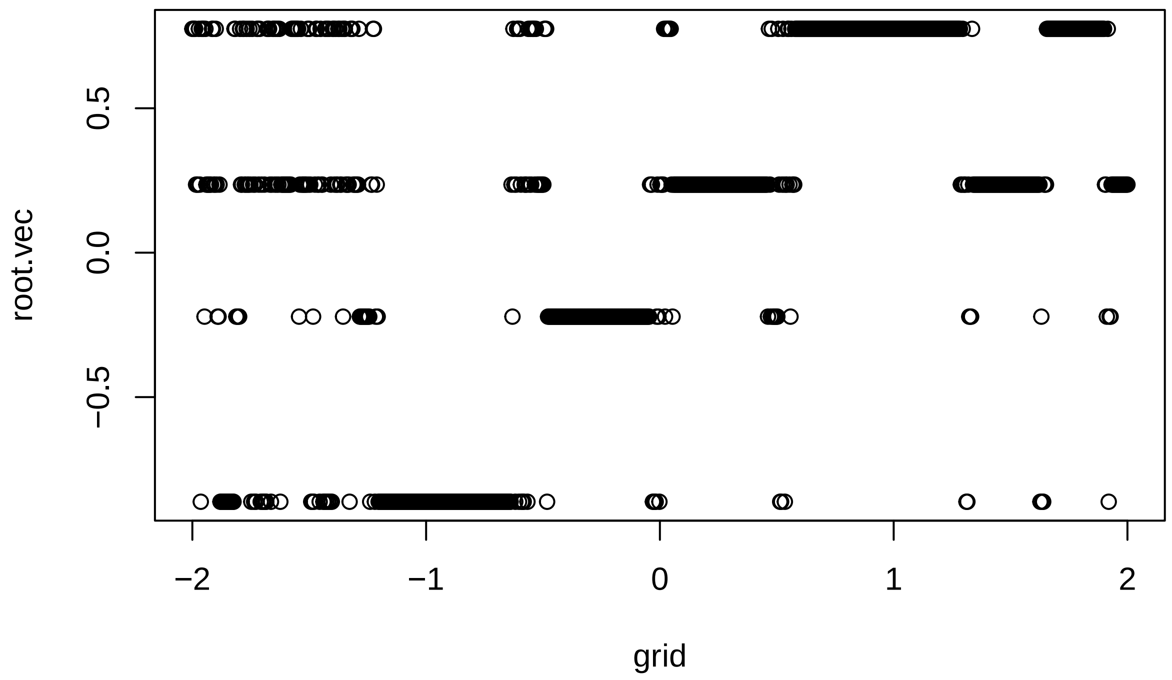
```
grid<-seq(-2,2,length.out=1000)
system.time(sapply(grid, newton.raphson, f=f))
```

```
##        elapsed
##    0.35    0.00    0.34
```

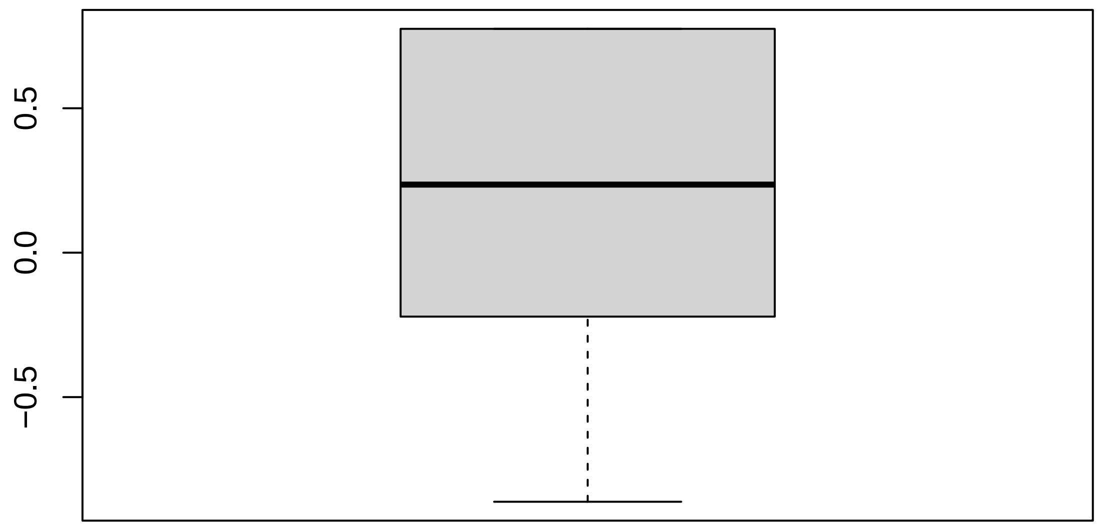
```
root.vec<-sapply(grid, newton.raphson, f=f)

summary(root.vec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.8622 -0.2214  0.2357  0.1224  0.7749  0.7749
```

```
plot(grid, root.vec)
```



```
boxplot(root.vec)
```



4.

(a)

```r
gradDescent<-function(X, y, beta, step.size=1e-1, num_iters=1000, tol=1e-9){
  n <- length(y)
  grad <- rep(0, num_iters)
  beta<-c(beta)
  for(i in 1:num_iters){
    beta <- c(beta - step.size*(1/n)*(t(X)%*%(X%*%beta - y)))
    grad[i] <- sum((X%*%beta- y)^2)/(2*n)
    if(sum(grad^2) <= tol){
      break
    }
  }

  results<-list(beta, grad)
  return(results)
}

X<-model.matrix(lm(y~Operator, data.2.m))
y<-data.2.m$y
```

```r
beta<-rep(0,2)
results <- gradDescent(X, y, beta)
beta <- results[[1]]
cost_hist <- results[[2]]
print(beta)
```

```
## [1]  4.81366649 -0.05233328
```

(b)

My stopping rule is either iteration is completed or sum of squared error is lower than tolerance. Under the assumption that we know the true value, we can replicate the process until sum of squared error is lower than tolerance without setting iteration number to make values more exactly. However, this method might not be a good way to run algorithm because of computation burden. The problem of this algorithm is that it can provide local minimum, rather than local minimum. Thus, initial value should be set as a value that is not in a range of local minimum. Values which is close to true value can be a good guess for initial value.

(c)

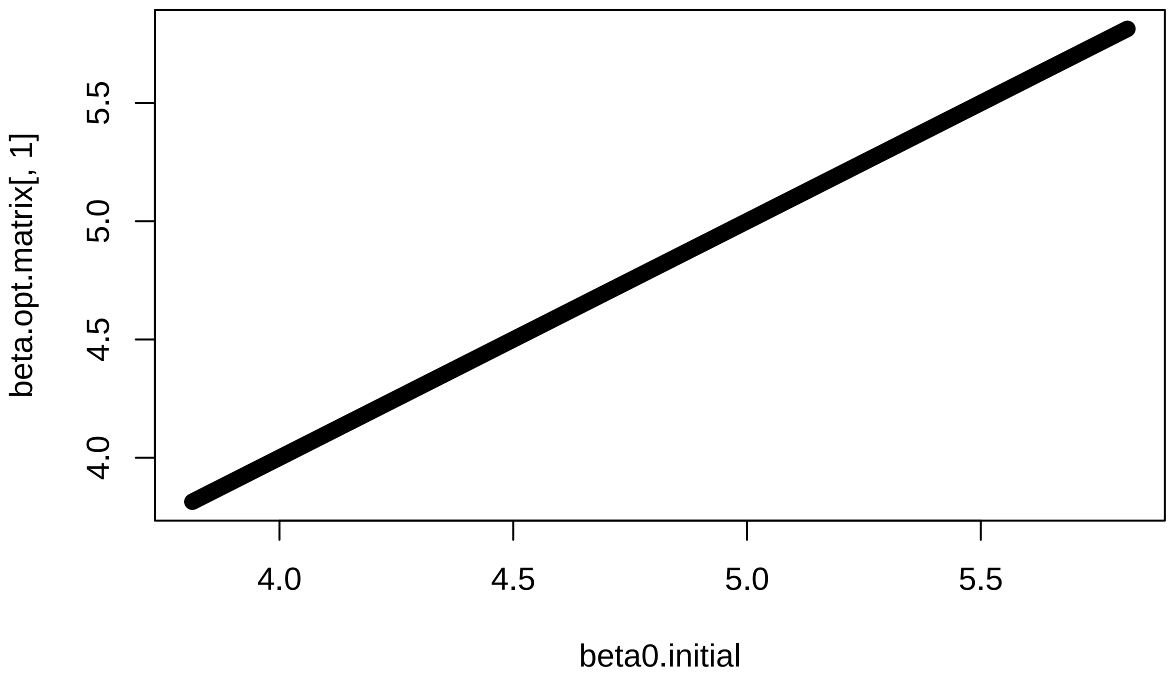
```r
betas<-lm(y~., data.2.m)$coef

betas
```

```
## (Intercept)    Operator
##  4.81366667 -0.05233333
```

```r
beta0.initial<-seq(betas[1]-1,betas[1]+1,length.out=1000)
beta1.initial<-seq(betas[2]-1,betas[2]+1,length.out=1000)

X<-model.matrix(lm(y~Operator, data.2.m))
y<-data.2.m$y

beta.matrix<-cbind(beta0.initial, beta1.initial)


beta.opt.matrix<-matrix(0, nrow = 1000, ncol=2)

for(i in 1:1000){
  beta.opt.matrix[i,]<-gradDescent(X,y, beta.matrix[i,], step.size = 1e-7, tol = 1e-9)[[1]]
}
```

(d)

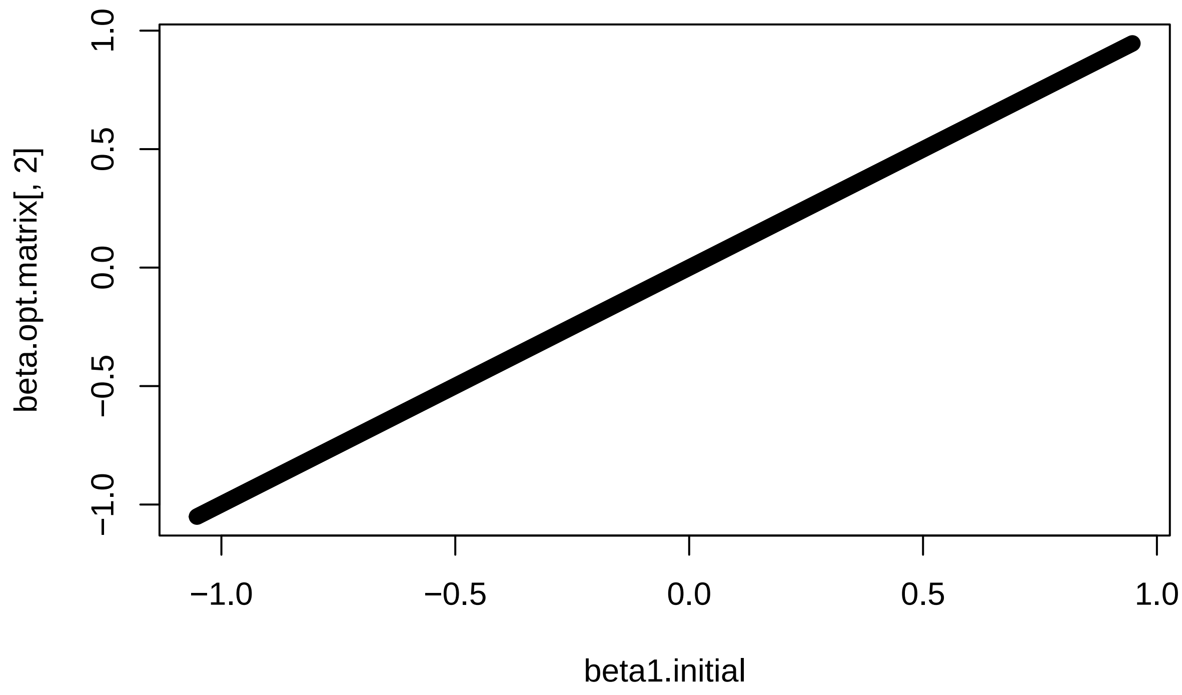
```r
plot(beta0.initial, beta.opt.matrix[,1])
```



```
summary(beta.opt.matrix[,1])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.814   4.314   4.814   4.814   5.313   5.813
```

```
plot(beta1.initial, beta.opt.matrix[,2])
```



```
summary(beta.opt.matrix[,2])
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.05093 -0.55163 -0.05233 -0.05233  0.44697  0.94627
```

When learning rate is very small, this algorithm cannot provide optimal value.