STP598-Assignment 4

Hao Wang
hwang306

Nov 30, 2017

Write estimation functions

```
library(MASS)
library(glmnet)
library(caret)


#ridge
ridge <- function(Data){
  cv.fit <- cv.glmnet(Data$X, Data$y, alpha = 0)
  return(as.numeric(coef(cv.fit, s = cv.fit$lambda.min)))
}

# lasso
lasso <- function(Data){
  require(glmnet)
  cv.fit <- cv.glmnet(Data$X, Data$y, alpha = 1)
  return(as.numeric(coef(cv.fit, s = cv.fit$lambda.min)))
}


# elastic net (fixed alpha)
enet <- function(Data){
require(glmnet)
cvfit <- cv.glmnet(Data$X,Data$y,alpha = 0.5)
return(as.numeric(coef(cvfit,s = cvfit$lambda.min)))
}


# adaptive elastic net (fixed alpha)
aenet <- function(Data){
require(glmnet)
cvfit <- cv.glmnet(Data$X,Data$y,alpha = 0.5)
cvfit2 <- cv.glmnet(Data$X,Data$y, alpha = 0.5, penalty.factor = abs(1/as.numeric(coef(c
return(as.numeric(coef(cvfit2,s = cvfit2$lambda.min)))
}
```

```
# adatpive lad alsso
aladlasso <- function(Data){
 require(quantreg)
        tempcfQR1 <- coef(rq(Data$y~Data$X, .5, method = "lasso",lambda = 1))
              lam = log(length(Data$y)) / abs(tempcfQR1)
        lam[1] = 0
        tempcfQR <- coef(rq(Data$y~Data$X,.5,method = "lasso",lambda = lam))
     return(tempcfQR)
 }
```

1 Data Simulation

I first generating the testing data, a `seed` argument is included to ensure the same replication.

```
genData <- function(n, p, beta, rho){
  require(mvtnorm)
  CovMatrix <- outer(1:p, 1:p, function(x,y) {rho^abs(x - y)})
  X <- mvrnorm(n, rep(0,p), CovMatrix)
  y <- rnorm(n, X %*% beta, 2.2)
  return(list(X = X, y = y))
}
```

2 Evaluate Performance

The first step is the global setting, the size of n is increased to 250. And the rho is a set of values.

```
set.seed(1)
n <- 250    # Number of observations
p <- 220     # Number of predictors included in model
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0,212)) #beta value, 8 nonzeros, 212
rho.set <- c(0.2, 0.4, 0.6, 0.8) #set of rhos
rho <- numeric(length(rho.set))
row.names <- NULL
for (i in 1:p) {
  row.names[i] <- paste("var", i, sep = "")
}
```

2.1 ridge regression

```r
# This matrix stores coefficient data
coef.matrix <- matrix(0, p, length(rho.set))

# ridge
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
coeff <- ridge(data)
coef.matrix[, i] <- coeff[-1] #get rid of first column
}
```

```
## Loading required package: mvtnorm
```

```r
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
coef.ridge <- coef.matrix
```

2.2 lasso regression

```r
#lasso
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
coeff <- lasso(data)
coef.matrix[, i] <- coeff[-1] #get rid of first column
}
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
coef.lasso <- coef.matrix
```

2.3 elastic net with fixed alpha

```r
#elastic net, alpha = 0.5
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
coeff <- lasso(data)
```

```
coef.matrix[, i] <- coeff[-1] #get rid of first column
}
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
coef.enetfixed <- coef.matrix
```

2.4 adatpive lasso

The first stage is ridge regression with lambda chose by CV. In the `glmnet` setting, change the `alpha` value to 0

```
alasso <- function(Data){
require(glmnet)
cvfit <- cv.glmnet(Data$X,Data$y, alpha = 0)
cvfit2 <- cv.glmnet(Data$X, Data$y, penalty.factor = abs(1/as.numeric(coef(cvfit,s = cvf
return(as.numeric(coef(cvfit2,s = cvfit2$lambda.min)))
}

for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
coeff <- alasso(data)
coef.matrix[, i] <- coeff[-1] #get rid of first column
}
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
coef.alasso <- coef.matrix
```

2.5 least sq after adpative lasso

To to the least square estimate, I use the matrix generated from the adaptive lasso stage.

```
lm <- list()
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
varlist <- coef.alasso[,4, drop = FALSE]
varlist <- varlist[which(varlist != 0), , drop = FALSE]
# extract row names, covert that into a vector
a <- row.names(varlist)
a <- gsub(pattern = "var", replacement = "", a)
```

```r
a <- c(as.numeric(a))
data$X <- data$X[ , a]
lm.fit <- lm(y ~ X, data = data)
coef <- coef(lm.fit)[-1]
lm[[paste0("lmcoef", i)]] <- coef
}
lm
```

```
## $lmcoef1
##            X1            X2            X3            X4            X5
##  2.0365820631 -1.9009128625  0.1767436447 -0.0008949999 -0.0344252236
##            X6            X7            X8            X9           X10
##  0.1127527975 -0.2100384369 -0.0393711026  0.2149510160 -0.0079194547
##           X11           X12           X13
## -0.1449219178  0.0015812261  0.1502953745
##
## $lmcoef2
##           X1           X2           X3           X4           X5
##  1.891778864 -1.909832345 -0.203037632 -0.082416676 -0.140559992
##           X6           X7           X8           X9          X10
##  0.042980647 -0.169402217  0.209205642 -0.224555623  0.302620195
##          X11          X12          X13
## -0.040449443 -0.006048323  0.011262760
##
## $lmcoef3
##           X1           X2           X3           X4           X5
##  1.780102382 -1.682585494  0.008474993  0.060086617  0.144871174
##           X6           X7           X8           X9          X10
## -0.061772065 -0.192350980 -0.002180260  0.108562452  0.184060944
##          X11          X12          X13
## -0.520240229  0.039374439 -0.066511381
##
## $lmcoef4
##          X1          X2          X3          X4          X5          X6
##  2.20939543 -1.69437564  0.25559198  0.25367492  0.25885598  0.26197976
##          X7          X8          X9         X10         X11         X12
##  0.21997309  0.34006873 -0.05846405  0.04938860  0.15854421  0.21576521
##         X13
##  0.23083424
```

2.6 adaptive lad lasso

```
#adaptive lad lasso
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
coeff <- aladlasso(data)
coef.matrix[, i] <- coeff[-1] #get rid of first column
}
```

```
## Loading required package: quantreg

## Loading required package: SparseM

##
## Attaching package: 'SparseM'

## The following object is masked from 'package:base':
##
##     backsolve
```

```
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
coef.aladlasso <- coef.matrix
```

2.7 adaptive elastic net with alpha unfixed

This questions requires both changing alpha and lambda. Besides, the value of `rho` is looped through the simulation process. I use package `caret` for tuning `alpha` and `lambda` simultaneously. To make it more comparable to the previous cases, I use the entire sample population as my train set (strictly speaking should be splited into test and training datasets).

```
library(caret)
set.seed(1)

# This matrix stores best tuning parameters
tune.matrix <- matrix(0, 4, 2)
row.names(tune.matrix) <- as.factor(rho.set)
colnames(tune.matrix) <- c("alpha", "lambda")

# This matrix stores coefficients
coef.matrix <- matrix(0, p, length(rho.set))
colnames(coef.matrix) <- as.factor(rho.set)
rownames(coef.matrix) <- row.names
```

```r
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
colnames(data$X) <- row.names
ctrl <- trainControl(method = "repeatedcv", number = 10)
tune.grid <- expand.grid(alpha = (1:10)*0.1, lambda = (1:10)*0.1)
model.fit <- train(data$X, data$y, method = "glmnet", tuneGrid = tune.grid, trControl =
tune.matrix[i, ] <- as.numeric(model.fit$bestTune)
a <- as.matrix(coef(model.fit$finalModel, model.fit$bestTune$lambda))
a <- a[-1,]
coef.matrix[, i] <- a
}
coef.aenet <- coef.matrix
```

2.8 least squares after adaptive elastic net

```r
lm2 <- list()
for (i in 1:length(rho.set)) {
set.seed(1)
rho <- rho.set[i]
data <- genData(n, p, beta, rho)
varlist <- coef.aenet[,4, drop = FALSE]
varlist <- varlist[which(varlist != 0), , drop = FALSE]
# extract row names, covert that into a vector
a <- row.names(varlist)
a <- gsub(pattern = "var", replacement = "", a)
a <- c(as.numeric(a))
data$X <- data$X[ , a]
lm.fit <- lm(y ~ X, data = data)
coef <- coef(lm.fit)[-1]
lm2[[paste0("lmcoef", i)]] <- coef
}
lm2
```

```
## $lmcoef1
##          X1          X2          X3          X4          X5          X6
##  2.06612973 -1.98762649 -0.81626894  0.16630029 -0.19891000  0.20426886
##          X7          X8          X9         X10         X11         X12
##  0.08935074 -0.20146197 -0.03858608  0.07218767  0.13802463  0.17090548
##         X13         X14         X15         X16         X17         X18
## -0.05468307  0.13669171  0.08076909 -0.06547407 -0.03160025 -0.14127287
```

```
##          X19          X20          X21          X22          X23          X24
##  0.05321151 -0.14670664  0.06907208 -0.17096228  0.17587422  0.06744562
##          X25          X26          X27          X28          X29          X30
## -0.01828686  0.02505587  0.08647583 -0.12251343 -0.20031520  0.05443441
##          X31          X32          X33          X34          X35          X36
## -0.02021602 -0.19861569  0.10612659 -0.15854874 -0.22895371  0.10736927
##          X37          X38          X39          X40          X41          X42
##  0.03700094 -0.01844090 -0.07707037 -0.05175235 -0.01308207  0.02864305
##          X43          X44          X45          X46          X47
##  0.08797180  0.09556289 -0.06241625 -0.09357267 -0.06279244
##
## $lmcoef2
##           X1           X2           X3           X4           X5
##  1.961547862 -1.845869334 -0.538748012  0.346944938 -0.434897319
##           X6           X7           X8           X9          X10
## -0.088521190 -0.326137037 -0.135381382 -0.002420665 -0.239094256
##          X11          X12          X13          X14          X15
##  0.189499880 -0.149949841  0.071788196 -0.126513054 -0.068135983
##          X16          X17          X18          X19          X20
##  0.151132036  0.071116076 -0.280069091 -0.084656290 -0.287473315
##          X21          X22          X23          X24          X25
## -0.051283457  0.109937989 -0.076914557 -0.039746697 -0.044513646
##          X26          X27          X28          X29          X30
## -0.167553187  0.055466255 -0.198204364 -0.249695141  0.093629607
##          X31          X32          X33          X34          X35
##  0.208821618 -0.019760551 -0.184175234 -0.074278743  0.030367128
##          X36          X37          X38          X39          X40
## -0.026886430 -0.114405988 -0.111929948 -0.038500046 -0.007427726
##          X41          X42          X43          X44          X45
##  0.313711109 -0.009571153 -0.094360072  0.097923736  0.072100366
##          X46          X47
## -0.171310242  0.192204487
##
## $lmcoef3
##           X1           X2           X3           X4           X5
##  1.692299876 -1.530669322 -0.224405481  0.106811426 -0.333992796
##           X6           X7           X8           X9          X10
## -0.005383681  0.132436364 -0.517473572  0.213736961 -0.315878744
##          X11          X12          X13          X14          X15
##  0.292512188 -0.108460907  0.088865160 -0.071922483  0.035397796
##          X16          X17          X18          X19          X20
## -0.009479769 -0.204325757 -0.165383326  0.141566943 -0.158785024
##          X21          X22          X23          X24          X25
##  0.148872628 -0.215064947 -0.072314662  0.041925951 -0.148937937
##          X26          X27          X28          X29          X30
```

```
##  0.236105473  0.046090105 -0.050987068 -0.306753961 -0.042122983
##          X31          X32          X33          X34          X35
##  0.372951037 -0.513060617 -0.310543105 -0.047839550 -0.067527286
##          X36          X37          X38          X39          X40
## -0.027611790 -0.117782135 -0.351592061  0.178827889  0.159902440
##          X41          X42          X43          X44          X45
##  0.006959118  0.155274230  0.083895330 -0.118711976  0.268026252
##          X46          X47
## -0.170415959  0.286532030
##
## $lmcoef4
##           X1           X2           X3           X4           X5
##  2.247557351 -1.427199139 -0.681506940  0.750448164 -0.358564445
##           X6           X7           X8           X9          X10
##  0.029746193 -0.121289487 -0.209694564  0.143362314  0.076986866
##          X11          X12          X13          X14          X15
##  0.155608980  0.297534137 -0.204994260  0.209847102  0.174592636
##          X16          X17          X18          X19          X20
## -0.221360578 -0.135504447  0.008887394 -0.222518980  0.376407553
##          X21          X22          X23          X24          X25
##  0.069416033 -0.142361327 -0.004381557  0.080355749 -0.029019900
##          X26          X27          X28          X29          X30
##  0.207960215  0.258664782 -0.268418713  0.317345423  0.449545321
##          X31          X32          X33          X34          X35
##  0.187991006  0.046304108 -0.311000561 -0.082449255 -0.124943712
##          X36          X37          X38          X39          X40
## -0.099509437 -0.135118437  0.188242970  0.067487524  0.019422050
##          X41          X42          X43          X44          X45
##  0.119216432  0.131039899  0.237065724 -0.119665837  0.102860955
##          X46          X47
## -0.420197206  0.304432465
```

3 graphic presentation

3.1 global setting

```
n <- 250    # Number of observations
p <- 220     # Number of predictors included in model
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0,212)) #beta value, 8 nonzeros, 212
rho.set <- c(0.2, 0.4, 0.6, 0.8) #set of rhos
results <- array(NA,dim = c(n,p,6),
dimnames = list(1:n,1:p,c("Lasso","Ridge","El-Net","Ad. Lasso", "Ad. El-Net","Ad.LAD-Las
```

3.2 image plot

```
myImagePlot <- function(x, ...){
     min <- min(x)
     max <- max(x)
     yLabels <- rownames(x)
     xLabels <- colnames(x)
     title <-c()
  # check for additional function arguments
  if( length(list(...)) ){
    Lst <- list(...)
    if( !is.null(Lst$zlim) ){
       min <- Lst$zlim[1]
       max <- Lst$zlim[2]
    }
    if( !is.null(Lst$yLabels) ){
       yLabels <- c(Lst$yLabels)
    }
    if( !is.null(Lst$xLabels) ){
       xLabels <- c(Lst$xLabels)
    }
    if( !is.null(Lst$title) ){
       title <- Lst$title
    }
  }
# check for null values
if( is.null(xLabels) ){
   xLabels <- c(1:ncol(x))
}
if( is.null(yLabels) ){
   yLabels <- c(1:nrow(x))
}

layout(matrix(data=c(1,2), nrow=1, ncol=2), widths=c(4,1), heights=c(1,1))

 # Red and green range from 0 to 1 while Blue ranges from 1 to 0
 ColorRamp <- rgb( seq(0,1,length=256),  # Red
                   seq(0,1,length=256),  # Green
                   seq(1,0,length=256))  # Blue
 ColorLevels <- seq(min, max, length=length(ColorRamp))

 # Reverse Y axis
 reverse <- nrow(x) : 1
 yLabels <- yLabels[reverse]
```

```
 x <- x[reverse,]

 # Data Map
 par(mar = c(3,5,2.5,2))
 image(1:length(xLabels), 1:length(yLabels), t(x), col=ColorRamp, xlab="",
 ylab="", axes=FALSE, zlim=c(min,max))
 if( !is.null(title) ){
    title(main=title)
 }
axis(BELOW<-1, at=1:length(xLabels), labels=xLabels, cex.axis=0.7)
 axis(LEFT <-2, at=1:length(yLabels), labels=yLabels, las= HORIZONTAL<-1,
 cex.axis=0.7)

 # Color Scale
 par(mar = c(3,2.5,2.5,2))
 image(1, ColorLevels,
      matrix(data=ColorLevels, ncol=length(ColorLevels),nrow=1),
      col=ColorRamp,
      xlab="",ylab="",
      xaxt="n")

 layout(1)
}
```

3.3 rho =0.2

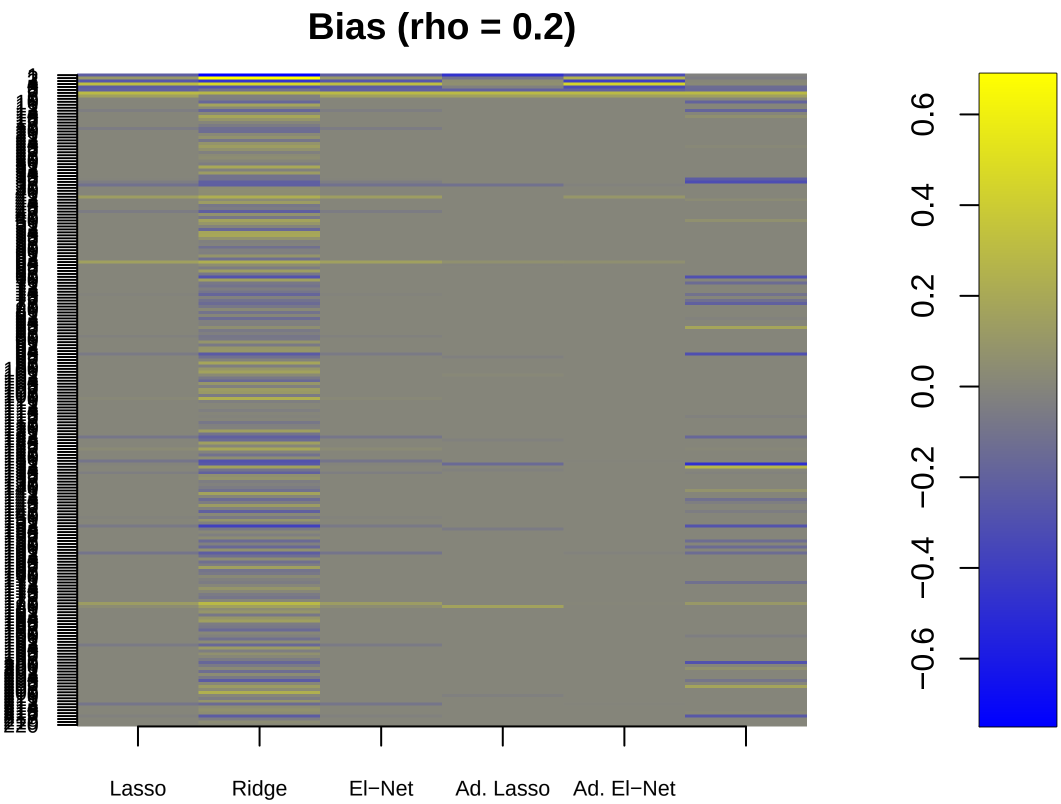
```
for (i in 1:n) {
results[i,,1] <- coef.lasso[,1]
results[i,,2] <- coef.ridge[,1]
results[i,,3] <- coef.enetfixed[,1]
results[i,,4] <- coef.alasso[,1]
results[i,,5] <- coef.aenet[,1]
results[i,,6] <- coef.aladlasso[,1]
}

B <- apply(results,2:3,mean) - beta
V <- apply(results,2:3,var)
MSE <- B^2 + V
apply(MSE,2,sum)
```

```
##        Lasso        Ridge       El-Net    Ad. Lasso    Ad. El-Net
##    0.5662123    4.3809980    0.5662123    0.4578011     0.7952864
## Ad.LAD-Lasso
```

```
##     1.4028800
```

```r
library(ggplot2)
library(reshape2)
B <- apply(results,2:3,mean) - beta
B <- as.data.frame(B)
myImagePlot(B, title = "Bias (rho = 0.2)")
```



Based on MSE, adaptive lasso and elastic net perform better at rho =0.2.

3.4 rho =0.4

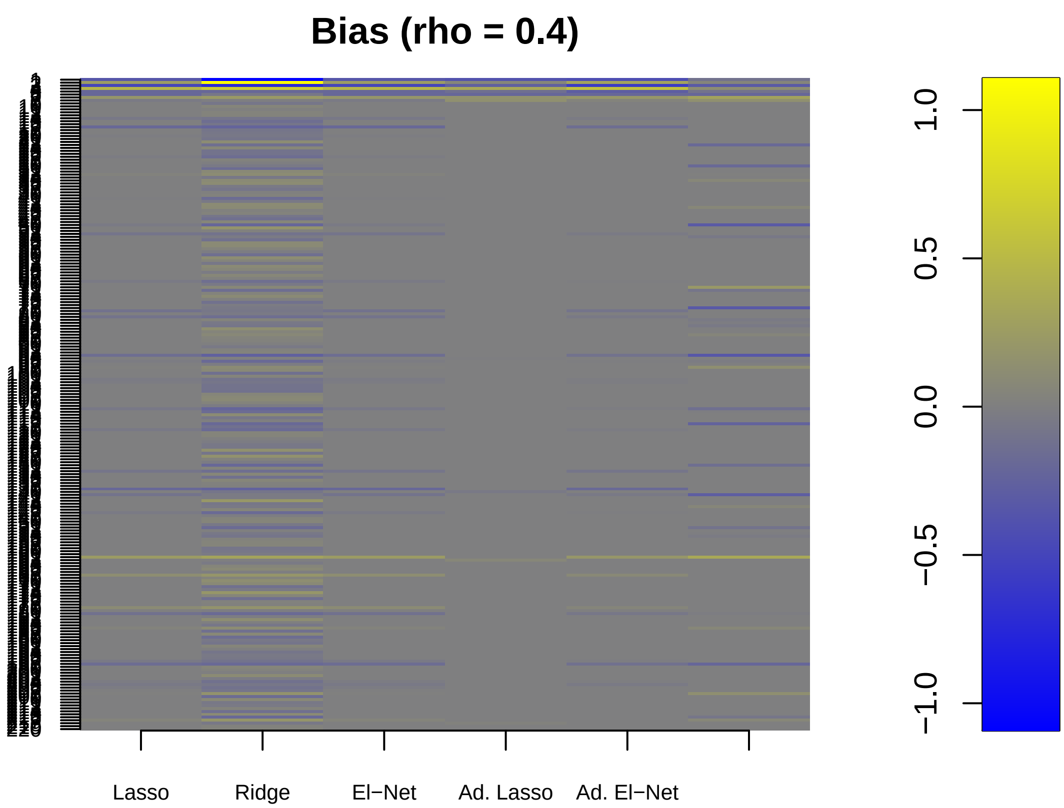
```r
for (i in 1:n) {
results[i,,1] <- coef.lasso[,2]
results[i,,2] <- coef.ridge[,2]
results[i,,3] <- coef.enetfixed[,2]
results[i,,4] <- coef.alasso[,2]
results[i,,5] <- coef.aenet[,2]
results[i,,6] <- coef.aladlasso[,2]
}

B <- apply(results,2:3,mean) - beta
```

```
V <- apply(results,2:3,var)
MSE <- B^2 + V
apply(MSE,2,sum)
```

```
##         Lasso         Ridge        El-Net     Ad. Lasso    Ad. El-Net
##     1.0536554     5.5249841     1.0536554     0.4587499     1.2383567
## Ad.LAD-Lasso
##     1.1669723
```

```
library(ggplot2)
library(reshape2)
B <- apply(results,2:3,mean) - beta
B <- as.data.frame(B)
myImagePlot(B, title = "Bias (rho = 0.4)")
```



3.5 rho =0.6

When rho =0.6, adaptive lad lasso has the best performance.

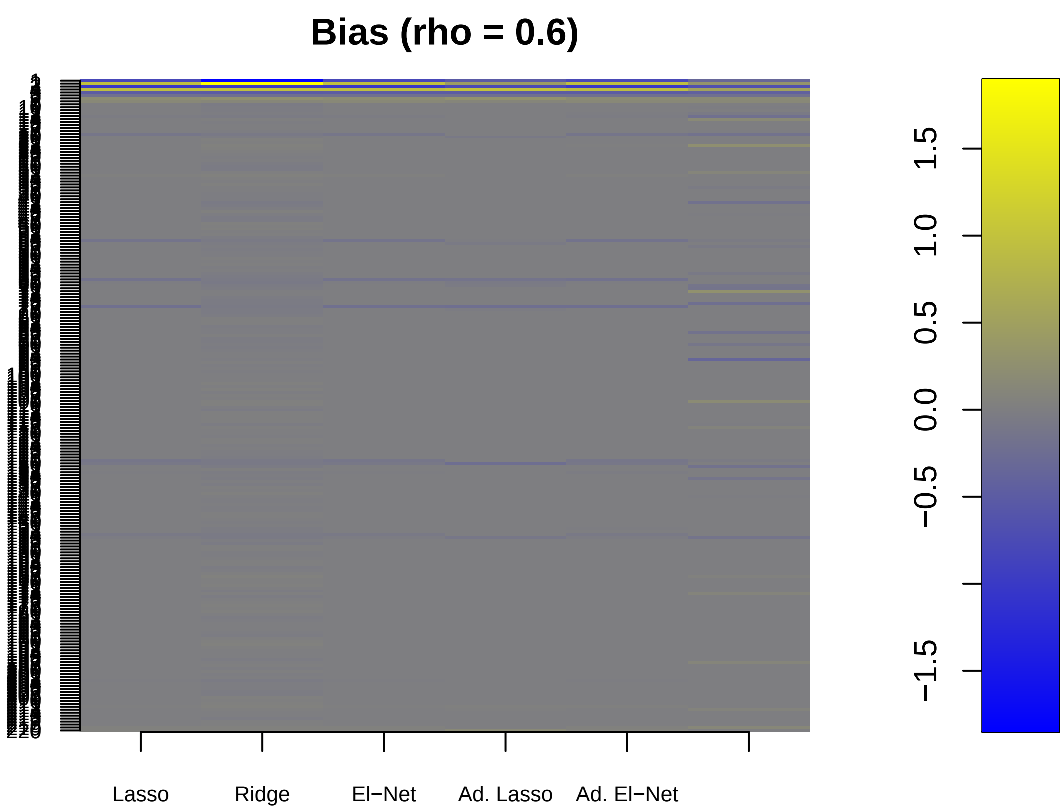
```
for (i in 1:n) {
results[i,,1] <- coef.lasso[,3]
results[i,,2] <- coef.ridge[,3]
results[i,,3] <- coef.enetfixed[,3]
```

```r
results[i,,4] <- coef.alasso[,3]
results[i,,5] <- coef.aenet[,3]
results[i,,6] <- coef.aladlasso[,3]
}

B <- apply(results,2:3,mean) - beta
V <- apply(results,2:3,var)
MSE <- B^2 + V
apply(MSE,2,sum)
```

```
##        Lasso        Ridge       El-Net    Ad. Lasso   Ad. El-Net
##     3.934939     9.367225     3.934939     2.623446     3.846759
## Ad.LAD-Lasso
##     1.893931
```

```r
library(ggplot2)
library(reshape2)
B <- apply(results,2:3,mean) - beta
B <- as.data.frame(B)
myImagePlot(B, title = "Bias (rho = 0.6)")
```



3.6 rho =0.8

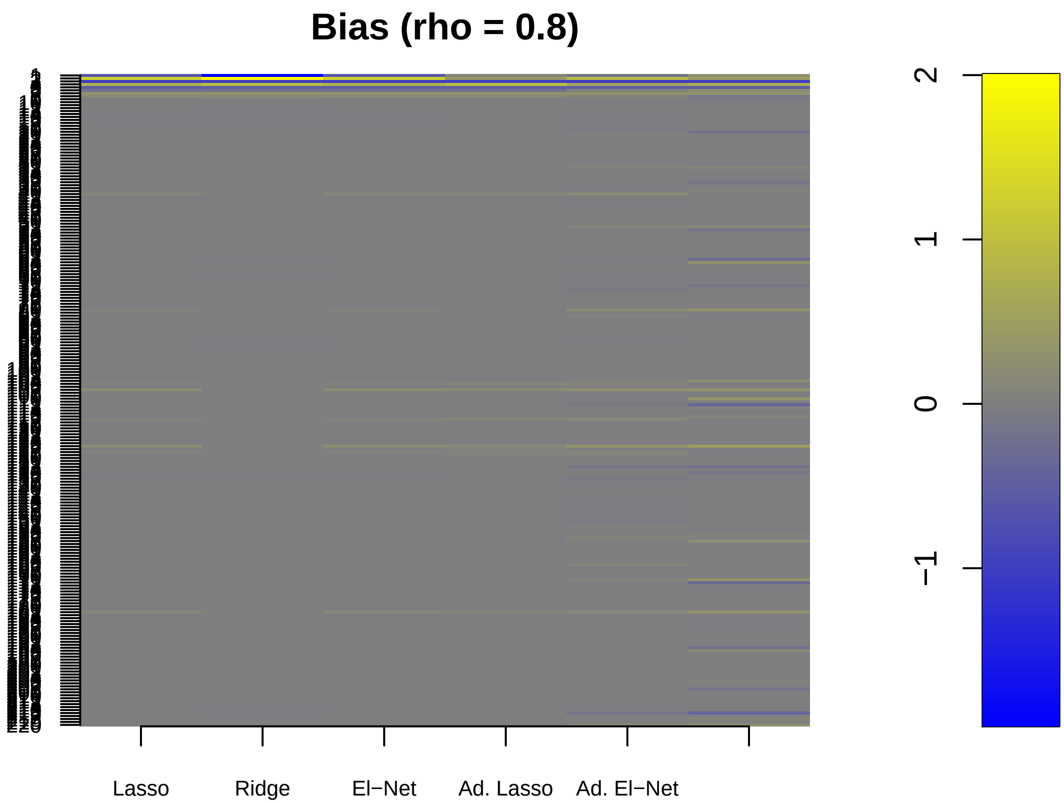
When rho =0.8, adaptive lasso and adaptive elastic net are better.

```
for (i in 1:n) {
results[i,,1] <- coef.lasso[,4]
results[i,,2] <- coef.ridge[,4]
results[i,,3] <- coef.enetfixed[,4]
results[i,,4] <- coef.alasso[,4]
results[i,,5] <- coef.aenet[,4]
results[i,,6] <- coef.aladlasso[,4]
}

B <- apply(results,2:3,mean) - beta
V <- apply(results,2:3,var)
MSE <- B^2 + V
apply(MSE,2,sum)
```

```
##         Lasso         Ridge        El-Net     Ad. Lasso    Ad. El-Net
##      4.161752     10.194544      4.161752      2.711686      3.048505
## Ad.LAD-Lasso
##      3.992885
```

```
library(ggplot2)
library(reshape2)
B <- apply(results,2:3,mean) - beta
B <- as.data.frame(B)
myImagePlot(B, title = "Bias (rho = 0.8)")
```



4 pararell precessing

To illustrate how pararell computing can save time, I use ridge, lasso, elastic net, adaptive lasso as an example (rho=0.2).

For some unknown reason my laptop won't even perform the time1 function. It crashed several times when I tried to run all the functions together.

```
n <- 250    # Number of observations
p <- 220     # Number of predictors included in model
beta <- c(2, -2, 1, -1, 0.5, 0.2, -0.3, -0.15, rep(0,212)) #beta value, 8 nonzeros, 212
rho <- 0.2
results <- array(NA,dim = c(n,p,4))
dimnames = list(1:n,1:p,c("Lasso","Ridge","El-Net","Ad. Lasso"))
Data <- genData(n, p, beta, rho)


set.seed(1)
time1 <- system.time(
for (i in 1:n){
results[i,,1] <- lasso(Data)[-1]
results[i,,2] <- ridge(Data)[-1]
```

```
results[i,,3] <- enet(Data)[-1]
results[i,,4] <- alasso(Data)[-1]
})

install.packages('doParallel')
library(doParallel)
getDoParWorkers()
registerDoSEQ()
getDoParWorkers()
registerDoParallel(cores=4)
getDoParWorkers()

results <- foreach(i=1:n, .export=c('lasso', 'ridge', 'enet', 'alasso')) %dopar% {data.f
```