Monte Carlo Experiment of Penalized Estimators

Saar Yalov

December 26, 2017

In this homework assignment we created several new functions for LASSO type estimators and used a monte carlo experiment to see how they compared in terms of a MSE for a multivariate random sample.

We utilized the FIRST function described by Ghosal and colleagues in a 2009 paper. (http://www4.stat.ncsu.edu/ sghosal/papers.html) To do so we compiled the c code, instantiated a shareable object and then used it in R.

Furthermore, we used the foreach package in R to shorten the computation time for this procedure.

The functions are:

```
aenet2 <- function(Data){#New
  cvfit <- cv.glmnet(Data$X,Data$y,alpha=0)
  cva.list = list()
  min.mse = rep(0,11)
  for(i in seq(1,11,1)) {
    cv = cv.glmnet(Data$X,Data$y, alpha=(i-1)/10, penalty.factor =
        abs(1/as.numeric(coef(cvfit,s=cvfit$lambda.min)+(1/n))))
    cva.list[[i]] =as.numeric(coef(cv,s=cv$lambda.min))
    min.mse[i] = cv$cvm
  }
  min.ind = which.min(min.mse)
  cva.min = cva.list[[min.ind]]
  return(cva.min)
}

lm.after.aenet2 <- function(Data){#New
  require(glmnet)
```

```
  coef.in = as.integer(aenet2(Data)!=0)
  coef.in=coef.in[-1] # Do not include intercept
  coef.in.index = which(coef.in != 0)
  result.vector = rep(0,ncol(Data$X))
  if(length(coef.in.index) == 0 ){
    return(result.vector)
  }
  vector = as.numeric(coef(lm(Data$y~Data$X[,coef.in.index])))[-1]
  j = 1
  for(i in coef.in.index){
    result.vector[i] = vector[j]
    j = j +1
  }
  return(result.vector)
}
alasso2 <- function(Data){#New
  require(glmnet)
  cvfit <- cv.glmnet(Data$X,Data$y,alpha=0)
  cvfit2 <- cv.glmnet(Data$X,Data$y, alpha =1,penalty.factor =
      abs(1/as.numeric(coef(cvfit,s=cvfit$lambda.min)+(1/n))))
  return(as.numeric(coef(cvfit2,s=cvfit2$lambda.min)))
}

lm.after.alasso2 <- function(Data){ #new
  require(glmnet)
  coef.in <- as.integer(alasso2(Data)!=0)
  coef.in=coef.in[-1] # Do not include intercept
  coef.in.index = which(coef.in != 0)
  result.vector = rep(0,ncol(Data$X))
  if(length(coef.in.index) == 0 ){
    return(result.vector)
  }
  vector = as.numeric(coef(lm(Data$y~Data$X[,coef.in.index])))[-1]
  j = 1
  for(i in coef.in.index){
    result.vector[i] = vector[j]
    j = j +1
  }
  return(result.vector)
}
```

```
aladlasso.2 <- function(Data,n.lam=301){#new
  require(quantreg)
  tempcfQR1 = coef(rq(Data$y~Data$X,.5,method="lasso",lambda = .1))
  lam = c(0,exp(seq(log(1e-2),log(1e7),len=n.lam)))
  tempcfQR = list()
  mse = rep(0,n.lam)
  for(i in 1:n.lam){
    tempcfQR[[i]] = rq(Data$y~Data$X,.5,method="lasso",lambda =
        lam[i])
    mse[i] = sum((tempcfQR[[i]]$residuals)^2 )
  }
  min.ind = which.min(mse)
  result = coef(tempcfQR[[min.ind]])
  return(result)
}
```

1 Compare Computation Time

We will compare the computation time excluding the computation for the different ρ's.

```
> set.seed(1)
> results <- array(NA,dim=c(N,p,8+6))
> strt<-Sys.time()
> for(i in 1:N){
+   Data <- genData.2(n,p,beta,rho=rh)
+   results[i,,1] <- bestsub(Data)[-1]
+   results[i,,2] <- lasso(Data)[-1]
+   results[i,,3] <- ridge(Data)[-1]
+   results[i,,4] <- enet(Data)[-1]
+   results[i,,5] <- alasso(Data)[-1]
+   results[i,,6] <- aenet(Data)[-1]
+   results[i,,7] <- aladlasso(Data)[-1]
+   results[i,,8] <- coef(lm(y~X,Data))[-1]
+
+   results[i,,9] <- aenet2(Data)[-1]
+   results[i,,10] <- lm.after.aenet2(Data)
+   results[i,,11] <- alasso2(Data)[-1]
+   results[i,,12] <- lm.after.alasso2(Data)
+   results[i,,13] <- aladlasso.2(Data)[-1]
+   #results[i,,14] <- first(Data$y,Data$X,1,10)$estimates
+ }
There were 50 or more warnings (use warnings() to see the first 50)
> print(Sys.time()-strt)
Time difference of 12.01878 mins

> strt<-Sys.time()
> set.seed(1)
> foreach (i=1:N) %dopar%
+ {
+ #Data <- genData.2(n,p,beta,rho=rh)
+ Data <- genData(n,p,beta)
+ results[i,,1] <- bestsub(Data)[-1]
+ results[i,,2] <- lasso(Data)[-1]
+ results[i,,3] <- ridge(Data)[-1]
+ results[i,,4] <- enet(Data)[-1]
```

```
+ results[i,,5] <- alasso(Data)[-1]
+ results[i,,6] <- aenet(Data)[-1]
+ results[i,,7] <- aladlasso(Data)[-1]
+ results[i,,8] <- coef(lm(y~X,Data))[-1]
+
+ results[i,,9] <- aenet2(Data)[-1]
+ results[i,,10] <- lm.after.aenet2(Data)
+ results[i,,11] <- alasso2(Data)[-1]
+ results[i,,12] <- lm.after.alasso2(Data)
+ results[i,,13] <- aladlasso.2(Data)[-1]
+ results[i,,14] <- first(Data$y,Data$X,1,10)$estimates
+
+
+
+ displayProgressBar(i,N)
+
+ }

> print(Sys.time()-strt)
Time difference of 2.88874 mins
```

The computation time using without parallel computing was 12.5 minutes.

With foreach the computation reduced to a mere 2.88874 minutes.

2 Compare the Different Methods

Now, comparing the MSE for $\rho = .2$

```
> B <- apply(results[,,,1],2:3,mean)-beta
> V <- apply(results[,,,1],2:3,var)
> MSE <- B^2+V
> apply(MSE,2,sum)
 [1] 4.213415 5.387085 6.568936 5.745712 4.887200 5.180859
     4.963785 9.589672 6.094189 8.042539 5.900710 6.666098 8.895208
     6.213525
> MSE.table = apply(MSE,2,sum)
> MSE.table = round(MSE.table,4)
> print(rbind.data.frame(dimnames[[3]],MSE.table))
```

```
Subset              Lasso              Ridge
 0.7151             0.8186             1.0505

El-Net
 0.847

Ad. Lasso             Ad. El-Net             Ad. LAD-Lasso
0.6199                 0.6586                 0.8508

 OLS                aenet2          lm.after.aenet2
1.1234              0.8263              0.8762

alasso2      lm.after.alasso2          aladlasso.2
 0.6893          0.8325                 1.2009

FIRST
 0.8131
```

This is done similarly for $rho = .4, .6, .8$ and presented bellow:

rho	Subset	Lasso	Ridge	El-Net	Ad.Lasso	Ad. El-Net	Ad. LAD-Lasso	OLS	aenet2	lm.a.aenet2	alasso2	lm.after.alasso2	aladlasso.2	FIRST
.2	0.7151	0.8186	1.0505	0.847	0.6199	0.6586	0.8508	1.1234	0.8263	0.8762	0.6893	0.8325	1.2009	0.8131
.4	0.9095	1.0887	1.3715	1.1558	0.8291	0.8879	1.1816	1.425	1.0724	1.1539	0.875	1.0246	1.5769	1.336
.6	1.3466	1.6325	1.9669	1.7069	1.2541	1.283	1.6954	2.1555	1.634	1.7493	1.3306	1.5532	2.3091	2.2034
.8	2.6702	3.1891	3.6726	3.3012	2.4689	2.6605	3.117	4.446	3.4615	3.9373	2.9096	3.2507	4.2812	4.2356

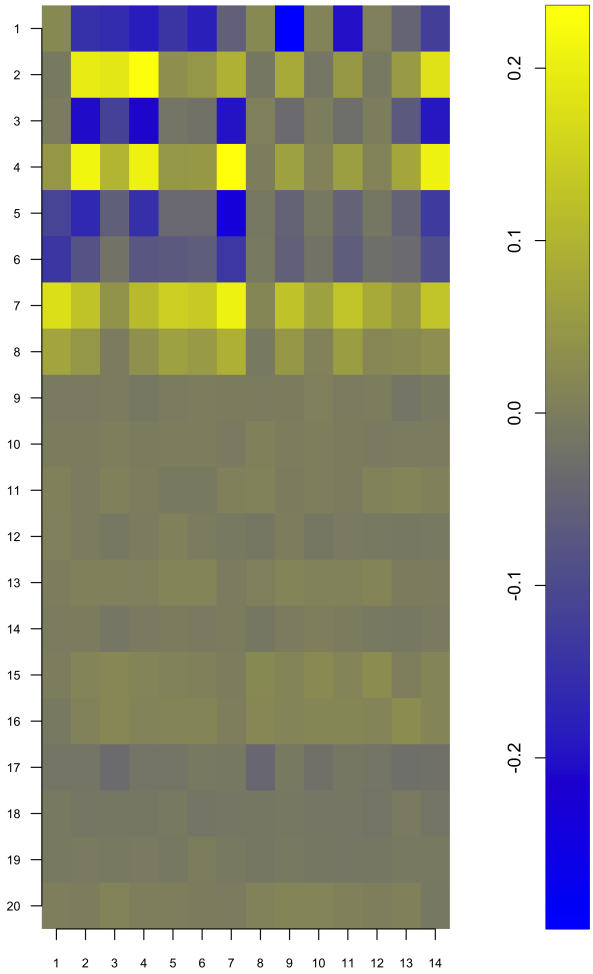
Table 1: MSE of Algorithms Across ρ's

Adaptive lasso minimizes MSE in all cases of ρ!

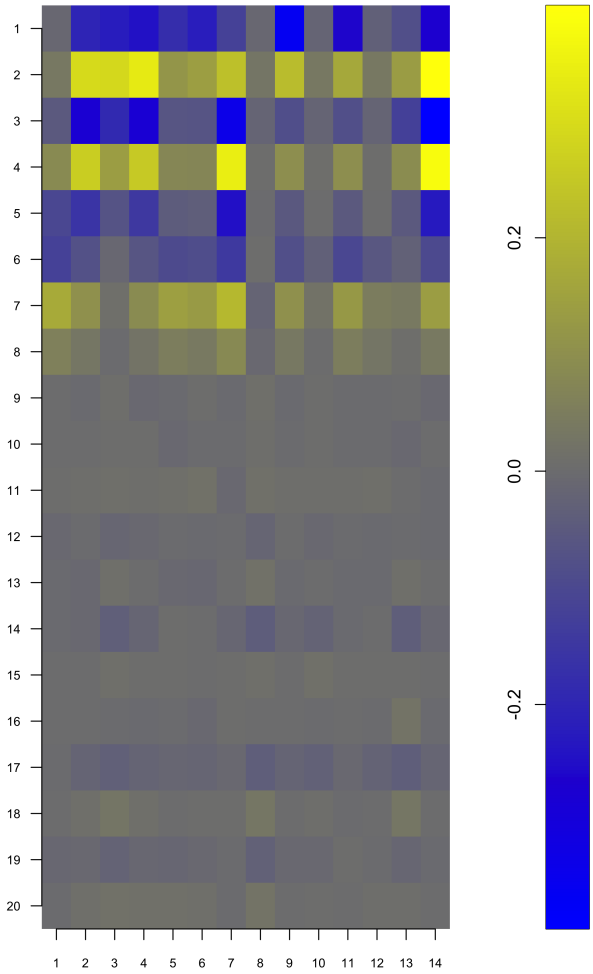
3 Bias and Variance of the Different Methods Figures

In the figures bellow, the methods utilizing least squares after the initial algorithm seem to be less biased then the reminder of the algorithms. The degree of bias in such algorithms. increases with ρ. Adaptive type algorithms reduce variance with increase in ρ relative to other methods.

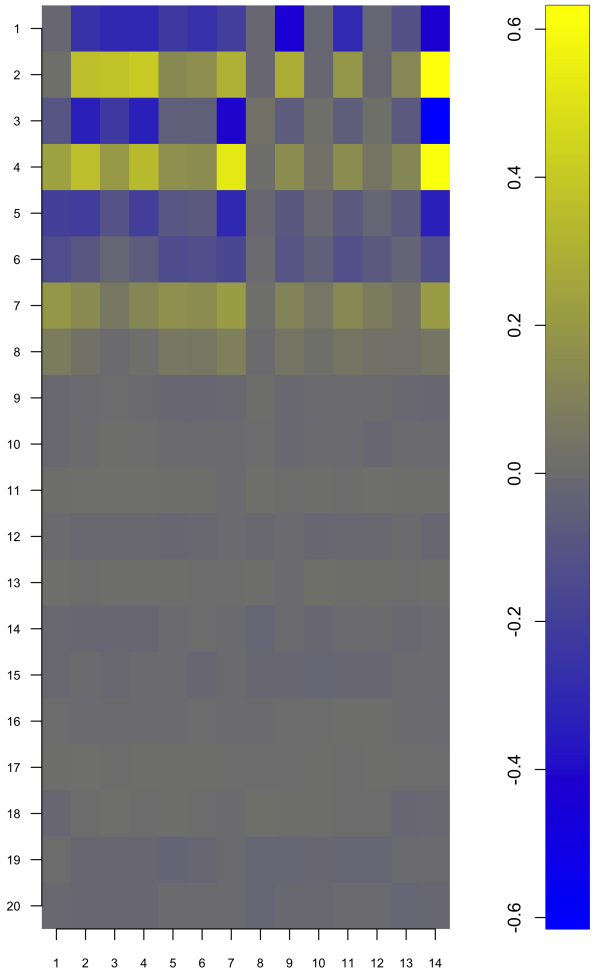
Bias $\rho = .2$


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14
0.2
0.1
0.0
-0.1
-0.2

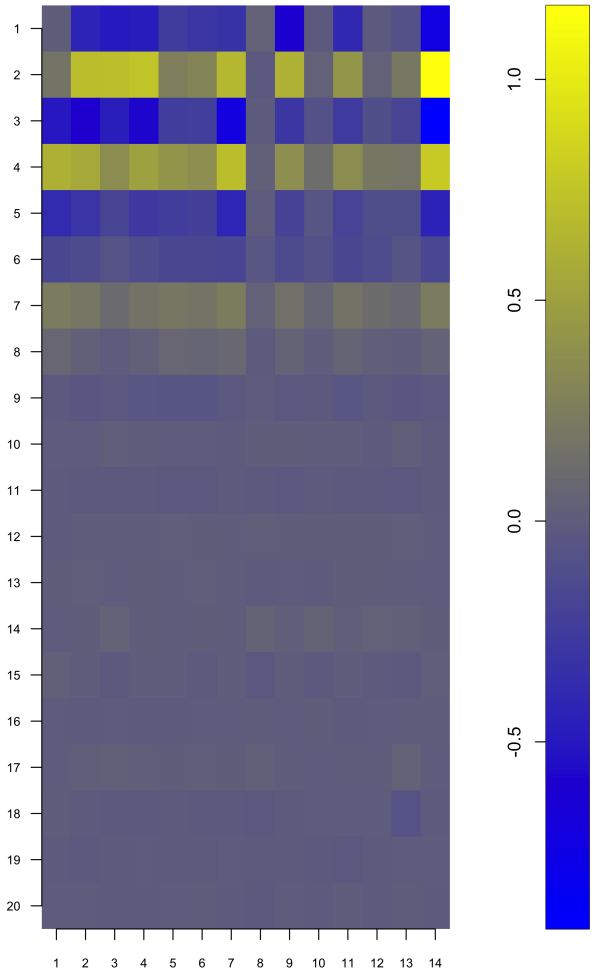
Bias $\rho = .4$


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14
0.2
0.0
-0.2

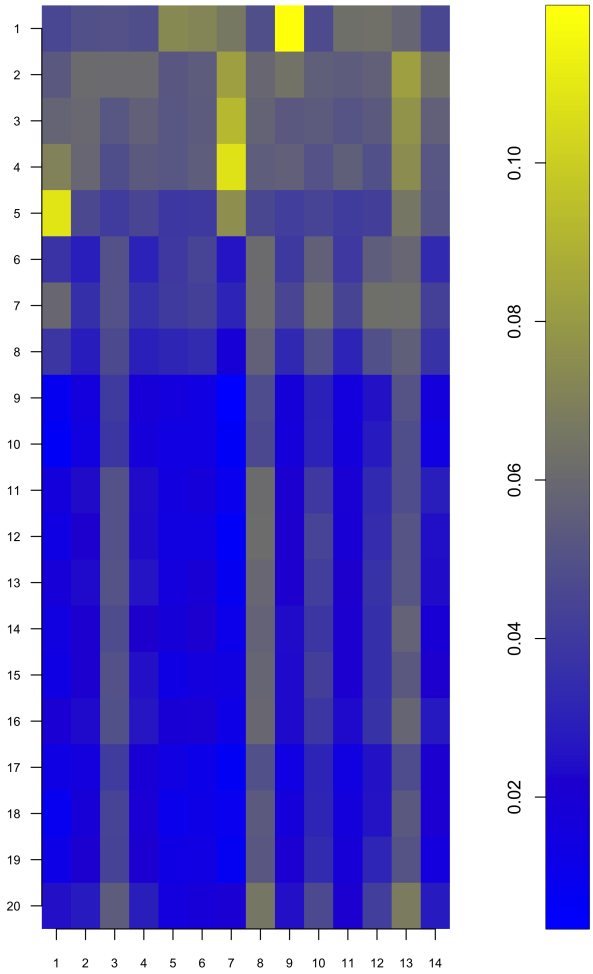
Bias $\rho = .6$


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6

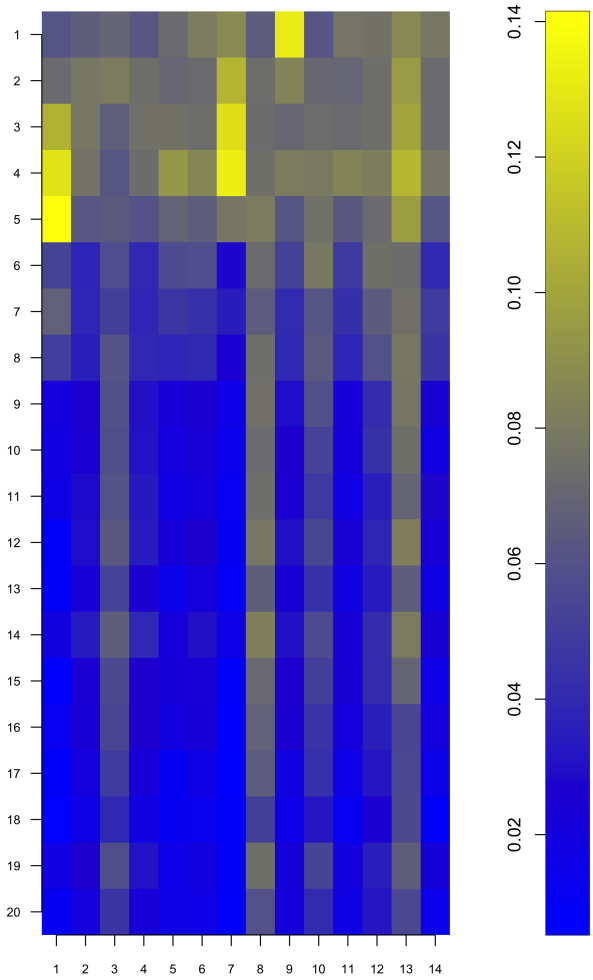
Bias $\rho = .8$


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14
1.0
0.5
0.0
-0.5

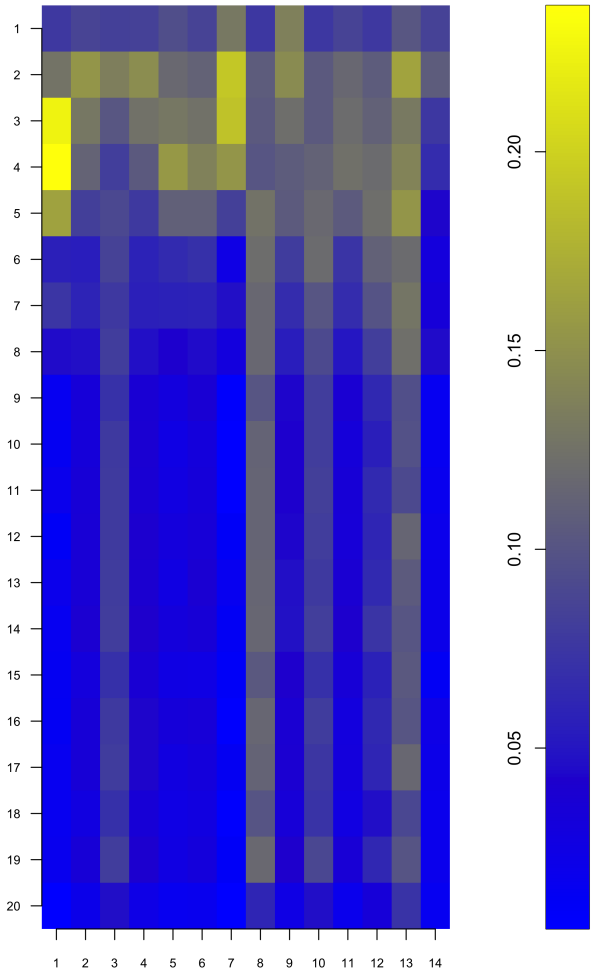
Variance $\rho = .2$



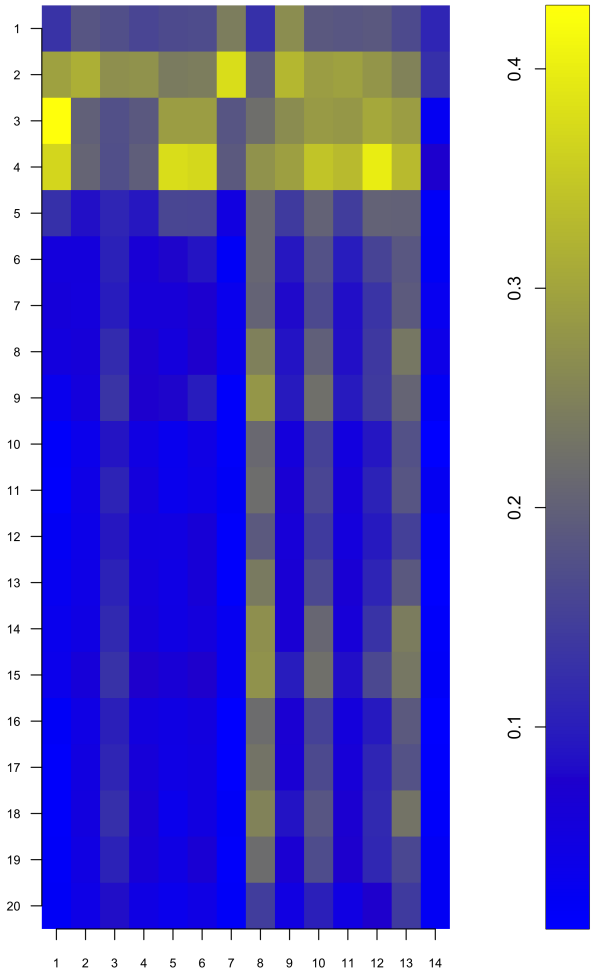
Variance $\rho = .4$



Variance $\rho = .6$



Variance $\rho = .8$



```
# How it is done
#start time
strt<-Sys.time()
for(j in 1:4){
foreach (i=1:N) %dopar%
{
#Data <- genData.2(n,p,beta,rho=rh)
Data <- genData.2(n,p,beta,rho=j/5)
```

```
results[i,,1,j] <- bestsub(Data)[-1]
results[i,,2,j] <- lasso(Data)[-1]
results[i,,3,j] <- ridge(Data)[-1]
results[i,,4,j] <- enet(Data)[-1]
results[i,,5,j] <- alasso(Data)[-1]
results[i,,6,j] <- aenet(Data)[-1]
results[i,,7,j] <- aladlasso(Data)[-1]
results[i,,8,j] <- coef(lm(y~X,Data))[-1]

results[i,,9,j] <- aenet2(Data)[-1]
results[i,,10,j] <- lm.after.aenet2(Data)
results[i,,11,j] <- alasso2(Data)[-1]
results[i,,12,j] <- lm.after.alasso2(Data)
results[i,,13,j] <- aladlasso.2(Data)[-1]
results[i,,14,j] <- first(Data$y,Data$X,1,10)$estimates



#displayProgressBar(i,N)

}
}
print(Sys.time()-strt)
```