STATS790 HW2

```r
library(matlib)
library(microbenchmark)
library(readr)
library(dplyr)
library(ggplot2)
library(glmnet)
```

Question 1

Naive linear Algebra:

To estimate the coefficients of linear regression β, we can use the ordinary least squares, so we have,

$$\begin{aligned}
S(\beta) &= \sum_{n=1}^{i} e_i \\
&= \epsilon'\epsilon \\
&= (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) \\
&= \mathbf{y}'\mathbf{y} - 2\beta'\mathbf{X}\mathbf{y} - \beta'\mathbf{X}'\mathbf{X}\beta
\end{aligned}$$

Then, we want to take the derivative of $S(\beta)$,

$$\frac{\partial S(\beta)}{\partial \beta} = -2\mathbf{X}\mathbf{y} + 2\mathbf{X}'\mathbf{X}\beta$$

Set the derivative to zero, we get

$$\begin{aligned} -2\mathbf{X}\mathbf{y} + 2\mathbf{X}'\mathbf{X}\hat{\beta} &= 0 \\ \mathbf{X}'\mathbf{X}\hat{\beta} &= \mathbf{X}\mathbf{y} \\ \hat{\beta} &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \end{aligned}$$

QR decomposition

(Due to time constraint, I will use a less regorous matrix representation, such as using X to represent $\mathbf{X}$. The hand-written script with more details is available in the **appendix** at the end of this document.)

Q is orthonormal and R is upper triangular. Decompose X as $X = QR$ and substitute it into $X'X\hat{\beta} = Xy$, we have

$$\hat{\beta} = R^{-1}Q'y$$

SVD

$$\hat{\beta} = VD^{-1}U'y$$

For detailed calculation process, check appendix.

Cholesky Decomposition

$$\hat{\beta} = (L')^{-1}L^{-1}X'y$$

For detailed calculation process, check appendix.

Computation part

Assume there is no intercept β_0.

```r
set.seed(101)
sim_fun <- function(n, p) {
    y <- rnorm(n)
    X <- matrix(rnorm(p*n), ncol = p)
    list(X = X, y = y)
}
```

```
naive<- function(X, y) {
  beta <- inv(t(X) %*% X) %*% t(X) %*% y
  return(beta)
}
```

```
qr_fun <- function(X, y) {
  QR<-qr(x=X)
  Q<-qr.Q(QR)
  R<-qr.R(QR)
  beta <- inv(R) %*% t(Q) %*% y
  return(beta)
}
```

```
svd_fun <- function(X, y) {
  SVD <- svd(X)
  U <- SVD$u
  D <- diag(SVD$d)
  V <- SVD$v
  beta<-V%*%inv(D)%*%t(U)%*%y
  return(list(beta))
}
```

Fix p=10,observe the average time change with n

From the result of the linear regressions, we know that, for SVD, 1 unit increase in log(n) will increase around 0.6845 unit of change in log of computational time. For naive approach, 1 unit increase in log(n) will increase 0.3121 unit of change in log of computational time.For QR decomposition approach, 1 unit increase in log(n) will increase 0.5567 unit of change in log of computational time.

```
set.seed(101)
avg_times<-list()
n_range<-round(10^seq(2, 5, by = 0.25))#seq(10,1100,100)

svd_times<-list()
for (n in n_range){
    s<-sim_fun(n,10)
    svd_m<-microbenchmark(svd_fun(s$X,s$y))
    svd_times[[paste(n)]] <- svd_m$time
}
avg_times$svd <- sapply(svd_times, mean)
fit.svd <- lm(log(avg_times$svd) ~ log(n_range))

qr_times<-list()
for (n in n_range){
    s<-sim_fun(n,10)
    qr_m<-microbenchmark(qr_fun(s$X,s$y))
    qr_times[[paste(n)]] <- qr_m$time
}
avg_times$qr <- sapply(qr_times, mean)
fit.qr <- lm(log(avg_times$qr) ~ log(n_range))

naive_times<-list()
for (n in n_range){
    s<-sim_fun(n,10)
    naive_m<-microbenchmark(naive(s$X,s$y))
    naive_times[[paste(n)]] <- naive_m$time
}

avg_times$naive <- sapply(naive_times, mean)
fit.naive <- lm(log(avg_times$naive) ~ log(n_range))

fit.qr
```

```

Call:
lm(formula = log(avg_times$qr) ~ log(n_range))

Coefficients:
 (Intercept)  log(n_range)
     10.8639        0.6121
```

```
fit.naive
```

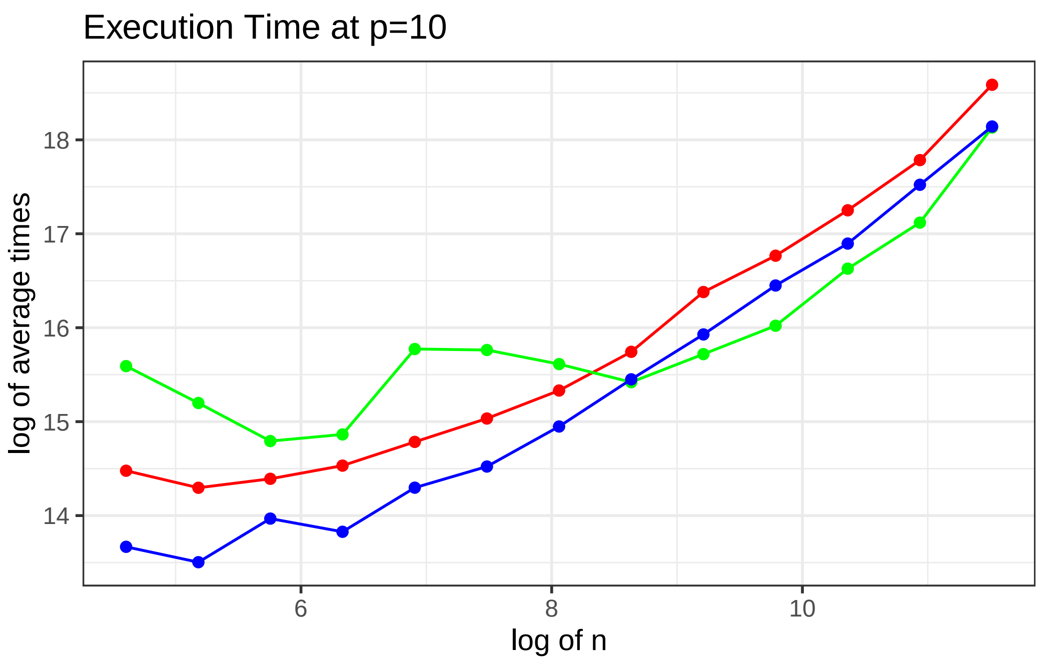
```

Call:
lm(formula = log(avg_times$naive) ~ log(n_range))

Coefficients:
 (Intercept)  log(n_range)
      13.187         0.336
```

```
fit.svd
```

```

Call:
lm(formula = log(avg_times$svd) ~ log(n_range))

Coefficients:
 (Intercept)  log(n_range)
      9.8788        0.6748
```

The log-log plot is shown as below. The red line and dots represents for the qr decomposition, green lines and green dots represents for the naive approach, and blue line and blue dots represents for the singular value decomposition approach. Overall, the time required for each algorithm increases exponentially when n increases. In addition, the naive approach did not perform well when n is small; when n become larger, QR decomposition and SVD takes even more time and naive approach become efficent. In our example, when n get's larger, QR decomposition performs the best.

```
p<-ggplot(data=avg_times%>%as.data.frame())+geom_line(aes(x=log(n_range),y=log(avg_times$q

  geom_line(aes(x=log(n_range),y=log(avg_times$naive)),color='green')+geom_point(aes(x=log
    geom_line(aes(x=log(n_range),y=log(avg_times$svd)),color='blue')+geom_point(aes(x=log(
  ggtitle("Execution Time at p=10")+theme_bw()
p
```



Fix n=100,observe the average time change with p

Overall, the average time increases as p increases. The SVD performs use the least average time, whereas the naive approach have the highest average time. This indicates that, when the number of covariates increases, it is better to use SVD or QR decomposition compared to using naive approach.

```
set.seed(101)
p_range<-c(5,25,50)#seq(10,1100,100)

svd_times<-list()
for (p in p_range){
    s<-sim_fun(100,p)
    svd_m<-microbenchmark(svd_fun(s$X,s$y))
    svd_times[[paste(n, p)]] <- svd_m$time
}
avg_times$svd <- sapply(svd_times, mean)
fit.svd <- lm(log(avg_times$svd) ~ log(p_range))

qr_times<-list()
for (p in p_range){
  s<-sim_fun(100,p)
```

```r
    qr_m<-microbenchmark(qr_fun(s$X,s$y))
    qr_times[[paste(n, p)]] <- qr_m$time
}
avg_times$qr <- sapply(qr_times, mean)
fit.qr <- lm(log(avg_times$qr) ~ log(p_range))

naive_times<-list()
for (p in p_range){
    s<-sim_fun(100,p)
    naive_m<-microbenchmark(naive(s$X,s$y))
    naive_times[[paste(n, p)]] <- naive_m$time
}

avg_times$naive <- sapply(naive_times, mean)
fit.naive <- lm(log(avg_times$naive) ~ log(p_range))

fit.qr
```

```

Call:
lm(formula = log(avg_times$qr) ~ log(p_range))

Coefficients:
 (Intercept)  log(p_range)
      10.958         1.955
```

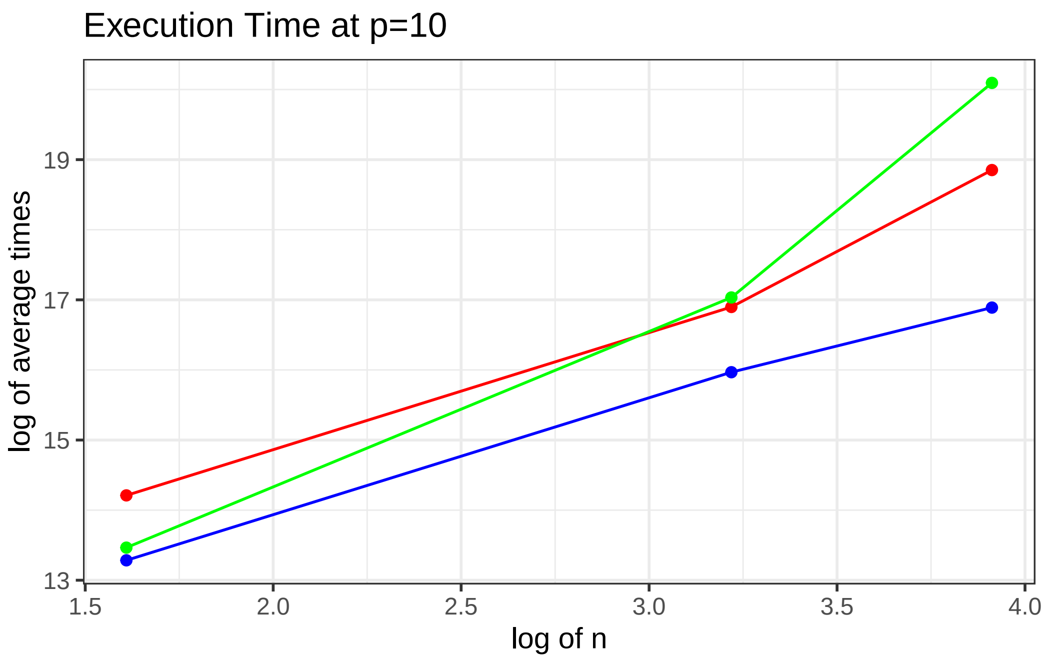
```r
fit.naive
```

```

Call:
lm(formula = log(avg_times$naive) ~ log(p_range))

Coefficients:
 (Intercept)  log(p_range)
       8.816         2.763
```

```r
fit.svd
```

```
Call:
lm(formula = log(avg_times$svd) ~ log(p_range))

Coefficients:
 (Intercept)  log(p_range)  
      10.766         1.584  
```

```
p2<-ggplot(data=avg_times%>%as.data.frame())+geom_line(aes(x=log(p_range),y=log(avg_times$

  geom_line(aes(x=log(p_range),y=log(avg_times$naive)),color='green')+geom_point(aes(x=log
    geom_line(aes(x=log(p_range),y=log(avg_times$svd)),color='blue')+geom_point(aes(x=log(
  ggtitle("Execution Time at p=10")+theme_bw()
p2
```



Question 2

Implement ridge regression by data augmentation.

```
prostate<-read_table("https://hastie.su.domains/ElemStatLearn/datasets/prostate.data")
X <- as.matrix(prostate[, 1:8])
y<-as.matrix(prostate[,9])
X <- scale(X, center = TRUE, scale = TRUE)

ridge_aug<-function(X,y,lam){
  X_aug <- rbind(X, sqrt(lam) * diag(ncol(X)))
  y_aug <- c(y, rep(0, ncol(X)))
  #fit ridge regression model
  beta <- inv(t(X_aug) %*% X_aug + lam * diag(ncol(X_aug))) %*% t(X_aug) %*% y_aug
  return(beta)

}
```

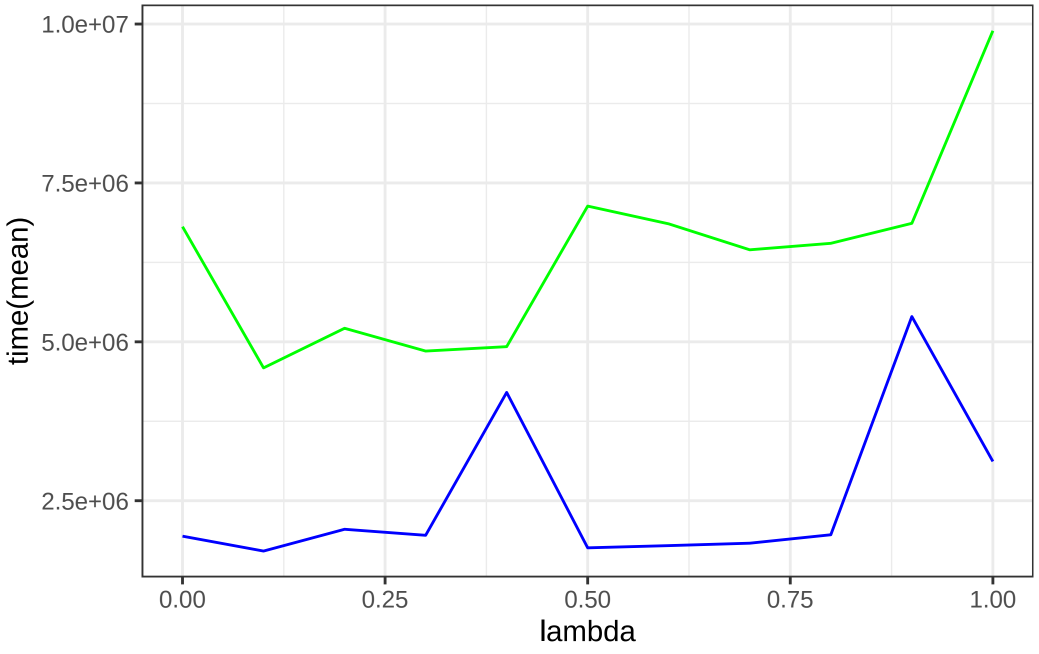
```
lambda<-seq(0,1,0.1)
ridge_naive<-function(X,y,lambda){
  fit <- glmnet(X, y, alpha = 0, lambda =lambda)
  beta_glmnet <- coef(fit, s = lambda)[-1]
  beta_glmnet
}
fit <- glmnet(X, y, alpha = 0, lambda =lambda)
fit
```

```
Call:  glmnet(x = X, y = y, alpha = 0, lambda = lambda)

   Df  %Dev Lambda
1   8 66.98    1.0
2   8 67.00    0.9
3   8 67.01    0.8
4   8 67.03    0.7
5   8 67.04    0.6
6   8 67.05    0.5
7   8 67.06    0.4
8   8 67.06    0.3
9   8 67.07    0.2
10  8 67.07    0.1
```

```
11  8 67.07    0.0
```

```
ridge_naive_time<-list()
ridge_aug_time<-list()
for (i in seq_along(lambda)){
  lam<-lambda[i]
  ridge_naive_m<-microbenchmark(ridge_naive(X,y,lam))
  ridge_naive_time[i]=mean(ridge_naive_m$time)

  ridge_aug_m<-microbenchmark(ridge_aug(X,y,lam))
  ridge_aug_time[i]=mean(ridge_aug_m$time)
}
```

```
ggplot()+
  geom_line(aes(x=lambda,y=as.numeric(ridge_aug_time)),color="blue")+
  geom_line(aes(x=lambda,y=as.numeric(ridge_naive_time)),color="green") +labs(x="lambda",y
```



In the graph above, the green line represents for the naive approach and the blue line represents for the augmented approach. Overall, for different lambdas, the naive approach takes more time compared with the augmented ridge.

Question 3

3.6

$$P(\beta|y) \propto P(y|\beta)P(\beta)$$

where $y \sim N(X\beta, \sigma^2 I), \beta \sim (0, \tau I)$.

$$P(\beta|y) \propto e^{(-\frac{1}{2})(\frac{y-\mu}{\sigma})^2} e^{(-\frac{1}{2})(\frac{\beta-\mu}{\sigma})^2}$$

Take the log of both side, we have

$$\begin{aligned}
\log p(y, \beta|X) &\propto (-\frac{1}{2})(\frac{(y-X\beta)'(y-X\beta)}{\sigma^2}) + (-\frac{1}{2})(\frac{\beta'\beta}{\tau^2}) \\
&\propto -\frac{(y-X\beta)'(y-X\beta)}{2\sigma^2} - \frac{\beta'\beta}{2\tau} \\
&\propto -\frac{(y-X\beta)'(y-X\beta)\tau}{2\sigma^2\tau} - \frac{\beta'\beta\sigma^2}{2\sigma^2\tau} \\
&\propto -(y-X\beta)'(y-X\beta)\tau - \beta'\beta\sigma^2 \\
&\propto -(y-X\beta)'(y-X\beta) - \frac{\sigma^2}{\tau}\beta'\beta
\end{aligned}$$

Finally, we relate the regularization parameters λ in the ridge regression formulas. If we let $\lambda = \frac{\sigma^2}{\tau}$, then, it is equivalent to,

$$\propto -(y-X\beta)'(y-X\beta) - \lambda\beta'\beta$$

Maximize the ridge regression function is equilaent to maximize the posterior distribution.Thus the regularization parameter. λ in the ridge regression controls the regularization penalty applied to $beta$. In addition, the strength of the penalty is controlled by the ratio of prior variance of regression coefficients and the error terms in the sampling model.

3.19

Ridge

By textbook 3.47, we can get the solution by decomposing X by singular value decomposition, the solution is,

$$X\hat{\beta}^{ridge} = \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2 + \lambda} u_j' y.$$

From this expression, we are able to see that λ is in the denominator. As $\lambda \to 0$, numerator does not change, and the denominator gets smaller. Thus, the entire RHS of the equation will become larger. In addition, we know the X on the LHS is fixed, so we can conclude β will increase as $\lambda \to 0$.

Lasso

The same property does not hold for lasso. Here is a counter example: Textbook table 3.4 shows the formula for orthonormal columns. The formula for lasso is $sign(\hat{\beta})(\|\hat{\beta}_j\| - \lambda)$. Thus, when $\lambda \to 0$, we are not sure if β will increase as well since it also depends on the sign of the $\hat{\beta}_j$.

3.28

check appendix

3.30

$X^* = [X, a\mathbf{I}_p]'$, where a is constant, I_p is the identity matrix. $y^* = [y, \mathbf{0}_p]'$ Then,

$$\begin{aligned}\hat{\beta} &= argmin_{\beta}(\|y^* - X^*\beta\|_2^2 + \lambda^*|\beta|_1) \\ &= argmin_{\beta}(\|y - X\beta\|_2^2 + a^2\|\beta\|_2^2 + \lambda^*|\beta|_1)\end{aligned}$$

If we take $a^2 = \lambda\alpha$ and $\lambda^* = \lambda(1-\alpha)$, we have

$$argmin_{\beta}(\|y - X\beta\|_2^2 + \lambda\alpha\|\beta\|_2^2 + \lambda(1-\alpha)|\beta|_1)$$

Appendix

QR decomposition

From the Naive approach we know that $\vec{X}'\vec{X}\hat{\vec{\beta}} = \vec{X}y$.

Substitute $\vec{X}$ by $\bar{Q}\bar{R}$, we have:

$$(\bar{Q}\bar{R})'(\bar{Q}\bar{R})\hat{\bar{\beta}} = (\bar{Q}\bar{R})'\bar{y}$$

$$\bar{R}'\underbrace{\bar{Q}'\bar{Q}}_{I}\bar{R}\hat{\bar{\beta}} = \bar{R}'\bar{Q}'\bar{y}$$

$$\vec{R}'\bar{R}\hat{\vec{\beta}} = \bar{R}'\bar{Q}'\bar{y}$$

$$\underbrace{(\bar{R}')^{-1}\bar{R}'}_{I}\bar{R}\hat{\bar{\beta}} = \underbrace{(\bar{R}')^{-1}\bar{R}'}_{I}\bar{Q}'\bar{y}$$

$$\bar{R}\hat{\bar{\beta}} = \bar{Q}'\bar{y}$$

$$\hat{\bar{\beta}} = (\bar{R})^{-1}Q'\bar{y}$$

Singular Value Decomposition

Assume A is full rank, $A = UDV'$

Let Un be the first n colums of U, then

$$\arg\min \|Ax - b\|^2 = \arg\min \| UDV'x - b\|^2 = \arg\min \| \underbrace{U'U}_{I}DV'x - U'b\|^2$$

$$= \arg\min \| DV'x - U'b\|^2$$

We can get the minimum when $DV'x = U'b$

$$\underbrace{D^{-1}D}_{I}V'x = D^{-1}U'b$$

$$VV'x = VD^{-1}U'b$$

$$x = VD^{-1}U'b$$

$$\boxed{\begin{array}{c} Ax = b \\ \downarrow \\ \bar{X}\bar{\beta} = \bar{y} \end{array}}$$

① ② Thus, the x we solved in ① is equivalent to the β in ②.

$$\bar{\beta} = VD^{-1}U'\bar{y}$$

Cholesky decomposition

We want to minimize $\|\bar{A}\bar{x} - \bar{b}\|_2$, that is to solve $A^TAx = A^Tb$.

$$\begin{tikzcd} A^T A x \arrow[d] & = A^T b \arrow[d] \\ X^T X \beta & = X^T b. \end{tikzcd}$$

If A is positive definite, $A^TA = LL^T$

$$LL^T x = A^T b$$
$$\underbrace{L^{-1} L}_{I} L^T x = L^{-1} A^T b$$
$$x = (L^T)^{-1} L^{-1} A^T b$$

· The x in the $Ax = b$ is equivalent to the β in $\underline{X\beta = b}$.

$$\hat{\beta} = (L^T)^{-1} L^{-1} X^T b.$$

3.28

$X^{new} = [X, X]$

Then the new coefficients are $\beta^{new} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$

$$X^{new}\beta^{new} = [X \quad X]\begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = X\beta_1 + X\beta_2 = X(\beta_1 + \beta_2)$$

Then

$$\text{argmin}\left\{ \|y - X^{new}\beta^{new}\|^2 + \lambda\left(\sum_{j=1}^{p}|\beta_{1j}| + \sum_{j=1}^{p}|\beta_{2j}|\right) \right\}$$

$$\text{argmin}\left\{ \|y - X(\beta_1 + \beta_2)\|^2 + \lambda\left(\sum_{j=1}^{p}\left(|\beta_{1j}| + |\beta_{2j}| + |\beta_{1j} + \beta_{2j}| - |\beta_{1j} + \beta_{2j}|\right)\right) \right\}$$

$$\text{argmin}\left\{ \underbrace{\|y - X(\beta_1 + \beta_2)\|^2 + \lambda\left(\sum_{j=1}^{p}|\beta_{1j} + \beta_{2j}|\right)}_{①} + \underbrace{\lambda\sum_{j=1}^{p}\left(|\beta_{1j}| + |\beta_{2j}| - |\beta_{1j} + \beta_{2j}|\right)}_{②} \right\}$$

① This part become a common lasso minimization problem

We can look at ②:

$$|\beta_{1j} + \beta_{2j}| \le |\beta_{1j}| + |\beta_{2j}|$$

$|a+b| \le |a| + |b| \Rightarrow |a| + |b| - |a+b| \ge 0$

$$|\beta_{2j}| + |\beta_{1j}| - |\beta_{1j} + \beta_{2j}| \ge 0$$

We can see that ② reaches it's minimum 0 when β_{1j} and β_{2j} have the same sign, i.e. $\beta_{1j}\beta_{2j} > 0$

In addition, we know that the fitted lasso coefficient $\hat{\beta}_j = a$, comparing ① with 3.52, our new model is minimized when $\beta_{1j} + \beta_{2j} = a$

3.30

$X^* = \begin{bmatrix} X \\ aI_p \end{bmatrix}$, a is constant , I_p is the identity matrix

$y^* = \begin{bmatrix} y \\ 0_p \end{bmatrix}$

Then, $\hat{\beta} = \underset{\beta}{\arg\min} \left(\| y^* - X^*\beta \|_2^2 + \overset{*}{\lambda} |\beta|_1 \right) = \arg\min \left(\left\| \begin{matrix} y - X\beta \\ a\beta \end{matrix} \right\|_2^2 + \overset{*}{\lambda} |\beta|_1 \right)$

$= \underset{\beta}{\arg\min} \left(\| y - X\beta \|_2^2 + a^2 \|\beta\|_2^2 + \overset{*}{\lambda} |\beta|_1 \right)$

Take $a^2 = \lambda\alpha$ and $\lambda^* = \lambda(1-\alpha)$

$= \underset{\beta}{\arg\min} \left(\| y - X\beta \|_2^2 + \underline{\lambda\alpha \|\beta\|_2^2 + \lambda(1-\alpha) |\beta|_1} \right)$