R Notebook

Connor Johnson

03/19/21

Exercise 2

Problem 2

(a)

iii. because lasso is less flexible than least squares by removing variables

(b)

iii. for the same reason as lasso

(c)

ii. because non-linear methods are much more flexible than linear least squares

Problem 3

(a)

iv. An increasing s will restrict the coefficients less making it more flexible, which causes a lower error on the training set

(b)

ii. The test RSS will decrease at first as the model becomes more flexible and accurate. As s continues to increase, the model will become overfit which cause the error to increase.

(c)

iii. The variance will continually increase because the model will become more flexible. Higher flexibility models have higher variances

(d)

iv. The bias will continually decrease because the model will become more flexible. Higher flexibility models have lower biases.

(e)

v. Remains constant because this error cannot be changed for any model with different parameters

Problem 4

(a)

iii. Increasing λ makes the model less flexible. A less flexible model will have a lower error on the training set.

(b)

i. It will increase initially because the model becomes less flexible and more accurate at first. As λ continutes to decrease, the model becomes underfit which increases the error on the test set.

(c)

iv. The variance steadily decreases because the model becomes less flexible as λ increases.

(d)

iii. The bias steadily increases because the model becomes less flexible and bias is higher with more flexible models.

(e)

v. Remains constant because the irreducible error can't be changed by changing the model.

Exercise 3

Problem 9

(a)

```
set.seed(9)
idx = sample(1:777, 389)
college_train = College[idx,]
college_test = College[-idx,]
```

(b)

```
reg = lm(Apps~., data=college_train)
pred = predict(reg, college_test)
mean((pred-college_test[,"Apps"])^2)
```

```
## [1] 1536083
```

(c)

```
train.mat = model.matrix(Apps~., data=college_train)
test.mat = model.matrix(Apps~., data=college_test)
lambda_seq = 10 ^ seq(4,-2, length=100)
ridge = cv.glmnet(train.mat, college_train[,"Apps"], alpha=0, lambda = lambda_seq)
lambda_best = ridge$lambda.min
lambda_best
```

```
## [1] 43.28761
```

```
ridge.pred = predict(ridge, newx=test.mat, s=lambda_best)
mean((ridge.pred - college_test[,"Apps"])^2)
```

```
## [1] 1724421
```

(d)

```
lambda_seq = 10 ^ seq(4,-2, length=100)
ridge = cv.glmnet(train.mat, college_train[,"Apps"], alpha=1, lambda = lambda_seq)
lambda_best = ridge$lambda.min
lambda_best
```

```
## [1] 6.135907
```

```
lasso.pred = predict(ridge, newx=test.mat, s=lambda_best)

mean((lasso.pred - college_test[,"Apps"])^2)
```

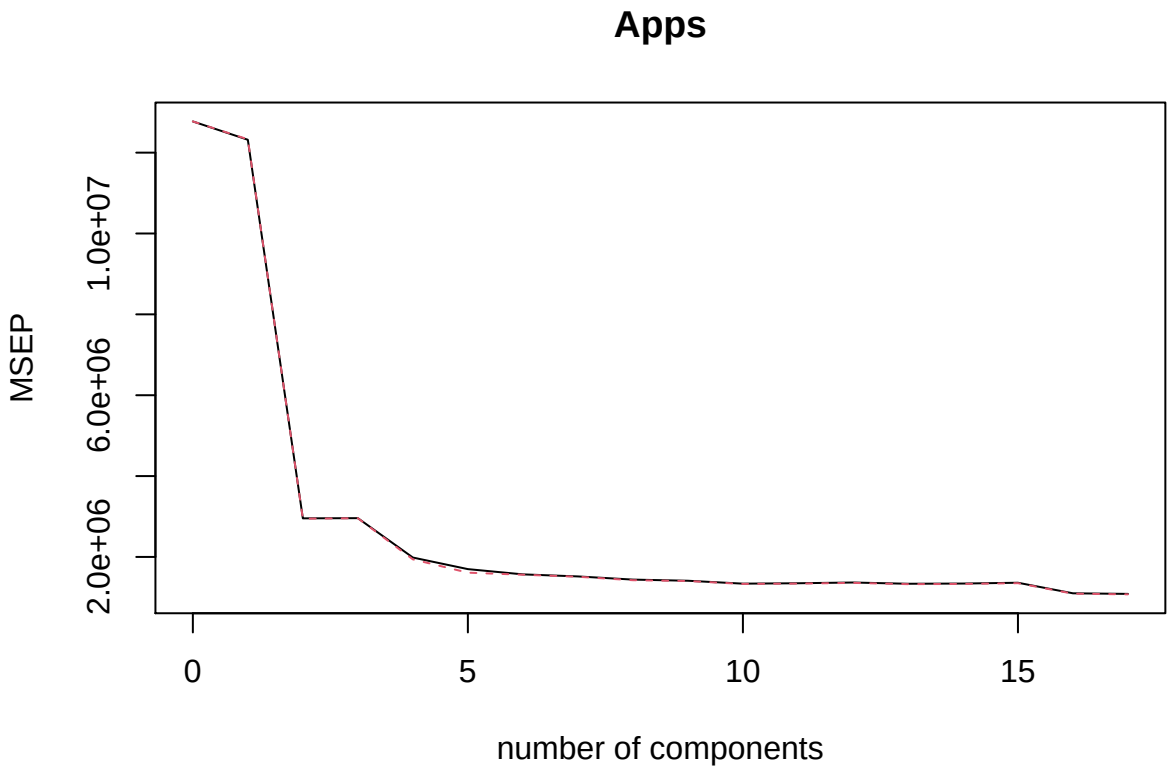
```
## [1] 1539823
```

```
predict(ridge, s=lambda_best, type="coefficients")
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -1.311953e+03
## (Intercept)  .
## PrivateYes  -3.935843e+02
## Accept       1.201212e+00
## Enroll       4.010940e-03
## Top10perc    3.611586e+01
## Top25perc   -7.217253e+00
## F.Undergrad  6.092621e-02
## P.Undergrad  1.849811e-02
## Outstate    -1.014462e-01
## Room.Board   2.387604e-01
## Books        1.095473e-01
## Personal     1.188289e-02
## PhD         -8.871612e+00
## Terminal    -3.363513e+00
## S.F.Ratio    3.121092e+01
## perc.alumni -2.727083e+00
## Expend       1.302111e-01
## Grad.Rate    8.791405e+00
```

(e)

```
pcr_model = pcr(Apps~., data=college_train, scale=TRUE, validation = "CV")
validationplot(pcr_model, val.type="MSEP")
```




```
pred = predict(pcr_model, college_test, ncomp=10)
mean((pred - college_test[,"Apps"])^2)
```

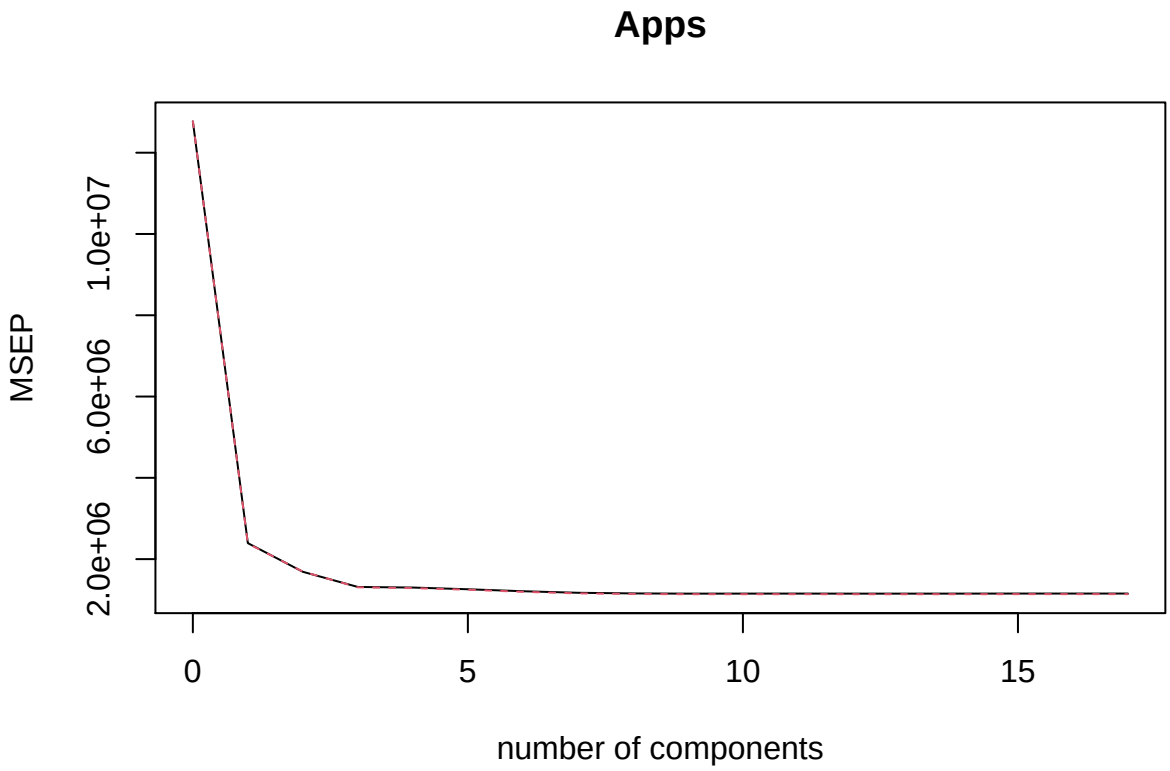
```
## [1] 3005341
```

```
print("M = 10")
```

```
## [1] "M = 10"
```

(f)

```
pls_model = plsr(Apps~., data=college_train, validation = "CV", scale=T)
validationplot(pls_model, val.type="MSEP")
```




```
pred = predict(pls_model, college_test, ncomp=11)
mean((pred - college_test[,"Apps"])^2)
```

```
## [1] 1551305
```

```
print("M = 11")
```

```
## [1] "M = 11"
```

(g) Most of the models performed reasonably well in predicting the number of college application. They all has very similar errors on the test set except for one model. The PCR model performed much more poorly on the test set compared to the four others.

Problem 10

(a)

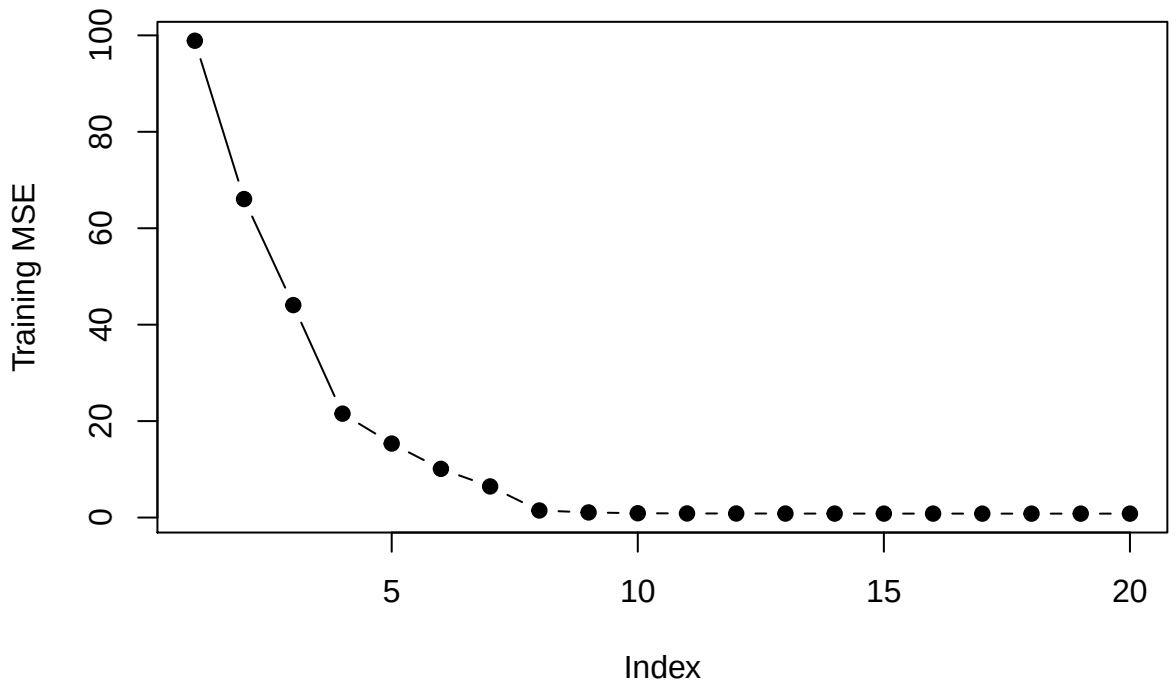
```
n = 1000
p = 20
x = matrix(rnorm(n*p), n, p)
B = rnorm(p,0,4)
idx = sample(1:p, 10)
B[idx]=0
epsilon = rnorm(n)
y = x %*% B + epsilon
```

(b)

```
idx = sample(1:n, 100)
y_train = y[idx]
y_test = y[-idx]
x_train = x[idx,]
x_test = x[-idx,]
```

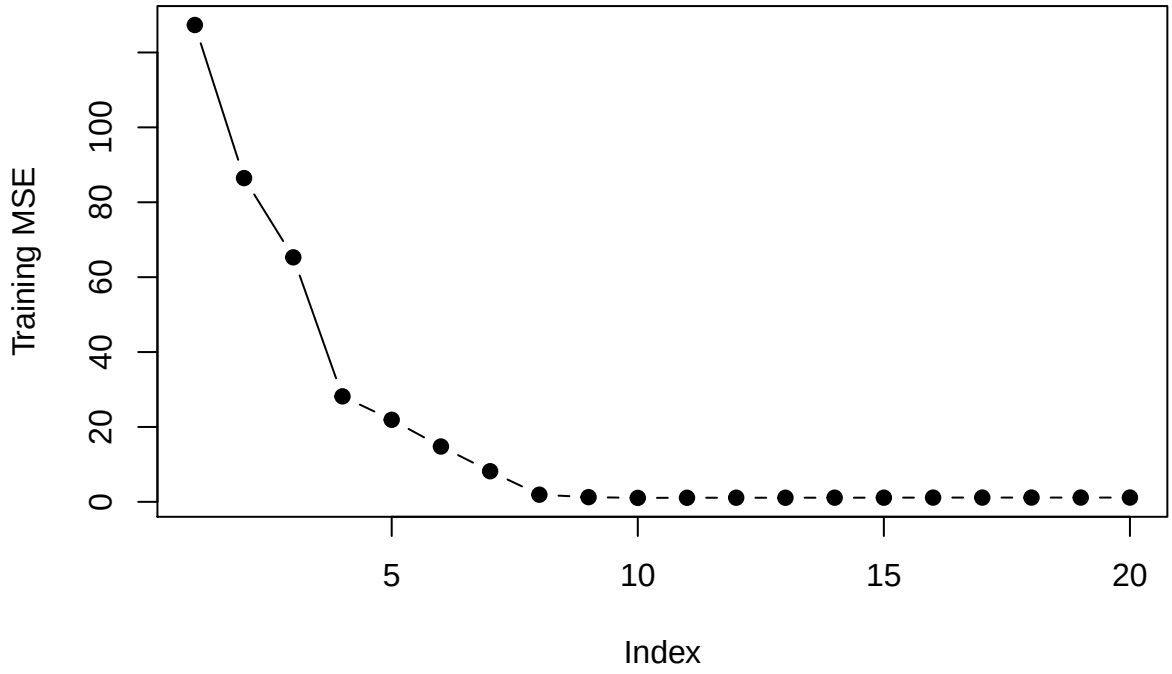
(c)

```
bestsub = regsubsets(y~., data = data.frame(x = x_train, y=y_train), nvmax = p)
errors = rep(0, p)
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.")
for (i in 1:p) {
  coefi = coef(bestsub, id = i)
  pred = as.matrix(x_train[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in%
                                                                  x_cols]
  errors[i] = mean((y_train - pred)^2)
}
plot(errors, ylab = "Training MSE", pch = 19, type = "b")
```



(d)

```r
errors = rep(0, p)
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.")
for (i in 1:p) {
  coefi = coef(bestsub, id = i)
  pred = as.matrix(x_test[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in%
                                                                 x_cols]
  errors[i] = mean((y_test - pred)^2)
}
plot(errors, ylab = "Training MSE", pch = 19, type = "b")
```



(e)

The test set MSE takes on its minimum value once it reaches the number of features that are not equal to one. Once features start getting added that are equal to one, the MSE starts to increase or stay constant on the test set.

(f)

```r
print(coef(bestsub, id = 10))
```

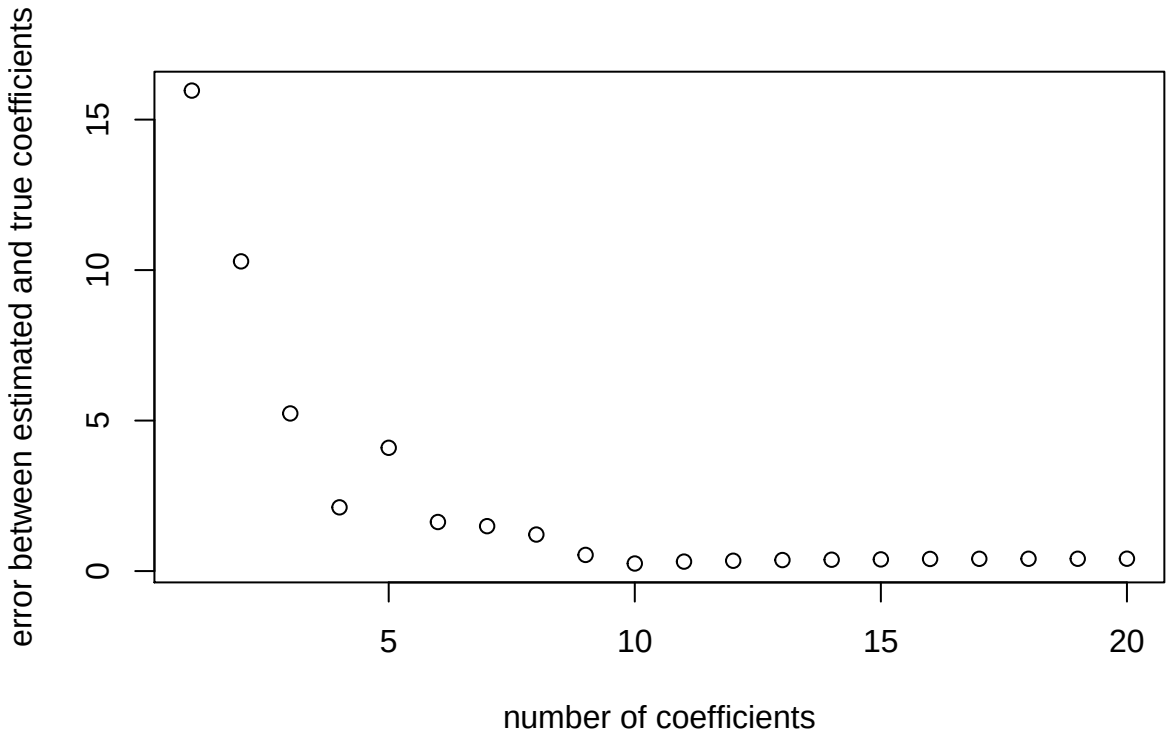
```
## (Intercept)         x.1         x.2         x.3         x.6         x.8
## -0.04803166  0.65900933 -5.80255297 -6.29084849 -2.56009408  0.39525029
##         x.9        x.12        x.13        x.14        x.15
## -5.32691077  2.49498084 -2.35447955 -7.14300027  2.57247884
```

```
print(B[B != 0])
```

```
##  [1]  0.7145116 -5.7132540 -6.2932920 -2.5651654  0.4230299 -5.1454775
##  [7]  2.5451850 -2.3639910 -7.0287702  2.5187023
```

As seen above, the coefficients for the minimized test set MSE are very similar to the coefficients that was used to generate the data.

```
b = rep(NA, p)
for (i in 1:p) {
  coefi = coef(bestsub, id = i)
  b[i] = sqrt(sum((B[x_cols %in% names(coefi)] - coefi[-1])^2) +
                sum(B[!(x_cols %in% names(coefi))])^2)
}
plot(x = 1:20, y = b, xlab = "number of coefficients", ylab = "error between estimated and true coeffic
```



The results from this graph are very similar to the MSE plot. It shows 10 as having the lowest error with lower numbers of coefficients having much higher error values.

Exercise 4

```
set.seed(7)
X = matrix(rnorm(1000), nrow=100,ncol=10)
```

```r
colnames(X) <- c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")
B = c(1,1,1,1,1,0,0,0,0,0)
epsilon = rnorm(100)
Y = X %*% B + epsilon
```

```r
X_large = matrix(rnorm(100000), nrow=10000,ncol=10)
colnames(X_large) <- c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")
B = c(1,1,1,1,1,0,0,0,0,0)
epsilon_large = rnorm(10000)
Y_large = X_large %*% B + epsilon_large
```

```r
lasso = cv.glmnet(X, Y, alpha=1, lambda = lambda_seq)
lambda_best = lasso$lambda.min
lasso.pred = predict(lasso, newx=X_large, s=lambda_best)

mean((lasso.pred - Y_large)^2)
```

```
## [1] 1.179072
```

```r
err.lasso_1 = mean((lasso.pred - Y_large)^2)
lasso_coef = predict(lasso, s=lambda_best, type="coefficients")
lasso_coef
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.1520151
## X1           1.2350491
## X2           0.8766204
## X3           1.0317901
## X4           0.7812645
## X5           0.8369576
## X6           .
## X7          -0.0180841
## X8           0.1065125
## X9           .
## X10          0.0980524
```

```r
X_d = data.frame(X[,c(-6,-9)])
fit.lm = lm(Y~., data=X_d)
summary(fit.lm)
```

```
##
## Call:
## lm(formula = Y ~ ., data = X_d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.08126 -0.64584 -0.04257  0.62635  2.88393
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) -0.16590    0.09816  -1.690   0.0944 .
## X1           1.29336    0.10791  11.986  < 2e-16 ***
## X2           0.88308    0.11133   7.932 5.30e-12 ***
## X3           1.06788    0.09551  11.181  < 2e-16 ***
## X4           0.80705    0.09707   8.314 8.55e-13 ***
## X5           0.88702    0.11005   8.060 2.87e-12 ***
## X7          -0.06234    0.11304  -0.552   0.5826
## X8           0.14526    0.10138   1.433   0.1553
## X10          0.13584    0.10633   1.278   0.2047
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9492 on 91 degrees of freedom
## Multiple R-squared:  0.876,  Adjusted R-squared:  0.8651
## F-statistic: 80.33 on 8 and 91 DF,  p-value: < 2.2e-16
```

```
X_large_d = data.frame(X_large)

OLS.pred = predict(fit.lm, X_large_d)
err.OLS_1 = mean((OLS.pred - Y_large)^2)
err.OLS_1
```

```
## [1] 1.213267
```

```
p = 10
err.lasso = numeric(p)
err.OLS = numeric(p)
set.seed(7)
B = c(1,1,1,1,1,0,0,0,0,0)
sigma=1

for (i in 1:p){
  X = matrix(rnorm(1000), nrow=100,ncol=10)
  colnames(X) <- c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")

  epsilon = rnorm(100, 0, sigma)
  Y = X %*% B + epsilon



  lasso = cv.glmnet(X, Y, alpha=1, lambda = lambda_seq)
  lambda_best = lasso$lambda.min
  lasso.pred = predict(lasso, newx=X_large, s=lambda_best)

  err.lasso[i] = mean((lasso.pred - Y_large)^2)
  lasso_coef = predict(lasso, s=lambda_best, type="coefficients")

  for (a in 2:11){
    if(lasso_coef[,1][a] == 0){
      X = X[,-(a-1)]
    }
  }
  X = data.frame(X)
```

```
  fit.lm = lm(Y~., data=X)

  OLS.pred = predict(fit.lm, X_large_d)
  err.OLS[i] = mean((OLS.pred - Y_large)^2)
}

mean(err.OLS)
```

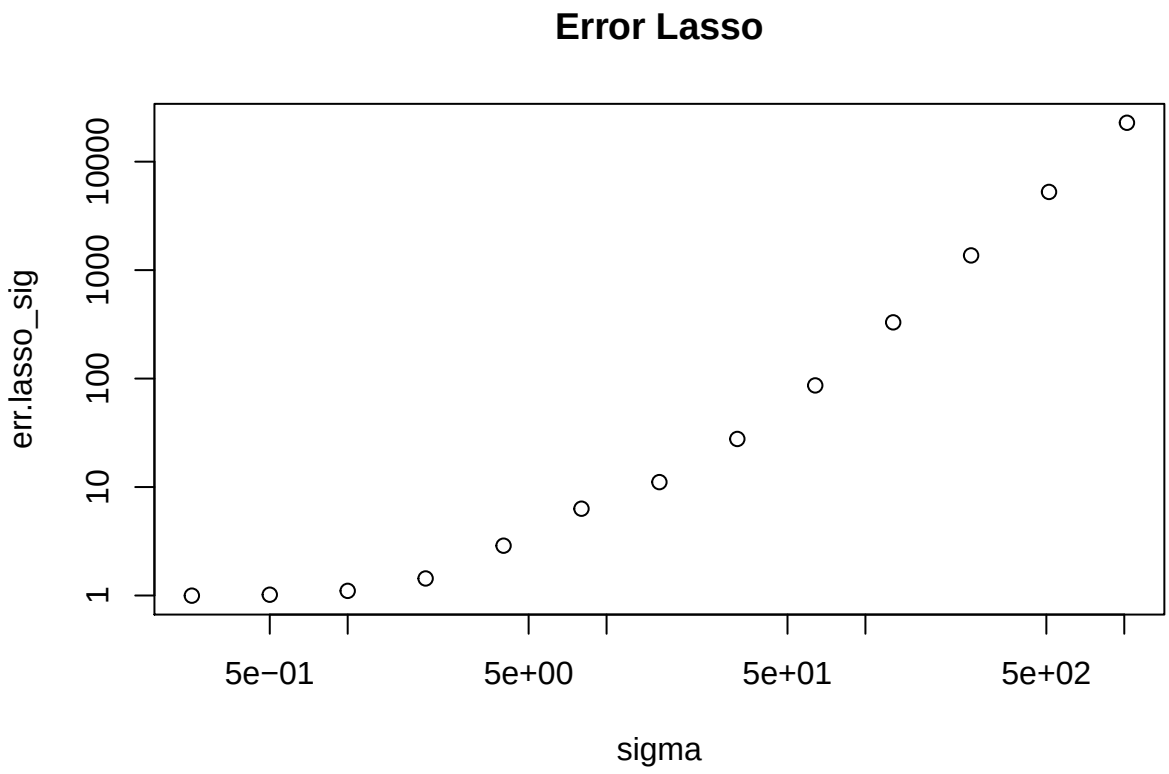
```
## [1] 1.137825
```

```
mean(err.lasso)
```

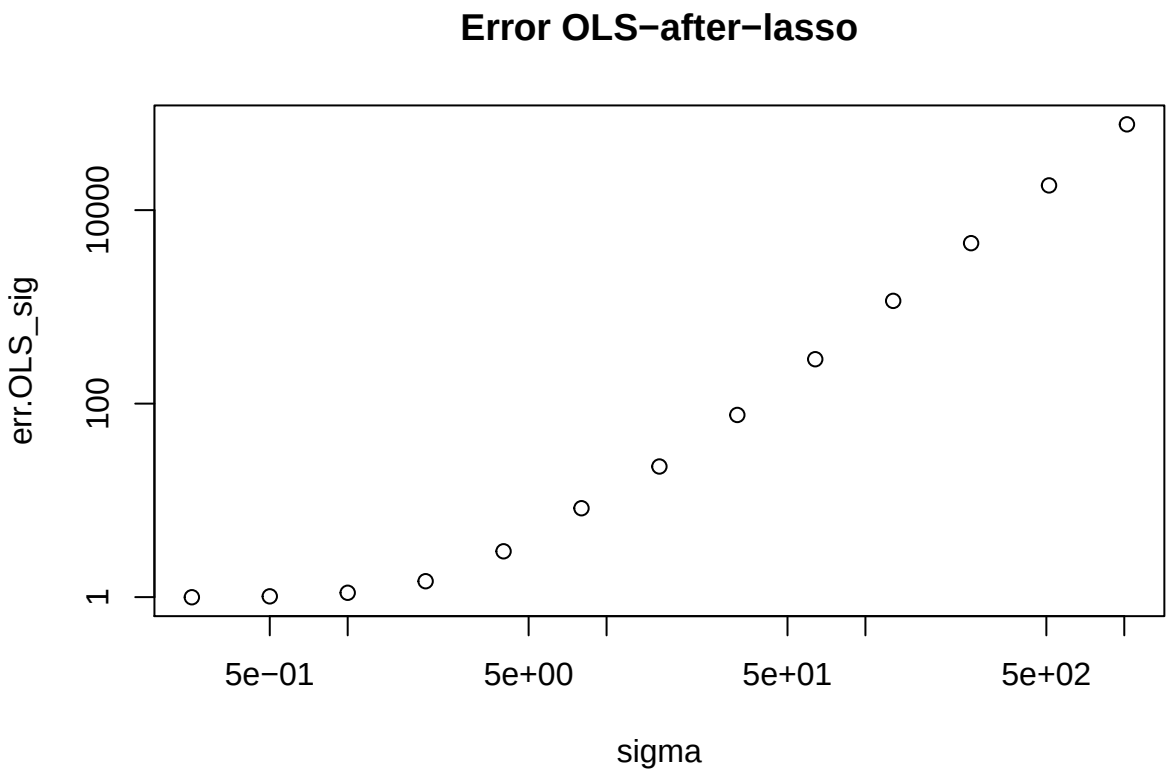
```
## [1] 1.128099
```

```
p = 1000
err.lasso = numeric(p)
err.OLS = numeric(p)
set.seed(7)
B = c(1,1,1,1,1,0,0,0,0,0)
sigma= 2^seq(-2,10, length=13)

err.lasso_sig = numeric(length(sigma))
err.OLS_sig = numeric(length(sigma))

for (num in 1:length(sigma)){
  for (i in 1:p){
    X = matrix(rnorm(1000), nrow=100,ncol=10)
    colnames(X) <- c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")

    epsilon = rnorm(100, 0, sigma[num])
    Y = X %*% B + epsilon



    lasso = cv.glmnet(X, Y, alpha=1, lambda = lambda_seq)
    lambda_best = lasso$lambda.min
    lasso.pred = predict(lasso, newx=X_large, s=lambda_best)

    err.lasso[i] = mean((lasso.pred - Y_large)^2)
    lasso_coef = predict(lasso, s=lambda_best, type="coefficients")

    for (a in 2:11){
      if(lasso_coef[,1][a] == 0){
        X = X[,-(a-1)]
      }
    }
    X = data.frame(X)
    fit.lm = lm(Y~., data=X)

    OLS.pred = predict(fit.lm, X_large_d)
    err.OLS[i] = mean((OLS.pred - Y_large)^2)
  }
```

```
  err.lasso_sig[num] = mean(err.lasso)
  err.OLS_sig[num] = mean(err.OLS)
}
plot(sigma, err.lasso_sig, log="xy", main="Error Lasso")
```




```
plot(sigma, err.OLS_sig, log="xy", main = "Error OLS-after-lasso")
```



(f) We can see from the graphs that at low SNRs, the procedures with high degrees of freedom do poorly. When the SNR is lower, they perform very well. This is because it is harder for a model to predict the signal when there is lot of noise covering it. This is especially true in models with high degrees of freedom becuase they are more likely to overfit.