Devon Zuegel // 3 Nov 2015 // STATS 202

Problem Set 5

web.stanford.edu/class/stats202/content/viewhw.html?hw5

Notes

- The **least squares** fitting procedure estimates B_0, B_1, ..., B_p by minimizing:

$$\text{RSS} = \sum_i^n \left(y_i - \beta_0 - \sum_j^p \beta_j x_{ij} \right)^2$$

- **Ridge regression** is very similar to least squares, except the coefficients are estimated by minimizing:

$$\text{Ridge} = \text{RSS} + \lambda \sum_j^p \beta_j^2$$

Problem 1

Chapter 6, Exercise 3 (p. 260)

$$\sum_i^n \left(y_i - \beta_0 - \sum_j^p \beta_j x_{ij} \right)^2 \text{ subject to } \sum_j^p |\beta_j| \le s$$

a. iii, **training RSS** steadily increases — as `s` increases, we force more of the coefficients to be set to `0` (placing tighter constraints on the model).
b. ii, **test RSS** decreases then increases — as `s` increases initially, we will remove the noisier variables which were causing overfitting, which improves our model by decreasing variance. At some point though, the constraints will become to much and remove important coefficients.
c. iv, **variance** steadily decreases — noiser variables will be removed as `s` increases.
d. iii, **squared bias** steadily increases — as more variables are removed, we depend more on a

smaller number of predictors and the biases they introduce.

e. v, **irreducible error** remains constant — our decisions don't effect this, it is an immutable feature of the problem.

Problem 2

Chapter 6, Exercise 4 (p. 260)

$$\sum_{i}^{n}\left(y_i - \beta_0 - \sum_{j}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j}^{p}\beta_j^2$$

a. iii, **training RSS** steadily increases — (same reasoning as 1a)
b. ii, **test RSS** decreases then increases — (same reasoning as 1b)
c. iv, **variance** steadily decreases — (same reasoning as 1c)
d. iii, **squared bias** steadily increases — (same reasoning as 1d)
e. v, **irreducible error** remains constant — (same reasoning as 1e)

Problem 3

Chapter 6, Exercise 1 (p. 259)

We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain p + 1 models, containing 0, 1, 2, … , p predictors.

Part A

*Which of the three models with k predictors has the **smallest training RSS**?*

We will get the smallest training RSS by using **best subset selection**. Forward and backward won't be able to minimize the training RSS as much, because best chooses the model with the lowest training RSS out of all *all possible* `k` *-predictor models*. Forward and backward can come up with this same model, but because they explore the space of models heuristically rather than how best explores the space exhaustively.

Part B

> *Which of the three models with k predictors has the **smallest test RSS**?*

The answer to this question depends on the specifics of the problem. Any of these methods can overfit to the data.

Part C

SUBPART I

> *The predictors in the `k` -variable model identified by forward stepwise are a subset of the predictors in the `k + 1` -variable model identified by forward stepwise selection.*

True — the variable model with `k + 1` predictors has the same group of predictors as in the model with just `k` predictors, with the addition of the `k + 1` th-best predictor predictor (a.k.a. the next predictor that results in the largest improvement to RSS).

SUBPART II

> *The predictors in the `k` -variable model identified by backward stepwise are a subset of the predictors in the `k + 1` - variable model identified by backward stepwise selection.*

True – similar to part i, the `k` -predictor model has all of the predictors in the `k + 1` -predictor model, minus the worst predictor (a.k.a. the predictor that results in the smallest improvement to RSS).

SUBPART III

> *The predictors in the `k` -variable model identified by backward stepwise are a subset of the predictors in the `k + 1` - variable model identified by forward stepwise selection.*

False — Forward and backward can result in different / disjoint sets.

SUBPART IV

> *The predictors in the `k` -variable model identified by forward stepwise are a subset of the predictors in the `k + 1` -variable model identified by backward stepwise selection.*

False — Forward and backward can result in different / disjoint sets.

SUBPART V

> *The predictors in the `k` -variable model identified by best subset are a subset of the predictors in the `k + 1` -variable model identified by best subset selection.*

False — Forward and backward can result in different / disjoint sets.

Problem 4

Chapter 6, Exercise 8 (p. 262). For consistency, in parts (b) and (f) make all non-zero coefficients equal to 1.

In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

Part A

> Use the `rnorm()` function to generate a predictor `X` of length `n = 100`, as well as a noise vector `ε` of length `n = 100`.

```
set.seed(5)
X = rnorm(100, mean = 0, sd = 1)
e = rnorm(100, mean = 0, sd = 0.5)
```

Part B

> Generate a response vector `Y` of length `n = 100` according to the model \(Y = β0 +β1X +β2 X_2 +β3X_3 +ε\), where `β0`, `β1`, `β2`, and `β3` are constants of your choice.

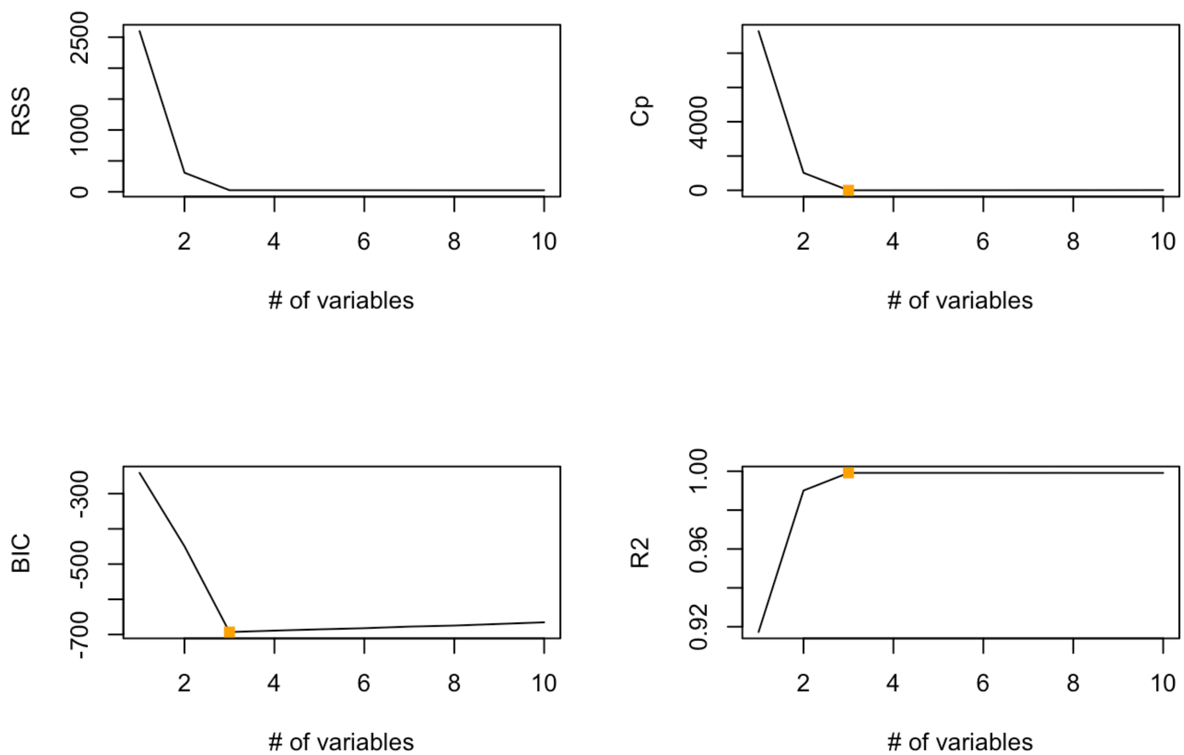
```
b0 = 2
b1 = 3
b2 = 4
b3 = 5

Y = b0 + b1*X + b2*X^2 + b3*X^3 + e
```

Part C

> Use the `regsubsets()` function to perform best subset selection in order to choose the best model containing the predictors \(X, X_2, . . . , X_{10}\). What is the best model obtained according to Cp, BIC, and adjusted R2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained. Note you will need to use the `data.frame()` function to create a single data set containing both `X` and `Y`.

```
library(ISLR)
library(leaps)

df      = data.frame(Y = Y, X = X)
fit     = regsubsets(Y ~ poly(X, 10, raw = T), data = df, nvmax = 10)
fit_sum = summary(fit)

par(mfrow = c(2,2))  # Display graphs in two rows, two columns
x_axis_label = '# of variables'
```

```r
plot   (fit_sum$rss,xlab = x_axis_label, ylab = 'RSS', type = 'l')

plot   (fit_sum$cp,xlab = x_axis_label, ylab = 'Cp', type = 'l')
points (which.min(fit_sum$cp), fit_sum$cp[which.min(fit_sum$cp)], col = 'orange', pch = 15, cex

plot   (fit_sum$bic,xlab = x_axis_label,  ylab = 'BIC', type = 'l')
points (which.min(fit_sum$bic), fit_sum$bic[which.min(fit_sum$bic)], col = 'orange', pch = 15, 

plot   (fit_sum$adjr2,xlab = x_axis_label, ylab = 'R2', type = 'l')
points (which.max(fit_sum$adjr2), fit_sum$adjr2[which.max(fit_sum$adjr2)], col = 'orange', pch
```




```r
coef(fit, 3)
```

```
##      (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##          2.036014          3.208468          3.972592
## poly(X, 10, raw = T)3
##          4.944881
```

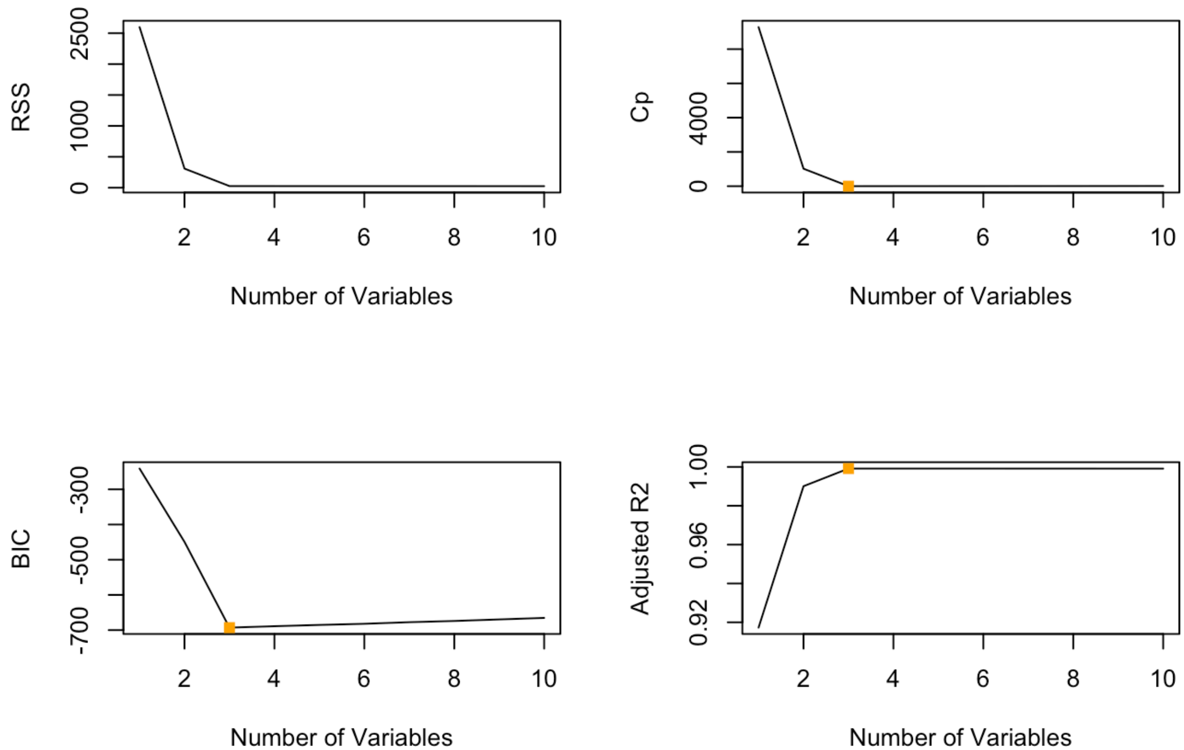
```r
coef(fit, 4)
```

```
##      (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##       2.04776954       3.30073103       3.96472905
## poly(X, 10, raw = T)3 poly(X, 10, raw = T)5
##       4.84498919       0.01750676
```

Part D

> Repeat c using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in c?

```
fit     = regsubsets(Y~poly(X, 10, raw = T), data = df, nvmax = 10, method = 'forward')
fit_sum = summary(fit)

par(mfrow = c(2, 2))

plot(fit_sum$rss, xlab = 'Number of Variables', ylab = 'RSS', type = 'l')

plot(fit_sum$cp, xlab = 'Number of Variables', ylab = 'Cp', type = 'l')
points(which.min(fit_sum$cp), fit_sum$cp[which.min(fit_sum$cp)], col = 'orange', pch = 15, cex =

plot(fit_sum$bic, xlab = 'Number of Variables', ylab = 'BIC',  type = 'l')
points(which.min(fit_sum$bic), fit_sum$bic[which.min(fit_sum$bic)], col = 'orange', pch = 15, co

plot(fit_sum$adjr2, xlab = 'Number of Variables', ylab = 'Adjusted R2', type = 'l')
points(which.max(fit_sum$adjr2), fit_sum$adjr2[which.max(fit_sum$adjr2)], col = 'orange', pch =
```




```
coef(fit, 3)   # Cp
```

```
##      (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##        2.036014         3.208468         3.972592
## poly(X, 10, raw = T)3
##        4.944881
```

```
coef(fit, 3)   # BIC
```

```
##      (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##        2.036014         3.208468         3.972592
## poly(X, 10, raw = T)3
##        4.944881
```

```
coef(fit, 3)   # Adjusted R2
```

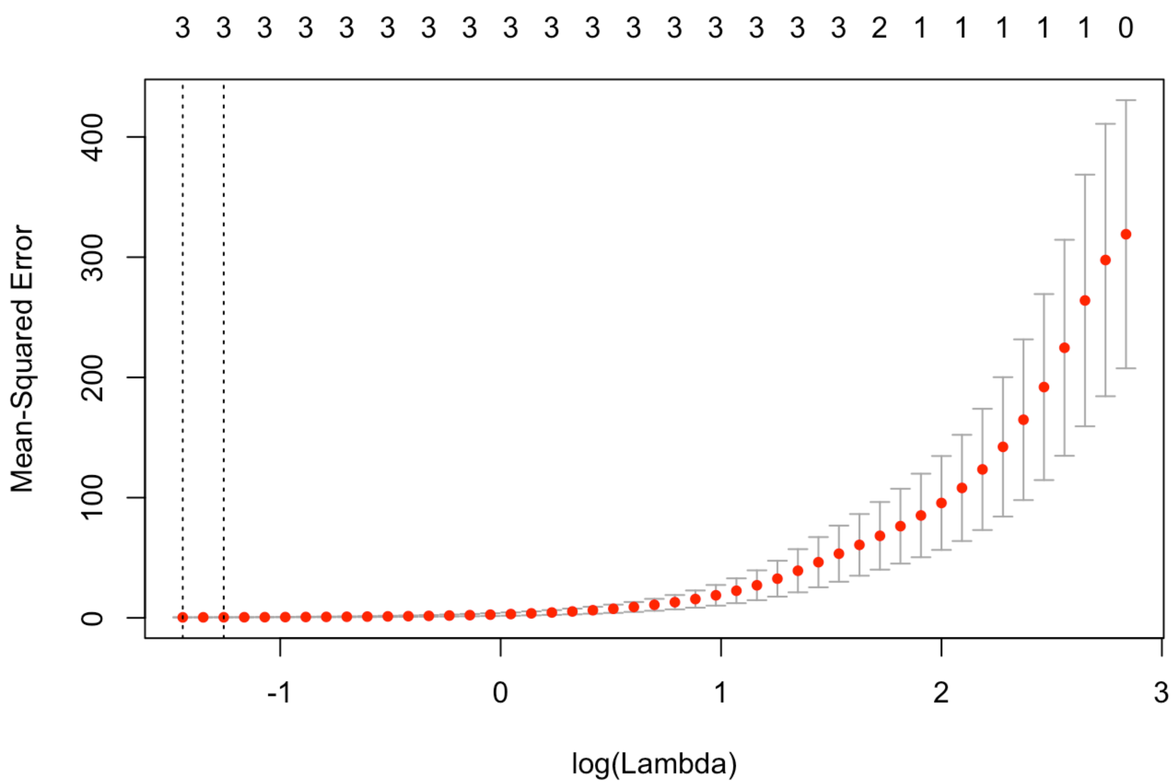
```
##      (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)2
##         2.036014            3.208468            3.972592
## poly(X, 10, raw = T)3
##         4.944881
```

Part E

> Now fit a lasso model to the simulated data, again using \(X,X_{2}, . . . , X_{10}\) as predictors. Use cross-validation to select the optimal value of `λ`. Create plots of the cross-validation error as a function of `λ`. Report the resulting coefficient estimates, and discuss the results obtained.

```
library(glmnet)
```

```
matrix = model.matrix(Y ~ poly(X, 10, raw = T), data = df)[, -1]

cv_results = cv.glmnet(matrix, Y, alpha = 1)
plot(cv_results)
```




```
min_lambda = cv_results$lambda.min
```

```
## [1] 0.2363694
```

```
lasso_model = glmnet(matrix, Y, alpha = 1, lambda = min_lambda)
coef(lasso_model)
```

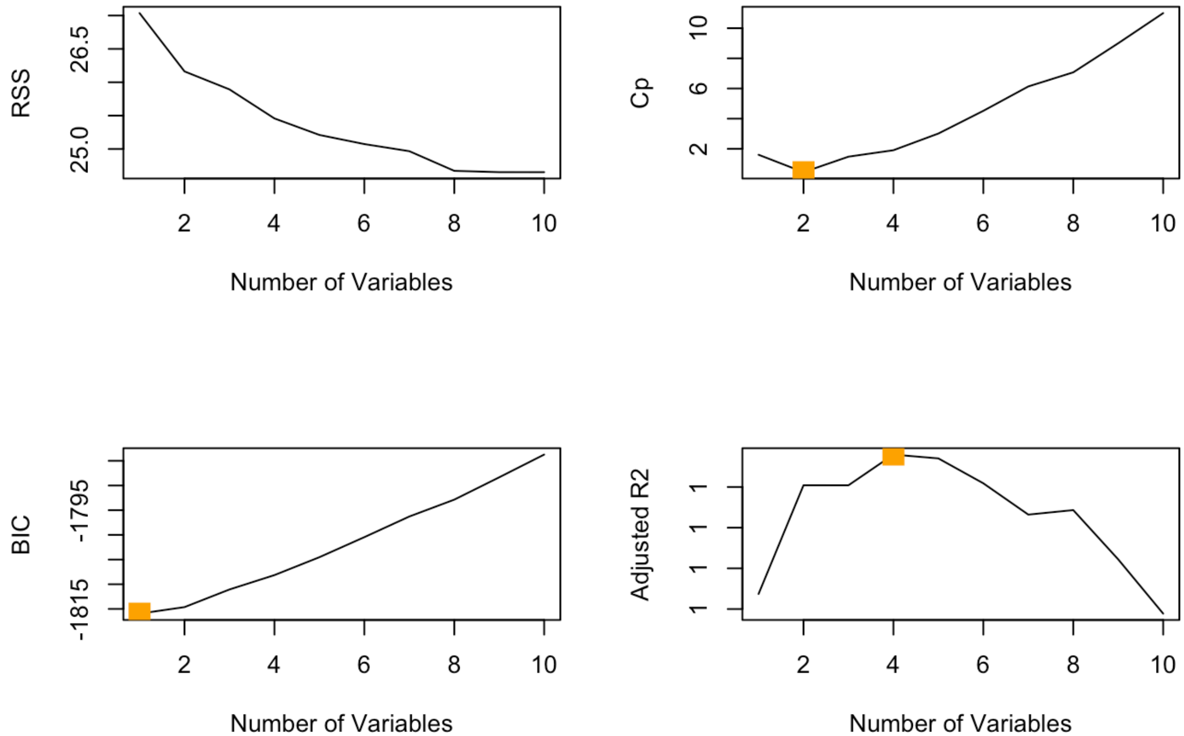
```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     s0
## (Intercept)         2.200167
## poly(X, 10, raw = T)1  3.098367
## poly(X, 10, raw = T)2  3.800216
## poly(X, 10, raw = T)3  4.901091
## poly(X, 10, raw = T)4  .
## poly(X, 10, raw = T)5  .
## poly(X, 10, raw = T)6  .
## poly(X, 10, raw = T)7  .
## poly(X, 10, raw = T)8  .
```

```
## poly(X, 10, raw = T)9 .
## poly(X, 10, raw = T)10 .
```

Part F

> Now generate a response vector `Y` according to the model \(Y = β0 + β7X_7 + ε\), and perform best subset selection and the lasso. Discuss the results obtained.

```
set.seed(5)
X = rnorm(100, mean = 0, sd = 1)
e = rnorm(100, mean = 0, sd = 0.5)

b0 = 8
b7 = 77

Y = b0 + b7*X^7 + e

df = data.frame(Y = Y, X = X)
fit = regsubsets(Y~poly(X, 10, raw = T), data = df, nvmax = 10)
fit_sum = summary(fit)

par(mfrow = c(2, 2))
plot(fit_sum$rss,   xlab = 'Number of Variables', ylab = 'RSS',          type = 'l')

plot(fit_sum$cp,    xlab = 'Number of Variables', ylab = 'Cp',           type = 'l')
points(which.min(fit_sum$cp), fit_sum$cp[which.min(fit_sum$cp)], col = 'orange', pch = 15, cex =

plot(fit_sum$bic,   xlab = 'Number of Variables', ylab = 'BIC',          type = 'l')
points(which.min(fit_sum$bic), fit_sum$bic[which.min(fit_sum$bic)], col = 'orange', pch = 15, c

plot(fit_sum$adjr2, xlab = 'Number of Variables', ylab = 'Adjusted R2',  type = 'l')
points(which.max(fit_sum$adjr2), fit_sum$adjr2[which.max(fit_sum$adjr2)], col = 'orange', pch =
```




```
coef(fit, 2) # Cp
```

```
##     (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)7
##       8.0085539          0.1222336         76.9985505
```

```
coef(fit, 1) # BIC
```

```
##     (Intercept) poly(X, 10, raw = T)7
##        8.009321           76.999629
```

```
coef(fit, 4) # R2
```

```
##        (Intercept) poly(X, 10, raw = T)1 poly(X, 10, raw = T)3
##          8.0346145           0.4754667          -0.5159204
## poly(X, 10, raw = T)5 poly(X, 10, raw = T)7
##          0.1770933          76.9812606
```

```
matrix = model.matrix(Y~poly(X, 10, raw = T), data = df)[, -1]
```

Problem 5

Chapter 6, Exercise 9 (p. 263)

In this exercise, we will predict the number of applications received using the other variables in the College data set.

Part A

> Split the data set into a training set and a test set.

```
library(ISLR)
library(Matrix)

df.train      = sample(c(T, F), nrow(College), rep = T)
df.test       = (!df.train)
College.train = College[df.train, , drop = F]
College.test  = College[df.test, , drop = F]
```

Part B

> Fit a linear model using least squares on the training set, and report the test error obtained.

```
fit = lm(Apps~., data = College.train)
summary(fit)
```

```
##
## Call:
## lm(formula = Apps ~ ., data = College.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4262.9  -438.1     5.0   336.2  6491.3
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  281.47228  561.63509   0.501 0.616562
## PrivateYes  -573.10200  210.03693  -2.729 0.006675 **
## Accept         1.81439    0.05255  34.529  < 2e-16 ***
## Enroll        -0.59354    0.35991  -1.649 0.100002
## Top10perc     46.98739    7.90855   5.941  6.7e-09 ***
## Top25perc    -12.19785    6.36139  -1.917 0.055972 .
## F.Undergrad   -0.10458    0.07105  -1.472 0.141896
## P.Undergrad    0.08193    0.05563   1.473 0.141688
## Outstate      -0.09189    0.02757  -3.333 0.000949 ***
## Room.Board     0.08215    0.06568   1.251 0.211826
## Books          0.25487    0.29696   0.858 0.391311
## Personal      -0.09442    0.08686  -1.087 0.277755
## PhD           -4.05192    6.23272  -0.650 0.516041
## Terminal      -5.03678    6.77242  -0.744 0.457534
## S.F.Ratio     -8.01260   16.21690  -0.494 0.621546
## perc.alumni   -0.80362    5.78221  -0.139 0.889543
## Expend         0.03129    0.01565   2.000 0.046285 *
## Grad.Rate     10.76838    4.57660   2.353 0.019167 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1000 on 358 degrees of freedom
## Multiple R-squared:  0.9401, Adjusted R-squared:  0.9373
## F-statistic: 330.5 on 17 and 358 DF,  p-value: < 2.2e-16
```

```
pred = predict(fit, College.test)
tss  = sum((College.test$Apps - mean(College.test$Apps))^2)
rss  = sum((pred - College.test$Apps)^2)

rsq = 1 - rss/tss
rsq
```

```
## [1] 0.8987648
```

Part C

> Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```
library(glmnet)
```

```
## Loading required package: foreach
## Loaded glmnet 2.0-2
```

```
College.train.x = scale(model.matrix(Apps ~ ., data = College.train)[,-1],scale = T, center =
College.train.y = College.train$Apps

College.test.x = scale(model.matrix(Apps ~ ., data = College.test)[,  - 1], attr(College.trair
College.test.y = College.test$Apps

cv_result = cv.glmnet(College.train.x, College.train.y, alpha = 0)
min_lambda = cv_result$lambda.min
min_lambda
```

```
## [1] 418.4641
```

```
lasso_model = glmnet(College.train.x, College.train.y, alpha = 0, lambda = min_lambda)

pred = predict(lasso_model, College.test.x, s = min_lambda)
rss  = sum((pred - College.test$Apps)^2)
tss  = sum((College.test$Apps - mean(College.test$Apps))^2)

rsq = 1 - rss/tss
rsq
```

```
## [1] 0.9132978
```

Part D

> Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
cv_result = cv.glmnet(College.train.x, College.train.y, alpha = 1)
min_lambda = cv_result$lambda.min
min_lambda
```

```
## [1] 3.555913
```

```
lasso_model = glmnet(College.train.x, College.train.y, alpha = 1, lambda = min_lambda)

pred = predict(lasso_model, College.test.x, s = min_lambda)
rss = sum((pred - College.test$Apps)^2)
tss = sum((College.test$Apps - mean(College.test$Apps))^2)
```

```
rsq = 1 - rss/tss
rsq
```

```
## [1] 0.8997127
```

```
sum(coef(lasso_model)[, 1] == 0)    # Get the # of coefficients that equal 0
```

```
## [1] 0
```

```
names(coef(lasso_model)[, 1][coef(lasso_model)[, 1] == 0])
```

```
## character(0)
```

Part E

> Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
library(pls)
```

```
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:stats':
##
##     loadings
```

```
fit  = pcr(Apps ~ ., data = College.train, scale = T, validation = 'CV')
summary(fit)
```

```
## Data:    X dimension: 376 17
##  Y dimension: 376 1
## Fit method: svdpc
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3999     3988     2222     2230     2227     1933     1875
## adjCV         3999     3991     2217     2227     2223     1885     1864
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1881     1839     1792      1787      1813      1819      1833
## adjCV     1871     1834     1783      1779      1804      1809      1825
##        14 comps  15 comps  16 comps  17 comps
## CV         1845      1680      1155      1165
## adjCV      1838      1641      1144      1153
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      31.173    56.31    63.78    69.89    75.25    80.32    84.23
## Apps    1.022    70.77    70.83    71.49    79.91    81.11    81.27
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       87.40    90.40     92.76     94.89     96.79     97.79     98.70
## Apps    81.81    82.93     83.13     83.45     83.57     83.75     83.77
##       15 comps  16 comps  17 comps
## X        99.42     99.91    100.00
## Apps     91.44     93.99     94.01
```

```
pred = predict(fit, College.test, ncomp = 17)
rss  = sum((pred - College.test$Apps)^2)
tss  = sum((College.test$Apps - mean(College.test$Apps))^2)

rsq  = 1 - rss/tss
rsq
```

```
## [1] 0.8987648
```

Part F

> Fit a PLS model on the training set, with M chosen by cross- validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
fit = plsr(Apps ~ ., data = College.train, scale = T, validation = 'CV')
summary(fit)
```

```
## Data:   X dimension: 376 17
##  Y dimension: 376 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           3999     2090     1790     1701     1569     1340     1187
## adjCV        3999     2082     1774     1687     1526     1308     1173
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1166     1160     1160      1161      1162      1163      1165
## adjCV     1155     1149     1149      1150      1150      1151      1154
##        14 comps  15 comps  16 comps  17 comps
## CV         1164      1165      1164      1164
## adjCV      1153      1153      1153      1153
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       25.13    35.93    61.20    63.64    67.73    72.45    77.29
## Apps    75.80    84.65    86.77    91.77    93.39    93.85    93.92
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       80.58    83.10     86.47     89.97     91.89     93.70     95.19
## Apps    93.96    93.99     94.00     94.00     94.00     94.01     94.01
##       15 comps  16 comps  17 comps
## X        96.95     99.07    100.00
## Apps     94.01     94.01     94.01
```

```
pred = predict(fit, College.test, ncomp = 7)
rss  = sum((pred - College.test$Apps)^2)
tss  = sum((College.test$Apps - mean(College.test$Apps))^2)

rsq = 1 - rss/tss
rsq
```

```
## [1] 0.8951694
```

Part G

> Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

- All models had similar performance, with RSQ values of 89.5%, 89.3%, 89.6%, 89.5%, and 89.5& for least squares, ridge regression, lasso model, PCR, and PLS respectively.
- PLS chose a model with the fewest number of predictors (7 of the 17 available) and minimized the majority of the variance while at the same time performing well on test sets.

- Lasso performed similarly well, but it used 14 of the predictors.

Problem 6

Chapter 6, Exercise 10 (p. 263)

> We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set.

Part A

> Generate a data set with `p = 20` features, `n = 1000` observations, and an associated quantitative response vector generated according to the model `Y = Xβ + ε`, where `β` has some elements that are exactly equal to `0`.

```
set.seed(19)   # Some randomization in the setup.

p = 20
n = 1000

X = matrix(rnorm(p * n), ncol = p, nrow = n)

# Randomly set some elements in beta equal to zero.
beta                 = rnorm(p, sd = 10)
num_rand_zeroes    = sample(0:p/3)
rand_zeroes        = sample(seq(1, length(beta)), num_rand_zeroes, replace = F)
beta[rand_zeroes] = 0

e = rnorm(n)
Y = as.vector(X * beta + e)
```

Part B

> Split your data set into a training set containing `100` observations and a test set containing `900` observations.

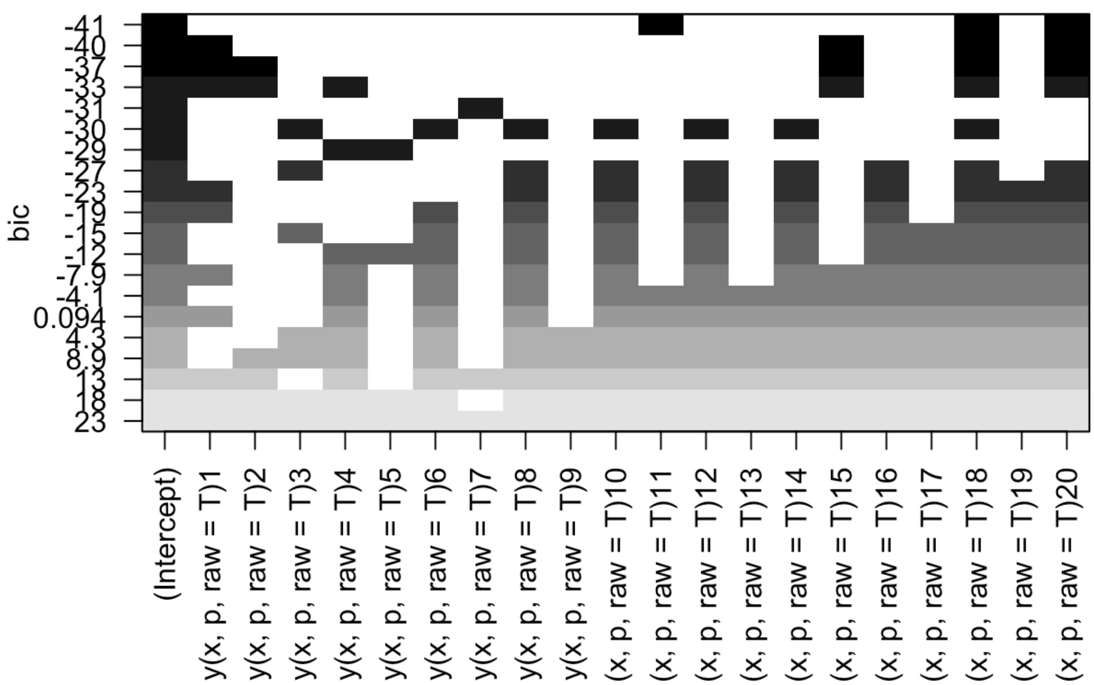
```
n_training_observations = 100
train    = sample(1:nrow(X), n_training_observations)
test     = (-train)

X.train = X[train]
Y.train = Y[train]
X.test  = X[test ]
Y.test  = Y[test ]
```

```
df.train = data.frame(y = Y.train, x = X.train)
df.test  = data.frame(y = Y.test,  x = X.test)
```

Part C

> Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

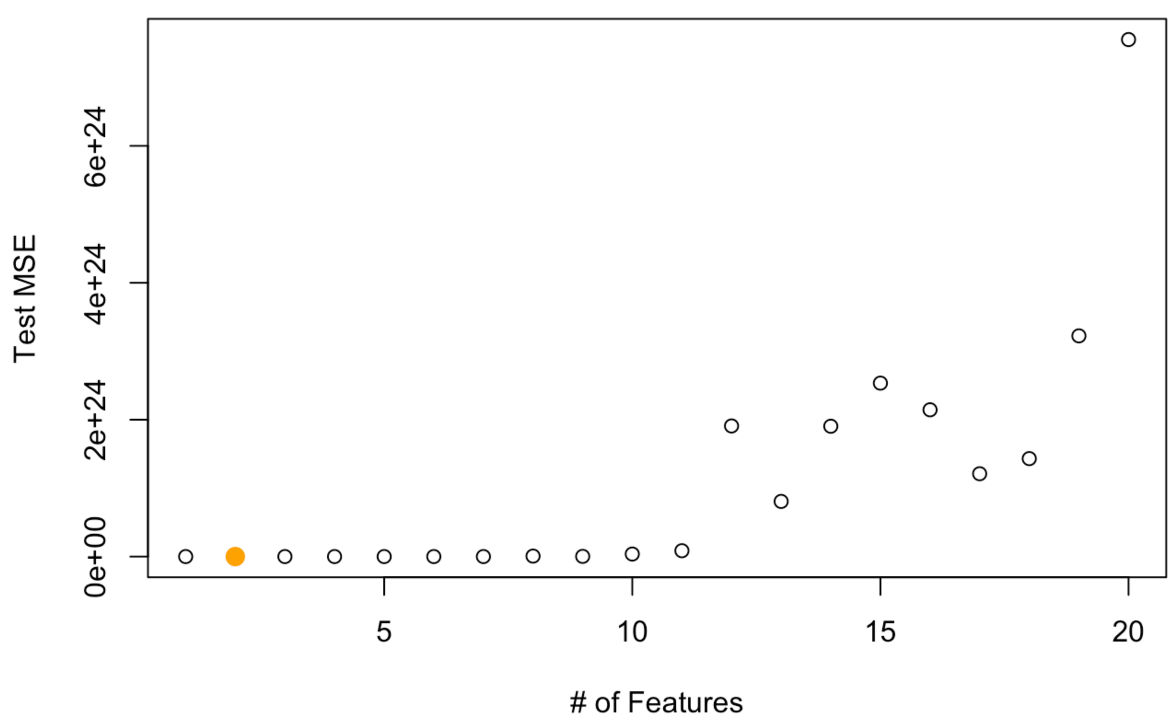
```
library(leaps)
fit      = regsubsets(y ~ poly(x, p, raw = T), data = df.train, nvmax = p)

plot(fit)
```



Part D

> Plot the test set MSE associated with the best model of each size.

```
mse = function(prediction, real) {
  mean((prediction - real)^2)
}

predict_regsubsets = function(obj, newdata, id) {
  form   = as.formula(obj$call[[2]]) # Extract formula.
  matrix = model.matrix(form, newdata)
  coefic = coef(obj, id = id)
  xvars  = names(coefic)
  matrix[, xvars] * coefic
}

test.mse = sapply(1:p, function(id) {
  prediction = predict_regsubsets(fit, df.test, id)
  mse(prediction, Y.test)
})

plot(seq(1:p), test.mse, xlab = '# of Features', ylab = 'Test MSE')
points(which.min(test.mse), test.mse[which.min(test.mse)], col = 'orange', cex = 2, pch = 20)
```




```
coef(fit, id = which.min(test.mse))
```

```
##       (Intercept) poly(x, p, raw = T)4 poly(x, p, raw = T)5
##        -1.0638214          0.1926629          -0.2965513
```

Part E

> For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

The test MSE is low and approximately constant from 0-11 features. After that, it shoots up. This is expected – since we set beta to `0` for some of the features, it's better to simply throw those out of our model since they don't provide any information.

Part F

> How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

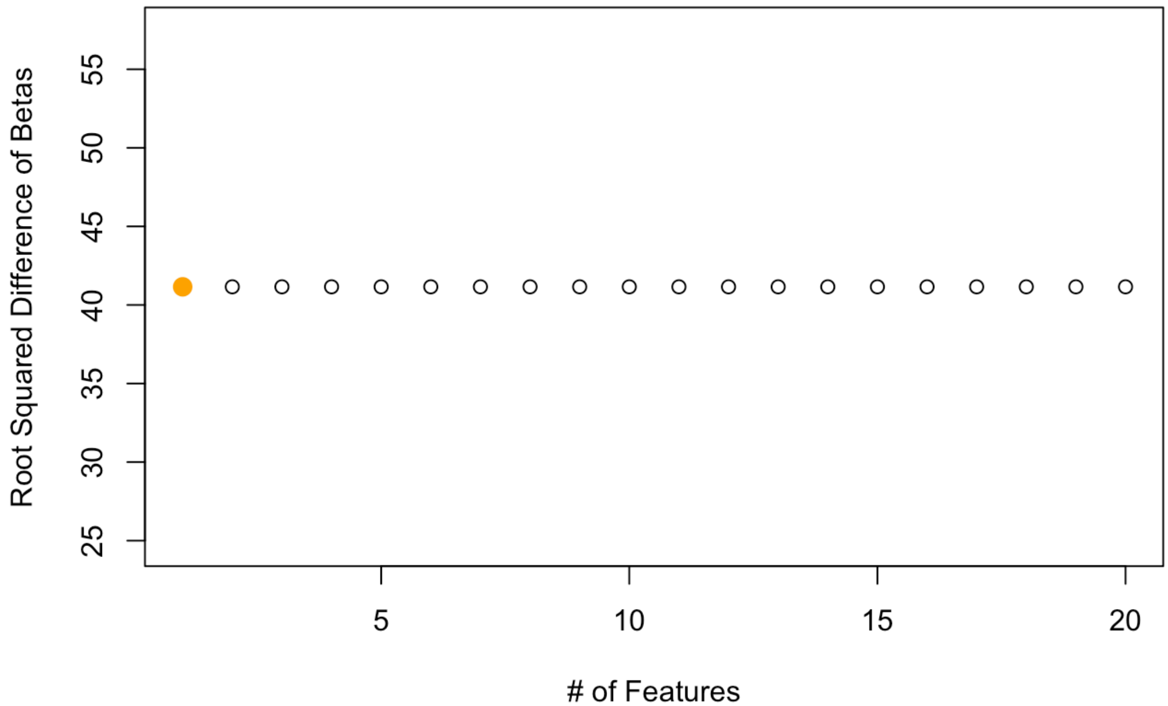
The test MSE is low until we begin to include the features whose beta values are `0`. This makes sense and matches the reality of our model.

Part G

> Create a plot displaying \[\sqrt{ \sum_{j=1}^{p} \Big(\beta_j - \hat{\beta_j^{r}} \Big)^{2} }\] for a range of values of `r`, where \(\hat{\beta_j^{r}}\) is the `j` th coefficient estimate for the best model containing `r` coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

```
rsqdiffs = sapply(1:p, function(r) {
  coefics    = coef(fit, id = r)
  coef_names = names(coefics)
  beta.est   = sapply(1:p, function(i) {
    id = sprintf('Feature #%d', i)
    if (id %in% coef_names) {
      return(coefics[id])
    } else return(0)
  })
  return(sqrt( sum((beta - beta.est)^2)) )
})

plot(seq(1:p), rsqdiffs, xlab = '# of Features', ylab = 'Root Squared Difference of Betas')
```

```
points(which.min(rsqdiffs), rsqdiffs[which.min(rsqdiffs)], col = 'orange', cex = 2, pch = p)
```



All possibilities of `k` features have approximately the same Root Squared Difference of Betas.

Problem 7

Chapter 6, Exercise 6 (p. 261)

Expression 6.12: \[\sum_j^{p} (y_j - \beta_j)^{2} + \alpha \sum_j^{p} \beta_j^{2}\] Expression 6.13: \[\sum_j^{p} (y_j - \beta_j)^{2} + \alpha \sum_j | \beta_j |\] Expression 6.14: \[\hat\beta_j ^R=\frac{y_j }{1+\alpha}\] Expression 6.15: \[\hat\beta_j ^L=

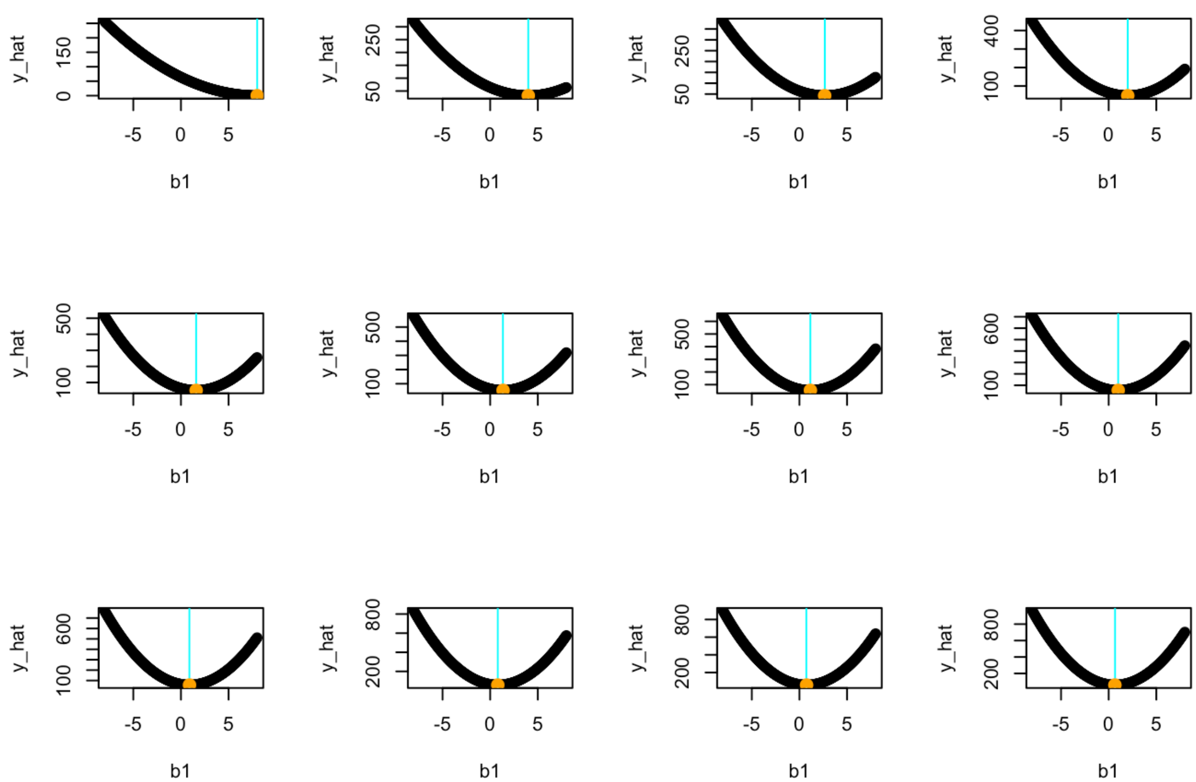
$$\{ y_j - \alpha/2 \quad \text{if } y_j > \alpha/2;\ \ y_j + \alpha/2 \quad \text{if } y_j < -\alpha/2;\ \ 0 \quad \text{if } |y_j| \leq \alpha/2.$$

\]

Part A

> Consider (6.12) with `p = 1`. For some choice of `y1` and `λ > 0`, plot (6.12) as a function of `β1`. Your plot should confirm that (6.12) is solved by (6.14).

```
par(mfrow = c(3, 4))   # Draw graphs in 3 rows, 4 columns.
for (A in seq(0, 11)) {
  y1    = 8
  b1    = seq(-8, 8, by = 0.05)
  y_hat = ((y1 - b1)^2) + (A*b1^2)

  plot(b1, y_hat)
  abline(v = y1/(1 + A), col = 'cyan', lwd = 1)
  points(b1[which.min(y_hat)], y_hat[which.min(y_hat)],  col = 'orange', cex = 2, pch = 20)
}
```



Part B

Consider (6.13) with `p = 1`. For some choice of `y1` and `λ > 0`, plot (6.13) as a function of `β1`. Your plot should confirm that (6.13) is solved by (6.15).

```
opt.y.lasso = function(y, a) {
  if (y > a/2)          return(y- a/2)
  if (y < -a/2)         return(y + a/2)
  if (abs(y) <= a/2)  return(0)
}

par(mfrow = c(3, 4))   # Draw graphs in 3 rows, 4 columns.
for (A in seq(0, 11)) {
  y1 = 8
  b1 = seq(-8, 8, by = 0.05)
  yhat = (y1 - b1)^2 + A*abs(b1)

  plot(b1, yhat)
  abline(v = opt.y.lasso(y1, A), col = 'cyan', lwd = 1)
  points(b1[which.min(yhat)], yhat[which.min(yhat)], col = 'orange', cex = 2, pch = 20)
}
```