Quiz2

Nam Jun Lee

10/11/2021

a. Simulate a data set as follows.

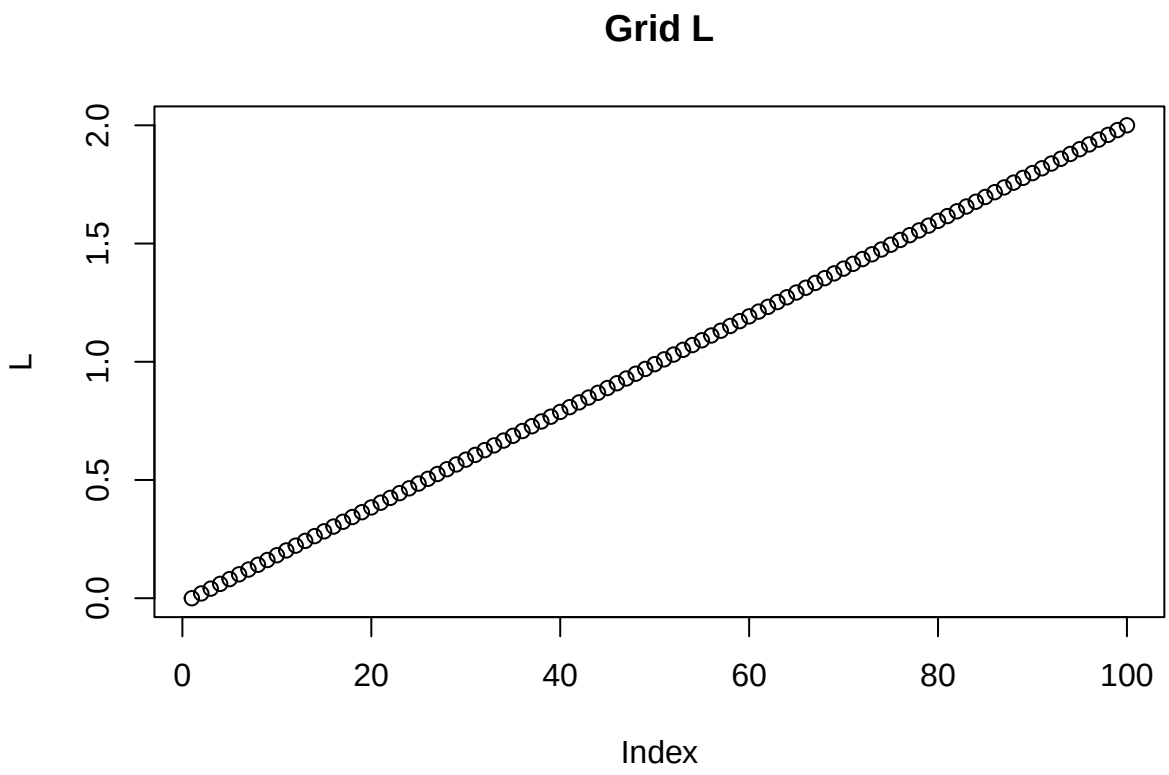
```
# dataset
set.seed(1)
n = 300
p = 200
s = 5
x = matrix(rnorm(n * p), n, p)
b = c(rep(1, s), rep(0, p - s))
y = 1 + x %*% b + rnorm(n)
# create data frame with all variables
dat = data.frame(y, x)
```

b. Define a grid L = {0,. . . ,2} of 100 numbers between 0 and 2,this grid shall serve as potential values for the regularizer λ.

```
# grid 100 numbers between 0 and 2
grid_lam = seq(0, 2, length.out = 100)
grid_lam
```

```
##   [1] 0.00000000 0.02020202 0.04040404 0.06060606 0.08080808 0.10101010
##   [7] 0.12121212 0.14141414 0.16161616 0.18181818 0.20202020 0.22222222
##  [13] 0.24242424 0.26262626 0.28282828 0.30303030 0.32323232 0.34343434
##  [19] 0.36363636 0.38383838 0.40404040 0.42424242 0.44444444 0.46464646
##  [25] 0.48484848 0.50505051 0.52525253 0.54545455 0.56565657 0.58585859
##  [31] 0.60606061 0.62626263 0.64646465 0.66666667 0.68686869 0.70707071
##  [37] 0.72727273 0.74747475 0.76767677 0.78787879 0.80808081 0.82828283
##  [43] 0.84848485 0.86868687 0.88888889 0.90909091 0.92929293 0.94949495
##  [49] 0.96969697 0.98989899 1.01010101 1.03030303 1.05050505 1.07070707
##  [55] 1.09090909 1.11111111 1.13131313 1.15151515 1.17171717 1.19191919
##  [61] 1.21212121 1.23232323 1.25252525 1.27272727 1.29292929 1.31313131
##  [67] 1.33333333 1.35353535 1.37373737 1.39393939 1.41414141 1.43434343
##  [73] 1.45454545 1.47474747 1.49494949 1.51515152 1.53535354 1.55555556
##  [79] 1.57575758 1.59595960 1.61616162 1.63636364 1.65656566 1.67676768
##  [85] 1.69696970 1.71717172 1.73737374 1.75757576 1.77777778 1.79797980
##  [91] 1.81818182 1.83838384 1.85858586 1.87878788 1.89898990 1.91919192
##  [97] 1.93939394 1.95959596 1.97979798 2.00000000
```

```r
# show grid L plot
plot(grid_lam, main = "Grid L", xlab = "Index", ylab = "L")
```



c. Use the function glmnet() to obtain lasso estimates for each value in the grid that you define in part (b). Using the function coef(), extract the estimated coefficient vector when $\lambda = L[10]$, i.e., the {10}ˆth value in the grid L.

```r
# obtain lasso estimates for each value grid_lam
las = glmnet(x, y, lambda = grid_lam, alpha = 1)
# extract the estimated coef vector 10th value
v10 = coef(las, s = 10)
# L[10] value
as.numeric(v10)
```

```
##   [1] 0.932437 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##   [9] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [17] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [25] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [33] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [41] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [49] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [57] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [65] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

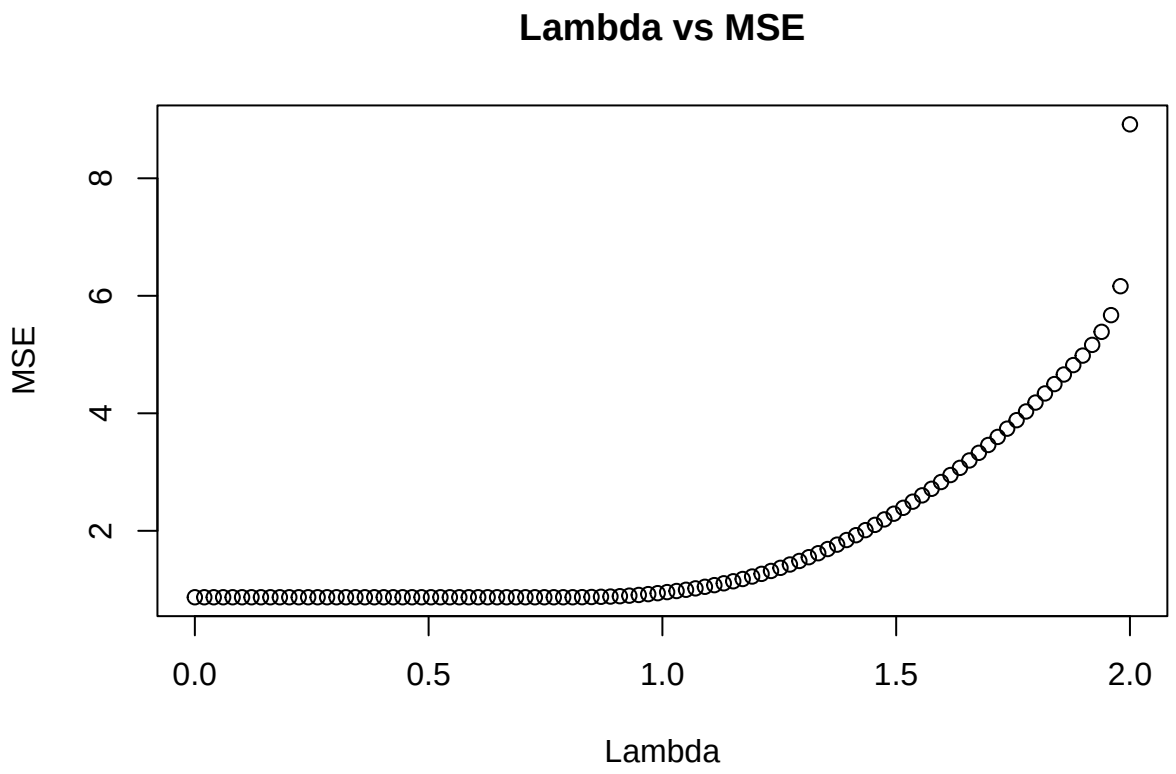
```
##  [73] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [81] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [89] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [97] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [105] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [113] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [121] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [129] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [137] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [145] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [153] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [161] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [169] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [177] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [185] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [193] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [201] 0.000000
```

d. For each value of λ in the grid L of part (b), compute the mean squared error on the entire dataset.(Use a for() loop for this purpose). This will provide a vector of 100 values of mse, one for each value of λ. Plot λ vs mse. What do you observe?

```
mse = c()
# repeat 1 to 100 times
for (i in 1:100) {
    # compute mean squared error
    mse[i] = sum(coef(las)[, i]^2)
}
# show mse each value of grid L
mse
```

```
##   [1] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387
##   [8] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387
##  [15] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387
##  [22] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387
##  [29] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8705259
##  [36] 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8694387 0.8705259
##  [43] 0.8733783 0.8769233 0.8812203 0.8868853 0.8965678 0.9084407 0.9225141
##  [50] 0.9387882 0.9567971 0.9752775 0.9964691 1.0203719 1.0469858 1.0755065
##  [57] 1.1065880 1.1413249 1.1797171 1.2217647 1.2674675 1.3168256 1.3698391
##  [64] 1.4265078 1.4868318 1.5508111 1.6184457 1.6897356 1.7646808 1.8432813
##  [71] 1.9255371 2.0114482 2.1010145 2.1942362 2.2911132 2.3916454 2.4958330
##  [78] 2.6036758 2.7151740 2.8303274 2.9491362 3.0716002 3.1977195 3.3274941
##  [85] 3.4609240 3.5980093 3.7387498 3.8831456 4.0311966 4.1829030 4.3382647
##  [92] 4.4972817 4.6604996 4.8205088 4.9822485 5.1644805 5.3866995 5.6707638
##  [99] 6.1631263 8.9179782
```

```
# plot grid L vs mse
plot(grid_lam, mse, main = "Lambda vs MSE", xlab = "Lambda", ylab = "MSE")
```



Plot λ vs MSE shows that weak flexibility.

e. Using a for() loop. compute the cross validation error, for each value of λ in the grid L, under a k= 5 fold cross validation setup. (you will need to repeatedly divide the data into testing and training, this will require another for() loop).

```
# fold cross setup
k = 5
n = 100
ncv = ceiling(n/k)
cv = rep(1:k, ncv)
cv.random = sample(cv, n, replace = F)

# cross validation
MSE = c()
cv.err = c()
for (i in 1:100) {
    for (j in 1:k) {
        # train the first model on all (first fold)
        train = dat[cv.random != j, ]
        res = train$y
        des = train[, (2:(i + 1))]
        m = lm(res ~ as.matrix(des))
        coef = coef(m)
        # MSE on test data (first fold)
```

```
        test = dat[cv.random == j, ]
        resp_val = test$y
        # recall predicted values
        fit_val = as.matrix(cbind(1, test[, (2:(i + 1))])) %*% coef
        MSE[j] = mean((resp_val - fit_val)^2)
    }
    # calculate cross validation error each values
    cv.err[i] = mean(MSE)
}
# show cross validation error
cross_validation_error <- cv.err
cross_validation_error
```

```
##   [1] 5.761579 4.556169 3.407527 2.067355 1.130907 1.138664 1.149990 1.151437
##   [9] 1.154695 1.154043 1.166456 1.170745 1.169661 1.173764 1.178202 1.187521
##  [17] 1.188811 1.187560 1.203946 1.219209 1.217513 1.228319 1.204672 1.221968
##  [25] 1.242003 1.258150 1.254784 1.256242 1.259569 1.237670 1.227470 1.239700
##  [33] 1.274572 1.284145 1.284968 1.290873 1.291702 1.299395 1.294282 1.296309
##  [41] 1.316164 1.330284 1.335775 1.346526 1.343921 1.354922 1.357633 1.397325
##  [49] 1.405699 1.415750 1.428363 1.462911 1.468085 1.488631 1.495712 1.497339
##  [57] 1.536502 1.523809 1.533324 1.541088 1.540115 1.550038 1.554351 1.591035
##  [65] 1.581911 1.593490 1.597835 1.610526 1.631222 1.654840 1.669043 1.694565
##  [73] 1.711046 1.731866 1.700375 1.731849 1.734710 1.756263 1.764741 1.790261
##  [81] 1.794746 1.814925 1.828463 1.900423 1.923851 1.932986 1.935818 1.949284
##  [89] 1.970640 1.977411 1.983457 2.014449 2.037596 2.054723 2.081920 2.107070
##  [97] 2.112863 2.121881 2.145431 2.150175
```

f. Compile all your code into a custom function with input arguments x, y, k and a grid L. This function should output the following results.

```
# custom
set.seed(1)
n = 100
p = 10
x = rnorm(n)
y = 1 + x + x^2 + x^3 + x^4 + rnorm(n)
# create a matrix with all predictor variable
z = matrix(0, n, p)
for (j in 1:p) {
    z[, j] = x^j
}
# create data frame with all variables
df = data.frame(y, z)
k = 10
# grid L
grid = seq(1, 5, length.out = 100)
```

(i) the best fit model with a k-fold cross validation.

```
# specify the cross-validation method
cvm <- trainControl(method = "cv", number = k)

# fit a regression model and use k-fold CV to evaluate performance
model <- train(y ~ ., data = df, method = "lm", trControl = cvm)

# view summary of k-fold CV
summary(model)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9774 -0.5895 -0.1238  0.4923  2.1505
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.17283    0.19971   5.873 7.28e-08 ***
## X1           1.51409    0.59009   2.566    0.012 *
## X2          -0.13146    1.29174  -0.102    0.919
## X3           0.06886    1.68567   0.041    0.968
## X4           2.90383    2.14977   1.351    0.180
## X5           0.55110    1.35654   0.406    0.686
## X6          -1.26499    1.31956  -0.959    0.340
## X7          -0.15569    0.39731  -0.392    0.696
## X8           0.31987    0.32511   0.984    0.328
## X9           0.01628    0.03817   0.426    0.671
## X10         -0.02690    0.02749  -0.979    0.330
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9719 on 89 degrees of freedom
## Multiple R-squared:  0.9861, Adjusted R-squared:  0.9846
## F-statistic: 633.7 on 10 and 89 DF,  p-value: < 2.2e-16
```

```
# Compare best fit model in k-fold cross validation
best_fit = model$resample
best_fit
```

```
##         RMSE  Rsquared       MAE Resample
## 1  1.1092218 0.9485364 0.8634573   Fold01
## 2  0.9306311 0.9168095 0.7635943   Fold02
## 3  0.8221909 0.9619048 0.7195076   Fold03
## 4  0.9821345 0.8051279 0.7556794   Fold04
## 5  1.0252242 0.9981253 0.8704718   Fold05
## 6  1.0542874 0.9928667 0.8511021   Fold06
## 7  1.0246995 0.9237335 0.8359904   Fold07
## 8  0.8202275 0.9553416 0.7051319   Fold08
```

```
## 9  1.0103498 0.8809333 0.8358679   Fold09
## 10 0.8345149 0.9950008 0.7091069   Fold10
```

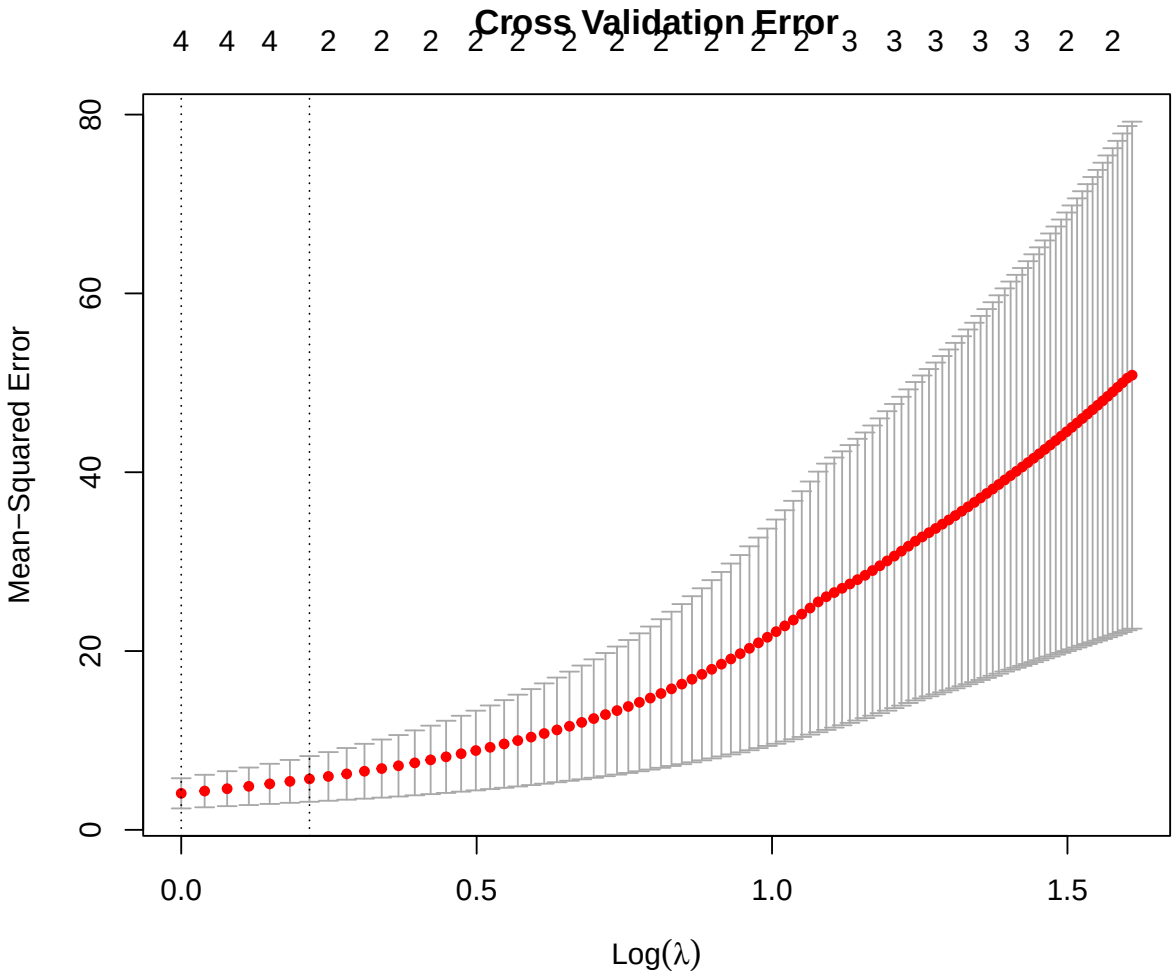
The smaller the RMSE prediction error value, the more accurate the model is. According to summary of model(1 to 10). Lowest RMSE is 0.8202275 in Fold06. Therefore, **sixth** is best fit model.

(ii) the vector of cross validation errors (one for each value of λ). (iii) the grid L used for cross validation.

```r
# obtain lasso estimates for each value grid
l.mod = glmnet(z, y, lambda = grid, alpha = 1)
mse = c()
# repeat 1 to 100 times
for (i in 1:100) {
    # compute mean squared error
    mse[i] = sum(coef(l.mod)[, i]^2)
}
# show mse each value of grid L
mse
```

```
##   [1] 11.052758 10.953171 10.856138 10.761351 10.668546 10.577500 10.493633
##   [8] 10.411803 10.330238 10.247737 10.167199 10.086884 10.006845  9.925918
##  [15]  9.846905  9.768122  9.689619  9.610272  9.535856  9.456775  9.377715
##  [22]  9.297781  9.217302  9.136540  9.055712  8.970504  8.887521  8.805377
##  [29]  8.723108  8.643048  8.562884  8.484858  8.407688  8.330492  8.255403
##  [36]  8.180309  8.107254  8.035066  7.962950  7.892836  7.822817  7.754733
##  [43]  7.687529  7.620495  7.555357  7.490413  7.427301  7.365082  7.302812
##  [50]  7.242473  7.186450  7.131206  7.076741  7.023055  6.970148  6.918019
##  [57]  6.866669  6.816099  6.766307  6.717293  6.669059  6.621604  6.574927
##  [64]  6.529029  6.483910  6.439570  6.396009  6.353226  6.311223  6.269998
##  [71]  6.229552  6.189885  6.150996  6.112887  6.075556  6.039005  6.003232
##  [78]  5.968238  5.934022  5.900586  5.867928  5.836050  5.804950  5.774629
##  [85]  5.745086  5.716323  5.688338  5.661133  5.634707  5.609060  5.584191
##  [92]  5.560102  5.536792  5.513957  5.467023  5.418649  5.342979  5.267575
##  [99]  5.195715  5.128009
```

```r
set.seed(1)
fit.las = cv.glmnet(x = z, y = y, lambda = l.mod$lambda, aplha = 1)
# plot cross validation
plot(fit.las, main = "Cross Validation Error")
```




```
# best lambda
bestlam = fit.las$lambda.min
# Predction lasso model
lasso_pred = predict(l.mod, s = bestlam, newx = z)
# MSE
mean((lasso_pred - y)^2)
```

```
## [1] 2.689154
```

(iv) the value of lambda at which the best fit model is obtained.

```
lasso_coef = predict(fit.las, type = "coefficients", s = bestlam)[1:11, ]
# Coefficient estimates
lasso_coef
```

```
## (Intercept)          V1          V2          V3          V4          V5
##  1.79238449  0.20284901  0.08085003  0.92140005  1.00930864  0.00000000
```

```
##          V6          V7          V8          V9         V10
##  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
```

However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 6 of the 10 coefficient estimates are exactly zero. So the lasso model with λ chosen by cross-validation contains only **four variables**.

g. Finally, use the function you make in Part (f) with k = 5,then extract the vector of cross validation errors (say, CVV) and the grid L that is used. Make a plot of L vs. CVV.

```
k = 5
set.seed(1)
ncv = ceiling(n/k)
cv = rep(1:k, ncv)
cv.sample = sample(cv, n, replace = F)

# cross validation
MSE = c()
cvv.err = c()
for (i in 1:100) {
    for (j in 1:k) {
        # train the first model on all (first fold)
        train = dat[cv.random != j, ]
        response = train$y
        design = train[, (2:(i + 1))]
        m = lm(response ~ as.matrix(design))
        coef = coef(m)
        # MSE on test data (first fold)
        test = dat[cv.random == j, ]
        resp.values = test$y
        # recall predicted values
        fitted.values = as.matrix(cbind(1, test[, (2:(i + 1))])) %*% coef
        MSE[j] = mean((resp.values - fitted.values)^2)
    }
    # calculate cross validation error each values
    cvv.err[i] = mean(MSE)
}
# CVV Error
CVV <- cvv.err
CVV
```

```
##   [1] 5.761579 4.556169 3.407527 2.067355 1.130907 1.138664 1.149990 1.151437
##   [9] 1.154695 1.154043 1.166456 1.170745 1.169661 1.173764 1.178202 1.187521
##  [17] 1.188811 1.187560 1.203946 1.219209 1.217513 1.228319 1.204672 1.221968
##  [25] 1.242003 1.258150 1.254784 1.256242 1.259569 1.237670 1.227470 1.239700
##  [33] 1.274572 1.284145 1.284968 1.290873 1.291702 1.299395 1.294282 1.296309
##  [41] 1.316164 1.330284 1.335775 1.346526 1.343921 1.354922 1.357633 1.397325
##  [49] 1.405699 1.415750 1.428363 1.462911 1.468085 1.488631 1.495712 1.497339
##  [57] 1.536502 1.523809 1.533324 1.541088 1.540115 1.550038 1.554351 1.591035
##  [65] 1.581911 1.593490 1.597835 1.610526 1.631222 1.654840 1.669043 1.694565
```

```
##  [73] 1.711046 1.731866 1.700375 1.731849 1.734710 1.756263 1.764741 1.790261
##  [81] 1.794746 1.814925 1.828463 1.900423 1.923851 1.932986 1.935818 1.949284
##  [89] 1.970640 1.977411 1.983457 2.014449 2.037596 2.054723 2.081920 2.107070
##  [97] 2.112863 2.121881 2.145431 2.150175
```

```r
# Show grid L vs CVV
plot(grid, CVV, main = "Plot of L vs. CVV", xlab = "Grid L", ylab = "CVV")
```