Stats 102B HW 2

Charles Liu (304804942)

4/19/2020

Homework 2 Requirements

There is no separate instruction file. This file is the instructions and you will modify it for your submission.

You will submit two files.

The files you submit will be:

1. `102b_hw2_output_First_Last.Rmd` Take this R Markdown file and make the necessary edits so that it generates the requested output.
2. `102b_hw2_output_First_Last.pdf` OR `102b_hw_02_output_First_Last.html` Your output file. This can be a PDF or an HTML file. This is the primary file that will be graded. **Make sure all requested output is visible in the output file.**

Failure to submit all files will result in an automatic 40 point penalty.

Academic Integrity

Modifying the following statement with your name.

"By including this statement, I, Charles Liu, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students."

If you collaborated verbally with other students, please also include the following line to credit them.

"I did discuss ideas related to the homework with Shirley Mach for Tasks 4 and 5, regarding the comments. We merely discussed how each method would affect the outcome. I used a good portion of what Professor Chen was talking about in his lecture videos for my homework comments. At no point did I show another student my code, nor did I look at another student's code."

Part 1: Gradient Descent for Linear Regression

In this problem you will use gradient descent to estimate the parameters for a Linear Regression model.

Start with Some artificial Data

Here is some artificial data. The variable t depends on two variables: x_1 and x_2. The true relationship between t and x_1 and x_2 is $t = 2x_1 + x_2$ but we add some random noise.

```
RNGkind(sample.kind = "Rejection")
set.seed(6)
x1 <- 5 * runif(10)
x2 <- 4 * runif(10)
t <- 2 * x1 + x2 + rnorm(10, sd = 2)
X <- cbind(1, x1, x2)
```

`lm()` can find the OLS coefficient estimates.

```
lm(t ~ x1 + x2)$coefficients
```

```
## (Intercept)          x1          x2
##    2.388634    1.342180    1.118182
```

You will now try to arrive at the same coefficient estimates by applying gradient descent to the Ordinary Least Squares Loss Function.

The Loss function for Ordinary Least Squares Regression is:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t}^T\mathbf{t} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{t} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

The gradient of $\mathcal{L}$ with respect to $\mathbf{w}$ is

$$\nabla\mathcal{L} = \frac{-2}{N}\mathbf{X}^T\mathbf{t} + \frac{2}{N}\mathbf{X}^T\mathbf{X}\mathbf{w}$$

Task 1:

Refer to slides 3 through 28 in Lecture 2-2.

Refering to the gradient in matrix notation, write a function to calculate the value of the gradient at a particular value of $\mathbf{w}$.

The function will accept a single vector `w = c(intercept, w1, w2)`. You can use values of `X` and `t` or any other variables that are available in the global environment.

```
loss_grad <- function(w){
  N <- length(t)
  loss_deriv <- ((1/N) * (-2 * (t(X) %*% t) + (2 * (t(X) %*% X %*% w))))
  return(loss_deriv)
  }
```

Use the function you just wrote to find the gradient of the OLS loss function for a model that has $w_0 = w_1 = w_2 = 0$

```
# No need to modify. This chunk is used to check your results for grading.
loss_grad(c(0, 0, 0))
```

```
##          [,1]
##    -17.68109
```

```
## x1 -62.57804
## x2 -39.42205
```

Use the function to find the gradient of the OLS loss function for a model that has the OLS coefficient estimates. The gradient should be very close to 0 indicating that it is indeed a local minimum.

```
# No need to modify. This chunk is used to check your results for grading.
best_w <- lm(t ~ x1 + x2)$coefficients
print(best_w)
```

```
## (Intercept)          x1          x2
##    2.388634    1.342180    1.118182
```

```
loss_grad(best_w) # should be a vector of ~0 values.
```

```
##             [,1]
##    2.842171e-15
## x1 1.136868e-14
## x2 0.000000e+00
```

Part 1 B: Numeric Gradient Checking

One way for to check if the gradient function is correctly calculated and coded is to perform numeric gradient checking.

Numeric gradient checking is the idea of adding a small perturbation to each element in the vector or matrix $\mathbf{w}$ one element at a time, and to measure the effect it has on the loss.

See slides 41 to 43 of Lecture 2-2.

Task 2:

Perform numeric gradient checking to check the gradient at the location $\mathbf{w} = (0, 0, 0)$.

Complete the code in the following loop to find the numeric gradient estimate for w at point `c(0, 0, 0)`.

```
w <- c(0, 0, 0)
epsilon <- 1e-4
numeric_gradient <- matrix(0, nrow = 3)

# create our loss function
loss <- function(w) {
  N <- length(t)
  loss_func <- (1/N) * ((t(t) %*% t) - (2 * t(w) %*% t(X) %*% t) +
    (t(w) %*% t(X) %*% X %*% w))
  return(loss_func)
}



for(i in seq_along(w)){
  w_epsilon <- w
  w_epsilon[i] <- w_epsilon[i] + epsilon
  numeric_gradient[i, 1] <- (loss(w_epsilon) - loss(w))/(epsilon)
}
numeric_gradient
```

```
##           [,1]
## [1,] -17.68099
```

```
## [2,] -62.57683
## [3,] -39.42153
```

Print the results of your numeric gradient estimate and print the result you calculated with your gradient function. Compare the two. What is the element-wise percentage difference?

```
num_estim <- numeric_gradient
num_func <- loss_grad(c(0, 0, 0))

# Values of gradient estimate and gradient function:
num_estim
```

```
##           [,1]
## [1,] -17.68099
## [2,] -62.57683
## [3,] -39.42153
```

```
num_func
```

```
##         [,1]
##    -17.68109
## x1 -62.57804
## x2 -39.42205
```

```
# Percentage difference element-wise:
((num_func - num_estim)/num_estim) *100 # No need for function (just simple calculations)
```

```
##            [,1]
##    0.0005655791
## x1 0.0019308442
## x2 0.0013181737
```

Part 1 C: Gradient Descent

Task 3:

Write a loop to perform gradient descent. Start at the point `w = c(0,0,0)` and iterate to estimate the coefficients to fit the linear model.

Using trial and error, play around with your choice of gamma (hint: start small) and how many iterations to run (probably between a few thousand to a hundred thousand iterations).

Print out your choice of gamma (no right answer) and the resulting coefficient estimates at the end of the gradient descent algorithm (should be within 0.001 of the estimates made by lm).

```
gamma <- 0.001
total_iter <- 10e4 # 10,000 iterations
w <- c(0, 0, 0)

for(i in 1:total_iter) {
  w <- w - gamma * loss_grad(w)
}

best_w <- lm(t ~ x1 + x2)$coefficients

best_w
```

```
## (Intercept)          x1          x2
##    2.388634    1.342180    1.118182
```

```
w
```

```
##        [,1]
##    2.388634
## x1 1.342180
## x2 1.118182
```

```
# We see little to no difference between best_w and w, and it is within 0.001 of the estimates made by
```

Part 2: The Effects of Regularization on the variation of parameter estimates

In this section, we will look at the effects of regularization on parameter estimates, especially in relation to the Bias-Variance Tradeoff. This section follows the work in Lecture 3-1 closely, so refer to those notes.

From Wikipedia:

The Gauss-Markov theorem states that in a linear regression model in which the errors have expectation zero, are uncorrelated and have equal variances, the best linear unbiased estimator (BLUE) of the coefficients is given by the ordinary least squares (OLS) estimator. Here "best" means giving the lowest variance of the estimate, as compared to other unbiased, linear estimators.

https://en.wikipedia.org/wiki/Gauss%E2%80%93Markov_theorem

In that regard, OLS provides estimates of coefficients with nice properties. These coefficient estimates, however, often have relatively high variance if there is high correlation between the predictors.

To illustrate this, I will generate some random data and use OLS to fit models to them.

For the generation of data, I will use library mvtnorm which is used to generate values from a multivariate-normal distribution.

```
library(mvtnorm)
```

Study and understand the code below.

The matrix `sigma` is a variance-covariance matrix. I use `sigma` as an argument in the function `rmvnorm()` when generating random multivariate normal data.

I generate matrices `x1x2` (representing variables x1 and x2) and `x3x4` (variables x3 and x4) and use `cbind` to create a matrix X.

```
set.seed(4)
sigma <- matrix(c(1,.9,.9,1), nrow = 2)
x1x2 <- rmvnorm(30, sigma = sigma)
x3x4 <- rmvnorm(30, sigma = 2 * sigma)  # x3 and x4 has larger variance than x1 and x2
X <- cbind(x1x2,x3x4) # creation of matrix X
round(cor(X), 3) # rounded for easier reading
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,]  1.000  0.924 -0.059 -0.104
## [2,]  0.924  1.000  0.013 -0.055
## [3,] -0.059  0.013  1.000  0.905
## [4,] -0.104 -0.055  0.905  1.000
```

When we look at the correlation matrix of X, we see that variables x1 and x2 have high correlation (around 0.9) with each other, and low correlation with variables x3 and x4. Meanwhile x3 and x4 have high correlation with each other and low correlation with variables x1 and x2. This makes sense as we generated matrix `x1x2` independently from matrix `x3x4`.

```
error = rnorm(30, sd = 1)
true_y <- 1.2 * X[,1] - 1.5 * X[,3] # creation of true_y
y <- true_y + error
```

I now generate values of y. We can see that the true values of y `true_y` depend only on variables x1 and x3 but we have also added some random noise.

```
model <- lm(y ~ X)
coefs <- model$coefficients
coefs
```

```
## (Intercept)          X1          X2          X3          X4
##  -0.2382451   2.0567622  -0.6050709  -1.0880234  -0.1952981
```

Based on the 'true_y' we know that the true coefficients should be:

- intercept: 0
- x1: 1.2
- x2: 0
- x3: -1.5
- x4: 0

However because of the random noise added to our data, we get something in the ballpark but different.

I've written some code to explore how the estimates of the coefficients vary with randomness.

I run a couple thousand iterations of a loop. In each iteration, I generate new random noise to add to the "true values" to produce new values of y. We fit a linear model between our new noisy y values and the X matrix. I store the model coefficients into matrix `coef` to keep track of these results.

```
reps <- 2000
coefs <- matrix(NA, nrow = reps, ncol = 5)  # we will store the model coefficients here

for(i in seq_len(reps)){
  set.seed(i)
  error = rnorm(30, sd = 1)
  y <- true_y + error
  model <- lm(y ~ X)
  coefs[i,] <- model$coefficients
}
```

Now that the loop has finished running, we can see the mean and variation of our coefficient estimates.

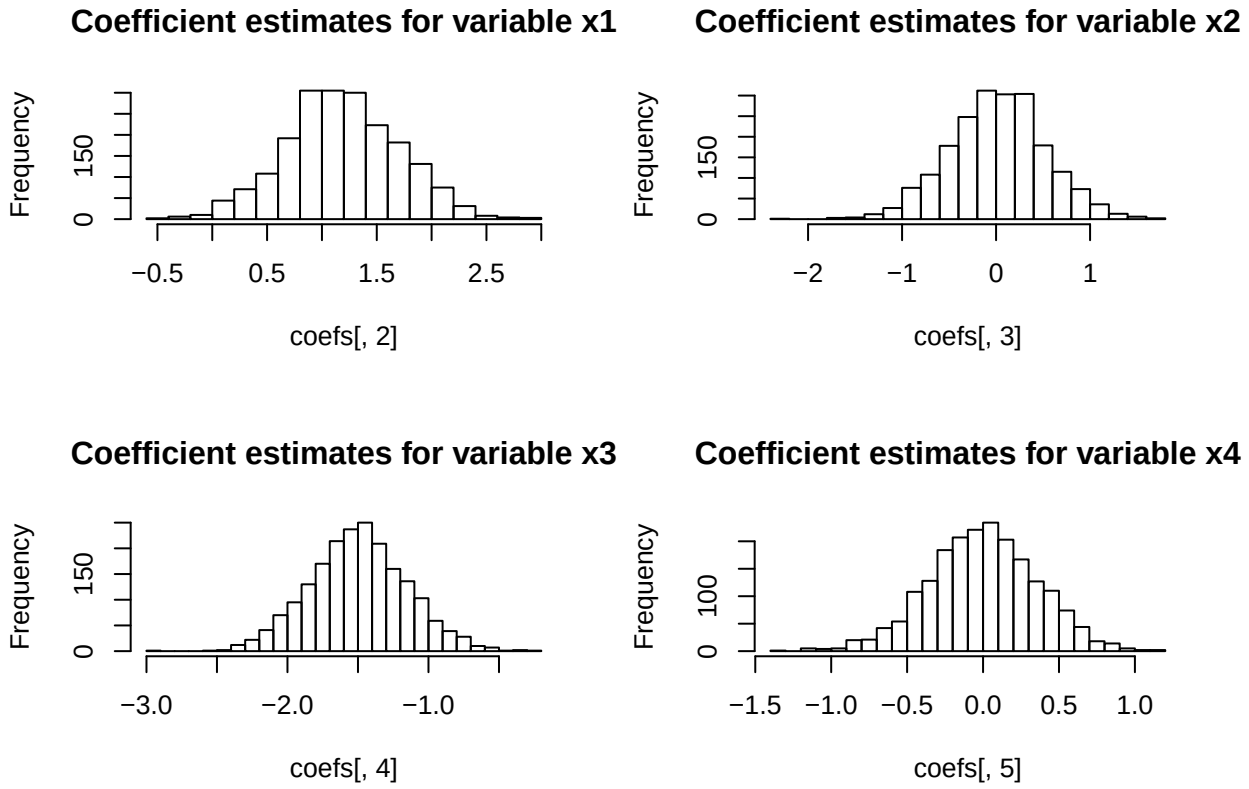
```
means <- round(colMeans(coefs), 4)
medians <- round(apply(coefs, 2, FUN = median), 4)
variances <- round(apply(coefs, 2, FUN = var), 4)
rbind(means, medians, variances)
```

```
##             [,1]   [,2]   [,3]    [,4]    [,5]
## means     0.0007 1.1856 0.0133 -1.4968 -0.0027
## medians   0.0031 1.1675 0.0240 -1.4973  0.0000
## variances 0.0415 0.2673 0.2660  0.1181  0.1275
```

We can see that the parameter estimates indeed are unbiased. The mean values of the parameter estimates indeed align closely with what we know to be the true parameter values (intercept: 0, x1: 1.2, x2: 0, x3: -1.5, x4: 0).

We also see the medians and the variances of these coefficient estimates.

```
par(mfrow = c(2,2))
hist(coefs[,2], breaks = 25, main = "Coefficient estimates for variable x1")
hist(coefs[,3], breaks = 25, main = "Coefficient estimates for variable x2")
hist(coefs[,4], breaks = 25, main = "Coefficient estimates for variable x3")
hist(coefs[,5], breaks = 25, main = "Coefficient estimates for variable x4")
```



The above plots confirm the same: the coefficients are unbiased in that their distributions are centered around the true coefficient values. We also get a sense of the variation that exists.

Getting Parameter estimates for LASSO and Ridge regression models

We can do something similar for LASSO and Ridge regression models. Instead of using `lm()` to fit an OLS linear regression model, we use `glmnet()`.

A LASSO fit is run by default, and ridge regression is used when we set the paramter `alpha = 0`. (Elasticnet is used for values of alpha between 0 and 1.)

The coefficients of glmnet can be extracted with the function `coef()`, along with a specified `s`. The argument `s` corresponds to the λ that was used in the loss function. Lower values of λ correspond to loss function that have low complexity penalties, while larger values of λ correspond to loss functions with larger complexity penalties.

When run on a glmnet fit, `coef()` by default returns a sparse column matrix. So I call `t()` to transpose it, and `as.matrix()` to fill in the blanks with zeros.

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.6.3
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

```
set.seed(1)
error = rnorm(30, sd = 1)
y <- true_y + error
lassofit = glmnet(X, y)
# we can see lasso will push a paramter value down to 0
as.matrix(t(coef(lassofit, s = 0.5)))
```

```
##   (Intercept)        V1 V2         V3         V4
## 1   0.3274755 0.5541817  0 -0.8973309 -0.1439535
```

```
ridgefit = glmnet(X, y, alpha = 0)
as.matrix(t(coef(ridgefit, s = 0.5)))
```

```
##   (Intercept)        V1        V2        V3         V4
## 1   0.1622714 0.5995137 0.3382422 -0.784621 -0.5233592
```

Task 4: Simulate random data, fit LASSO and Ridge, track the coefficients

Use package glmnet to fit LASSO and ridge regression models between y and X.

Follow my code example where I fit 2000 linear models to the data where different random noise was applied to the "true" values of y.

For each iteration, fit LASSO and Ridge regression. Extract the coefficient estimates for $\lambda = 0.1$, $\lambda = 0.3$, and $\lambda = 0.5$ for both LASSO and Ridge regression models. Keep track of all the coefficient estimates in matrices (there are 6 matrices to make).

```
reps <- 2000

# I've already made the matrices where you will store your coefficients
coefsLasso1 <- matrix(NA, nrow = reps, ncol = 5)
coefsLasso3 <- matrix(NA, nrow = reps, ncol = 5)
coefsLasso5 <- matrix(NA, nrow = reps, ncol = 5)
coefsRidge1 <- matrix(NA, nrow = reps, ncol = 5)
coefsRidge3 <- matrix(NA, nrow = reps, ncol = 5)
coefsRidge5 <- matrix(NA, nrow = reps, ncol = 5)

for(i in 1:reps){
  set.seed(i)
  error = rnorm(30, sd = 1)
  y <- true_y + error
  coefsLasso1[i,] <- as.matrix(t(coef(glmnet(X, y), s = 0.1)))
  coefsLasso3[i,] <- as.matrix(t(coef(glmnet(X, y), s = 0.3)))
  coefsLasso5[i,] <- as.matrix(t(coef(glmnet(X, y), s = 0.5)))
  coefsRidge1[i,] <- as.matrix(t(coef(glmnet(X, y, alpha = 0), s = 0.1)))
  coefsRidge3[i,] <- as.matrix(t(coef(glmnet(X, y, alpha = 0), s = 0.3)))
  coefsRidge5[i,] <- as.matrix(t(coef(glmnet(X, y, alpha = 0), s = 0.5)))
}
```

Once the loop finishes running, create summary tables showing the mean, median, and variance of the coefficient estimates.

```
means_LASSO_0.1 <- round(colMeans(coefsLasso1), 4)
medians_LASSO_0.1 <- round(apply(coefsLasso1, 2, FUN = median), 4)
```

```
variances_LASSO_0.1 <- round(apply(coefsLasso1, 2, FUN = var), 4)
rbind(means_LASSO_0.1, medians_LASSO_0.1, variances_LASSO_0.1)
```

```
##                       [,1]   [,2]   [,3]    [,4]    [,5]
## means_LASSO_0.1     0.0388 0.9560 0.1448 -1.3262 -0.1245
## medians_LASSO_0.1   0.0442 0.9964 0.0000 -1.3608  0.0000
## variances_LASSO_0.1 0.0389 0.0921 0.0635  0.0493  0.0400
```

```
means_LASSO_0.3 <- round(colMeans(coefsLasso3), 4)
medians_LASSO_0.3 <- round(apply(coefsLasso3, 2, FUN = median), 4)
variances_LASSO_0.3 <- round(apply(coefsLasso3, 2, FUN = var), 4)
rbind(means_LASSO_0.3, medians_LASSO_0.3, variances_LASSO_0.3)
```

```
##                       [,1]   [,2]   [,3]    [,4]    [,5]
## means_LASSO_0.3     0.1168 0.8202 0.0762 -1.1916 -0.1084
## medians_LASSO_0.3   0.1222 0.8425 0.0000 -1.2247  0.0000
## variances_LASSO_0.3 0.0386 0.0657 0.0330  0.0451  0.0348
```

```
means_LASSO_0.5 <- round(colMeans(coefsLasso5), 4)
medians_LASSO_0.5 <- round(apply(coefsLasso5, 2, FUN = median), 4)
variances_LASSO_0.5 <- round(apply(coefsLasso5, 2, FUN = var), 4)
rbind(means_LASSO_0.5, medians_LASSO_0.5, variances_LASSO_0.5)
```

```
##                       [,1]   [,2]   [,3]    [,4]    [,5]
## means_LASSO_0.5     0.1971 0.6569 0.0372 -1.0584 -0.0922
## medians_LASSO_0.5   0.2023 0.6693 0.0000 -1.0884  0.0000
## variances_LASSO_0.5 0.0383 0.0494 0.0146  0.0414  0.0300
```

Comment on the effect of lambda on the distributions of the coefficient estimates of Lasso.

As we increase our λ value, we can see our loss functions have a larger amount of complexity penalties. Many of our coefficients under LASSO are exactly zeroed for an increase in λ. We can see for an increasing λ, our means and medians approach closer to zero (but not zero), and the variances change slightly.

```
means_Ridge_0.1 <- round(colMeans(coefsRidge1), 4)
medians_Ridge_0.1 <- round(apply(coefsRidge1, 2, FUN = median), 4)
variances_Ridge_0.1 <- round(apply(coefsRidge1, 2, FUN = var), 4)
rbind(means_Ridge_0.1, medians_Ridge_0.1, variances_Ridge_0.1)
```

```
##                       [,1]   [,2]   [,3]    [,4]    [,5]
## means_Ridge_0.1     0.0114 0.8975 0.2307 -1.1330 -0.3355
## medians_Ridge_0.1   0.0141 0.8861 0.2380 -1.1304 -0.3349
## variances_Ridge_0.1 0.0376 0.0689 0.0650  0.0349  0.0398
```

```
means_Ridge_0.3 <- round(colMeans(coefsRidge3), 4)
medians_Ridge_0.3 <- round(apply(coefsRidge3, 2, FUN = median), 4)
variances_Ridge_0.3 <- round(apply(coefsRidge3, 2, FUN = var), 4)
rbind(means_Ridge_0.3, medians_Ridge_0.3, variances_Ridge_0.3)
```

```
##                       [,1]   [,2]   [,3]    [,4]    [,5]
## means_Ridge_0.3     0.0187 0.8292 0.2733 -1.0471 -0.4000
## medians_Ridge_0.3   0.0215 0.8179 0.2732 -1.0446 -0.3986
## variances_Ridge_0.3 0.0372 0.0478 0.0441  0.0282  0.0273
```

```
means_Ridge_0.5 <- round(colMeans(coefsRidge5), 4)
medians_Ridge_0.5 <- round(apply(coefsRidge5, 2, FUN = median), 4)
variances_Ridge_0.5 <- round(apply(coefsRidge5, 2, FUN = var), 4)
rbind(means_Ridge_0.5, medians_Ridge_0.5, variances_Ridge_0.5)
```

```
##                          [,1]   [,2]   [,3]    [,4]    [,5]
## means_Ridge_0.5     0.0341 0.7395 0.3184 -0.9310 -0.4698
## medians_Ridge_0.5   0.0397 0.7301 0.3176 -0.9276 -0.4692
## variances_Ridge_0.5 0.0367 0.0286 0.0256  0.0177  0.0162
```

Comment on the effect of lambda on the distributions of the coefficient estimates of Ridge Regression.

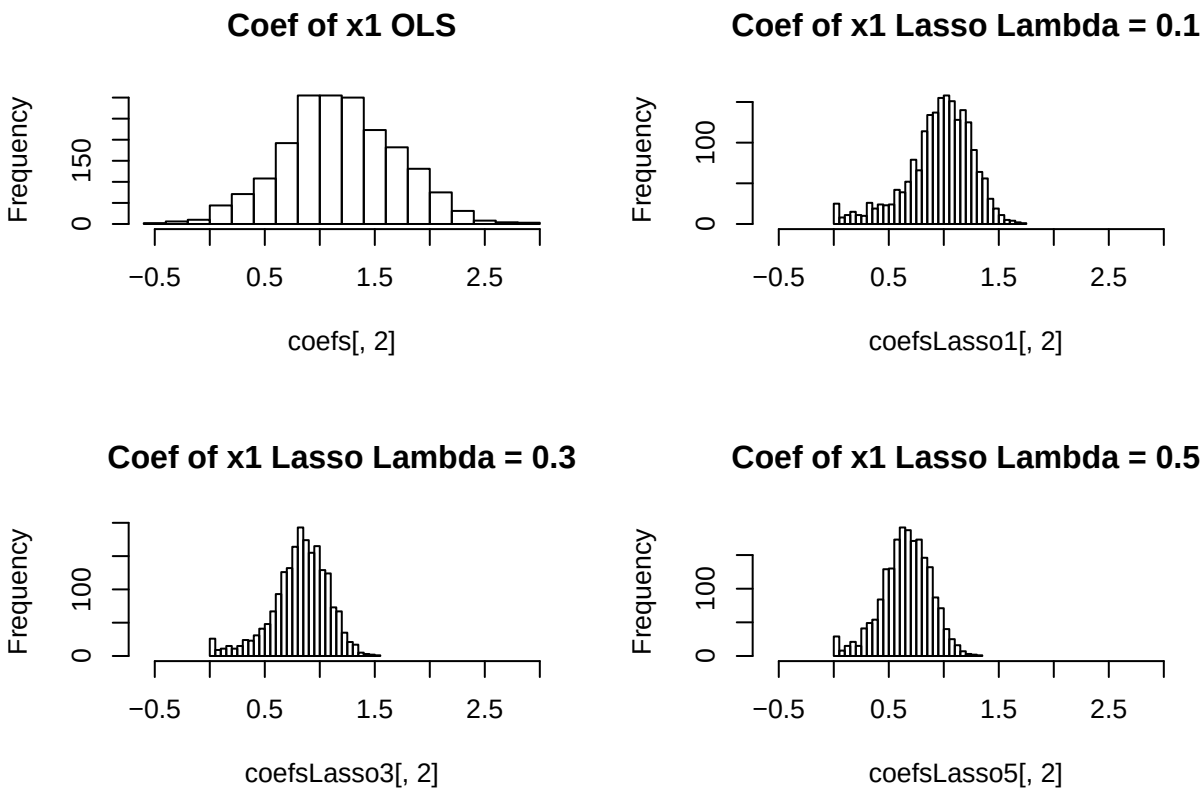
Many of our coefficients under Ridge Regression are penalized if they are too far from zero, thus making them to be smaller continuously. We do NOT zero our coefficients out at all under Ridge Regression. This helps decrease our complexity of our model, while keeping the variables. We can see for an increasing λ, our variances are decreasing.

Task 5: Comment on a bunch of plots

If you succesfully ran your loop and stored the values, then the following code should work and produce plots. You might need to make some minor changes to get it to work.

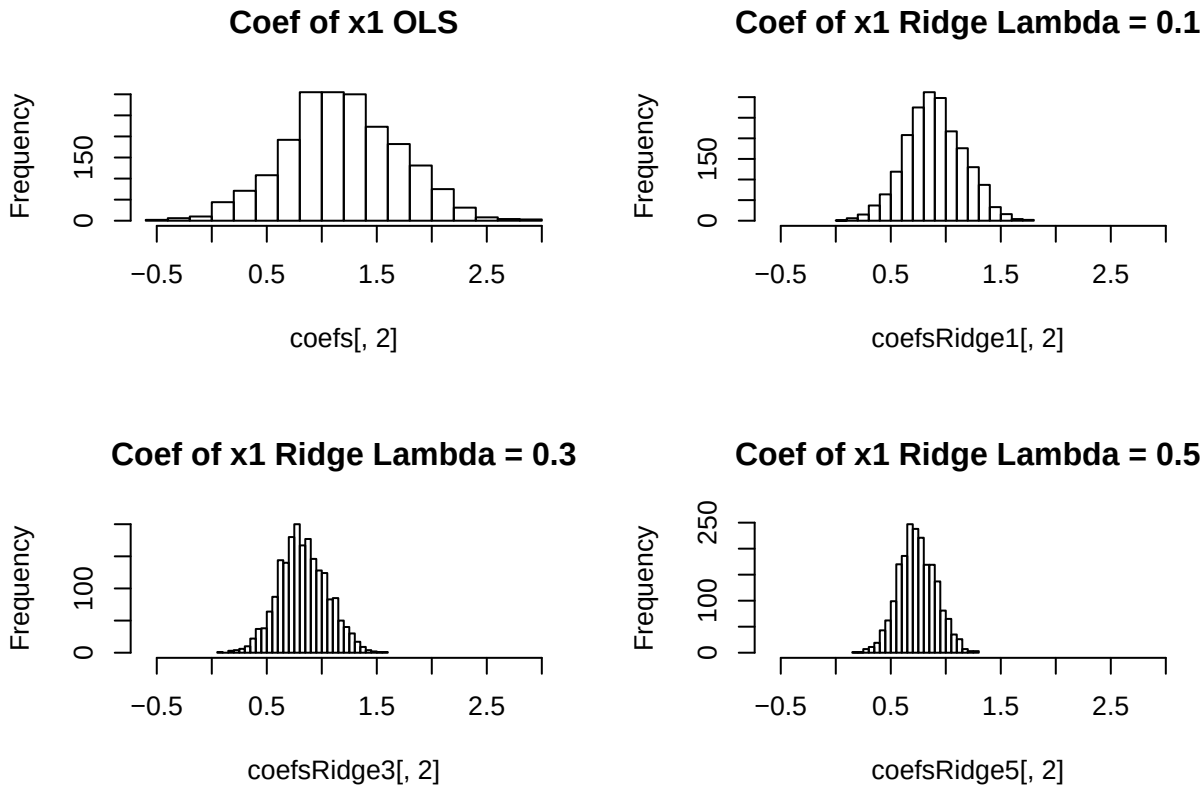
I've written the code for you. You need to explain what it means.

```
par(mfrow = c(2,2))
xlimits = c(min(coefs[,2]), max(coefs[,2]))
hist(coefs[,2], breaks = 25, main = "Coef of x1 OLS", xlim = xlimits)
hist(coefsLasso1[,2], breaks = 25, main = "Coef of x1 Lasso Lambda = 0.1", xlim = xlimits)
hist(coefsLasso3[,2], breaks = 25, main = "Coef of x1 Lasso Lambda = 0.3", xlim = xlimits)
hist(coefsLasso5[,2], breaks = 25, main = "Coef of x1 Lasso Lambda = 0.5", xlim = xlimits)
```




```
hist(coefs[,2], breaks = 25, main = "Coef of x1 OLS", xlim = xlimits)
hist(coefsRidge1[,2], breaks = 25, main = "Coef of x1 Ridge Lambda = 0.1", xlim = xlimits)
```

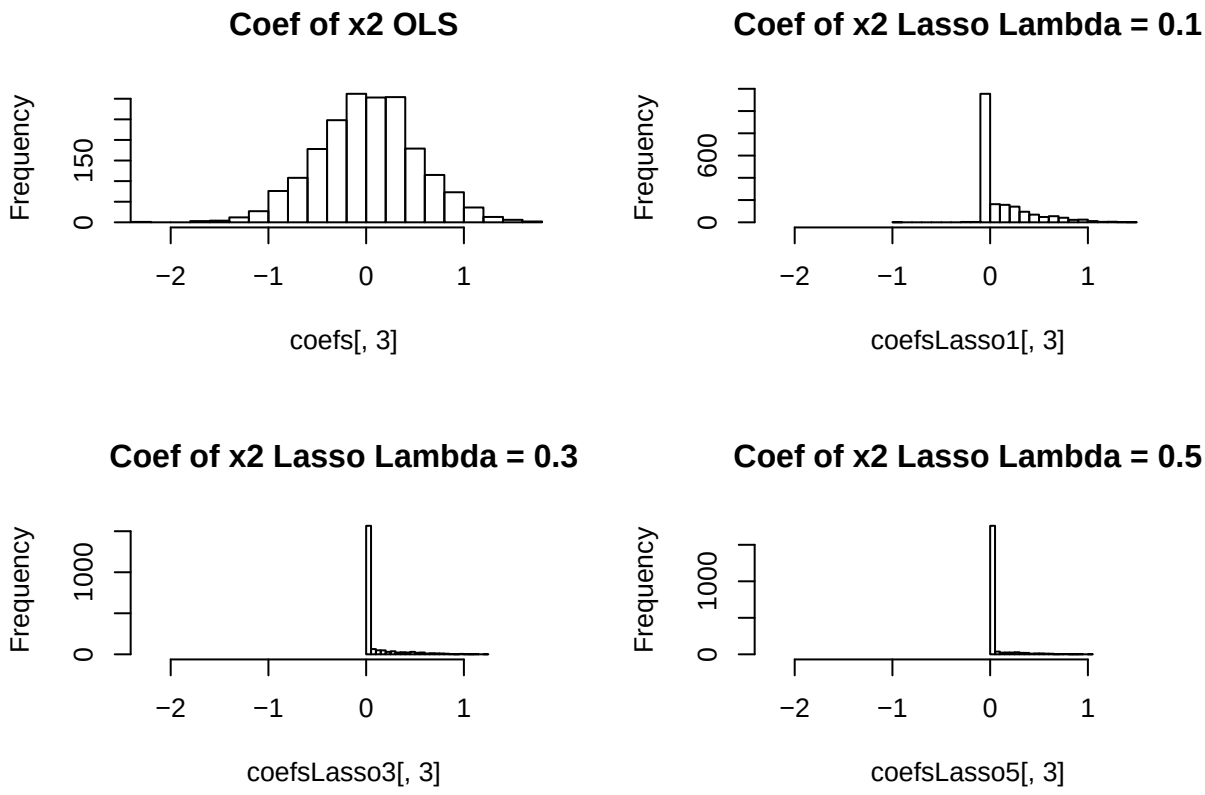
```
hist(coefsRidge3[,2], breaks = 25, main = "Coef of x1 Ridge Lambda = 0.3", xlim = xlimits)
hist(coefsRidge5[,2], breaks = 25, main = "Coef of x1 Ridge Lambda = 0.5", xlim = xlimits)
```



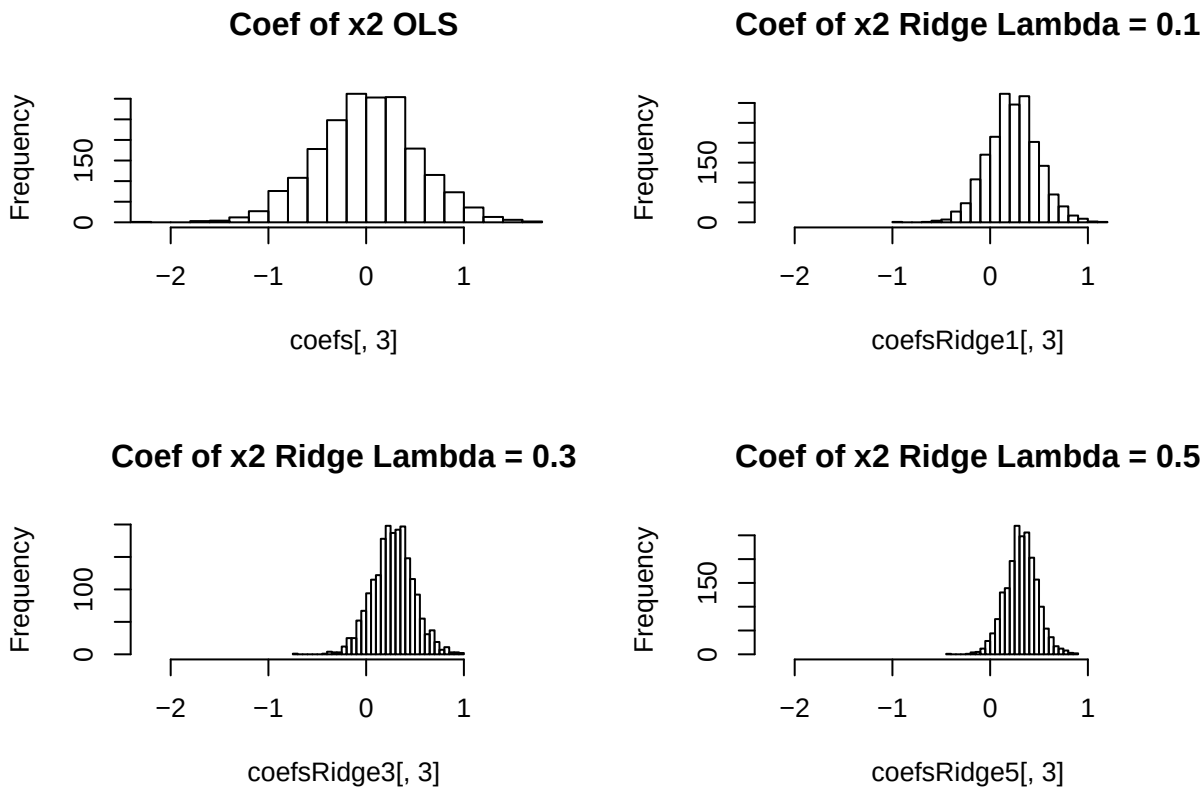
Comment on the plots, talk about the effect of lambda. How does lasso and ridge regression affect the parameter estimates you get? Keep in mind what the true value of the coefficient should be. What kind of bias do you see? What can you say about the variance of estimates?

The true coefficient value for X1 = 1.2, but we can see as we increase our λ we are getting further away from our true value. We can see there is a bit of bias in play in the Ridge Regression and LASSO, but the Variance of our estimate is decreasing as we increase λ. The bias for Ridge Regression and LASSO is around 0.75. As we decrease our Variance, we see an increase in a bias (Bias-Variance Tradeoff).

```
par(mfrow = c(2,2))
xlimits = c(min(coefs[,3]), max(coefs[,3]))
hist(coefs[,3], breaks = 25, main = "Coef of x2 OLS", xlim = xlimits)
hist(coefsLasso1[,3], breaks = 25, main = "Coef of x2 Lasso Lambda = 0.1", xlim = xlimits)
hist(coefsLasso3[,3], breaks = 25, main = "Coef of x2 Lasso Lambda = 0.3", xlim = xlimits)
hist(coefsLasso5[,3], breaks = 25, main = "Coef of x2 Lasso Lambda = 0.5", xlim = xlimits)
```



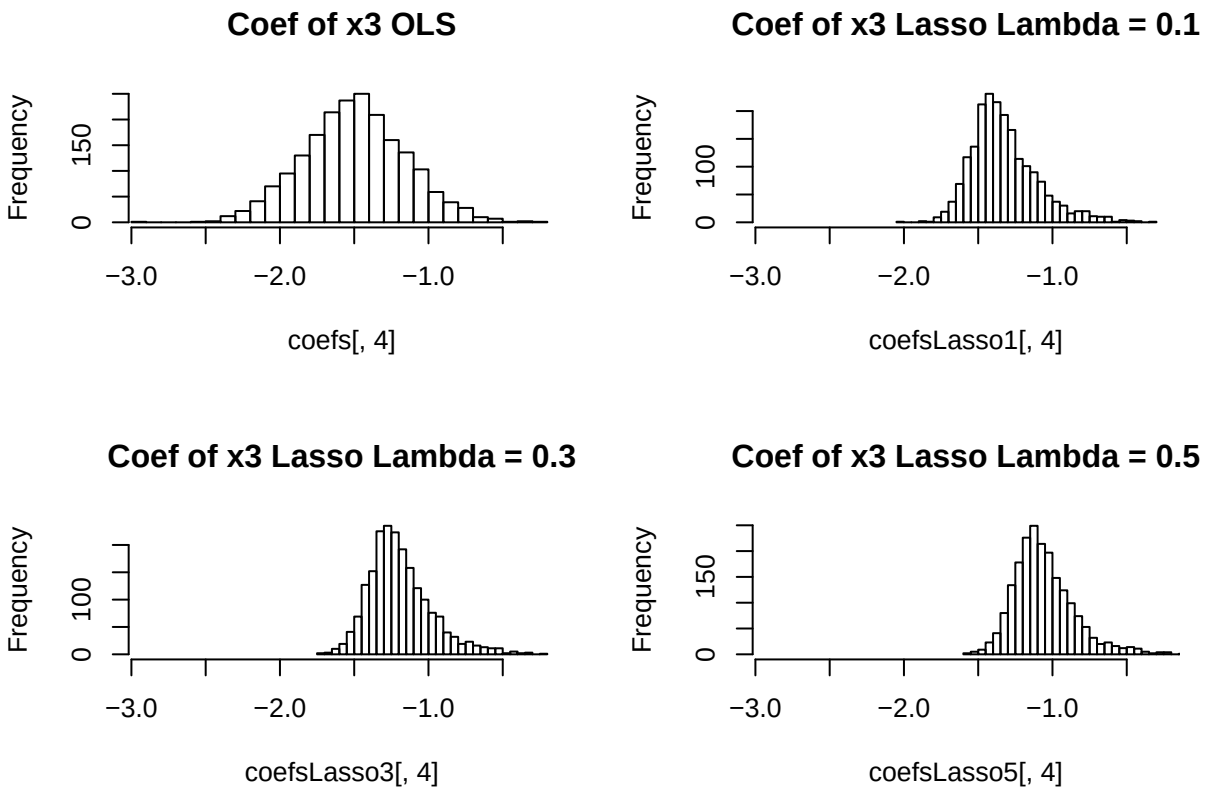

```
hist(coefs[,3], breaks = 25, main = "Coef of x2 OLS", xlim = xlimits)
hist(coefsRidge1[,3], breaks = 25, main = "Coef of x2 Ridge Lambda = 0.1", xlim = xlimits)
hist(coefsRidge3[,3], breaks = 25, main = "Coef of x2 Ridge Lambda = 0.3", xlim = xlimits)
hist(coefsRidge5[,3], breaks = 25, main = "Coef of x2 Ridge Lambda = 0.5", xlim = xlimits)
```



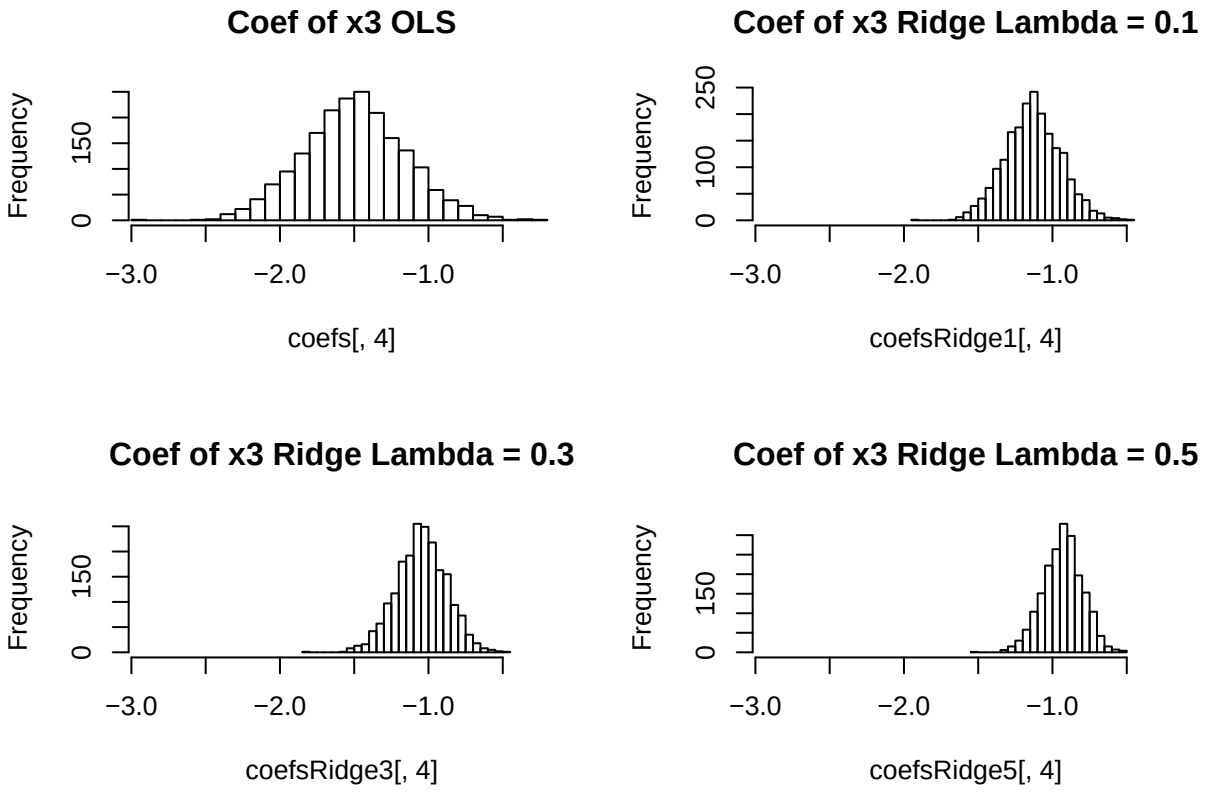
Comment on the plots, talk about the effect of lambda. How does lasso and ridge regression affect the parameter estimates you get? Keep in mind what the true value of the coefficient should be. What kind of bias do you see? What can you say about the variance of estimates?

The true coefficient value for X2 = 0, but we can see as we increase our λ we are getting further away from our true value for Ridge Regression. We can see there is a bit of bias in play in the Ridge Regression, but the Variance of our estimate is decreasing as we increase λ. The bias for Ridge Regression is around 0.50. As we decrease our Variance, we see an increase in a bias (Bias-Variance Tradeoff). As for LASSO, we can see that overall, our X2 coefficient estimate is approximately the same as our true value (X2 = 0). Increasing our λ for LASSO will bring us closer to the true value in this case and decreasing our Variance. There is still a little bit of bias around 0.50 but not very much.

```
par(mfrow = c(2,2))
xlimits = c(min(coefs[,4]), max(coefs[,4]))
hist(coefs[,4], breaks = 25, main = "Coef of x3 OLS", xlim = xlimits)
hist(coefsLasso1[,4], breaks = 25, main = "Coef of x3 Lasso Lambda = 0.1", xlim = xlimits)
hist(coefsLasso3[,4], breaks = 25, main = "Coef of x3 Lasso Lambda = 0.3", xlim = xlimits)
hist(coefsLasso5[,4], breaks = 25, main = "Coef of x3 Lasso Lambda = 0.5", xlim = xlimits)
```



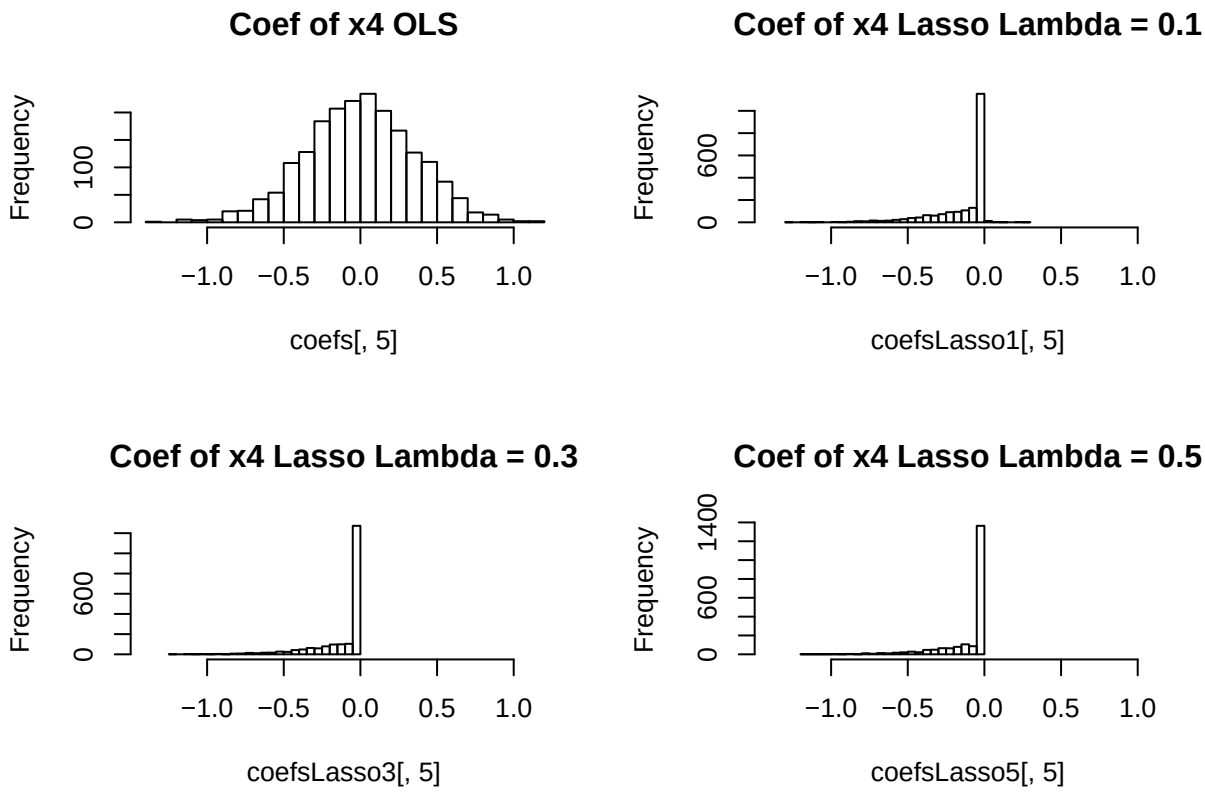

```r
hist(coefs[,4], breaks = 25, main = "Coef of x3 OLS", xlim = xlimits)
hist(coefsRidge1[,4], breaks = 25, main = "Coef of x3 Ridge Lambda = 0.1", xlim = xlimits)
hist(coefsRidge3[,4], breaks = 25, main = "Coef of x3 Ridge Lambda = 0.3", xlim = xlimits)
hist(coefsRidge5[,4], breaks = 25, main = "Coef of x3 Ridge Lambda = 0.5", xlim = xlimits)
```



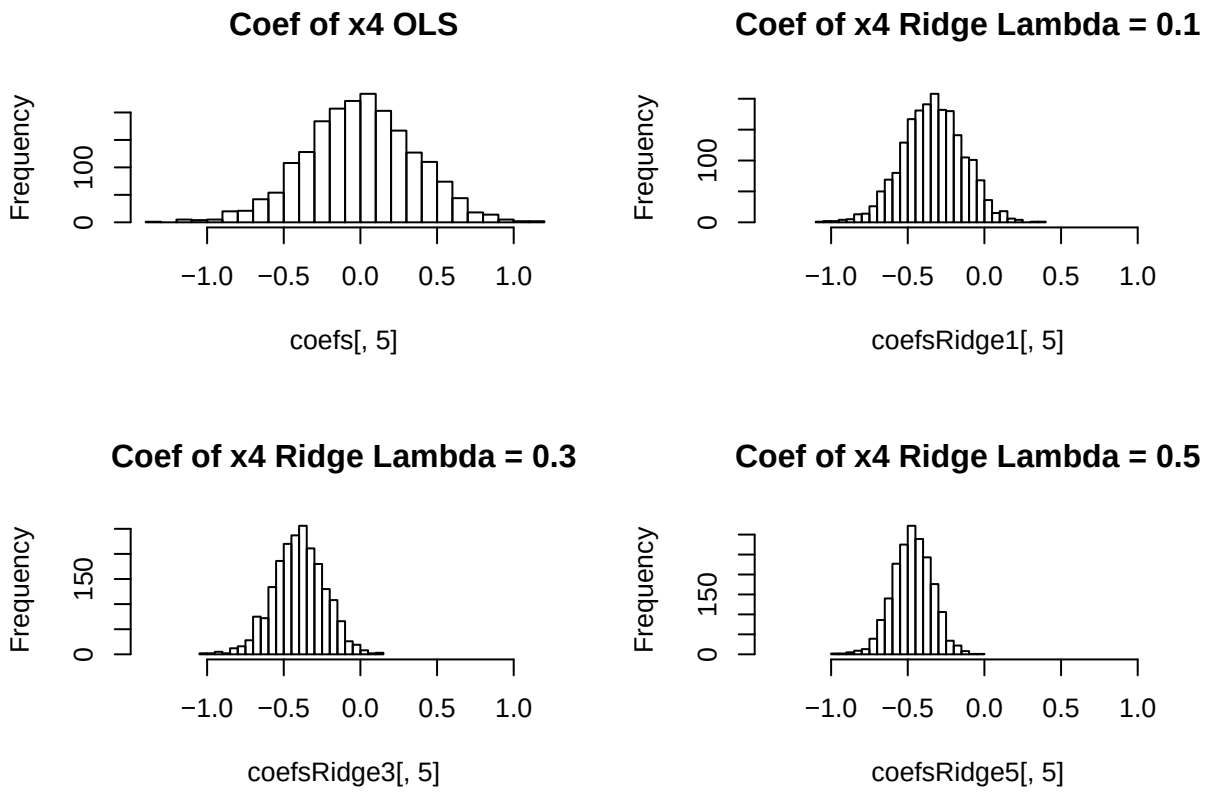
Comment on the plots, talk about the effect of lambda. How does lasso and ridge regression affect the parameter estimates you get? Keep in mind what the true value of the coefficient should be. What kind of bias do you see? What can you say about the variance of estimates?

The true coefficient value for X3 = -1.5, but we can see as we increase our λ we are getting further away from our true value. We can see there is a bit of bias in play in the Ridge Regression and LASSO, but the Variance of our estimate is decreasing as we increase λ. The bias is -1.2 (LASSO) and -0.9 (Ridge Regression). As we decrease our Variance, we see an increase in a bias (Bias-Variance Tradeoff). In this case, both methods are getting us further away from our true value (X3 = -1.5) for increasing λ.

```
par(mfrow = c(2,2))
xlimits = c(min(coefs[,5]), max(coefs[,5]))
hist(coefs[,5], breaks = 25, main = "Coef of x4 OLS", xlim = xlimits)
hist(coefsLasso1[,5], breaks = 25, main = "Coef of x4 Lasso Lambda = 0.1", xlim = xlimits)
hist(coefsLasso3[,5], breaks = 25, main = "Coef of x4 Lasso Lambda = 0.3", xlim = xlimits)
hist(coefsLasso5[,5], breaks = 25, main = "Coef of x4 Lasso Lambda = 0.5", xlim = xlimits)
```




```r
hist(coefs[,5], breaks = 25, main = "Coef of x4 OLS", xlim = xlimits)
hist(coefsRidge1[,5], breaks = 25, main = "Coef of x4 Ridge Lambda = 0.1", xlim = xlimits)
hist(coefsRidge3[,5], breaks = 25, main = "Coef of x4 Ridge Lambda = 0.3", xlim = xlimits)
hist(coefsRidge5[,5], breaks = 25, main = "Coef of x4 Ridge Lambda = 0.5", xlim = xlimits)
```



Comment on the plots, talk about the effect of lambda. How does lasso and ridge regression affect the parameter estimates you get? Keep in mind what the true value of the coefficient should be. What kind of bias do you see? What can you say about the variance of estimates?

The true coefficient value for X4 = 0, but we can see as we increase our λ we are getting further away from our true value for Ridge Regression. We can see there is a bit of bias in play in the Ridge Regression, but the Variance of our estimate is decreasing as we increase λ. The bias for Ridge Regression is around -0.50. As we decrease our Variance, we see an increase in a bias (Bias-Variance Tradeoff). As for LASSO, we can see that overall, our X4 coefficient estimate is approximately the same as our true value (X4 = 0). Increasing our λ for LASSO will bring us closer to the true value in this case and decreasing our Variance. There is still a little bit of bias around -0.50 but not very much.

Part 3: Ridge Regression: Gradients and Gradient Descent

Let's revisit the exact same artificial data we created at the beginning of the homework in Part 1.

Start with the same artificial Data

The variable t depends on two variables: x_1 and x_2. The true relationship between t and x_1 and x_2 is $t = 2x_1 + x_2$ but we add some random noise.

```
set.seed(6)
x1 <- 5 * runif(10)
x2 <- 4 * runif(10)
t <- 2 * x1 + x2 + rnorm(10, sd = 2)
X <- cbind(1, x1, x2)
```

The Loss function for Ordinary Least Squares Regression is:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w}) = \frac{1}{N}(\mathbf{t}^T\mathbf{t} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{t} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w})$$

The Loss function for Ridge Regression is:

$$\mathcal{L}' = \mathcal{L} + \lambda\mathbf{w}^T\mathbf{w}$$

Task 6:

Write a function to calculate the ridge regression loss.

The function will accept a vector `w = c(intercept, w1, w2)` and a value of lambda.

```
ridge_loss = function(w, lambda){
  ridge_func <- (loss(w) + lambda * (t(w) %*% w))
  return(ridge_func)
}
```

```
# for grading. I got 12.11428 and 27.11428
ridge_loss(c(1, 1, 1), lambda = 0)
```

```
##           [,1]
## [1,] 12.11428
```

```
ridge_loss(c(1, 1, 1), lambda = 5)
```

```
##           [,1]
## [1,] 27.11428
```

Use `optim()` to estimate the best parameter estimates.

```
# done for you
optim(par = c(0, 0, 0), ridge_loss, lambda = 0 )$par # should return the same value as OLS
```

```
## [1] 2.388649 1.341944 1.118625
```

```
optim(par = c(0, 0, 0), ridge_loss, lambda = 1 )$par
```

```
## [1] 0.7085387 1.5993243 1.2010740
```

Task 7:

Find the gradient of Ridge Regression Loss function.

Typeset your answer here:

$$\frac{\partial\mathcal{L}'}{\partial\mathbf{w}} = \frac{-2}{N}\mathbf{X}^T\mathbf{t} + \frac{2}{N}\mathbf{X}^T\mathbf{X}\mathbf{w} + 2\lambda\mathbf{w}$$

Set the gradient equal to 0, and solve for the best estimate of $\mathbf{w}$ that achieves this.

Typeset your answer here:

$$\hat{\mathbf{w}}_{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$$

Task 8:

Using matrix operation, verify that your closed form solution of

$$\hat{\mathbf{w}}_{ridge}$$

(closely) matches the best parameter estimates found by using `optim() for lambda = 1`

```
lambda = 1
w_hat_ridge <- solve(t(X) %*% X + lambda * diag(3)) %*% t(X) %*% t
w_hat_ridge
```

```
##        [,1]
##    1.591306
## x1 1.473182
## x2 1.231537
```

```
optim(par = c(0, 0, 0), ridge_loss, lambda = 1 )$par
```

```
## [1] 0.7085387 1.5993243 1.2010740
```

Task 9: Gradient Descent

Write a function to calculate the value of the gradient at a particular value of **w**.

The function will accept a vector `w = c(intercept, w1, w2)` and a value of lambda. You can use values of `X` and `t` or any other variables that are available in the global environment.

```
ridge_loss_grad <- function(w, lambda){
  N <- length(t)
  ridge_grad_func <- ((1/N) * (-2 * (t(X) %*% t) + (2 * (t(X) %*% X %*% w))))
  + 2 * lambda * w
  return(ridge_grad_func)
  }
```

Write a loop to perform gradient descent with lambda = 1. Start at the point `w = c(0,0,0)` and iterate.

Using trial and error, play around with your choice of gamma (hint: start small) and how many iterations to run (probably between 10 and 100 thousand).

Print out your choice of gamma (no right answer) and the resulting coefficient estimates at the end of the gradient descent algorithm (should be sorta close to the estimates produced by running `optim()` on the ridge regression loss function).

```
gamma = 0.001
total_iter <- 10e4
w = c(0,0,0)
lambda <- 1
for(i in 1:total_iter){
  w <- w - gamma * ridge_loss_grad(w, lambda)
}

w
```

```
##        [,1]
##    2.388634
## x1 1.342180
## x2 1.118182
```