Stat 432 Homework 3

Qianhua Zhou

Contents

Question 1: Another Simulation Study

Here is a GPT-4 prompt I used:

> What is the advantage of a Ridge regression? Can you suggest a simulation study to confirm that? Please write your code in R.

The following is the answer produced by GPT-4

Ridge regression is a regularization technique that helps to solve the problem of multicollinearity in linear regression. It can handle high-dimensional data where the number of predictors may exceed the number of observations. The advantages include:

1. **Bias-Variance Tradeoff**: By adding a penalty term, ridge regression can reduce variance, thus preventing overfitting. While it does introduce bias, this tradeoff often leads to a better overall predictive performance.
2. **Stability**: It can handle ill-posed problems where multicollinearity exists, providing stable solutions.
3. **Including All Predictors**: Unlike Lasso regression, ridge regression includes all predictors in the model, shrinking their coefficients but not setting them to zero.

You can simulate the advantage of Ridge regression by comparing its performance with ordinary linear regression on a dataset with multicollinearity. Here's an R script to conduct this study:

The script generates a dataset with multicollinearity and then compares the mean squared error (MSE) of ordinary linear regression and ridge regression. Typically, you'll observe that ridge regression provides a better performance in this scenario.

---

a. [15 points] Based on what we have learned, do you find this answer correct? does this code confirm the mentioned three properties of Ridge regressions? Please explain your judgement for each of them. Comment on how you would do to demonstrate these properties. You **do not need to** write the code.

   - Bias-Variance Trade-off
   - Stability
   - Including All Predictors

- Bias-Variance Trade-off: The code partially confirms this property by checking the performance via MSE, but it doesn't explicitly decompose MSE into bias and variance terms. To better demonstrate this tradeoff, it would need a method to separately quantify bias and variance. For instance, it should perform a cross-validation study or bootstrap the data to observe how the variance of the Ridge model changes relative to OLS.
- Stability: The code does not explicitly demonstrate stability. It only fit the model on one data sample. To test this property, it would need to examine how sensitive the coefficients are to changes in the dataset. One approach would be to resample the data multiple times and observe how much the coefficients fluctuate between OLS and Ridge regression. In Ridge, you should see less variability in the coefficients across samples, confirming its stability under multicollinearity.
- Including All Predictors: The code doesn't directly show this property. To confirm that Ridge keeps all predictors, it would need to inspect the coefficients of the Ridge model after fitting it. None of the coefficients should be exactly zero, as Ridge shrinks but does not eliminate predictors. Including a step to output the coefficients would confirm this behavior.

b. [25 points] To properly demonstrate the bias-variance trade-off, we could consider using a (correct) simulation. Adapt this existing code into a simulation study to show this properties. While you are doing this, please consider the following:
   - You can borrow similar ideas of simulation we used in previous lecture notes
   - Modify the GPT-4 code with the following settings to generate the data:
     * trainning sample size $trainn = 50$
     * Testing sample size $testn = 200$
     * $p = 200$
     * Fix $b = rep(0.1, p)$ for all simulation runs
   - Since linear regression doesn't work in this setting, you only need to consider `glmnet()`
   - Use a set of λ values `exp(seq(log(0.5), log(0.01), out.length = 100))*trainn`
   - Instead of evaluating the bias and variance separately (we will do that in the future), we will **use the testing error as the metric**.
   - Demonstrate your result using plots and give a clear explanation of your findings. Particularly, which side of the result displays a large bias, and which side corresponds to a large variance?

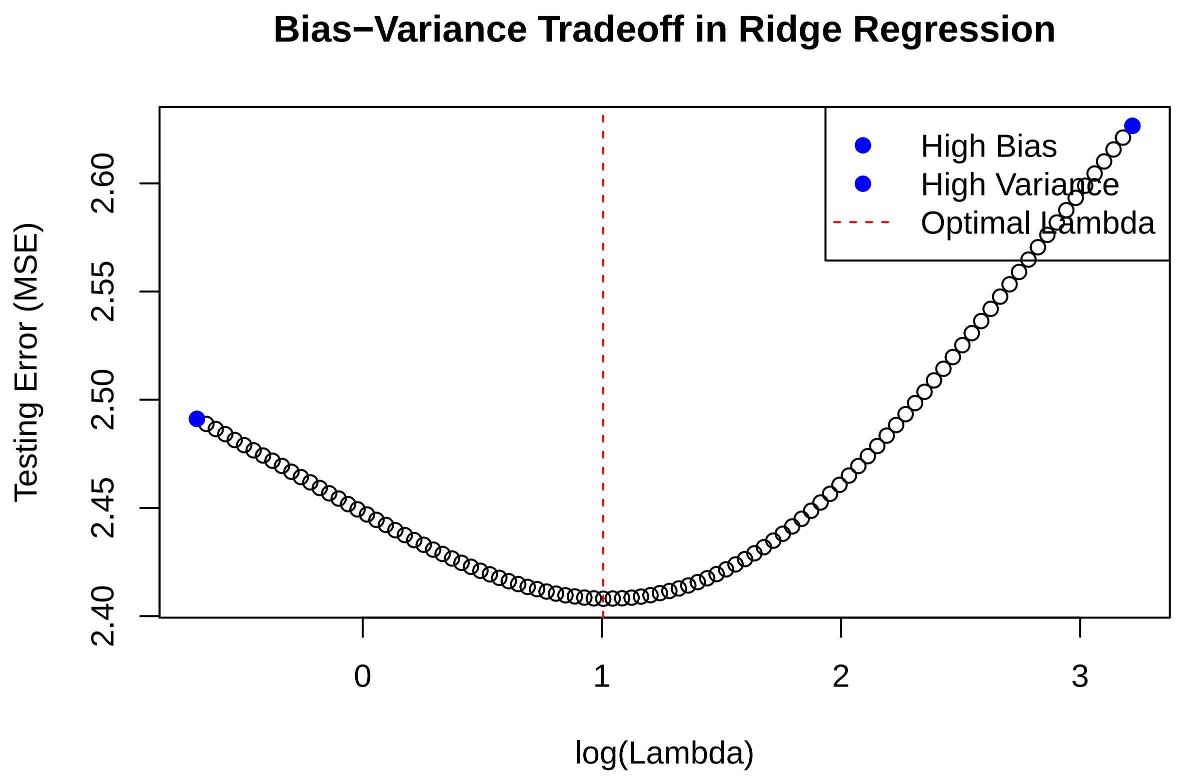
```
library(MASS)
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
set.seed(42)
# Simulation parameters
trainn <- 50 # Training sample size
testn <- 200 # Testing sample size
```

```r
p <- 200 # Number of predictors
b <- rep(0.1, p) # Coefficient vector (fixed)
# Lambda values (penalty parameter)
lambda_seq <- exp(seq(log(0.5), log(0.01), length.out = 100)) * trainn
# Function to generate data
generate_data <- function(n, p, b) {
X <- matrix(rnorm(n * p), n, p)
y <- X %*% b + rnorm(n) # Adding noise to the response
list(X = X, y = y)
}
# Generate training data
train_data <- generate_data(trainn, p, b)
X_train <- train_data$X
y_train <- train_data$y
# Generate testing data
test_data <- generate_data(testn, p, b)
X_test <- test_data$X
y_test <- test_data$y
# Initialize vector to store test errors
test_errors <- numeric(length(lambda_seq))
# Loop through the lambda values and fit ridge regression
for (i in seq_along(lambda_seq)) {
lambda_val <- lambda_seq[i]
# Fit ridge regression model (alpha=0 specifies ridge)
ridge_model <- glmnet(X_train, y_train, alpha = 0, lambda = lambda_val)
# Predict on the testing set
ridge_pred <- predict(ridge_model, newx = X_test, s = lambda_val)
# Compute test error (MSE)
test_errors[i] <- mean((y_test - ridge_pred)^2)
}
# Plot the testing errors as a function of lambda
plot(log(lambda_seq), test_errors, type = "b",
xlab = "log(Lambda)", ylab = "Testing Error (MSE)",
main = "Bias-Variance Tradeoff in Ridge Regression")
# Highlight areas of bias and variance
abline(v = log(lambda_seq[which.min(test_errors)]), col = "red", lty = 2)
text(log(lambda_seq[which.min(test_errors)]) + 0.2,
min(test_errors) + 0.5,
"Optimal Lambda", col = "red")
# Show points of high bias and high variance
points(log(lambda_seq[1]), test_errors[1], col = "blue", pch = 19) # Large variance
points(log(lambda_seq[length(lambda_seq)]), test_errors[length(lambda_seq)], col = "blue", pch = 19) #
legend("topright", legend = c("High Bias", "High Variance", "Optimal Lambda"),
col = c("blue", "blue", "red"), pch = c(19, 19, NA), lty = c(NA, NA, 2))
```



The plot demonstrates the bias-variance tradeoff in regularized regression using different values of λ. **On the left side, where λ is small, the model has low bias but high variance. As λ increases, the model becomes more regularized, reducing variance but increasing bias**, causes the testing error to rise. The red dashed line indicates the optimal λ, where the model achieves the best balance between bias and variance, minimizing the testing error. This reflects how increasing λ shifts the model from low-bias/high-variance to high-bias/low-variance.

The plot shows a U-shaped curve where both very low and very high values of lambda result in higher test MSE due to underfitting and overfitting, respectively. The red dashed line indicates the optimal λ, where the model achieves the best balance between bias and variance. The optimal λ value is where the testing error is minimized, striking a balance between bias and variance.

Question 2: Modeling High-Dimensional Data

We will use the `golub` dataset from the `multtest` package. This dataset contains 3051 genes from 38 tumor mRNA samples from the leukemia microarray study Golub et al. (1999). This package is not included in `R`, but on `bioconductor`. Install the latest version of this package from `bioconductor`, and read the documentation of this dataset to understand the data structure of `golub` and `golub.cl`.

a. [25 points] We will not use this data for classification (the original problem). Instead, we will do a toy regression example to show how genes are highly correlated and could be used to predict each. Carry out the following tasks:

   - Perform marginal association test for each gene with the response `golub.cl` using `mt.teststat()`. Use `t.equalvar` (two sample t test with equal variance) as the test statistic.
   - Sort the genes by their p-values and select the top 100 genes

- Construct a dataset with the top 10 genes and another one (call it X) with the remaining genes
- Perform principal component analysis (PCA) on the top 100 genes and extract the first principal component, **use this as the outcome** y. Becareful about the oriantation of the data matrix.
- Perform ridge regression with 19-fold cross-validation on X and the outcome y. Does your model fit well? Can you provide detailed model fitting results to support your claim?
- Fit ridge regression but use GCV as the criterion. Does your model fit well?

```r
#if (!requireNamespace("BiocManager", quietly = TRUE)) {
#    install.packages("BiocManager")
#}
#BiocManager::install("multtest")
library(multtest)
data(golub)
```

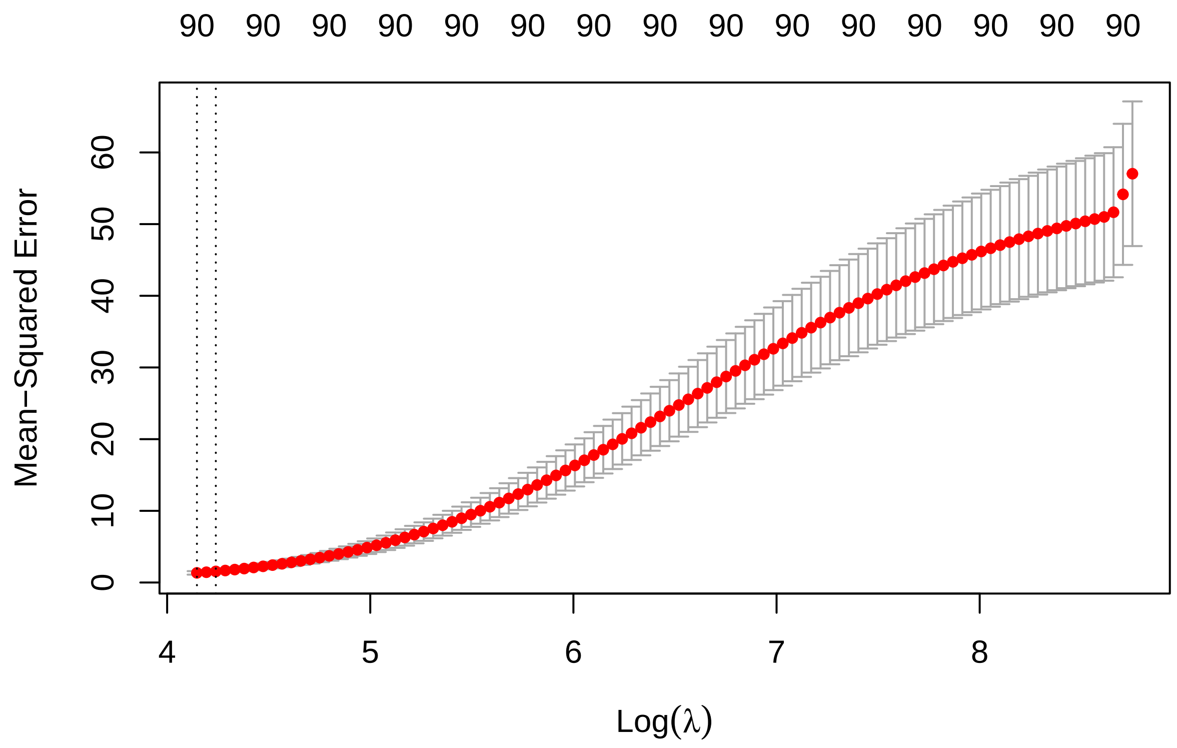
```r
t_stats <- mt.teststat(golub, golub.cl, test="t.equalvar")
p_values <- 2 * pt(-abs(t_stats), df = 36) # two-tailed p-values
sorted_indices <- order(p_values)
top_100_genes <- golub[sorted_indices[1:100], ]
top_10_genes <- golub[sorted_indices[1:10], ]
X <- golub[sorted_indices[11:100], ] # X
# PCA
pca_result <- prcomp(t(top_100_genes), scale. = TRUE)
# Extract the first few principal components
y <- pca_result$x[, 1]
# Load glmnet and perform ridge regression with cross-validation
library(glmnet)
set.seed(1)
cv_ridge <- cv.glmnet(t(X), y, alpha = 0, nfolds = 19)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```r
best_lambda <- cv_ridge$lambda.min
cat("Lambda value under 19-fold cross-validation:", best_lambda, "\n")
```

```
## Lambda value under 19-fold cross-validation: 63.22453
```

```r
# Plot to visualize lambda selection
plot(cv_ridge)
```



```r
# Extract coefficients at the optimal lambda
best_lambda <- cv_ridge$lambda.min
# Fit the final model with the best lambda
ridge_model <- glmnet(t(X), y, alpha = 0, lambda = best_lambda)
# Display coefficients
coef(ridge_model)
```

```
## 91 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)  3.75003331
## V1           0.22344308
## V2          -0.14724662
## V3           0.10096671
## V4           0.14810036
## V5           0.06678804
## V6           0.12122677
## V7          -0.11714696
## V8           0.12087425
## V9          -0.12508958
## V10         -0.15529850
## V11          0.13799593
## V12         -0.11466920
## V13         -0.18661617
## V14          0.09855022
## V15         -0.21397053
## V16          0.13420353
## V17          0.06236390
## V18          0.18348531
## V19          0.14296630
## V20         -0.25306392
## V21          0.10459252
## V22         -0.13880184
## V23         -0.08845951
## V24         -0.15844504
## V25          0.11333216
## V26          0.13555132
## V27          0.08183742
## V28          0.09968876
## V29          0.10101710
## V30         -0.10821511
## V31          0.16089232
## V32          0.11786225
## V33         -0.16114859
## V34          0.13276820
## V35         -0.12555321
## V36         -0.18420356
## V37          0.13743163
## V38          0.14320129
## V39          0.09175576
## V40          0.11431908
## V41          0.20881719
## V42          0.18589524
## V43          0.07547327
## V44          0.15256574
## V45         -0.16543395
## V46          0.10618685
## V47          0.10556113
## V48          0.20096384
## V49         -0.17664762
## V50         -0.17043693
## V51          0.12791491
## V52         -0.08337769
## V53         -0.16581938
## V54         -0.21597924
## V55          0.14273287
## V56         -0.15180050
## V57         -0.16110609
## V58         -0.15357680
## V59         -0.17943429
## V60          0.13188646
## V61         -0.15002133
## V62          0.11754486
## V63          0.06047721
## V64          0.17606099
## V65          0.15940233
## V66          0.09337962
## V67          0.11678008
## V68         -0.14147172
## V69         -0.06722247
## V70         -0.17972962
## V71          -0.08887630
## V72          -0.12962388
## V73          -0.09913002
## V74           0.09675740
## V75          -0.16274143
## V76           0.14423786
## V77          -0.20144932
## V78          -0.09314216
## V79          -0.11934818
## V80          -0.17083549
## V81          -0.15297295
## V82           0.08440054
## V83           0.17497310
## V84          -0.10051948
## V85          -0.22256635
## V86           0.09877243
## V87          -0.15733827
## V88          -0.23749983
## V89          -0.24227097
## V90          -0.13876792
```

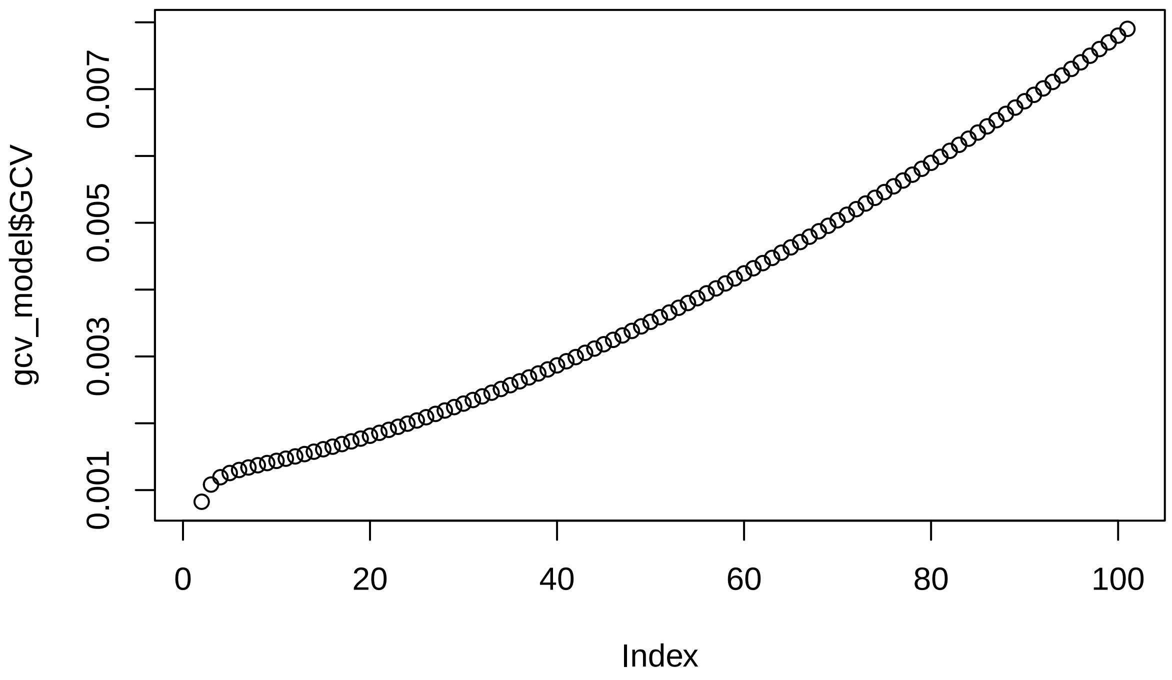
```
predictions <- predict(ridge_model, newx = t(X))
mse <- mean((y - predictions)^2)
r_squared <- 1 - sum((y - predictions)^2) / sum((y - mean(y))^2)
cat("Mean Squared Error:", mse, "\n")
```

```
## Mean Squared Error: 1.249251
```

```
cat("R-squared:", r_squared, "\n")
```

```
## R-squared: 0.9772938
```

```
library(MASS)
# Fit ridge regression using GCV criterion
gcv_model <- lm.ridge(y ~ t(X), lambda = seq(0, 100, by = 1))
# Plot GCV values
plot(gcv_model$GCV)
```



```r
# Get the lambda with the lowest GCV
best_gcv_lambda <- gcv_model$lambda[which.min(gcv_model$GCV)]
cat("Lambda value under GCV:", best_gcv_lambda, "\n")
```

```
## Lambda value under GCV: 1
```

```r
# Fit the final ridge model using the best GCV lambda
gcv_final_model <- lm.ridge(y ~ t(X), lambda = best_gcv_lambda)

# Display coefficients
gcv_coef <- coef(gcv_final_model)
# Calculate fitted values using the GCV model
# Predictions
intercept <- gcv_coef[1]
coefficients <- gcv_coef[-1]
predictions_gcv <- intercept + as.matrix(t(X)) %*% coefficients
# Calculate MSE
mse <- mean((y - predictions_gcv)^2)
# Calculate R-squared
ss_res <- sum((y - predictions_gcv)^2)
ss_tot <- sum((y - mean(y))^2)
r_squared <- 1 - (ss_res / ss_tot)
cat("Mean Squared Error:", mse, "\n")
```

```
## Mean Squared Error: 0.0002225726
```

```r
cat("R-squared:", r_squared, "\n")
```

```
## R-squared: 0.999996
```

For the ridge regression with 19-fold cross-validation, since the R-squared is 0.9772938, which is above 0.7,and very close to 1, the model is a good fit.

For the ridge regression using GCV, since the R-squared is0.999996, which is above 0.7, and very close to 1, also, mse is 0.0002225726, very close to 0, the model is agood fit.

b. [5 points] Based on your results, do you observe any bias-variance trade-off? If not, can you explain why?

Based on the results, I don't observe bias-variance trade-off. the curve is relatively flat, indicating that the model is not sensitive to changes in λ. And cross-validation curve doesn't show a U-shape with a minimum point for λ. It might because:

- The model is not overly sensitive to changes in regularization.When reviewing the coefficients from the ridge regression model fit with the optimal λ. From the output, many coefficients are close to zero, indicating regularization is not significantly penalizing them. This further supports the idea that there might not be enough variability in X to observe a strong bias-variance trade-off.
- The response y (first principal component) might not be highly complex, or the data matrix X may not have enough variability for a noticeable trade-off to occur.
- The optimal λ could simply be at one extreme of the λ range, which might not exhibit a strong trade-off in the tested range.

Question 3: Linear Regression with Coordinate Descent

Recall the previous homework, we have a quadratic function for minimization. We know that analytical solution exist. However, in this example, let's use coordinate descent to solve the problem. To demonstrate this, let's consider the following simulated dataset, with design matrix x (without intercept) and response vector y:

```r
set.seed(432)
n <- 100
x <- matrix(rnorm(n*2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)
```

We will consider a model without the intercept term. In this case, our objective function (of β_1 and β_2 for linear regression is to minimize the sum of squared residuals:

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2$$

where x_{ij} represents the jth variable of the ith observation.

a. [10 points] Write down the objective function in the form of

$$f(x, y) = a\beta_1^2 + b\beta_2^2 + c\beta_1\beta_2 + d\beta_1 + e\beta_2 + f$$

by specifying what are coefficients a, b, c, d, e, and f, using the simulated data. Calculate them in R, **using vector operations rather than for-loops**.

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}\left[y_i^2 - 2y_i\beta_1 x_{i1} - 2y_i\beta_2 x_{i2} + \beta_1^2 x_{i1}^2 + 2\beta_1\beta_2 x_{i1}x_{i2} + \beta_2^2 x_{i2}^2\right]$$

$$a = \frac{1}{n}\sum_{i=1}^{n} x_{i1}^2, \quad b = \frac{1}{n}\sum_{i=1}^{n} x_{i2}^2, \quad c = \frac{2}{n}\sum_{i=1}^{n} x_{i1}x_{i2}$$

$$d = \frac{-2}{n}\sum_{i=1}^{n} y_i x_{i1}, \quad e = \frac{-2}{n}\sum_{i=1}^{n} y_i x_{i2}, \quad f = \frac{1}{n}\sum_{i=1}^{n} y_i^2$$

where x_{ij} represents the jth variable of the ith observation.

```r
set.seed(432)
n <- 100
x <- matrix(rnorm(n * 2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)

# Calculate coefficients using vector operations
a <- mean(x[, 1]^2)
b <- mean(x[, 2]^2)
c <- 2 * mean(x[, 1] * x[, 2])
d <- -2 * mean(y * x[, 1])
e <- -2 * mean(y * x[, 2])
f <- mean(y^2)

# Output the coefficients
cat("a =", a, "b =", b, "c =", c, "d =", d, "e =", e, "f =", f)
```

```
## a = 0.9241812 b = 0.8625308 c = -0.2694035 d = -1.122192 e = -0.5161552 f = 1.25586
```

b. [10 points] A coordinate descent algorithm essentially does two steps: i. Update β_1 to its optimal value while keeping β_2 fixed ii. Update β_2 to its optimal value while keeping β_1 fixed

   Write down the updating rules for β_1 and β_2 using the coordinate descent algorithm. Use those previously defined coefficients in your fomula and write them in Latex. Implement them in a for-loop algorithm in R that iterates at most 100 times. Use the initial values $\beta_1 = 0$ and $\beta_2 = 0$. Decide your stopping criterion based on the change in β_1 and β_2. Validate your solution using the lm() function.

The coordinate descent algorithm for linear regression iteratively updates each coefficient while keeping the other fixed. The objective function is given by:

Updating β_1:

When β_2 is fixed, we treat the objective function as a quadratic function of β_1:

$$\frac{\partial f}{\partial \beta_1} = 2a\beta_1 + c\beta_2 + d = 0$$

Solving for β_1:

$$\beta_1 = \frac{-c\beta_2 - d}{2a}$$

Updating β_2:

Similarly, when β_1 is fixed, we treat the objective function as a quadratic function of β_2:

$$\frac{\partial f}{\partial \beta_2} = 2b\beta_2 + c\beta_1 + e = 0$$

Solving for β_2:

$$\beta_2 = \frac{-c\beta_1 - e}{2b}$$

```r
# Set seed for reproducibility
set.seed(432)
n <- 100
x <- matrix(rnorm(n * 2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)

# Calculate coefficients from part a
a <- mean(x[, 1]^2)
b <- mean(x[, 2]^2)
c <- 2 * mean(x[, 1] * x[, 2])
d <- -2 * mean(y * x[, 1])
e <- -2 * mean(y * x[, 2])

# Initialize beta1 and beta2
beta1 <- 0
beta2 <- 0

# Set convergence threshold and max iterations
threshold <- 1e-6
max_iter <- 100

# Coordinate descent algorithm
for (iter in 1:max_iter) {
  # Store previous values of beta1 and beta2
  beta1_old <- beta1
  beta2_old <- beta2

  # Update beta1 and beta2
  beta1 <- (-c * beta2 - d) / (2 * a)
  beta2 <- (-c * beta1 - e) / (2 * b)

  # Check for convergence
  if (abs(beta1 - beta1_old) < threshold && abs(beta2 - beta2_old) < threshold) {
    break
  }
}

# Output the final beta values and number of iterations
cat("Final beta1:", beta1, "\n")
```

```
## Final beta1: 0.6658955
```

```r
cat("Final beta2:", beta2, "\n")
```

```
## Final beta2: 0.4032028
```

```r
cat("Number of iterations:", iter, "\n")
```

```
## Number of iterations: 5
```

```r
# Validate with lm()
model <- lm(y ~ x[, 1] + x[, 2] - 1) # -1 to exclude intercept
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x[, 1] + x[, 2] - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.44478 -0.60651 -0.01577  0.53748  2.26058
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## x[, 1]  0.66590    0.09377   7.102 1.98e-10 ***
## x[, 2]  0.40320    0.09706   4.154 6.98e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8911 on 98 degrees of freedom
## Multiple R-squared:  0.3804, Adjusted R-squared:  0.3677
## F-statistic: 30.08 on 2 and 98 DF,  p-value: 6.525e-11
```