

HW03_zilinw3

Zilin Wang (zilinw3)

2024-09-17

Contents

Question 1: Another Simulation Study	1
Question 2: Modeling High-Dimensional Data	6
Question 3: Linear Regression with Coordinate Descent	12

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:

```

# Load required library
library(MASS)
library(glmnet)

# Simulate data with multicollinearity
set.seed(42)
n <- 100
p <- 10
X <- matrix(rnorm(n * p), n, p)
X[,2] <- X[,1] + rnorm(n) * 0.01 # Introducing multicollinearity
beta <- runif(p)
y <- X %*% beta + rnorm(n)

# Ordinary Linear Regression
lm_model <- lm(y ~ X)
lm_pred <- predict(lm_model, newdata=data.frame(X=X))

# Ridge Regression
ridge_model <- glmnet(X, y, alpha=0)
ridge_pred <- predict(ridge_model, s=0.01, newx=X)

# Compare Performance
lm_mse <- mean((y - lm_pred)^2)
ridge_mse <- mean((y - ridge_pred)^2)

cat("MSE of Ordinary Linear Regression:", lm_mse, "\n")
cat("MSE of Ridge Regression:", ridge_mse, "\n")

```

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).

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

Answer:

I find that the answer is not correct. Because a lower MSE indicates better fit, and the MSE of ridge regression is 0.8499741, which is larger than the MSE of ordinary linear regression(0.8251734). Therefore,

ridge regression does not provide a better performance in this scenario, instead, ordinary linear regression provides a better performance.

The code confirms the Stability and Including All Predictors of Ridge regressions, but not confirm Bias-Variance Trade-off.

Bias-Variance Trade-off: By adding a penalty term, ridge regression can reduce variance, thus preventing overfitting. To select the correct ridge regression model, we need to select the correct lambda. We can perform ridge regression with cross-validation and pick the model with the lowest CV MSE, which indicates that there is an appropriate balance between the bias and the variance (Bias-Variance Trade-off). However, the code does not use cross-validation to pick the best lambda. So the lambda used may not be the best for the ridge regression model. Thus, the MSE of the ridge regression is larger than the ordinary linear regression.

Stability: Ridge regression is known to provide more stable estimates in the presence of multicollinearity because it tends to shrink the coefficients, distributing the effect across correlated predictors and thus mitigating the problem of inflated variances. The code uses a dataset with induced multicollinearity by making columns of X highly correlated ($X[,2] <- X[,1] + \text{rnorm}(n) * 0.01$), and it compares the MSE between Ridge and ordinary regression. However, the demonstration of stability isn't explicit, the code does not directly examine the coefficients to show that Ridge shrinks them, stabilizing the estimates. Stability could be better shown by comparing the coefficient variance of OLR and ridge regression across multiple simulations by printing them to show that Ridge produces smaller, more stable coefficients when multicollinearity is present.

Including All Predictors: In the code, the ridge regression includes all predictors in the model, shrinking their coefficients but not setting them to zero through the glmnet function with alpha=0 (ridge penalty). However, there's no direct observation or reporting of how coefficients are shrunk, not eliminated. We can add this.

(b).

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:
 - * training sample size \$trainn = 50\$
 - * Testing sample size \$testn = 200\$
 - * \$p = 200\$
 - * Fix \$b = \text{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 \$\lambda\$ 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 ...
- * Demonstrate your result using plots and give a clear explanation of your findings. Particularly, ...

Answer:

```
library(glmnet)
library(MASS)

set.seed(42)
trainn <- 50
```

```

testn <- 200
p <- 200

# Simulate predictors with multicollinearity
X <- matrix(rnorm(trainn * p), trainn, p)
b <- rep(0.1, p)
noise <- rnorm(trainn)
Y <- X %*% b + noise

# Prepare test data
X_test <- matrix(rnorm(testn * p), testn, p)
Y_test <- X_test %*% b + rnorm(testn)

# Define a range of lambda for ridge regression
lambda_values <- exp(seq(log(0.5), log(0.01), length.out = 100)) * trainn

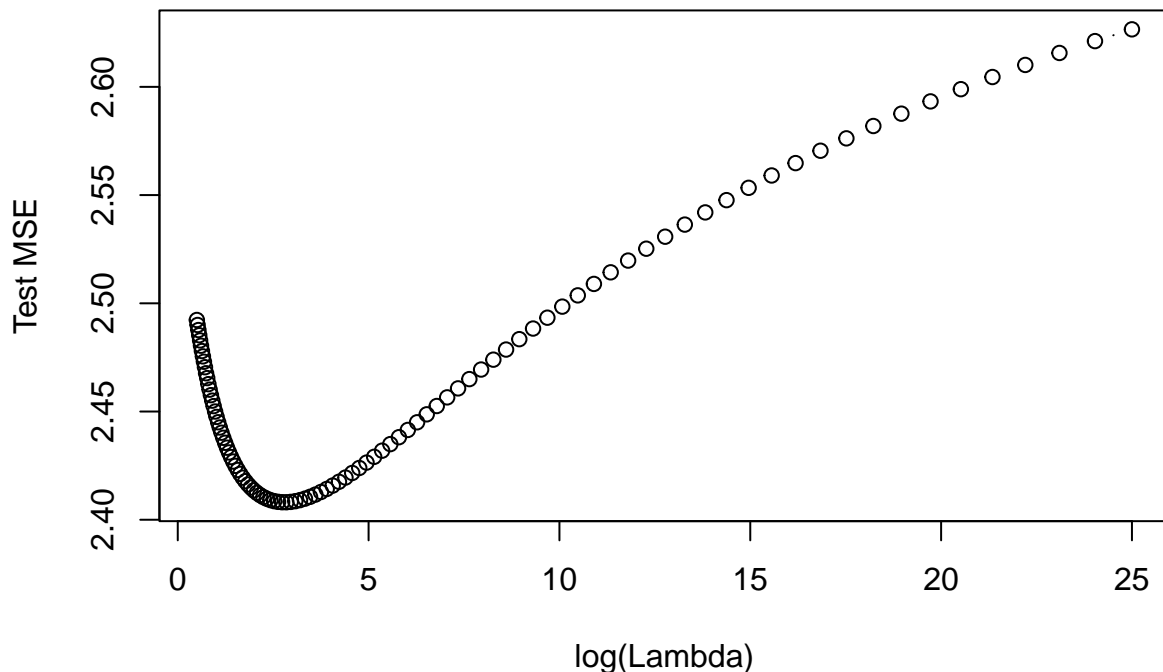
# Fit ridge regression model
ridge_model <- glmnet(X, Y, alpha = 0, lambda = lambda_values)

# Predict on test set and calculate MSE for each lambda
mse <- sapply(lambda_values, function(l) {
  pred <- predict(ridge_model, newx = X_test, s = l)
  mean((Y_test - pred)^2)
})

# Plot MSE against log(lambda)
plot(lambda_values, mse, type = 'b', xlab = "log(Lambda)", ylab = "Test MSE",
      main = "Ridge Regression: Bias-Variance Tradeoff")

```

Ridge Regression: Bias–Variance Tradeoff



```
# Find the minimum lambda value with the test errors
lambda_min <- lambda_values[which.min(mse)]
# Print the lambda value with the minimum test error
cat("Lambda with minimum test error:", lambda_min, "\n")
```

```
## Lambda with minimum test error: 2.844905
```

A lower MSE at a certain level of lambda suggests an optimal tradeoff between bias and variance. So, when lambda is 2.844905, the model is optimally balanced. As lambda increases (more penalty), we should observe an increase in bias but a decrease in 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.

When lambda is large (right side of the plot where log(Lambda) is higher), the model's complexity is significantly reduced. In this scenario, ridge regression heavily penalizes the coefficients, potentially leading to underfitting. A larger λ increases bias (simpler models) but decreases variance, so the right side of the result displays a large bias.

Conversely, when lambda is small (left side of the plot where log(Lambda) is lower), the model allows more complexity by applying less shrinkage to the coefficients. This scenario can lead to overfitting. A smaller λ decreases bias (more complex models) but increases variance, so the left side of the result corresponds to a large 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).

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()`.
- * 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 `XX`) with the remaining genes
- * Perform principal component analysis (PCA) on the top 100 genes and extract the first principal component
- * Perform ridge regression with 19-fold cross-validation on `XX` and the outcome `Y`. Does your model fit well?
- * Fit ridge regression but use GCV as the criterion. Does your model fit well?

Answer:

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

##  num [1:3051, 1:38] -1.458 -0.752 0.457 3.135 2.766 ...
##  - attr(*, "dimnames")=List of 2
##    ..$ : NULL
##    ..$ : NULL

str(golub.cl)

##  num [1:38] 0 0 0 0 0 0 0 0 0 0 ...

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

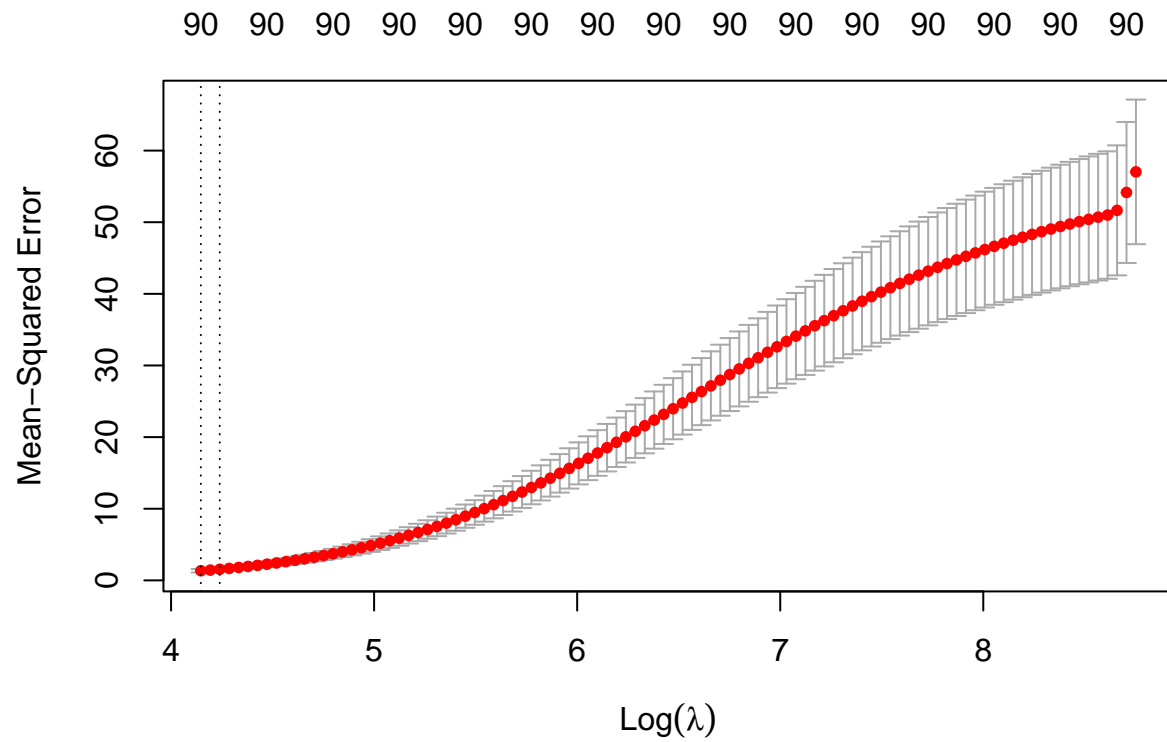
# PCA
pca_result <- prcomp(t(top_100_genes), scale. = TRUE)
# Extract the first few principal components
```

```

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)
# Plot to visualize lambda selection
plot(cv_ridge)

```



```

# 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
```

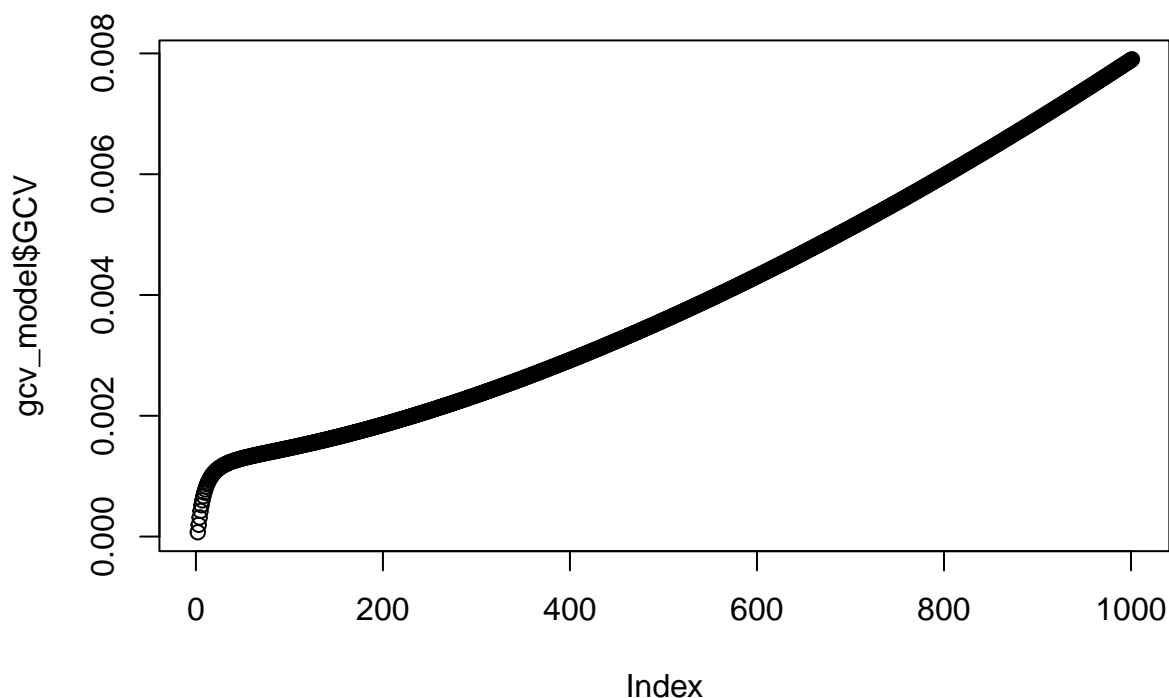
```
predictions <- predict(ridge_model, newx = t(X))
mse <- mean((y - predictions)^2)
r_squared <- 1 - sum((y - predictions)^2) / sum((y - mean(y))^2)
cat("Mean Squared Error:", mse, "\n")
```

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

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

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

```
# Ridge regression with GCV
library(MASS)
# Fit ridge regression using GCV criterion
gcv_model <- lm.ridge(y ~ t(X), lambda = seq(0, 100, by = 0.1))
# Plot GCV values
plot(gcv_model$GCV)
```



```
# Get the lambda with the lowest GCV
best_gcv_lambda <- gcv_model$lambda[which.min(gcv_model$GCV)]
# 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: 2.938193e-06
```

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

```
## R-squared: 0.9999999
```

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 is 0.9999999, which is above 0.7, and very close to 1, also, mse is 2.938193e-06, very close to 0, the model is a good fit.

(b).

Based on your results, do you observe any bias-variance trade-off? If not, can you explain why?

Answer:

Based on my results, I observed bias-variance trade-off.

The bias-variance trade-off can be discussed based on how well the model performs across different values of the regularization parameter λ . Ridge regression typically increases bias but reduces variance by introducing shrinkage.

The plot from `cv.glmnet()` visualizes the trade-off by showing the mean squared error across different values of λ . We performed ridge regression with cross-validation and picked the model with the lowest CV MSE, which indicates that there is an appropriate balance between the bias and the variance (Bias-Variance Trade-off). The output indicates the best lambda, which has the lowest MSE, by using cross-validation and GCV. So the lambda used is the best for the ridge regression model, which suggests an optimal tradeoff between bias and variance.

Also, by using the first principal component of the top 100 genes, the model simplifies the high-dimensional gene expression data into a single predictor, which inherently involves a trade-off between losing information (higher bias) and reducing noise (lower variance).

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 :

```
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 j th variable of the i th observation.

(a).

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**.

Answer:

Given the objective function:

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

Then we expand the objective function:

$$f(\beta_1, \beta_2) = \frac{1}{n} \sum_{i=1}^n (y_i^2 - 2y_i\beta_1x_{i1} - 2y_i\beta_2x_{i2} + \beta_1^2x_{i1}^2 + 2\beta_1\beta_2x_{i1}x_{i2} + \beta_2^2x_{i2}^2)$$

By grouping terms by coefficients, we can re-arrange it to match the form $f(x, y) = a\beta_1^2 + b\beta_2^2 + c\beta_1\beta_2 + d\beta_1 + e\beta_2 + f$:

- $a = \frac{\sum x_{i1}^2}{n}$
- $b = \frac{\sum x_{i2}^2}{n}$
- $c = \frac{2\sum x_{i1}x_{i2}}{n}$
- $d = \frac{-2\sum y_i x_{i1}}{n}$

- $e = \frac{-2 \sum_i y_i x_{i2}}{n}$
- $f = \frac{\sum_i y_i^2}{n}$

We then calculate these coefficients in R.

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

a <- sum(x[,1]^2) / n
b <- sum(x[,2]^2) / n
c <- 2 * sum(x[,1] * x[,2]) / n
d <- -2 * sum(y * x[,1]) / n
e <- -2 * sum(y * x[,2]) / n
f <- sum(y^2) / n
cat("a is :", a, "\n")
```

```
## a is : 0.9241812
```

```
cat("b is :", b, "\n")
```

```
## b is : 0.8625308
```

```
cat("c is :", c, "\n")
```

```
## c is : -0.2694035
```

```
cat("d is :", d, "\n")
```

```
## d is : -1.122192
```

```
cat("e is :", e, "\n")
```

```
## e is : -0.5161552
```

```
cat("f is :", f, "\n")
```

```
## f is : 1.25586
```

(b).

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

Answer:

Update for β_1 :

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

Update for β_2 :

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

Given the function $f(\beta_1, \beta_2)$ as previously defined, we derive the update rules by solving for the minimum (setting the derivatives to zero).

The derivatives of the objective function with respect to β_1 and β_2 are set to zero to find the points where the function has stationary points, which are potential minima:

- For β_1 :

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

Solving for β_1 :

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

- For β_2 :

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

Solving for β_2 :

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

Then we implement them in a for-loop algorithm in R.

```
beta1 <- 0
beta2 <- 0
epsilon <- 1e-6
max_iter <- 100

for (i in 1:max_iter) {
  beta1_old <- beta1
  beta2_old <- beta2

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

  # Check convergence
  if (abs(beta1 - beta1_old) < epsilon && abs(beta2 - beta2_old) < epsilon) {
    cat("Stop when convergence reached in", i, "iterations.\n")
  }
}
```

```

    break
  }
}

```

```
## Stop when convergence reached in 5 iterations.
```

```
cat("Final estimates: Beta1 =", beta1, ", Beta2 =", beta2, "\n")
```

```
## Final estimates: Beta1 = 0.6658955 , Beta2 = 0.4032028
```

```
# Validation using lm() function
```

```
lm_fit <- lm(y ~ x - 1)
```

```
summary(lm_fit)
```

```
##
```

```
## Call:
```

```
## lm(formula = y ~ x - 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|)
```

```
## x1  0.66590     0.09377   7.102 1.98e-10 ***
```

```
## x2  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
```