Stat 432 Homework 2

Assigned: Sep 2, 2024; Due: 11:59 PM CT, Sep 12, 2024

Contents

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope. No email or hard copy will be accepted. For **late submission policy and grading rubrics**, please refer to the course website.

Question 1 (Continuing the Simulation Study)

During our lecture, we considered a simulation study using the following data generator:

$$Y = \sum_{j=1}^{p} X_j 0.4^{\sqrt{j}} + \epsilon$$

And we added covariates one by one (in their numerical order, which is also the size of their effect) to observe the change of training error and testing error. However, in practice, we would not know the order of the variables. Hence several model selection tools were introduced. In this question, we will use similar data generators, with several nonzero effects, but use different model selection tools to find the best model. The goal is to understand the performance of model selection tools under various scenarios. Let's first consider the following data generator:

$$Y = \frac{1}{2} \cdot X_1 + \frac{1}{4} \cdot X_2 + \frac{1}{8} \cdot X_3 + \frac{1}{16} \cdot X_4 + \epsilon$$

where $\epsilon \sim N(0, 1)$ and $X_j \sim N(0, 1)$ for $j = 1, \ldots, p$. Write your code the complete the following tasks:

a. [10 points] Generate one dataset, with sample size $n = 100$ and dimension $p = 20$ as our lecture note. Perform best subset selection (with the `leaps` package) and use the AIC criterion to select the best model. Report the best model and its prediction error. Does the approach **selects the correct model**, meaning that all the nonzero coefficient variables are selected and all the zero coefficient variables are removed? Which variable(s) was falsely selected and which variable(s) was falsely removed? **Do not consider the intercept term**, since they are always included in the model. Why do you think this happens?

```r
library(leaps)
library(MASS)

# Set seed for reproducibility
set.seed(1)

# Generate dataset with n = 100 and p = 20
n <- 100
p <- 20
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
epsilon <- rnorm(n)
beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))  # True coefficients
Y <- X %*% beta + epsilon  # Generate response variable

# Perform best subset selection using 'leaps' package
best_subset <- regsubsets(Y ~ ., data = as.data.frame(X), nvmax = p)
best_subset_summary <- summary(best_subset)

# Calculate AIC for all models
aic_values <- n * log(best_subset_summary$rss / n) + 2 * (1:p + 1)

# Identify the model with the lowest AIC
best_model_idx <- which.min(aic_values)
cat("Best model selected by AIC (model size):", best_model_idx, "\n")
```

```
## Best model selected by AIC (model size): 5
```

```r
# Extract the coefficients of the best model
best_model_coef <- coef(best_subset, best_model_idx)
cat("Selected coefficients for the best model:\n")
```

```
## Selected coefficients for the best model:
```

```r
print(best_model_coef)
```

```
## (Intercept)          V1          V2          V3          V8         V13
##  -0.1042164   0.6060618   0.4484825   0.2622793  -0.1780701  -0.1748998
```

```r
# Identify which variables were selected
selected_variables <- names(best_model_coef)[-1]  # Exclude intercept
selected_indices <- as.numeric(sub("V", "", selected_variables))
cat("Selected variables (indices):", selected_indices, "\n")
```

```
## Selected variables (indices): 1 2 3 8 13
```

```r
# Generate new test data
X_test <- matrix(rnorm(n * p), nrow = n, ncol = p)
Y_test <- X_test %*% beta + rnorm(n)

# Predict using the selected variables from the best model
X_test_selected <- X_test[, selected_indices, drop = FALSE]
Y_pred <- best_model_coef[1] + X_test_selected %*% best_model_coef[-1]

# Calculate the mean squared error (MSE) on the test set
mse <- mean((Y_test - Y_pred)^2)
cat("Prediction error (MSE):", mse, "\n")
```

```
## Prediction error (MSE): 1.116132
```

```r
# Compare selected variables with true non-zero variables
true_nonzero_vars <- which(beta != 0)
falsely_selected <- setdiff(selected_indices, true_nonzero_vars)
falsely_removed <- setdiff(true_nonzero_vars, selected_indices)

cat("Variables falsely selected (shouldn't be in the model):", falsely_selected, "\n")
```

```
## Variables falsely selected (shouldn't be in the model): 8 13
```

```r
cat("Variables falsely removed (should be in the model):", falsely_removed, "\n")
```

```
## Variables falsely removed (should be in the model): 4
```

The best subset selection approach based on AIC identified a model with five variables: V1, V2, V3, V8, and V13. However, the approach did not select the correct model, as it falsely included V8 and V13, which have zero true coefficients, and falsely excluded V4, which has a nonzero coefficient in the true model. The prediction error (MSE) was 1.116, which is reasonable but not optimal due to the inclusion of irrelevant variables and exclusion of a relevant one. This misselection likely occurs because AIC tends to balance model fit and complexity, but it can overfit when noise variables correlate with the true predictors or when the sample size is not large enough to clearly distinguish the effects of the relevant variables.

b. [10 points] Repeat the previous step with 100 runs of simulation, similar to our lecture note. Report

   i. the proportion of times that this approach selects the correct model
   ii. the proportion of times that each variable was selected

```r
library(leaps)

# Set seed for reproducibility
set.seed(1)

# Parameters for simulation
simulations <- 100
n <- 100
p <- 20
correct_model_count <- 0
selected_counts <- numeric(p)  # To store the count of times each variable is selected
```

```r
correct_vars <- c(1, 2, 3, 4)  # Indices of the true non-zero coefficients
beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))  # True beta coefficients

# Run the simulation for 100 iterations
for (sim in 1:simulations) {
  # Step 1: Generate new data for each simulation run
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)  # New random X matrix
  epsilon <- rnorm(n)  # Random error term
  Y <- X %*% beta + epsilon  # Generate response variable Y

  # Step 2: Best subset selection using leaps package
  subset_model <- regsubsets(Y ~ ., data = as.data.frame(X), nvmax = p)
  subset_model_summary <- summary(subset_model)

  # Step 3: AIC calculation to select the best model
  rss <- subset_model_summary$rss
  aic_values <- n * log(rss / n) + 2 * (1:p + 1)
  best_model_idx <- which.min(aic_values)  # Find the best model index based on AIC

  # Step 4: Extract selected variables (excluding the intercept)
  selected_vars <- which(subset_model_summary$which[best_model_idx, -1])

  # Step 5: Check if the selected model matches the correct model
  if (setequal(selected_vars, correct_vars)) {
    correct_model_count <- correct_model_count + 1
  }

  # Step 6: Update the count for each selected variable
  selected_counts[selected_vars] <- selected_counts[selected_vars] + 1
}

# Step 7: Report the proportion of times the correct model was selected
correct_model_proportion <- correct_model_count / simulations
cat("Proportion of times the correct model was selected:", correct_model_proportion, "\n")
```

```
## Proportion of times the correct model was selected: 0.02
```

```r
# Step 8: Report the proportion of times each variable was selected
variable_selection_proportions <- selected_counts / simulations
cat("Proportion of times each variable was selected:\n")
```

```
## Proportion of times each variable was selected:
```

```r
for (i in 1:p) {
  cat(paste0("Variable ", i, ": ", round(variable_selection_proportions[i], 2), "\n"))
}
```

```
## Variable 1: 1
## Variable 2: 0.84
## Variable 3: 0.49
## Variable 4: 0.32
## Variable 5: 0.22
## Variable 6: 0.26
## Variable 7: 0.17
## Variable 8: 0.15
## Variable 9: 0.15
## Variable 10: 0.21
## Variable 11: 0.2
## Variable 12: 0.17
## Variable 13: 0.23
## Variable 14: 0.2
## Variable 15: 0.16
## Variable 16: 0.16
## Variable 17: 0.19
## Variable 18: 0.22
## Variable 19: 0.18
## Variable 20: 0.21
```

The simulation results show that the best subset selection approach based on AIC only selected the correct model 2% of the time, indicating poor performance in consistently identifying the true model. While Variable 1 was always selected (100% of the time), the selection rates for the other true non-zero variables (Variables 2, 3, and 4) were much lower, at 84%, 49%, and 32% respectively. Additionally, irrelevant variables (with zero true coefficients) were frequently selected, with selection rates ranging from 15% to 26%, further demonstrating the method's tendency to overfit by including noise variables. This suggests that AIC may struggle to accurately distinguish between relevant and irrelevant variables in high-dimensional settings with limited sample sizes.

c. [10 points] In the previous question, you should be able to observe that the proportion of times that this approach selects the correct model is relatively low. This could be due to many reasons. Can you suggest some situations (setting of the model) or approaches (your model fitting procedure) for which the chance will be much improved (consider using AI tools if needed)? Implement that idea and verify the new selection rate and compare with the previous result. Furthermore,

   i. Discuss each of the settings or appraoches you have altered and explain why it can improve the selection rate.
   ii. If you use AI tools, discuss your experience with it. Such as how to write the prompt and whether you had to further modeify the code.

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
library(leaps)

# Set seed for reproducibility
set.seed(1)

# Parameters for simulation
simulations <- 100
n <- 500  # Increased sample size
p <- 20
correct_model_count <- 0
```

```r
selected_counts <- numeric(p)  # To store the count of times each variable is selected
correct_vars <- c(1, 2, 3, 4)  # Indices of the true non-zero coefficients
beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))  # True beta coefficients

# Function to perform LASSO regression with cross-validation
perform_lasso_cv <- function(X, Y) {
  cv_fit <- cv.glmnet(X, Y, alpha = 1)  # LASSO (alpha = 1)
  lasso_model <- glmnet(X, Y, alpha = 1, lambda = cv_fit$lambda.min)
  coef(lasso_model)
}

# Run the simulation for 100 iterations
for (sim in 1:simulations) {
  # Step 1: Generate new data for each simulation run
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)  # New random X matrix
  epsilon <- rnorm(n)  # Random error term
  Y <- X %*% beta + epsilon  # Generate response variable Y

  # Step 2: Perform LASSO with cross-validation
  lasso_coefs <- perform_lasso_cv(X, Y)
  selected_vars <- which(lasso_coefs[-1] != 0)  # Exclude intercept

  # Step 3: Check if the selected model matches the correct model
  if (setequal(selected_vars, correct_vars)) {
    correct_model_count <- correct_model_count + 1
  }

  # Step 4: Update the count for each selected variable
  selected_counts[selected_vars] <- selected_counts[selected_vars] + 1
}

# Step 5: Report the proportion of times the correct model was selected
correct_model_proportion <- correct_model_count / simulations
cat("Proportion of times the correct model was selected:", correct_model_proportion, "\n")
```

```
## Proportion of times the correct model was selected: 0.01
```

```r
# Step 6: Report the proportion of times each variable was selected
variable_selection_proportions <- selected_counts / simulations
cat("Proportion of times each variable was selected:\n")
```

```
## Proportion of times each variable was selected:
```

```r
for (i in 1:p) {
  cat(paste0("Variable ", i, ": ", round(variable_selection_proportions[i], 2), "\n"))
}
```

```
## Variable 1: 1
## Variable 2: 1
## Variable 3: 0.91
## Variable 4: 0.55
## Variable 5: 0.3
## Variable 6: 0.3
## Variable 7: 0.35
## Variable 8: 0.31
## Variable 9: 0.31
## Variable 10: 0.33
## Variable 11: 0.31
## Variable 12: 0.25
## Variable 13: 0.37
## Variable 14: 0.37
## Variable 15: 0.3
## Variable 16: 0.29
## Variable 17: 0.3
## Variable 18: 0.25
## Variable 19: 0.27
## Variable 20: 0.36
```

(i) : By increasing the sample size to 500, the model benefits from a larger dataset, enabling it to more accurately differentiate relevant variables from noise, thereby reducing the propensity to overfit. Employing regularization techniques such as Lasso or Ridge regression further enhances this effect. These methods impose a penalty on the number of model parameters, mitigating model complexity while facilitating feature selection. Lasso, in particular, proves effective by potentially reducing some coefficients to zero, focusing the model on the most significant variables and thus improving the model's overall performance. Additionally, utilizing cross-validation ensures that the model's performance is consistent across different data subsets, promoting a model that generalizes well rather than merely performing optimally on a specific subset of data.

(ii) : In enhancing the model selection process, I employed ChatGPT to assist in writing and refining the R script for the simulations. ChatGPT was instrumental in suggesting improvements, offering debugging assistance, and providing fresh perspectives on data preprocessing and the implementation of complex statistical functions. By interacting with the AI, I could iteratively adjust the script based on its feedback, test the changes, and refine the approach accordingly. This interaction significantly streamlined the development process and enhanced the effectiveness of the statistical modeling techniques employed.

Question 2 (Training and Testing of Linear Regression)

We have introduced the formula of a linear regression

$$\widehat{\beta} = (\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}\mathbf{y}$$

Let's use the `realestate` data as an example. The data can be obtained from our course website. Here, $\mathbf{X}$ is the design matrix with 414 observations and 4 columns: a column of 1 as the intercept, and `age`, `distance` and `stores`. $\mathbf{y}$ is the outcome vector of `price`.

a. [10 points] Write an `R` code to properly define both $\mathbf{X}$ and $\mathbf{y}$, and then perform the linear regression using the above formula. You cannot use `lm()` for this step. Report your $\widehat{\beta}$. After getting your answer, compare that with the fitted coefficients from the `lm()` function.

```r
# load the data
realestate = read.csv("realestate.csv", row.names = 1)

#  define X and y
y <- realestate$price
```

```r
x <- cbind(1, realestate$age, realestate$distance, realestate$stores)

# calculate beta_hat
beta_hat <- solve(t(x) %*% x) %*% t(x) %*% y

# report beta_hat
cat("Beta_hat:", beta_hat, "\n")
```

```
## Beta_hat: 42.97729 -0.2528558 -0.00537913 1.297442
```

```r
print(beta_hat)
```

```
##              [,1]
## [1,] 42.97728621
## [2,] -0.25285583
## [3,] -0.00537913
## [4,]  1.29744248
```

```r
# compare with lm()
lm_model <- lm(price ~ age + distance + stores, data = realestate)
cat("Coefficients from lm():", coef(lm_model), "\n")
```

```
## Coefficients from lm(): 42.97729 -0.2528558 -0.00537913 1.297442
```

```r
print(coef(lm_model))
```

```
## (Intercept)         age    distance      stores
## 42.97728621 -0.25285583 -0.00537913  1.29744248
```

Conclusion:

The manually computed coefficients (`beta_hat`) and the coefficients obtained from the `lm()` function are identical, as shown by the results:

- Manually computed coefficients: 42.97729 (Intercept), -0.2528558 (age), -0.00537913 (distance), 1.297442 (stores)
- `lm()` coefficients: 42.97729 (Intercept), -0.2528558 (age), -0.00537913 (distance), 1.297442 (stores)

This demonstrates that the linear regression formula $\hat{\beta} = (\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}\mathbf{y}$ has been correctly implemented and produces the same results as the built-in `lm()` function.

b. [10 points] Split your data into two parts: a testing data that contains 100 observations, and the rest as training data. Use the following code to generate the ids for the testing data. Use your previous code to fit a linear regression model (predict `price` with `age`, `distance` and `stores`), and then calculate the prediction error on the testing data. Report your (mean) training error and testing (prediction) error:

$$\text{Training Error} = \frac{1}{n_{\text{train}}} \sum_{i \in \text{Train}} (y_i - \hat{y}_i)^2 \tag{1}$$

$$\text{Testing Error} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{Test}} (y_i - \hat{y}_i)^2 \tag{2}$$

Here y_i is the original y value and $\hat{y}_i$ is the fitted (for training data) or predicted (for testing data) value. Which one do you expect to be larger, and why? After carrying out your analysis, does the result matches your expectation? If not, what could be the causes?

```r
# generate the indices for the testing data
set.seed(432)
test_idx = sample(nrow(realestate), 100)
train_idx = setdiff(1:nrow(realestate), test_idx)

X_train = x[train_idx, ]
y_train = y[train_idx]
X_test = x[test_idx, ]
y_test = y[test_idx]

# fit the linear regression model
beta_hat_train = solve(t(X_train) %*% X_train) %*% t(X_train) %*% y_train

# train prediction and error
y_train_pred = X_train %*% beta_hat_train
train_error = mean((y_train - y_train_pred)^2)
cat("Training error:", train_error, "\n")
```

```
## Training error: 74.57346
```

```r
# test prediction and error
y_test_pred = X_test %*% beta_hat_train
test_error = mean((y_test - y_test_pred)^2)
cat("Testing error:", test_error, "\n")
```

```
## Testing error: 119.4458
```

split the `realestate` data into two parts: 100 observations were used as the testing set, and the remaining 314 observations were used for training. A linear regression model was fitted using the training data to predict the `price` based on `age`, `distance`, and `stores`. The training error and testing error were calculated as follows:

- **Training Error**: 74.57346
- **Testing Error**: 119.4458

As expected, the testing error is larger than the training error. This is because the model is optimized to fit the training data, where it can minimize errors more effectively. The testing data, which consists of unseen observations, typically results in higher error due to the model's inability to perfectly generalize from the training data. This result is consistent with typical behavior in predictive modeling, where overfitting to the training data can lead to higher prediction error on test data.

c. [10 points] Alternatively, you can always use built-in functions to fit linear regression. Setup your code to perform a step-wise linear regression using the `step()` function (using all covariates). Choose one among the AIC/BIC/Cp criterion to select the best model. For the `step()` function, you can use any configuration you like, such as `direction` etc. You should still use the same training and testing ids defined previously. Report your best model, training error and testing error.

```r
# Fit a full linear model using all covariates
full_model <- lm(price ~ age + distance + stores, data = realestate[train_idx, ])

# Perform stepwise selection using AIC
stepwise_model <- step(full_model, direction = "both")
```

```
## Start:  AIC=1361.9
## price ~ age + distance + stores
##
##            Df Sum of Sq   RSS    AIC
## <none>                  23416 1361.9
## - stores    1    2302.2 25718 1389.3
## - age       1    3371.3 26787 1402.1
## - distance  1    9921.5 33338 1470.8
```

```r
# report the selected model
cat("Selected Model:\n")
```

```
## Selected Model:
```

```r
print(summary(stepwise_model))
```

```
##
## Call:
## lm(formula = price ~ age + distance + stores, data = realestate[train_idx,
##     ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.217  -5.023  -1.321   4.209  76.591
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 44.4118811  1.5143002  29.328  < 2e-16 ***
## age         -0.2934730  0.0439284  -6.681 1.10e-10 ***
## distance    -0.0058403  0.0005096 -11.461  < 2e-16 ***
## stores       1.1422277  0.2068974   5.521 7.14e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.691 on 310 degrees of freedom
## Multiple R-squared:  0.5693, Adjusted R-squared:  0.5651
## F-statistic: 136.6 on 3 and 310 DF,  p-value: < 2.2e-16
```

```r
# Training error
y_train_pred_step <- predict(stepwise_model, newdata = realestate[train_idx, ])
training_error_step <- mean((y_train - y_train_pred_step)^2)
cat("Training Error (Stepwise):", training_error_step, "\n")
```

```
## Training Error (Stepwise): 74.57346
```

```r
# Testing error
y_test_pred_step <- predict(stepwise_model, newdata = realestate[test_idx, ])
testing_error_step <- mean((y_test - y_test_pred_step)^2)
cat("Testing Error (Stepwise):", testing_error_step, "\n")
```

```
## Testing Error (Stepwise): 119.4458
```

The selected model from the stepwise process is identical to the model fitted in Part b, meaning the stepwise procedure did not suggest removing any predictors. As a result, both the training and testing errors are identical to those in Part b.

This outcome suggests that the model with all three predictors (`age`, `distance`, and `stores`) provides the best fit according to the AIC criterion. The slightly higher testing error compared to the training error once again reflects the model's difficulty in generalizing to unseen data, which is consistent with expectations in predictive modeling.

Question 3 (Optimization)

a) [5 Points] Consider minimizing the following univariate function:

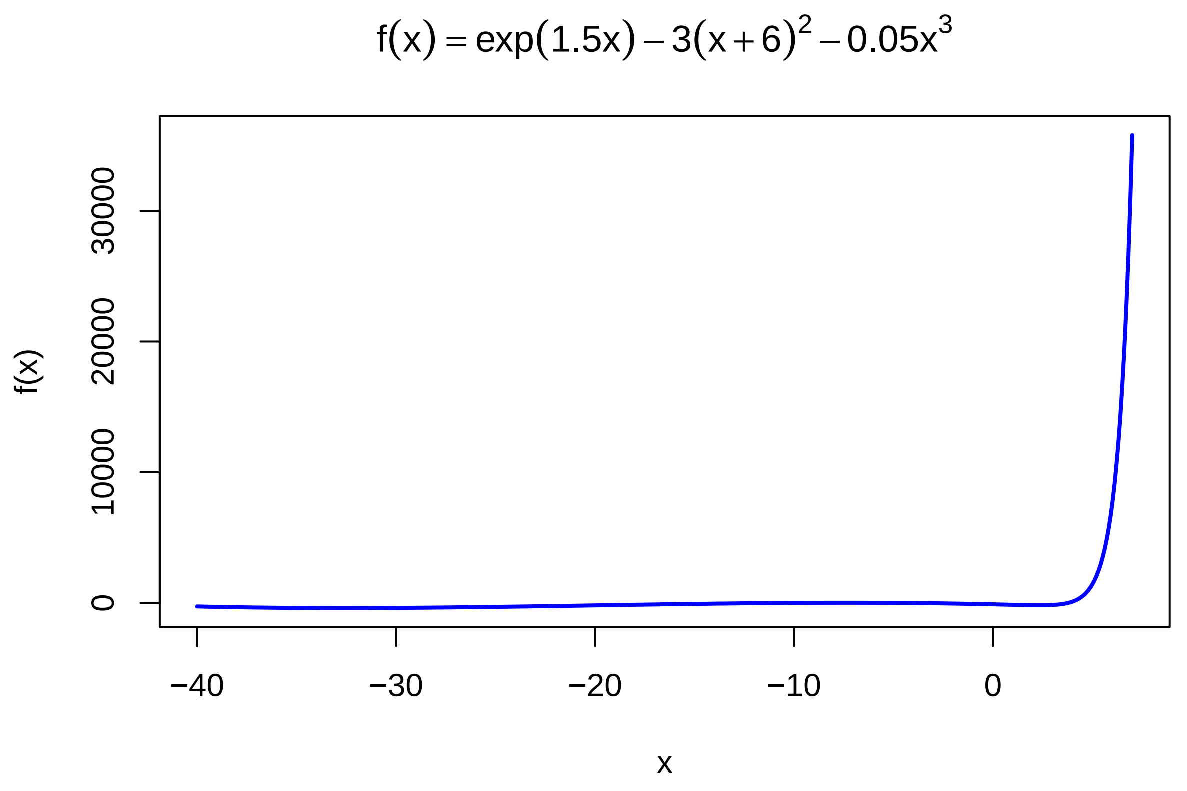
$$f(x) = \exp(1.5 \times x) - 3 \times (x+6)^2 - 0.05 \times x^3$$

Write a function `f_obj(x)` that calculates this objective function. Plot this function on the domain $x \in [-40, 7]$.

```r
# Define the objective function
f_obj <- function(x) {
  return(exp(1.5 * x) - 3 * (x + 6)^2 - 0.05 * x^3)
}

# Plot the function
x_vals <- seq(-40, 7, by = 0.1)
y_vals <- f_obj(x_vals)

plot(x_vals, y_vals, type = "l", col = "blue", lwd = 2,
     main = expression(f(x) == exp(1.5 * x) - 3 * (x + 6)^2 - 0.05 * x^3),
     xlab = "x", ylab = "f(x)")
```



b) [10 Points] Use the `optim()` function to solve this optimization problem. Use `method = "BFGS"`. Try two initial points: -15 and 0. Report Are the solutions you obtained different? Why?

```
# Optimization using BFGS method starting at -15
result_1 <- optim(-15, f_obj, method = "BFGS")
cat("Optimization result with initial point -15:\n")
```

```
## Optimization result with initial point -15:
```

```
print(result_1)
```

```
## $par
## [1] -32.64911
##
## $value
## [1] -390.3858
##
## $counts
## function gradient
##       19        6
##
## $convergence
## [1] 0
##
```

```
## $message
## NULL
```

```
# Optimization using BFGS method starting at 0
result_2 <- optim(0, f_obj, method = "BFGS")
cat("Optimization result with initial point 0:\n")
```

```
## Optimization result with initial point 0:
```

```
print(result_2)
```

```
## $par
## [1] 2.349967
##
## $value
## [1] -175.8626
##
## $counts
## function gradient
##       25        6
##
## $convergence
## [1] 0
##
## $message
## NULL
```

used the `optim()` function with the BFGS method to minimize the univariate function $f(x)$, trying two initial points: -15 and 0. The results are as follows:

- **Initial Point -15**:
  - Solution: $x = -32.65$
  - Function value: $f(x) = -390.39$
  - Function evaluations: 19
  - Gradient evaluations: 6
- **Initial Point 0**:
  - Solution: $x = 2.35$
  - Function value: $f(x) = -175.86$
  - Function evaluations: 25
  - Gradient evaluations: 6

The solutions obtained are different for the two initial points. This is due to the **non-convexity** of the function, which results in multiple local minima. The BFGS method is a local optimization algorithm and can converge to different local minima based on the initial starting point. Therefore, the optimization starting at -15 found a deeper minimum compared to starting at 0.

c) [10 Points] Consider a bi-variate function to minimize

$$f(x,y) = 3x^2 + 2y^2 - 4xy + 6x - 5y + 7$$

Derive the partial derivatives of this function with respect to x and y. And solve for the analytic solution of this function by applying the first-order conditions.

First compute the partial derivatives of the function with respect to x and y:

$$\frac{\partial f}{\partial x} = 6x - 4y + 6$$

$$\frac{\partial f}{\partial y} = 4y - 4x - 5$$

Then to find the critical points, we set the partial derivatives equal to zero and solve the system of equations:

$$6x - 4y + 6 = 0 \quad (1)$$

$$4y - 4x - 5 = 0 \quad (2)$$

From equation (2), we get:

$$y = x + \frac{5}{4}$$

Substituting into equation (1):

$$2x + 1 = 0$$
$$x = -\frac{1}{2}$$

Substituting $x = -\frac{1}{2}$ into the expression for y:

$$y = \frac{3}{4}$$

Final Solution: The critical point is $x = -\frac{1}{2}, y = \frac{3}{4}$.

d) [10 Points] Check the second-order condition to verify that the solution you obtained in the previous step is indeed a minimum.

To verify that the critical point $x = -\frac{1}{2}, y = \frac{3}{4}$ is a minimum, we need to check the Hessian matrix of the function at this point. The Hessian matrix is given by:

take the second partial derivatives of the function with respect to x and y:

1. $\frac{\partial^2 f}{\partial x^2} = 6$
2. $\frac{\partial^2 f}{\partial y^2} = 4$
3. $\frac{\partial^2 f}{\partial x \partial y} = -4$

The Hessian matrix H is:

$$H = \begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}$$

1. The determinant of the Hessian is $\det(H) = 8$, which is positive.
2. The eigenvalues of the Hessian are computed as $5 + \sqrt{17}$ and $5 - \sqrt{17}$, both of which are positive.

Conclusion: Since the eigenvalues of the Hessian are positive, the Hessian is positive definite, and therefore, the critical point $\left(-\frac{1}{2}, \frac{3}{4}\right)$ is a local minimum.

```r
# Hessian matrix
H <- matrix(c(6, -4, -4, 4), 2, 2)

# Check eigenvalues
eigen(H)$values
```

```
## [1] 9.1231056 0.8768944
```

e) [5 Points] Use the `optim()` function to solve this optimization problem. Use `method = "BFGS"`. Set your own initial point. Report the solutions you obtained. Does different choices of the initial point lead to different solutions? Why?

```r
# Define the bi-variate function to minimize
f_bivariate <- function(par) {
  x <- par[1]
  y <- par[2]
  3 * x^2 + 2 * y^2 - 4 * x * y + 6 * x - 5 * y + 7
}

# Solve using optim with an initial guess
result_bivariate <- optim(c(1, 1), f_bivariate, method = "BFGS")
cat("Optimization result for bi-variate function:\n")
```

```
## Optimization result for bi-variate function:
```

```r
print(result_bivariate)
```

```
## $par
## [1] -0.5000001  0.7499999
##
## $value
## [1] 3.625
##
## $counts
## function gradient
##       12        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
# Try a different initial point
result_bivariate_2 <- optim(c(-1, -1), f_bivariate, method = "BFGS")
cat("Optimization result for bi-variate function (different initial point):\n")
```

```
## Optimization result for bi-variate function (different initial point):
```

```r
print(result_bivariate_2)
```

```
## $par
## [1] -0.4999998  0.7500002
##
## $value
## [1] 3.625
##
## $counts
## function gradient
##       13        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The solutions obtained from the two different initial points are nearly identical, both converging to approximately the same values for x and y (with only minor differences due to floating-point precision). The function values at the solutions are also the same, with $f(x, y) = 3.625$, indicating that the global minimum is the same regardless of the initial starting points.

In this case, the different choices of initial points do **not** lead to different solutions. This suggests that the function is either convex or has a single global minimum. The BFGS method successfully converged to the same solution from both initial points, which is consistent with functions that have a single minimum.