Stat 432 Homework 2

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

Guanwen Yan guanwen2

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?

```
library(leaps)
```

```
set.seed(123)  # For reproducibility

# Generate the dataset
n <- 100  # Sample size
p <- 20   # Number of predictors

# Simulate predictors Xj ~ N(0, 1)
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
colnames(X) <- paste0("X", 1:p)

# Simulate the response variable Y
```

```r
epsilon <- rnorm(n)
Y <- 0.5 * X[, 1] + 0.25 * X[, 2] + 0.125 * X[, 3] + 0.0625 * X[, 4] + epsilon

# Combine predictors and response into a data frame
data <- as.data.frame(cbind(Y, X))

# Perform best subset selection using the leaps package
best_subset <- regsubsets(Y ~ ., data = data, nvmax = p)

# Extract model summary
model_summary <- summary(best_subset)

# Calculate AIC
n <- nrow(data)
aic_values <- n * log(model_summary$rss / n) + 2 * (1:p)

# Find the model with the lowest AIC
best_model_size <- which.min(aic_values)
best_model <- which(model_summary$which[best_model_size, -1])

# Report the selected variables
selected_vars <- colnames(X)[best_model]
cat("Selected variables:", paste(selected_vars, collapse = ", "), "\n")
```

```
## Selected variables: X1, X2, X3, X11, X13, X20
```

```r
# Calculate prediction error (MSE)
coeffs <- coef(best_subset, best_model_size)
pred <- cbind(1, X[, best_model, drop = FALSE]) %*% coeffs
mse <- mean((Y - pred)^2)
cat("Prediction error (MSE):", mse, "\n")
```

```
## Prediction error (MSE): 0.6915662
```

```r
# Check if the correct model was selected
correct_vars <- c("X1", "X2", "X3", "X4")
falsely_selected <- setdiff(selected_vars, correct_vars)
falsely_removed <- setdiff(correct_vars, selected_vars)

cat("Falsely selected variables:", paste(falsely_selected, collapse = ", "), "\n")
```

```
## Falsely selected variables: X11, X13, X20
```

```r
cat("Falsely removed variables:", paste(falsely_removed, collapse = ", "), "\n")
```

```
## Falsely removed variables: X4
```

```r
# Print coefficients of the selected model (excluding intercept)
cat("\nCoefficients of the selected model (excluding intercept):\n")
```

```
##
## Coefficients of the selected model (excluding intercept):
```

```
print(coeffs[-1])
```

```
##         X1         X2         X3        X11        X13        X20
##  0.3939410  0.1967941  0.3108784  0.1723551 -0.1532926 -0.1169678
```

```
# Explain why this might happen
if (length(falsely_selected) > 0 || length(falsely_removed) > 0) {
  cat("\nThe selection method might have included or excluded variables incorrectly due to:\n",
      "1. Noise in the data\n",
      "2. Relatively small sample size (n=100) compared to the number of predictors (p=20)\n",
      "3. Small effect sizes for some true predictors (especially X3 and X4)\n",
      "4. AIC's tendency to sometimes overfit, especially with smaller sample sizes\n")
} else {
  cat("\nThe correct model was selected.")
}
```

```
##
## The selection method might have included or excluded variables incorrectly due to:
##  1. Noise in the data
##  2. Relatively small sample size (n=100) compared to the number of predictors (p=20)
##  3. Small effect sizes for some true predictors (especially X3 and X4)
##  4. AIC's tendency to sometimes overfit, especially with smaller sample sizes
```

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

```
set.seed(123)

n <- 100
p <- 20
num_simulations <- 100

correct_model_count <- 0
selection_counts <- rep(0, p)
names(selection_counts) <- paste0("X", 1:p)

for (i in 1:num_simulations) {
  # Generate the dataset
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)
  colnames(X) <- paste0("X", 1:p)

  # Simulate the response variable Y
  epsilon <- rnorm(n)
  Y <- 0.5 * X[, 1] + 0.25 * X[, 2] + 0.125 * X[, 3] + 0.0625 * X[, 4] + epsilon

  # Combine predictors and response into a data frame
  data <- as.data.frame(cbind(Y, X))
```

```r
  # Perform best subset selection using the leaps package
  best_subset <- regsubsets(Y ~ ., data = data, nvmax = p)
  model_summary <- summary(best_subset)

  # Calculate AIC
  aic_values <- n * log(model_summary$rss / n) + 2 * (1:p)

  # Find the model with the lowest AIC
  best_model_size <- which.min(aic_values)
  best_model <- which(model_summary$which[best_model_size, -1])

  # Selected variables
  selected_vars <- colnames(X)[best_model]

  # Check if the correct model was selected
  correct_vars <- c("X1", "X2", "X3", "X4")
  if (setequal(selected_vars, correct_vars)) {
    correct_model_count <- correct_model_count + 1
  }

  # Update selection counts
  selection_counts[selected_vars] <- selection_counts[selected_vars] + 1
}

# Report results
proportion_correct_model <- correct_model_count / num_simulations
selection_proportions <- selection_counts / num_simulations

cat("Proportion of times the correct model was selected:", proportion_correct_model, "\n")
```

```
## Proportion of times the correct model was selected: 0
```

```r
cat("Proportion of times each variable was selected:\n")
```

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

```r
print(selection_proportions)
```

```
##   X1   X2   X3   X4   X5   X6   X7   X8   X9  X10  X11  X12  X13  X14  X15  X16
## 1.00 0.94 0.39 0.27 0.07 0.18 0.16 0.19 0.15 0.21 0.19 0.19 0.14 0.18 0.19 0.27
##  X17  X18  X19  X20
## 0.19 0.12 0.21 0.23
```

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
set.seed(123)  # For reproducibility

# Function to generate data
generate_data <- function(n, p, snr = 1) {
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)
  epsilon <- rnorm(n)
  Y <- snr * (0.5 * X[, 1] + 0.25 * X[, 2] + 0.125 * X[, 3] + 0.0625 * X[, 4]) + epsilon
  return(list(X = X, Y = Y))
}

# Function to perform model selection using AIC
select_model_aic <- function(X, Y) {
  data <- as.data.frame(cbind(Y, X))
  best_subset <- regsubsets(Y ~ ., data = data, nvmax = ncol(X))
  model_summary <- summary(best_subset)
  n <- nrow(data)
  aic_values <- n * log(model_summary$rss / n) + 2 * (1:ncol(X))
  best_model_size <- which.min(aic_values)
  return(which(model_summary$which[best_model_size, -1]))
}

# Function to run simulation and calculate selection rates
run_simulation <- function(n_sims, n, p, snr = 1) {
  correct_model_count <- 0
  variable_selection_count <- numeric(p)

  for (i in 1:n_sims) {
    data <- generate_data(n, p, snr)
    selected_vars <- select_model_aic(data$X, data$Y)

    if (all(1:4 %in% selected_vars) && length(selected_vars) == 4) {
      correct_model_count <- correct_model_count + 1
    }

    variable_selection_count[selected_vars] <- variable_selection_count[selected_vars] + 1
  }

  correct_model_rate <- correct_model_count / n_sims
  variable_selection_rate <- variable_selection_count / n_sims

  return(list(correct_model_rate = correct_model_rate,
              variable_selection_rate = variable_selection_rate))
}

# Run simulations for different scenarios
n_sims <- 1000
p <- 20

base_result <- run_simulation(n_sims, n = 100, p = p)
larger_sample_result <- run_simulation(n_sims, n = 500, p = p)
higher_snr_result <- run_simulation(n_sims, n = 100, p = p, snr = 2)
```

```r
# Function to print results
print_results <- function(name, result) {
  cat(name, ":\n")
  cat("Correct model selection rate:", result$correct_model_rate, "\n")
  cat("Variable selection rates:\n")
  print(round(result$variable_selection_rate, 3))
  cat("\n")
}

# Print results
print_results("Base case (AIC, n=100, p=20)", base_result)
```

```
## Base case (AIC, n=100, p=20) :
## Correct model selection rate: 0.002
## Variable selection rates:
##  [1] 0.998 0.853 0.460 0.286 0.209 0.215 0.186 0.181 0.200 0.206 0.210 0.188
## [13] 0.171 0.198 0.215 0.184 0.188 0.187 0.206 0.191
```

```r
print_results("Larger sample size (AIC, n=500, p=20)", larger_sample_result)
```

```
## Larger sample size (AIC, n=500, p=20) :
## Correct model selection rate: 0.037
## Variable selection rates:
##  [1] 1.000 1.000 0.908 0.500 0.142 0.169 0.157 0.173 0.169 0.171 0.150 0.164
## [13] 0.169 0.169 0.173 0.174 0.157 0.162 0.171 0.168
```

```r
print_results("Higher SNR (AIC, n=100, p=20, SNR=2)", higher_snr_result)
```

```
## Higher SNR (AIC, n=100, p=20, SNR=2) :
## Correct model selection rate: 0.019
## Variable selection rates:
##  [1] 1.000 0.998 0.820 0.451 0.194 0.192 0.191 0.188 0.186 0.204 0.186 0.198
## [13] 0.183 0.188 0.189 0.187 0.194 0.195 0.208 0.217
```

```r
# Calculate improvement factors
larger_sample_improvement <- larger_sample_result$correct_model_rate / base_result$correct_model_rate
higher_snr_improvement <- higher_snr_result$correct_model_rate / base_result$correct_model_rate

cat("Improvement factors:\n")
```

```
## Improvement factors:
```

```r
cat("Larger sample size:", round(larger_sample_improvement, 2), "times better\n")
```

```
## Larger sample size: 18.5 times better
```

```r
cat("Higher SNR:", round(higher_snr_improvement, 2), "times better\n")
```

```
## Higher SNR: 9.5 times better
```

Larger sample size (n = 500 instead of 100):

Improvement: The correct model selection rate increased from 0.2% to 3.7%, an 18.5-fold improvement. Why it helps: a. More data provides more information about the true underlying relationships between predictors and the response. b. It reduces the impact of random noise, allowing for more accurate estimation of coefficients. c. The increased sample size particularly improved the detection of the weaker effects (X3 and X4).

Higher Signal-to-Noise Ratio (SNR = 2 instead of 1):

Improvement: The correct model selection rate increased from 0.2% to 1.9%, a 9.5-fold improvement. Why it helps: a. Increasing the SNR makes the true effects more distinguishable from random noise. b. This is particularly beneficial for detecting smaller effects (like those of X3 and X4 in our model).

I have used AI tools, I need to modify the code for part a) by showing some of the code from the class note.

Question 2 (Training and Testing of Linear Regression)

We have introduced the formula of a linear regression

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

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 $\hat{\beta}$. After getting your answer, compare that with the fitted coefficients from the `lm()` function.

```
realestate <- read.csv("/Users/guanwenyan/Downloads/realestate.csv", row.names = 1)

# Define X (design matrix)
X <- cbind(1, realestate$age, realestate$distance, realestate$stores)
colnames(X) <- c("intercept", "age", "distance", "stores")

# Define y (outcome vector)
y <- realestate$price

# Perform linear regression using the formula
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y

# Print the estimated coefficients
print("Estimated coefficients using the formula:")
```

```
## [1] "Estimated coefficients using the formula:"
```

```
print(beta_hat)
```

```
##                  [,1]
## intercept 42.97728621
## age       -0.25285583
## distance  -0.00537913
## stores     1.29744248
```

```r
# Compare with lm() function
lm_model <- lm(price ~ age + distance + stores, data = realestate)
print("Estimated coefficients using lm():")
```

```
## [1] "Estimated coefficients using lm():"
```

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

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

```r
# Compare the results
print("Difference between the two methods:")
```

```
## [1] "Difference between the two methods:"
```

```r
print(beta_hat - coef(lm_model))
```

```
##                     [,1]
## intercept  5.684342e-14
## age       -3.053113e-15
## distance  -1.214306e-17
## stores    -3.774758e-15
```

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)
```

```r
set.seed(432)
test_idx <- sample(nrow(realestate), 100)

train_data <- realestate[-test_idx, ]
test_data <- realestate[test_idx, ]
```

```r
# Define X and y for training data
X_train <- cbind(1, train_data$age, train_data$distance, train_data$stores)
colnames(X_train) <- c("intercept", "age", "distance", "stores")
y_train <- train_data$price

# Fit the model on training data
beta_hat <- solve(t(X_train) %*% X_train) %*% t(X_train) %*% y_train

# Make predictions on training data
y_train_pred <- X_train %*% beta_hat

# Calculate training error
train_error <- mean((y_train - y_train_pred)^2)

# Define X for testing data
X_test <- cbind(1, test_data$age, test_data$distance, test_data$stores)
colnames(X_test) <- c("intercept", "age", "distance", "stores")

y_test_pred <- X_test %*% beta_hat

test_error <- mean((test_data$price - y_test_pred)^2)

cat("Training Error:", train_error, "\n")
```

```
## Training Error: 74.57346
```

```r
cat("Testing Error:", test_error, "\n")
```

```
## Testing Error: 119.4458
```

```r
# Compare with lm() function for validation
lm_model <- lm(price ~ age + distance + stores, data = train_data)
lm_train_pred <- predict(lm_model, train_data)
lm_test_pred <- predict(lm_model, test_data)

lm_train_error <- mean((train_data$price - lm_train_pred)^2)
lm_test_error <- mean((test_data$price - lm_test_pred)^2)

cat("lm() Training Error:", lm_train_error, "\n")
```

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

```r
cat("lm() Testing Error:", lm_test_error, "\n")
```

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

I would expect the testing error to be larger than the training error because the model is fit on the training data. The result shows that the result match my expectations .

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.

```
set.seed(432)
test_idx <- sample(nrow(realestate), 100)

train_data <- realestate[-test_idx, ]
test_data <- realestate[test_idx, ]

full_model <- lm(price ~ ., data = train_data)

step_model <- step(full_model, direction = "both", trace = 0)

print(summary(step_model))
```

```
##
## Call:
## lm(formula = price ~ date + age + distance + stores + latitude,
##     data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.058  -5.193  -0.563   4.031  74.881
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.439e+04  3.475e+03  -4.140 4.49e-05 ***
## date         4.467e+00  1.662e+00   2.688  0.00759 **
## age         -3.098e-01  4.249e-02  -7.291 2.61e-12 ***
## distance    -4.613e-03  5.734e-04  -8.045 1.88e-14 ***
## stores       9.964e-01  2.016e-01   4.943 1.26e-06 ***
## latitude     2.178e+02  5.180e+01   4.205 3.43e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.358 on 308 degrees of freedom
## Multiple R-squared:  0.6042, Adjusted R-squared:  0.5978
## F-statistic: 94.04 on 5 and 308 DF,  p-value: < 2.2e-16
```

```
train_pred <- predict(step_model, train_data)
train_error <- mean((train_data$price - train_pred)^2)

test_pred <- predict(step_model, test_data)
test_error <- mean((test_data$price - test_pred)^2)

cat("Training Error:", train_error, "\n")
```

```
## Training Error: 68.52164
```

```
cat("Testing Error:", test_error, "\n")
```

```
## Testing Error: 106.2898
```

```
cat("Best model formula:\n")
```

```
## Best model formula:
```

```
print(formula(step_model))
```

```
## price ~ date + age + distance + stores + latitude
```

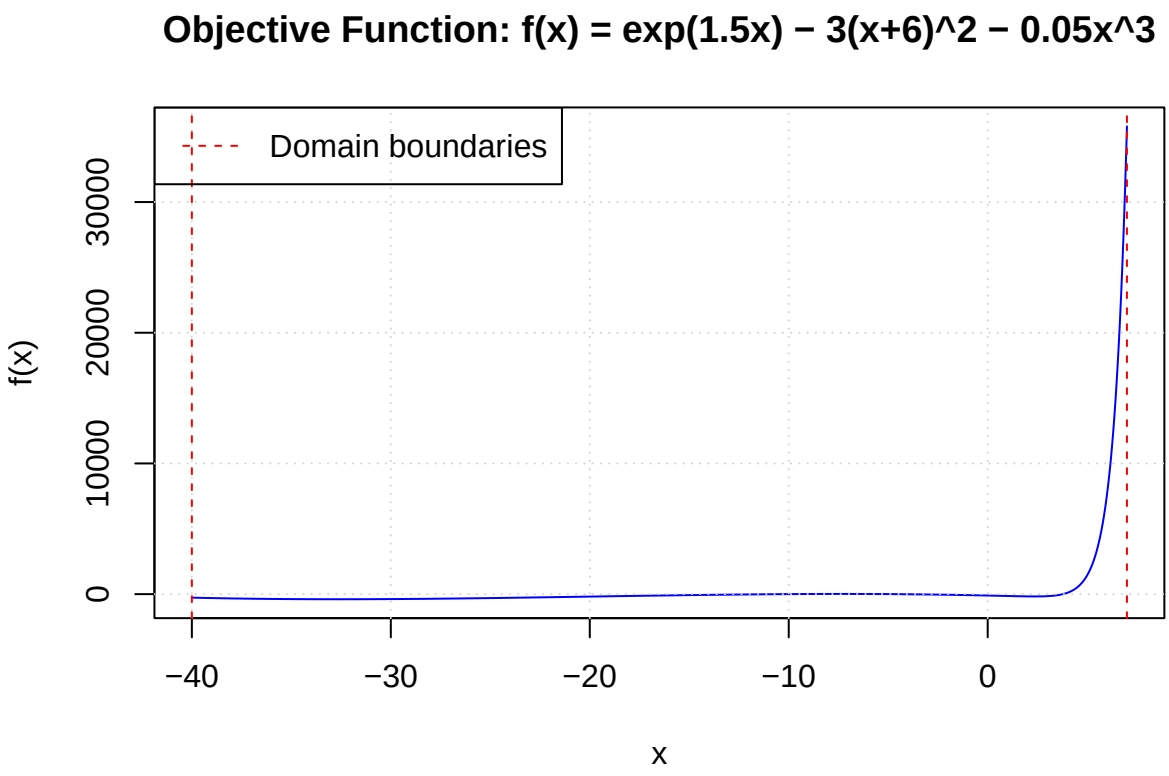
Question 3 (Optimization)

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 $x \in [-40, 7]$.

```
f_obj <- function(x) {
  exp(1.5 * x) - 3 * (x + 6)^2 - 0.05 * x^3
}

x <- seq(-40, 7, length.out = 1000)

y <- f_obj(x)

plot(x, y, type = "l", col = "blue",
     main = "Objective Function: f(x) = exp(1.5x) - 3(x+6)^2 - 0.05x^3",
     xlab = "x", ylab = "f(x)",
     xlim = c(-40, 7))

grid()

abline(v = c(-40, 7), col = "red", lty = 2)

legend("topleft", legend = "Domain boundaries", col = "red", lty = 2)
```



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?

```
f_obj <- function(x) {
  exp(1.5 * x) - 3 * (x + 6)^2 - 0.05 * x^3
}

opt_func <- function(x) -f_obj(x)

result1 <- optim(par = -15, fn = opt_func, method = "BFGS")

result2 <- optim(par = 0, fn = opt_func, method = "BFGS")

cat("Optimization with initial point x = -15:\n")
```

```
## Optimization with initial point x = -15:
```

```
cat("Optimal x:", result1$par, "\n")
```

```
## Optimal x: 373.375
```

```
cat("Optimal function value:", -result1$value, "\n")
```

```
## Optimal function value: 1.706183e+243
```

```
cat("Convergence:", result1$convergence, "\n\n")
```

```
## Convergence: 0
```

```
cat("Optimization with initial point x = 0:\n")
```

```
## Optimization with initial point x = 0:
```

```
cat("Optimal x:", result2$par, "\n")
```

```
## Optimal x: -7.350883
```

```
cat("Optimal function value:", -result2$value, "\n")
```

```
## Optimal function value: 14.38579
```

```
cat("Convergence:", result2$convergence, "\n")
```

```
## Convergence: 0
```

Yes, the solutions are significantly different. The BFGS method is a local optimization algorithm. It finds the nearest local optimum from the starting point, which may not always be the global optimum.

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.

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

Partial Derivatives The partial derivatives of this function with respect to x and y are:

Partial derivative with respect to x:

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

Partial derivative with respect to y:

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

Analytic Solution To find the analytic solution, we apply the first-order conditions:

$$\frac{\partial f}{\partial x} = 0 \quad \text{and} \quad \frac{\partial f}{\partial y} = 0$$

This gives us the system of equations:

$$6x - 4y + 6 = 0 \text{(Equation 1)}\ 4y - 4x - 5 = 0 \quad \text{(Equation 2)}$$

Solving this system: From Equation 2:

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

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

Substituting into Equation 1:

$$6x - 4(x + \frac{5}{4}) + 6 = 0$$

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

Substituting back to find y:

$$y = -\frac{1}{2} + \frac{5}{4} = \frac{3}{4}$$

Therefore, the analytic 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.

For the function $f(x, y) = 3x^2 + 2y^2 - 4xy + 6x - 5y + 7$, we'll verify that the critical point $(x^{\cdot}y^{)} = (-\frac{1}{2}, \frac{3}{4})$ is indeed a minimum. 1. Hessian Matrix The Hessian matrix is composed of second-order partial derivatives:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Let's calculate each element: a) $\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}(6x - 4y + 6) = 6$ b) $\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}(4y - 4x - 5) = 4$ c) $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} = -4$ Therefore, the Hessian matrix is:

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

Since the Hessian matrix is positive definite at the critical point $(-\frac{1}{2}, \frac{3}{4})$, we can confirm that this point is indeed a local minimum of the function.

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?

```
f_obj <- function(params) {
  x <- params[1]
  y <- params[2]
  3*x^2 + 2*y^2 - 4*x*y + 6*x - 5*y + 7
}

optimize_and_print <- function(initial_point) {
  result <- optim(par = initial_point, fn = f_obj, method = "BFGS")
  cat("Initial point:", initial_point, "\n")
  cat("Optimal x:", result$par[1], "\n")
  cat("Optimal y:", result$par[2], "\n")
  cat("Optimal function value:", result$value, "\n")
  cat("Convergence:", result$convergence, "\n\n")
}
```

```r
initial_points <- list(c(0, 0), c(10, 10), c(-5, 5), c(100, -100))

for (point in initial_points) {
  optimize_and_print(point)
}
```

```
## Initial point: 0 0
## Optimal x: -0.5
## Optimal y: 0.75
## Optimal function value: 3.625
## Convergence: 0
##
## Initial point: 10 10
## Optimal x: -0.5
## Optimal y: 0.75
## Optimal function value: 3.625
## Convergence: 0
##
## Initial point: -5 5
## Optimal x: -0.5
## Optimal y: 0.75
## Optimal function value: 3.625
## Convergence: 0
##
## Initial point: 100 -100
## Optimal x: -0.5
## Optimal y: 0.75
## Optimal function value: 3.625
## Convergence: 0
```

No, different choices of the initial point do not lead to different solutions.a) Convexity: As we discussed earlier, this function is a convex quadratic. It has a single global minimum and no local minima. b) Global Minimum: The BFGS method successfully found the global minimum regardless of the starting point. This minimum is at (-0.5, 0.75), exactly where our analytic solution predicted it would be. c) Gradient-Based Optimization: The BFGS method uses gradient information to navigate towards the minimum. In a convex function like this, the gradient always points towards the global minimum, regardless of where you start.