HW02_zilinw3

Zilin Wang (zilinw3)

2024-09-10

Contents

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).

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?

Answer:

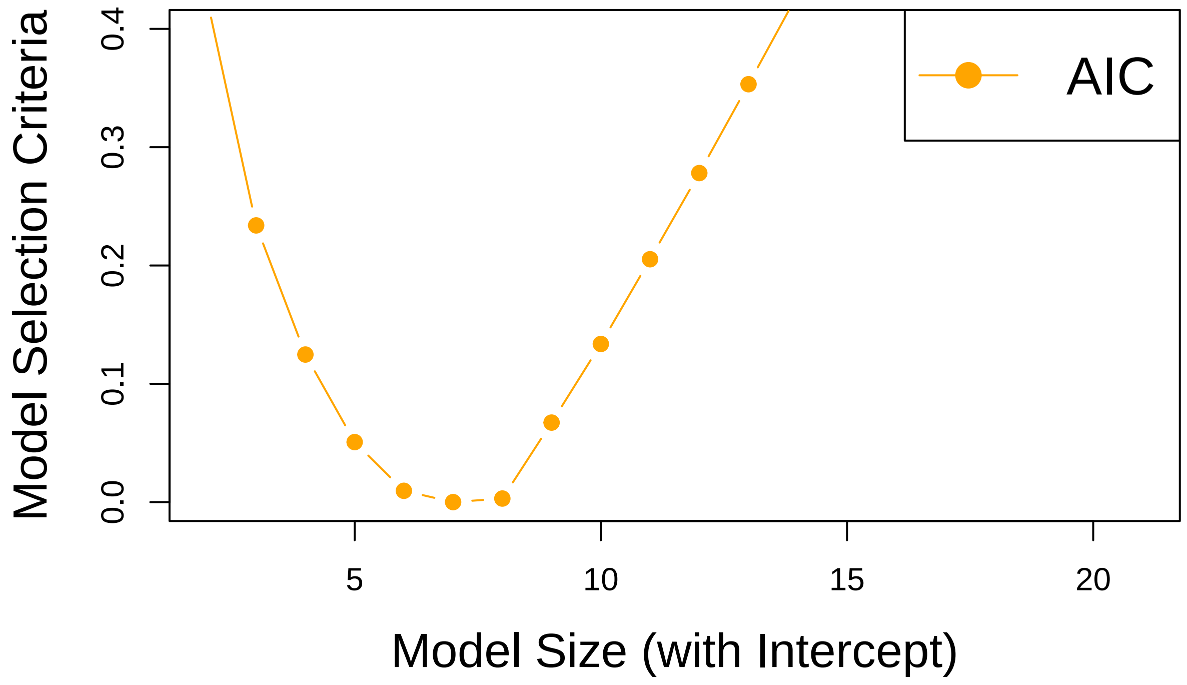
```r
library(leaps)
set.seed(123)

# Generate data
n <- 100
p <- 20
X <- matrix(rnorm(n*p), n, p)
beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))
epsilon <- rnorm(n)
Y <- X %*% beta + epsilon

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

# Choose the best model by AIC
summary_fit$rss
```

```
##  [1] 84.67918 79.25928 75.67314 72.86411 70.71631 69.15662 67.83655 67.52960
##  [9] 67.26066 67.07658 66.91235 66.78501 66.67539 66.62573 66.57865 66.55697
## [17] 66.53371 66.52031 66.50713 66.50712
```

```r
summary_fit$which
```

```
##    (Intercept)   V1    V2    V3    V4    V5    V6    V7    V8    V9   V10   V11
## 1          TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2          TRUE TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3          TRUE TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 4          TRUE TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 5          TRUE TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 6          TRUE TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 7          TRUE TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 8          TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 9          TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 10         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 11         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 12         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 13         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 14         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## 15         TRUE TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
## 16         TRUE TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
## 17         TRUE TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
## 18         TRUE TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
## 19         TRUE TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## 20         TRUE TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##      V12   V13   V14   V15   V16   V17   V18   V19   V20
## 1  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 4  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 5  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 6  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 7  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 8  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## 9  FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
## 10 FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
## 11 FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
## 12 FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
## 13 FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
## 14 FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 15 FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 16 FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 17 FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 18  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 19  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 20  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
modelsize = apply(summary_fit$which,1, sum)
AIC = n * log(summary_fit$rss/n) + 2 * modelsize;
inrange <- function(x) {
  (x - min(x)) / (max(x) - min(x)) }
AIC = inrange(AIC)
plot(range(modelsize), c(0, 0.4), type="n",
     xlab = "Model Size (with Intercept)",
     ylab = "Model Selection Criteria", cex.lab = 1.5)
points(modelsize, AIC, col = "orange", type = "b", pch = 19)
legend("topright", legend=c("AIC"),
       col=c("orange"),
       lty = rep(1, 3), pch = 19, cex = 1.7)
```



```
bm_index <- which.min(AIC)
best_model <- which(summary_fit$which[bm_index, -1])
best_model_variables <- names(which(summary_fit$which[bm_index, ]))
cat("Best model variables (including intercept):", best_model_variables, "\n")
```

```
## Best model variables (including intercept): (Intercept) V1 V2 V3 V11 V13 V20
```

```
# Calculate prediction error
coeffs <- coef(subset, bm_index)
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
```

The best model selected by AIC has 6 variables (excluding intercept): X1, X2, X3, X11, X13, and X20. The prediction error is 0.6915662. The approach doesn't select the correct model. The falsely selected variables are X11, X13, and X20. The falsely removed variable is X4. I think this happens because of overfitting, and noise in the data. This approach evaluates every possible combination of predictors, which can lead to models that overfit the data, capturing the noise in the data as if it were signal. This noise can then influence which variables are selected, leading to incorrect inclusions (false positives like X11, X13, and X20) and exclusions (false negatives such as X4). Given that the noise ϵ N(0,1) in the dataset is relatively significant compared to the very small effect size of 1/16 for X4, it is plausible that the noise overshadowed the true effect of X4, making it harder for the selection procedure to identify X4 as a significant predictor.

(b).

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
```

Answer:

```r
n <- 100
p <- 20
n_sim <- 100
correct_models_count <- 0
variable_selection_counts <- rep(0, p)

set.seed(123)

for (i in 1:n_sim) {
    X <- matrix(rnorm(n * p), n, p)
    beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))
    epsilon <- rnorm(n)
    Y <- X %*% beta + epsilon

    data <- as.data.frame(cbind(Y, X))
    names(data) <- c("Y", paste("X", 1:p, sep=""))

    # Best subset selection
    subset <- regsubsets(Y ~ ., data = data, nvmax = p)
    summary_fit <- summary(subset)

    # Calculate AIC for each model
    model_size = apply(summary_fit$which, 1, sum)
    AIC = n * log(summary_fit$rss/n) + 2 * model_size
    best_model_index <- which.min(AIC)
    selected_vars <- summary_fit$which[best_model_index, -1]

    # Check for correct model
    if (all(selected_vars[1:4]) && all(!selected_vars[5:p])) {
        correct_models_count <- correct_models_count + 1
    }

    # Record variable selection
    variable_selection_counts <- variable_selection_counts + as.numeric(selected_vars)
}

# Proportion of correct models
proportion_correct <- correct_models_count / n_sim

# Proportion of each variable being selected
proportion_variable_selection <- variable_selection_counts / n_sim
```

```r
# Output results
cat("Proportion of times the correct model was selected:", proportion_correct, "\n")
```

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

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

```
## Proportion of times each variable was selected: 1 0.94 0.39 0.27 0.07 0.18 0.16 0.19 0.15 0.21 0.19
```

The proportion of times the correct model was selected is 0. The proportion of times each variable was selected is 1, 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, 0.19, 0.12, 0.21, 0.23.

(c).

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 sel
  ii. If you use AI tools, discuss your experience with it. Such as how to write the prompt and whether
```

Answer:

We can use Mallows' Cp or use larger sample size to improve variable selection accuracy.

We can first use Mallows' Cp.

```r
library(leaps)

# Simulation setup
set.seed(123)
n <- 100
p <- 20
n_sim <- 100
correct_models_count <- 0

# Enhanced simulation with Mallows' Cp and increased sample size
for (i in 1:n_sim) {
    X <- matrix(rnorm(n * p), n, p)
    beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))
    epsilon <- rnorm(n)
    Y <- X %*% beta + epsilon

    data <- as.data.frame(cbind(Y, X))
    names(data) <- c("Y", paste("X", 1:p, sep=""))

    # Best subset selection
    subset <- regsubsets(Y ~ ., data = data, nvmax = p)
    summary_fit <- summary(subset)

    # Calculate Mallows' Cp
    cp_values <- summary_fit$cp
    best_model_index <- which.min(cp_values)
    best_model <- summary_fit$which[best_model_index, ]

    # Check for correct model
    if (all(best_model[1:4]) && all(!best_model[5:p])) {
        correct_models_count <- correct_models_count + 1
    }
}

# Proportion of correct models
proportion_correct <- correct_models_count / n_sim
cat("Proportion of times the correct model was selected using Mallows' Cp:", proportion_correct, "\n")
```

```
## Proportion of times the correct model was selected using Mallows' Cp: 0.03
```

The selection rate improved from 0 to 0.03.

We can also use larger sample size.

```r
library(leaps)

# Parameters
n_large <- 200  # Increased sample size
p <- 20
n_sim <- 100
correct_models_count_large <- 0
variable_selection_counts_large <- rep(0, p)

set.seed(123)

# Simulation with increased sample size
for (i in 1:n_sim) {
    X <- matrix(rnorm(n_large * p), n_large, p)
    beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))
    epsilon <- rnorm(n_large)
    Y <- X %*% beta + epsilon

    data <- as.data.frame(cbind(Y, X))
    names(data) <- c("Y", paste("X", 1:p, sep=""))

    # Best subset selection
    subset <- regsubsets(Y ~ ., data = data, nvmax = p)
    summary_fit <- summary(subset)

    # Calculate AIC for each model
    model_size = apply(summary_fit$which, 1, sum)
    AIC = n_large * log(summary_fit$rss/n_large) + 2 * model_size
    best_model_index <- which.min(AIC)
    selected_vars <- summary_fit$which[best_model_index, ]
    # Check for correct model
    if (all(selected_vars[1:4]) && all(!selected_vars[5:p])) {
        correct_models_count_large <- correct_models_count_large + 1
    }
}

# Proportion of correct models with increased sample size
proportion_correct_large <- correct_models_count_large / n_sim

# Output results
cat("Proportion of times the correct model was selected with larger sample size (n=200):", proportion_c
```

```
## Proportion of times the correct model was selected with larger sample size (n=200): 0.07
```

The selection rate improved from 0 to 0.07.

```
  i. Mallows' Cp provides a balance between the goodness of fit (as measured by the residual sum of squa
   Larger sample sizes reduce the standard error of the regression coefficients. This reduction in varia

  ii. I used ChatGPT in (b) and (c). For the prompt, I copied the question and the code I used in (a), a
```

Question 2 (Training and Testing of Linear Regression)

We have introduced the formula of a linear regression

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

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).

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.

Answer:

```r
setwd("C:/zilin/senior/Stat432/hw2")
# load the data
realestate = read.csv("realestate.csv", row.names = 1)

# Defining X and y for the regression model
X <- as.matrix(cbind(1, realestate[, c("age", "distance", "stores")]))
y <- realestate$price

# Performing linear regression using matrix operations
```

```r
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
# Report beta_hat
print(beta_hat)
```

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

```r
# Compare with lm() function
model_lm <- lm(price ~ age + distance + stores, data = realestate)
summary(model_lm)$coefficients
```

```
##                 Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept) 42.97728621 1.3845423882  31.040788 1.085576e-109
## age         -0.25285583 0.0401053292  -6.304794  7.470473e-10
## distance    -0.00537913 0.0004530322 -11.873615  3.764064e-28
## stores       1.29744248 0.1942898311   6.677871  7.908452e-11
```

The beta_hat is the same as the fitted coefficients from the lm() function.

(b).

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?

Answer:

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

# Split the data into training and testing
train_data <- realestate[-test_idx,]
test_data <- realestate[test_idx,]

# Fit model on training data
```

```r
train_X <- as.matrix(cbind(1, train_data[, c("age", "distance", "stores")]))
train_y <- train_data$price
train_beta <- solve(t(train_X) %*% train_X) %*% t(train_X) %*% train_y

# Calculate predictions
train_pred <- train_X %*% train_beta
test_X <- as.matrix(cbind(1, test_data[, c("age", "distance", "stores")]))
test_pred <- test_X %*% train_beta
test_y <- test_data$price

# Calculate mean squared errors
train_error <- mean((train_y - train_pred)^2)
test_error <- mean((test_y - test_pred)^2)

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

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

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

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

I expected the testing data to be larger. The training data is typically lower because the model is directly fit on this data, the model parameters are optimized to minimize the error, potentially capturing both the underlying pattern and some noise (overfitting). And models are likely to perform better on the data they were trained on compared to new, unseen data. The result matches my expectation.

(c).

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.

Answer:

```r
# Full model fitting
full_model <- lm(price ~ ., data = train_data)

# Stepwise model selection based on AIC
stepwise_model <- step(full_model, direction="both", trace=0)

# Summarize the best model
summary(stepwise_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
```

```r
# Calculate predictions and errors
best_train_pred <- predict(stepwise_model, newdata=train_data)
best_test_pred <- predict(stepwise_model, newdata=test_data)

# Compute MSE for training and testing
best_train_error <- mean((train_y - best_train_pred)^2)
best_test_error <- mean((test_y - best_test_pred)^2)

# Output best model results
cat("Best Model Training Error:", best_train_error, "\n")
```

```
## Best Model Training Error: 68.52164
```

```r
cat("Best Model Testing Error:", best_test_error, "\n")
```

```
## Best Model Testing Error: 106.2898
```

The best model is price ~ date + age + distance + stores + latitude. The training error is 68.52164, the testing error is 106.2898.

Question 3 (Optimization)

(a).

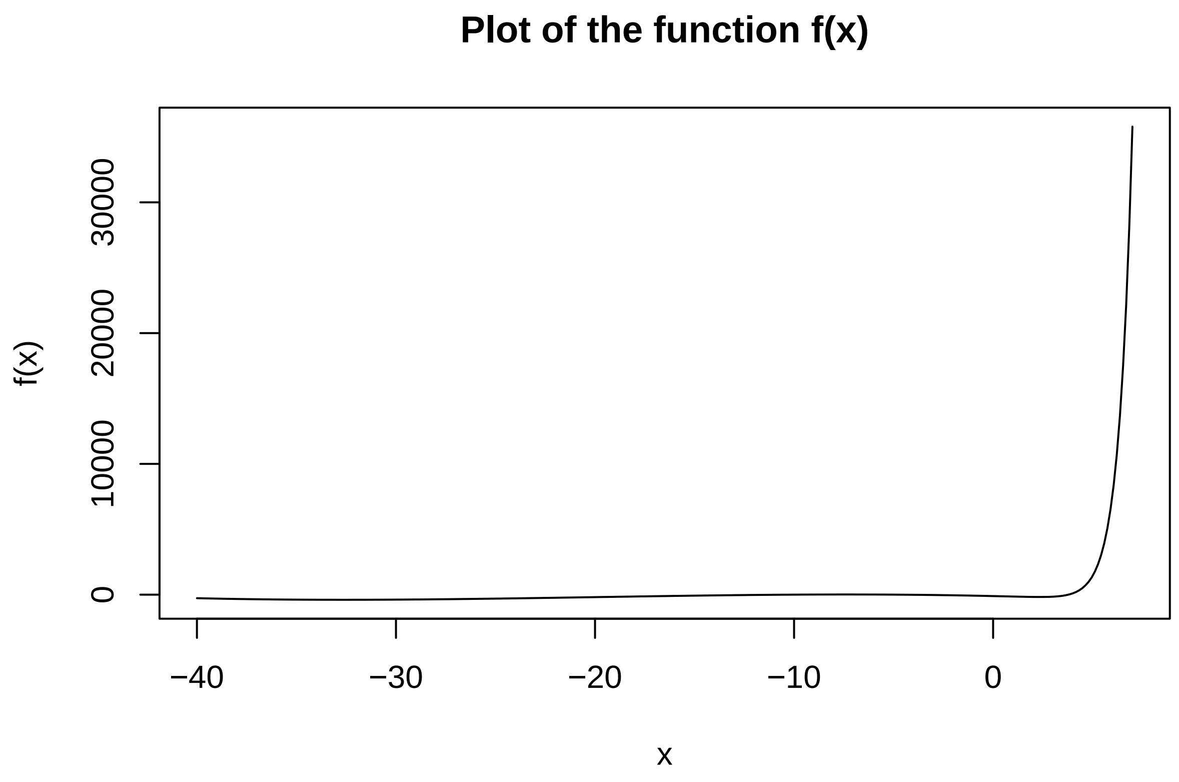
Consider minimizing the following univariate function:

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

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

Answer:

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

# Plot the function
x_vals <- seq(-40, 7, length.out = 300)
y_vals <- sapply(x_vals, f_obj)

plot(x_vals, y_vals, type = "l", main = "Plot of the function f(x)", xlab = "x", ylab = "f(x)")
```



(b).

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?

Answer:

```r
# Optimization from initial point x0 = -15
optim_1 <- optim(-15, f_obj, method = "BFGS")
```

```r
# Optimization from initial point x0 = 0
optim_2 <- optim(0, f_obj, method = "BFGS")

# Results
print("Result for initial point x = -15:")
```

```
## [1] "Result for initial point x = -15:"
```

```r
cat("The optimal value is",optim_1$value, "\n")
```

```
## The optimal value is -390.3858
```

```r
cat("the value of x is",optim_1$par, "\n")
```

```
## the value of x is -32.64911
```

```r
print("Result for initial point x = 0:")
```

```
## [1] "Result for initial point x = 0:"
```

```r
cat("The optimal value is",optim_2$value, "\n")
```

```
## The optimal value is -175.8626
```

```r
cat("the value of x is",optim_2$par, "\n")
```

```
## the value of x is 2.349967
```

The solutions I obtained are different. Because BFGS method is a local optimization algorithm, it finds the nearest local optimum from the starting point. Different initial points lead to different local minima due to the non-convex nature of the function, which have multiple local minima and maxima. The BFGS method is sensitive to starting points in this case.

(c).

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.

Answer:

```r
# Set function
f <- expression(3 * x^2 + 2 * y^2 - 4 * x * y + 6 * x - 5 * y + 7)

# Compute the partial derivative of f with respect to x
df_dx <- D(f, "x")
cat("Partial derivative with respect to x:\n")
```

```
## Partial derivative with respect to x:
```

```r
print(df_dx)
```

```
## 3 * (2 * x) - 4 * y + 6
```

```r
# Compute the partial derivative of f with respect to y
df_dy <- D(f, "y")
cat("\nPartial derivative with respect to y:\n")
```

```
##
## Partial derivative with respect to y:
```

```r
print(df_dy)
```

```
## 2 * (2 * y) - 4 * x - 5
```

```r
# Solve the system of equations df/dx = 0 and df/dy = 0 using 'solve()'

# Define the equations in matrix form Ax = b
A <- matrix(c(6, -4, -4, 4), nrow = 2, byrow = TRUE)
b <- c(-6, 5)

# Solve for x and y
solution <- solve(A, b)
x_analytic <- solution[1]
y_analytic <- solution[2]

cat("\nAnalytic solution: x =", x_analytic, ", y =", y_analytic, "\n")
```

```
##
## Analytic solution: x = -0.5 , y = 0.75
```

The partial derivatives are: f/ x = 6x - 4y + 6 f/ y = 4y − 4x − 5

Solving the First-Order Conditions: Setting the derivatives to zero to solve for x and y: 6x − 4y + 6 = 0 4y − 4x − 5 = 0 Solving these equations: x = -0.5, y = 0.75

(d).

Check the second-order condition to verify that the solution you obtained in the previous step is indeed a minimum.

Answer:

```
# Compute second-order partial derivatives (Hessian matrix components)
d2f_dx2 <- D(df_dx, "x")
d2f_dy2 <- D(df_dy, "y")
d2f_dxdy <- D(df_dx, "y")  # or D(df_dy, "x") since cross-derivatives are equal

cat("\nSecond-order partial derivative wrt x (d²f/dx²):\n")
```

```
##
## Second-order partial derivative wrt x (d²f/dx²):
```

```
print(d2f_dx2)
```

```
## 3 * 2
```

```
cat("\nSecond-order partial derivative wrt y (d²f/dy²):\n")
```

```
##
## Second-order partial derivative wrt y (d²f/dy²):
```

```
print(d2f_dy2)
```

```
## 2 * 2
```

```
cat("\nSecond-order partial derivative wrt x and y (d²f/dxdy):\n")
```

```
##
## Second-order partial derivative wrt x and y (d²f/dxdy):
```

```
print(d2f_dxdy)
```

```
## -4
```

```
# Hessian matrix
Hessian <- matrix(c(eval(d2f_dx2), eval(d2f_dxdy), eval(d2f_dxdy), eval(d2f_dy2)), nrow = 2, byrow = TRU
cat("\nHessian matrix:\n")
```

```
##
## Hessian matrix:
```

```
print(Hessian)
```

```
##      [,1] [,2]
## [1,]    6   -4
## [2,]   -4    4
```

```r
# Calculate the determinant of the Hessian
det_Hessian <- det(Hessian)
cat("\nDeterminant of the Hessian matrix:", det_Hessian, "\n")
```

```
##
## Determinant of the Hessian matrix: 8
```

```r
# Check for minimum condition: det(Hessian) > 0 and d²f/dx² > 0
is_minimum <- ifelse(det_Hessian > 0 && Hessian[1, 1] > 0, "Yes", "No")
cat("\nIs the critical point a minimum? ", is_minimum, "\n")
```

```
##
## Is the critical point a minimum?  Yes
```

The solution I obtained in the previous step is a minimum.

(e).

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?

Answer:

```r
# Define the function for optim
f_optim_xy <- function(params) {
  x <- params[1]
  y <- params[2]
  return(3*x^2 + 2*y^2 - 4*x*y + 6*x - 5*y + 7)
}

# Use optim to minimize f(x, y) starting from different initial points
initial_point_1 <- c(0, 0)
initial_point_2 <- c(-6, 10)

optim_result_1 <- optim(initial_point_1, f_optim_xy, method = "BFGS")
optim_result_2 <- optim(initial_point_2, f_optim_xy, method = "BFGS")

# Output the results
print("Result for initial point (0,0):")
```

```
## [1] "Result for initial point (0,0):"
```

```r
cat("The optimal value is",optim_result_1$value, "\n")
```

```
## The optimal value is 3.625
```

```r
cat("the value of x and y is",optim_result_1$par, "\n")
```

```
## the value of x and y is -0.5 0.75
```

```r
print("Result for initial point (-6,10):")
```

```
## [1] "Result for initial point (-6,10):"
```

```r
cat("The optimal value is",optim_result_2$value, "\n")
```

```
## The optimal value is 3.625
```

```r
cat("the value of x and y is",optim_result_2$par, "\n")
```

```
## the value of x and y is -0.5 0.75
```

The solution I obtained is x = -0.50, y = 0.75. Different choices of the initial point lead to same solutions. Because if the function has a convex nature in the region explored by the initial points, the optimization algorithm will always converge to the same global minimum, regardless of where it starts. In the case of the function given, although the function appears to have mixed terms and is not straightforwardly convex over all values, the specific landscape where the initial points lie might still guide the optimizer towards the same minimum. Also, even if a function is not globally convex, it might still have a unique critical point acting as the global minimum within a broad area. The critical point derived in parts (c) and (d), $(x,y)=(-0.5,0.75)$, might be the only minimum in a large portion of the function's domain, which both initial points sufficiently approximate to reach.