Stat 432 Homework 2

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

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 (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.
- **When using AI tools**, try to document your prompt and any follow-up prompts that further modify or correct the answer. You are also required to briefly comment on your experience with it, especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is $\geq 4.0.0$. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

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 performence of model selection tools under various scenarios. Let's first consider the following data generator:

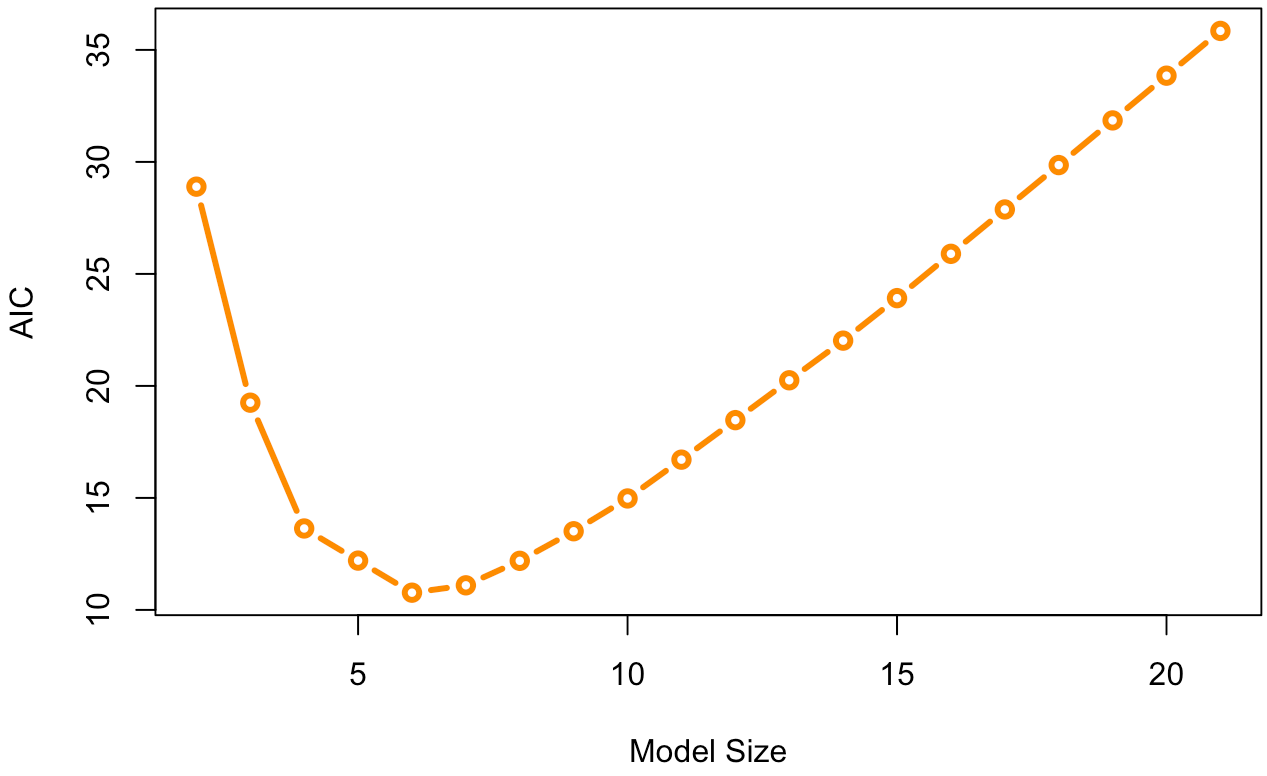
$$Y = \frac{1}{2} \cdot X_1 + \frac{1}{4} \cdot X_2 + \frac{1}{8} \cdot X_3 + \frac{1}{16} \cdot X_4 + \epsilon$$

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.

Answer:

```
# set seed
set.seed(1)

# generate data using the given data generator
n = 100
p = 20
x = matrix(rnorm(n*p), n, p)
y = 0.5*x[, 1] + 0.25*x[, 2] + 0.125*x[, 3] + 0.0625*x[, 4] + rnorm(n)

# Load the required library
library(leaps)

# Best subset selection
best_subsets <- regsubsets(x, y, nvmax = p)
sumleaps <- summary(best_subsets, matrix = T)

# Extract AIC for each model size
modelsize = apply(sumleaps$which,1,sum)

lm.fit <- lm(y ~ x)

AIC = n*log(sumleaps$rss/n) + 2*modelsize

# Plot AIC
 plot(modelsize, AIC, xlab = "Model Size", ylab = "AIC", type = "b", col = "darkorange", lwd =
3)
```




```
# check if the model is correct
all(sumleaps$which[which.min(AIC), -1] == c(1, 1, 1, 1, rep(0, 16)))
```

```
## [1] FALSE
```

For this particular dataset, our approach did not select the correct model. Variable `X_5` is selected and variable `X_4` is not selected. This is because the AIC criterion is not perfect.

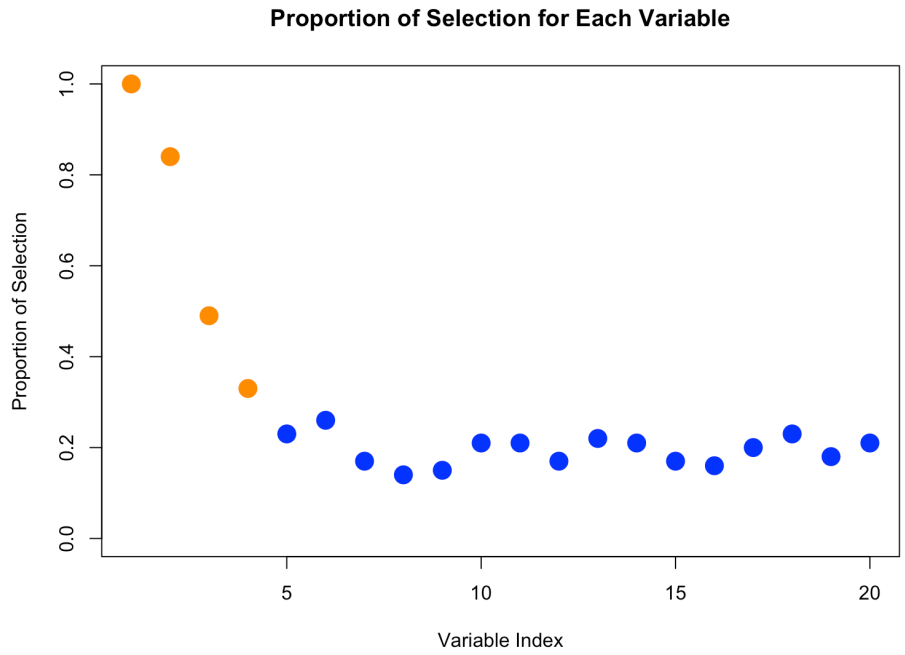
b. [10 points] Repeat the previous step with 100 runs of simulation, similar to our lecture note. Report 1) the proportion of times that this approach selects the correct model 2) the proportion of times that each variable was selected

Answer:

```r
nsim = 100
n = 100
p = 20

selection = matrix(NA, nsim, p)
correct = rep(NA, nsim)

for (i in 1:nsim)
{
  # the data generator
  x = matrix(rnorm(n*p), n, p)
  y = 0.5*x[, 1] + 0.25*x[, 2] + 0.125*x[, 3] + 0.0625*x[, 4] + rnorm(n)

  # Best subset selection
  best_subsets <- regsubsets(x, y, nvmax = p)
  sumleaps <- summary(best_subsets, matrix = T)

  # Extract AIC for each model size
  modelsize = apply(sumleaps$which,1,sum)
  lm.fit <- lm(y ~ x)
  AIC = n*log(sumleaps$rss/n) + 2*modelsize

  # check if the model is correct
  selection[i, ] = sumleaps$which[which.min(AIC), -1]
  correct[i] = all(selection[i, ] == c(1, 1, 1, 1, rep(0, 16)))
}

mean(correct)
```

```
## [1] 0.02
```

```r
plot(colMeans(selection), pch = 19, ylim = c(0, 1), cex = 2,
xlab = "Variable Index", ylab = "Proportion of Selection",
main = "Proportion of Selection for Each Variable",
col = c(rep("darkorange", 4), rep("blue", 16)))
```



The proportion of time that this approach selects the correct model is 0.02, which is quite low.

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.

Answer:

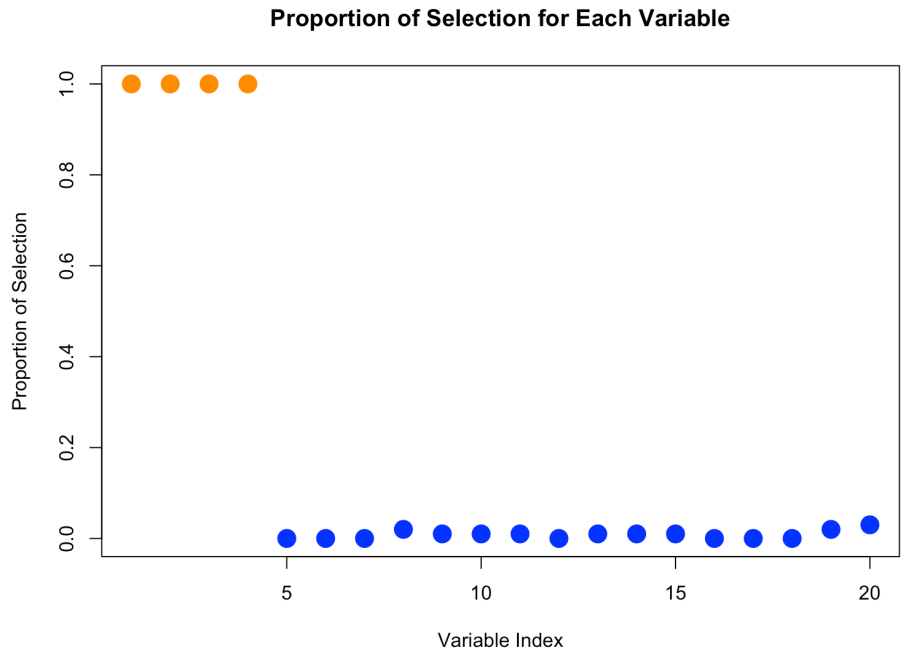
Here is my code, without using AI tools. I changed from AIC to BIC because BIC has a better chance to select the true variables, while AIC concerns mostly the prediction accuracy. I further made two changes:

- I increased the sample size to 500. This would help reduce the variation of the results because the noise variables would have conditional correlation almost zero with the response.
- I also changed the noise standard deviation to 0.1. This is as effective as increasing the effect size of the true variables.

```
  nsim = 100
  n = 500
  p = 20

  selection = matrix(NA, nsim, p)
  correct = rep(NA, nsim)

  for (i in 1:nsim)
  {
    # the data generator
    x = matrix(rnorm(n*p), n, p)
    y = 0.5*x[, 1] + 0.25*x[, 2] + 0.125*x[, 3] + 0.0625*x[, 4] + 0.1*rnorm(n)

    # Best subset selection
    best_subsets <- regsubsets(x, y, nvmax = p)
    sumleaps <- summary(best_subsets, matrix = T)

    # Extract AIC for each model size
    modelsize = apply(sumleaps$which,1,sum)
    lm.fit <- lm(y ~ x)

    # I changed the AIC to BIC
    BIC = n*log(sumleaps$rss/n) + modelsize*log(n);
    # AIC = n*log(sumleaps$rss/n) + 2*modelsize

    # check if the model is correct
    selection[i, ] = sumleaps$which[which.min(BIC), -1]
    correct[i] = all(selection[i, ] == c(1, 1, 1, 1, rep(0, 16)))
  }

  mean(correct)
```

```
## [1] 0.88
```

```
  plot(colMeans(selection), pch = 19, ylim = c(0, 1), cex = 2,
  xlab = "Variable Index", ylab = "Proportion of Selection",
  main = "Proportion of Selection for Each Variable",
  col = c(rep("darkorange", 4), rep("blue", 16)))
```



Overall, this boosts the selection rate to 0.88.

Alternatively, I tested the capability of using GPT-4. Here is the prompt

> **I have this code: "…" (I attached our class note code chunk). I want to adapt it to a version that uses best subset selection with AIC, using the leaps package. Can you help me with this?**

Basically it suggested some nonsense. And ultimately, I cannot use GPT-4 to produce a code that do exactly what I wanted. But here are the steps I followed to obtain a basic structure:

- Can you give me a code to only generate the data as the previous code?
- Remove the testing data
- Can you fit a best subset model using the leaps package for the data?
- Extract the best model for each model size.
- For each model, can you calculate the AIC score?
- Can you store these AIC scores in a vector so that I can compare them later.
- Instead of saving all the AIC scores, I just need the index for which model is the best.
- Great. Now, for this best model in each simulation run, can you look at which variables where selected? and compare it to the truth to see if it recovers the truth?
- This is not correct. recovered_truth should be defined within each simulation, and it should be a TRUE or FALSE variable. If the model recovers all correct variables, then its true. Otherwise false.
- Can you create a vector to save this recovered_truth for each simulation run?
- No. You defined this for all 100 simulation runs. I need this for each simulation run.

And up to here, it basically gave me the code structure I wanted. Then I went on modified the code slightly which works pretty much the same as my previous one. Hence the experience was not great, but this is only because there is no example code in the training process that can be used to suggest such a complicated task. With a step-by-step approach, you can still utilize its capability of generating code.

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.

Answer:

```
# read in and define X and y
realestate = read.csv("realestate.csv", row.names = 1)
y = realestate$price
X = data.matrix(cbind(1, realestate[, c("age", "distance", "stores")]))

# solve linear regression
b = solve(t(X) %*% X) %*% t(X) %*% y
print(b)
```

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

```
# to compare that with the fitted coefficients from the lm() function
lm(price ~ age + distance + stores, data = realestate)
```

```
##
## Call:
## lm(formula = price ~ age + distance + stores, data = realestate)
##
## Coefficients:
## (Intercept)          age     distance       stores
##   42.977286    -0.252856    -0.005379     1.297442
```

The coefficients are the same.

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:

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

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

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?

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

Answer:

```
  # define training and testing data
  y = realestate$price
  X = data.matrix(cbind(1, realestate[, c("age", "distance", "stores")]))

  set.seed(432)
  test_idx = sample(nrow(realestate), 100)

  trainx = X[ -test_idx, ]
  testx  = X[ test_idx, ]
  trainy = y[ -test_idx ]
  testy  = y[ test_idx ]

  # fit linear regression using my own code
  b = solve(t(trainx) %*% trainx) %*% t(trainx) %*% trainy

  # predicting on the testing data
  pred = testx %*% b

  # training error
  mean((trainy - trainx %*% b)^2)
```

```
## [1] 74.57346
```

```
  # testing error
  mean((testy - pred)^2)
```

```
## [1] 119.4458
```

```
  # the overall variance of y
  var(y)
```

```
## [1] 185.1365
```

The testing error is slightly higher. This is expected since the training error can always be affected by over-fitting.

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.

Answer:

```
realestate = read.csv("realestate.csv", row.names = 1)
train = realestate[-test_idx, ]
test  = realestate[test_idx, ]

lm_fit = lm(price ~ ., data = train)
lm_fit
```

```
##
## Call:
## lm(formula = price ~ ., data = train)
##
## Coefficients:
## (Intercept)         date          age     distance       stores     latitude
##  -1.118e+04    4.438e+00   -3.098e-01   -4.889e-03    9.931e-01    2.140e+02
##   longitude
##  -2.509e+01
```

```
# perform step wise regression using AIC
step(lm_fit, direction = "both", trace = 0)
```

```
##
## Call:
## lm(formula = price ~ date + age + distance + stores + latitude,
##     data = train)
##
## Coefficients:
## (Intercept)         date          age     distance       stores     latitude
##  -1.439e+04    4.467e+00   -3.098e-01   -4.613e-03    9.964e-01    2.178e+02
```

```
# the best model
lm_fit = lm(price ~ date + age + distance + stores + latitude, data = train)

# prediction error
pred = predict(lm_fit, newdata = test)
pred_err = mean((pred - test$price)^2)
print(pred_err)
```

```
## [1] 106.2898
```

```
# training error
pred = predict(lm_fit, newdata = train)
pred_err = mean((pred - train$price)^2)
print(pred_err)
```

```
## [1] 68.52164
```

This model is slightly better than the previous one.

Question 3 (Optimization)

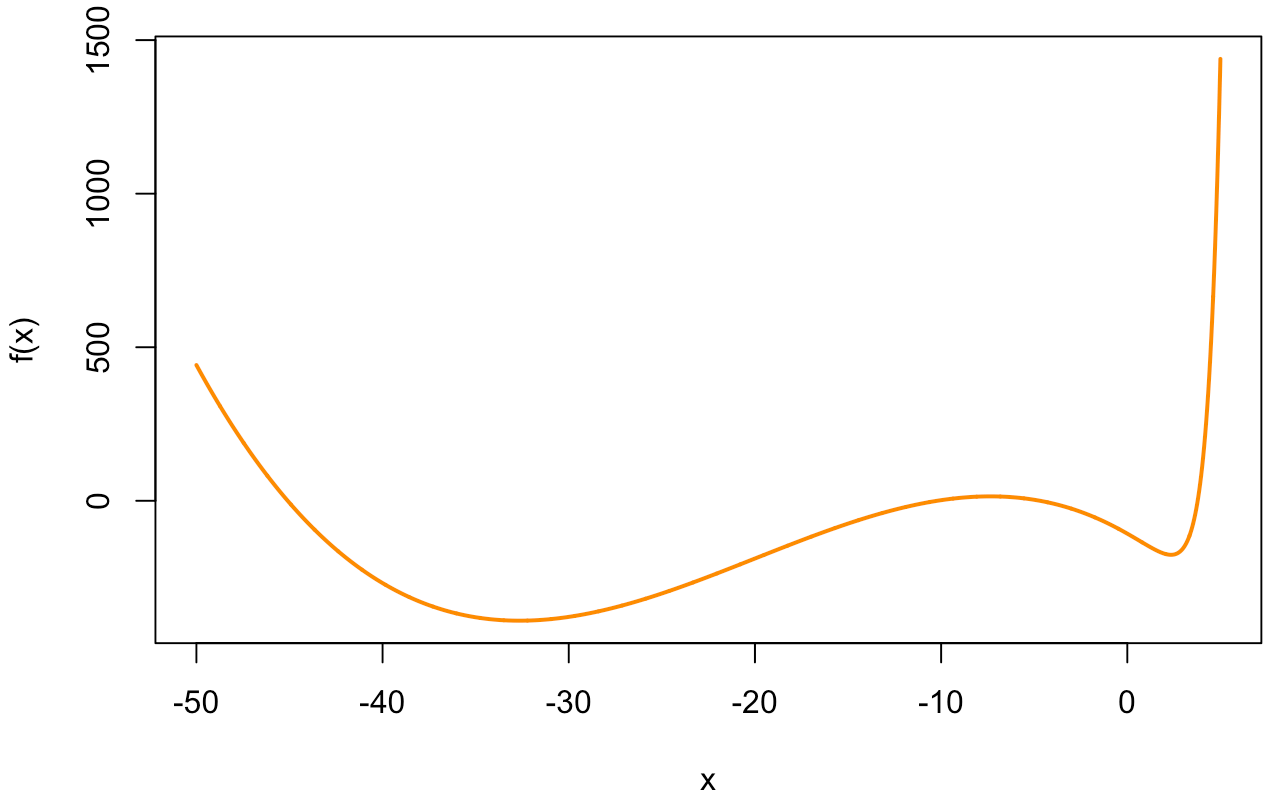
a. [10 Points] Consider minimizing the following univariate function:

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

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

Answer:

```
f_obj <- function(x) exp(1.5*x) - 3*(x+6)^2 - 0.05*x^3
x = seq(-50, 5, 0.01)
plot(x, f_obj(x), type = "l", col = "darkorange", lwd = 2, 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?

Answer:

```
# perform BFGS with different initializations
sol1 = optim(-15, fn = f_obj, method = "BFGS")
print(c(sol1$par, sol1$value))
```

```
## [1]  -32.64911 -390.38577
```

```
sol2 = optim(0, fn = f_obj, method = "BFGS")
print(c(sol2$par, sol2$value))
```

```
## [1]    2.349967 -175.862620
```

The two solutions are different. One is at -32.6491107, and other one at 2.3499671. This is because BFGS method only converged to a local minimum. Only the first one corresponds to a global minimum.

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.

Answer:

The partial derivatives are:

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

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

Setting them to zero, we have:

$$6x - 4y + 6 = 0$$

$$4y - 4x - 5 = 0$$

Solving these two equations, we have $x = -0.5$ and $y = 9/4$.

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

Answer:

The Hessian matrix is:

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

We can calculate the eigenvalues of this matrix, which are all positive. Hence, the solution is indeed a minimum.

```
H = matrix(c(6, -4, -4, 4), 2, 2)
eigen(H)
```

```
## eigen() decomposition
## $values
## [1] 9.1231056 0.8768944
##
## $vectors
##            [,1]       [,2]
## [1,] -0.7882054 -0.6154122
## [2,]  0.6154122 -0.7882054
```

e. [10 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?

Answer:

```r
f_obj2 <- function(x) 3*x[1]^2 + 2*x[2]^2 - 4*x[1]*x[2] + 6*x[1] - 5*x[2] + 7
sol = optim(c(0, 0), fn = f_obj2, method = "BFGS")
print(c(sol$par, sol$value))
```

```
## [1] -0.500  0.750  3.625
```

The solution is at -0.5, 0.75. Different choices of the initial point does not matter in this case because the function is strictly convex.