# Operations Research, Spring 2024 (112-2)

## Homework 2

B11705018 Shu-Ting Hsu

1. Consider the following LP

$$\begin{aligned} \min \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \geq 4 \\ & x_1 + 2x_2 \leq 9 \\ & x_2 \geq 3 \\ & x_i \geqslant 0 \quad \forall i = 1, 2. \end{aligned}$$

(a) Find the standard form of this LP.

solution:

$$\begin{aligned} \min \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 - x_3 = 4 \\ & x_1 + 2x_2 + x_4 = 9 \\ & x_2 - x_5 = 3 \\ & x_i \geqslant 0 \quad \forall i = 1, ..., 5 \end{aligned}$$

(b) List all the basic solutions and basic feasible solutions for the standard form of this LP.

solution:

The set of basic solution is $\{(3,3,2,0,0), (1,3,0,2,0), (-1,5,0,0,2), (9,0,5,0,-3), (4,0,0,5,-3), (0,3,-1,3,0), (0,\frac{9}{2},\frac{1}{2},0,\frac{3}{2}), (0,4,0,1,1), (0,0,-4,9,3)\}$. The set of bfs is $\{(3,3,2,0,0), (1,3,0,2,0), (0,\frac{9}{2},\frac{1}{2},0,\frac{3}{2}), (0,4,0,1,1)\}$. The six bases and the associated basic variables are listed below.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | basis | BFS |
|---|---|---|---|---|---|---|
| 3 | 3 | 2 | 0 | 0 | $\{x_1, x_2, x_3\}$ | YES |
| 1 | 3 | 0 | 2 | 0 | $\{x_1, x_2, x_4\}$ | YES |
| -1 | 5 | 0 | 0 | 2 | $\{x_1, x_2, x_5\}$ | NO |
| 9 | 0 | 5 | 0 | -3 | $\{x_1, x_3, x_5\}$ | NO |
| 4 | 0 | 0 | 5 | -3 | $\{x_1, x_4, x_5\}$ | NO |
| 0 | 3 | -1 | 3 | 0 | $\{x_2, x_3, x_4\}$ | NO |
| 0 | $\frac{9}{2}$ | $\frac{1}{2}$ | 0 | $\frac{3}{2}$ | $\{x_2, x_3, x_5\}$ | YES |
| 0 | 4 | 0 | 1 | 1 | $\{x_2, x_4, x_5\}$ | YES |
| 0 | 0 | -4 | 9 | 3 | $\{x_3, x_4, x_5\}$ | NO |

(c) List all the extreme points of the feasible region of the original LP. DO NOT prove that they are extreme points; just list them. Then show that there is a one-to-one correspondence between basic feasible solutions and extreme points.

solution:

The set of extreme points is $\{(3,3),(1,3),(0,\frac{9}{2}),(0,4)\}$. The one-to-one correspondence between basic feasible solutions and extreme points is listed below.

| extreme point | correspond bfs |
|---|---|
| $(3,3)$ | $(3,3,2,0,0)$ |
| $(1,3)$ | $(1,3,0,2,0)$ |
| $(0,\frac{9}{2})$ | $(-1,5,0,0,2)$ |
| $(0,4)$ | $(9,0,5,0,-3)$ |

(d) Use the simplex method (with the two-phase implementation, if needed) and the smallest index rule to solve the LP. Write down the complete process, optimal solution to the original LP, and its associated objective value. Is there any iteration that has no improvement?

solution:

Phase-I standard form LP:

$$
\begin{aligned}
\min\quad & x_6 + x_7 \\
\text{s.t.}\quad & x_1 + x_2 - x_3 + x_6 = 4 \\
& x_1 + 2x_2 + x_4 = 9 \\
& x_2 - x_5 + x_7 = 3 \\
& x_i \geqslant 0 \quad \forall i = 1, ..., 7
\end{aligned}
$$

| 0 | 0 | 0 | 0 | 0 | -1 | -1 | 0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | -1 | 0 | 0 | 1 | 0 | $x_6 = 4$ |
| 1 | 2 | 0 | 1 | 0 | 0 | 0 | $x_4 = 9$ |
| 0 | 1 | 0 | 0 | -1 | 0 | 1 | $x_7 = 3$ |

adjust $\longrightarrow$

| 1 | 2 | -1 | 0 | -1 | 0 | 0 | 7 |
|---|---|---|---|---|---|---|---|
| $\boxed{1}$ | 1 | -1 | 0 | 0 | 1 | 0 | $x_6 = 4$ |
| 1 | 2 | 0 | 1 | 0 | 0 | 0 | $x_4 = 9$ |
| 0 | 1 | 0 | 0 | -1 | 0 | 1 | $x_7 = 3$ |

$\longrightarrow$

| 0 | 1 | 0 | 0 | -1 | 0 | 3 |
|---|---|---|---|---|---|---|
| 1 | 1 | -1 | 0 | 0 | 0 | $x_1 = 4$ |
| 0 | 1 | 1 | 1 | 0 | 0 | $x_4 = 5$ |
| 0 | $\boxed{1}$ | 0 | 0 | -1 | 1 | $x_7 = 3$ |

$\longrightarrow$

| 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|
| 1 | 0 | -1 | 0 | 1 | $x_1 = 1$ |
| 0 | 0 | 1 | 1 | 1 | $x_4 = 2$ |
| 0 | 1 | 0 | 0 | -1 | $x_2 = 3$ |

Phase-II iteration:

| -1 | -3 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|
| 1 | 0 | -1 | 0 | 1 | $x_1 = 1$ |
| 0 | 0 | 1 | 1 | 1 | $x_4 = 2$ |
| 0 | 1 | 0 | 0 | -1 | $x_2 = 3$ |

adjust $\longrightarrow$

| 0 | 0 | -1 | 0 | -2 | 10 |
|---|---|---|---|---|---|
| 1 | 0 | -1 | 0 | 1 | $x_1 = 1$ |
| 0 | 0 | 1 | 1 | 1 | $x_4 = 2$ |
| 0 | 1 | 0 | 0 | -1 | $x_2 = 3$ |

From iteration, we have optimal solution $(x_1, x_2) = (1,3)$ with objective value $= 10$. There is no iteration that has no improvement.

2. Consider the integer program

$$
\begin{aligned}
\max\quad & 2x_1 + 2x_2 + 5x_3 + 11x_4 + 10x_5 + 3x_6 - 6x_7 \\
\text{s.t.}\quad & x_1 + 4x_2 + 3x_3 + 5x_4 + 3x_5 - 4x_6 + 2x_7 \leq 6 \\
& x_i \in \{0,1\} \quad \forall i = 1, ..., 7,
\end{aligned}
$$

which is a variant of the knapsack problem.

(a) Use the greedy algorithm introduced in class to solve the linear relaxation of this integer program.

solution:

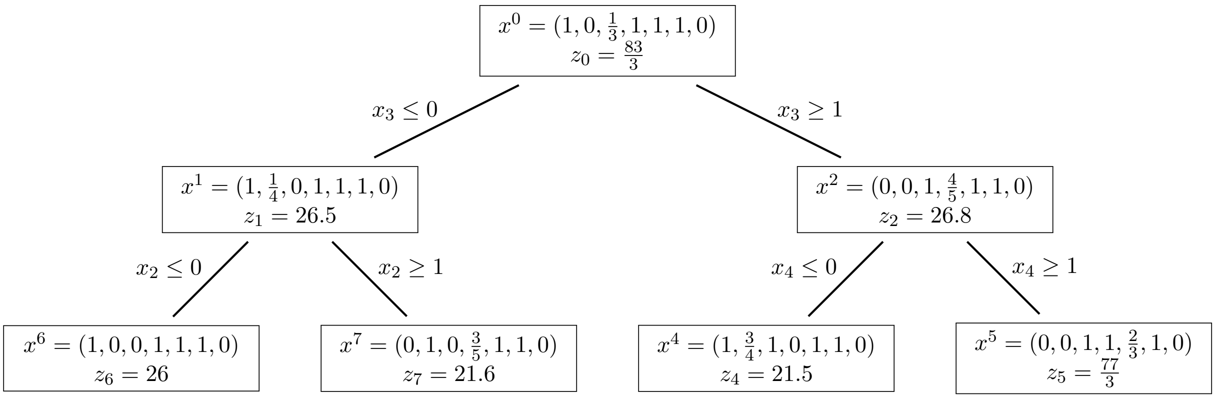
Linear relaxation:

$$
\begin{aligned}
\max \quad & 2x_1 + 2x_2 + 5x_3 + 11x_4 + 10x_5 + 3x_6 - 6x_7 \\
\text{s.t.} \quad & x_1 + 4x_2 + 3x_3 + 5x_4 + 3x_5 - 4x_6 + 2x_7 \le 6 \\
& x_i \in [0,1] \quad \forall i = 1, ..., 7
\end{aligned}
$$

From $x_1$ to $x_7$, the value-weight ratios are 2, $\frac{1}{2}$, $\frac{5}{3}$, $\frac{11}{5}$, $\frac{10}{3}$, $\frac{-3}{4}$, -3. We can see that both the value-weight ratio of $x_6$ and $x_7$ are negative. However, from the integer program, it shows that the value of $x_6$ is positive while its weight is negative. Since it will increase the capacity, we should choose it first. But for $x_7$, the value is negative while the weight is positive. Hence, we shouldn't choose it in any circumstance. Now, we know the order of choosing variables is $x_6 \to x_5 \to x_4 \to x_1 \to x_3 \to x_2$. Hence, by the order, we get the solution $x^{ALG} = (1, 0, \frac{1}{3}, 1, 1, 1, 0)$ with objective value $= \frac{83}{3}$.

(b) Use the branch-and-bound algorithm to solve the original integer program. Depict the full branch-and-bound tree. Do not write down the solution process of each node; write down just an optimal solution and its objective value of each node.

solution:




From the branch-and-bound tree above, we can find the optimal solution for the original IP is $x^* = (1, 0, 0, 1, 1, 1, 0)$ with objective value $z^* = 26$.

3. There are $m$ towns in a county. The county government is considering where to build at least $p$ landfills in n potential locations. The distance between town $i$ and location $j$ is $d_{ij}$. The population at town $i$ is $h_i$. The government wants to maximize the average distances between each person and her/his closest landfill.

(a) Formulate an integer program to achieve this goal. Use your own words to explain the formulation in details.

solution:

Let $J = 1, ..., n$ be the set of candidate locations and $I = 1, ..., m$ be the set of towns. Let $x_j = 1$ if a landfill is built at location $j \in J$ or 0 otherwise, and $y_i$ be the distance between town $i \in I$ and its closest landfill. The formulation is

$$
\begin{aligned}
\max \quad & \sum_{i \in I} y_i h_i \\
\text{s.t.} \quad & \sum_{j \in J} x_j \geqslant p \\
& y_i \leqslant x_j d_{ij} + M_i(1 - x_j) \quad \forall i \in I, j \in J \\
& x_j \in \{0, 1\} \quad \forall j \in J
\end{aligned}
$$

where $M_i$ is a very large number. Here we set $M_i = \max_{j \in J}\{d_{ij}\}$.

(b) Consider the two instances contained in the file "OR112-2 hw02 data.xlsx" (if you do not use Microsoft Excel, you may use Google Spreadsheet to open the file). In each sheet, which contains an instance, parameter symbols are in blue cells, indices are in orange cells, and parameter values are in green cells. For example, in instance 2, $m = 20, n = 10, p = 5, h_2 = 28, and\ d_{12,5} = 164$.

solution:

```python
import pandas as pd
from gurobipy import *

# Instance 1
instance1 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 1')
towns = range(10)
locations = range(instance1.iloc[0, 1])
landfill_min = instance1.iloc[1, 1]
human = instance1.iloc[4:14, 1]
distances = instance1.iloc[4:14, 4:9]
h = human.values
d = distances.values

eg1 = Model("eg1")

x = []
for j in locations:
    x.append(eg1.addVar(lb = 0, vtype = GRB.BINARY, name = "x" + str(j+1)))

y = []
for i in towns:
    y.append(eg1.addVar(lb = 0, vtype = GRB.INTEGER, name="y" + str(i+1)))

M = [max(d[i][j] for j in range(len(locations))) for i in towns]

# setting the objective function
eg1.setObjective(quicksum(h[i]*y[i] for i in towns), GRB.MAXIMIZE)

# add constraints and name them
eg1.addConstr(quicksum(x[j] for j in locations) >= landfill_min, "demand_fulfillment1")
eg1.addConstrs((y[i] <= x[j]*d[i][j] + M[i]*(1-x[j]) for i in towns for j in locations), "min_distance")

eg1.optimize()

# Instance 2
instance2 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 2')
towns = range(20)
locations = range(instance2.iloc[0, 1])
landfill_min = instance2.iloc[1, 1]
human = instance2.iloc[4:24, 1]
distances = instance2.iloc[4:24, 4:14]
h = human.values
d = distances.values

eg1.optimize()
```

Instance 1:
optimal solution $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 0, 0)$
objective value $z^* = 39027$
Instance 2:
optimal solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (1, 0, 1, 0, 1, 0, 1, 1, 0, 0)$
objective value $z^* = 68665$

4. Continue from Problem 3.

(a) Consider the following greedy algorithm designed for the landfill location problem. The algorithm runs in $n - p$ iterations. Initially, the algorithm starts with a feasible solution that builds a landfill in each candidate locations. In each iteration, it examines all landfills to see if one and exactly one landfill can be removed, removing which one results in the maximum objective value in that iteration (and if there is a tie, it chooses the landfill with the smallest index). It keeps removing landfills until only $p$ landfills remain.

solution:

```
import pandas as pd
from gurobipy import *

# Instance 1
instance1 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 1')
towns = range(10)
locations = range(instance1.iloc[0, 1])
landfill_min = instance1.iloc[1, 1]
human = instance1.iloc[4:14, 1]
distances = instance1.iloc[4:14, 4:9]
h = human.values
d = distances.values

x = [1] * len(locations)

max_obj = 0
M = [max(d[i][j] for j in range(len(locations))) for i in towns]
x_now = x[:]
for i in range(len(locations) - landfill_min):
    temp_max = 0
    x_new = x[:]
    for j in range(len(locations)):  # Iterate over all landfills
        x_temp = x_new[:]  # Make a copy of x for each iteration
        x_temp[j] = 0  # Remove the j-th landfill temporarily
        y = M[:]
        for k in range(len(towns)):
            for n in range(len(locations)):
                y[k] = min(y[k], x_temp[n]*d[k][n] + M[k]*(1-x_temp[n]))

        sol = sum(h[m] * y[m] for m in range(len(towns)))

        if sol > temp_max:
            temp_max = sol
            x_now = x_temp[:]
    max_obj = max(max_obj, temp_max)
    x = x_now[:]

for j in range(len(locations)):
    print('x' + str(j + 1), '=', x[j])

print('objective value =', max_obj)

# Instance 2
instance2 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 2')
towns = range(20)
locations = range(instance2.iloc[0, 1])
landfill_min = instance2.iloc[1, 1]
human = instance2.iloc[4:24, 1]
distances = instance2.iloc[4:24, 4:14]
h = human.values
d = distances.values

x = [1] * len(locations)
max_obj = 0
M = [max(d[i][j] for j in range(len(locations))) for i in towns]
x_now = x[:]
```

```python
for i in range(len(locations) - landfill_min):
    temp_max = 0
    x_new = x[:]
    for j in range(len(locations)):  # Iterate over all landfills
        x_temp = x_new[:]  # Make a copy of x for each iteration
        x_temp[j] = 0  # Remove the j-th landfill temporarily
        y = M[:]
        for k in range(len(towns)):
            for n in range(len(locations)):
                y[k] = min(y[k], x_temp[n]*d[k][n] + M[k]*(1-x_temp[n]))

        sol = sum(h[m] * y[m] for m in range(len(towns)))

        if sol > temp_max:
            temp_max = sol
            x_now = x_temp[:]
    max_obj = max(max_obj, temp_max)
    x = x_now[:]

for j in range(len(locations)):
    print('x' + str(j + 1), '=', x[j])

print('objective value =', max_obj)
```

Instance 1:
optimal solution $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 0, 0)$
objective value $z^* = 39027$
Instance 2:
optimal solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (0, 1, 0, 1, 0, 1, 0, 0, 1, 1)$
objective value $z^* = 65364$

(b) Consider another heuristic algorithm designed for the landfill location problem based on linear relaxation. The idea is simple. Given an instance, first we solve its linear relaxation to obtain a probably fractional LP-optimal solution $x^{LP}$ . We then pick the largest $p$ values and set the corresponding $x_j^{LP}$ to 1 (if there is a tie, pick those with smaller indices); the remaining $n - p$ variables are set to 0.

solution:

```python
import pandas as pd
from gurobipy import *

# Instance 1
instance1 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 1')
towns = range(10)
locations = range(instance1.iloc[0, 1])
landfill_min = instance1.iloc[1, 1]
human = instance1.iloc[4:14, 1]
distances = instance1.iloc[4:14, 4:9]
h = human.values
d = distances.values

eg1 = Model("eg1")

x = []
for j in locations:
    x.append(eg1.addVar(lb = 0, vtype = GRB.CONTINUOUS, name = "x" + str(j+1)))

y = []
for i in towns:
    y.append(eg1.addVar(lb = 0, vtype = GRB.CONTINUOUS, name="y" + str(i+1)))

```

```python
M = [max(d[i][j] for j in range(len(locations))) for i in towns]

# setting the objective function
eg1.setObjective(quicksum(h[i]*y[i] for i in towns), GRB.MAXIMIZE)

# add constraints and name them
eg1.addConstr(quicksum(x[j] for j in locations) >= landfill_min, "
    demand_fulfillment1")
eg1.addConstrs((y[i] <= x[j]*d[i][j] + M[i]*(1-x[j]) for i in towns for j in
     locations), "min_distance")
eg1.addConstrs((x[j] >= 0 for j in locations), "x_value1")
eg1.addConstrs((x[j] <= 1 for j in locations), "x_value2")

# Solve the linear relaxation
eg1.optimize()

# Get the solution
if eg1.status == GRB.OPTIMAL:
    xLP = [var.x for var in x]
    # Determine the number of variables to set to 1
    p = landfill_min  # determine the value of p
    # Select the top p variables with the largest values and set them to 1
    top_indices = sorted(range(len(xLP)), key=lambda i: xLP[i], reverse=True
    )[:p]
    for j in range(len(x)):
        if j in top_indices:
            x[j].lb = 1.0
            x[j].ub = 1.0
        else:
            x[j].lb = 0.0
            x[j].ub = 0.0
    # Update the model
    eg1.update()
    # Resolve the model
    eg1.optimize()
    # Print or use the solution as needed
    if eg1.status == GRB.OPTIMAL:
        # Print or use the solution
        pass  # Placeholder, you need to add code here
else:
    print("No solution found for the linear relaxation.")

# Instance 2
instance2 = pd.read_excel('OR112-2_hw02_data.xlsx','Problem 3 Instance 2')
towns = range(20)
locations = range(instance2.iloc[0, 1])
landfill_min = instance2.iloc[1, 1]
human = instance2.iloc[4:24, 1]
distances = instance2.iloc[4:24, 4:14]
h = human.values
d = distances.values

# Solve the linear relaxation
eg1.optimize()

# Get the solution
if eg1.status == GRB.OPTIMAL:
    xLP = [var.x for var in x]
    # Determine the number of variables to set to 1
    p = landfill_min  # determine the value of p
    # Select the top p variables with the largest values and set them to 1
    top_indices = sorted(range(len(xLP)), key=lambda i: xLP[i], reverse=True
    )[:p]
    for j in range(len(x)):
        if j in top_indices:
            x[j].lb = 1.0
```

```
            x[j].ub = 1.0
        else:
            x[j].lb = 0.0
            x[j].ub = 0.0
    # Update the model
    eg1.update()
    # Resolve the model
    eg1.optimize()
    # Print or use the solution as needed
    if eg1.status == GRB.OPTIMAL:
        # Print or use the solution
        pass  # Placeholder, you need to add code here
else:
    print("No solution found for the linear relaxation.")
```

Instance 1:
optimal solution $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 0, 0)$
objective value $z^* = 39027$
Instance 2:
optimal solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}) = (1, 0, 0, 0, 1, 1, 0, 0, 1, 1)$
objective value $z^* = 63044$

(c) Let $z_k^G$ be the objective value of the solution found by the greedy algorithm in Part (a) for instance $k \in \{1, 2\}$. Similarly, let $z_k^R$ be that by the heuristic algorithm in Part (b) for instance $k \in \{1, 2\}$. For each of the two instances, report $z_k^{IP}, z_k^{LP}, z_k^G, z_k^R$, and the four percentage optimality gaps (each algorithm has two optimality gaps, one uses $z_k^{IP}$ and one uses $z_k^{LP}$). In average, which algorithm performs better in these two instances?

solution:

Instance 1:
$z_1^{IP} = 39027,\ z_1^{LP} = 53434.9,\ z_1^G = 39027,\ z_1^R = 39027$
$z_1^G$ optimality gap($z_1^{IP}$) = 0%
$z_1^G$ optimality gap($z_1^{LP}$) = 26.96%
$z_1^R$ optimality gap($z_1^{IP}$) = 0%
$z_1^R$ optimality gap($z_1^{LP}$) = 26.96%
Instance 2:
$z_2^{IP} = 68665,\ z_2^{LP} = 117810.4,\ z_2^G = 65364,\ z_2^R = 63044$
$z_2^G$ optimality gap($z_2^{IP}$) = 4.81%
$z_2^G$ optimality gap($z_2^{LP}$) = 44.52%
$z_2^R$ optimality gap($z_2^{IP}$) = 8.19%
$z_2^R$ optimality gap($z_2^{LP}$) = 46.49%
From the result above, we can see that there is no difference between two algorithm in instance 1, but it is clear that the greedy algorithm performs better in instance 2. In average, the greedy algorithm performs better in these two instances.

(d) Comment on the time complexity (with the big-O notation) and performance of the two heuristic algorithms. If you need to solve the landfill location problem with $m = 500, n = 100$, and $p = 20$, which algorithm do you prefer? Why?

solution:

The greedy algorithm:
Time complexity is $O(n^3 m)$, where $n$ is the number of potential locations and $m$ is the number of towns.
The heuristic algorithm in part(b):
Time complexity is $O((nm)^3)$, where $n$ is the number of potential locations.

It shows that the time complexity of the greedy algorithm in part(b) is better, and from part(c), we know that the greedy algorithm also perform better. Hence, if I need to solve the landfill location problem with $m = 500, n = 100,$ and $p = 20$, I prefer the greedy algorithm.

5. Consider the following nonlinear program

$$\min \quad 3x_1^2 + 2x_2^2 + 4x_1x_2 + 6e^{x_1} + x_2.$$

Later when needed, use numerical solutions rather than analytical solutions. For example, when solving $-x = e^x$, using any calculator or software to find $x = -0.567$ as a numerical solution is good enough. There is no need to analytically solve $-x = e^x$.

(a) Start from $(x_1, x_2) = (0, 0)$ to run one iteration of the gradient descent method to solve this instance. In this iteration, move to the global minimum along the direction you choose. Write down the detailed process of the iteration.

solution:

Let $f(x) = 3x_1^2 + 2x_2^2 + 4x_1x_2 + 6e^{x_1} + x_2$

$$\nabla f(x) = \begin{bmatrix} 6x_1 + 4x_2 + 6e^{x_1} \\ 4x_1 + 4x_2 + 1 \end{bmatrix}$$

$x^0 = (x_1, x_2) = (0, 0)$

$\nabla f(x^0) = (6, 1)$

$a_0 = \text{argmin}_{a \geq 0}\ f((0,0) - a(6,1)) = 134a^2 + 6e^{-6a} - a$

$a_0 = 0.08$

$x^1 = (0, 0) - 0.08(6, 1) = (-0.48, -0.08)$

(b) Start from $(x_1, x_2) = (0, 0)$ to run one iteration of the Newton's method to solve this instance. Write down the detailed process of the iteration.

solution:

$$\nabla^2 f(x) = \begin{bmatrix} 6 + 6e^{x_1} & 4 \\ 4 & 4 \end{bmatrix}$$

$$\begin{aligned} x^1 &= x^0 - [\nabla^2 f(x^0)]^{-1} \nabla f(x^0) \\ &= (0, 0) - (\frac{5}{8}, \frac{-3}{8}) \\ &= (\frac{-5}{8}, \frac{3}{8}) \end{aligned}$$