ChBE 4746/6746

Homework 2 Solution

Fani Boukouvala, Pengfei Cheng, Zachary Kilwein

Update date: February 27, 2021

Problem 1 Simplex tableau

1. Formulate the problem into the standard form:

$$\begin{aligned} \max\ & 3x_1 + 2x_2 \\ \text{s.t.}\ & -x_1 + 2x_2 + x_3 = 4 \\ & 3x_1 + 2x_2 + x_4 = 14 \\ & x_1 - x_2 + x_5 = 3 \\ & x_1, \ldots, x_5 \geq 0. \end{aligned}$$

2. form the simplex tableau: Table 1.

Table 1: Tableau T1

basic variables	x_1	x_2	x_3	x_4	x_4	$\bar{b}$
x_3	-1	2	1	0	0	4
x_4	3	2	0	1	0	14
x_5	1	-1	0	0	1	3
z	-3	-2	0	0	0	0

3. Find pivot column: x_1 has the coefficient with largest absolute values. Find ratio: $4/-1 = -4$, ignore; $14/3 = 4.333$, $3/1 = 3$, so the ratio is 3. New tableau: Table 2.

Table 2: Tableau T2

basic variables	x_1	x_2	x_3	x_4	x_4	$\bar{b}$
x_3	0	1	1	0	1	7
x_4	0	5	0	1	-3	5
x_1	1	-1	0	0	1	3
z	0	-5	0	0	3	9

   New basic variables: x_1, x_3, x_4.
4. Find pivot column: only x_2 has negative coefficient. Find ratio: $7/1 = 7, 5/5 = 1, 3/(-1) = -3$, ignore; so the ratio is 1. New tableau: Table 3.

Table 3: Tableau T3

basic variables	x_1	x_2	x_3	x_4	x_4	$\bar{b}$
x_3	0	0	1	-0.2	1.6	6
x_2	0	1	0	0.2	-0.6	1
x_1	1	0	0	0.2	0.4	4
z	0	0	0	1	0	14

5. All the coefficients in the last line are nonnegative, so the optimum is found: $\mathbf{x} = (4, 1, 6, 0, 0)$, obj $= 14$.

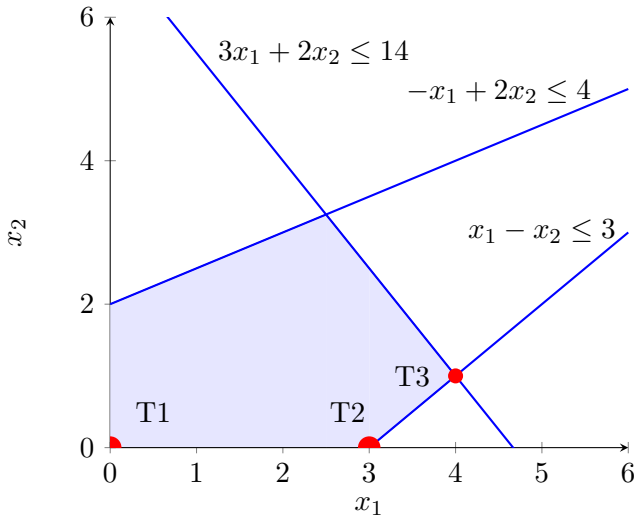

Figure 1: Problem 1 visualization

The feasible region of Problem 1 is shown in Figure 1. The solution points corresponding to simplex tableaus are highlighted in red dots.

Problem 2 Linear regression

This problem is a classic linear regression problem. The difference between this course and 6745 is that here we treat it as an optimization problem, and show the relationship between different formulations and different accuracy metrics.

Model variables:

- a, b: coefficients to be regressed on;
- $\hat{y}_i$ (optional): predicted y values.

(a) The NLP formulation is straightforward: we want to minimize SSE:

$$\begin{aligned} \min \quad & \sum_{i \in \{1,\dots,11\}} (\hat{y}_i - y_i)^2 \\ \text{s.t. } & \hat{y}_i = T_i \cdot a + b, \qquad i \in \{1, \dots, 11\}. \end{aligned}$$

We can also skip $\hat{y}_i$ definition and have an unconstrained problem:

$$\min \sum_{i \in \{1,\dots,11\}} (T_i \cdot a + b - y_i)^2.$$

The Python code for this part is as follows:

```
import numpy as np
from pyomo.environ import *
from pyomo.opt import SolverStatus, TerminationCondition

m = ConcreteModel()

T = np.linspace(90,110,11)
y = [0.045, 0.115, 0.45, 0.175, 0.205, 0.235, 0.325, 0.325, 0.2, 0.455, 0.5]

# define variables
m.a = Var(within=Reals)
m.b = Var(within=Reals)

# define the nonlinear objective function
def obj_rule(m):
    return sum((m.a*Ti + m.b - yi)**2 for Ti, yi in zip(T, y))
    m.obj = Objective(rule=obj_rule, sense=minimize)

# solve the model
solver = SolverFactory('ipopt')
results = solver.solve(m)

# print results
print(f"Termination condition: {results.solver.termination_condition}")
print(f"a: {value(m.a):.4f}")
print(f"b: {value(m.b):.4f}")
print(f"optimal SSE: {value(m.obj):.4f}")
```

The output is as follows:

```
Termination condition: optimal
a: 0.0150
b: -1.2268
optimal SSE: 0.1200
```

(b) The objective LP formulation is to minimize the sum of absolute errors:

$$\min \sum_{i\in\{1,\dots,11\}} |\hat{y}_i - y_i| \,.$$

There are two techniques to transform absolute values into pure linear constraints:

i. Use two constraints and inequalities. The absolute value function can be written as $f(x) = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$, which can be seen as a convex piecewise linear function. Also we want to minimize the absolute errors in the objective. Similar to Problem 2 in HW 1, we can use two constraints and inequalities to represent this piecewise function.

Extra variable:

- z_i: The absolute errors.

Model formulation:

$$\begin{aligned} \min \quad & \sum_{i\in\{1,\dots,11\}} z_i \\ \text{s.t.} \quad & z_i \geq \hat{y}_i - y_i, && i \in \{1,\dots,11\} \\ & z_i \geq -(\hat{y}_i - y_i), && i \in \{1,\dots,11\} \\ & \hat{y}_i = T_i \cdot a + b, && i \in \{1,\dots,11\} \,. \end{aligned}$$

The Python code for this part is as follows:

```
import numpy as np
from pyomo.environ import *
from pyomo.opt import SolverStatus, TerminationCondition

m = ConcreteModel()

I = list(range(11))
T = np.linspace(90,110,11)
y = [0.045, 0.115, 0.45, 0.175, 0.205, 0.235, 0.325, 0.325, 0.2, 0.455, 0.5]

# define variables
m.a = Var(within=Reals)
m.b = Var(within=Reals)
m.z = Var(I, within=NonNegativeReals)

# define constraints
def con1(m, i):
    return m.z[i] >= m.a*T[i] + m.b - y[i]
m.con1 = Constraint(I, rule=con1)
def con2(m, i):
    return m.z[i] >= - m.a*T[i] - m.b + y[i]
```

```
m.con2 = Constraint(I, rule=con2)

# define the objective function
def obj_rule(m):
    return sum(m.z[i] for i in I)
m.obj = Objective(rule=obj_rule, sense=minimize)

# solve the model
solver = SolverFactory('glpk')
results = solver.solve(m)

# print results
print(f"Termination condition: {results.solver.termination_condition}")
print(f"a: {value(m.a):.4f}")
print(f"b: {value(m.b):.4f}")
print(f"optimal sum of absolute errors: {value(m.obj):.4f}")
```

ii. Use two variables. Another way to represent the absolute error is to introduce two nonnegative variables for a single error variable.
Extra variables:

- z_i^+, z_i^-: a set of nonnegative variables to represent z_i and absolute values:

$$\begin{cases} z_i = z_i^+ - z_i^- \\ |z_i| = z_i^+ + z_i^- \end{cases}.$$

The problem can be formulated as

$$\begin{aligned} \min \quad & \sum_{i\in\{1,\dots,11\}} z_i^+ + z_i^- \\ \text{s.t. } & z_i^+ - z_i^- = \hat{y}_i - y_i, \quad i \in \{1,\dots,11\} \\ & \hat{y}_i = T_i \cdot a + b, \qquad i \in \{1,\dots,11\}. \end{aligned}$$

Our intention is to have $z_i = z_i^+$ when it is positive and $z_i = -z_i^-$ when it is negative, and it is done by minimizing the summation of z_i^+ and z_i^-. At an optimal solution, and for each i, we must have either $z_i^+ = 0$ or $z_i^- = 0$, as otherwise we could reduce both terms by the same amount and improve the solution, which contradicts with the solution being optimal. Therefore, when $z_i^+ = 0$, $z_i = -z_i^-$, and in the objective function we have $z_i^+ + z_i^- = -z_i = |z_i|$; similarly we know $z_i^+ + z_i^- = |z_i|$ when $z_i^- = 0$.
The Python code for this part is as follows:

```
import numpy as np
from pyomo.environ import *
from pyomo.opt import SolverStatus, TerminationCondition

m = ConcreteModel()

I = list(range(11))
T = np.linspace(90,110,11)
y = [0.045, 0.115, 0.45, 0.175, 0.205, 0.235, 0.325, 0.325, 0.2, 0.455, 0.5]
```

```
# define variables
m.a = Var(within=Reals)
m.b = Var(within=Reals)
m.zplus = Var(I, within=NonNegativeReals)
m.zminus = Var(I, within=NonNegativeReals)

# define constraints
def con1(m, i):
    return m.zplus[i] - m.zminus[i] == m.a*T[i] + m.b - y[i]
m.con1 = Constraint(I, rule=con1)

# define the objective function
def obj_rule(m):
    return sum(m.zplus[i] + m.zminus[i] for i in I)
m.obj = Objective(rule=obj_rule, sense=minimize)

# solve the model
solver = SolverFactory('glpk')
results = solver.solve(m)

# print results
print(f"Termination condition: {results.solver.termination_condition}")
print(f"a: {value(m.a):.4f}")
print(f"b: {value(m.b):.4f}")
print(f"optimal sum of absolute errors: {value(m.obj):.4f}")
```

The results from two methods are the same:

```
Termination condition: optimal
a: 0.0227
b: -2.0025
optimal sum of absolute errors: 0.6595
```

The regression models are shown in Figure 2.

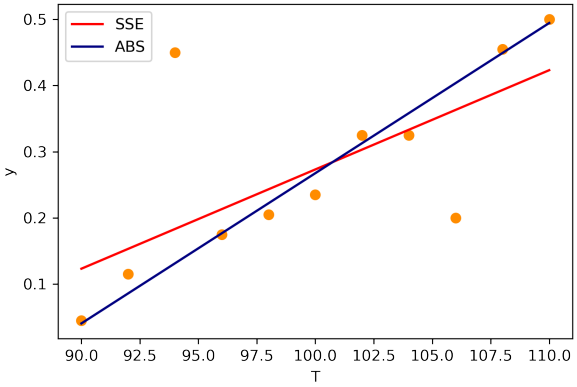

Figure 2: Problem 2 regression models

Problem 3 Characterizing convexity

Generally speaking, there are two ways to check if a function is convex:

1. calculate the eigenvalues of its Hessian matrix to see if the Hessian is positive (semi-)definite;
2. use the preservation rules of convexity: summation of two convex functions; positive scaling of convex functions; composition of convex function and monotonically increasing function.

The solution below follows the first or the second way (or their hybrid) to identify convexity. It is possible that other methods exists, and they should receive full credits as long as the logic and the conclusion are correct.

(a)

$$f(x) = e^{-x^2}$$

i. calculate first-order derivative:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = e^{-x^2} \cdot (-2x) = -2xe^{-x^2}$$

ii. calculate second-order derivative:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = -2e^{-x^2} - 2xe^{-x^2} \cdot (-2x)$$

Since this is a 1-D function, the eigenvalue of its Hessian is $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ itself.

iii. factor:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = -2e^{-x^2}(1 - 2x^2)$$

iv. calculate roots of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$:

$$(1 - 2x^2) = 0 \Longrightarrow x = \pm 1/\sqrt{2}$$

v. Is there a sign change over domain of $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$?

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \begin{cases} \geq 0, & x \in (-\infty, -1/\sqrt{2}] \cup [1/\sqrt{2}, \infty) \\ \leq 0, & x \in [-1/\sqrt{2}, 1/\sqrt{2}] \end{cases}.$$

As the eigenvalue of the Hessian is not always positive, the function is **nonconvex**.

(b)

$$f(x_1, x_2) = x_1^2 - 4x_1x_2 + 4x_2^2 - \ln(x_1x_2), \quad x_1, x_2 > 0$$

i. factor and use properties of natural log:

$$f(x_1, x_2) = -\ln(x_1) - \ln(x_2) + (x_1 - 2x_2)^2$$

ii. $-\ln(x_i)$ is convex in terms of x_1 and x_2, as

$$H(-\ln(x_1)) = \begin{bmatrix} \frac{1}{x_1^2} & 0 \\ 0 & 0 \end{bmatrix}, H(-\ln(x_2)) = \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{x_2^2} \end{bmatrix},$$

both of which are positive semi-definite.

iii. $(x_1 - 2x_2)^2$ is also convex:

$$H((x_1 - 2x_2)^2) = \begin{bmatrix} 2 & -4 \\ -4 & 8 \end{bmatrix}, \lambda_1 = 0, \lambda_2 = 10,$$

thus the matrix is positive semi-definite.

iv. As each part is convex, the overall f is also convex.

(c)

$$f(x_1, x_2, x_3) = e^{x_1^2 + x_2^2 + x_3^2}.$$

It is helpful to think of this one as a composite function:

$$f(x_1, x_2, x_3) = g(f'(x_1, x_2, x_3)), \text{ where } g(x) = e^x, f'(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2.$$

i. g is monotonically increasing, as

$$\frac{\mathrm{d}g}{\mathrm{d}x} = \frac{\mathrm{d}e^x}{\mathrm{d}x} = e^x > 0 \, \forall \, x \in \mathbb{R}.$$

ii. f' is convex, as its Hessian

$$H([x_1^2 + x_2^2 + x_3^2]) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

has positive eigenvalues (all 2).

iii. Composite function is thus **convex**.

Problem 4

$$\begin{aligned} \min\ & f(x_1, x_2) = 1 - 2x_1 - 4x_1x_2 \\ \text{s.t. } & x_1 + 4x_1x_2 \leq 4 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(a) The convexity of an optimization problem is determined on the convexity of both its objective function and its constraints. The problem is convex only if both the objective and the constraints are convex.
The Hessian of the objective function is

$$H(f) = \begin{bmatrix} 0 & -4 \\ -4 & 0 \end{bmatrix},$$

whose eigenvalues are 4 and -4. As $H(f)$ is not positive (semi-)definite, f is not convex, thus the problem is nonconvex.

(b) The Lagrange function is

$$\mathcal{L}(x_1, x_2, \mu_1, \mu_2, \mu_3) = 1 - 2x_1 - 4x_1x_2 + \mu_1(-4 + x_1 + 4x_1x_2) + \mu_2(-x_1) + \mu_3(-x_2).$$

The KKT conditions are

$$\begin{aligned} &\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_1} = -2 - 4x_2 + \mu_1 - \mu_2 + 4\mu_1x_2 = 0 && \text{(stationary point condition)} \\ &\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_2} = -4x_1 + 4\mu_1x_1 - \mu_3 = 0 && \text{(stationary point condition)} \\ &x_1 + 4x_1x_2 \leq 4 && \text{(feasibility)} \\ &x_1 \geq 0 && \text{(feasibility)} \\ &x_2 \geq 0 && \text{(feasibility)} \\ &\mu_1(4 + x_1 + 4x_1x_2) = 0 && \text{(complementary constraints)} \\ &\mu_2(-x_1) = 0 && \text{(complementary constraints)} \\ &\mu_3(-x_2) = 0 && \text{(complementary constraints)} \\ &\mu_1, \mu_2, \mu_3 \geq 0 && \text{(complementary constraints).} \end{aligned}$$

(c) The rewriting is straightforward: as $x_1, x_2 \geq 0, y_1 \geq 0$, and $y_2 = x_1x_2 \geq 0$. The LP formulation is

$$\begin{aligned} \min\ & 1 - 2y_1 - 4y_2 \\ \text{s.t. } & y_1 + 4y_2 \leq 4 \\ & y_1, y_2 \geq 0. \end{aligned}$$

The python code is attached below just for reference (we do not require code submission for this part).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 19 18:37:29 2021

@author: zkilwein
"""
from pyomo.environ import *

## linearized Problem

model = ConcreteModel()

model.y1 = Var(domain=NonNegativeReals)
model.y2 = Var(domain=NonNegativeReals)

# declare objective
model.obj = Objective(expr = 1-2*model.y1 -4*model.y2,sense = minimize)

# declare constraints
model.con1 = Constraint(expr = model.y1+4*model.y2 <= 4)

SolverFactory('glpk').solve(model)

print('y1 = '+ str(value(model.y1)))
print('y2 = '+ str(value(model.y2)))
print('Objective Function = '+ str(value(model.obj)))
```

The solution of the LP problem is $y_1 = 4, y_2 = 0$ with optimal objective value -7, which is identical to the optimal solution of the nonconvex NLP problem ($x_1 = 4, x_2 = 0, \text{obj} = -7$). In this case, the reformulation allows us to solve an LP problem to obtain the global optimum of the original nonconvex NLP problem, which makes the solving process much easier.