Case Study 2

**Case 1 (Book problem 4-9, page: 222):**

Wobbly Office Equipment (WOE) makes two models of tables for libraries and other university facilities. Both models use the same tabletops, but model A has 4 short (18-inch) legs and model B has 4 longer ones (30-inch). It takes 0.10 labor hour to cut and shape a short leg from stock, 0.15 labor hour to do the same for a long leg, and 0.50 labor hour to produce a tabletop. An additional 0.30 labor hour is needed to attach the set of legs for either model after all parts are available. Estimated profit is $30 for each model A sold and $45 for each model B. Plenty of top material is on hand, but WOE wants to decide how to use the available 500 feet of leg stock and 80 labor hours to maximize profit, assuming that everything made can be sold.

(a) Formulate an operations management LP to choose an optimal plan using the decision variables x1: number of model A’s assembled and sold, x2: number of model B’s assembled and sold, x3: number of short legs manufactured, x4: number of long legs manufactured, and x5: number of tabletops manufactured.

(b) Which of the constraints of your model are balance constraints? Explain.

(c) Solve your model with the simplex method in two ways:

1. built-in python library linprog()
2. user-defined function.

***Solution:***

**Objective Function:**

Maximize, Z = 30\*x1 + 45\*x2

**Variables**

* x1: number of model A’s assembled and sold
* x2: number of model B’s assembled and sold
* x3: number of short legs manufactured
* x4: number of long legs manufactured
* x5: number of tabletops manufactured.

**(a) Constraints:**

1. Labor Hour Constraint:  
   0.1\*x3 + 0.15\*x4 + 0.5\*x5 + 0.3\*(x1 + x2) <= 80  
   0.3\*x1 + 0.3\*x2 + 0.1\*x3 + 0.15\*x4 + 0.5\*x5 <= 80 (Normalized form)
2. Leg stock Constraint: (inches converted into feet)  
   1.5\*x3 + 2.5\*x4 <= 500  
   0\*x1 + 0\*x2 + 1.5\*x3 + 2.5\*x4 + 0\*x5 <= 500 (Normalized form)
3. Short-leg to Model A Constraint:  
   x3 = 4\*x1  
   4\*x1 + 0\*x2 - x3 + 0\*x4 + 0\*x5 = 0 (Normalized form)
4. Long-leg to Model B Constraint:  
   x4 = 4\*x2  
   0\*x1 + 4\*x2 + - 0\*x3 - 1\*x4 + 0\*x5 = 0 (Normalized form)
5. Tabletops to Model A & B Constraint:  
   x5 = x1 + x2  
   x1 + x2 + 0\*x3 + 0\*x4 - x5 = 0 (Normalized form)
6. Non-negativity Constraints:  
   x1, x2, x3, x4, x5 >= 0

**(b)** The balance constraints of this solution is as follow:

1. x3 = 4\*x1
2. x4 = 4\*x2
3. x5 = x1 + x2

**(c)**

**1. Using linprog**

****

from scipy.optimize import linprog

# Matrix relating to constraints derieved

lhs\_ineq            = [[0.30, 0.30, 0.10, 0.15, 0.50], [0, 0, 1.5, 2.5, 0], [4, 0, -1, 0, 0], [0, 4, 0, -1, 0], [1, 1, 0, 0, -1]]

# RHS of contraints

rhs\_ineq            = [80, 500, 0, 0, 0]

# Objective Function

objective           = [-30, -45, 0, 0, 0]

x\_values            = [(0, None), (0, None), (0, None), (0, None), (0, None)]

result              = linprog(objective, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq, bounds=x\_values, method='simplex')

x1, x2, x3, x4, x5  = result.x

max\_profit          = -result.fun

print(f"Value of x1: {x1:.2f}")

print(f"Value of x2: {x2:.2f}")

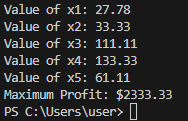
print(f"Value of x3: {x3:.2f}")

print(f"Value of x4: {x4:.2f}")

print(f"Value of x5: {x5:.2f}")

print(f"Maximum Profit: ${max\_profit:.2f}")

**Output Obtained**



**2. Using user-defined function**

****

import numpy as np

def simplex(c, A, b):

    tableau = to\_tableau(c, A, b)

    while can\_be\_improved(tableau):

        pivot\_position = get\_pivot\_position(tableau)

        tableau = pivot\_step(tableau, pivot\_position)

    return get\_solution(tableau)

def to\_tableau(c, A, b):

    xb = [eq + [x] for eq, x in zip(A, b)]

    z = c + [0]

    return xb + [z]

def can\_be\_improved(tableau):

    z = tableau[-1]

    return any(x > 0 for x in z[:-1])

def get\_pivot\_position(tableau):

    z = tableau[-1]

    column = next(i for i, x in enumerate(z[:-1]) if x > 0)

    restrictions = []

    for eq in tableau[:-1]:

        el = eq[column]

        restrictions.append(math.inf if el <= 0 else eq[-1] / el)

    row = restrictions.index(min(restrictions))

    return row, column

def pivot\_step(tableau, pivot\_position):

    new\_tableau = [[] for eq in tableau]

    i, j = pivot\_position

    pivot\_value = tableau[i][j]

    new\_tableau[i] = np.array(tableau[i]) / pivot\_value

    for eq\_i, eq in enumerate(tableau):

        if eq\_i != i:

            multiplier = np.array(new\_tableau[i]) \* tableau[eq\_i][j]

            new\_tableau[eq\_i] = np.array(tableau[eq\_i]) - multiplier

    return new\_tableau

def is\_basic(column):

    return sum(column) == 1 and len([c for c in column if c == 0]) == len(column) - 1

def get\_solution(tableau):

    columns = np.array(tableau).T

    solutions = []

    for column in columns[:-1]:

        solution = 0

        if is\_basic(column):

            one\_index = column.tolist().index(1)

            solution = columns[-1][one\_index]

        solutions.append(solution)

    return solutions

c = [-30, -45, 0, 0, 0]

A = [

            [0.30, 0.30, 0.10, 0.15, 0.50],  # assembly time

            [0, 0, 1.5, 2.5, 0], # total Leg feet

            [4, 0, -1, 0, 0], # short leg

            [0, 4, 0, -1, 0], # long leg

            [1, 1, 0, 0, -1] # total number of table tops

           ]

b = [80, 500, 0, 0, 0]

solution = simplex(c, A, b)

print(f"solution - {solution}")

**Case 2 (Book problem 4-10, page: 222):**

Perfect Stack builds standard and extra long wooden palettes for a variety of manufacturers. Each model consists of 3 heavy separators of length equal to the palette. The standard model has 5 cross pieces above and 5 below the separators and requires 0.25 hour to assemble. The extralong version has 9 similar cross pieces on top and bottom and consumes 0.30 hour to assemble. The supply of wood is essentially unlimited, but it requires 0.005 hour to fabricate a standard separator, 0.007 hour to fabricate an extralong separator, and 0.002 hour to fabricate a cross piece. Assuming that it can sell as many standard models as can be made at $5 profit each and as many extralongs at $7 profit, Perfect wants to decide what to produce with the available 200 hours of assembly time and 40 hours of fabrication.

(a) Formulate an operations management LP to choose an optimal plan using the decision variables x1: number of standard palettes assembled and sold, x2: number of extralongs assembled and sold, x3: number of standard separators manufactured, x4: number of extralong separators manufactured, and x5: number of cross pieces manufactured.

(b) Which of the constraints of your model are balance constraints? Explain.

(c) Solve your model with the simplex method in two ways:

1. built-in python library linprog()

2. user-defined function.

***Solution:***

**Objective Function:**

Maximize, Z = 5\*x1 + 7\*x2

**Variables**

* x1: number of standard palettes assembled and sold
* x2: number of extralongs assembled and sold
* x3: number of standard separators manufactured
* x4: number of extralong separators manufactured
* x5: number of cross pieces manufactured

**(a) Constraints:**

1. Assembly Time Constraint:  
   0.25\*x1 + 0.30\*x2 ≤ 200  
   0.25\*x1 + 0.3\*x2 + 0\*x3 + 0\*x4 + 0\*x5 ≤ 200 (Normalized form)
2. Fabrication Time Constraint:  
   0.005\*x3 + 0.007\*x4 + 0.002\*x5 ≤ 40   
   0\*x1 + 0\*x2 + 0.005\*x3 + 0.007\*x4 + 0.002\*x5 ≤ 40 (Normalized form)
3. Separator Balance Constraint for Standard:  
   3\*x1 = x3  
   3\*x1 + 0\*x2 - x3 + 0\*x4 + 0\*x5 = 0 (Normalized form)
4. Separator Balance Constraint for extra-long:  
   3\*x2 = x4  
   0\*x1 + 3\*x2 + 0\*x3 - x4 + 0\*x5 = 0 (Normalized form)
5. Cross Piece Balance Constraint for Standard:  
   10\*x1 + 18\*x2 = x5  
   10\*x1 + 18\*x2 + 0\*x3 + 0\*x4 - x5 = 0 (Normalized form)
6. Non-negativity Constraints:  
   x1, x2, x3, x4, x5 ≥ 0

**(b)** The balance constraints are as follows:

1. 3\*x1 = x3
2. 3\*x2 = x4
3. 10\*x1 + 18\*x2 = x5

**(c)**

**1. Using linprog**

****

from scipy.optimize import linprog

# Matrix relating to constraints derieved

lhs\_ineq            = [[0.25, 0.30, 0, 0, 0], [0.005, 0.007, 0.002, 0, 0], [-3, 0, 1, 0, 0], [0, -3, 0, 1, 0], [0, -18, 0, 0, 1]]

# RHS of contraints

rhs\_ineq           = [200, 40, 0, 0, 0]

# Objective Function

objective           = [-5, -7, 0, 0, 0]

x\_bounds            = [(0, None), (0, None), (0, None), (0, None), (0, None)]

result              = linprog(objective, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq, bounds=x\_bounds, method='simplex')

x1, x2, x3, x4, x5  = result.x

max\_profit          = -result.fun

print(f"Value of x1: {x1:.2f}")

print(f"Value of x2: {x2:.2f}")

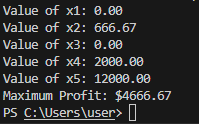
print(f"Value of x3: {x3:.2f}")

print(f"Value of x4: {x4:.2f}")

print(f"Value of x5: {x5:.2f}")

print(f"Maximum Profit: ${max\_profit:.2f}")

**Output Obtained**

****

**2. Using user-defined function**

****

import numpy as np

def simplex(c, A, b):

    tableau = to\_tableau(c, A, b)

    while can\_be\_improved(tableau):

        pivot\_position = get\_pivot\_position(tableau)

        tableau = pivot\_step(tableau, pivot\_position)

    return get\_solution(tableau)

def to\_tableau(c, A, b):

    xb = [eq + [x] for eq, x in zip(A, b)]

    z = c + [0]

    return xb + [z]

def can\_be\_improved(tableau):

    z = tableau[-1]

    return any(x > 0 for x in z[:-1])

def get\_pivot\_position(tableau):

    z = tableau[-1]

    column = next(i for i, x in enumerate(z[:-1]) if x > 0)

    restrictions = []

    for eq in tableau[:-1]:

        el = eq[column]

        restrictions.append(math.inf if el <= 0 else eq[-1] / el)

    row = restrictions.index(min(restrictions))

    return row, column

def pivot\_step(tableau, pivot\_position):

    new\_tableau = [[] for eq in tableau]

    i, j = pivot\_position

    pivot\_value = tableau[i][j]

    new\_tableau[i] = np.array(tableau[i]) / pivot\_value

    for eq\_i, eq in enumerate(tableau):

        if eq\_i != i:

            multiplier = np.array(new\_tableau[i]) \* tableau[eq\_i][j]

            new\_tableau[eq\_i] = np.array(tableau[eq\_i]) - multiplier

    return new\_tableau

def is\_basic(column):

    return sum(column) == 1 and len([c for c in column if c == 0]) == len(column) - 1

def get\_solution(tableau):

    columns = np.array(tableau).T

    solutions = []

    for column in columns[:-1]:

        solution = 0

        if is\_basic(column):

            one\_index = column.tolist().index(1)

            solution = columns[-1][one\_index]

        solutions.append(solution)

    return solutions

c = [-5, -7, 0, 0, 0]

A = [[0.25, 0.30, 0, 0, 0], [0.005, 0.007, 0.002, 0, 0], [-3, 0, 1, 0, 0], [0, -3, 0, 1, 0], [0, -18, 0, 0, 1]]

b = [200, 40, 0, 0, 0]

solution = simplex(c, A, b)

print(f"solution - {solution}")