**OPTIMIZATION ASSIGNMENT**

Laxman Babu K

PGDABI01202116285

**1. Consider a motor manufacturing company that produces only two types of bike – A and B. Both the bikes**

**Require M and C instruments. To manufacture each unit of A and B, the following units are required:**

**• Each unit of A requires 1 unit of M and 3 units of C**

**• Each unit of B requires 1 unit of M and 2 units of C**

**The company has a total of 5 units of M and 12 units of C. On each sale, the company makes a profit of**

**• Rs 25 per unit A sold**

**• Rs 55 per unit B sold.**

**Now, the company wishes to maximize its profit. How many units of A and B should it produce respectively?**

**Write a python program using solver of your choice.**

X1 = Number of units of A manufactured.

X2 = Number of units of B manufactured.

The objective function: Max Z = 25X1+ 55x2

Subjective Constraints For manufactured A x1+ x2 ≤ 5 For manufactured B 3x1+ 2x2 ≤ 12 Non negativity x1, x2 ≥0

**CODE:**

%matplotlib inline

# pulp is one such library which is used for the optimization of linear programming

# import the library pulp

#!pip install pulp

import pulp as p

import matplotlib.pyplot as plt #it is used for creating scatter line bars, charts, figure, scatter plots etc.

import numpy as np # a whole-some package for performing basic scientific operations

Lp\_prob = p.LpProblem('Problem\_1\_Max\_Profite', p.LpMaximize)

    # Create problem Variables

X1 = p.LpVariable("X1", lowBound = 0)   # Create a variable x >= 0

X2 = p.LpVariable("X2", lowBound = 0)   # Create a variable y >= 0

# Objective Function

Lp\_prob += 25 \* X1 + 55 \* X2   # x, and y are defined variables

# Constraints:

Lp\_prob += 1\* X1 + 1 \*X2 <=5

Lp\_prob += 3\*X1 + 2\*X2 <= 12

# Display the problem

print(Lp\_prob)

status = Lp\_prob.solve()   # Calling the default Solver  (CBC - coin or branch and cut)

print(p.LpStatus[status])   # The solution status  if 1-optimal, 0- no solution

# Printing the final solution

print(p.value(X1), p.value(X2), p.value(Lp\_prob.objective))

**OUTPUT**:

Problem\_1\_Max\_Profite:

MAXIMIZE

25\*X1 + 55\*X2 + 0

SUBJECT TO

\_C1: X1 + X2 <= 5

\_C2: 3 X1 + 2 X2 <= 12

VARIABLES

X1 Continuous

X2 Continuous

Optimal

0.0 5.0 275.0

**Q2. Minimize:**2x12+ x22+ x1 x2+x1 +x2

**Subject to:** x1≥ 0, x1≥ 0, x1 +x2=1

**Rewriting the above set of equation in the standard format of quadratic programing**

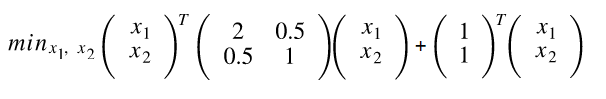
**Standard form of Quadratic Programing**

minx : XTPx+QTx

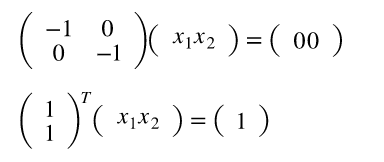
***Subject to:*** Gx≤ h, Ax= b

Standard notation is followed. Matrix are represented with capital letter while vector are denoted with bold small letter.

**The above equation can be represented as**



**The first matrix is the Q matrix while the second one is p. Now we have to define our G,h, A, and b respectively**



**CODE**:

# Now arranging the above in the matrix format for the program.

Q = 2\*matrix([ [2, .5], [.5, 1] ])

p = matrix([1.0, 1.0])

G = matrix([[-1.0,0.0],[0.0,-1.0]])

h = matrix([0.0,0.0])

A = matrix([1.0, 1.0], (1,2))

b = matrix(1.0)

sol=solvers.qp(Q, p, G, h, A, b)

print(sol['x'])

**OUTPUT:**

pcost dcost gap pres dres

0: 1.8889e+00 7.7778e-01 1e+00 3e-16 2e+00

1: 1.8769e+00 1.8320e+00 4e-02 2e-16 6e-02

2: 1.8750e+00 1.8739e+00 1e-03 2e-16 5e-04

3: 1.8750e+00 1.8750e+00 1e-05 6e-17 5e-06

4: 1.8750e+00 1.8750e+00 1e-07 2e-16 5e-08

Optimal solution found.

[ 2.50e-01]

[ 7.50e-01]

# **3. Solve the ILP problem: Write a python code using the library PuLP or solver of your choice**

Min: x0 + x1 + x2 + x3 + x4 + x5

Given the following constraints:

x0 + x1 ≥ 1

x0 + x1 + x5 ≥ 1

x2 + x3 ≥ 1

x2 + x3 + x4 ≥ 1

x3 + x4 + x5 ≥ 1

x1 + x4 + x5 ≥ 1

x0, x1, x2, x3, x4, x5 ∈ Z

**CODE:**

#command to install pulp library

!pip install pulp

#Importing all the required libraries

%matplotlib inline

import pulp as p

import matplotlib.pyplot as plt #it is used for creating scatter line bars, charts, figure, scatter plots etc.

import numpy as np # a whole-some package for performing basic scientific operations.

Lp\_prob = p.LpProblem('Cashflow\_Problem', p.LpMinimize)

    # Create problem Variables

x = p.LpVariable("x", lowBound = 0)   # Create a variable x >= 0

y = p.LpVariable("y", lowBound = 0)   # Create a variable y >= 0

z = p.LpVariable("z", lowBound = 0)   # Create a variable z >= 0

w = p.LpVariable("w", lowBound = 0)   # Create a variable w >= 0

v = p.LpVariable("v", lowBound = 0)   # Create a variable v >= 0

# Objective Function

Lp\_prob += 1 \* x + 1 \* y + 1 \* z + 1 \* w + 1 \* v  # x, and y are defined variables

# Constraints:

Lp\_prob += 1\*x + 1\*y >= 1

Lp\_prob += 1\*x + 1\*y + 1\*v >= 1

Lp\_prob += 1\*y + 1\*z >= 1

Lp\_prob += 1\*y + 1\*z + 1\*w >= 1

Lp\_prob += 1\*z + 1\*w + 1\*v >= 1

Lp\_prob += 1\*y + 1\*w + 1\*v >= 1

# Display the problem

print(Lp\_prob)

status = Lp\_prob.solve()   # Calling the default Solver  (CBC - coin or branch and cut)

print(p.LpStatus[status])   # The solution status  if 1-optimal, 0- no solution

# Printing the final solution

print(p.value(x), p.value(y), p.value(z), p.value(w),p.value(v),p.value(Lp\_prob.objective))

**OUTPUT:**

Cashflow\_Problem:

MINIMIZE

1\*v + 1\*w + 1\*x + 1\*y + 1\*z + 0

SUBJECT TO

\_C1: x + y >= 1

\_C2: v + x + y >= 1

\_C3: y + z >= 1

\_C4: w + y + z >= 1

\_C5: v + w + z >= 1

\_C6: v + w + y >= 1

VARIABLES

v Continuous

w Continuous

x Continuous

y Continuous

z Continuous

Optimal

0.5 0.5 0.5 0.5 0.0 2.0

# **4. a) Compare the feasible solutions of the three following integer linear programs using python programming.**

1) Max: 14x1 + 8x2 + 6x3 + 6x4

Subject to: 28x1 + 15x2 + 13x3 + 12x4 ≤ 39

x1, x2, x3, x4 ∈ {0, 1} 1

2) Max: 14x1 + 8x2 + 6x3 + 6x4

Subject to: 2x1 + x2 + x3 + x4 ≤ 2

x1, x2, x3, x4 ∈ {0, 1}

3) Max: 14x1 + 8x2 + 6x3 + 6x4

Subject to: x2 + x3 + x4 ≤ 2

x1 + x2 ≤ 1 x1 + x3 ≤ 1

x1 + x4 ≤ 1

x1, x2, x3, x4 ∈ {0, 1}

Write python code to compare the relaxations of the above integer programs obtained by replacing x1, x2, x3, x4 ∈ 0, 1 by 0 ≤ xj ≤ 1 for j =1,2,3,4. Which is the best among 1),2),3) for obtaining a tight bound from linear programming relaxation ?.

***4.1) Max: 14x1 + 8x2 + 6x3 + 6x4***

***Subject to: 28x1 + 15x2 + 13x3 + 12x4 ≤ 39***

***x1, x2, x3, x4 ∈ {0, 1} 1***

**CODE:**

import pulp as p

model = p.LpProblem('Problem', p.LpMaximize)

# Define the decision variables

x = {i: p.LpVariable(name=f"x{i}", lowBound=0) for i in range(1, 5)}

print(x)

# Add constraints

#model += (p.lpSum(x.values()) <= 39, "Problem -1")

model += (28 \* x[1] + 15 \* x[2] + 13\* x[3] + 12 \* x[4]<= 39, "Problem -1")

# Set the objective

model += 14 \* x[1] + 8 \* x[2] + 6 \* x[3] + 6 \* x[4]

# Solve the optimization problem

status = model.solve()

print(model)

# Get the results

print(f"status: {model.status}, {p.LpStatus[model.status]}")

print(f"objective: {model.objective.value()}")

for var in x.values():

    print(f"{var.name}: {var.value()}")

for name, constraint in model.constraints.items():

    print(f"{name}: {constraint.value()}")

**OUTPUT:**

{1: x1, 2: x2, 3: x3, 4: x4}

Problem:

MAXIMIZE

14\*x1 + 8\*x2 + 6\*x3 + 6\*x4 + 0

SUBJECT TO

Problem\_\_1: 28 x1 + 15 x2 + 13 x3 + 12 x4 <= 39

VARIABLES

x1 Continuous

x2 Continuous

x3 Continuous

x4 Continuous

status: 1, Optimal

objective: 20.8

x1: 0.0

x2: 2.6

x3: 0.0

x4: 0.0

Problem\_\_1: 0.0

***4.2) Max: 14x1 + 8x2 + 6x3 + 6x4***

***Subject to: 2x1 + x2 + x3 + x4 ≤ 2***

**CODE:**

#import pulp as p

model = p.LpProblem('Problem', p.LpMaximize)

# Define the decision variables

x = {i: p.LpVariable(name=f"x{i}", lowBound=0) for i in range(1, 5)}

print(x)

# Add constraints

model += (2 \* x[1] + 1 \* x[2] + 1\* x[3] + 1 \* x[4]<= 2, "Problem -2")

# Set the objective

model += 14 \* x[1] + 8 \* x[2] + 6 \* x[3] + 6 \* x[4]

# Solve the optimization problem

status = model.solve()

print(model)

# Get the results

print(f"status: {model.status}, {p.LpStatus[model.status]}")

print(f"objective: {model.objective.value()}")

for var in x.values():

    print(f"{var.name}: {var.value()}")

for name, constraint in model.constraints.items():

    print(f"{name}: {constraint.value()}")

**OUTPUT:**

{1: x1, 2: x2, 3: x3, 4: x4}

Problem:

MAXIMIZE

14\*x1 + 8\*x2 + 6\*x3 + 6\*x4 + 0

SUBJECT TO

Problem\_\_2: 2 x1 + x2 + x3 + x4 <= 2

VARIABLES

x1 Continuous

x2 Continuous

x3 Continuous

x4 Continuous

status: 1, Optimal

objective: 16.0

x1: 0.0

x2: 2.0

x3: 0.0

x4: 0.0

Problem\_\_2: 0.0

***4.3) Max: 14x1 + 8x2 + 6x3 + 6x4***

***Subject to: x2 + x3 + x4 ≤ 2***

x1 + x2 ≤ 1 x1 + x3 ≤ 1

x1 + x4 ≤ 1

x1, x2, x3, x4 ∈ {0, 1}

**CODE**:

#import pulp as p

model = p.LpProblem('Problem', p.LpMaximize)

# Define the decision variables

x = {i: p.LpVariable(name=f"x{i}", lowBound=0) for i in range(1, 5)}

print(x)

# Add constraints

model += ( 1 \* x[2] + 1\* x[3] + 1 \* x[4]<= 2, "S1")

model += (1 \* x[1] + 1 \* x[2] <= 1, "S2")

model += (1 \* x[1] + 1\* x[3] <= 1, "S3")

model += (1 \* x[1] + 1\* x[4] <= 1, "S4")

# Set the objective

model += 14 \* x[1] + 8 \* x[2] + 6 \* x[3] + 6 \* x[4]

# Solve the optimization problem

status = model.solve()

print(model)

# Get the results

print(f"status: {model.status}, {p.LpStatus[model.status]}")

print(f"objective: {model.objective.value()}")

for var in x.values():

    print(f"{var.name}: {var.value()}")

for name, constraint in model.constraints.items():

    print(f"{name}: {constraint.value()}")

**OUTPUT**:

{1: x1, 2: x2, 3: x3, 4: x4}

Problem:

MAXIMIZE

14\*x1 + 8\*x2 + 6\*x3 + 6\*x4 + 0

SUBJECT TO

S1: x2 + x3 + x4 <= 2

S2: x1 + x2 <= 1

S3: x1 + x3 <= 1

S4: x1 + x4 <= 1

VARIABLES

x1 Continuous

x2 Continuous

x3 Continuous

x4 Continuous

status: 1, Optimal

objective: 18.00000002

x1: 0.33333333

x2: 0.66666667

x3: 0.66666667

x4: 0.66666667

S1: 1.0000000161269895e-08

S2: 0.0

S3: 0.0

S4: 0.0

# **Results**

Best Solution is 1) Max: 14x1 + 8x2 + 6x3 + 6x4 Subject to: 28x1 + 15x2 + 13x3 + 12x4 ≤ 39 x1, x2, x3, x4 ∈ {0, 1}

# **Q5. Solve the following problem using python programming. Hint:- use cvxopt**

Minx,y : x 2 + 3x + 4y

Subject to:

x, y ≥ 0 x + 3y ≥ 15

2x + 5y ≤ 100 3x + 4y ≤ 80

**CODE**:

from cvxopt import matrix

P = matrix([[1.0,0.0],[0.0,0.0]])

q = matrix([3.0,4.0])

G = matrix([[-1.0,0.0,-1.0,2.0,3.0],[0.0,-1.0,-3.0,5.0,4.0]])

h = matrix([0.0,0.0,-15.0,100.0,80.0])

import numpy

from cvxopt import matrix

P = matrix(numpy.diag([1,0]), tc='d')

q = matrix(numpy.array([3,4]), tc='d')

G = matrix(numpy.array([[-1,0],[0,-1],[-1,-3],[2,5],[3,4]]), tc='d')

h = matrix(numpy.array([0,0,-15,100,80]), tc='d')

from cvxopt import solvers

sol = solvers.qp(P,q,G,h)

#sol = solvers.qp(P,q,G,h,A,b)

print(sol['x'])

**OUTPUT:**

pcost dcost gap pres dres

0: 1.0780e+02 -7.6366e+02 9e+02 0e+00 4e+01

1: 9.3245e+01 9.7637e+00 8e+01 6e-17 3e+00

2: 6.7311e+01 3.2553e+01 3e+01 2e-16 1e+00

3: 2.6071e+01 1.5068e+01 1e+01 2e-16 7e-01

4: 3.7092e+01 2.3152e+01 1e+01 1e-16 4e-01

5: 2.5352e+01 1.8652e+01 7e+00 6e-17 3e-16

6: 2.0062e+01 1.9974e+01 9e-02 9e-17 2e-16

7: 2.0001e+01 2.0000e+01 9e-04 2e-16 1e-16

8: 2.0000e+01 2.0000e+01 9e-06 1e-16 3e-16

Optimal solution found.

[ 7.13e-07]

[ 5.00e+00]