# Solving Linear Program Optimization Problems Using Pyomo

```
!pip install -q pyomo
```

```
#Install Solvers
%%capture
import sys
import os

if 'google.colab' in sys.modules:
    !pip install idaes-pse --pre
    !idaes get-extensions --to ./bin
    os.environ['PATH'] += ':bin'
```

```
import pyomo.environ as pyo
```

## Problem 1:

- Red T-shirts sell for 24 dollars
- black t-shirts sell for 18 dollars.
- The production cost for each T-shirt is 8 dollars.

At most 500 shirts can be produced in any month at this rate.

However, t-shirts in excess of 500/month can be produced at a cost of 1.25 times the normal production costs.

Also, we can store inventory from one period to the next, but it costs $2.00 per t-shirt that we keep in inventory.

The demands are given in the following table:

```
Month:     T-shirt Demand   Red | Black
Month 1:                    250 | 200
Month 2:                    90  | 160
Month 3:                    250 | 450
Month 4:                    350 | 300
Month 5:                    320 | 200
Month 6:                    380 | 200
```

## Index

$t$ = Months (t = 1, 2, 3, 4, 5, 6)
$j$ = shirt colors (1 = red, 2 = black)

## Decision Variables

$x_{tj}$ = Number of t-shirt j produced in month t
$y_{tj}$ = Number of t-shirt color j in inventory at the end of month t
$s_{tj}$ = Number of t-shirts color j in excess of 500 t-shirts in month t
$z_{tj}$ = Number of t-shirts color j sold in month t

## Parameters

$d_{tj}$ = demand for t-shirt color j in month t $r_j$ = revenue for each t-shirt color j sold

```
TIME = range(1, 7)
COLORS = [1, 2]  # 1 = red, 2 = black
```

```
# Parameters
d = {
    (1, 1): 250, (1, 2): 200,
    (2, 1): 90,  (2, 2): 160,
    (3, 1): 250, (3, 2): 450,
    (4, 1): 350, (4, 2): 300,
    (5, 1): 320, (5, 2): 200,
    (6, 1): 380, (6, 2): 200
}
r = {1: 24, 2: 18}  # revenue per shirt
prod_cost = 8  # production cost per shirt
overtime_factor = 1.25  # overtime production cost factor
inventory_cost = 2  # cost per shirt in inventory
```

## Model

$$\boldsymbol{Max} \sum_{t=1}^{6} \sum_{j=1}^{2} (r_j z_{tj} - 8x_{tj} - (1.25 * 8)s_{tj} - 2y_{tj}$$

$\boldsymbol{s.t}$:

$$y_{1j} = x_{1j} + s_{1j} - z_{1j} \quad \forall j = 1,2$$

$$y_{tj} = y_{t-1} + x_{tj} + s_{tj} - z_{tj} \quad \forall t = 1,2, \dots ,6 \ \& \ \forall j = 1,2$$

$$z_{tj} \leq d_{tj} \quad \forall t = 1,2, \dots 6 \ \& \ \forall j = 1,2$$

$$\sum_{j=1}^{2} x_{tj} \leq 500 \quad \forall t = 1,2, \dots ,6$$

$$x_{tj}, s_{tj}, y_{tj}, z_{tj} \geq 0 \quad \forall t = 1,2, \dots ,6 \ \& \ \forall j = 1,2$$

```
m = pyo.ConcreteModel()
m.clear()

# Define decision variables
m.x = pyo.Var(TIME, COLORS, domain=pyo.NonNegativeReals)  # regular production
m.s = pyo.Var(TIME, COLORS, domain=pyo.NonNegativeReals)  # excess production
m.y = pyo.Var(TIME, COLORS, domain=pyo.NonNegativeReals)  # inventory
m.z = pyo.Var(TIME, COLORS, domain=pyo.NonNegativeReals)  # sales
```

## Define Objective and Constraints

```
# Objective function: maximize profit
def objective_rule(m):
    revenue = sum(r[j] * m.z[t,j] for t in TIME for j in COLORS)
    production_cost = sum(prod_cost * m.x[t,j] for t in TIME for j in COLORS)
    excess_production_cost = sum(overtime_factor * prod_cost * m.s[t,j] for t in TIME for j in
    inventory_holding_cost = sum(inventory_cost * m.y[t,j] for t in TIME for j in COLORS)
    return revenue - production_cost - excess_production_cost - inventory_holding_cost

m.obj = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Initial inventory is zero
def initial_inventory_rule(m, j):
    # Create a virtual period 0 inventory that equals 0
    return m.y[1,j] == m.x[1,j] + m.s[1,j] - m.z[1,j]
m.initial_inventory = pyo.Constraint(COLORS, rule=initial_inventory_rule)

# Inventory balance constraints for periods 2-6
def inventory_balance_rule(m, t, j):
    if t == 1:
        return pyo.Constraint.Skip
    return m.y[t,j] == m.y[t-1,j] + m.x[t,j] + m.s[t,j] - m.z[t,j]
m.inventory_balance = pyo.Constraint(TIME, COLORS, rule=inventory_balance_rule)

# Sales cannot exceed demand
def sales_limit_rule(m, t, j):
    return m.z[t,j] <= d[t,j]
m.sales_limit = pyo.Constraint(TIME, COLORS, rule=sales_limit_rule)

# Regular production limit (500 units total per month)
def regular_production_limit_rule(m, t):
    return sum(m.x[t,j] for j in COLORS) <= 500
m.regular_production_limit = pyo.Constraint(TIME, rule=regular_production_limit_rule)
```

## Solving the Model

```
#Declare the solver as CBC
opt = pyo.SolverFactory('cbc')
#Solve the model
opt.solve(m).write()
```

```
# ==========================================================
Solver Results
Problem:
- Name: unknown
  Lower bound: 40440.0
  Upper bound: 40440.0
  Number of objectives: 1
  Number of constraints:
  30 Number of variables:
  48 Number of nonzeros:
  36 Sense: maximize

Solver:
- Status: ok
  User time: -1.0
  System time: 0.0
  Wallclock time: 0.0
  Termination condition:
  optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an opt
  Statistics:
    Branch and bound:
      Number of bounded subproblems: None
      Number of created subproblems:
      None
Black box:
      Number of iterations: 27
  Error rc: 0
  Time: 0.01580333709716797

Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

# Optimal Solution

```
#Output the optimal objective value
print('Optimal objective value:',pyo.value(m.obj()))

    Optimal objective value: 40440.0

import pandas as pd

# Create an empty list to store data
data = []

# Iterate over TIME and COLORS to extract values
for t in TIME:
    for j in COLORS:
        data.append({
            'Month': t,
            'Color': j,
            'Regular Production': pyo.value(m.x[t, j]),
            'Excess Production': pyo.value(m.s[t, j]),
            'Inventory': pyo.value(m.y[t, j]),
            'Sales': pyo.value(m.z[t, j])
        })

# Convert list to DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

| Month | Color | Regular Production | Excess Production | Inventory | Sales |
|---|---|---|---|---|---|
| 1 | 1 | 250.0 | 0.0 | 0.0 | 250.0 |
| 1 | 2 | 200.0 | 0.0 | 0.0 | 200.0 |
| 2 | 1 | 90.0 | 0.0 | 0.0 | 90.0 |
| 2 | 2 | 160.0 | 0.0 | 0.0 | 160.0 |
| 3 | 1 | 250.0 | 0.0 | 0.0 | 250.0 |
| 3 | 2 | 250.0 | 200.0 | 0.0 | 450.0 |
| 4 | 1 | 350.0 | 0.0 | 0.0 | 350.0 |
| 4 | 2 | 150.0 | 150.0 | 0.0 | 300.0 |
| 5 | 1 | 300.0 | 20.0 | 0.0 | 320.0 |
| 5 | 2 | 200.0 | 0.0 | 0.0 | 200.0 |
| 6 | 1 | 300.0 | 80.0 | 0.0 | 380.0 |
| 6 | 2 | 200.0 | 0.0 | 0.0 | 200.0 |

# Problem 2:

**Production Data**

- **Labor availability:**
  - Plant 1: 240 hours
  - Plant 2: 150 hours
- **Labor requirements per unit:**
  - Table: 10 hours
  - Chair: 1 hour
- **Production costs per unit:**
  - **Plant 1:** 1200(*table*),100 (chair)
  - **Plant 2:** 1275(*table*),95 (chair)

**Demand Data**

- **Store 1:** 8 tables, 85 chairs
- **Store 2:** 12 tables, 60 chairs

**Shipping Costs and Constraints**

- Each product can be shipped directly from a plant to a store or routed through a distribution center (DC1 or DC2).

## Decision Variables

*$x_{ij}$ = Number of tables shipped along arc $(i, j) \in A$*

*$y_{ij}$ = Number of chairs shipped along arc $(i, j) \in A$*

*$t_i$ = Number of tables produced at plant $i \in 1, 2$*

*$c_i$ = Number of chairs produced at plant $i \in 1, 2$*

## Parameters

*$ct_{ij}$ = Shipping cost per table along arc $(i, j) \in A$*

*$cc_{ij}$ = Shipping cost per chair along arc $(i, j) \in A$*

*$pt_i$ = Production cost per table at plant $i = 1, 2$*

*$p_{ci}$ = Production cost per chair at plant $i = 1, 2$*

*$L_i$ = Hours of available labor at plant $i =1,2$*

*$d_{tj}$ = Demand for tables at store $j = 5, 6$*

*$d_{cj}$ = Demand for chairs at store $j = 5, 6$*

*$M_k$ = Number of chairs equivalent to one table at DC $k = 3, 4$*

*$C_k$ =Maximum table equivalent to capacity at DC $k = 3, 4$*

```
NODES = [1, 2, 3, 4, 5, 6]  # 1,2 are plants; 3,4 are DCs; 5,6 are stores
ARCS = [(1, 5), (1, 3), (1, 4), (2, 3), (2, 4), (2, 6), (3, 5), (3, 6), (4, 5), (4, 6)]

# Parameters
# Tables and chairs demand at nodes
t = {1: 0, 2: 0, 3: 0, 4: 0, 5: 8, 6: 12}  # Demand for tables
c = {1: 0, 2: 0, 3: 0, 4: 0, 5: 85, 6: 60}  # Demand for chairs

# Shipping costs
ct = {
    (1,5): 150, (1,3): 30, (1,4): 70,
    (2,3): 35, (2,4): 70, (2,6): 50,
    (3,5): 55, (3,6): 10, (4,5): 25, (4,6): 15
}
cc = {
    (1,5): 18, (1,3): 5, (1,4): 8,
    (2,3): 3, (2,4): 2, (2,6): 6,
    (3,5): 9, (3,6): 10, (4,5): 5, (4,6): 2
}

# Production costs
pt = {1: 1200, 2: 1275}  # Production cost per table
pc = {1: 100, 2: 95}     # Production cost per chair

# Labor hours available at plants
L = {1: 240, 2: 150}

# Demand at stores
dt = {5: 8, 6: 12}
dc = {5: 85, 6: 60}

# Distribution center parameters
M = {3: 50/6, 4: 42/5}  # Chairs equivalent to one table
C = {3: 6, 4: 5}        # Max tables capacity
```

## Model

$$\boldsymbol{Min} \sum_{(i,J)\in A} ct_{ij}x_{ij} + \sum_{(i,j)\in A}(cc_{ij}y_{ij} + \sum_{i=1}^{2} pt_i\, t_i + \sum_{i=1}^{2} pc_i\, c_i \qquad \text{(Total Cost)}$$

$\boldsymbol{s.t}$:

$10t_i + c_i \leq L_i \quad \forall i = 1,2$ (Labor constraint at the plants)

$t_i = \sum_{j:(i,j)\epsilon A} x_{ij} \quad \forall i = 1,2$ (Production-shipping balance for tables)

$c_i = \sum_{j:(i,j)\epsilon A} y_{ij} \quad \forall i = 1,2$ (Production-shipping balance for chairs)

$\sum_{j:(i,k)\epsilon A} x_{ik} = \sum_{j:(k,j)\epsilon A} x_{kj} \quad \forall k = 3,4$ (Flow balance for tables at DCs)

$\sum_{j:(i,k)\epsilon A} y_{ik} = \sum_{j:(k,j)\epsilon A} y_{kj} \quad \forall k = 3,4$ (Flow balance for chairs at DCs)

$dt_j = \sum_{i:(i,j)\epsilon A} x_{ij} \quad \forall j = 5,6$ (Demand satisfaction for tables)

$dc_j = \sum_{i:(i,j)\epsilon A} y_{ij} \quad \forall j = 5,6$ (Demand satisfaction for chairs)

$\sum_{i:(i,k)\epsilon A} x_{ik} + \frac{\sum_{i:(i,k)\epsilon A} y_{ik}}{M_k} \leq C_k \ \forall$ (Distribution center capacity constraint)

$x_{ij}, y_{ij}, t_i, k_i \geq 0$ (Nonnegativity constraints)

## Model

```
# Define the model
m = pyo.ConcreteModel()
m.clear()

# Decision Variables
m.x = pyo.Var(ARCS, domain=pyo.NonNegativeReals)  # tables shipped
m.y = pyo.Var(ARCS, domain=pyo.NonNegativeReals)  # chairs shipped
m.t = pyo.Var([1,2], domain=pyo.NonNegativeReals)  # tables produced
m.c = pyo.Var([1,2], domain=pyo.NonNegativeReals)  # chairs produced

# Objective Function: Minimize total cost
def objective_rule(m):
    shipping_cost = (
        sum(ct[i,j] * m.x[i,j] for i,j in ARCS) +
        sum(cc[i,j] * m.y[i,j] for i,j in ARCS)
    )
    production_cost = (
        sum(pt[i] * m.t[i] for i in [1,2]) +
        sum(pc[i] * m.c[i] for i in [1,2])
    )
    return shipping_cost + production_cost

m.obj = pyo.Objective(rule=objective_rule, sense=pyo.minimize)

# Labor Constraints at Plants
def labor_constraint_rule(m, i):
```

```
    if i in [1,2]:  # only for plants
            return 10 * m.t[i] + m.c[i] <= L[i]
        return pyo.Constraint.Skip
    m.labor_constraint = pyo.Constraint([1,2], rule=labor_constraint_rule)

    # Production Balance at Plants for Tables
    def table_prod_balance_rule(m, i):
        if i in [1,2]:  # only for plants
            return m.t[i] == sum(m.x[i,j] for j in NODES if (i,j) in ARCS)
        return pyo.Constraint.Skip
    m.table_prod_balance = pyo.Constraint([1,2], rule=table_prod_balance_rule)

    # Production Balance at Plants for Chairs
    def chair_prod_balance_rule(m, i):
        if i in [1,2]:  # only for plants
            return m.c[i] == sum(m.y[i,j] for j in NODES if (i,j) in ARCS)
        return pyo.Constraint.Skip
    m.chair_prod_balance = pyo.Constraint([1,2], rule=chair_prod_balance_rule)

    # Flow Balance at DCs for Tables
    def dc_table_flow_rule(m, k):
        if k in [3,4]:  # only for DCs
            return sum(m.x[i,k] for i in NODES if (i,k) in ARCS) == \
                   sum(m.x[k,j] for j in NODES if (k,j) in ARCS)
        return pyo.Constraint.Skip
    m.dc_table_flow = pyo.Constraint([3,4], rule=dc_table_flow_rule)

    # Flow Balance at DCs for Chairs
    def chair_flow_rule(m, k):
        if k in [3,4]:  # only for DCs
            return sum(m.y[i,k] for i in NODES if (i,k) in ARCS) == \
                   sum(m.y[k,j] for j in NODES if (k,j) in ARCS)
        return pyo.Constraint.Skip
    m.dc_chair_flow = pyo.Constraint([3,4], rule=chair_flow_rule)

    # Demand Satisfaction for Tables
    def table_demand_rule(m, j):
        if j in [5,6]:  # only for stores
            return sum(m.x[i,j] for i in NODES if (i,j) in ARCS) == dt[j]
        return pyo.Constraint.Skip
    m.table_demand = pyo.Constraint([5,6], rule=table_demand_rule)

    # Demand Satisfaction for Chairs
    def chair_demand_rule(m, j):
        if j in [5,6]:  # only for stores
            return sum(m.y[i,j] for i in NODES if (i,j) in ARCS) == dc[j]
        return pyo.Constraint.Skip
    m.chair_demand = pyo.Constraint([5,6], rule=chair_demand_rule)

    # DC Capacity Constraints
    def dc_capacity_rule(m, k):
        if k in [3,4]:  # only for DCs
            incoming_tables = sum(m.x[i,k] for i in NODES if (i,k) in ARCS)
            incoming_chairs_equivalent = sum(m.y[i,k] for i in NODES if (i,k) in ARCS) / M[k]
            return incoming_tables + incoming_chairs_equivalent <= C[k]
        return pyo.Constraint.Skip
    m.dc_capacity = pyo.Constraint([3,4], rule=dc_capacity_rule)
```

### Solving the Model

```
#Declare the solver as CBC
opt = pyo.SolverFactory('cbc')

#Solve the model
opt.solve(m).write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 41669.56522
  Upper bound: 41669.56522
  Number of objectives: 1
  Number of constraints: 16
  Number of variables: 24
  Number of nonzeros: 22
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  User time: -1.0
  System time: 0.0
  Wallclock time: 0.0
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an opt
  Statistics:
    Branch and bound:
    Number of bounded subproblems: None Number of created subproblems: None
    Black box:
      Number of iterations: 17
  Error rc: 0
  Time: 0.025279998779296875
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

```
print('\nOptimal Distribution Plan to Minimize Cost:') #
Header for tables
  print('\nTABLES SHIPMENTS:')
  print(f"{'From':^6}{'To':^6}{'Cost ($)':^10}{'Units':^10}")

  # Only show non-zero table shipments
  for i,j in ARCS:
      if pyo.value(m.x[i,j]) > 0.01:  # Only show if quantity > 0.01
          print(f"{i:^6}{j:^6}{ct[i,j]:^10.2f}{pyo.value(m.x[i,j]):^10.2f}")

  # Header for chairs
  print('\nCHAIRS SHIPMENTS:')
  print(f"{'From':^6}{'To':^6}{'Cost ($)':^10}{'Units':^10}")

  # Only show non-zero chair shipments
  for i,j in ARCS:
      if pyo.value(m.y[i,j]) > 0.01:  # Only show if quantity > 0.01
          print(f"{i:^6}{j:^6}{cc[i,j]:^10.2f}{pyo.value(m.y[i,j]):^10.2f}")

  # Summary of production
  print('\nPRODUCTION SUMMARY:')
  print(f"{'Plant':^8}{'Tables':^12}{'Chairs':^12}")
  for i in [1,2]:
      print(f"{i:^8}{pyo.value(m.t[i]):^12.2f}{pyo.value(m.c[i]):^12.2f}")
  # Cost summary
  print('\nCOST SUMMARY:')
  print(f"Total Cost: ${pyo.value(m.obj):.2f}")
```

```
Optimal Distribution Plan to Minimize Cost:

TABLES SHIPMENTS:
 From   To   Cost ($)   Units
  1     5     150.00     8.00
  1     3     30.00      6.00
  1     4     70.00      0.65
  2     6     50.00      5.35
  3     6     10.00      6.00
  4     6     15.00      0.65

CHAIRS SHIPMENTS:
 From   To   Cost ($)   Units
  1     5     18.00     48.48
  2     4      2.00     36.52
  2     6      6.00     60.00
  4     5      5.00     36.52

PRODUCTION SUMMARY:
 Plant     Tables      Chairs
   1       14.65       48.48
   2        5.35       96.52

COST SUMMARY:
Total Cost: $41669.57
```