› Set-Up

[ ] ↳ *1 cell hidden*

Problem A: Practice writing an optimizing problem

A rectangular box must have volume 500 in^3 (Recall: a box's volume is defined by its $length * width * height$).

Find the shape that has the smallest "mailing length" where the mailing length is defined as the sum of the three edge lengths.

What is the optimal shape of the box that meets this volume requirement with the smallest mailing length possible? (Use ipopt since this is a nonlinear problem).

```
#initialize a "Concrete Model"
modelA = ConcreteModel()

#initialize DVs

modelA.length = Var(domain = NonNegativeReals)
modelA.width = Var(domain = NonNegativeReals)
modelA.height = Var(domain = NonNegativeReals)

# Define the objective
def obj_function(model):
    return sum([modelA.length, modelA.width, modelA.height])

modelA.Objective = Objective(expr=obj_function, sense = minimize)

#specify the constraints
modelA.Constraints = ConstraintList()
modelA.Constraints.add(expr = modelA.length * modelA.width * modelA.height >= 500)

#(Optional) You can use model.pprint() to see what you've done so far
modelA.pprint()
```

```
3 Var Declarations
    height : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals
    length : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals
    width : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  None :  None : False :  True : NonNegativeReals

1 Objective Declarations
    Objective : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
```

```
            None :   True : minimize : length + width + height

    1 Constraint Declarations
        Constraints : Size=1, Index={1}, Active=True
            Key : Lower : Body                 : Upper : Active
              1 : 500.0 : length*width*height :  +Inf :   True

    5 Declarations: length width height Objective Constraints
```

```
#solve model (Note: you should use "ipopt" because this is a nonlinear problem)
opt = SolverFactory('ipopt')
results = opt.solve(modelA, tee = False) #setting tee = False hides the diagnostic outputs


#print optimal solution and the smallest mailing length it achieves
print("length* = ", modelA.length())
print("width* = ", modelA.width())
print("height* = ", modelA.height())
print("Mailing Length* = ", modelA.Objective())
```

```
length* =  7.937005234219425
width* =  7.937005234219425
height* =  7.937005234219425
Mailing Length* =  23.811015702658274
```

Problem B: Optimizing an Existing Function (office building)

Below, I've included a completed "office building" model as a function. Use Pyomo to solve for the price per square foot in each year that maximizes the total earnings after tax. Use ipopt (since this is a nonlinear problem).

```
def office_earnings(total_sqft = 180000,
           m = -0.05,
           b = 1.5,
           op_expense_per_sqft = 1.20,
           heating_surcharge_per_sqft = .2,
           op_exp_annual_growth = .12,
           annual_mortgage = 1500000,
           tax_rate = .34,
           price_per_sqft = [15, 15, 15, 15, 15],
           num_years = 5):
  #rev calc
  perc_occ = [m*price_per_sqft[i] + b for i in range(num_years)]
  sqft_occ = [perc_occ[i]*total_sqft for i in range(num_years)]
  revenue = [sqft_occ[i]*price_per_sqft[i] for i in range(num_years)]
  #operating expense calculations
  base_op_cost_as_percY1 = [(1+op_exp_annual_growth)**i for i in range(num_years)] #note that range(nu
  base_op_cost = [op_expense_per_sqft*total_sqft*base_op_cost_as_percY1[i] for i in range(num_years)]
  heating_surcharge = [perc_occ[i]*base_op_cost[i]*heating_surcharge_per_sqft for i in range(num_years
  mortgage = [annual_mortgage for i in range(num_years)]
  operating_costs = [base_op_cost[i] + heating_surcharge[i] + mortgage[i] for i in range(num_years)]
  #before and after-tax earnings
  ebt = [revenue[i] - operating_costs[i] for i in range(num_years)]
```

```
  taxes = [ebt[i]*tax_rate for i in range(num_years)]
  earnings_after_tax = [ebt[i] - taxes[i] for i in range(num_years)]
  total_earnings_after_tax = sum(earnings_after_tax)
  return total_earnings_after_tax


#initialize a "Concrete Model"
modelB = ConcreteModel()

#initialize DVs

modelB.office_dv = Var(range(5), domain = NonNegativeReals)

#define the objective

modelB.Objective = Objective(expr = office_earnings(price_per_sqft = modelB.office_dv), sense = maximi

#(Optional) You can use model.pprint() to see what you've done so far
modelB.pprint()
```

```
1 Var Declarations
    office_dv : Size=5, Index={0, 1, 2, 3, 4}
        Key : Lower : Value : Upper : Fixed : Stale : Domain
          0 :     0 :  None :  None : False :  True : NonNegativeReals
          1 :     0 :  None :  None : False :  True : NonNegativeReals
          2 :     0 :  None :  None : False :  True : NonNegativeReals
          3 :     0 :  None :  None : False :  True : NonNegativeReals
          4 :     0 :  None :  None : False :  True : NonNegativeReals

1 Objective Declarations
    Objective : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : (-0.05*office_dv[0] + 1.5)*180000*office_dv[0] - (216000.0 + (·

2 Declarations: office_dv Objective
```

```
#solve model (Note: you should use "ipopt" because this is a nonlinear problem)
opt = SolverFactory('ipopt')
results = opt.solve(modelB, tee = False) #setting tee = False hides the diagnostic outputs



#print optimal solution and the largest earnings it achieves

print("optimal sqft in Y1 = ", modelB.office_dv[0]())
print("optimal sqft in Y2 = ", modelB.office_dv[1]())
print("optimal sqft in Y3 = ", modelB.office_dv[2]())
print("optimal sqft in Y4 = ", modelB.office_dv[3]())
print("optimal sqft in Y5 = ", modelB.office_dv[4]())
print("Optimal earning * = ", modelB.Objective())
```

```
optimal sqft in Y1 =  15.120000000025168
optimal sqft in Y2 =  15.134400000025144
optimal sqft in Y3 =  15.150528000025117
optimal sqft in Y4 =  15.168591360025088
optimal sqft in Y5 =  15.188822323225056
Optimal earning * =  691696.8342059832
```

Problem C: Coding with Lists of Lists and Constraint Lists

Solve this small version of the Stigler problem shown below using lists, `ConstraintLists`, and `for` loops. This version only has **4 decision variables (DVs)** and **3 constraints** but please code in a way that would be scalable for larger problems by following the structure I've started for you below. Print out the optimal (x)'s and the total optimal cost.

Minimize cost =

$$0.36 * x_{\text{wheat}} + 0.141 * x_{\text{mac}} + 0.242 * x_{\text{cereal}} + 0.300 * x_{\text{milk}}$$

Subject to:

$$16.1 * x_{\text{wheat}} + 1.6 * x_{\text{mac}} + 2.9 * x_{\text{cereal}} + 12.5 * x_{\text{milk}} \geq 3 \quad \text{(Calories Daily Min Constraint)}$$
$$7.9 * x_{\text{wheat}} + 58.9 * x_{\text{mac}} + 91.2 * x_{\text{cereal}} + 42.3 * x_{\text{milk}} \geq 1.8 \quad \text{(Protein Daily Min Constraint)}$$
$$80.5 * x_{\text{wheat}} + 3.0 * x_{\text{mac}} + 7.2 * x_{\text{cereal}} + 15.4 * x_{\text{milk}} \geq 2.5 \quad \text{(Fiber Daily Min Constraint)}$$

$$x_{\text{wheat}}, x_{\text{mac}}, x_{\text{cereal}}, x_{\text{milk}} \geq 0$$

```
#I've put in these input parameters for you
num_commodities = 4    #this is how many food commodities to decide on
num_nutrients = 3      #this is how many nutrient constraints there are
cost_coef = [.36, 0.141, 0.242, 0.300]   #this is a list of the cost coefficients in the objective
constraint_coef = [[16.1, 1.6, 2.9, 12.5],
                   [7.9, 58.9, 91.2, 42.3],
                   [80.5, 3.0, 7.2, 15.4]]     #this is the list of lists of all the constraint coeffi
daily_mins = [3, 1.8, 2.5]        #these are the right hand sides


#Fill in the ??? to complete this code
modelC = ConcreteModel()

#dvs
modelC.x = Var(range(num_commodities), domain = NonNegativeReals)

#objective
modelC.Objective = Objective(expr = sum(cost_coef[i] * modelC.x[i] for i in range(num_commodities)), s

#constraints
modelC.nutrient_constraints = ConstraintList()
for i in range(num_nutrients):
  modelC.nutrient_constraints.add(expr = sum(constraint_coef[i][j] * modelC.x[j] for j in range(num_cc

#model pprint()
modelC.pprint()
```

```
1 Var Declarations
    x : Size=4, Index={0, 1, 2, 3}
        Key : Lower : Value : Upper : Fixed : Stale : Domain
          0 :     0 :  None :  None : False :  True : NonNegativeReals
          1 :     0 :  None :  None : False :  True : NonNegativeReals
          2 :     0 :  None :  None : False :  True : NonNegativeReals
          3 :     0 :  None :  None : False :  True : NonNegativeReals
```

```
1 Objective Declarations
    Objective : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : minimize : 0.36*x[0] + 0.141*x[1] + 0.242*x[2] + 0.3*x[3]

1 Constraint Declarations
    nutrient_constraints : Size=3, Index={1, 2, 3}, Active=True
        Key : Lower : Body                                          : Upper : Active
          1 :   3.0 :  16.1*x[0] + 1.6*x[1] + 2.9*x[2] + 12.5*x[3] :  +Inf :   True
          2 :   1.8 : 7.9*x[0] + 58.9*x[1] + 91.2*x[2] + 42.3*x[3] :  +Inf :   True
          3 :   2.5 :  80.5*x[0] + 3.0*x[1] + 7.2*x[2] + 15.4*x[3] :  +Inf :   True

3 Declarations: x Objective nutrient_constraints
```

```
#solve model with cbc because this is a linear program
opt = SolverFactory('cbc')
results = opt.solve(modelC, tee = True)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.10
Build Date: Jun  7 2023

command line - /content/bin/cbc -printingOptions all -import /tmp/tmp1qqa995k.pyomo.lp -stat=1 -sc
Option for printingOptions changed from normal to all
Presolve 3 (0) rows, 4 (0) columns and 12 (0) elements
Statistics for presolved model


Problem has 3 rows, 4 columns (4 with objective) and 12 elements
Column breakdown:
4 of type 0.0->inf, 0 of type 0.0->up, 0 of type lo->inf,
0 of type lo->up, 0 of type free, 0 of type fixed,
0 of type -inf->0.0, 0 of type -inf->up, 0 of type 0.0->1.0
Row breakdown:
0 of type E 0.0, 0 of type E 1.0, 0 of type E -1.0,
0 of type E other, 0 of type G 0.0, 0 of type G 1.0,
3 of type G other, 0 of type L 0.0, 0 of type L 1.0,
0 of type L other, 0 of type Range 0.0->1.0, 0 of type Range other,
0 of type Free
Presolve 3 (0) rows, 4 (0) columns and 12 (0) elements
0  Obj 0 Primal inf 0.23712785 (3)
2  Obj 0.067266607
Optimal - objective value 0.067266607
Optimal objective 0.06726660713 - 2 iterations time 0.002
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00
```

```
#print optimal amounts of each food and the optimal cost it achieves

for i in range(num_commodities):
  print(f'x{i} = {modelC.x[i]()}')

print("obj* = ", modelC.Objective())
```

```
x0 = 0.17929518
x1 = 0.0
x2 = 0.0
x3 = 0.0090678024
```

```
obj* =  0.06726660552000001
```

Start coding or generate with AI.