L2_HW_Solution

August 31, 2020

SA405 · Advanced Math Programming · Fall 2020 · Goran, Lourenco, Skipper

Lesson 2: Pyomo Review

Homework

(1) Implement the abstract model provided below using the same format as in Lesson 2. Implement each line of code in its own cell, running the cells as you go. Practice using `model.display()`, `print(model.obj.expr)`, and `print(model.constraint_name['nutrient_name'].expr)` to view your model as you go.

Sets - I = set of feed ingredients ({oats, corn, alfalfa}) - N = set of RDA nutrients ({protein, fat, fiber})

[ ]:
```
# Set I
# (Choose short, descriptive names for your sets and
#  parameters using Python naming conventions.
#  They should not be the single letter names
#  used in the model.)
```

[ ]:
```
# Set I will be called INGR. Definition is below
INGR = ['oats','corn','alfalfa']
```

[ ]:
```
# Set N  will be called NUTR
NUTR = ['protein','fat','fiber']
```

Parameters - c_i = the cost per ton of ingredient i, for all $i \in I$ - $a_{i,n}$ = the RDA amount of nutrient n possessed by ingredient i, for all $i \in I, n \in N$ - l_n = the lower bound on the percent RDA of nutrient n that must be in the feed, for $n \in N$ - u_n = upper bound on the percent RDA of nutrient n that can be in the feed, for $n \in N$

[ ]:
```
# c_i will be called COST
COST = {'oats':80, 'corn':110  ,'alfalfa':90 }
```

[ ]:
```
# a_{i,n} will be caled AMMOUNT
AMOUNT = {('oats','protein'):0.13, ('oats','fat'):0.22, ('oats','fiber'):0.4,
          ('corn','protein'):0.07, ('corn','fat'): 0.10, ('corn','fiber'):0.30,
          ('alfalfa','protein'):0.04, ('alfalfa','fat'):0.15, ('alfalfa',␣
 ↪'fiber'):0.60  }
```

```
[ ]: # l_n  (We must define a lower bound for EVERY n in N) will be called LOWER_BOUND
LOWER_BOUND = {'protein':0.08 , 'fat':0.12 , 'fiber':0 }
```

```
[ ]: # u_n  (We must define an upper bound for EVERY n in N) will be caled UPPER_BOUND
UPPER_BOUND = {'protein':1 , 'fat':0.16 , 'fiber':0.5 }
```

Build model

```
[ ]: # import Pyomo
import pyomo.environ as pyo
```

```
[ ]: # instantiate ConcreteModel
model = pyo.ConcreteModel()
```

Decision variables - x_i = the fraction(portion) of a ton of ingredient i to put into one ton of feed, for all $i \in I$ - The decision variables are nonngeative reals which is indicated by the following constraint: $x_i \geq 0, \forall i \in I$ and is incorporated into the definition of the variables in the 'domain' part.

```
[ ]: # add x_i
model.x = pyo.Var(INGR, domain=pyo.NonNegativeReals)
```

```
[ ]: # checking the model at this stage
model.display()
```

Objective - Minimize $\sum_{i \in I} c_i x_i$

```
[ ]: # objective function helper function
def obj_rule(model):
    return sum(COST[ingr]*model.x[ingr] for ingr in INGR)
```

```
[ ]: # add objective function
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.minimize)
```

```
[ ]: # View the objective function
print(model.obj.expr)
```

Constraints Subject to - $\sum_{i \in I} x_i = 1$ (one ton) - $\sum_{i \in I} a_{i,n} x_i \geq l_n, \quad \forall n \in N$ (lower bound) - $\sum_{i \in I} a_{i,n} x_i \leq u_n, \quad \forall n \in N$ (upper bound)

```
[ ]: # one ton helper function
def one_ton_rule(model):
    return sum(model.x[ingr] for ingr in INGR) == 1
# Pay attention! The equality inthe constraint is written as '==' NOT '='. The
↪'=' is reserved for writing the commands.
```

```
[ ]: # add one ton constraint
model.one_ton_constraint = pyo.Constraint(rule=one_ton_rule)
```

[ ]:
```python
# lower bound helper function
def lower_bound_rule(model, nutr):
    return sum(AMOUNT[ingr,nutr]*model.x[ingr] for ingr in INGR) >=␣
 ↪LOWER_BOUND[nutr]
```

[ ]:
```python
# add lower bound constraint
model.lower_bound_constraint = pyo.Constraint(NUTR, rule=lower_bound_rule)
```

[ ]:
```python
# upper bound helper function
def upper_bound_rule(model, nutr):
    return sum(AMOUNT[ingr,nutr]*model.x[ingr] for ingr in INGR) <=␣
 ↪UPPER_BOUND[nutr]
```

[ ]:
```python
# add upper bound constraint
model.upper_bound_constraint = pyo.Constraint(NUTR, rule=upper_bound_rule)
```

Solve

[ ]:
```python
# solve model
solver_result = pyo.SolverFactory('glpk').solve(model)#Check how the solver␣
 ↪terminated
print(solver_result.solver.termination_condition)
```

[ ]:
```python
#Check how the solver terminated
print(solver_result.solver.termination_condition)
```

[ ]:
```python
model.display()
```

Print solution

[ ]:
```python
# Make a for loop over the ingredients and print how
# much of each ingredient is used.  Use f-strings
# to print.
print(f"The amount of the ingredients in one ton MIX are listed below" )
for ingr in INGR:
    print(f"x_{ingr} = {model.x[ingr].value}")
```

[ ]:
```python
# Print the value of the objective function
print(f' The cost of the MIX is {model.obj()}')
```

(2) Copy the code into a 5-cell format as done at the bottom of Lesson 2: (1) import Pyomo (2) construct data (3) build model (4) solve (5) print solution. Make sure everything still runs!

[8]:
```python
#Import pyomo
import pyomo.environ as pyo
```

[9]:
```python
#Construct data
```

```
#Sets
INGR = ['oats','corn','alfalfa']

NUTR = ['protein','fat','fiber']


#Parameters
COST = {'oats':80, 'corn':110  ,'alfalfa':90 }

AMOUNT = {('oats','protein'):0.13, ('oats','fat'):0.22, ('oats','fiber'):0.4,
          ('corn','protein'):0.07, ('corn','fat'): 0.10, ('corn','fiber'):0.30,
          ('alfalfa','protein'):0.04, ('alfalfa','fat'):0.15, ('alfalfa',␣
 ↪'fiber'):0.60  }

LOWER_BOUND = {'protein':0.08 , 'fat':0.12 , 'fiber':0 }

UPPER_BOUND = {'protein':1 , 'fat':0.16 , 'fiber':0.5 }
```

[10]:
```
#Build model
model = pyo.ConcreteModel()

#Add decision variables
model.x = pyo.Var(INGR, domain=pyo.NonNegativeReals)

#Add objective function
def obj_rule(model):
    return sum(COST[ingr]*model.x[ingr] for ingr in INGR)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.minimize)

#Add constraints
# one ton constraint
def one_ton_rule(model):
    return sum(model.x[ingr] for ingr in INGR) == 1
model.one_ton_constraint = pyo.Constraint(rule=one_ton_rule)
# lower bound constraint
def lower_bound_rule(model, nutr):
    return sum(AMOUNT[ingr,nutr]*model.x[ingr] for ingr in INGR) >=␣
 ↪LOWER_BOUND[nutr]
model.lower_bound_constraint = pyo.Constraint(NUTR, rule=lower_bound_rule)
#  upper bound constraint
def upper_bound_rule(model, nutr):
    return sum(AMOUNT[ingr,nutr]*model.x[ingr] for ingr in INGR) <=␣
 ↪UPPER_BOUND[nutr]
model.upper_bound_constraint = pyo.Constraint(NUTR, rule=upper_bound_rule)
```

[11]:
```python
#Solve model
solver_result = pyo.SolverFactory('glpk').solve(model)
#Check if the solver found an optimal solution
print(solver_result.solver.termination_condition)

#or...

#Solve model, displaying solver output
#solver_result = pyo.SolverFactory('glpk').solve(model, tee=True)
```

```
optimal
```

[12]:
```python
#Print solution (if optimal solution was found)
print(f"The amount of the ingredients in one ton MIX are listed below" )
for ingr in INGR:
    print(f"x_{ingr} = {model.x[ingr].value}")
    print(f'\n')
print(f'Cost of the MIX = {model.obj()}')
```

```
The amount of the ingredients in one ton MIX are listed below
x_oats = 0.348484848484849


x_corn = 0.287878787878788


x_alfalfa = 0.363636363636364


Cost of the MIX = 92.27272727272737
```

[ ]:
```python

```