Optimization problems in Python using Pyomo: An Introduction

Camilo Velasquez A

Overview

- Introduction
- Pyomo Modeling: Warehouse Problem
- More Examples!

Introduction

- Definition of Optimization Problems
- Why Pyomo?
- Sample Applications

Definition of Optimization Problem

Components:

- **Objective/Goal:** What wants to be maximized, minimized, eg.
  Eg: Minimize costs, maximize income, Is it possible/feasible?
- **Constraints/Relations:** Set of constraints or requirements that must be satisfied
  Eg: Total machines are 10, Strictly positive price, etc
- **Decision Variables:** Set of variables or parameters that can be tuned to fulfil requirements while reaching the objective
  Eg: Price of rice, Number of machines, Indicator of use machine.

This is independent of how we solve it!

PYOMO

Sandia National Laboratories

COIN|OR

- ❑ It is Pythonic and Object Oriented
- ❑ Open Source!
- ❑ Customize Capability: Easy to modularize components
- ❑ Solver Agnostic: Can use multiple open source or commercial solvers (AMPL, IPOPT, GLPK, ...)
- ❑ High level API.
- ❑ Extended documentation
- ❑ Supports analysis of complex optimization problems
- ❑ Can tackle advanced optimization problems (Mixed Integer, Discrete, Nonlinear, Stochastic, Disjunctive, etc)

Sample Applications

- Job Scheduling
- Logistic/Transportation
- Industrial Production
- Portfolio Optimization
- Resource Allocation
- Parameter Estimation
- Blending Problems
- Network designing
- Prices Design
- Much more applications!!

PyOmo Modeling: The Warehouse Problem

- Definition and Formulation
- Pyomo approach

Definition

- We have to attend 5 countries from 4 possible warehouses locations.
- We have to define which warehouse locations are we going to use, and how much demand percentage each warehouse is going to attend from each country.
- We need to minimize the costs of having the warehouses, meeting all the demand within the countries.
- We can only build P warehouses

Adapted from: Pyomo. Optimization Modeling in Python.

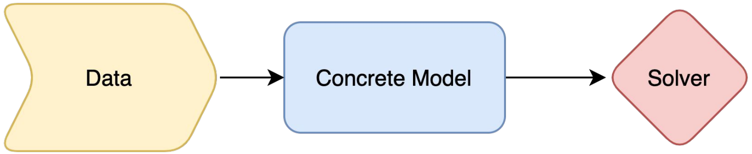
Formulation

Objective/Goal	Minimize the cost of fulfilling the demand
Constraints	Ensure that all demand was satisfied for each country Ensure that you can only use the built warehouses Ensure that you can only use P warehouses The demand attended from each warehouse is a fraction A warehouse is used or not (Binary)
Decision Variables	Which warehouses are going to be used? How much supply should a warehouse deliver to a customer?

PyOmo Modeling Approaches

Concrete Model

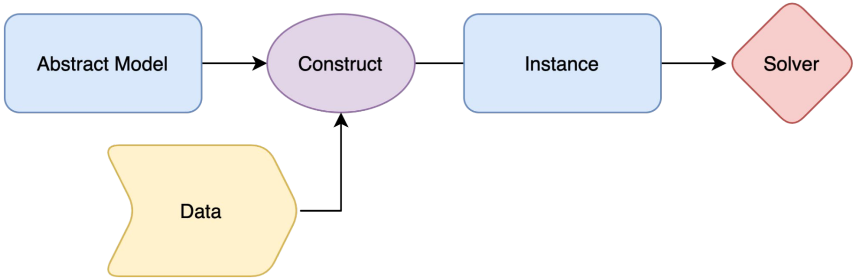
$$
\begin{array}{ll}
\min & 2x_1 + 3x_2 \\
\text{s.t.} & 3x_1 + 4x_2 \geq 1 \\
 & x_1, x_2 \geq 0
\end{array}
$$



- Similar to Eager Mode
- Nice for debug. Can watch and track the model components (Constraints, objective function, etc)
- Fix size of parameters

Abstract Model

$$
\begin{array}{lll}
\min & \sum_{j=1}^{n} c_j x_j & \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i = 1 \ldots m \\
 & x_j \geq 0 & \forall j = 1 \ldots n
\end{array}
$$



- Similar to lazy execution
- Define a graph/flow of the data and the interaction in the model
- Dynamic size of parameters

Concrete Model

Candidate Warehouse Location	CiudadPanama	Bogota	Lima	RioJaneiro
Customers Country				
Panama	10	130	90	420
Colombia	100	50	110	340
Peru	200	150	20	330
Guatemala	70	180	160	450
Brazil	300	380	320	40

Costs (d_mn)

N : Set of candidate warehouse locations

M : Set of customer locations

$d_{m,n}$: Cost of delivering product to customer **m** from warehouse **n**

y_n : 1 if warehouse **n** is selected. 0 otherwise.

$x_{n,m}$: Fraction of the demand for customer **m** served from warehouse **n**

P : Limit of warehouses that can be built

```
N = costs_df.columns # Warehouse locations (Cities)
M = costs_df.index # Customers (Countries)
# d: Costs as Series. Dict[Tuple[City, Country],DemandFraction]
d = costs_df.unstack(0).to_dict()
P = 2 # Limit of Warehouses

model = ConcreteModel(name="Warehouse Example")
model.x = Var(N, M, bounds=(0,1)) # Constraint (5)
model.y = Var(N, within=Binary,) # Constraint (6)
```

Concrete Model

Minimize Costs | $$\min_{x,y} \sum_{n\in N} \sum_{n\in N} d_{n,m} x_{n,m} \quad (1)$$

$$s.t.$$

All demand satisfied | $$\sum_{n\in N} x_{n,m} = 1, \forall m \in M \quad (2)$$

Only use built WH | $$x_{n,m} \leq y_n, \forall n \in N, m \in M \quad (3)$$

P Warehouses Limit | $$\sum_{n\in N} y_n \leq P \quad (4)$$

Demand as Fraction | $$0 \leq x_{n,m} \leq 1 \quad (5)$$

WH used or not | $$y_n \in \{0, 1\} \quad (6)$$

```
# Objective (1)
model.obj = Objective(
    expr=np.sum([d[n,m]*model.x[n,m] for n in N for m in M]),
    sense=minimize, name="Minimize Cost")

# Constraint (2)
def one_per_customer_rule(model, m):
    return np.sum([model.x[n,m] for n in N]) == 1
model.customers_complete_frac = Constraint(
    M, rule=one_per_customer_rule,
    name="Constraint 2 - Customer complete fraction")

# Constraint (3)
def warehouse_active_rule(model, n, m):
    return model.x[n,m] <= model.y[n]
model.warehouse_active = Constraint(
    N, M, rule=warehouse_active_rule,
    name="Constraint 3 - Only existing warehouses")

# Constraint (4)
model.warehouses_limit = Constraint(
    expr=np.sum([model.y[n] for n in N]) <= P,
    name="Constraint 4 - Warehouse Limit")

# Solve the model and report the results
solver = SolverFactory('glpk')
solver.solve(model)
```

Solve the model using **GLPK** |

Solution Display

Decision variables Outcome

```
Variables:
  x : Size=20, Index=x_index
      Key                          : Lower : Value : Upper : Fixed : Stale : Domain
             ('Bogota', 'Brazil') :     0 :   0.0 :     1 : False : False :  Reals
           ('Bogota', 'Colombia') :     0 :   0.0 :     1 : False : False :  Reals
          ('Bogota', 'Guatemala') :     0 :   0.0 :     1 : False : False :  Reals
             ('Bogota', 'Panama') :     0 :   0.0 :     1 : False : False :  Reals
               ('Bogota', 'Peru') :     0 :   0.0 :     1 : False : False :  Reals
       ('CiudadPanama', 'Brazil') :     0 :   0.0 :     1 : False : False :  Reals
     ('CiudadPanama', 'Colombia') :     0 :   1.0 :     1 : False : False :  Reals
    ('CiudadPanama', 'Guatemala') :     0 :   1.0 :     1 : False : False :  Reals
       ('CiudadPanama', 'Panama') :     0 :   1.0 :     1 : False : False :  Reals
         ('CiudadPanama', 'Peru') :     0 :   1.0 :     1 : False : False :  Reals
               ('Lima', 'Brazil') :     0 :   0.0 :     1 : False : False :  Reals
             ('Lima', 'Colombia') :     0 :   0.0 :     1 : False : False :  Reals
            ('Lima', 'Guatemala') :     0 :   0.0 :     1 : False : False :  Reals
               ('Lima', 'Panama') :     0 :   0.0 :     1 : False : False :  Reals
                 ('Lima', 'Peru') :     0 :   0.0 :     1 : False : False :  Reals
         ('RioJaneiro', 'Brazil') :     0 :   1.0 :     1 : False : False :  Reals
       ('RioJaneiro', 'Colombia') :     0 :   0.0 :     1 : False : False :  Reals
      ('RioJaneiro', 'Guatemala') :     0 :   0.0 :     1 : False : False :  Reals
         ('RioJaneiro', 'Panama') :     0 :   0.0 :     1 : False : False :  Reals
           ('RioJaneiro', 'Peru') :     0 :   0.0 :     1 : False : False :  Reals
  y : Size=4, Index=y_index
      Key          : Lower : Value : Upper : Fixed : Stale : Domain
            Bogota :     0 :   0.0 :     1 : False : False : Binary
      CiudadPanama :     0 :   1.0 :     1 : False : False : Binary
              Lima :     0 :   0.0 :     1 : False : False : Binary
        RioJaneiro :     0 :   1.0 :     1 : False : False : Binary
```

Candidate Warehouse Location	CiudadPanama	Bogota	Lima	RioJaneiro
Customers Country				
Panama	10	130	90	420
Colombia	100	50	110	340
Peru	200	150	20	330
Guatemala	70	180	160	450
Brazil	300	380	320	40

Total Cost: 420

```
Objectives:
  obj : Size=1, Index=None, Active=True
      Key  : Active : Value
      None :   True : 420.0
```

Abstract Model

```python
N = costs_df.columns # Warehouse locations (Cities)
M = costs_df.index # Customers (Countries)
# d: Costs as Series. Dict[Tuple[City, Country],DemandFraction]
d = costs_df.unstack(0).to_dict()
P = 2 # Limit of Warehouses

model = ConcreteModel(name="Warehouse Example")
model.x = Var(N, M, bounds=(0,1)) # Constraint (5)
model.y = Var(N, within=Binary,) # Constraint (6)
```

```python
model = AbstractModel(name="Warehouse Example Abstract")
# Set: Pyomo Model component to express List or Indexables
model.dual = Suffix(direction=Suffix.IMPORT)
model.N = Set()
model.M = Set()
model.d = Param(model.N,model.M)
model.P = Param()

model.x = Var(model.N, model.M, bounds=(0,1))
model.y = Var(model.N, within=Binary)
```

Abstract Model: Instantiate with Python

```python
data={
  "namespace1": {
    "N": {None: N},
    "M": {None: M},
    "d": d,
    "P": {None: P}
  },
  "namespace2": {
    "N": {None: N},
    "M": {None: M},
    "d": d,
    "P": {None: 3}
  }
}
```

```python
instance = model.create_instance(namespace="namespace1",
                                 data=data)
# With the other namespace
instance2 = model.create_instance(namespace="namespace2",
                                  data=data)
solver = SolverFactory('cbc')
solver.solve(instance)
```

Example:
Sudoku Solver

- Definition and Formulation
- Pyomo approach

Definition

						2		
	8				7		9	
6		2				5		
	7			6				
			9		1			
				2			4	
		5				6		3
	9		4				7	
		6						

(a) Sudoku Puzzle

9	5	7	6	1	3	2	8	4
4	8	3	2	5	7	1	9	6
6	1	2	8	4	9	5	3	7
1	7	8	3	6	4	9	5	2
5	2	4	9	7	1	3	6	8
3	6	9	5	2	8	7	4	1
8	4	5	7	9	2	6	1	3
2	9	1	4	3	6	8	7	5
7	3	6	1	8	5	4	2	9

(b) Solution

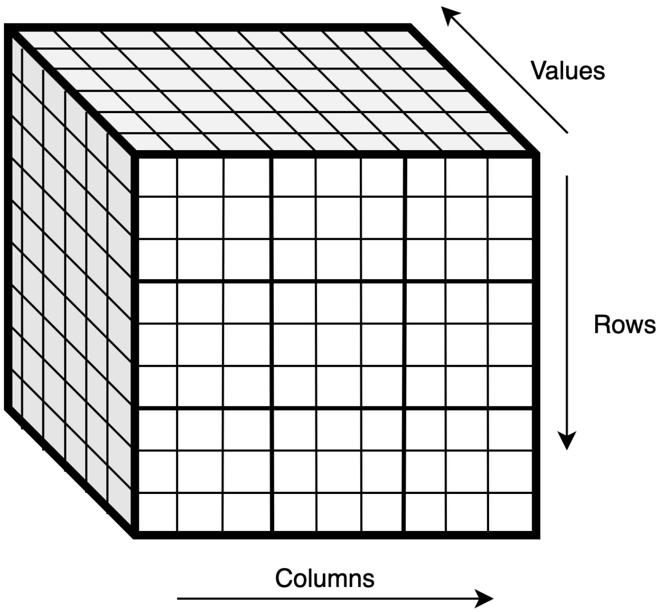
Get whether a Sudoku is solvable, and get all feasible solutions.

Taken from: Pyomo. Optimization Modeling in Python.

Formulation

Objective/Goal	Anything would work, we just need to fulfill the constraints
Constraints	Only one of a value is contained within a row Only one of a value is contained within a column Only one of a value is contained within a SubSquare (3x3 square) A cell can only contain one value
Decision Variables	Which values are used in each cell?

Modeling



A Cube of Binary (Indicator) variables.

$$y[r, c, v] = 1 \text{ if v is the value in r, c. 0 otherwise}$$

Constraints

$$\sum_{c \in Cols} y[r, c, v] = 1, \forall r \in Rows, \forall v \in Values \quad (1)$$

$$\sum_{r \in Rows} y[r, c, v] = 1, \forall c \in Cols, \forall v \in Values \quad (2)$$

$$\sum_{r,c \in SubSquares[i]} y[r, c, v] = 1, \forall i \in SubSquares \quad (3)$$

$$\sum_{v \in Values} y[r, c, v] = 1, \forall r \in Rows, \forall c \in Cols \quad (4)$$

Abstract Model

```python
model = pyo.AbstractModel()
# Starting board as internal variable
model.board = pyo.Set()
# Create sets for constraints
model.rows = pyo.RangeSet(1,9)
model.cols = pyo.RangeSet(1,9)
model.subsquares = pyo.RangeSet(1,9)
model.values_ = pyo.RangeSet(1,9)
# Create Y as Binary: row, col, val
model.y = pyo.Var(model.rows, model.cols, model.values_,
                  within=pyo.Binary)
```

```python
# This is a feasability problem (Objective doesn't matter which)
model.obj = pyo.Objective(expr=1.0)

# Constraint of row
def row_constraint(model, r, v):
    return sum(model.y[r,c,v] for c in model.cols) == 1
model.row_constraint = pyo.Constraint(
    model.rows, model.values_,
    rule=row_constraint,
    name="Constraint 1 - One of a val per row")
# Constraint of Column
def col_constraint(model, c, v):
    return sum(model.y[r,c,v] for r in model.rows) == 1
model.col_constraint = pyo.Constraint(
    model.cols, model.values_,
    rule=col_constraint,
    name="Constraint 2 - One of a val per Column")
# Constraint on SubSquare
def subsquare_constraint(subsq_to_row_col):
    def _sq_constraint(model, s, v):
        return sum(model.y[r,c,v] for (r, c) in subsq_to_row_col[s]) == 1
    return _sq_constraint
model.subsq_constraint = pyo.Constraint(
    model.subsquares, model.values_,
    rule=subsquare_constraint(subsq_to_row_col),
    name="Constraint 3 - One of a val per subsquare")
# Constraint of Values
def value_constraint(model, r, c):
    return sum(model.y[r,c,v] for v in model.values_) == 1
model.value_constraint = pyo.Constraint(
    model.rows, model.cols, rule=value_constraint,
    name="Constraint 4 - One val per Cell")
```

Additional Resources

```
# Fix initial board values
def build_model(model):
    # Fix variables based on the current board
    for (r,c,v) in model.board:
        model.y[r,c,v].fix(1)

# Remove previously seen solutions
def add_integer_cut(model):
    if not hasattr(model, "integer_cuts"):
        model.integer_cuts = pyo.ConstraintList()
    # Add the integer cut, corresponding to the current solution in the model
    # Note that if it is exactly one of the previous solution it would not satisfy the constraint
    # To satisfy the constraint, at least 1 number should be different
    cut_expr = 0.0
    for r in model.rows:
        for c in model.cols:
            for v in model.values_:
                if not model.y[r,c,v].fixed:
                    # Note, it may not be exactly 1 (Precision error)
                    if model.y[r,c,v].value >= 0.5:
                        cut_expr += (1.0 - model.y[r,c,v])
                    else:
                        cut_expr += model.y[r,c,v]
    model.integer_cuts.add(cut_expr >= 1)
```

Define 2 Sets: S_0 and S_1 :

S_0 : Indices for those variables whose current solution is 0.

S_1 : Indices for those variables whose current solution is 1.

$$\sum_{r,c,v \in S_0} y[r,c,v] + \sum_{r,c,v \in S_1} (1 - y[r,c,v]) \geq 1 \quad (5)$$

Solving a Sudoku

```
instance = model.create_instance(namespace="sudoku3", data=data)
build_model(instance)
solutions = []
while True:
    with pyo.SolverFactory("glpk") as opt:
        results = opt.solve(instance)
        if results.solver.termination_condition != pyo.TerminationCondition.optimal:
            print("All board solutions have been found")
            break
    add_integer_cut(instance)
    solutions.append(instance.clone())
print(f"Number of solutions: {len(solutions)}")

-----------------------------------------------------------------------------------
>>> WARNING: Constant objective detected, replacing with a placeholder to prevent
    solver failure.
>>> WARNING: Constant objective detected, replacing with a placeholder to prevent
    solver failure.
>>> WARNING: Constant objective detected, replacing with a placeholder to prevent
    solver failure.
>>> All board solutions have been found
>>> Number of solutions: 2
```

0	0	0	0	0	0	4	0	0
0	1	5	0	6	0	0	0	9
0	3	8	0	4	9	0	6	5
0	0	2	0	9	0	0	0	4
5	0	0	0	0	0	0	0	1
8	0	0	0	2	0	9	0	0
9	6	0	8	3	0	1	7	0
2	0	0	0	1	0	5	9	0
0	0	3	0	0	2	0	4	0

Solutions

0	0	0	0	0	0	4	0	0
0	1	5	0	6	0	0	0	9
0	3	8	0	4	9	0	6	5
0	0	2	0	9	0	0	0	4
5	0	0	0	0	0	0	0	1
8	0	0	0	2	0	9	0	0
9	6	0	8	3	0	1	7	0
2	0	0	0	1	0	5	9	0
0	0	3	0	0	2	0	4	0

6	2	9	3	5	7	4	1	8
4	1	5	2	6	8	7	3	9
7	3	8	1	4	9	2	6	5
3	7	2	5	9	1	6	8	4
5	9	6	7	8	4	3	2	1
8	4	1	6	2	3	9	5	7
9	6	4	8	3	5	1	7	2
2	8	7	4	1	6	5	9	3
1	5	3	9	7	2	8	4	6

6	2	9	7	5	3	4	1	8
4	1	5	2	6	8	7	3	9
7	3	8	1	4	9	2	6	5
3	7	2	5	9	1	6	8	4
5	9	6	4	8	7	3	2	1
8	4	1	3	2	6	9	5	7
9	6	4	8	3	5	1	7	2
2	8	7	6	1	4	5	9	3
1	5	3	9	7	2	8	4	6

Summary

- **Definition of optimization problems:** Objectives, Constraints, Variables
- **Pyomo Overview:** Open Source, Solver Agnostic, Nice docs, Tackle Multiple Optimization problems.
- **Pyomo Examples:**
  - Define multiple optimization problems
  - Use different solvers
  - Display the solutions
- **Pyomo has much more!!**
  - Non linear, Disjunctive, MINLP, Differential Algebraic Equations, etc
  - Sensitivity Analysis
  - Using commercial solvers (Gurobi, NEOS, etc)

THANK YOU!

Factored

WE ARE HIRING!

Contact:

in linkedin.com/in/cvelas31

References:

- http://www.pyomo.org/documentation
- https://pyomo.readthedocs.io/en/stable/index.html#
- osti.gov/servlets/purl/1110661
- https://github.com/Pyomo/pyomo
- Hart, William E., Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo – Optimization Modeling in Python*. Second Edition. Vol. 67. Springer, 2017.
- https://github.com/jckantor/ND-Pyomo-Cookbook

Code and presentation is located in the this repo:

- https://github.com/cvelas31/pyomo_examples

Feel free to use it, add things, etc.

Pyomo Github Commit Contributions