PyVIN, Pyomo based optimization model

Mustafa Dogan, Ellie White, Jon Herman

Civil and Environmental Engineering Department

University of California, Davis

This document describes the modeling structure of PyVIN, a new version of CALVIN modeled on Pyomo (Python Optimization modeling Objects). Pyomo supports the formulation and analysis of mathematical models for optimization applications using algebraic modeling language format (AMPL).

PyVIN separates model structure from the data. So, the core of the model does not depend on data. Similar to CALVIN, PyVIN is a linear programming model, and currently uses GLPK as solver. The objective is to minimize total cost on each arc, subject to physical and regulatory constraints.

Objective function:

$$minimize\ Z = \sum_i \sum_j \sum_k c_{ijk} X_{ijk}$$

Subject to

$$\sum_i \sum_k X_{jik} = \sum_i \sum_k a_{ijk} X_{ijk}$$

$$X_{ijk} \leq u_{ijk}$$

$$X_{ijk} \geq l_{ijk}$$

where Z is total cost of flows throughout the networks, X_{ijk} flow on the kth arc leaving node i toward node j; c_{ijk} economic costs or loss of benefits; a^{ijk} gains/losses on flows in arc ijk; u_{ijk} upper bound on arc ijk; and l_{ijk} lower bound on arc ijk.

A simple Pyomo model consists of

Set
      Set data used to define a model instance
Param
      Parameter used to define a model instance
Var
      Decision variable
Objective
      Function that is minimized or maximized
Constraint
      Expressions that impose restrictions on variable values

The Core Structure of the Model (pyvin.py)

First three lines are Python packages. First line ensures that division arguments are converted to floating number before the division. The second line makes symbols used by Pyomo known to Python.

```python
from __future__ import division
from pyomo.environ import *
import itertools
```

The following line declares that the model will be an abstract model

```python
model = AbstractModel()
```

Declare set of nodes defined in data file

```python
# Nodes in the network
model.N = Set()
```

Declare set of links within possible combination of nodes (N*N) and number of piecewise links (k)

```python
# Network arcs
model.k = Set()
model.A = Set(within=model.N*model.N*model.k)
```

Declare model parameters. For 'source' and 'sink' nodes, mass balance constraint (flow rule) is not calculated. Other parameters are upper bound, lower bound, amplitude, and cost.

```python
# Source node
model.source = Param(within=model.N)
# Sink node
model.sink = Param(within=model.N)
# Flow capacity limits
model.u = Param(model.A)
# Flow lower bound
model.l = Param(model.A)
# Link amplitude (gain/loss)
model.a = Param(model.A)
# Link cost
model.c = Param(model.A)
```

Declare decision variables, flows on each arch. A decision variable should be a positive real number.

```python
# The flow over each arc
model.X = Var(model.A, within=NonNegativeReals)
```

Objective function to be minimized. In this model, the objective is to minimize total cost, which is the sum of flow on each arc multiplied by unit cost.

```python
# Minimize total cost
def total_rule(model):
return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
model.total = Objective(rule=total_rule, sense=minimize)
```

Define upper bound constraint, such as reservoir storage capacity, canal conveyance capacity or pumping capacity

```python
# Enforce an upper bound limit on the flow across each arc
def limit_rule_upper(model, i, j, k):
return model.X[i,j,k] <= model.u[i,j,k]
model.limit_upper = Constraint(model.A, rule=limit_rule_upper)
```

Define lower bound constraint. Lower bound constraints usually represent minimum in-stream flow requirements.

```python
# Enforce an upper bound limit on the flow across each arc
def limit_rule_upper(model, i, j, k):
return model.X[i,j,k] <= model.u[i,j,k]
model.limit_upper = Constraint(model.A, rule=limit_rule_upper)
```

The following lines are used to speed up the mass balance constraint

```python
arcs_in = {}
arcs_out = {}

def arc_list_hack(model, i,j,k):
  if j not in arcs_in:
    arcs_in[j] = []
  arcs_in[j].append((i,j,k))

  if i not in arcs_out:
    arcs_out[i] = []
  arcs_out[i].append((i,j,k))
  return [0]

model._ = Set(model.A, initialize=arc_list_hack)
```

Declare mass balance constraints for nodes. For each node, flow entering to node (inflow) must be equal to flow leaving that node (outflow).

```python
# Enforce flow through each node (mass balance)
def flow_rule(model, node):
  if node in [value(model.source), value(model.sink)]:
      return Constraint.Skip
  outflow  = sum(model.X[i,j,k]/model.a[i,j,k] for i,j,k in arcs_out[node])
  inflow = sum(model.X[i,j,k] for i,j,k in arcs_in[node])
  return inflow == outflow
model.flow = Constraint(model.N, rule=flow_rule)
```

Data Structure of the Model (data.dat)

The data file contains a list of nodes, set of piecewise arc parameters (k), parameters for which mass balance constraint will not be calculated, and list of network links with their properties.

Declare list of nodes

```
set N :=
D94.2001-08-31
D94.2001-09-30
INBOUND.2001-08-31
INBOUND.2001-09-30
OUTBOUND.2001-08-31
OUTBOUND.2001-09-30
SINK
SOURCE
SR_CLE.2001-08-31
SR_CLE.2001-09-30;
```

Declare set of 'k' parameters. Currently maximum number of piecewise arcs is assumed 10

```
set k := 0 1 2 3 4 5 6 7 8 9 10;
```

Super source and super sink parameters, where all inflows come and drain, respectively

```
param source := SOURCE;
param sink := SINK;
```

Declare the matrix for network links. This matrix consists of origin node (i), terminal node (j), piecewise arc (k), cost (c), amplitude (a), lower bound (l), and upper bound (u).

```
param: A: c a l u :=
SOURCE   INBOUND.2001-08-31       0        0        1        0        1000000
INBOUND.2001-08-31      SR_CLE.2001-08-310      0        1        230.580612        230.580612
OUTBOUND.2001-08-31      SINK    0        0        1        0        1000000
D94.2001-08-31   OUTBOUND.2001-08-31      0        0        1        230.580612        230.580612
SOURCE   INBOUND.2001-09-30       0        0        1        0        1000000
INBOUND.2001-09-30      SR_CLE.2001-09-300      0        1        213.177673        213.177673
OUTBOUND.2001-09-30      SINK    0        0        1        0        1000000
D94.2001-09-30   OUTBOUND.2001-09-30      0        0        1        213.177673        213.177673
SR_CLE.2001-08-31D94.2001-08-31   0        -0.8009061426161982        1        0        71.940552
SR_CLE.2001-08-31D94.2001-08-31   1        -0.7933839317774485        1        0        35.970276
SR_CLE.2001-08-31D94.2001-08-31   2        -0.7904757806139715        1        0        35.970276
SR_CLE.2001-08-31D94.2001-08-31   3        -0.7863525091203776        1        0        35.970275000000015
SR_CLE.2001-08-31D94.2001-08-31   4        -0.6791794424930186        1        0        35.970275999999984
SR_CLE.2001-08-31D94.2001-08-31   5        -0.6013748129149743        1        0        35.97027600000001
SR_CLE.2001-08-31D94.2001-08-31   6        0        1        0        1000000
SR_CLE.2001-09-30D94.2001-09-30   0        -0.7420960355688229        1        0        69.700592
SR_CLE.2001-09-30D94.2001-09-30   1        -0.7350739253575176        1        0        34.850288000000006
SR_CLE.2001-09-30D94.2001-09-30   2        -0.7327835074265064        1        0        34.850303999999994
SR_CLE.2001-09-30D94.2001-09-30   3        -0.7305389589254676        1        0        34.850280999999995
SR_CLE.2001-09-30D94.2001-09-30   4        -0.7269706403641447        1        0        34.850296000000014
SR_CLE.2001-09-30D94.2001-09-30   5        -0.7197610889732475        1        0        34.850295999999986
SR_CLE.2001-09-30D94.2001-09-30   6        0        1        0        1000000;
```