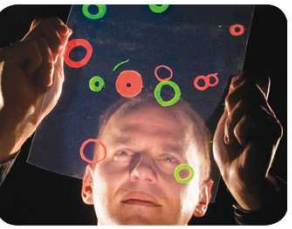


Pyomo 4.1

William Hart
Center for Computing Research
Sandia National Laboratories
wehart@sandia.gov

SAND2015-5766C

Sandia National Laboratories

Exceptional service in the national interest

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Pyomo Overview

Idea: a Pythonic framework for formulating optimization models

- Provide a natural syntax to describe mathematical models
- Formulate large models with a concise syntax
- Separate modeling and data declarations
- Enable data import and export in commonly used formats

Highlights:

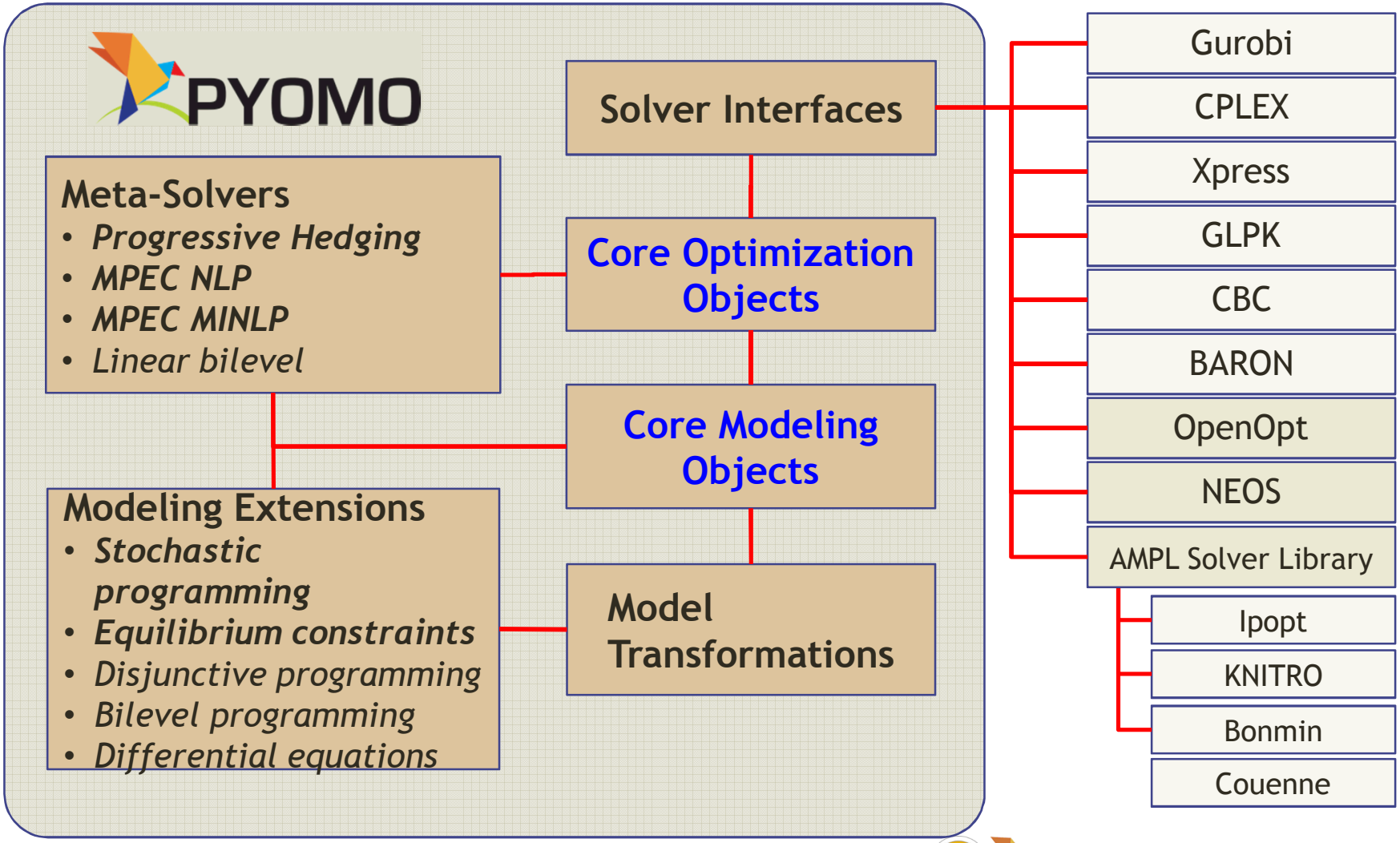
- Python provides a clean, intuitive syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
# simple.py
from pyomo.environ import *

M = ConcreteModel()
M.x1 = Var()
M.x2 = Var(bounds=(-1,1))
M.x3 = Var(bounds=(1,2))
M.o  = Objective(
       expr=M.x1**2 + (M.x2*M.x3)**4 + \
            M.x1*M.x3 + \
            M.x2*sin(M.x1+M.x3) + M.x2)

model = M
```

Pyomo at a Glance



Who Uses Pyomo?

- Students
  - Rose-Hulman, UC Davis, U Texas, Iowa State, NPS
- Researchers
  - Sandia National Labs, Lawrence Livermore National Lab, Argonne National Lab, Los Alamos National Lab, UC Davis, TAMU, Rose-Hulman, UT, USC, GMU, Iowa State, NCSU, Purdue U, U Washington, NPS, U de Santiago de Chile, U Pisa, Federal Energy Regulatory Agency, …
- Software Projects
  - Minpower – Power systems toolkit
  - OptiType - an HLA genotyping algorithm
  - SolverStudio – Excel plugin for optimization modeling
  - TEMOA – Energy economy optimization models
  - Water Security Toolkit – Planning/Response for water contamination

What's New in Pyomo 4.1

- Changes to model construction and results management
  - Includes a performance improvement
- Pyomo configuration files
- New expression trees
- Resolved issues with MPEC solver interfaces
- Various API Changes
  - Model transformation logic
  - Block, SOSConstraint, Suffix components

Model Construction (1)

Pyomo 4.0

```
model = AbstractModel()                  # Create model
…                                                # Add model components
instance = model.create()                # Create instance
results = solver.solve(instance)         # Apply optimizer
```

Pyomo 4.1

```
model = AbstractModel()                  # Create model
…                                                # Add model components
instance = model.create_instance()       # Create instance
results = solver.solve(instance)         # Apply optimizer
```

Model Construction (2)

Pyomo 4.0

```
instance = ConcreteModel()          # Create model
…                                           # Add model components
instance = instance.create()        # Preprocess instance
results = solver.solve(instance)    # Apply optimizer
```

Pyomo 4.1

```
instance = ConcreteModel()          # Create model
…                                           # Add model components
results = solver.solve(instance)    # Apply optimizer
```

Results Management (1)

Pyomo 4.0

```
results = solver.solve(model)          # Results object stores
                                       #   solutions
model.load(results)                    # Load solutions into
                                       #   the model
```

Pyomo 4.1

```
results = solver.solve(model)          # Solutions are stored in
                                       #   the model object.  One
                                       #   solution is loaded.
                                       # Results object contains
                                       #   meta-data.
```

Results Management (2)

Pyomo 4.0

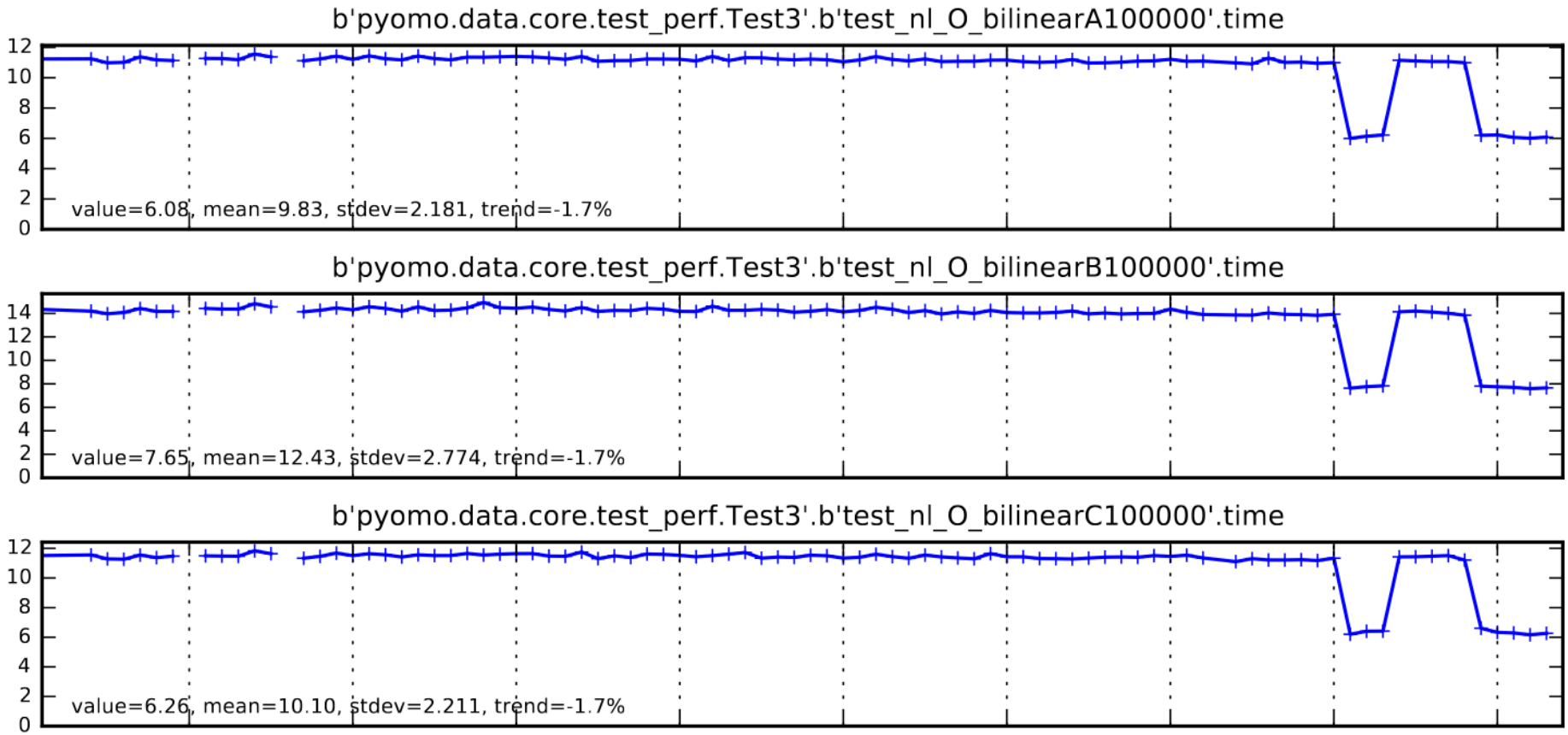
```
results = solver.solve(model)          # Results object stores
                                       #   solutions
model.update_results(results)          # Re-load results with
                                       #   model labels
```

Pyomo 4.1

```
results = solver.solve(model)          # Solutions are stored in
                                       #   the model object.  One
                                       #   solution is loaded.
model.solutions.store_to(results)      # Store solution to model
                                       #   with model labels
```

Performance Impact

Note: avoiding presolve for NL files significantly improved model generation



Pyomo Configuration Files

Idea: express complex optimization solves with a configuration file

- pyomo solve config.json
- pyomo solve config.yaml

- This will simplify the user experience
- This reflects current user practice (e.g. caching complex pyomo scripts in system shells)

Note: the configuration logic may depend on the solver

- Solver-specific options
- Solver-specific data blocks
  - E.g. integer program vs stochastic program

Configuration File Example (1a)

With command-line options:

```
pyomo solve –solver=glpk pmedian.py pmedian.dat
```

With YAML configuration file:

```
pyomo solve config.yaml
```

With the file config.yaml:

```
data:
  files: pmedian.dat
model:
  filename: pmedian.py
solvers:
  - solver name: glpk
```

Configuration File Example (1b)

With JSON configuration file:

```
pyomo solve config.json
```

With the file config.json:

```
{
    "data": {
        "files": "pmedian.dat"
        },
    "model": {
        "filename": "pmedian.py"
        },
    "solvers": [
        { "solver name": "glpk" }
        ]
}
```

Configuration File Example (2)

With command-line options:

```
pyomo solve --solver=glpk pmedian.py pmedian.dat \
            --solver-options="mipgap=0.02 cuts="
```

Corresponding YAML configuration file:

```
data:
  files: pmedian.dat
model:
  filename: pmedian.py
solvers:
  - solver name: glpk
    options:
      mipgap: 0.02
      cuts:
```

Pyomo Expression Trees

Expression trees are the data structure used to represent algebraic expressions in Pyomo

Old Design

- *n*-ary trees with 2 principal node types:
  - Sum: [argument list], [(constant) coefficient list]
  - Product: [numerator list], [denominator list]
- ☺ Very compact trees
- ☺ Only two primary data types (no subtraction, no division)
- ☹ Complicated to generate
- ☹ "In-place" expression generation leads to expression entangling:

Expression Entanglement

Problem: intermediate variables create an opportunity for expression trees to change each other

```
a = model.x[1] + model.x[2] + model.x[3]
b = a + model.x[4]
```

- Naïve implementation leaves `a == b == sum(x[1] … x[4])`

Solution: (Coopr 3.4)

- Check for external references to expressions before combining
- Use the Cpython `getrefcount()`

NOTE:

- Non-Cpython implementations do not implement reference checking
- This design prevents Pyomo from working on other Python implementations:
  - IronPython, Jython and PyPy

Pyomo 4.1 n-ary Trees

- 😐 5 principal node types (Add, Multiply, Divide, Negate, Linear)
  - ☺ More consistent layout
    - Except for Linear expressions: [argument list], {argument coefficient map}
  - ☺ ☺ Direct-to-linear expressions
    - Coefficient map supports constant (but mutable) expressions
- ☺ ☺ No longer reliant on getrefcount()
- ☹ Expression entanglement still an issue
  - Manage through pointers to parent nodes (challenging bookkeeping)
  - But, now we can walk *up* trees as well
- ☺ Reasonably fast to generate
  - No more special cases for subtraction, division
  - Preserve compact, balanced trees
- ☺ Reasonably fast to traverse
- ☺ More memory efficient

Configuration File Example (2)

With command-line options:

```
pyomo solve --solver=glpk \
            --generate-config-template=template.yaml
```

A template configuration file is generated:

```
# Configuration for a canonical model construction and optimization sequence
data:
  files: []                  # Model data files
  namespaces: []                 # A namespace that is used to select data in
                           #   Pyomo data files.
model:
  filename: null               # The Python module that specifies the model
  object name: model             # The name of the model object that is
                           #   created in the specified Pyomo module
  type: null                   # The problem type
  options:                     # Options used to construct the model
  linearize expressions: false   # An option intended for use on linear or
                           #   mixed-integer models in which expression
```

Modeling MPECs

Mathematical Programming with Equilibrium Constraints (MPEC)

- Engineering design, economic equilibrium, multilevel games
- Feasible region may be nonconvex and disconnected

Equilibrium Constraints

- Variational inequalities
- Complementarity conditions
- Optimality conditions (for bilevel problems)

Updates to MPEC Solvers in Pyomo 4.1

- Solver names
- NL writer
- PATH interface

MPEC Example

MacMPEC problem ralph1:

$$\begin{aligned} \min \quad & 2x - y \\ \text{s.t.} \quad & y \geq 0 \perp y \geq x \\ & x, y \geq 0 \end{aligned}$$

```
# ralph1.py
from pyomo.environ import *
from pyomo.mpec import *

model = ConcreteModel()
model.x = Var(within=NonNegativeReals)
model.y = Var(within=NonNegativeReals)

model.o = Objective(expr=2*model.x - model.y)

model.c = Complementarity(
            expr=complements(0<=model.y, model.y>=model.x) )
```

MPEC Solvers (1)

Solver mpec_nlp

- Apply nonlinear reformulation

$$\begin{aligned} \min \quad & 2x - y \\ \text{s.t.} \quad & y(y - x) \leq \varepsilon \\ & x, y \geq 0 \end{aligned}$$

- Iteratively tighten the tolerance value

Example configuration file:

```
model:
  filename: ralph1.py
solvers:
  - solver name: mpec_nlp
    options:
      epsilon_initial: 1e-1
      epsilon_final:   1e-7
```

MPEC Solvers (2)

Solver mpec_minlp

- Apply disjunctive reformulation

$$\begin{aligned} \min \quad & 2x - y \\ \text{s.t.} \quad & \begin{pmatrix} z \\ y \geq 0 \\ y - x = 0 \end{pmatrix} \vee \begin{pmatrix} \neg z \\ y = 0 \\ y \geq x \end{pmatrix} \\ & x, y \geq 0 \\ & z \in \{0,1\} \end{aligned}$$

- Use a big-M reformulation to create a MIP

Example configuration file:

```
model:
  filename: ralph1.py
solvers:
  - solver name: mpec_minlp
    options:
      solver: glpk
```

MPEC Solvers (3)

Solver path

- The PATH solver loads problems through AMPL NL files
- Pyomo has been extended to generate NL files with complementarity conditions
- A simple wrapper for PATH is provided in Pyomo

Example configuration file:

```
model:
  filename: ralph1.py
solvers:
  - solver name: path
```

Core Contributors to Pyomo 4.1

- Gabe Hackebeil
- William Hart
- John Siirola
- Jean-Paul Watson
- David Woodruff

For More Information

Project homepage

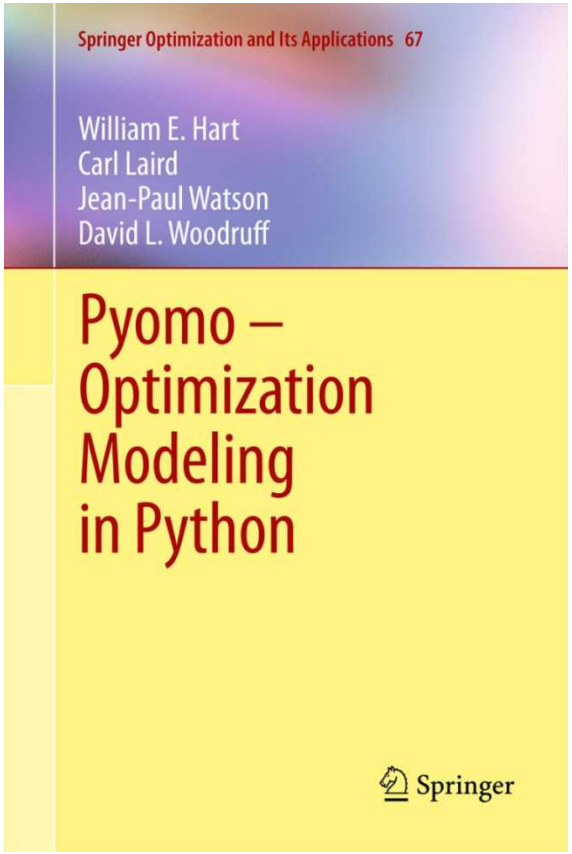
- http://www.pyomo.org
- https://software.sandia.gov/pyomo

Mailing lists

- "pyomo-forum" Google Group
- "pyomo-developers" Google Group

"The Book"

- New version this fall for Pyomo 5.0



Mathematical Programming Computation paper:

- Pyomo: Modeling and Solving Mathematical Programs in Python (3(3), 2011)