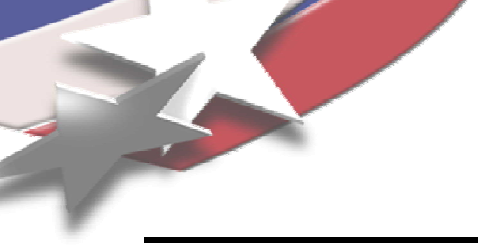


COOPR

SAND2013-9194C

Recent Developments in Coopr

William E. Hart, John D. Siirola, and Jean-Paul Watson

Discrete Math and Complex Systems Department

Sandia National Laboratories

Albuquerque, NM USA

INFORMS Annual Meeting

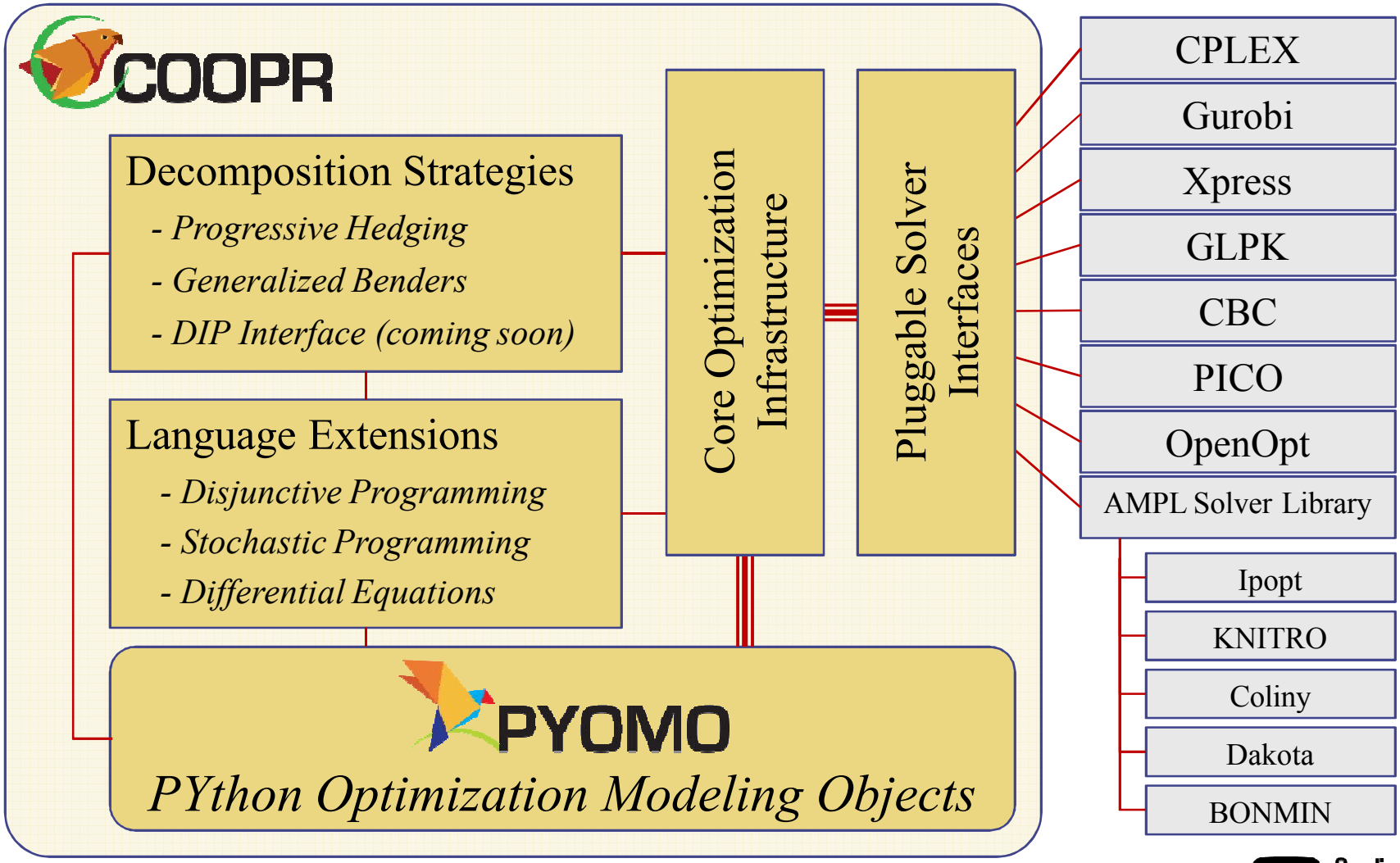
6-9 October, 2013

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.

NNSA National Nuclear Security Administration

Sandia National Laboratories

Coopr: *a COmmon Optimization Python Repository*



Why Coopr?

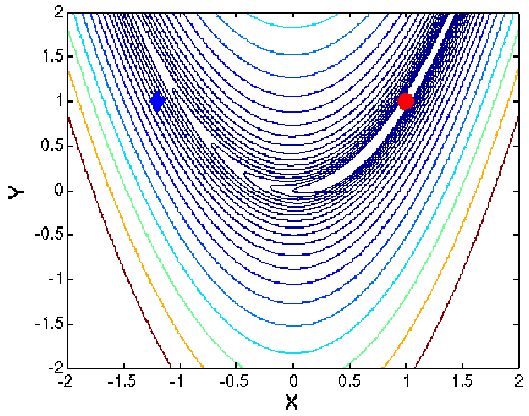
- *Not speed* (of model generation)

- *Flexibility and agility*
  - Natural problem representations
  - Problem transformations
  - “Meta-solvers”
  - Rapid prototyping

Pyomo: object-oriented math programming

- rosenbrock.py:

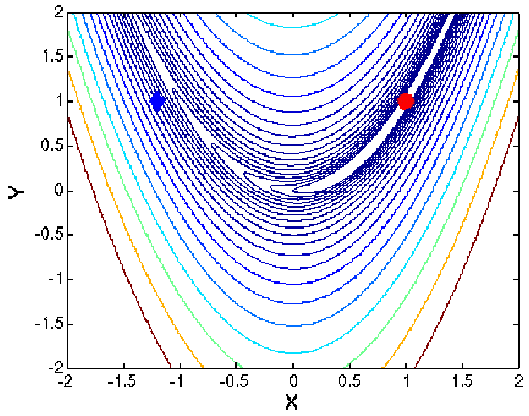
```
from coopr.pyomo import *

model = ConcreteModel()

model.x = Var( initialize=-1.2, bounds=(-2, 2) )
model.y = Var( initialize= 1.0, bounds=(-2, 2) )

model.obj = Objective(
    expr= (1-model.x)**2 + 100*(model.y-model.x**2)**2,
    sense= minimize )
```



Pyomo: object-oriented math programming

- Solve the model:
  - The pyomo command

```
% pyomo rosenbrock.py --solver=ipopt --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[    0.00] Applying solver
[    0.03] Processing results
    Number of solutions: 1
    Solution Information
      Gap: <undefined>
      Status: optimal
      Function Value: 2.98956421871e-17
    Solver results file: results.yml

====================================================
Solution Summary
====================================================
Model unknown
  Variables:
    x : Size=1, Index=None, Domain=Reals
        Key  : Lower : Value          : Upper : Initial : Fixed : Stale
        None :    -2 : 0.999999994543 :     2 :    -1.2 : False : False
    y : Size=1, Index=None, Domain=Reals
        Key  : Lower : Value          : Upper : Initial : Fixed : Stale
        None :    -2 : 0.999999989052 :     2 :     1.0 : False : False
  Objectives:
    obj : Size=1
        Key  : Value
        None : 2.98956421871e-17
  Constraints:
    None
[    0.03] Applying Pyomo postprocessing actions
[    0.03] Pyomo Finished
```



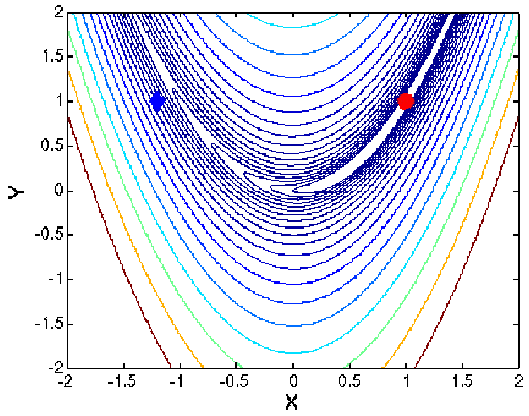
Pyomo: object-oriented math programming

- Solve the model:
  - The pyomo command
  - Custom driver

driver.py

```
from coopr.opt import SolverFactory
from rosenbrock import model
solver = SolverFactory('ipopt')
results = solver.solve(model)
print "Solution Status: %s" % results['Solution'][0]['Status']
model.load(results)
model.display()
```

```
% python driver.py
Solution Status: optimal
Model unknown

  Variables:
    x : Size=1, Index=None, Domain=Reals
        Key  : Lower : Value          : Upper : Initial : Fixed : Stale
        None :    -2 : 0.999999994543 :     2 :    -1.2 : False : False
    y : Size=1, Index=None, Domain=Reals
        Key  : Lower : Value          : Upper : Initial : Fixed : Stale
        None :    -2 : 0.999999989052 :     2 :     1.0 : False : False

  Objectives:
    obj : Size=1
        Key  : Value
        None : 2.98956421871e-17

  Constraints:
    None
```



Team, Collaborators, (Known) Users

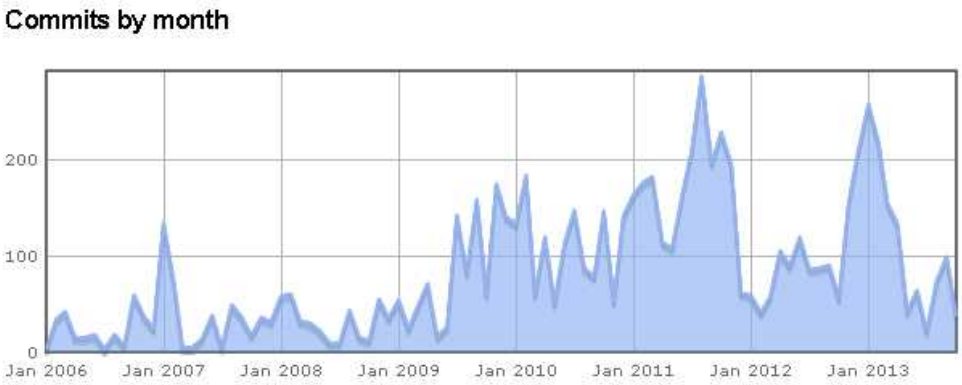
- Sandia National Laboratories
  - Bill Hart
  - Jean-Paul Watson
  - John Siirola
  - David Hart
  - Tom Brounstein
- University of California, Davis
  - Prof. David L. Woodruff
  - Prof. Roger Wets
- Texas A&M University
  - Prof. Carl D. Laird
  - Daniel Word
  - James Young
  - Gabe Hackebeil
- Carnegie Mellon University
  - Bethany Nicholson
- Texas Tech University
  - Zev Friedman
- Rose Hulman Institute
  - Tim Ekl
- William & Mary
  - Patrick Steele
- North Carolina State
  - Kevin Hunter

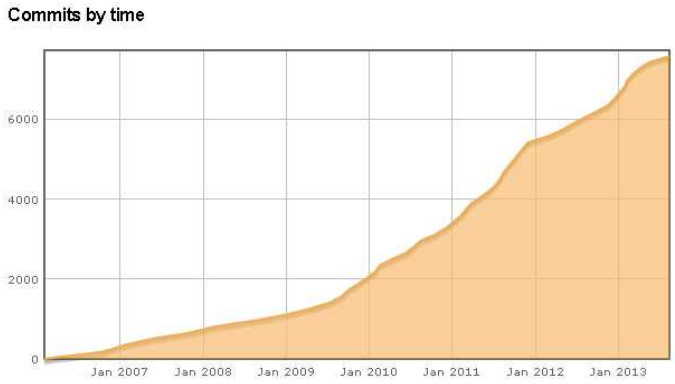
(Known) users:
- University of California, Davis
- Texas A&M University
- University of Texas
- Rose-Hulman Institute of Technology
- University of Southern California
- George Mason University
- Iowa State University
- N.C. State University
- University of Washington
- Naval Postgraduate School
- Universidad de Santiago de Chile
- University of Pisa
- Argonne National Lab
- Lawrence Livermore National Lab
- Los Alamos National Lab
- Federal Energy Regulatory Agency

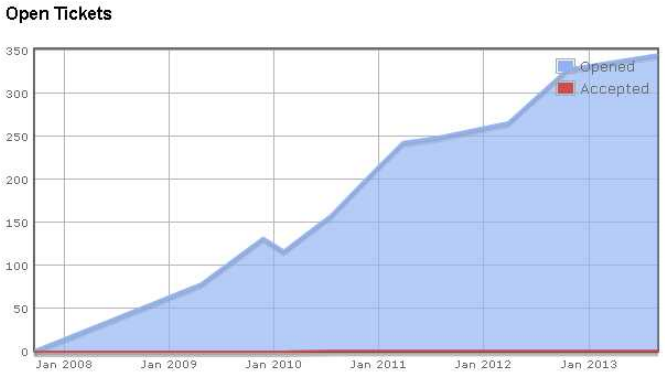
Development, Community activity

- Coopr Forum membership up 50% in 2013
  - Active discussion list (~200 discussions in the last year)

- Active developer community







Sandia National Laboratories

What's New in Coopr?

- Releases:
  - Current: 3.3.7114 (2 March 2013)
  - Upcoming: 3.4 (end of October)

- Expression objects
- Suffix support
- Differential Algebraic Equation support
- Data Portals
- PH lower bounds
- Performance improvements

Expression objects

- Where credit is deserved… Gabriel Hackebeil, Texas A&M
- Pyomo generates expression trees (via operator overloading) to represent expressions in the user model
  - all expressions are simple trees (shared sub-trees is disallowed)
- Expression objects
  - First-class model components
    - Named, attached to models, preserved when cloning the model
  - Used in the same manner as Var, Param

From coopr/data/cute/hs085_cute.py

```
model.y1  = Expression(initialize= model.x[2] + model.x[3] + 41.6 )
model.y7  = Expression(initialize= model.c8/model.y1 )
model.con2 = Constraint(expr= model.y1- 213.1 >= 0 )
model.con3 = Constraint(expr= 405.23 - model.y1 >=0 )
model.con35 = Constraint(expr= b[7] - model.y7 >=0 )
```

Suffix Support

- Where credit is deserved… Gabriel Hackebeil, Texas A&M
- (AMPL-style) Suffixes are now first-class model objects
  - As first class objects, Suffixes are preserved when cloning model

```
model.sfx = Suffix(
    direction= { LOCAL, IMPORT, EXPORT, IMPORT_EXPORT },
    datatype= {INT, FLOAT, None} )
```

- Values are set / retrieved using

```
model.sfx.setValue(model.x, value)
model.sfx.getValue(model.x)
```

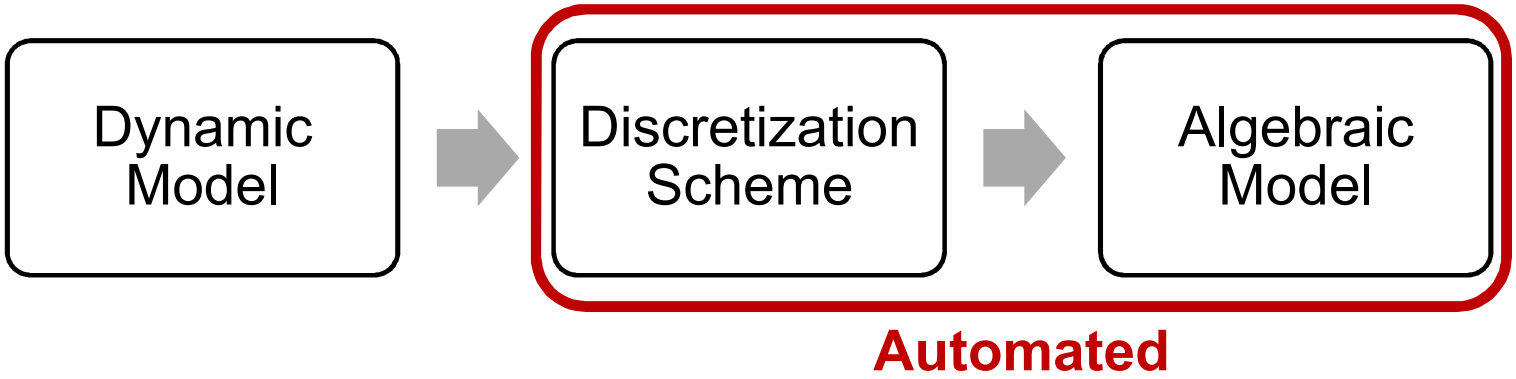
- Common use: retrieving dual information:

```
model.dual = Suffix(direction=Suffix.IMPORT)
print "Constraint c[1] Dual=%s" % \
    model.dual.getValue(model.c[1])
```

Differential-Algebraic Models

- Where credit is deserved…
  - Bethany Nicholson, Carnegie Mellon University

- Provide support for optimizing dynamic models
  - Implemented as a plug-in (completely independent of Pyomo)
  - Modeling components for differential equations
  - Automated transformations to discretize model to (N)LP



General Syntax

- Import the extension

```
from coopr.dae import *
```

- DifferentialSet

```
m.t = DifferentialSet(bounds=(0,10))
m.t = DifferentialSet(initialize=[1,2,3,4])
m.t = DifferentialSet()
```

- Differential

```
m.xdot = Differential(
        dvar=m.x,
        rule=_xdot,
        dset=m.t,
        bounds=(0,10),
        initialize=_init_xdot )
```

Optimal Control Example (1)

$$\min \; x_3(t_f)$$
$$s.t. \quad \dot{x_1} = x_2$$
$$\dot{x_2} = -x_2 + u$$
$$\dot{x_3} = x_1^2 + x_2^2 + 0.005 * u^2$$
$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$
$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$
$$t_f = 1$$

```
from coopr.pyomo import *
from coopr.dae import *

m = ConcreteModel()
m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)

def _obj(m):
    return m.x3[1]
m.obj = Objective(rule=_obj)

def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m,t):
    return -m.x2[t]+m.u[t]
m.x2dot = Differential(dv=m.x2, rule=_x2dot)

def _x3dot(m,t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, rule=_x3dot)

def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (2)

$$\min \; x_3(t_f)$$
$$s.t. \quad \dot{x_1} = x_2$$
$$\dot{x_2} = -x_2 + u$$
$$\dot{x_3} = x_1^2 + x_2^2 + 0.005 * u^2$$
$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$
$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$
$$t_f = 1$$

```
from coopr.pyomo import *
from coopr.dae import *

m = ConcreteModel()
```


```
from coopr.pyomo import *
from coopr.dae import *
```


```
m.obj = Objective(rule=_obj)

def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m,t):
    return -m.x2[t]+m.u[t]
m.x2dot = Differential(dv=m.x2, rule=_x2dot)

def _x3dot(m,t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, rule=_x3dot)

def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (3)

$$\min \; x_3(t_f)$$
$$s.t. \quad \dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_2 + u$$
$$\dot{x}_3 = x_1^2 + x_2^2 + 0.00$$
$$x_2 - 8 * (t - 0.5)^2 +$$
$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$
$$t_f = 1$$

```
from coopr.pyomo import *
from coopr.dae import *

m = ConcreteModel()
m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)

def _obj(m):
```

```
m = ConcreteModel()
m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)
```

```
def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (4)

$$\min \; x_3(t_f)$$

$$s.t. \qquad \dot{x_1} = x_2$$

$$\dot{x_2} = -x_2 + u$$

$$\dot{x_3} = x_1^2 + x_2^2 + 0.005 * u^2$$

$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

$$t_f = 1$$

```
from coopr.pyomo import *
from coopr.dae import *

m = ConcreteModel()
m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)

def _obj(m):
    return m.x3[1]
m.obj = Objective(rule=_obj)

def _x1dot(m,t):
    return m.x2[t]
m.x1dot = DifferentialDerivative(dy=m.x1, rule=_x1dot)
```


```
def _obj(m):
    return m.x3[1]
m.obj = Objective(rule=_obj)
```


```
def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (5)

$$\min \; x_3(t_f)$$

$$s.t. \quad \dot{x_1} = x_2$$

$$\dot{x_2} = -x_2 + u$$

$$\dot{x_3} = x_1^2 + x_2^2 + 0.005 * u^2$$

$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$

```
from coopr.pyomo import *
from coopr.dae import *

m = ConcreteModel()
m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)

def _obj(m):
    return m.x3[1]
m.obj = Objective(rule=_obj)

def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m,t):
```

```
def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, dset=m.t,
      rule=_x1dot)
```

```
def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (6)

$$\min$$

$$s.t. \quad \dot{x}_1 = x_2$$

$$\dot{x}_2 = -x_2 + u$$

$$\dot{x}_3 = x_1^2 + x_2^2 + 0.005 * u^2$$

$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

$$t_f = 1$$

```
def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)
```

```
def _obj(m):
    return m.x3[1]
m.obj = Objective(rule=_obj)

def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m,t):
    return -m.x2[t]+m.u[t]
m.x2dot = Differential(dv=m.x2, rule=_x2dot)

def _x3dot(m,t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, rule=_x3dot)

def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example (7)

$$\min \; x_3(t_f$$
$$s.t. \quad \dot{x}_1 =$$
$$\dot{x}_2 = -x_2 +$$
$$\dot{x}_3 = x_1^2 + x_2^2 + 0.($$
$$x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0$$
$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$
$$t_f = 1$$

```
from coopr.pyomo import *
from coopr.dae import *

def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)

m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m,t):
    return -m.x2[t]+m.u[t]
m.x2dot = Differential(dv=m.x2, rule=_x2dot)

def _x3dot(m,t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, rule=_x3dot)

def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <= 0
m.con = Constraint(m.t, rule=_con)
```

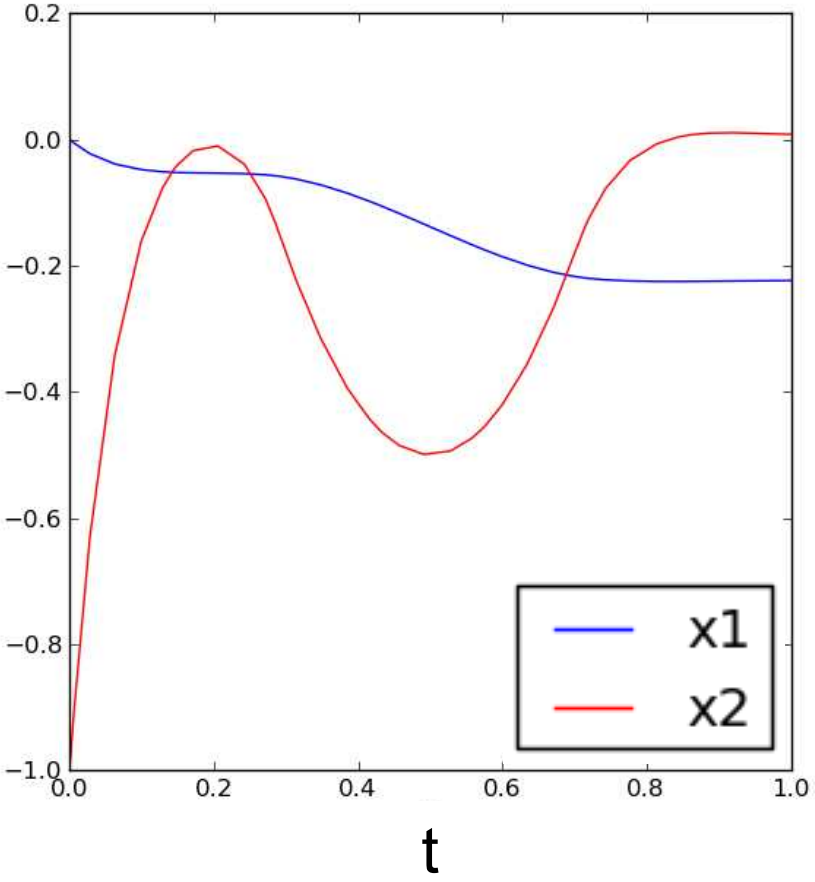

```
def _init_conditions(m,i):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(
                        rule=_init_conditions)
```

Optimal Control Example: driver script

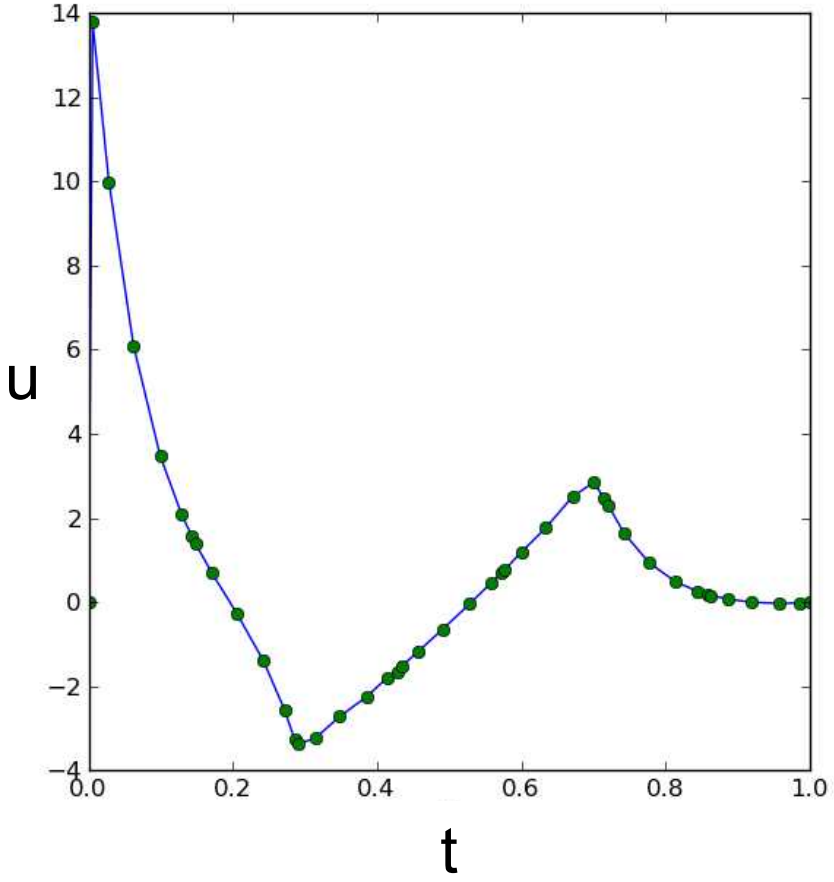
```
from coopr.pyomo import *
from coopr.dae import *
from coopr.opt import SolverFactory
from coopr.dae.colloc import Collocation_Discretization

# Import model
from example import m as model

# Create model instance
instance = model.create()

# Discretize instance using radau collocation
discretizer = Collocation_Discretization(instance)
discretized_instance = discretizer.apply(nfe=7, ncp=6)

# Solve discrete model
opt = SolverFactory('ipopt')
results = opt.solve(discretized_instance)

# Load Results
discretized_instance.load(results)
```

Optimal Control Example: results

Differential Variables

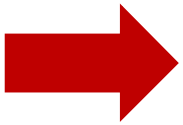


Control Variable



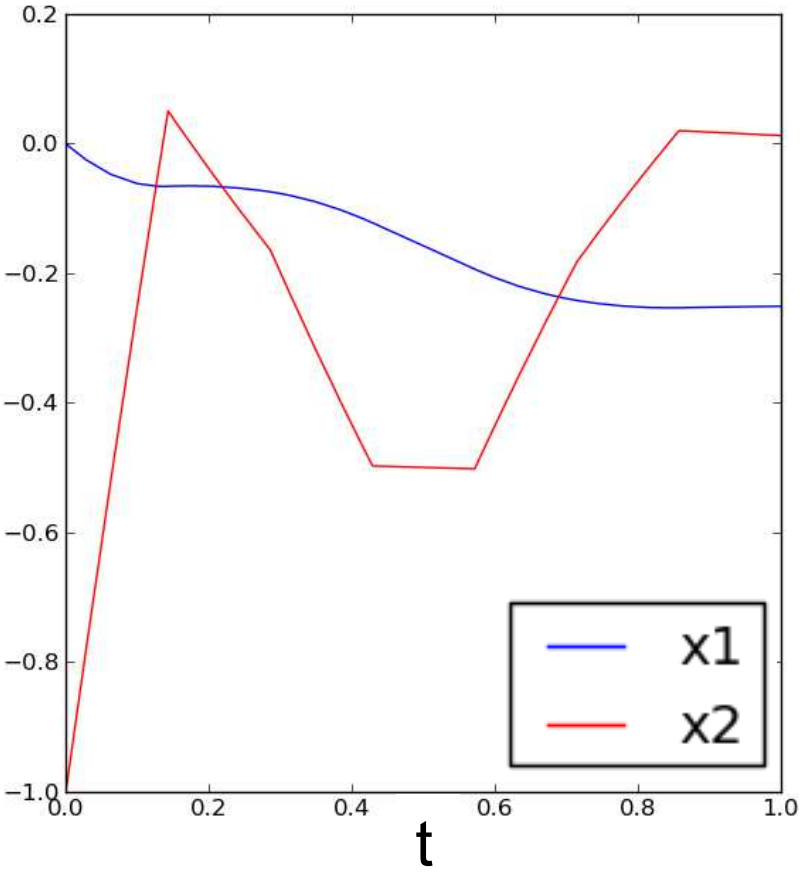
Optimal Control Example: alternate driver

```
from coopr.pyomo import *
from coopr.dae import *
from coopr.opt import SolverFactory
from coopr.dae.colloc import Collocation_Discretization
from smallExample import model

instance = model.create()

# Discretize instance using radau collocation
discretizer = Collocation_Discretization(instance)
instance = discretizer.apply(nfe=7, ncp=6)

# Control variable u made constant over each finite element
instance = discretizer.reduce_collocation_points(
    var=instance.u, ncp=1, diffset=instance.t )

opt = SolverFactory('ipopt')
results = opt.solve(instance)

instance.load(results)
```

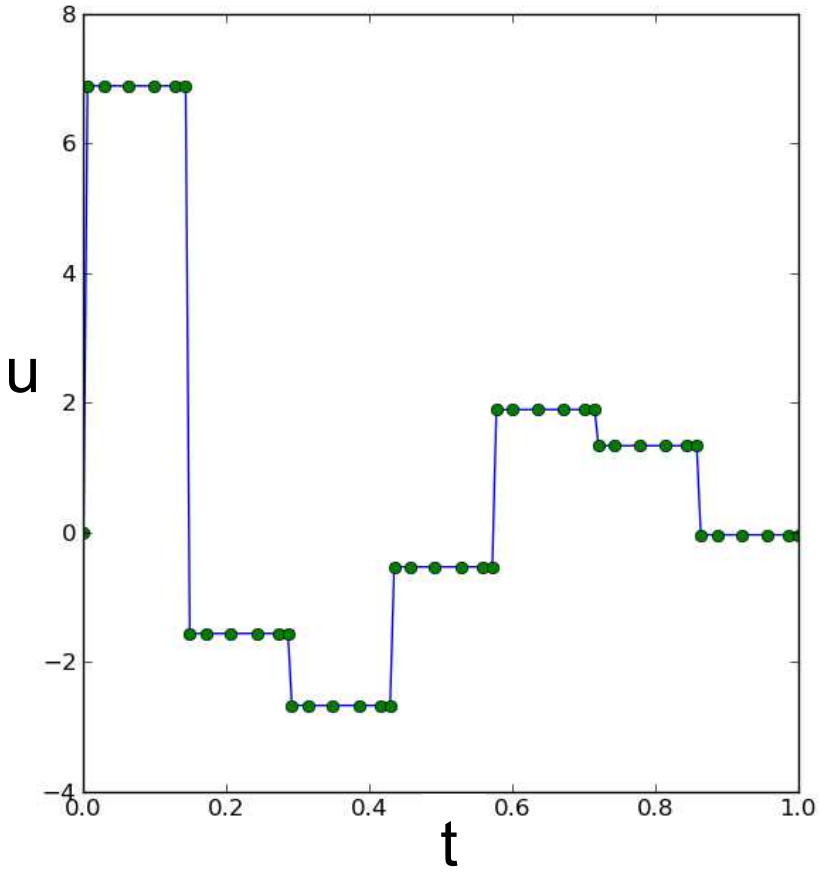


Optimal Control Example: results

Differential Variables



Control Variable



DataPortal: unified data import mechanism

- Where credit is deserved… Bill Hart, Sandia

- Observations:
  - Scripting has become a common activity for Coopr users
  - Original data management mechanism limited

- Data Portal Objects
  - Support loading and storing data
  - I/O for set, parameter, variables, suffix data, etc.
  - Explicitly manage connections to data sources
  - Initialize both abstract and concrete models
  - Data sources: csv, tab, odbc, xls/xlsm/xlsb/xlsx, yaml, json, xml, and Pyomo data command files

DataPortal Example: Abstract Model

```
M = AbstractModel()
M.A = Set(dimen=2)
M.q = Param(M.A)
M.p = Param(M.A)

dp = DataPortal()
dp.load(filename='foo.csv', param=(M.p,M.q), index=M.A)

Instance = M.create(dp)
```

foo.csv

```
A1,A2,p,q
a,b,1,2
c,d,3,4
```

DataPortal Example: Concrete Models

```
db = DataPortal()
db.load(filename='foo.csv', param='q', index='A')
db.load(filename='bar.tab', param='p')

M = ConcreteModel()
M.A = Set(dimen=2, initialize=db.data('A'))
M.q = Param(M.A, initialize=db.data('q'))
M.p = Param(M.A, initialize=db.data('p'))
Instance = M.create()
```

foo.csv

```
A1,A2,q
a,b,2
c,d,4
```

bar.tab

A1	A2	p
a	b	1
c	d	3

Lower bounding in PH

- Where credit is deserved…
  - Dinakar Gade, Sarah Ryan; Iowa State
  - Gabriel Hackebeil; Texas A&M
  - Jean-Paul Watson; Sandia
  - Roger Wets, David Woodruff; UC Davis

- Progressive Hedging (PH)
  - “Historically” PH is a heuristic for MIP
  - A lower bound on optimal cost can be obtained with approximately the same effort as one PH iteration.

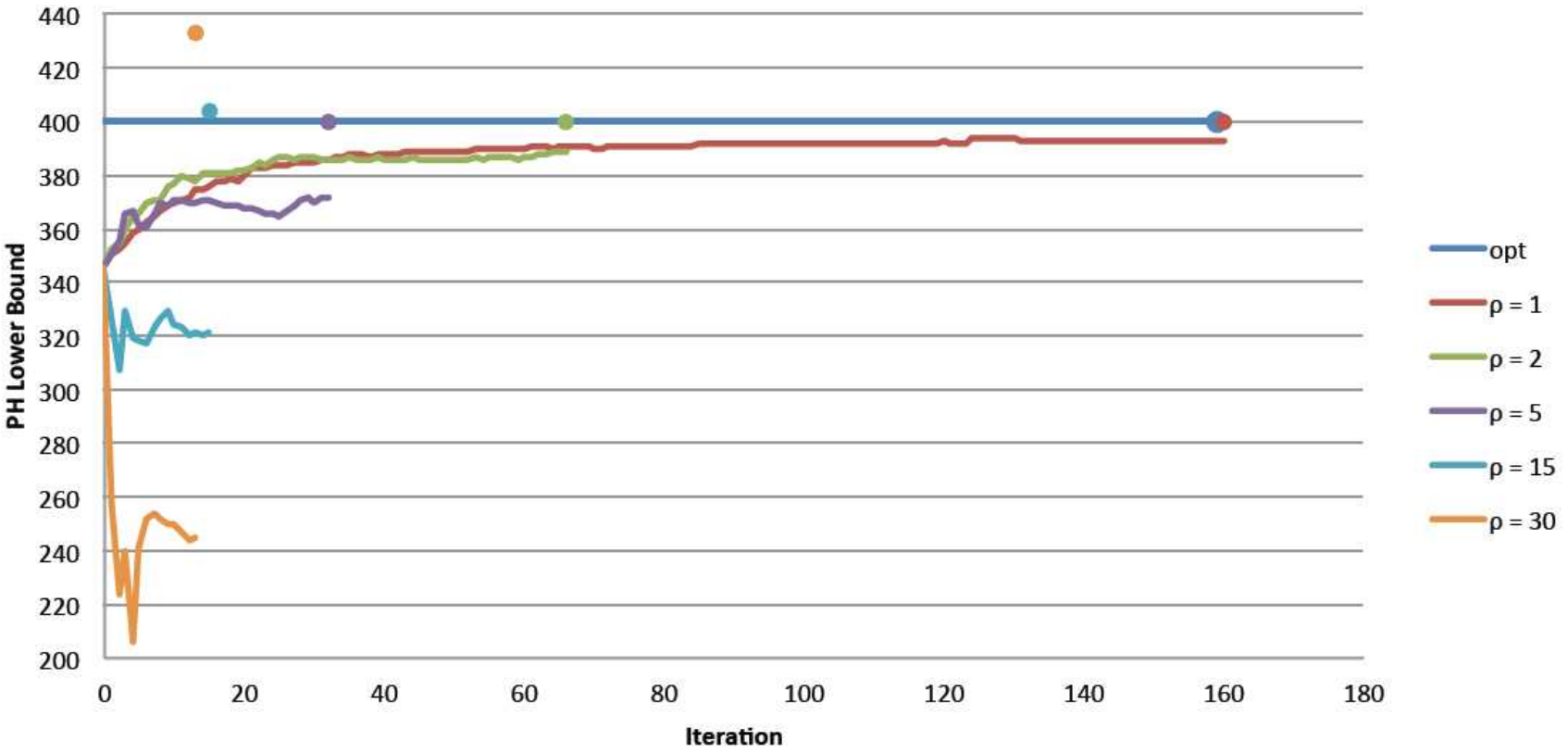
Obtaining PH Lower Bounds

- Implementation:
  - `phboundextension` computes a lower bound in any iteration.
  - Can compute a bound once every N iterations.
  - Can assign the lower bounding task to separate processors, one for each (bundle of) scenario sub-problem(s).

- How?

```
runph […] --user-defined-extension=coopr.pysp.phboundextension
```

PH Lower / Upper bounds



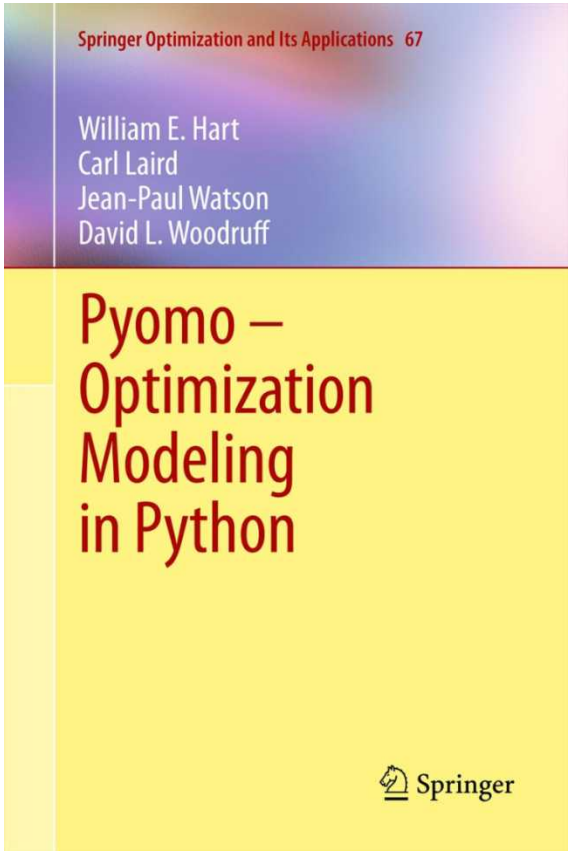
(Performance)

- Recent work has had significant focus on memory
  - 20% reduction over Coopr 3.2

- Removing bottlenecks
  - Direct solver interfaces
  - Transformations, especially for
    - Block manipulation
    - Disjunctive programming
    - Stochastic programming
  - Data management
    - Param initialization and manipulation
    - Data and model I/O

For More Information…

- Project homepage
  - http://software.sandia.gov/coopr
- Mailing lists
  - "coopr-forum" Google Group
  - "coopr-dev" Google Group
- "The Book"
- Online documentation
  - "Getting Started with Coopr"
  - "Coopr Installation Guide"



- Mathematical Programming Computation papers
  - Pyomo: Modeling and Solving Mathematical Programs in Python (3(3), 2011)
  - PySP: Modeling and Solving Stochastic Programs in Python (4(2), 2012)