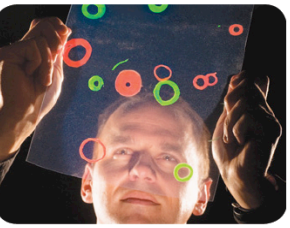
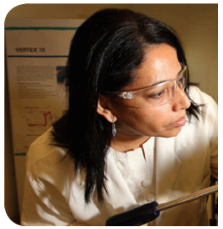




Sandia National Laboratories

Exceptional service in the national interest

Structured Modeling and Decomposition Methods in Pyomo

Bethany Nicholson

Carl Laird
John D. Siirola

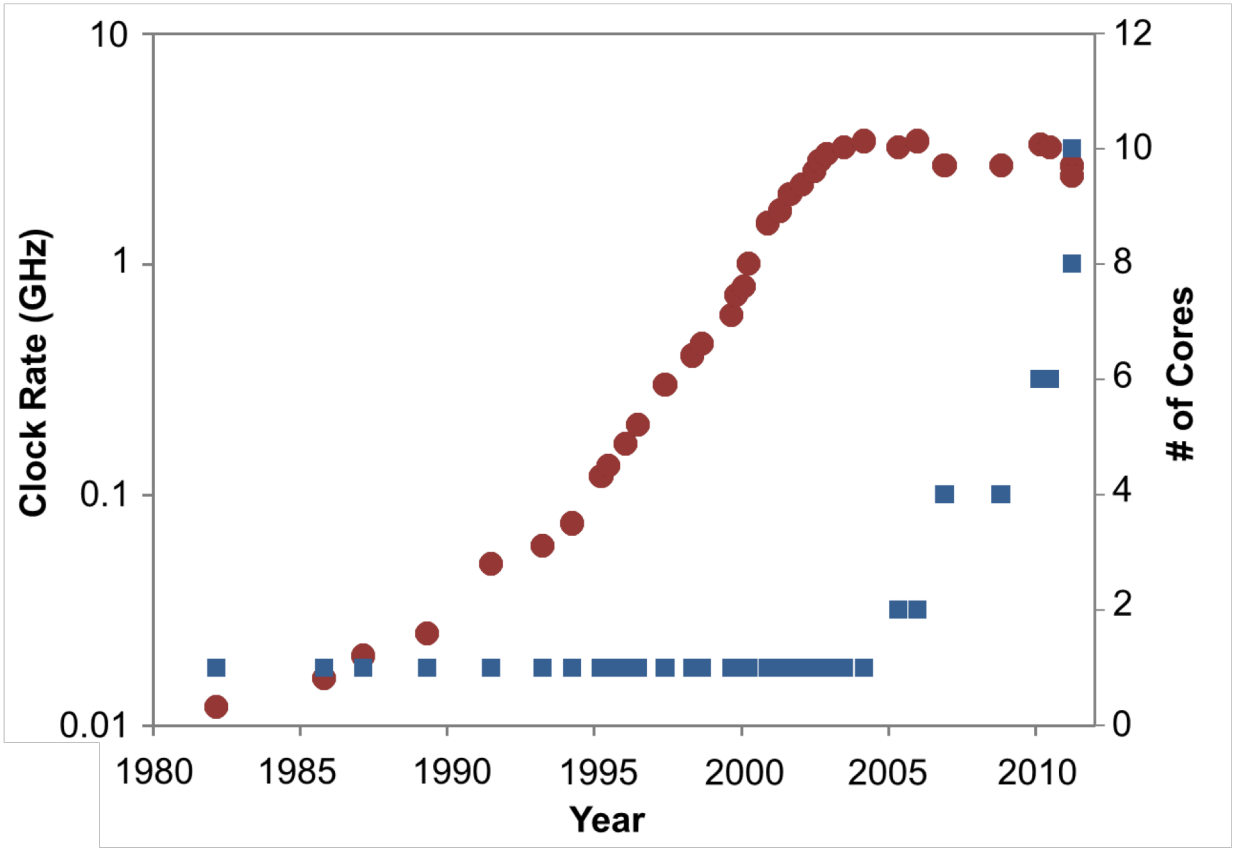
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM

SIAM CSE February 25 – March 1, 2019

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration | CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

Landscape of Computing Hardware

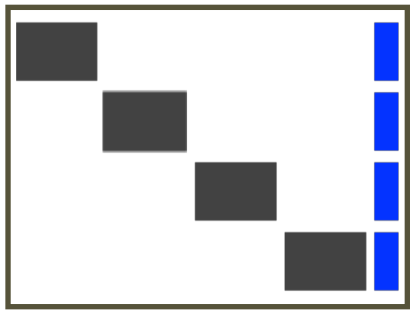


- Improvements in computing clock rates have slowed
- Increased focus on parallel computing architectures

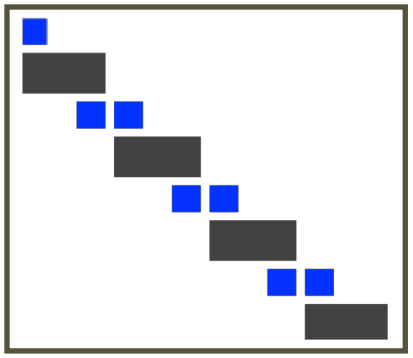
[Steven Edwards, Columbia University]

Exploiting Problem Structure

- Optimization Under Uncertainty
  - two-stage stochastic programming formulation
  - block structure because of coupled scenarios
  - common structure of many applications (parameter estimation, spatial decomposition)



- Dynamic Optimization
  - Simultaneous approach (discretization using OCFE)
  - block structure because of finite element discretization
  - pass-on variables couple neighboring blocks



[Laird, 2018]

Overview of Decomposition Algorithms

- Decomposition approaches allow for parallel/distributed computing
  - External decomposition
    - Progressive Hedging (PH)
    - Alternating direction method of multipliers (ADMM)
    - Benders decomposition, dual decomposition
  - Internal decomposition
    - Schur-complement decomposition
    - Block cyclic reduction
    - Reduced-space decomposition

External Decomposition	Internal Decomposition
Break full NLP into subproblems and coordinate solutions	*Build full NLP and decompose at linear algebra level of host algorithm*
Highly flexible and easier to implement	Harder to implement
Typically linear convergence	Convergence rates of host algorithm
Convergence not well understood for general nonconvex NLPs	Convergence properties host algorithm

[Rodriguez, AIChE 2018]

The Unspoken Implementation Challenge

Sandia National Laboratories

- Despite having well-established decomposition methods for exploiting common problem structures, there are very few general implementations of these approaches interfaced with popular algebraic modeling languages

- Why?
  - Few algebraic modeling languages are capable of capturing the high-level model structure that can be exploited by these algorithms

Software platform

Sandia National Laboratories

- Pyomo: Python Optimization Modeling Objects
- Formulate optimization models within Python

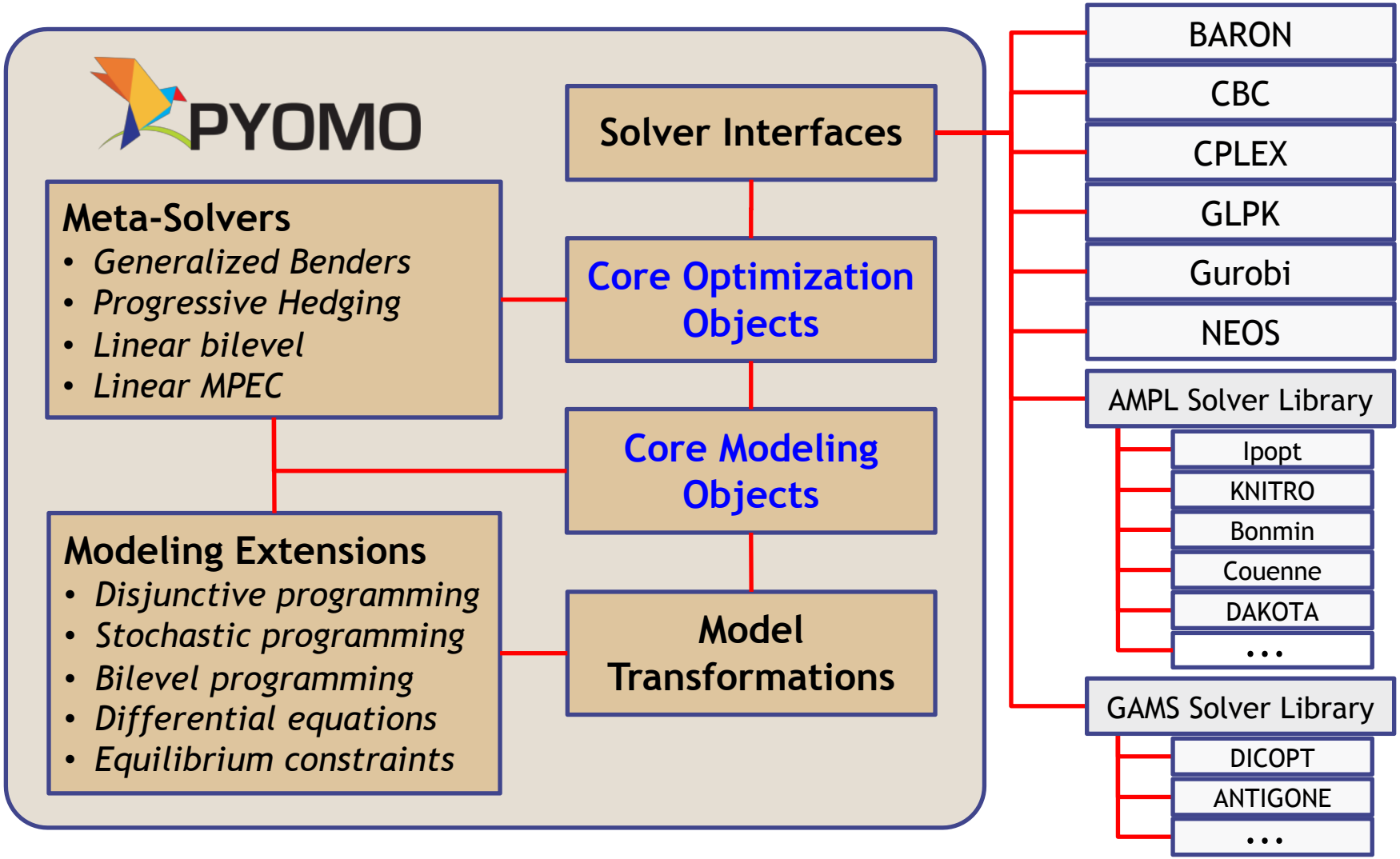

```
from pyomo.environ import *

m = ConcreteModel()

m.x1 = Var()
m.x2 = Var(bounds=(-1,1))
m.x3 = Var(bounds=(1,2))

m.obj = Objective(sense = minimize,
    expr = m.x1**2 + (m.x2*m.x3)**4 + m.x1*m.x3
     + m.x2 + m.x2*sin(m.x1+m.x3) )
```

- Utilize high-level programming language to write scripts and manipulate model objects
- Leverage third-party Python libraries

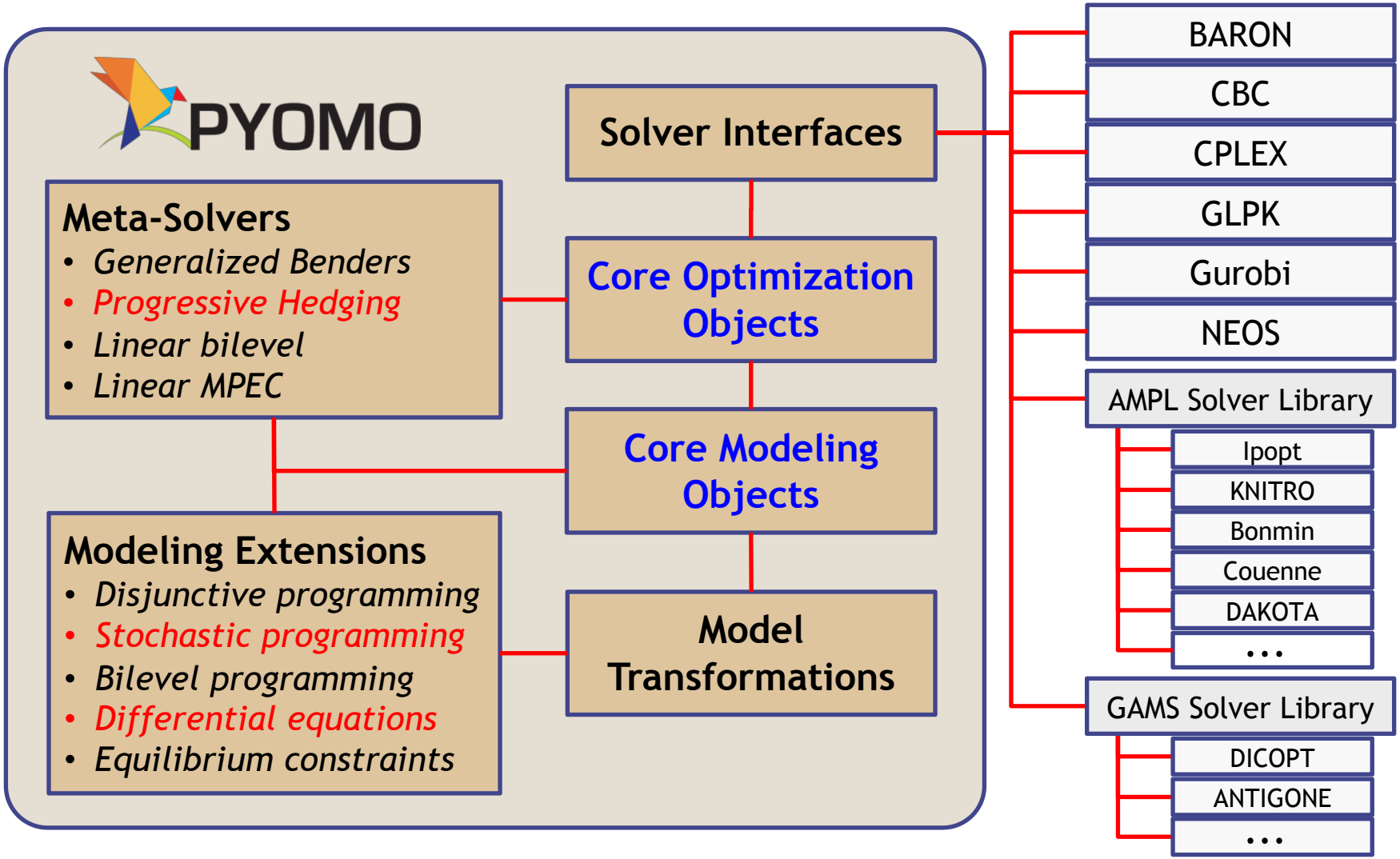
    e.g. SciPy, NumPy, MatPlotLib, Pandas

CCR Center for Computing Research

Pyomo at a Glance



Pyomo at a Glance



- Extend Pyomo syntax to represent:
  - Continuous domains
  - Ordinary or partial differential equations
  - Systems of differential algebraic equations (DAEs)
- Available discretization schemes:
  - Finite difference methods (Backward/Forward/Central)
  - Collocation (Lagrange polynomials with Radau or Legendre roots)
- Extensible framework
  - Write general implementations of custom discretization schemes
  - Build frameworks/meta-algorithms including dynamic optimization
- Interface with numerical simulators
  - Scipy for simulating ODEs
  - CasADi for simulating ODEs and DAEs

Simple Example

Sandia National Laboratories

$$\frac{dz}{dt} = z^2 - 2z + 1$$

$$z(0) = -3$$

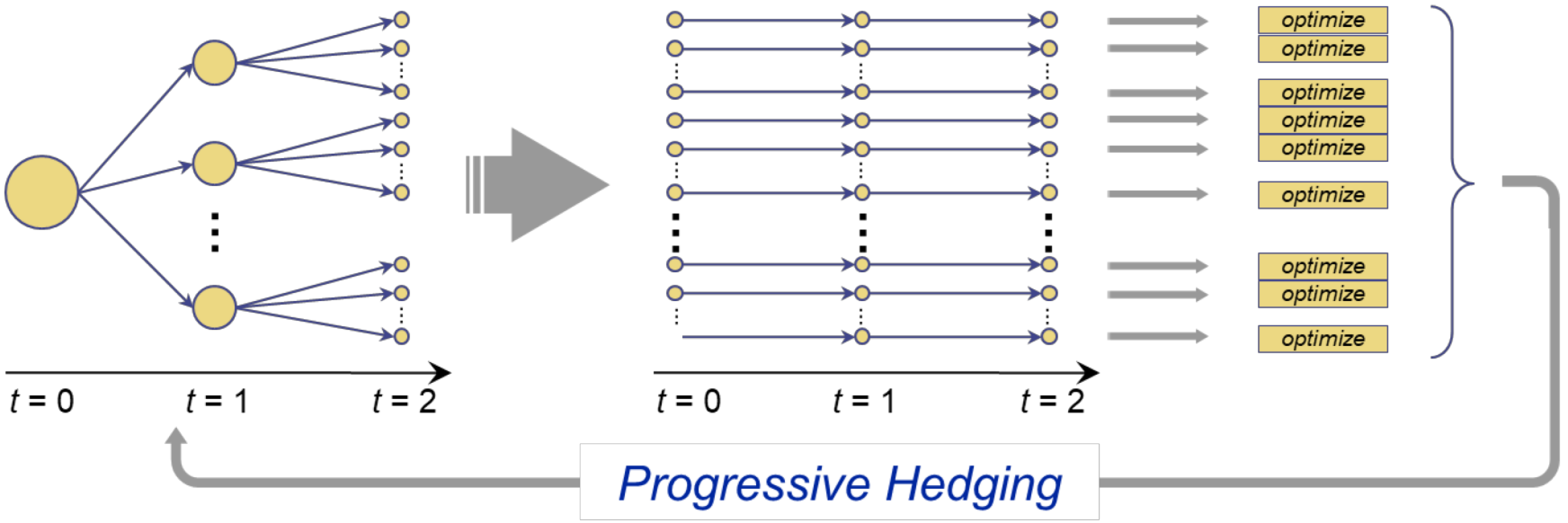
```
from pyomo.environ import *
from pyomo.dae import *

model = m = ConcreteModel()
m.t   = ContinuousSet(bounds=(0, 1))

m.z    = Var(m.t)
m.dzdt = DerivativeVar(m.z, wrt=m.t)

def _zdot(m, t):
    return m.dzdt[t] == m.z[t]**2 - 2*m.z[t] + 1
m.zdot = Constraint(m.t, rule=_zdot)

def _init_con(m):
    return m.z[0] == -3
m.init_con = Constraint(rule=_init_con)

# Discretize model using backward finite difference
discretizer = TransformationFactory('dae.finite_difference')
discretizer.apply_to(m,nfe=10,scheme='BACKWARD')
```

CCR Center for Computing Research

- Framework for representing stochastic programming models, only requiring:
  - deterministic base model
  - scenario tree defining the problem stages and uncertain parameters
- PySP provides two primary solution strategies
  - build and solve the deterministic equivalent (extensive form)
  - Progressive Hedging
  - (plus beta implementations of others, including 2-stage Benders and an interface to DDSIP)
- Parallel infrastructure for generating and solving subproblems on parallel (distributed) computing platforms

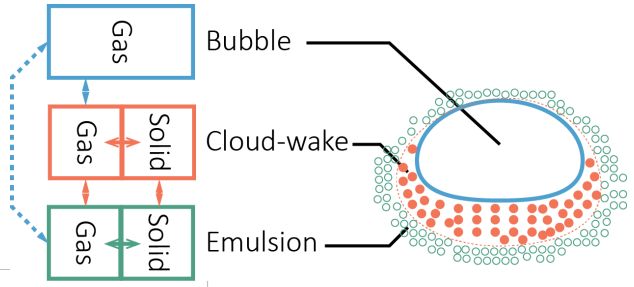
Progressive Hedging

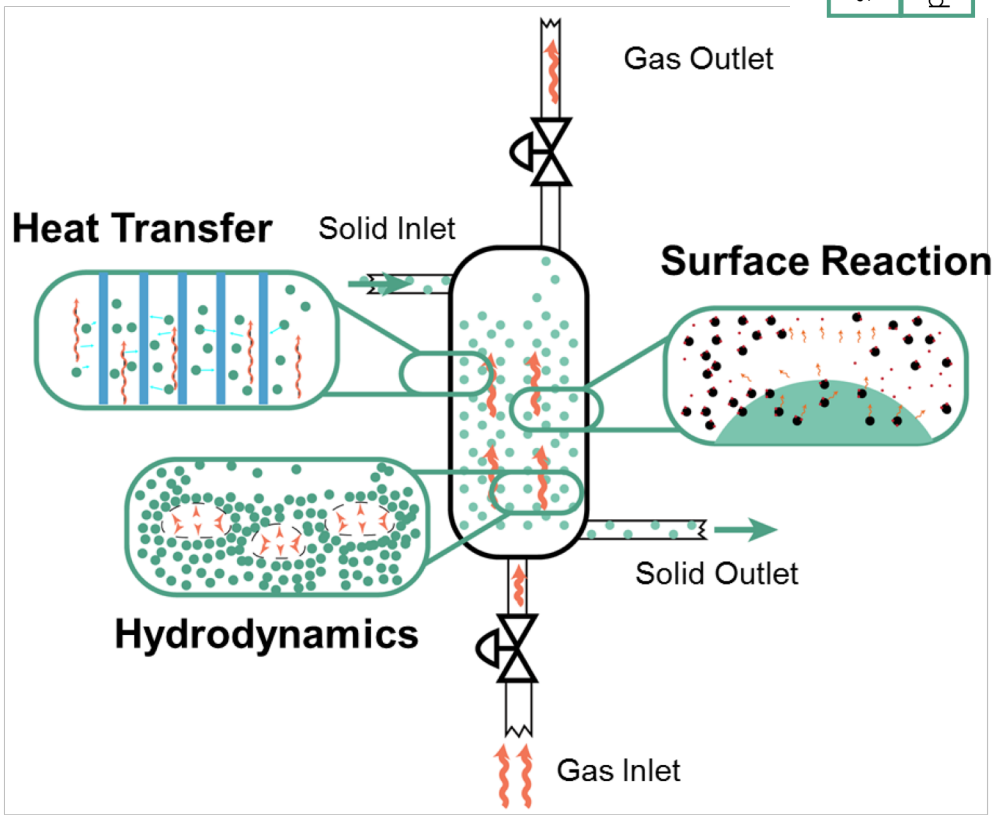
- Scenario-based decomposition algorithm
- Iteratively converge coupling constraints (non-anticipativity constraints) by penalizing deviation from consensus



Bubbling Fluidized Bed (BFB) Model

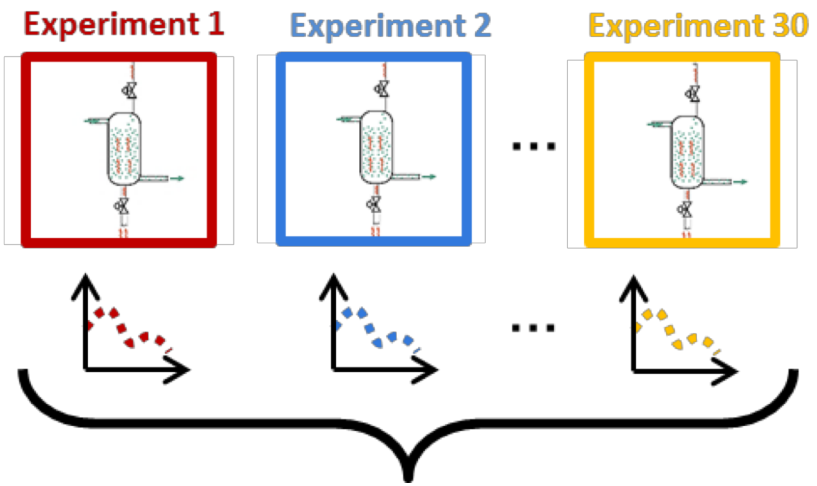
- Gas-solid, 3 region model [3]
- Steady state model with 1-D spatial variation





[Lee and Miller, Ind. Eng. Chem. Res., 2013]

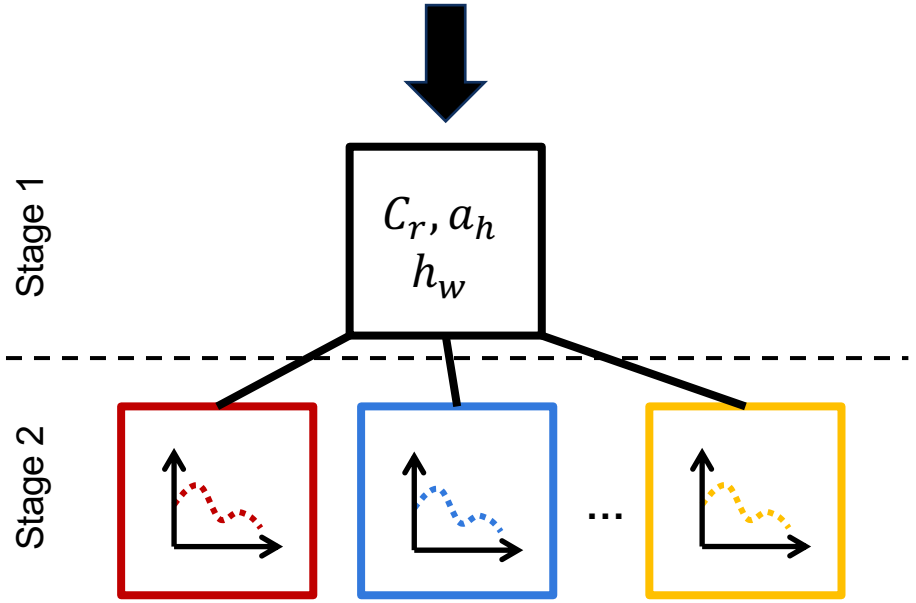
BFB Parameter Estimation



Heat Exchanger Model Parameters	
C_r	Average correction factor for tube model
a_h	Empirical factor for tube model
h_w	Heat transfer coefficient of tube walls

$$\min_{\{C_r, a_h, h_w\}} \sum_{exp.} (error_{meas})^2$$

$$s.t. \quad BFB\ model\ equations$$



Stochastic structure implementation in PySP

Sandia National Laboratories

```
def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model \
        import CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()
    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('cr')
    st_model.StageVariables[first_stage].add('ah')
    st_model.StageVariables[first_stage].add('hw')
    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'

    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):
    experiment = int(scenario_name.replace('Scenario',''))
    explist = [1,2,3] # Different data sets

    experiment = explist[experiment-1]
    instance = generate_model_paramest(experiment)

    return instance
```

CCR Center for Computing Research

BFB Parameter Estimation

- Create and solve extensive form

```
runef --solve --solver ipopt --output-solver-log -m bfb_param.py
```

- Solve using progressive hedging

```
runph --solver ipopt --output-solver-log -m bfb_param.py --default-rho=.25
```

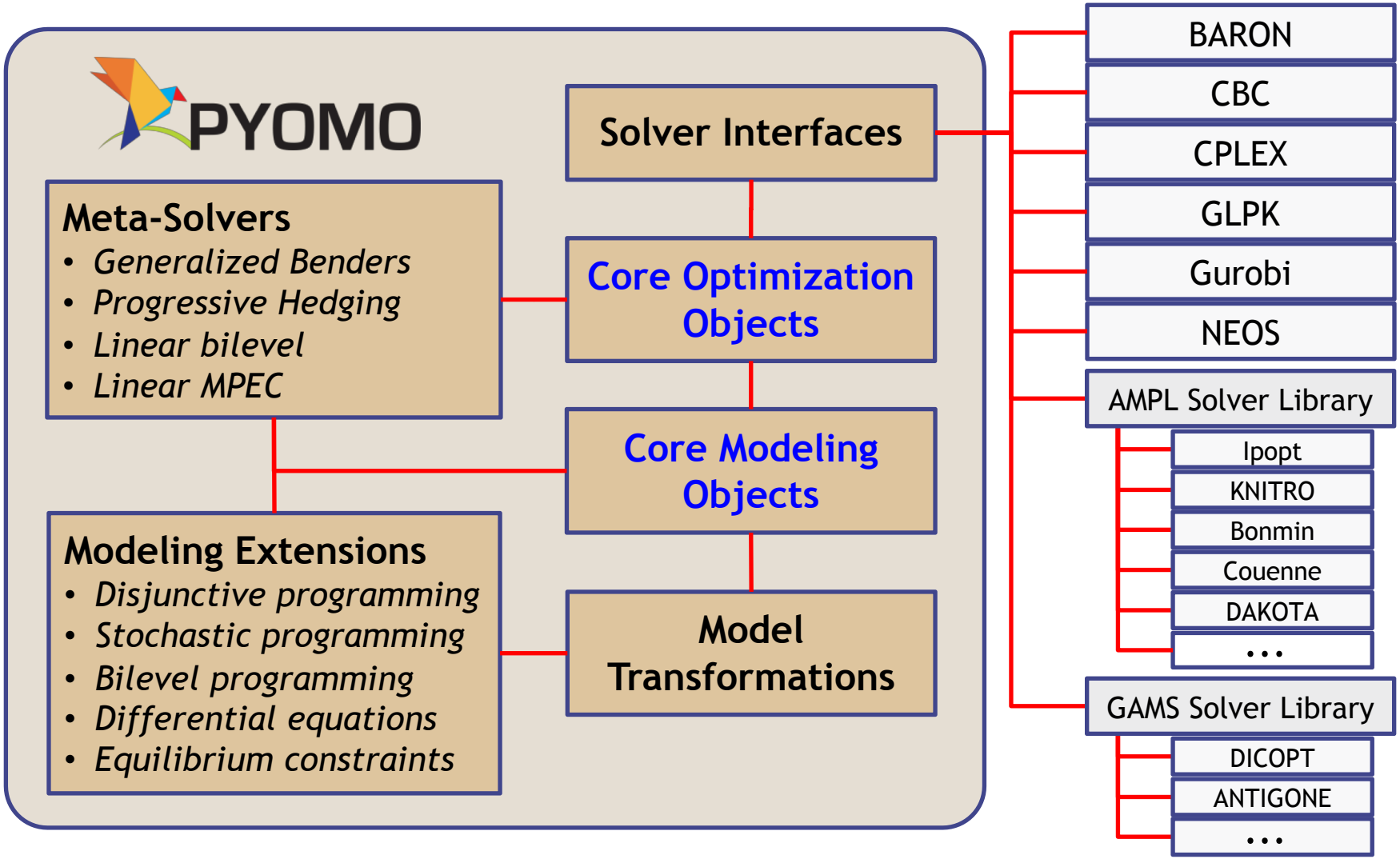
- Solve using progressive hedging in parallel

```
mpirun -np 1 pyomo_ns : -np 1 dispatch_srvr : -np 30 phsolverserver : \
    -np 1 runph --solver-manager=phpyro --shutdown-pyro \
    -m bfb_param.py --solver=ipopt --default-rho=0.25
```

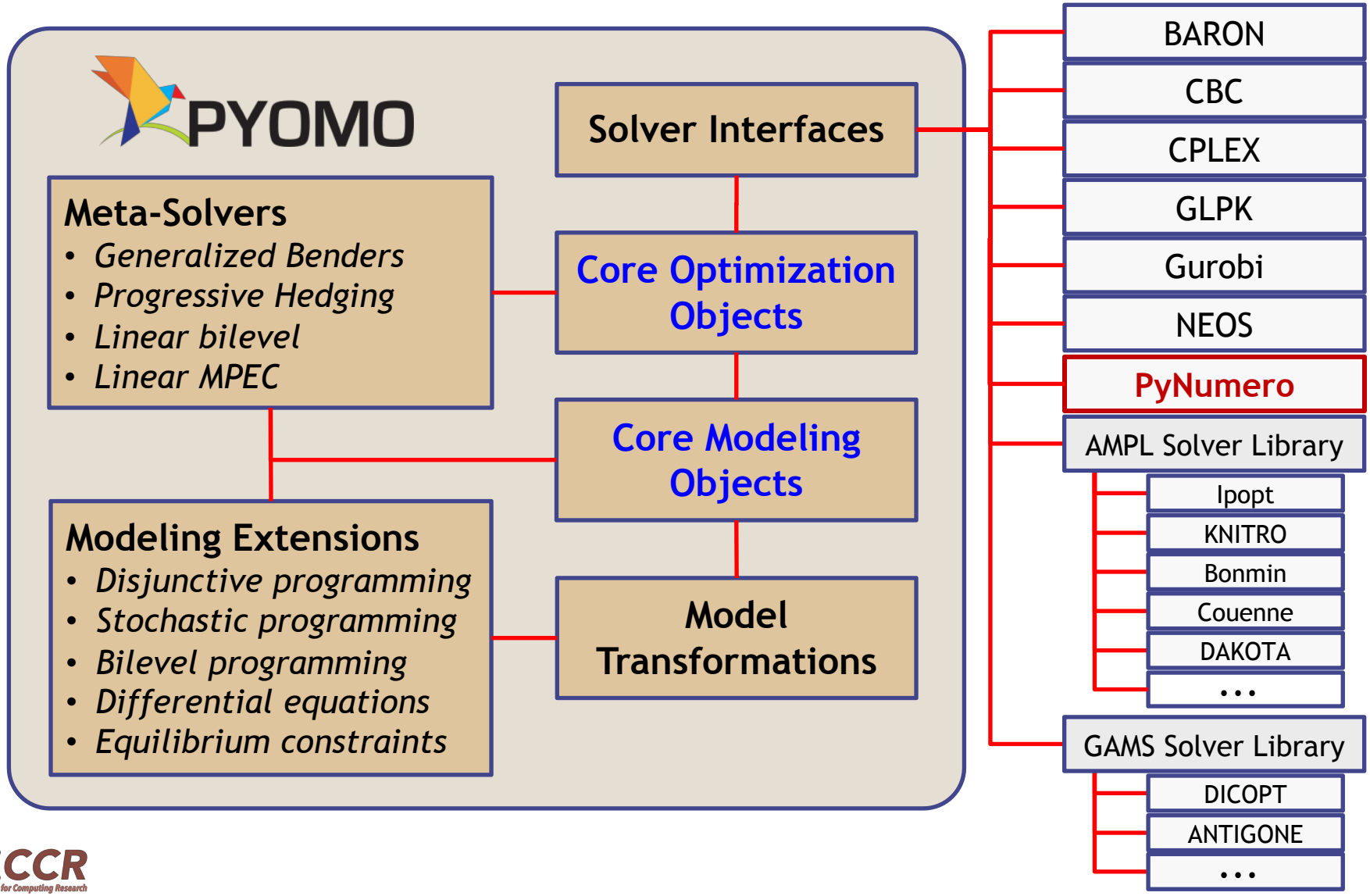
	C_r	a_h	h_w	Solve Time (s)
Actual	1.0	0.8	1500.0	-
Extensive Form	1.016	0.51	1450.35	604.45
Progressive Hedging (15 processors)	0.9824	0.7850	1501.74	610.98
Progressive Hedging (30 processors)	0.9824	0.7850	1501.74	459.10

Extensive form problem size ~400,000 variables and constraints
PH subproblem size ~13,000 variables and constraints

Pyomo at a Glance



Pyomo at a Glance



Purpose of PyNumero

- High-level Python framework for rapid development of nonlinear and parallel decomposition algorithms without large sacrifices in computational performance

- Dramatically reduce time required to prototype new algorithms while minimizing the performance penalty

- Develop a framework for the low-level numerical treatment of Pyomo models that can be used to:
  - Calculate efficient numerical derivatives
  - Implement algorithms that are natively aware of Pyomo model structure

[Rodriguez, AIChE 2018]

PyNumero

- Python C/C++ extension for nonlinear programming
  - Provides first and second derivatives via ASL
  - Interfaces with Numpy/Scipy for all array-operations
  - Supports python calls to HSL linear solver (MA27)
  - Computationally expensive operations performed in C/C++
  - Distributed with Pyomo and conda-forge

```
from pyomo.contrib.pynumero.interfaces import PyomoNLP
import pyomo.environ as aml

m = aml.ConcreteModel()
m.x = aml.Var([1, 2, 3], bounds=(0.0, None))
m.phys = aml.Constraint(expr=m.x[3]**2 + m.x[1] == 25)
m.rsrc = aml.Constraint(expr=m.x[2]**2 + m.x[1] <= 18.0)
m.obj = aml.Objective(expr=m.x[1]**4-m.x[3]*m.x[2]**3)

def my_algorithm(model):
    nlp = PyomoNLP(model)
    x = nlp.create_vector_x()
    c = nlp.evaluate_c(x)
    Jc = nlp.jacobian_c(x)
    …
```

PyNumero Performance

- Equality Constrained Problem with 100K variables

$$\text{minimize} \quad \int_{t_0}^{t_f} \alpha(y_{A,1} - y_{\text{ref}})^2 + \beta(u - u_{\text{ref}})^2 dt$$

$$\text{subject to} \quad \frac{dx_{A,0}}{dt} = \frac{1}{A_{\text{cond}}} V(y_{A,1} - x_{A,0})$$

$$\frac{dx_{A,i}}{dt} = \frac{1}{A_{\text{tray}}} [L_1(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})] \qquad \forall \ i \in \{1, ..., FT-1\}$$

$$\frac{dx_{A,FT}}{dt} = \frac{1}{A_{\text{tray}}} [F x_{A,\text{feed}} + L_1 x_{A,FT-1} - L_2 x_{A,FT} - V(y_{A,FT} - y_{A,FT+1})]$$

$$\frac{dx_{A,i}}{dt} = \frac{1}{A_{\text{tray}}} [L_2(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})] \qquad \forall \ i \in \{FT+1, ..., NT\}$$

$$\frac{dx_{A,NT+1}}{dt} = \frac{1}{A_{\text{reboiler}}} [L_2 x_{A,NT} - (F-D) x_{A,NT+1} - V y_{A,NT+1}]$$

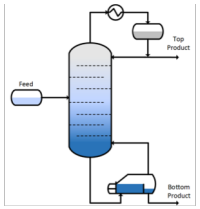
$$x_{A,i} = x_{0_{A,i}} \qquad \forall \ i \in \{0, ..., NT+1\}$$

$$V = L_1 + D$$

$$L2 = L_1 + F$$

$$u = \frac{L_1}{D}$$

$$\alpha_{A,B} = \frac{y_A(1 - x_A)}{x_A(1 - y_A)}$$



Basic SQP ~10% slower than IPOPT

Alternating Direction Method of Multipliers

Algorithm 2: Alternating Direction Method of Multipliers

Given barrier parameter $\rho > 0$, tolerances $\epsilon_r > 0, \epsilon_s > 0$, and estimates y^0, z^0

for $k = 0, 1, 2, \ldots$ **do**

  update partition variables:

  foreach $i \in \mathcal{P}$ **do**

$$x_i^{k+1} = \underset{x_i \in \mathcal{X}_i}{\arg\min}\, f_i(x_i) + \left(A_i x_i + B_i z^k\right)^T y_i^k + \frac{\rho}{2}\|A_i x_i + B_i z^k\|^2$$

  update coupling variables:

$$z^{k+1} = \underset{z}{\arg\min}\, \mathcal{L}_\rho(x^{k+1}, z, y^k)$$

  compute primal residual:

$$r^{k+1} = Ax^{k+1} + Bz^{k+1}$$

  compute dual residual:

$$s^{k+1} = \rho A^T B \cdot (z^{k+1} - z^k)$$

  update dual variables:

$$y^{k+1} = y^k + \rho \cdot r^{k+1}$$

  if $\|r^{k+1}\| \leq \epsilon_r$ *and* $\|s^{k+1}\| \leq \epsilon_s$ **then**

    stop

```python
from pyomo.contrib.pynumero.interfaces.nlp_transformations
import AdmmNLP
# …
for k in range(max_iter):

    # Step 3. Update partition variables
    for bid, nlp in enumerate(nlps):
        xs[bid] = basic_sqp(nlp, tee=False)

    # Step 6. Compute coupling variables
    z = [None] * len(nlps)
    for bid, nlp in enumerate(nlps):
        zi[bid] = xs[bid][nlp.zid_to_xid]
    z = np.mean(z, axis=0)

    # Step 8/10. Compute residuals
    r = [None] * len(nlps)
    for bid, nlp in enumerate(nlps):
        ri[bid] = xs[bid][nlp.zid_to_xid] - z
    s = z - old_z_estimates

    # Update estimates
    for bid, nlp in enumerate(nlps):
        nlp.z_estimates = z
        nlp.w_estimates = nlp.w_estimates + nlp.rho * r[bid]
        nlp.init_x = xs[bid]
        nlp.init_y = ys[bid]
    old_z_estimates = z

    # Step 14. Compute infeasibility norms
    r_norm = np.linalg.norm(np.concatenate(r))
    s_norm = np.linalg.norm(s)

    if r_norm < rtol and s_norm < stol:
        break
```

[J. S. Rodriguez, B. Nicholson, C. D. Laird, V. M. Zavala, "Benchmarking ADMM in nonconvex NLPs", Computers & Chemical Engineering, 2018.]

Summary

- Explicitly capturing high-level structure leads to significantly easier, faster, and more flexible implementations
- Pyomo provides high-level modeling constructs for capturing exploitable structure (www.pyomo.org)
- PyNumero is a promising tool for prototyping general implementations of decomposition algorithms

On-going work:

- Implementations of several internal decomposition methods using PyNumero (Schur-complement, cyclic reduction, etc.)
- Interface to the Rapid Optimization Library (ROL) to access several parallel-in-time algorithms under development