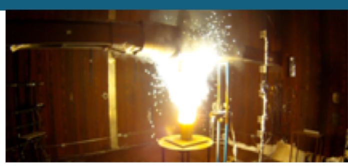
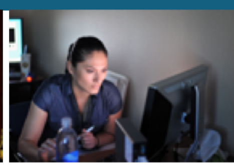
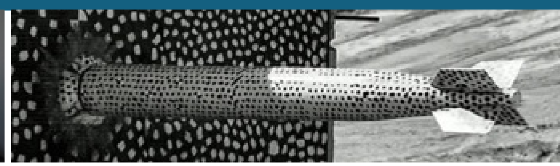
SAND2021-9229PE

Sandia National Laboratories

Rethinking the C++ / Python Boundary in Modeling and Optimization Tools







William Hart, Carl D. Laird
Sandia National Laboratories
wehart@sandia.gov, cdlaird@sandia.gov

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-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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-NA0003525.

Python Interfaces to IPOPT

There have been a number of interfaces to IPOPT developed over the years

- Pyipopt: Python interface to IPOPT
  - Developed on GitHub: https://github.com/xuy/pyipopt
  - Development is active (5 mo.)

- cyipopt: Cython-ized interface to IPOPT
  - Available through conda-forge (and pip), provides docker image
  - Development is active (days)
  - Provides mechanism for including HSL libraries

- … there are others

These tools are essentially a translation of IPOPT's C interface into Python

Python Interface to IPOPT

Problem is presented to the solver through implementation of methods to return problem statistics, objective, constraint residual, and derivatives

This has potential performance challenges (need to take care to use mechanisms for efficient evaluation).

```
cyipopt: Python wrapper for the Ipopt optimization package, written in Cython.

Copyright (C) 2012-2015 Amit Aides
Copyright (C) 2015-2018 Matthias Kümmerer

Author: Matthias Kümmerer <matthias.kuemmerer@bethgelab.org>
(original Author: Amit Aides <amitibo@tx.technion.ac.il>)
URL: https://github.com/matthias-k/cyipopt
License: EPL 1.0
"""

# Test the "ipopt" Python interface on the Hock & Schittkowski test problem
# #71. See: Willi Hock and Klaus Schittkowski. (1981) Test Examples for
# Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical
# Systems Vol. 187, Springer-Verlag.
#
# Based on matlab code by Peter Carbonetto.

from __future__ import print_function, unicode_literals

import numpy as np
import ipopt
```

```
class hs071(object):
    def __init__(self):
        pass

    def objective(self, x):
        #
        # The callback for calculating the objective
        #
        return x[0] * x[3] * np.sum(x[0:3]) + x[2]

    def gradient(self, x):
        #
        # The callback for calculating the gradient
        #
        return np.array([
                    x[0] * x[3] + x[3] * np.sum(x[0:3]),
                    x[0] * x[3],
                    x[0] * x[3] + 1.0,
```

Workflow / Software Stack with Pyomo

Decomposition Algorithm
(Progressive Hedging)

Algebraic Modeling Language
Pyomo

Optimization Problem

Core Expressions

Evaluation, Derivatives (AD), Linear Algebra
ASL, HSL

Nonlinear Solver

C is Hard

Python Code

Boundary

Compiled
C / Fortran code

C is FAST

Workflow / Software Stack with Pyomo

Decomposition Algorithm
(Progressive Hedging)

Algebraic Modeling Language
Pyomo

Optimization Problem

Core Expressions

Evaluation, Derivatives (AD), Linear Algebra
ASL, HSL

Nonlinear Solver

Have we struck the correct balance?

Workflow / Software Stack with Pyomo

Decomposition Algorithm
(Progressive Hedging)

Algebraic Modeling Language
Pyomo

Optimization Problem

Core Expressions

Evaluation, Derivatives (AD), Linear Algebra
ASL, HSL

Nonlinear Solver

POEK:
Python Optimization Expression Kernel

Have we struck the correct balance?

PyNumero:
Python Framework for NLP Algorithms

PyNumero: Python Numerical Optimization

Jose Santiago Rodriguez

Bethany Nicholson, John D. Siirola, Carl Laird

Davidson School of Chemical Engineering
Purdue University
Center for Computing Research
Sandia National Laboratories

What is PyNumero?

PyNumero: A high-level **python** framework focused on compatibility with **NumPy/SciPy** and **Pyomo** for rapid development of nonlinear algorithms without large sacrifices on **computational performance**.

PYOMO

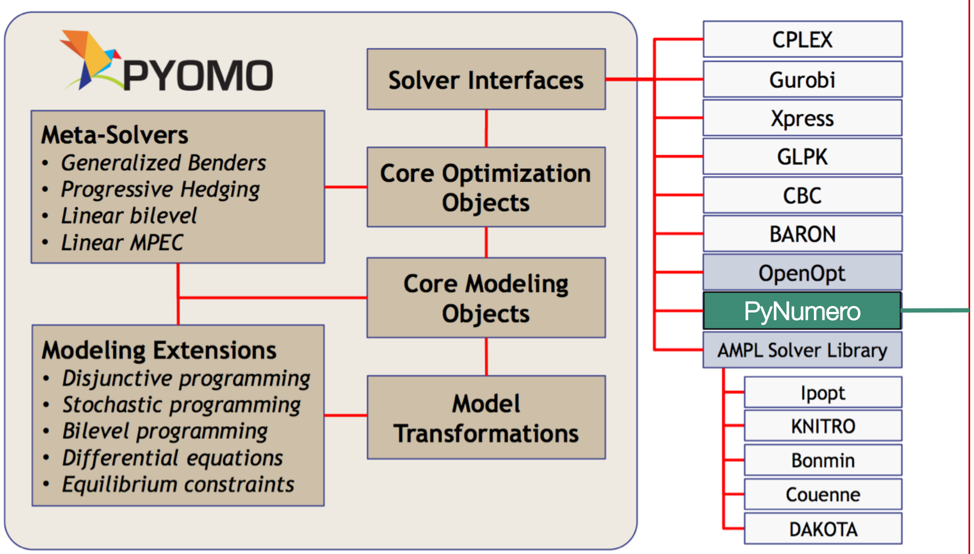
NumPy

Very similar goals to NLPy (provide high-level interface for building NLP algorithms)
* Focused on compatibility with NumPy/SciPy and Pyomo

Dramatically reduce time required to prototype new NLP algorithms and parallel decomposition while minimizing the performance penalty

What is PyNumero?

PyNumero: A high-level **python** framework focused on compatibility with NumPy/SciPy and Pyomo for rapid development of nonlinear algorithms without large sacrifices on **computational performance**.



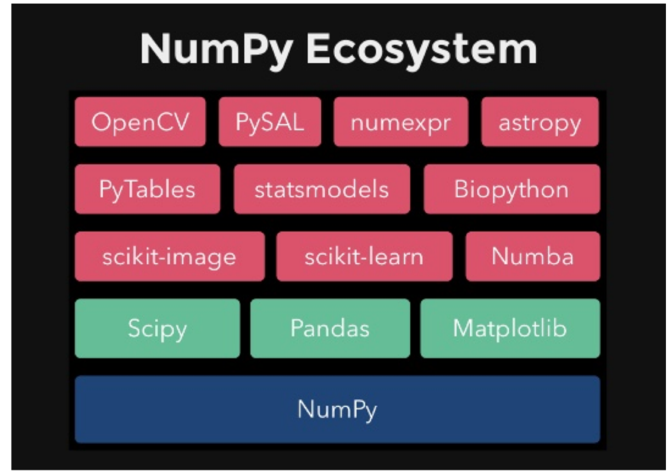

```python
from pyomo.contrib.pynumero.sparse import BlockSymMatrix, BlockMatrix
...
J = nlp.jacobian_c(x)
W = nlp.hessian_lag(x, y)

M = BlockSymMatrix(2)
M[0, 0] = W
M[1, 0] = J

Np = BlockMatrix(2, 1)
Np[0, 0] = nlp.hessian_lag(x, y, variables_cols=[m.p1, m.p2])
Np[1, 0] = nlp.jacobian_c(x, variables=[m.p1, m.p2])

M_array = M.toarray()
Np_array = Np.toarray()

ds = np.linalg.solve(M_array, Np_array)
```

Numpy: Numerical Python

- Fundamental for **scientific computing** in Python
  - Powerful N-dimensional array objects
  - Sophisticated broadcasting functions
  - Effective tool for integrating C/C++ and fortran
  - Mainly written in C
  - Core functionality for parallel packages mpi4py

- Suited for many applications
  - **Linear algebra**
  - Image processing
  - Signal Processing
  - and many others …

  - … **Nonlinear Programming**




```
import numpy as np
from scipy.sparse import coo_matrix
from scipy.sparse.linalg import spsolve

row = np.array([0, 3, 1, 2, 0, 2])
col = np.array([0, 3, 1, 2, 2, 0])
data = np.array([4, 5, 7, 9, 3, 3])

A = coo_matrix((data, (row, col)), shape=(4, 4))
b = np.array([1, 2, 3, 4])
x = spsolve(A, b)
```

PyNumero

- Python C/C++ extension for nonlinear programming
  - Access to all high-level features of python
  - Provides first and second derivatives via ASL (Pyomo aware)
  - 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)

nlp = PyomoNLP(m)
x = nlp.create_vector_x()
c = nlp.evaluate_c(x)
csub = nlp.evaluate_c(x, [m.phys])
Jc = nlp.jacobian_c(x)
```

ASL

Pynumero Software / Algorithms

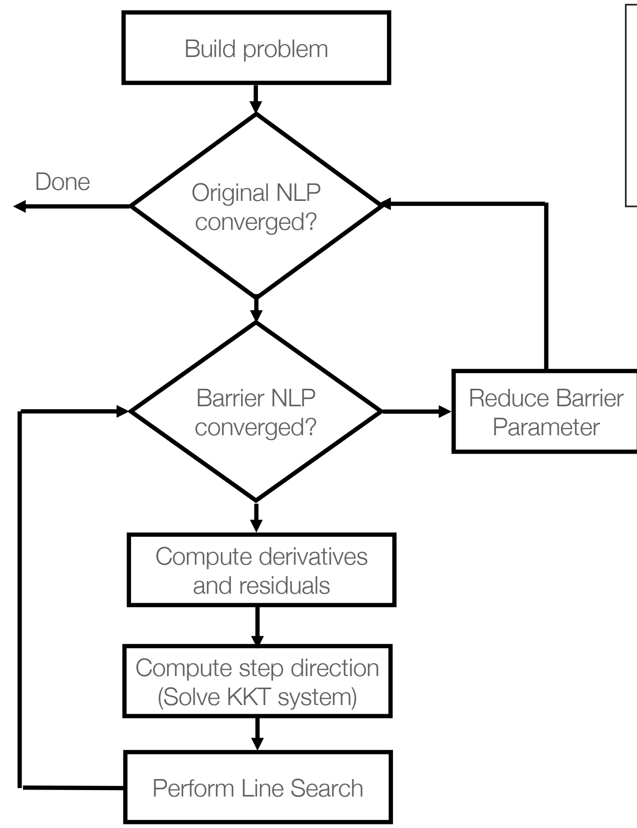
Compiled C | NLP Interfaces ← Your Algorithm → Linear Algebra Interfaces | Compiled C/Fortran

NLP Interfaces
- **PyomoNLP**
- AslNLP

Your Algorithm
- RGM
- **ALM**
- GPM
- **SQP**
- **IP**
- **NLP Sensitivity**

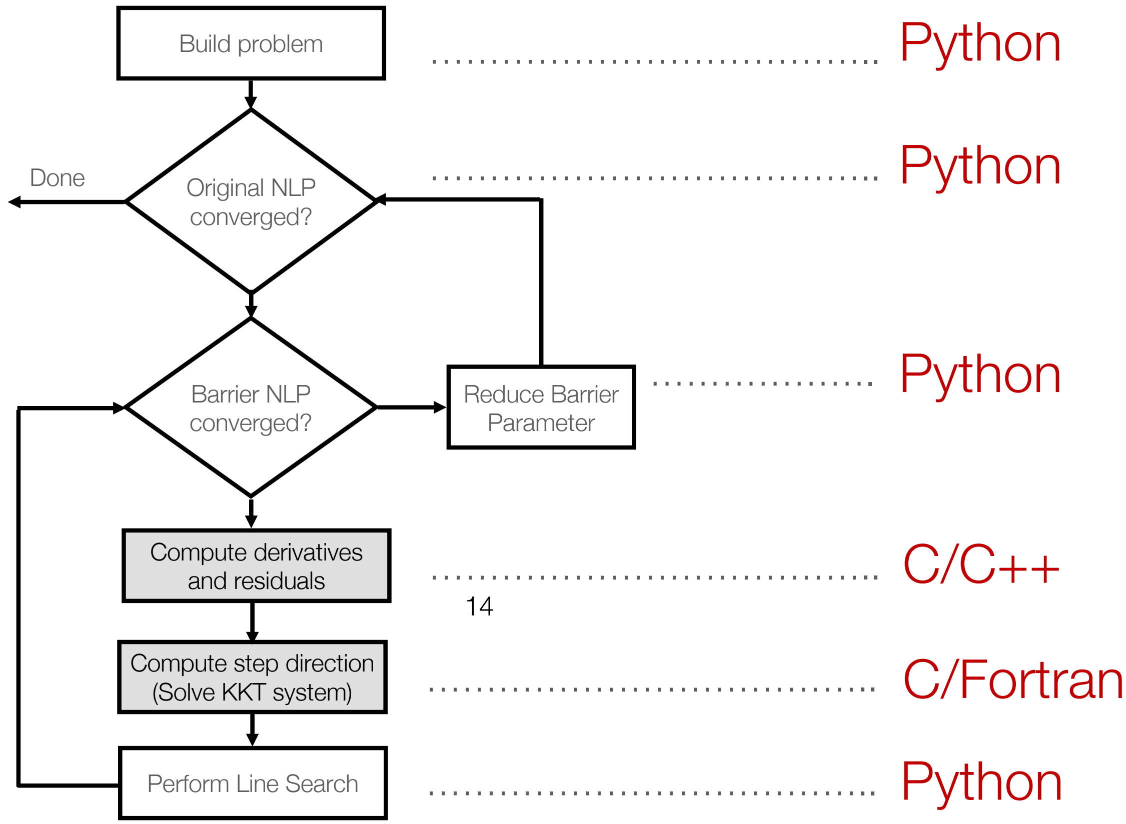
Linear Algebra Interfaces
- COOMatrix
- CSCMatrix
- CSRMatrix
- **BlockMatrix**
- MA27Solver
- **Scipy (Solvers)**
- **Numpy**

Interior-Point Algorithm



$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & d_L \leq d(x) \leq d_U \\ & x_L \leq x \leq x_U \end{aligned}$$

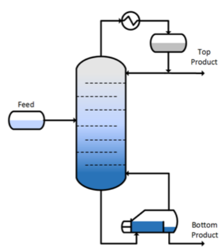
Structure in Interior-Point Algorithm



But it's Python – What about performance?

Equality Constrained Problem with 100K variables

$$
\begin{aligned}
\text{minimize} \quad & \int_{t_0}^{t_f} \alpha(y_{A,1} - y_{\text{ref}})^2 + \beta(u - u_{\text{ref}})^2 dt \\
\text{subject to} \quad & \frac{dx_{A,0}}{dt} = \frac{1}{A_{\text{cond}}} V(y_{A,1} - x_{A,0}) \\
& \frac{dx_{A,i}}{dt} = \frac{1}{A_{\text{tray}}} [L_1(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})] \qquad \forall \ i \in \{1, ..., FT-1\} \\
& \frac{dx_{A,FT}}{dt} = \frac{1}{A_{\text{tray}}} [F x_{A,\text{feed}} + L_1 x_{A,FT-1} - L_2 x_{A,FT} - V(y_{A,FT} - y_{A,FT+1})] \\
& \frac{dx_{A,i}}{dt} = \frac{1}{A_{\text{tray}}} [L_2(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})] \qquad \forall \ i \in \{FT+1, ..., NT\} \\
& \frac{dx_{A,NT+1}}{dt} = \frac{1}{A_{\text{reboiler}}} [L_2 x_{A,NT} - (F-D) x_{A,NT+1} - V y_{A,NT+1}] \qquad (2 \\
& x_{A,i} = x0_{A,i} \qquad \forall \ i \in \{0, ..., NT+1\} \\
& V = L_1 + D \\
& L2 = L_1 + F \\
& u = \frac{L_1}{D} \\
& \alpha_{A,B} = \frac{y_A(1 - x_A)}{x_A(1 - y_A)}
\end{aligned}
$$



Basic SQP
~10% slower
than IPOPT
(YMMV)

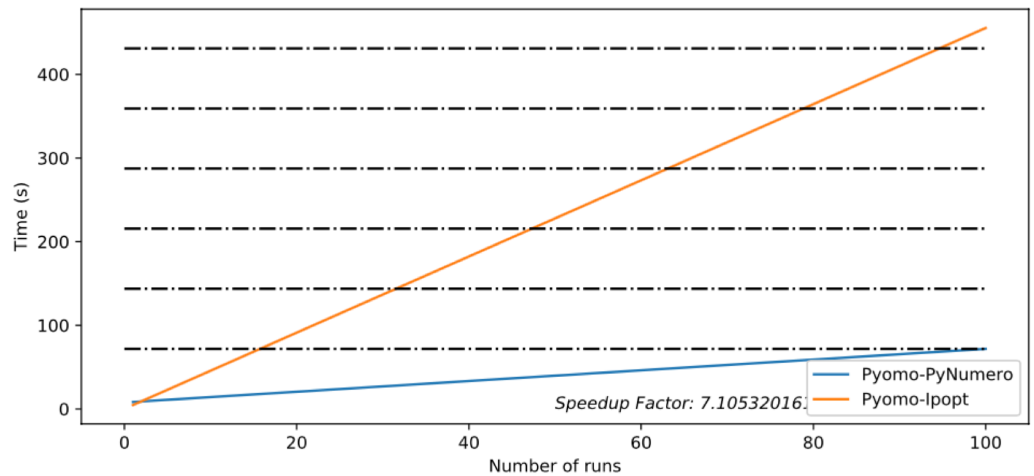
Motivation

- **Why bother when we have IPOPT?**
  - Standard optimization codes are powerful but complex
  - Can be difficult to modify or extend

- **Need simpler frameworks to support future research in nonlinear optimization**
  - Nonlinear decomposition algorithms
  - Embedding in MINLP solvers
  - Nonlinear Model Predictive Control NMPC
  - … examples

Iterative Analysis of Optimization Problems

$$
\begin{aligned}
\text{maximize} \quad & \int_{t_0}^{t_f} C_b \, dt \\
\text{subject to} \quad & \dot{C}_a = \frac{F_a}{V_r} - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \\
& \dot{C}_b = k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \\
& \dot{C}_c = k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b
\end{aligned}
$$



Testing Nonstandard Approaches

Algorithm 1: Method of Multipliers

Given penalty parameter $\rho > 0$, tolerance $\epsilon > 0$, and estimates y^0

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

  update primal variables:

  $$(x^{k+1}) = \arg\min_{x \in \mathcal{X}} \mathcal{L}_\rho(x, y^k)$$

  compute primal residual:

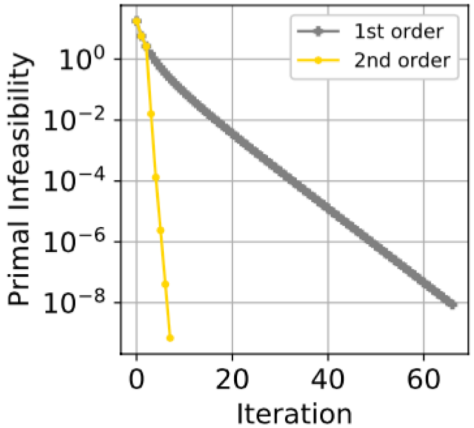
  $$r^{k+1} = c(x^{k+1})$$

  update dual variables:

  $$y^{k+1} = y^k + \cancel{\rho \cdot r^{k+1}} (J_c(x^{k+1}) \nabla^2_{xx} \mathcal{L}_\rho(x^{k+1}, y^{k+1})^{-1} J_c(x^k)^T)^{-1} c(x^{k+1})$$

  if $\|r^{k+1}\| \le \epsilon$ **then**

    stop



NLP Sensitivity

$$\begin{array}{ll} \min & f(x,p) \\ \text{s.t.} & c(x,p) = 0, \quad (y) \end{array}$$

$$s(p)^T = \left[x(p)^T y(p)^T\right]$$

$$\frac{ds(p_0)}{dp}^T = \begin{bmatrix} \nabla^2_{xx} L(s(p_0)) & J_c(s(p0))^T \\ J_c(s(p0)) & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla^2_{xp} L(s(p_0)) \\ \nabla_p c(s(p0)) \end{bmatrix}$$

$$\begin{array}{ll} \min & x_1^2 + x_2^2 + x_3^2 \\ \text{s.t.} & 6x_1 + 3x_2 + 2x_3 - p1 = 0 \\ & p_2 x_1 + x_2 - x_3 - 1 = 0 \end{array}$$

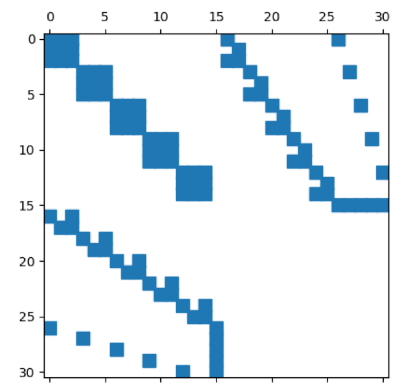
```python
from pyomo.contrib.pynumero.sparse import BlockSymMatrix, BlockMatrix
from pyomo.contrib.pynumero.interfaces.utils import compute_init_lam
from pyomo.contrib.pynumero.interfaces import PyomoNLP
import pyomo.environ as aml
import numpy as np

m = create_model(4.5, 1.0)
opt = aml.SolverFactory('ipopt').solve(m)

nlp = PyomoNLP(m)
x = nlp.x_init()
y = compute_init_lam(nlp, x=x)

J = nlp.jacobian_c(x)
W = nlp.hessian_lag(x, y)

M = BlockSymMatrix(2)
M[0, 0] = W
M[1, 0] = J

Np = BlockMatrix(2, 1)
Np[0, 0] = nlp.hessian_lag(x, y, variables_cols=[m.p1, m.p2])
Np[1, 0] = nlp.jacobian_c(x, variables=[m.p1, m.p2])

M_array = M.toarray()
Np_array = Np.toarray()

ds = np.linalg.solve(M_array, Np_array)
```

Structure and Decomposition

- **Optimization with inherent structure is ubiquitous in engineering applications**
  - Stochastic programming
  - Dynamic optimization
  - Network optimization problems
  - PDE optimization

$$\min_{x_i, z} \quad \sum_{i \in \mathcal{P}} f_i(x_i)$$
$$\text{s.t.} \quad c_i(x_i) \geq 0, \;\; i \in \mathcal{P}$$
$$A_i x_i + B_i z = 0, \;\; (y_i) \quad i \in \mathcal{P}$$

- Decomposition approaches allow for parallelization
  - Internal decomposition
    - Schur-complement decomposition
    - Cyclic reduction
  - External decomposition
    - Alternating direction method of multipliers (ADMM)
    - Progressive Hedging (PH)



- Structures in Python to support external and internal decomposition approaches and allow for parallelization
  - BlockMatrix, BlockVector
  - mpi4py

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**

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

  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

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

    # solve blocks independently
    for bid, nlp in enumerate(nlps):
        xs[bid], ys[bid] = basic_sqp(nlp, tee=False)

    # update 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)

    # 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

    # 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.]

Conclusions

- Efficient optimization solvers are typically complex codes that require significant software expertise to be extended
- PyNumero is a flexible framework for prototyping and developing NLP algorithms in Python
- PyNumero is designed to facilitate research of decomposition algorithms.
- PyNumero exploits the Numpy ecosystem and C++ python extensions to achieve good performance.
- PyNumero is distributed with Pyomo and conda-forge.

References

- Y. Cao, A. Seth, and C.D. Laird. A parallel augmented Lagrangian interior-point approach for large-scale NLP problems on graphics processing units. *Computers and Chemical Engineering*, 2015.
- Victor DeMiguel and Francisco J Nogales. On Decomposition Methods for a Class of Partially Separable Nonlinear Programs. *Mathematics of Operations Research*, 33(1):119– 139, 2008.
- Janick Frasch. Parallel Algorithms for Optimization of Dynamic Systems in Real-Time. PhD thesis, University of Leuven, 2014.
- Jacek Gondzio and Robert Sarkissian. Parallel Interior Point Solver for Structured Linear Programs. *Mathematical Programming, 96(3):561–584, 200*
- Jia Kang, Yankai Cao, Daniel P. Word, and C. D. Laird. An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. *Computers and Chemical Engineering*, 71:563–573, 2014
- Carl D. Laird, Angelica V.Wong, and Johan Akesson. Parallel solution of large-scale dynamic optimization problems. *Elsevier B.V*, 2011.
- Olaf Schenk and Klaus Gartner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475–487, 2004.
- Daniel P. Word, Jia Kang, Johan Akesson, and CarlD. Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–688, 2014.
- Victor M. Zavala, Carl D. Laird, and Lorenz T. Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chemical Engineering Science*, 63(19):4834–4845, 2008.
- Yu Zhu, Daniel Word, John Siirola, and Carl D. Laird. Exploiting modern computing architectures for efficient large-scale nonlinear programming. *Computer Aided Chemical Engineering*, 27(C):783–788, 2009.
- Masoud Asadzadeh, Bryan a. Tolson, and Robert McKillop. A Two Stage Optimization Approach for Calibrating Water Distribution Systems. *Water Distribution Systems Analysis 2010*, pages 1682–1694, 2011. doi: 10.1061/41203(425)148.

Acknowledgements

- Acknowledgements
  - This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy

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-NA0003525.

Workflow / Software Stack with Pyomo

Decomposition Algorithm
(Progressive Hedging)

Algebraic Modeling Language
Pyomo

Optimization Problem

Core Expressions

Evaluation, Derivatives (AD), Linear Algebra
ASL, HSL

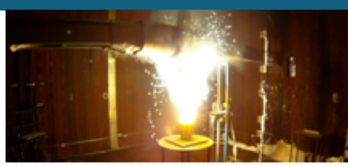
Nonlinear Solver

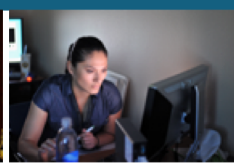
POEK:
Python Optimization Expression Kernel

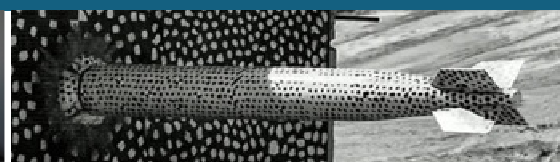
Have we struck the correct balance?

PyNumero:
Python Framework for NLP Algorithms

POEK: A Python Optimize Expression Kernel







William Hart

Sandia National Laboratories

wehart@sandia.gov

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-NA0003525.

Motivation – Improve Performance of Pyomo (Large/Nonlinear)

Pyomo team always looking for opportunities to improve performance

- Pyomo relies on file I/O for many solver interactions (ASL/nl)
- String manipulation and file I/O has major performance implications
- Comparisons with JuMP suggest that Pyomo can be orders of magnitude slower on some problems
- While speed isn't everything, it is hard to justify a performance gap like that to users

Recent development efforts have improved performance in certain cases

- New expression system provides full support for PyPy
  - Pyomo is often 3-4x faster on PyPy than Cpython
  - But PyPy is not commonly used, even by Pyomo developers
- Persistent solver interfaces allow for improved performance on repeated solves with similar models
  - More work needed to "automate" updates
  - Only implemented for Gurobi / CPLEX currently

Motivation – Improved / Rich Solver Interfaces

File I/O based interfaces have significant limitations

- Creating strings in Python is very slow
- Very difficult to support callbacks

Many solves with "similar" problems

- Examples: parameter changes, objective changes (e.g., PH), progressive refinement (e.g., cuts, MINLP)
- Pyomo supports persistent interfaces, but only for commercial solvers (CPLEX/Gurobi)
- Not supported for file I/O based interfaces
  - E.g., NL-based interfaces, re-solving a problem after mutating a constant is still expensive
- Users have written "custom" solvers to do some of this

Build a compact representation that is re-used in many models

- Idea: generate and serialize a compact sub-model representation
- Pyomo could support this with a custom writer
- A fast binary representation can really only be generated in a C/C++ layer

Advanced algorithms require rich interactions with the solvers

- Requires "direct" interfaces, but APIs are solver specific
- Benefits may require highly efficient callbacks (e.g., C++ side)
- Access to expressions in C++ allows for efficient model manipulation / interrogation (e.g., derivatives, FBBT)
  - But these are more difficult to implement (C++ is harder than Python)

Motivation – Opportunities for Improvements

Expression generation and translation in Python

- Pyomo expressions create **many** small objects, which is a performance bottleneck

Replace string-based file I/O with direct solver interfaces

Efficient modeling constructs and updates

- E.g., tailored expression objects for linear/quadratic expressions

Time to construct model and setup Ipopt

- P-median: N=M=640, P=1
- Cpython 3.6

	Pyomo
Build Model	22.7
Setup Ipopt	44.6
TOTAL	67.3

POEK: Python Optimization Expression Kernel

- Experiment to explore opportunities for performance improvement

Motivation – Pyomo expression kernels could be executed in C++

There are several obvious performance wins

- Create fewer Python objects for expressions
- Avoid creating Python expression objects with long lifetimes (which will help with memory)
- Avoid creating canonical expression representations
- Avoid file I/O (including expensive string manipulation for floating point numbers)

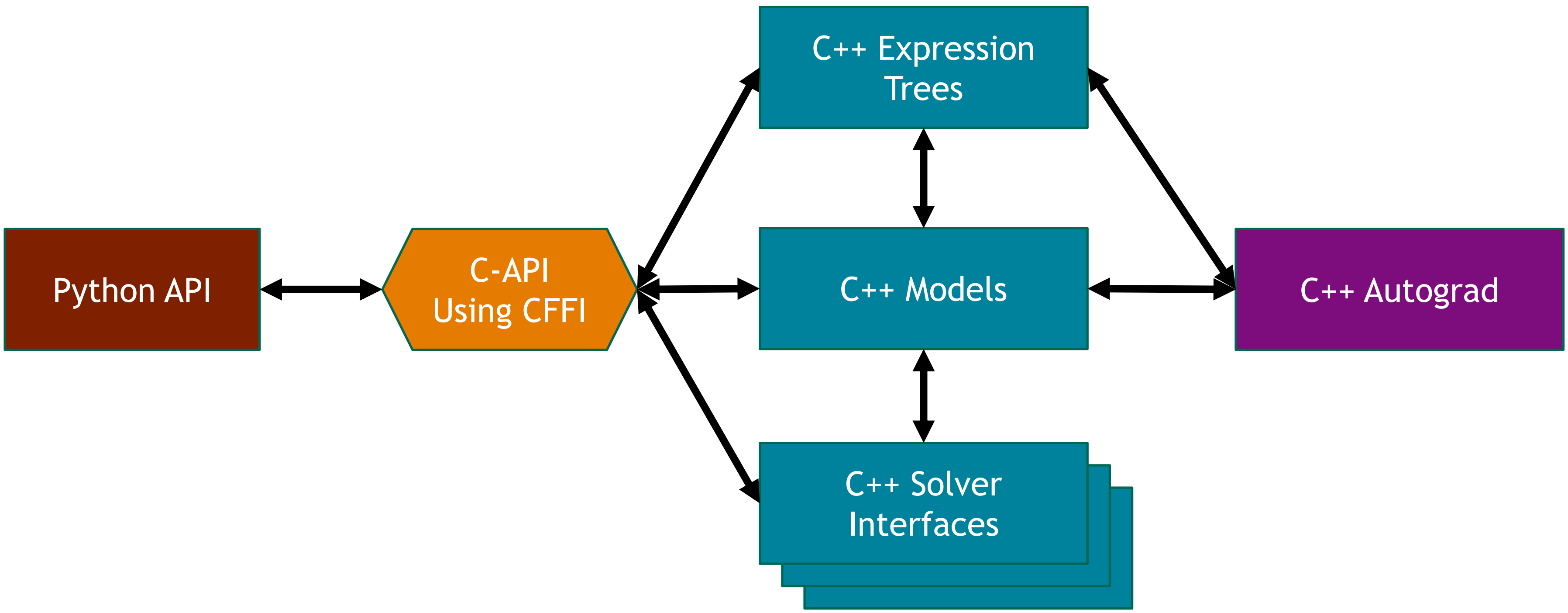
Many Performant Python frameworks move compute-intensive kernels into C

- Numpy/Pandas
- Tensorflow

POEK expressions could be generated with many Python-C calls

- Idea: overload operators on expressions objects: variables, constants, expressions
  - This is the same method used by Pyomo
- Each operator call results in a Python-C call to create a new expression

POEK Overview



Simple Example

```
from poek import *

x1 = variable('x1', lb=-1, ub=1, initialize=0.5)
x2 = variable('x2', initialize=1.5)

m = model()
m.add( -(x2-2)**2 )
m.add( x1**2 + x2 - 1 == 0 )

solver = Solver('ipopt’)
solver.solve(m)
```

NOTES:

- POEK expressions are merely pointers to C++ expression objects
- Same for variables, models and solvers
- Solution values are stored in the variables after optimization
- Autograd supports derivatives but not Hessians or Hessian-vector products

P-Median Model: POEK

```
x = {}
for n in range(N):
  for m in range(M):
    x[n,m] = variable(lb=0, ub=1, initialize=0)
y = variable(N, lb=0, ub=1, initialize=0)
d = {}
for n in range(N):
  for m in range(M):
    d[n,m] = random.uniform(0.0,1.0)

pmedian = model()

# objective
pmedian.add( quicksum(d[n,m]*x[n,m] for n in range(N) for m in range(M)) )

# single_x
for m in range(M):
  pmedian.add( quicksum(x[n,m] for n in range(N)) == 1 )

# bound_y
for n in range(N):
  for m in range(M):
    pmedian.add( x[n,m] - y[n] <= 0 )

# num_facilities
pmedian.add( quicksum(y[n] for n in range(N)) == P )
```

P-Median Model: Pyomo

```
model = ConcreteModel()

model.N = RangeSet(N)
model.M = RangeSet(M)
model.x = Var(model.N, model.M, bounds=(0,1), initialize=0)
model.y = Var(model.N, bounds=(0,1), initialize=0)
model.d = Param(model.N, model.M,
                initialize=lambda  n, m, model : random.uniform(1.0,2.0))

def rule(model):
        return quicksum(model.d[n,m]*model.x[n,m] for n in model.N for m in model.M)
model.obj = Objective(rule=rule)

def rule(model, m):
        return quicksum(model.x[n,m] for n in model.N) ==  1.0
model.single_x = Constraint(model.M, rule=rule)

def rule(model, n,m):
        return model.x[n,m] - model.y[n] <= 0.0
model.bound_y = Constraint(model.N, model.M, rule=rule)

def rule(model):
        return quicksum(model.y[n] for n in model.N) == P
model.num_facilities = Constraint(rule=rule)
```

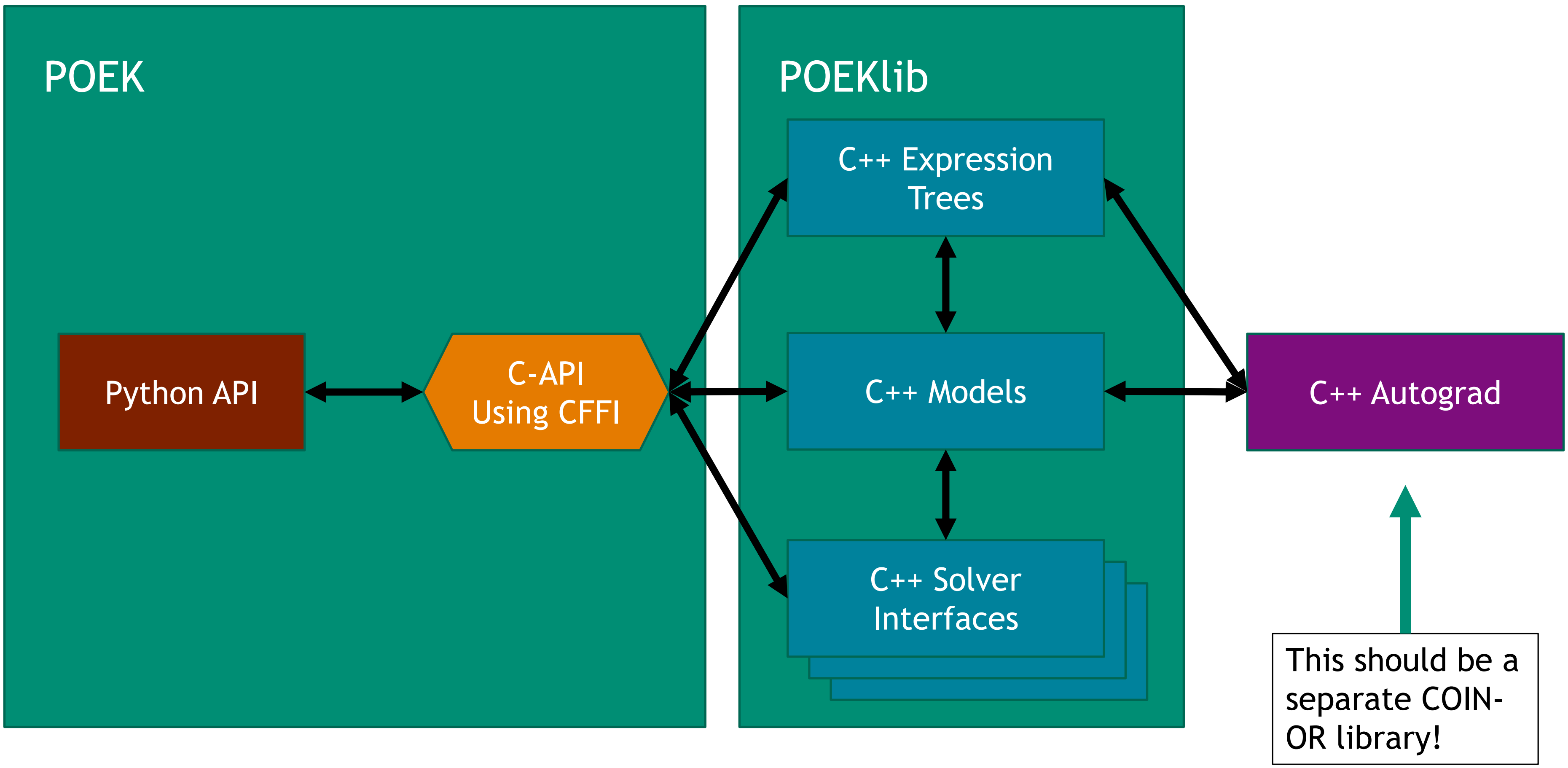
VERY Preliminary Performance Results

Time to construct model and setup Ipopt
- P-median: N=M=640, P=1
- Cpython 3.6

	Pyomo	POEK	Speedup
Build Model	22.7	10.3	2.2x
Setup Ipopt	44.6	6.8	6.6x
TOTAL	67.3	17.1	3.9x

Observations
- CFFI interface is fast enough to justify many Python-C calls when constructing expressions
- Eliminating expression translation and file I/O in NL writer is a big win (NL files)
- Matrix/Vector expressions would make model build faster

Rethinking POEK



Workflow / Software Stack with Pyomo

Decomposition Algorithm
(Progressive Hedging)

Algebraic Modeling Language
Pyomo

Optimization Problem

Core Expressions

Evaluation, Derivatives (AD), Linear Algebra
ASL, HSL

Nonlinear Solver

POEK:
Python Optimization Expression Kernel

Have we struck the correct balance?

PyNumero:
Python Framework for NLP Algorithms

Questions?