Qi Chen and Braulio Brunaud

Pyomo and JuMP – Modeling environments for the 21st century

EWO Seminar
Carnegie Mellon University
March 10, 2017
Disclaimer

1. All the information provided is coming from the standpoint of two somewhat expert users in Pyomo and JuMP, former GAMS users

2. All the content is presented to the best of our knowledge
The Optimization Workflow

- These tasks usually involve many tools:
  - Databases
  - Excel
  - GAMS/AMPL/AIMMS/CPLEX
  - Tableau

- Can a single tool be used to complete the entire workflow?
Optimization Environments Overview

- Different input sources
- Easy to model
- AMPL: Specific modeling constructs
  - Piecewise
  - Complementarity conditions
  - Logical Implications
- Solver independent models
- Developing complex algorithms can be challenging
- Large number of solvers available
- Commercial
Optimization Environments Overview

- Different input sources
- Build input forms
- Easy to model
- Multiplatform:
  - Standalone
  - Web
  - Mobile
- Solver independent models
- Build visualizations
- Commercial
Optimization Environments Overview

- Different input sources
- Hard to model
- Access to the full power of a solver
- Access to a broad range of tools
- Solver-specific models
- Building visualizations is hard
- Open source and free

Visualization → End User

Modeling

Advanced algorithms

Optimization Expert

solver-specific code
Optimization Environments Overview

- Different input sources
- Easy to model
- Access to the full power of a solver
- Access to a broad range of tools
- Helpful modeling extensions
  - Uncertainty
  - Multiobjective (MultiJuMP)
  - Pyomo/DAE
- Solver-independent models
- Building visualizations is hard
- Open source and free
Optimization Environments Overview

Visualization → End User

Modeling

Advanced algorithms → Optimization Expert

GAMS

AMPL

AIMMS

JuMP

solver-specific code
## Optimization Environments Overview

<table>
<thead>
<tr>
<th></th>
<th>GAMS</th>
<th>AMPL</th>
<th>AIMMS</th>
<th>GLPK</th>
<th>PYOMO</th>
<th>JuMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Input</td>
<td>hard</td>
<td>hard</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Data Manipulation</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Modeling</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>hard</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Advanced Algorithms</td>
<td>hard</td>
<td>hard</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Solvers Availability</td>
<td>✔</td>
<td>✔</td>
<td>limited</td>
<td>✗</td>
<td>limited</td>
<td>limited</td>
</tr>
<tr>
<td>Visualization</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>hard</td>
<td>hard</td>
<td>hard</td>
</tr>
<tr>
<td>License</td>
<td>$</td>
<td>$</td>
<td>$$$</td>
<td>free</td>
<td>free</td>
<td>free</td>
</tr>
</tbody>
</table>
version 0.5.1

http://julialang.org/

- New programming language for scientific computing
- Aims to combine
  - Flexibility from Python
  - Math power from Matlab and R
  - High performance from C++
- Designed with performance in mind
- Designed for parallel computing
- Metaprogramming
  - Code that generates code
- Very easy to code

Learn:
https://learnxinyminutes.com/docs/julia/
JuMP

- Julia module for Mathematical Programming
- Provides objects for Model, Variables, Constraints and Expressions
- Easy implementation of callbacks
- Supports Unicode characters
- Supports:
  - MINLP
  - Second order conic programming
  - Semi-definite programming
- @ sign means a macro (metaprogramming)

In [9]:

```julia
using JuMP

m = Model()
@variable(m, x[1:2] >= 0)
@variable(m, α == 5)

@constraint(m, con[i in 1:2], x[i] <= α)
@objective(m, Max, sum(x[i] for i in 1:2))

@show m
```

Out[9]:

```
max x₁ + x₂
Subject to
  x₁ - α ≤ 0
  x₂ - α ≤ 0
  xᵢ ≥ 0  ∀i ∈ {1, 2}
  α = 5
```

jump.readthedocs.io
1. Database

```julia
using MySQL

con = mysql_connect("localhost","bbrunaud","***","DFLdata")

# Get Demands
query = ""
    SELECT Customer, Product, Period, Demand
    FROM Demands
    WHERE
        (ProductNumber BETWEEN $firstP AND $lastP) 
        AND 
        (Period BETWEEN $t1 AND $tN) 
        AND 
        (SiteCode BETWEEN $firstCcode AND $lastCcode)
    ""

demands = mysql_execute(con,query)
```

For other databases check [JuliaDB](https://github.com/JuliaDB/JuliaDB.jl)
2. Excel

```
In [2]: using ExcelReaders
using DataFrames

demand = readxl(DataFrame, "demand.xlsx", "Sheet1!A1:D5")
```

<table>
<thead>
<tr>
<th></th>
<th>Customer</th>
<th>Product</th>
<th>Period</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CUS1</td>
<td>A</td>
<td>1.0</td>
<td>36.0</td>
</tr>
<tr>
<td>2</td>
<td>CUS1</td>
<td>B</td>
<td>1.0</td>
<td>57.0</td>
</tr>
<tr>
<td>3</td>
<td>CUS2</td>
<td>A</td>
<td>1.0</td>
<td>30.0</td>
</tr>
<tr>
<td>4</td>
<td>CUS2</td>
<td>B</td>
<td>1.0</td>
<td>44.0</td>
</tr>
</tbody>
</table>
3. Julia Code

Use dictionaries

```julia
In [4]:
demand = Dict(
    ("CUS1","A",1) => 36.0,
    ("CUS1","B",1) => 57.0,
    ("CUS2","A",1) => 30.0,
    ("CUS2","B",1) => 44.0
)
```

Use matrices

```julia
In [5]: TransportationCost = [ 100 1.5; 1.7 100]
Out[5]: 2×2 Array{Float64,2}:
  100.0 1.5
  1.7 100.0

In [6]: TransportationCost[1,2]
Out[6]: 1.5
```

In this case, indices are restricted to be integers
JuMP - A simple example

Solve the following problem using CPLEX

Load Packages

In [17]: using JuMP
   using CPLEX

Declare model

Solver options go inside the parenthesis of CplexSolver()

In [18]: m = Model(solver=CplexSolver())

Out[18]:

\[
\begin{align*}
\text{min} & \quad 0 \\
\text{Subject to} & \quad 0
\end{align*}
\]
### JuMP - A simple example

#### Declare variables and constraints

**In [19]:**
```julia
@variable(m, x)
@variable(m, y[1:4] >= 0)
@constraints m begin
end
```

#### Declare objective and solve

**In [20]:**
```julia
@objective(m, Min, x)
solve(m)
```

```
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 2 rows, 4 columns, and 8 nonzeros.
Presolve time = 0.00 sec. (0.00 ticks)

Iteration log ...
Iteration:  1  Dual infeasibility = 0.000000
Iteration:  2  Dual objective  = -56.375000
```

**Out[20]:**
```
:Optimal
```

#### Save the solution vector

**In [7]:**
```julia
using JLD
save("solution.jld", "sol", m.colVal)
```
JuMP - Under the hood

- All the layers are very thin. This is why creating and manipulating models is very fast.
- Julia is designed to make efficient calls to C or Fortran

JuMP objects

Interface

Solvers interface

C libraries
JuMP - Under the hood

JuMP

MathProgBase

CPLEX.jl

CPLEX C API

JuMP objects
JuMP objects

@variable(m, x >=0)

Interface
Interface

addvar!

Solvers interface
Solvers interface

addvar!

C libraries
C libraries

addcols

Example:
Adding a variable

@variable(m, x >=0)

addvar!

addvar!

addcols
JuMP - Accessing the low level objects

- Let's get the simplex tableau of the simplex example

```
In [24]:
mpb = m.internalModel  # MathProgBase Model
cpx = mpb.inner       # CPLEX Model

    tableau = zeros(length(m.linconstr), m.numCols)

    for k in 1:length(m.linconstr)
        row = CPLEX.get_tableau_row(cpx, k+1)
        tableau[k, :] = row'
    end

    tableau
```

```
Out[24]: 3×5 Array{Float64,2}:
   1.0  -98.0  0.0  -34.0  -10.0
   0.0   6.0  1.0   3.0    2.0
   0.0  -16.0  0.0  -5.0   -1.0
```
Let’s get the simplex tableau of the simplex example

```python
In [24]:
    mpb = m.internalModel  # MathProgBase Model
    cpx = mpb.inner        # CPLEX Model
    tableau = zeros(length(m.linconstr), m.numCols)
    for k in 1:length(m.linconstr)
        row = CPLEX.get_tableau_row(cpx, k+1)
        tableau[k, :] = row'
    end
    tableau

Out[24]:
3x5 Array{Float64,2}:
1.0  -98.0   0.0  -34.0  -10.0
0.0    6.0   1.0    3.0   2.0
0.0 -16.0   0.0  -5.0  -1.0
```
Let's get the simplex tableau of the simplex example

```julia
In [24]:
mpb = m.internalModel  # MathProgBase Model
    cpx = mpb.inner      # CPLEX Model
    tableau = zeros(length(m.linconstr), m.numCols)

    for k in 1:length(m.linconstr)
        row = CPLEX.get_tableau_row(cpx, k+1)
        tableau[k,:] = row'
    end

    tableau
```

```
Out[24]:
3x5 Array{Float64,2}:
  1.0  -98.0  0.0  -34.0  -10.0
  0.0   6.0  1.0   3.0   2.0
  0.0  -16.0  0.0  -5.0  -1.0
```
JuMP - Accessing the low level objects

- Let’s get the simplex tableau of the simplex example

In [24]:
```plaintext
mpb = m.internalModel # MathProgBase Model
cpx = mpb.inner       # CPLEX Model

tableau = zeros(length(m.linconstr),m.numCols)
for k in 1:length(m.linconstr)
    row = CPLEX.get_tableau_row(cpx,k+1)
    tableau[k,:] = row'
end

tableau
```

Out[24]:
```
3×5 Array{Float64,2}:
1.0 -98.0 0.0 -34.0 -10.0
0.0 6.0 1.0 3.0 2.0
0.0 -16.0 0.0 -5.0 -1.0
```

When declaring a model in JuMP it is possible to access every single function in the C API.
JuMP - Extensions

- JuMP is written in very few lines of code (9,000) and it is very simple to understand
- It is not difficult to write extensions
  - **JuMPeR.jl**: for robust optimization
  - **MultiJuMP.jl**: for multi-objective optimization
  - **JuMPChance.jl**: for probabilistic chance constraints
  - **StochDynamicProgramming.jl**: for discrete-time stochastic optimal control problems
  - **PolyJuMP.jl**: for polynomial optimization
  - **StructJuMP.jl**: for block-structured optimization
  - **NLOptControl.jl**: for formulating and solving nonlinear optimal control problems
  - **Complementarity.jl**: for complementarity problems
  - **DSP, Argonne National Lab**: Implements decomposition methods for stochastic mixed-integer programs

http://www.juliaopt.org/packages/
Pros

✓ It’s new
✓ Fast
✓ Free
✓ Easy and simple source code
✓ Access to low level objects
✓ Built with performance in mind
✓ Support through an active community at the Julia forums
✓ Plenty of libraries to support your workflow
  ✓ Data analysis
  ✓ Plotting
  ✓ Statistics
  ✓ It is also possible to call libraries from other languages within Julia: Python, C++, Fortran, R, Matlab, Java, etc
JuMP - Pros and Cons

Pros

✓ It’s new
✓ Fast
✓ Free
✓ Easy and simple source code
✓ Access to low level objects
✓ Built with performance in mind
✓ Support through an active community at the Julia forums
✓ Plenty of libraries to support your workflow
  ✓ Data analysis
  ✓ Plotting
  ✓ Statistics
  ✓ It is also possible to call libraries from other languages within Julia: Python, C++, Fortran, R, Matlab, Java, etc

Cons

☑ It’s new
  ☑ The platform and the supporting packages are not mature enough
  ☑ JuMP version 0.16
☑ No standard solution report
☑ Lack of modeling features
  ☑ Piecewise (SOS are supported though)
  ☑ Disjunctions
  ☑ Indicator Constraints
http://python.org/
Mathematical Modeling in Python

**Why Python?**

- High level coding language: **mature + stable**
  - Used in production by Google, Facebook, IBM, Nasdaq, etc.
  - Deep pool of **experienced developers**
- Enables **fast prototyping** and **integrated** work process
- Many useful **libraries**:
  - Numpy – linear algebra
  - Pandas – Data input/output + parsing
  - Networkx – Network graph analysis + display
  - PyQt – Graphical User Interface
  - Matplotlib – plotting results
  - Python interfaces common for external tools
- Ability to **aggregate data** from multiple sources

---

**Most Popular Coding Languages of 2016**

- **C++**
  - 9.9%
- **Python**
  - 26.7%
- **15-112 Fund. of Prog. & CS**
  - Taken by many CMU undergraduates
  - Python-oriented

**Contributions from 3 undergraduates**

- Eloy Fernandez
- Jacqueline Lewis
- Sunjeev Kale

---

@codeeval
Mathematical Modeling in Python

Why Python?

- High level coding language: **mature + stable**
  - Used in production by Google, Facebook, IBM, Nasdaq, etc.
  - Deep pool of **experienced developers**
- Enables **fast prototyping** and **integrated** work process
- Many useful **libraries**:
  - Numpy – linear algebra
  - Pandas – Data input/output + parsing
  - Networkx – Network graph analysis + display
  - PyQt – Graphical User Interfaces
  - Matplotlib – plotting results
  - Python interfaces common for external tools
- Ability to **aggregate data** from multiple sources
PyQt tool for model visualization

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>L.B.</th>
<th>U.B.</th>
<th>Type</th>
<th>Fixed</th>
<th>Active</th>
<th>Stale</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SimpleMeaSheet</td>
<td>Iinks</td>
<td></td>
<td></td>
<td>Measheet</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>absorb</td>
<td></td>
<td></td>
<td>Block</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>regen</td>
<td></td>
<td></td>
<td>Absorber</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xhx</td>
<td></td>
<td></td>
<td>Regenerator</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>memix</td>
<td>s1</td>
<td></td>
<td></td>
<td>General</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td>General</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s3</td>
<td></td>
<td></td>
<td>General</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s4</td>
<td></td>
<td></td>
<td>Constraint</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s5</td>
<td></td>
<td></td>
<td>Constraint</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s6</td>
<td></td>
<td></td>
<td>Constraint</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td>Constraint</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s6.expanded</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s7.expanded</td>
<td>mea_h2o_ratio</td>
<td>0.1124</td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eq_mea_h2o_ratio</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eq_mumiky_int_cuts</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmp_int_cuts</td>
<td></td>
<td></td>
<td></td>
<td>IndexedCons...</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Courtesy of John Eslick, NETL
Why Python? (cont.)

Tools for effective model stewardship

- **Programmatic documentation generation**
  - Easily generate webpage with code and model documentation
- **Standardized code style guide (PEP8)**
  - Automatic code formatting
  - Makes models more consistently readable to multiple users
- **Software version control (Git)**
- **Automatic testing**
  - Ensures that future changes do not silently break existing functionality
  - Identifies affected code
  - Automatically executed when changes made to code
- **Facilitates management of change from user to user**
Why Python? (cont.)

Tools for effective model stewardship

- **Programmatic documentation** generation
  - Easily generate webpage with code and model documentation
- **Standardized code style guide (PEP8)**
  - **Automatic code formatting**
    - Makes models more consistently readable to multiple users
- **Software version control (Git)**
- **Automatic testing**
  - Ensures that future changes do not silently break existing functionality
  - Identifies affected code
  - Automatically executed when changes made to code
- **Facilitates management of change** from user to user
Why Python? (cont.)

Tools for effective model stewardship

- Programmatic documentation generation
  - Easily generate webpage with code and model documentation
- Standardized code style guide (PEP8)
  - Automatic code formatting
  - Makes models more consistently readable to multiple users
- Software version control (Git)
- Automatic testing
  - Ensures that future changes do not silently break existing functionality
  - Identifies affected code
  - Automatically executed when changes made to code
- Facilitates management of change from user to user

```bash
elchen@QC-CMU-Tower:/git/superS nosetests > /test.log

.......
Ran 14 tests in 7.828s
OK
```
Pyomo: **Python Optimization Modeling Objects**

- **Meta-Solvers**
  - Generalized Benders
  - Progressive Hedging
  - Linear bilevel
  - Linear MPEC

- **Modeling Extensions**
  - Disjunctive programming
  - Stochastic programming
  - Bilevel programming
  - Differential equations
  - Equilibrium constraints

- **Core Optimization Objects**
  - Model
  - Transformations

- **Solver Interfaces**
  - CPLEX
  - Gurobi
  - Xpress
  - GLPK
  - CBC
  - BARON
  - OpenOpt
  - NEOS
  - AMPL Solver Library
    - Ipopt
    - KNITRO
    - Bonmin
    - Couenne

**Python library for optimization**

- Key optimization objects
  - Sets
  - Variables
  - Parameters
  - Constraints
  - Objective
  - Model
  - Solvers
- Able to manipulate core modeling and optimization objects
Mathematical Modeling in Pyomo

Pyomo features

- Pyomo library offers algebraic modeling language (like AMPL and GAMS) and solver interfaces in Python
- Allows programmatic access to modeling objects (sets, variables, constraints, etc.)
  - Enables advanced model design

# simple.py
from pyomo.environ import *

M = ConcreteModel()
M.x1 = Var()
M.x2 = Var(bounds=(-1,1))
M.x3 = Var(bounds=(1,2))
M.c1 = Constraint(expr=M.x1 == M.x2 + M.x3)
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)

opt_result = SolverFactory('ipopt').solve(M)
opt_model = M
Pyomo advantages

### Features
- Open source + free to use/modify commercially
- Modeling extensions
  - Pyomo.GDP
  - Pyomo.DAE
  - PySP (stochastic)
  - Bilevel programming
- Strong support for modular modeling
- Ability to perform programmatic model transformations
- Allows use of advanced solution algorithms

### Modeling extensions
- GDP automatic reformulations
- Express DAE in terms of differential equations
- PySP: support for scenario generation and progressive hedging algorithm
- Ability to build models at high level of abstraction

### Diagram
- GDP
- MINLP big-M
- MINLP Hull Reform.
- DAE
- NLP
Modular Modeling

Advantages of modularity

- Ability to create nested unit models
- Each unit model has its own namespace
  - No need to keep track of T1, T2, T3…
  - Instead: reactor.T, flash.T
- Easier maintenance of unit models
- Ability to test models independently

Chen, Qi; Grossmann, IE. “Recent Developments and Challenges in Optimization-Based Flowsheet Synthesis.” *Annual Review of Chemical and Biomolecular Engineering*. In press, expected July 2017.

Figure from Carnegie Mellon University
Programmatic model transformations

**Dynamic model transformations**
- Utility functions to add linear relaxations
- Dynamic activation or deactivation of nonlinear expressions
- Automatic generation and addition of linearizations
- Model state introspection
  - Propagation of fixed variables
  - Deactivation of redundant constraints

---

**Code**

```python
for s in streams:
    for c in components:
        add_mccormick_relaxation(mc_flow_block,
                                  z=fc[comp, s],
                                  x=F,
                                  y=mole_frac[c, s],
                                  nsegs=5, (s, c))
```

**Result**

Automatically adds variables and equations for a 5-segment piecewise McCormick relaxation for the bilinear relation $f_{c,s} = F_s x_s$ for each component $c$ and stream $s$
Programmatic model transformations

Dynamic model transformations

- Utility functions to add linear relaxations
- Dynamic activation or deactivation of nonlinear expressions
- Automatic generation and addition of linearizations
- Model state introspection
  - Propagation of fixed variables
  - Deactivation of redundant constraints

Nonlinear constraint deactivation/reactivation

```python
Model.deactivate_nonlinear_constraints()
Model.reactivate_nonlinear_constraints()
```

Implication

- Required in order to switch between linearized and nonlinear forms of model for LOA
- **Dynamically detects** which equations are nonlinear (vs. linear due to fixed variables)
- Ability to remember which equations need to be reactivated
**Programmatic model transformations**

<table>
<thead>
<tr>
<th>Dynamic model transformations</th>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Utility functions to add linear relaxations</td>
<td>Model.apply_OA_strategy()</td>
<td>▪ Analyzes nonlinear expressions and computes Jacobians</td>
</tr>
<tr>
<td>▪ Dynamic activation or deactivation of nonlinear expressions</td>
<td>Model.add_oa_cuts()</td>
<td>▪ Evaluates the Jacobians at the current variable values when add_oa_cuts is called in order to construct the outer approximation constraints</td>
</tr>
<tr>
<td>▪ Automatic generation and addition of linearizations</td>
<td></td>
<td>▪ Automatically adds the OA constraints to the model</td>
</tr>
<tr>
<td>▪ Model state introspection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Propagation of fixed variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Deactivation of redundant constraints</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Programmatic model transformations

**Dynamic model transformations**
- Utility functions to add linear relaxations
- Dynamic activation or deactivation of nonlinear expressions
- Automatic generation and addition of linearizations
- Model state introspection
  - Propagation of fixed variables
  - Deactivation of redundant constraints

**Code**

```python
Model.propagate_var_fix(tmp=True)
Model.introspect_flows()
```

**Result**

- Looks for simple constraints of form $A = B$. If $A$ or $B$ is fixed, then fix the other variable to the same value. Propagates for all members of the equality set.
- Looks to see if a stream has all of its flow variables deactivated. If so, deactivates the constraints associated with the stream.
  - Avoids redundant equations, making NLP solver more robust
Implementation of logic-based outer approximation (LOA)

- `do_LOA()`
  - `init_LOA()`
  - `build_model()`

- `apply_OA_strategy()`

- `solve_set_cover_MIP()`

- Compute + activate set covering objective
- Activate linearizations
- Deactivate nonlinear constraints
- Deactivate fathomed units
- Deactivate OA cuts
- Solve MIP
- Reactivate OA cuts
Implementation of logic-based outer approximation (LOA)

- **build_model()**
- **apply_OA_strategy()**
- **init_LOA()**
  - **solve_set_cover_MIP()**
  - **solve_local_NLP()**

- **do_LOA()**

- **Activate normal objective**
- **Activate relevant nonlinear constraints**
- **Deactivate linearizations**
- **Deactivate OA cuts**
- **Apply NLP initialization strategy**
- **Propagate fixed variables**
- **Introspect flows**
- **Deactivate trivial constraints**
- **Set minimum flows**
- **Solve NLP**
- **Unset minimum flows**
- **Undo pre-solve model transformations**
- **Update upper bound**
- **Save values if best so far**
- **If not optimal, reinitialize + try again**
Implementation of logic-based outer approximation (LOA)

- do_LOA()
  - init_LOA()
    - solve_set_cover_MIP()
    - solve_local_NLP()
    - add_oa_cuts()
  - apply_OA_strategy()
    - Evaluate Jacobians
    - Generate + add outer approximation (OA) constraints corresponding to all nonlinear constraints
Implementation of logic-based outer approximation (LOA)

- **do_LOA()**
  - **init_LOA()**
    - solve_set_cover_MIP()
    - solve_local_NLP()
    - add_oa_cuts()
    - add_int_cut()

- Generate + add integer cut
Implementation of logic-based outer approximation (LOA)

1. **do_LOA()**
   - **init_LOA()**
     - solve_set_cover_MIP()
     - solve_local_NLP()
     - add_oa_cuts()
     - add_int_cut()

   - Loop until all units covered

2. **build_model()**
3. **apply_OA_strategy()**
Implementation of logic-based outer approximation (LOA)

- do_LOA()
  - init_LOA()
  - solve_local_MIP()

- apply_OA_strategy()
  - Activate augmented Lagrangian OA objective
  - Activate linearizations
  - Deactivate nonlinear constraints
  - Deactivate fathomed units
  - Activate OA cuts
  - Solve MIP
  - Update lower bound
Implementation of logic-based outer approximation (LOA)

- build_model()
- apply_OA_strategy()
  - init_LOA()
  - solve_local_MIP()
  - solve_local_NLP()

- do_LOA()
  - Activate normal objective
  - Activate relevant nonlinear constraints
  - Deactivate linearizations
  - Deactivate OA cuts
  - Apply NLP initialization strategy
  - Propagate fixed variables
  - Introspect flows
  - Deactivate trivial constraints
  - Set minimum flows
  - Solve NLP
  - Unset minimum flows
  - Undo pre-solve model transformations
  - Update upper bound
  - Save values if best so far
  - If not optimal, reinitialize + try again
Implementation of logic-based outer approximation (LOA)

- `build_model()`
- `apply_OA_strategy()`
- `do_LOA()`
  - `init_LOA()`
  - `solve_local_MIP()`
  - `solve_local_NLP()`
  - If optimal
    - `add oa_cuts()`

- Evaluate Jacobians
- Generate + add outer approximation (OA) constraints corresponding to all nonlinear constraints
Implementation of logic-based outer approximation (LOA)

1. build_model()
2. apply_OA_strategy()
3. do_LOA()
   - init_LOA()
   - solve_local_MIP()
   - solve_local_NLP()
   - if optimal
     - add_oa_cuts()
     - add_int_cut()
   - generate + add integer cut
Implementation of logic-based outer approximation (LOA)

- do_LOA()
  - build_model()
  - apply_OA_strategy()
    - init_LOA()
    - solve_local_MIP()
    - solve_local_NLP()
    - if optimal
      - add_oa_cuts()
      - add_int_cut()

- Loop until upper bound and lower bound converge
Implementation of logic-based outer approximation (LOA)

- `do_LOA()`
  - `init_LOA()`
  - `solve_local_MIP()`
  - `build_model()`
  - `apply_OA_strategy()`

- Loop until upper bound and lower bound converge
Implementation of logic-based outer approximation (LOA)

- implement_logic_based_outer_approximation
- do_LOA()
  - init_LOA()
  - solve_local_MIP()
  - solve_local_NLP()
  - loop
    - build_model()
    - apply_OA_strategy()
      - add_oa_cuts()
      - add_int_cut()
      - if optimal
- loop
  - Loop until upper bound and lower bound converge

Conclusion
- Pyomo allows for advanced algorithm development
## Pyomo current shortcomings

<table>
<thead>
<tr>
<th>Opportunities for growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Does not ship with solvers</td>
</tr>
<tr>
<td>- Can be difficult for novice users to install solvers and set up solver licensing</td>
</tr>
<tr>
<td>- Lack of pre-processor [active development]</td>
</tr>
<tr>
<td>- Documentation is sparse for advanced usage and modeling extensions: difficult learning curve past basic modeling</td>
</tr>
<tr>
<td>- Plugin system is complex: difficult to figure out how to contribute new plugins</td>
</tr>
<tr>
<td>- Best practices for saving/loading of model state currently unclear</td>
</tr>
<tr>
<td>- Would be useful in some multiprocessing applications</td>
</tr>
<tr>
<td>- Backwards compatibility of new releases not guaranteed</td>
</tr>
<tr>
<td>- Old release will remain usable, but without newer features</td>
</tr>
</tbody>
</table>
Conclusions + Remarks

When to use Pyomo

- Perform data input, cleaning, problem formulation, optimization, analysis, and visualization in integrated workflow
- Construct models at a high level with advanced concepts and apply custom transformations into algebraic forms (GDP, DAE, stochastic, bilevel)
- Development or prototyping of advanced multi-step solution algorithms
- Second layer to traditional AMLs

When not to use Pyomo

- Model solution time in deployment is dominated by model compilation time
- Require access to certain unavailable solvers, e.g. DICOPT
- Individual solver licensing and deployment is a headache
- Onerous conversion cost

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.

Disclaimer: This presentation was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
Conclusion

Pyomo and JuMP are excellent open source modeling environments embedded in fully featured programming languages, suitable for users at all levels.
Appendix
1. Download and install Julia
   b. You can also try it without installing it at JuliaBox.com

2. Install JuMP
   a. julia> Pkg.add("JuMP")

3. Install your favorite solver (example CPLEX)
   b. You need to get your license

4. Install the julia package for your solver
   a. julia> Pkg.add("CPLEX")
### Pyomo – Installation and References

<table>
<thead>
<tr>
<th>Install Python</th>
</tr>
</thead>
</table>
| - [https://www.python.org/downloads/](https://www.python.org/downloads/)
| - Alternative implementations also popular, especially Anaconda: [https://www.python.org/download/alternatives/](https://www.python.org/download/alternatives/) |

<table>
<thead>
<tr>
<th>Install Pyomo</th>
</tr>
</thead>
</table>
| - [http://www.pyomo.org/installation](http://www.pyomo.org/installation)
| - `pip install pyomo` |

<table>
<thead>
<tr>
<th>Install Solvers</th>
</tr>
</thead>
</table>
| - IPOPT: [https://www.coin-or.org/Ipopt/documentation/node10.html](https://www.coin-or.org/Ipopt/documentation/node10.html)

<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
</table>
| - Pyomo docs: [http://www.pyomo.org/documentation](http://www.pyomo.org/documentation)
| - Help forum: [https://groups.google.com/forum/#!forum/pyomo-forum](https://groups.google.com/forum/#!forum/pyomo-forum) |