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

Data Input

Data Manipulation

Optimization

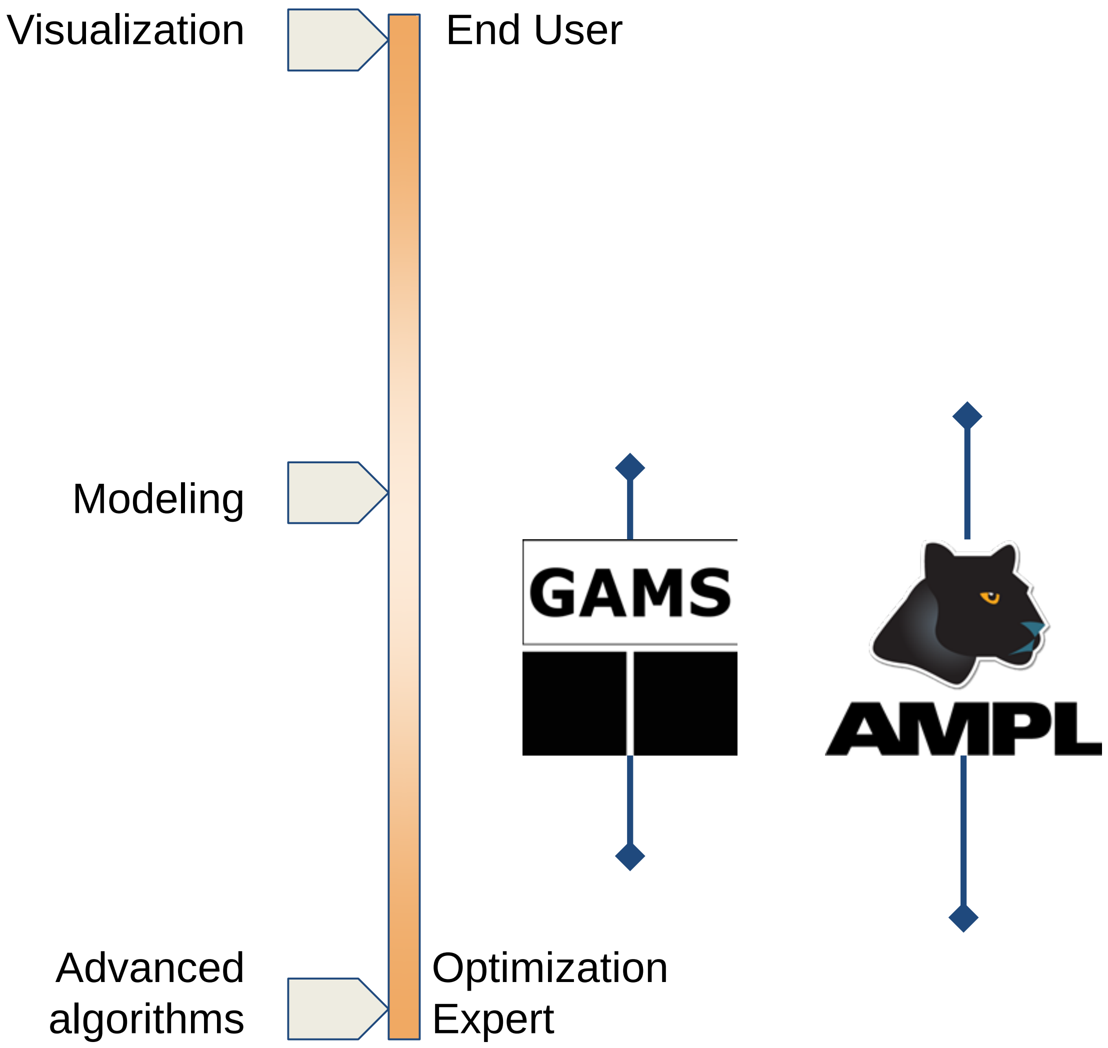
Analysis

Visualization

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

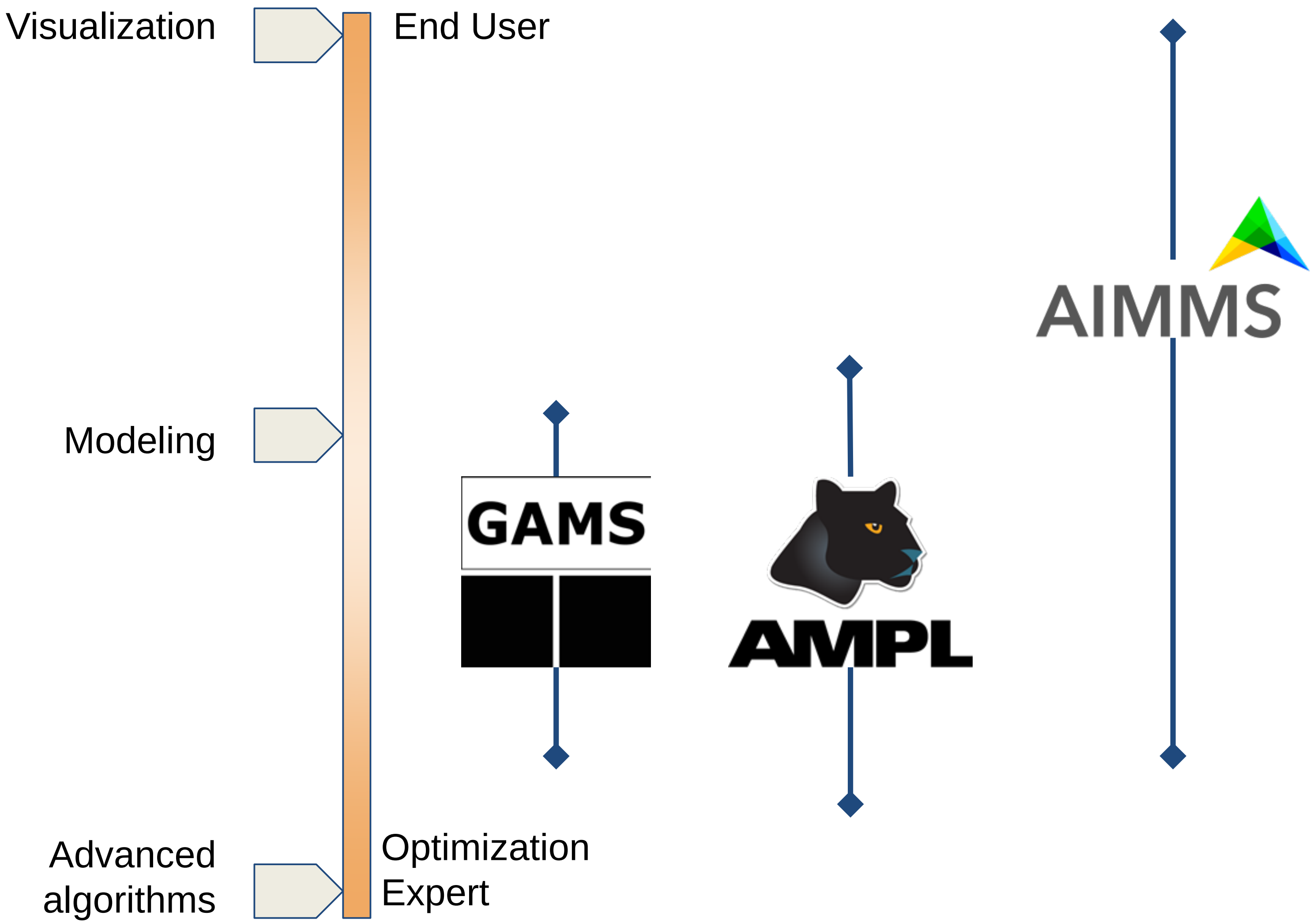
- Can a single tool be used to complete the entire workflow?

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

Visualization

End User

- 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

Modeling

Advanced algorithms

Optimization Expert

solver -specific code

Optimization Environments Overview

Visualization

End User

- 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

Modeling

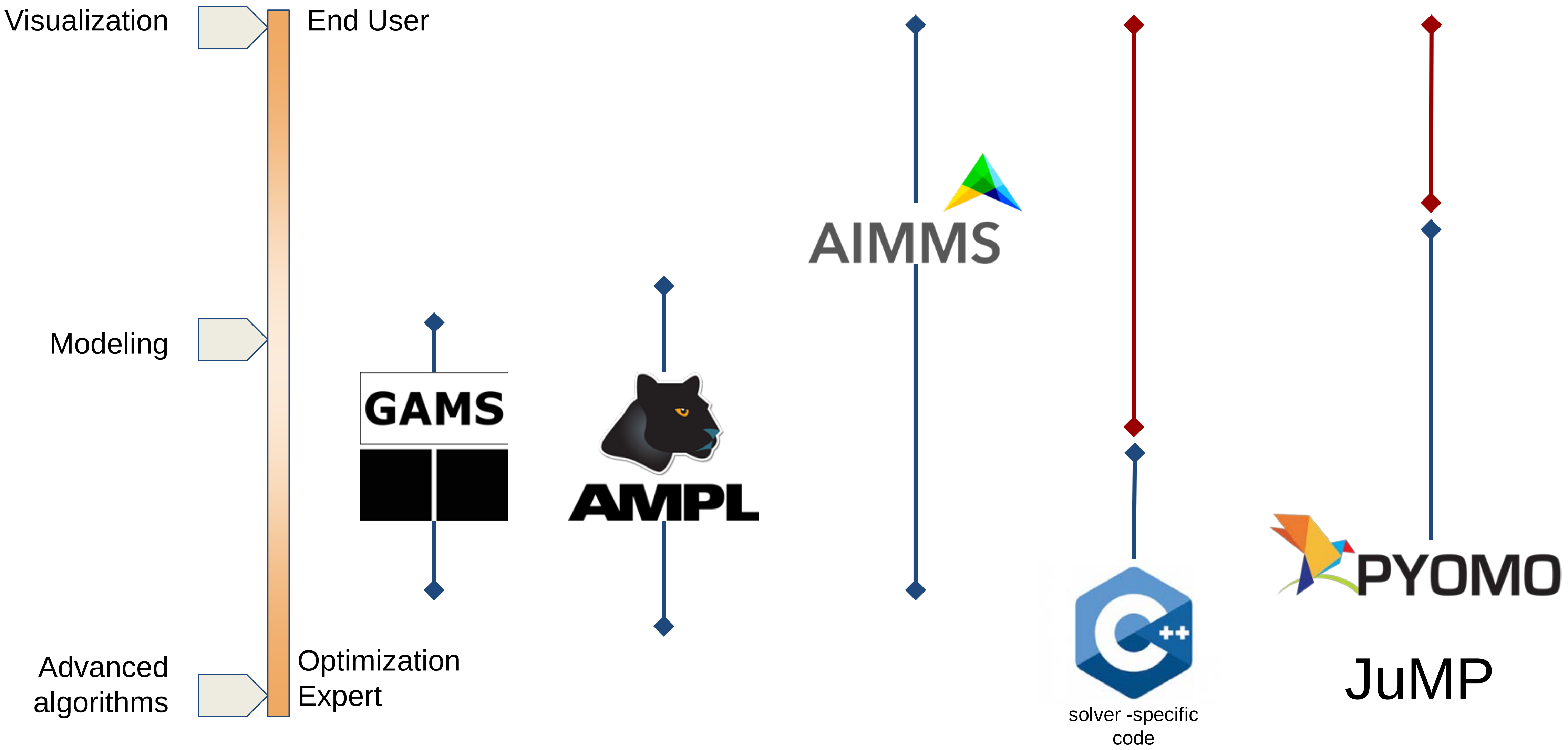
Advanced algorithms

Optimization Expert

solver -specific code

JuMP

Optimization Environments Overview



Optimization Environments Overview

	GAMS	AMPL	AIMMS	C++	PYOMO	JuMP
Data Input	hard	hard	✓	✓	✓	✓
Data Manipulation	✕	✕	✓	✓	✓	✓
Modeling	✓	✓	✓	hard	✓	✓
Advanced Algorithms	hard	hard	✓	✓	✓	✓
Solvers Availabilty	✓	✓	limited	✕	limited	limited
Visualization	✕	✕	✓	hard	hard	hard
License	$	$	$$	free	free	free

version 0.5.1

http://julialang.org/

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

- 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

JuMP

- **Ju**lia module for **M**athematical **P**rogramming
- 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]:

```
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
```

```
m = Maximization problem with:
 * 2 linear constraints
 * 3 variables
Solver is default solver
```

Out[9]:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{Subject to} \quad & x_1 - \alpha \leq 0 \\ & x_2 - \alpha \leq 0 \\ & x_i \geq 0 \quad \forall i \in \{1,2\} \\ & \alpha = 5 \end{aligned}$$

jump.readthedocs.io

JuMP - Data Input

1. Database

In [ ]:
```
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

JuMP - Data Input

2. Excel

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

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

Out[2]:

	Customer	Product	Period	Demand
1	CUS1	A	1.0	36.0
2	CUS1	B	1.0	57.0
3	CUS2	A	1.0	30.0
4	CUS2	B	1.0	44.0

JuMP - Data Input

3. Julia Code

Use dictionaries

```
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

```
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

In [22]:
```
@show m
```

```
m = Minimization problem with:
 * 3 linear constraints
 * 5 variables
Solver is Cplex
```

Out[22]:

$$\begin{aligned} \min \quad & x \\ \text{Subject to} \quad & x + 16y_1 + 19y_2 + 23y_3 + 28y_4 = 0 \\ & 2y_1 + 3y_2 + 4y_3 + 5y_4 \leq 9 \\ & 6y_1 + y_2 + 3y_3 + 2y_4 \leq 2 \\ & y_i \geq 0 \quad \forall i \in \{1,2,3,4\} \\ & x free \end{aligned}$$

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]:

$$\begin{aligned} \min \quad & 0 \\ \text{Subject to} \quad & \end{aligned}$$

JuMP - A simple example

Declare variables and constraints

In [19]:
```
@variable(m, x)
@variable(m, y[1:4] >= 0)

@constraints m begin
    x + 16y[1] + 19y[2] + 23y[3] + 28y[4]  == 0
    2y[1] + 3y[2] + 4y[3] + 5y[4]   <= 9
    6y[1] + 1y[2] + 3y[3] + 2y[4]   <= 2
end
```

Declare objective and solve

In [20]:
```
@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
```

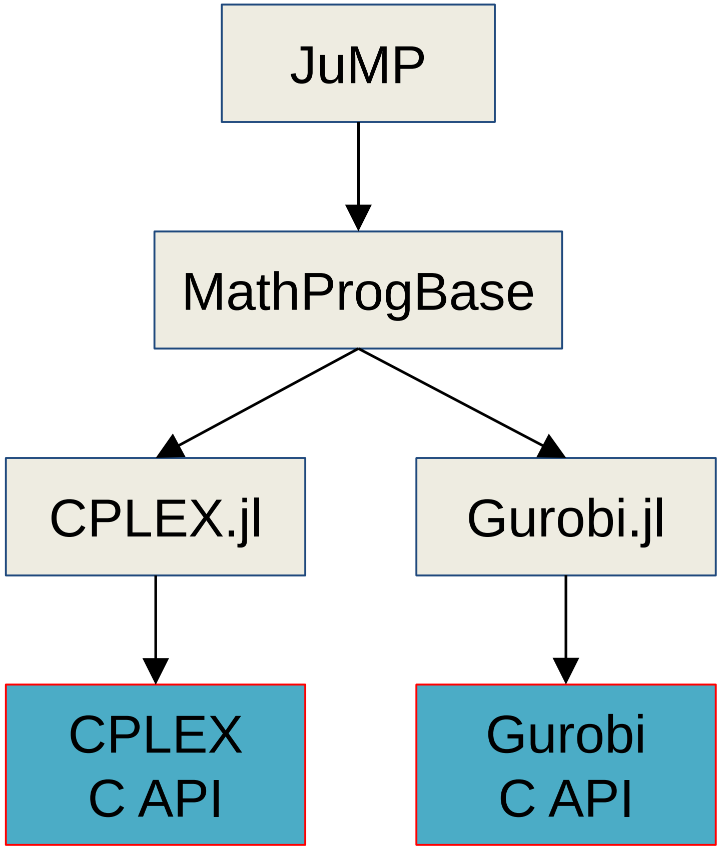
Out[20]: :Optimal

Save the solution vector

In [7]:
```
using JLD

save("solution.jld", "sol", m.colVal)
```

JuMP - Under the hood



JuMP objects

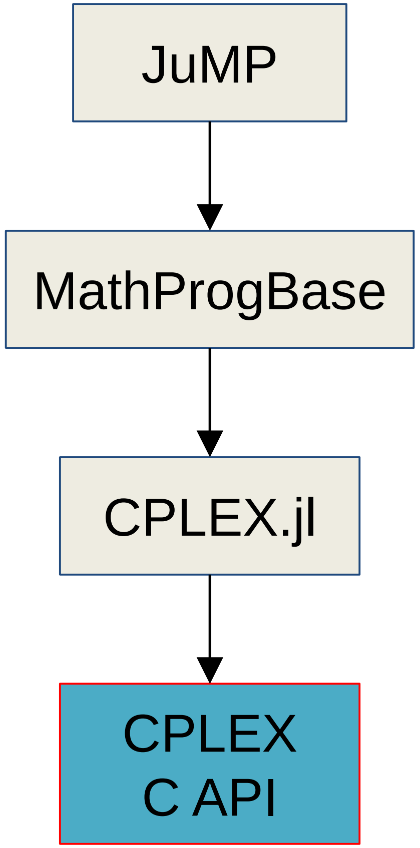
Interface

Solvers interface

C libraries

- 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 - Under the hood



JuMP objects	@variable(m, x >=0)
Interface	addvar!
Solvers interface	addvar!
C libraries	addcols

Example: Adding a variable

JuMP - Accessing the low level objects

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

In [24]:
```julia
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
```

JuMP - Accessing the low level objects

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

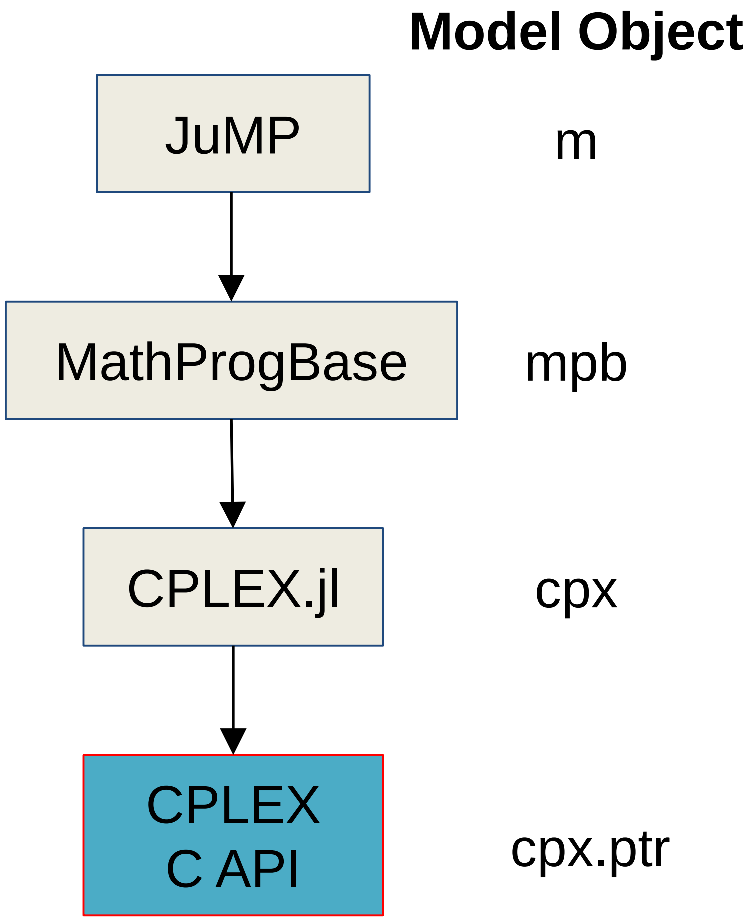
```
In [24]: mpb = m.internalModel # MathProgBase Model
         cpx = mpb.inner       # CPLEX Model

         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
```



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
```

	Model Object	Function
JuMP	m	
MathProgBase	mpb	
CPLEX.jl	cpx	get_tableau_row
CPLEX C API	cpx.ptr	CPXbinvarow

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
```

	Model Object	Function
JuMP	m	
MathProgBase	mpb	
CPLEX.jl	cpx	get_tableau_row
CPLEX C API	cpx.ptr	CPXbinvarow

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/

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

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

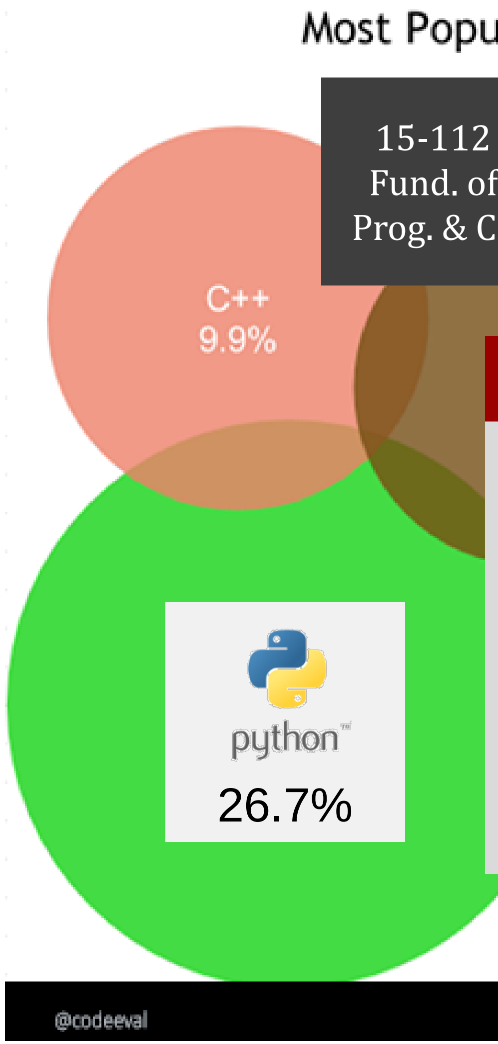
- ❑ It's new
  - ❑ The platform and the supporting packages are not mature enough
  - ❑ JuMP version 0.16
- ❑ No standard solution report
- ❑ Lack of modeling features
  - ❑ Piecewise (SOS are supported though)
  - ❑ Disjunctions
  - ❑ Indicator Constraints

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



15-112 Fund. of Prog. & CS

- Taken by many CMU undergraduates
- Python-oriented

Contributions from 3 undergraduates


Eloy Fernandez

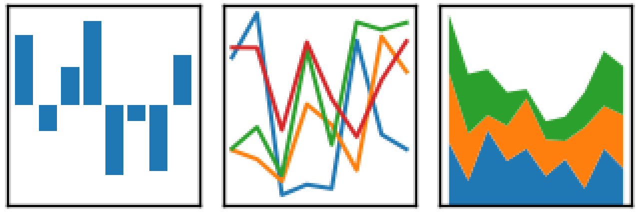

Jacqueline Lewis

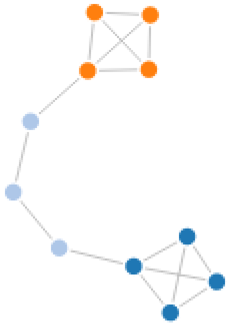

Sunjeev Kale

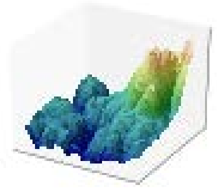
Mathematical Modeling in Python

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

Pyomo Model Viewer

Model | Solver

Name	Value	L.B.	U.B.	Type	Fixed	Active	Stale	Doc
_SimpleMeaSheet				MeaSheet		true		
links				Block		true		
absorb				Absorber		true		CO2 abso
regen				Regenerator		true		Solvent r
xhx				General		true		Lean/Rich
reboiler				General		true		Regenera
mumix				General		true		Makeup s
s1				Constraint		false		
s2				Constraint		false		
s3				Constraint		false		
s4				Constraint		false		
s5				Constraint		false		
s6				Constraint		false		
s7				Constraint		false		
s1.expanded				IndexedCons...		true		
s2.expanded				IndexedCons...		true		
s3.expanded				IndexedCons...		true		
s4.expanded				IndexedCons...		true		
s5.expanded				IndexedCons...		true		
s6.expanded				IndexedCons...		true		
s7.expanded				IndexedCons...		true		
mea_h2o_ratio	0.1124			Var	true	true	true	
eq_mea_h2o_ratio				Constraint		true		
eq_mumix_y				Constraint		true		
int_cuts				IndexedCons...		true		
tmp_int_cuts				IndexedCons...		true		

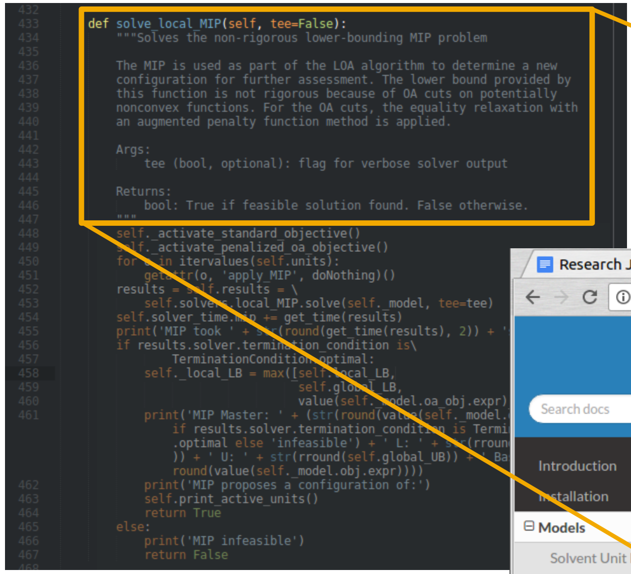
Courtesy of John Eslick, NETL

Solve | Load | Save | OK

Why Python? (cont.)

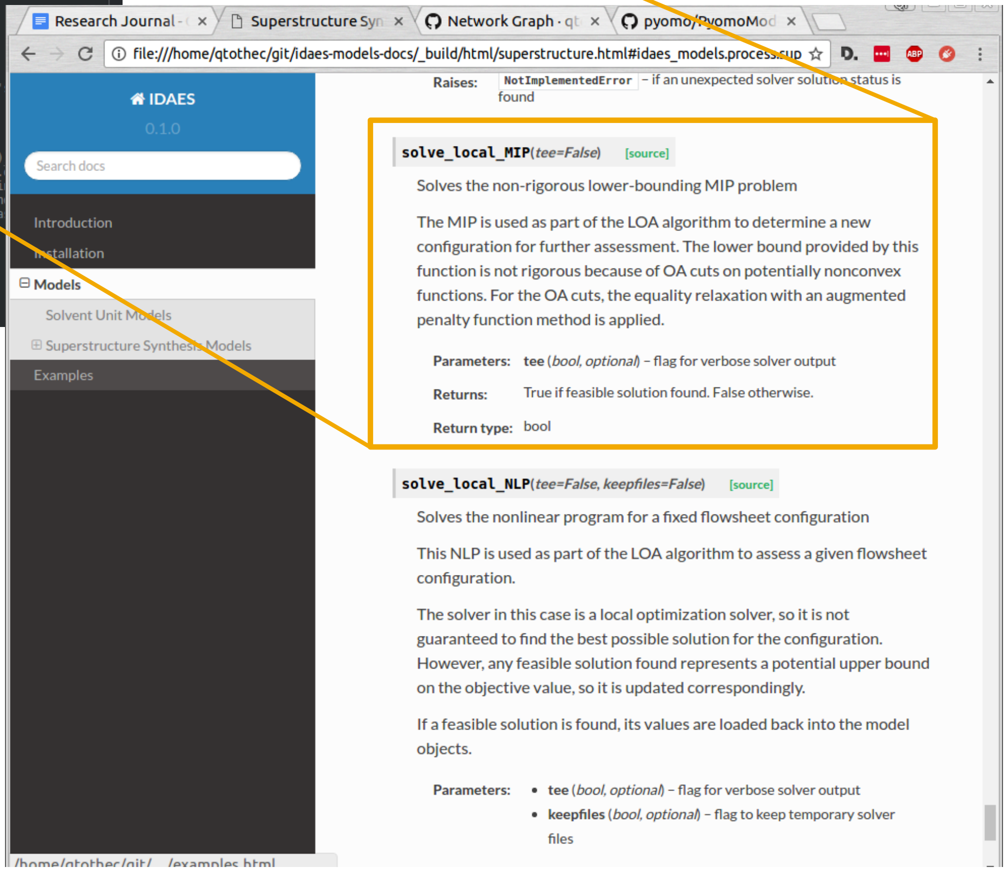
Tools for effective model stewardship

- 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



SPHINX

PYTHON DOCUMENTATION GENERATOR



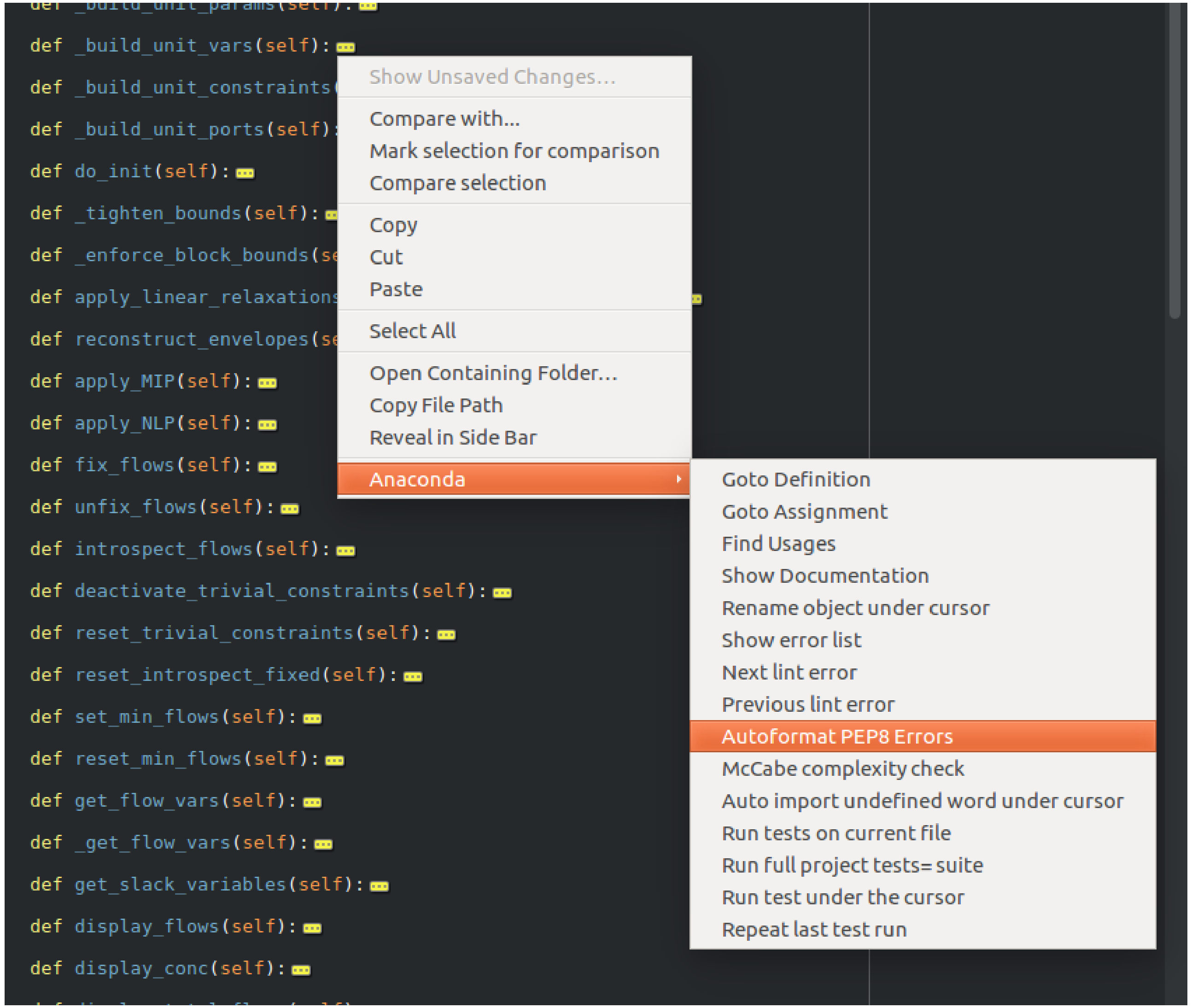
CAPD CENTER

Carnegie Mellon University

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



Carnegie Mellon University

Why Python? (cont.)

Tools for effective model stewardship

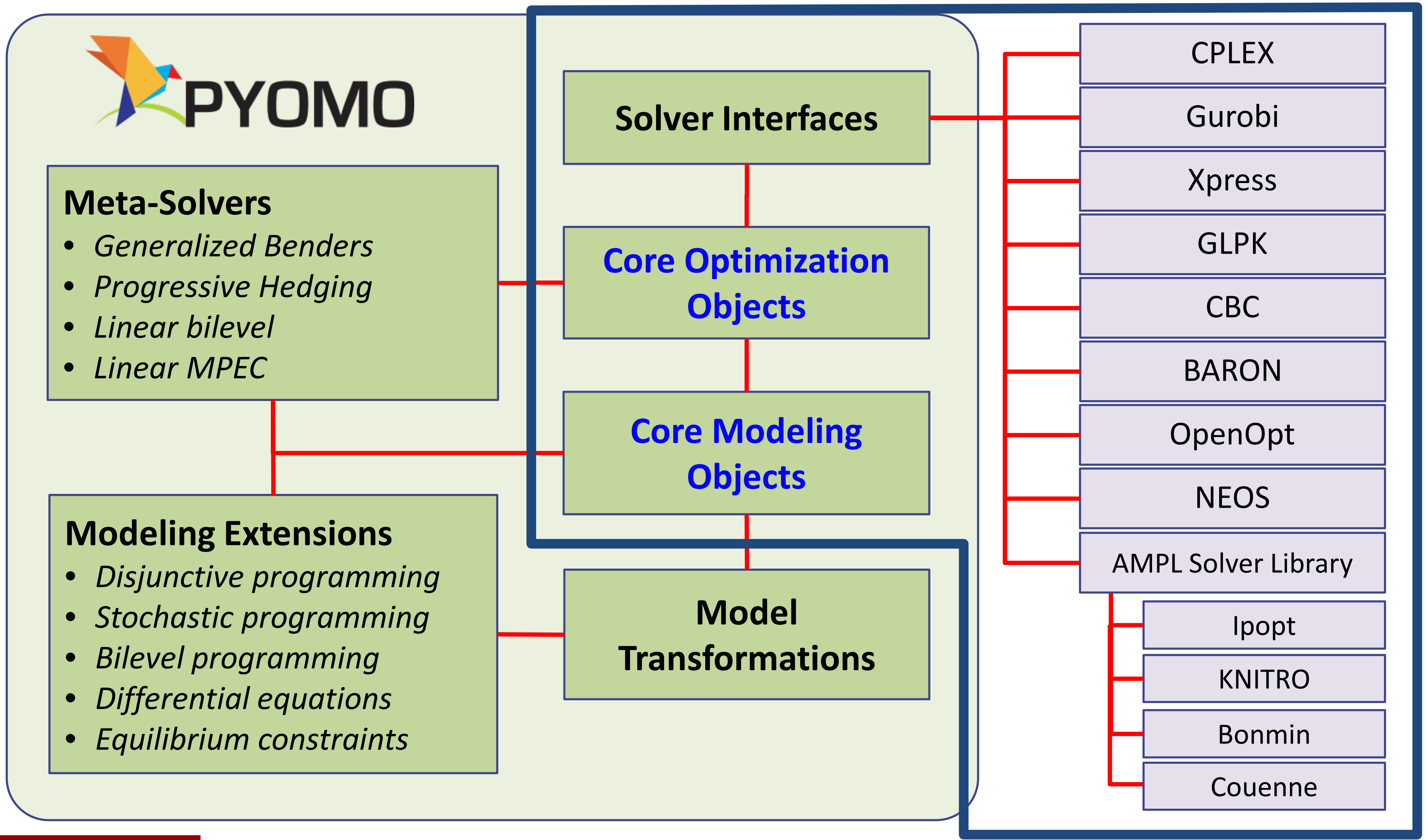
- Programmatic documentation generation
  - Easily generate webpage with code and model documentation
- Standardized code style guide (PEP8)
  - Automatic code formatting
  - Makes models more consistently readable to multiple users
- Software version control (Git)
- **Automatic testing**
  - Ensures 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

```
qichen@QC-CMU-Tower:~/git/super$ nosetests > ~/test.log
nose.config: INFO: Ignoring files matching ['^\\.', '^_', '^setup\\.py$']
test_case_1 (idaes_models.process.superstructure_synthesis.methanol.blocks.tests.test
_compressor.TestCompressor) ... ok
test_case_2 (idaes_models.process.superstructure_synthesis.methanol.blocks.tests.test
_compressor.TestCompressor) ... ok
test_case_1 (idaes_models.process.superstructure_synthesis.methanol.blocks.tests.test
_flash.TestFlash) ... ok
test_case_2 (idaes_models.process.superstructure_synthesis.methanol.blocks.tests.test
_flash.TestFlash) ... ok
test_case_1 (idaes_models.process.superstructure_synthesis.methanol.blocks.tests.test
_reactor.TestReactor) ... ok
test_LOA (idaes_models.process.superstructure_synthesis.methanol.test_main.TestMethan
ol) ... ok
test_case_pb (idaes_models.process.superstructure_synthesis.rpb.blocks.tests.test_pac
ked_bed.TestPackedBed) ... ok
test_case_rpb (idaes_models.process.superstructure_synthesis.rpb.blocks.tests.test_pa
cked_bed.TestPackedBed) ... ok
test_0rpb_1pb (idaes_models.process.superstructure_synthesis.rpb.test_rpb_main.TestPa
ckedBed) ... ok
test_1rpb_0pb (idaes_models.process.superstructure_synthesis.rpb.test_rpb_main.TestPa
ckedBed) ... ok
test_1rpb_1pb (idaes_models.process.superstructure_synthesis.rpb.test_rpb_main.TestPa
ckedBed) ... ok
test_2rpb_2pb (idaes_models.process.superstructure_synthesis.rpb.test_rpb_main.TestPa
ckedBed) ... ok
test_default_LOA (idaes_models.process.superstructure_synthesis.ruiz_water.test_main.
TestWater) ... ok
test_trial01_LOA (idaes_models.process.superstructure_synthesis.ruiz_water.test_main.
TestWater) ... ok

----------------------------------------------------------------------
Ran 14 tests in 7.828s

OK
qichen@QC-CMU-Tower:~/git/super$
```

Pyomo: *Python Optimization Modeling Objects*



Python library for optimization

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

Carnegie Mellon University

Mathematical Modeling in Pyomo

CAPD CENTER

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
```

Carnegie Mellon University

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

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

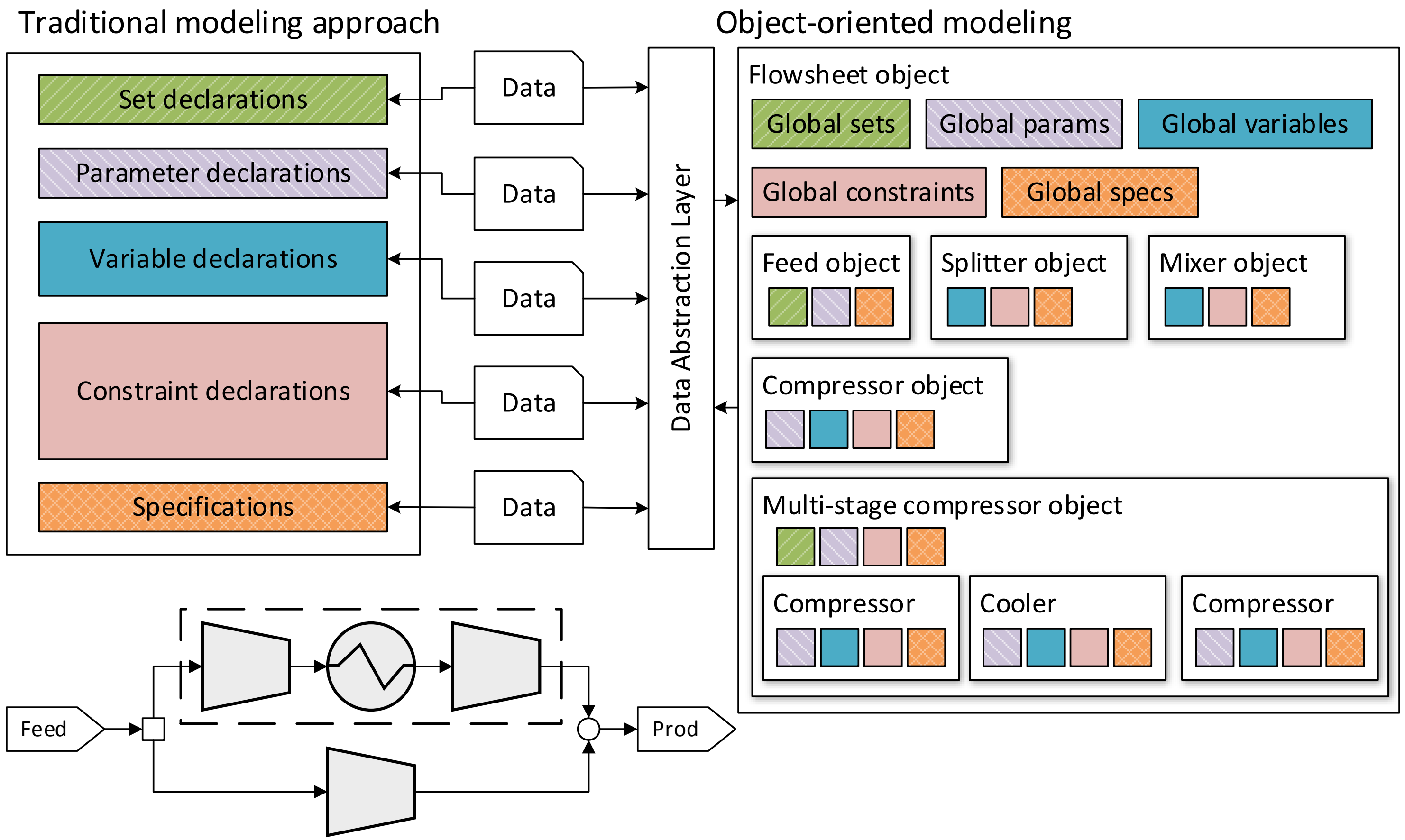


Figure from

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.

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

```
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

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

Nonlinear constraint deactivation/reactivation

```
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

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

```
Model.apply_OA_strategy()
Model.add_oa_cuts()
```

Result

- Analyzes nonlinear expressions and computes Jacobians
- Evaluates the Jacobians at the current variable values when `add_oa_cuts` is called in order to construct the outer approximation constraints
- Automatically adds the OA constraints to the model

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

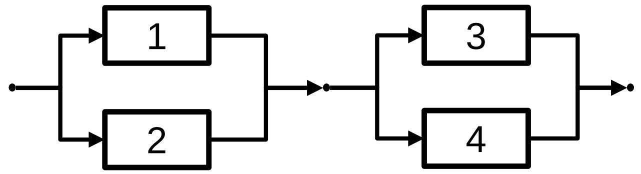
Code

```
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)



`build_model()`

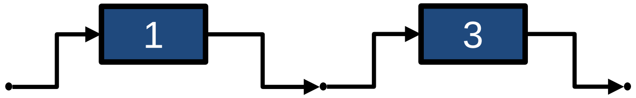
`apply_OA_strategy()`

`do_LOA()`

`init_LOA()` → `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()

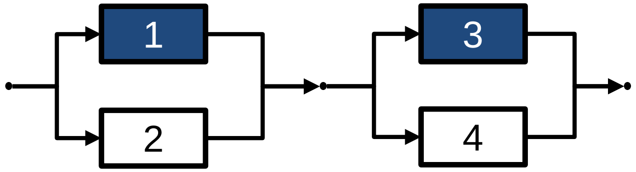
do_LOA()

init_LOA()

- solve_set_cover_MIP()
- solve_local_NLP()

- 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_set_cover_MIP()`
- `solve_local_NLP()`
- `add_oa_cuts()`

- Evaluate Jacobians
- Generate + add outer approximation (OA) constraints corresponding to all nonlinear constraints

Implementation of logic-based outer approximation (LOA)

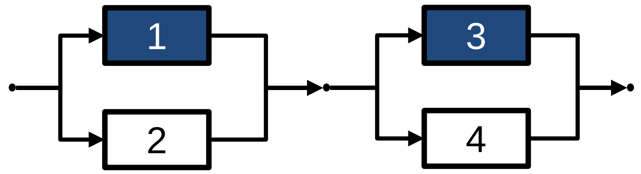
build_model()

apply_OA_strategy()

do_LOA()

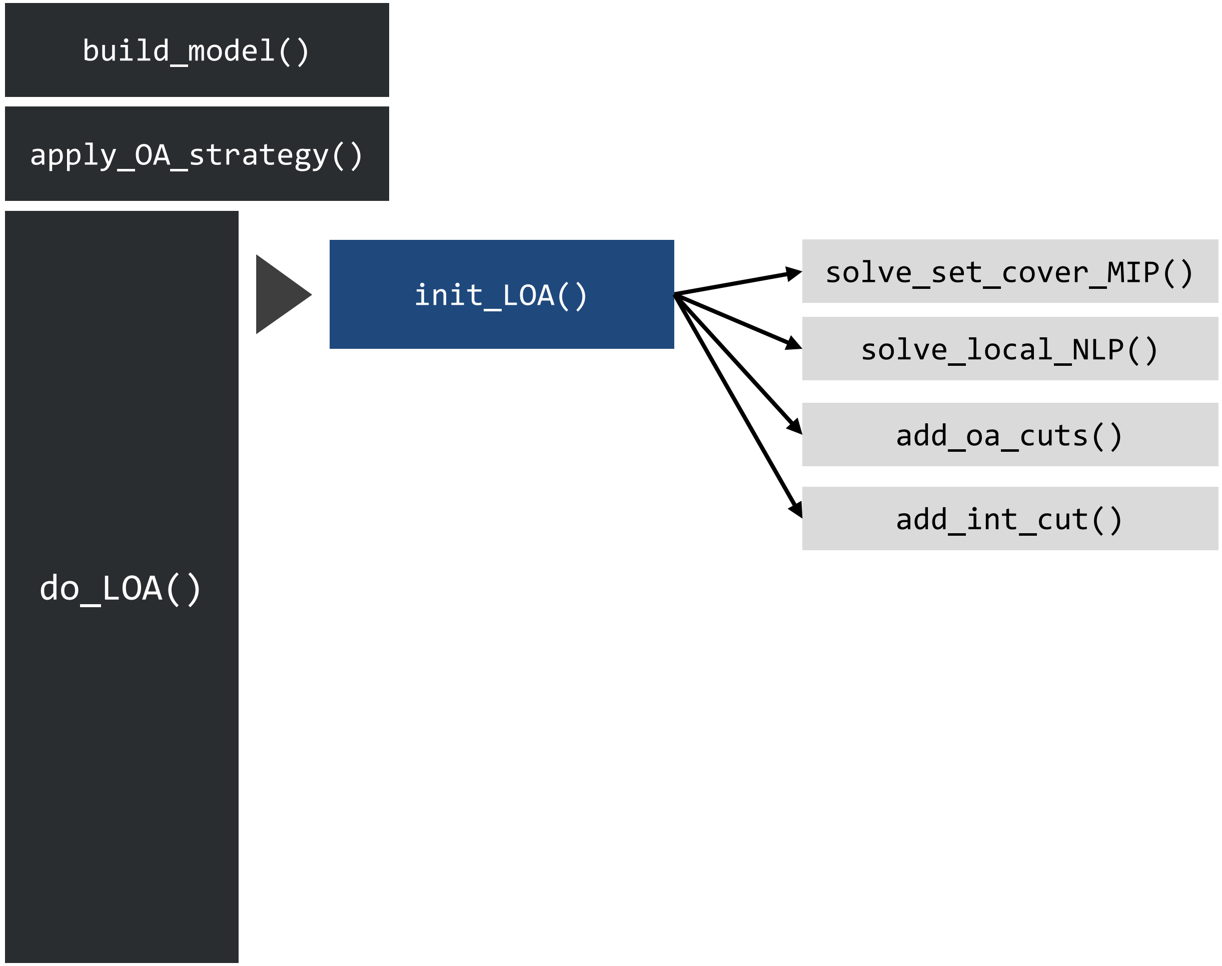
init_LOA()

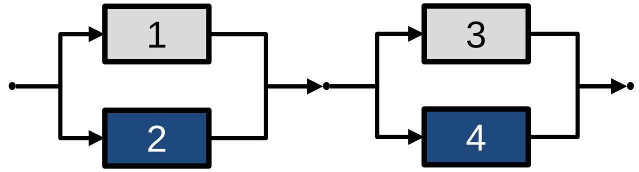
- solve_set_cover_MIP()
- solve_local_NLP()
- add_oa_cuts()
- add_int_cut()



- Generate + add integer cut

Implementation of logic-based outer approximation (LOA)





- Loop until all units covered

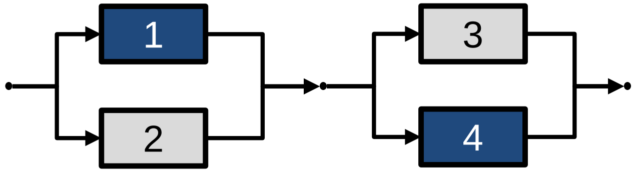
Implementation of logic-based outer approximation (LOA)

build_model()

apply_OA_strategy()

do_LOA()

- init_LOA()
- solve_local_MIP()



- Activate augmented Lagrangian OA objective
- Activate linearizations
- Deactivate nonlinear constraints
- Deactivate fathomed units
- Activate OA cuts
- Solve MIP
- Update lower bound

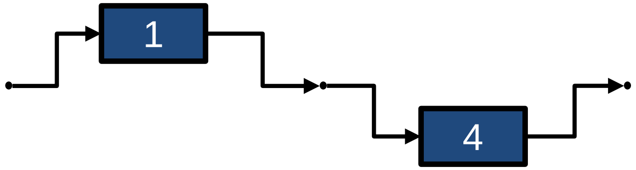
Implementation of logic-based outer approximation (LOA)

build_model()

apply_OA_strategy()

do_LOA()

- init_LOA()
- solve_local_MIP()
- solve_local_NLP()



- 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()

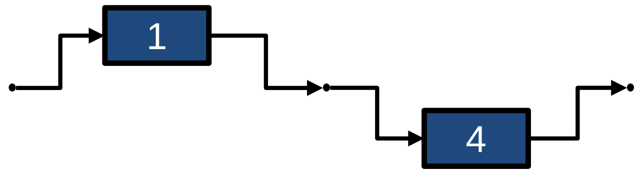
apply_OA_strategy()

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)

build_model()

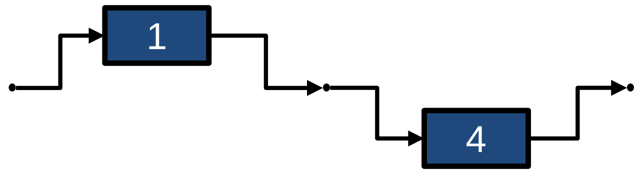
apply_OA_strategy()

do_LOA()

- init_LOA()
- solve_local_MIP()
- solve_local_NLP()

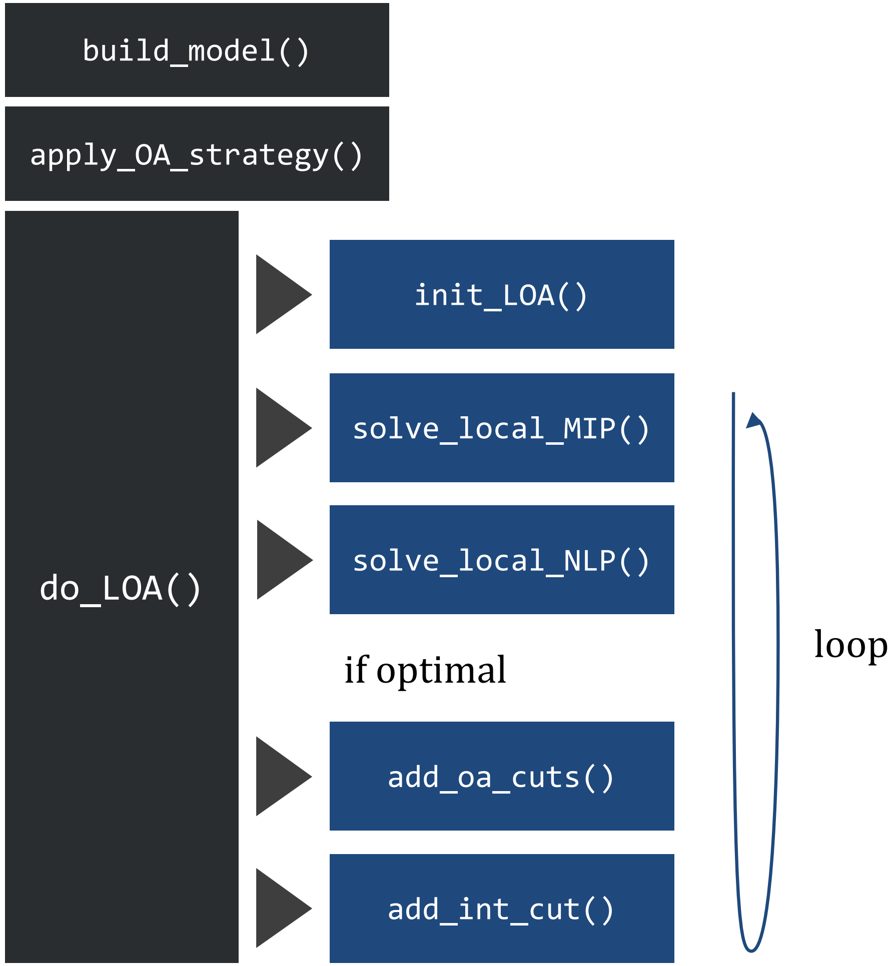
if optimal

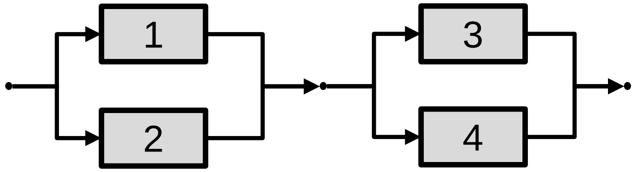
- add_oa_cuts()
- add_int_cut()



- Generate + add integer cut

Implementation of logic-based outer approximation (LOA)





- Loop until upper bound and lower bound converge

Implementation of logic-based outer approximation (LOA)

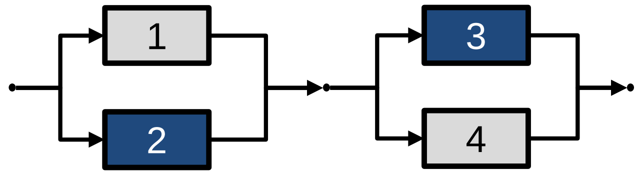
build_model()

apply_OA_strategy()

do_LOA()

- init_LOA()
- solve_local_MIP()

loop



- Loop until upper bound and lower bound converge

Implementation of logic-based outer approximation (LOA)

build_model()

apply_OA_strategy()

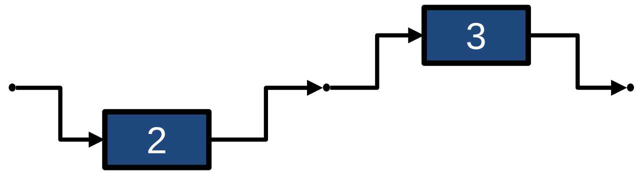
do_LOA()

- init_LOA()
- solve_local_MIP()
- solve_local_NLP()

if optimal

- add_oa_cuts()
- add_int_cut()

loop



- Loop until upper bound and lower bound converge

Conclusion

- Pyomo allows for advanced algorithm development

Pyomo current shortcomings

Opportunities for growth

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

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.

Conclusion

Pyomo and JuMP are excellent open source modeling environments embedded in fully featured programming languages, suitable for users at all levels

Appendix

JuMP - Quick Installation Guide

1. Download and install Julia
    a. http://julialang.org/downloads/
    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)
    a. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud
    b. You need to get your license
4. Install the julia package for your solver
    a. julia> Pkg.add(“CPLEX”)

Pyomo – Installation and References

Install Python	▪ https://www.python.org/downloads/ ▪ Alternative implementations also popular, especially Anaconda: https://www.python.org/download/alternatives/
Install Pyomo	▪ http://www.pyomo.org/installation ▪ `pip install pyomo`
Install Solvers	▪ IPOPT: https://www.coin-or.org/Ipopt/documentation/node10.html ▪ Gurobi: http://www.gurobi.com/registration/download-reg
References	▪ Pyomo docs: http://www.pyomo.org/documentation ▪ Help forum: https://groups.google.com/forum/#!forum/pyomo-forum