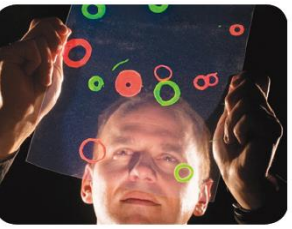
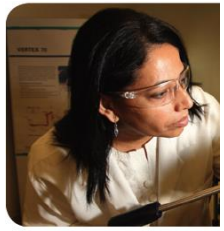
SAND2016-8082C





0. Installing Pyomo

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Use Anaconda3 [all platforms]

Sandia National Laboratories

- Install Anaconda3
  - https://www.continuum.io/downloads
  - Windows/Mac OS X/Linux
  - **Install the Python 3.5 version** for your OS
  - Includes several packages for scientific computing already
  - Supports easy installation of pyomo and solvers

- Install pyomo, solvers, and other packages:
  - From a terminal window, type:

```
conda install -c https://conda.anaconda.org/conda-forge pyomo pyomo.extras

conda install glpk ipopt_bin -c cachemeorg
```

CCR Center for Computing Research

PYOMO COIN-OR

Install with pip [all platforms]

Sandia National Laboratories

- Install Python
  - Linux & Mac OS/X typically have Python pre-installed
  - Scientific Python distributions have many utilities pre-installed
    - http://www.scipy.org/install.html
- Install Pyomo
  - Install in your system
    - `pip install Pyomo`
  - Install in a user directory
    - `pip install --user Pyomo`
- Install auxiliary software
  - Pyomo has conditional dependencies on various third-party packages
    - `pip install pyomo.extras`

CCR Center for Computing Research

PYOMO COIN-OR

Install with Download Scripts [Windows]

- Install Python
  - Linux & Mac OS/X typically have Python pre-installed
  - Scientific Python distributions have many utilities pre-installed
    - http://www.scipy.org/install.html
- Install pip
  - https://bootstrap.pypa.io/get-pip.py
- Install Pyomo: the `get_pyomo.py` script
  - https://software.sandia.gov/trac/pyomo/downloader
- Install auxiliary software: the `get_pyomo_extras.py` script
  - https://software.sandia.gov/trac/pyomo/downloader

Install Trunk / Offline [All platforms]

- Install Python
  - Linux & Mac OS/X typically have Python pre-installed
  - Scientific Python distributions have many utilities pre-installed
    - http://www.scipy.org/install.htm
- Install pyomo: the `pyomo_install` script
  - https://software.sandia.gov/trac/pyomo/downloader
  - Trunk install
    - `pyomo_install --trunk`
  - Install using a bundled zip file (no network access required)
    - `pyomo_install --zip=pyomo-zipfile.zip`
  - Install into a Virtual Python Environment
    - `pyomo_install --venv=pyomo`
  - *Trunk or zipfile options may be combined with virtual environments*

Install Solvers

Sandia National Laboratories

Open Source Solvers

- COIN-OR Binary Distributions
  - http://www.coin-or.org/download/binary/
- GLPK
  - http://ftp.gnu.org/gnu/glpk/
- SCIP
  - http://scip.zib.de/#download

Note: You need to add the solver installation to the PATH environment variable

License

Pyomo released under 3-clause BSD license

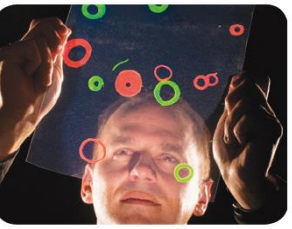
- No restrictions on deployment or commercial use

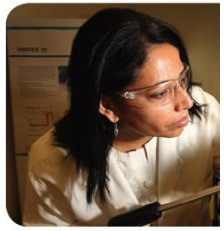
Getting help

- Homepage:
  - www.pyomo.org
- Developer site:
  - https://software.sandia.gov/pyomo
- Mailing lists
  - pyomo-forum@googlegroups.com
  - pyomo-developers@googlegroups.com

Acknowledgements

- William Hart
- Carl Laird
- John Siirola
- Jean-Paul Watson
- David Woodruff





Sandia National Laboratories

Exceptional service in the national interest

1. Overview of Pyomo

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Pyomo Overview

Sandia National Laboratories

Idea: a Pythonic framework for formulating optimization models

- Provide a natural syntax to describe mathematical models
- Formulate large models with a concise syntax
- Separate modeling and data declarations
- Enable data import and export in commonly used formats

Highlights:

- Python provides a clean, intuitive syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
# simple.py
from pyomo.environ import *

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

model = M
```

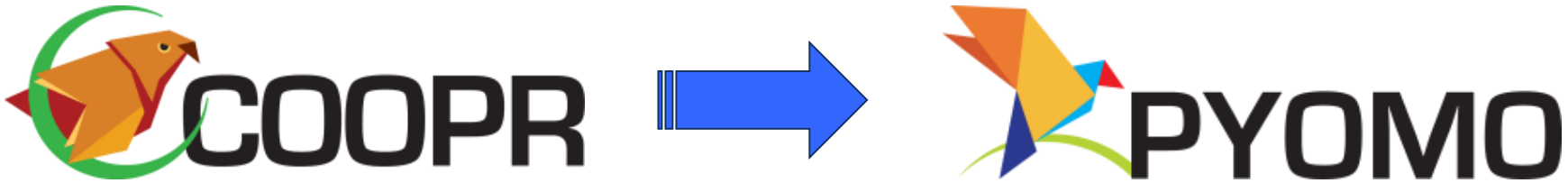
CCR Center for Computing Research

PYOMO COIN-OR

Overview

- What happened to Coopr?
- Three really good questions:
  - Why another Algebraic Modeling Language (AML)?
  - Why Python?
  - Why open-source?
- Pyomo: Software library infrastructure
- Pyomo: Team overview and collaborators / users
- Where to find more information…

What Happened to Coopr?



- Users were installing Coopr but using Pyomo
  - Pyomo modeling extensions were not distinct enough
  - Researchers cited “Coopr/Pyomo”
- Users/Developers were confused by the `coopr` and `pyomo` commands
- Developers were coding in Coopr but talking about Pyomo

We needed to provide clear branding this project!

Optimization Modeling

Sandia National Laboratories

Goal:

- Provide a natural syntax to describe mathematical models
- Formulate large models with a concise syntax
- Separate modeling and data declarations
- Enable data import and export in commonly used formats

Impact:

- Robustly model large constraint matrices (e.g. for MILPs)
- Integrated support of automatic differentiation for complex nonlinear models

Examples:

- AMPL, GAMS, AIMMS, …
- OptimJ, FlopCPP, PuLP, JuMP, …

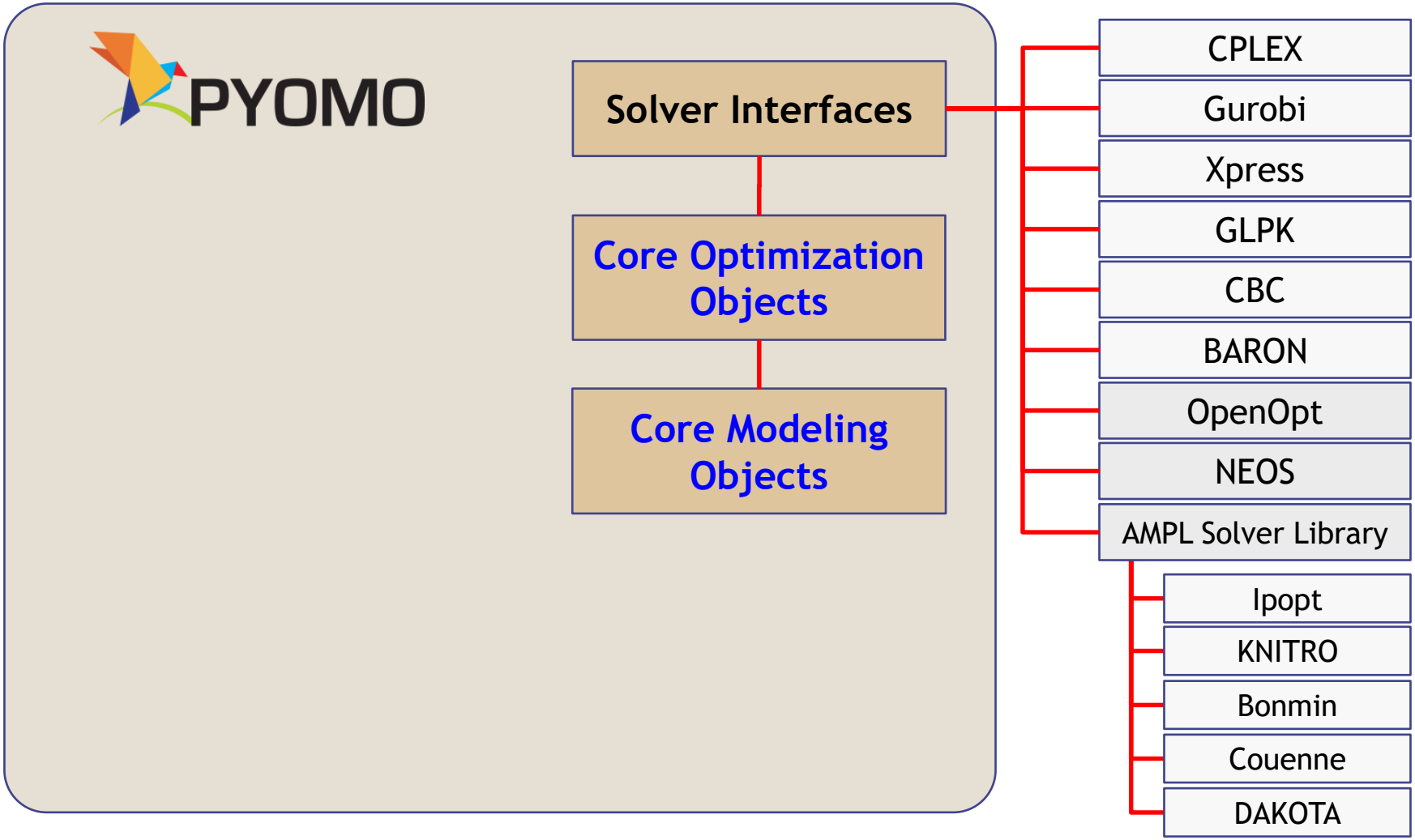
Why Model in Python?

- Full-Featured Library
  - Language features includes functions, classes, looping, namespaces, etc
  - Introspection facilitates the development of generic algorithms
  - Python's clean syntax facilitates rapid prototyping
- Open Source License
  - No licensing issues w.r.t. the language itself
- Extensibility and Robustness
  - Highly stable and well-supported
- Support and Documentation
  - Extensive online documentation and several excellent books
  - Long-term support for the language is not a factor
- Standard Library
  - Includes a large number of useful modules
- Portability
  - Widely available on many platforms

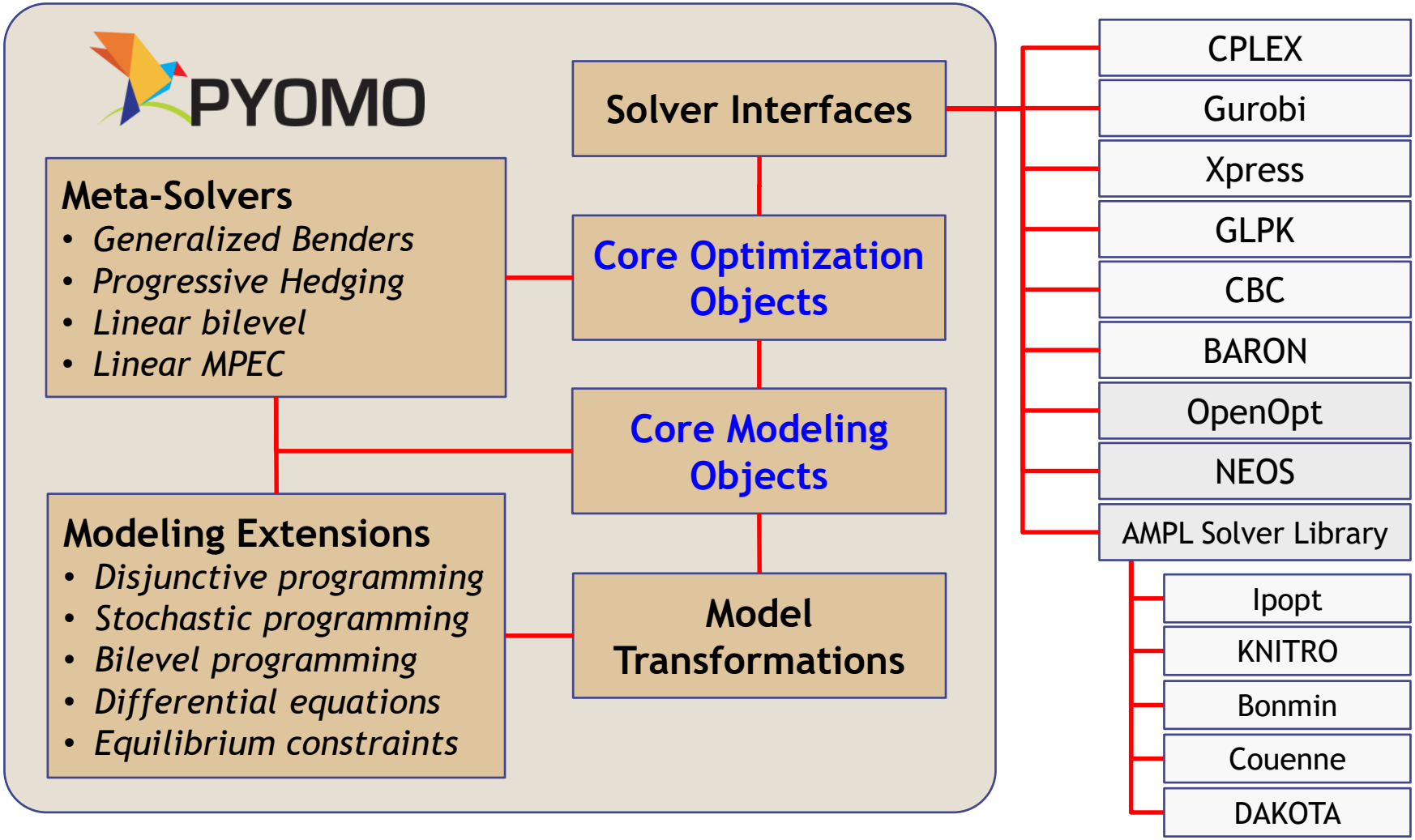
Why Open Source?

- Transparency and reliability
- Foster community involvement
  - Extend the modeling language
  - Develop new solvers / algorithms
  - Interface with additional external utilities
  - “Stone Soup” model
- Flexible licensing
  - Pyomo released under 3-clause BSD license
  - No restrictions on deployment or commercial use

Pyomo at a Glance



Pyomo at a Glance



Survey of Python Modeling Tools

- Pyomo
  - Supports concrete/abstract modeling for LP/MILP/NLP models
  - Modeling extensions for stochastic programming, bilevel, MPEC, etc
  - Separate model objects
- PuLP
  - Supports concrete modeling for LP/MILP models
  - Separate model objects
  - Simple object model
- APLEpy
  - Supports concrete modeling for LP/MILP models
  - Single global model object
- PyMathProg, pyglpk, cplex, gurobi
  - Python interfaces for specific solver tools

More than *just* mathematical modeling

Sandia National Laboratories

Scripting

- Construct models using native Python data
- Iterative analysis of models leveraging Python functionality
- Data analysis and visualization of optimization results

Model transformations (a.k.a. reformulations)

- Automate generation of one model from another
- Leverage Pyomo's object model to apply transformations sequentially
- E.g.: relax integrality, GDP -> Big M

Meta-solvers

- Integrate scripting and/or transformations into optimization solver
- Leverage Python's introspective nature to build "generic" capabilities
- E.g.: progressive hedging, SP extensive form -> MIP

CCR Center for Computing Research

PYOMO COIN-OR

Who Uses Pyomo?

- Students
  - Rose-Hulman, UC Davis, U Texas, Iowa State, NPS
- Researchers
  - Government laboratories
    - Sandia National Labs, Lawrence Livermore National Lab, Los Alamos National Lab, National Energy Technology Lab, Federal Energy Regulation Commission
  - Universities
    - UC Davis, TAMU, Rose-Hulman, UT, USC, GMU, Iowa State, NCSU, U Washington, NPS, U de Santiago de Chile, U Pisa, …
  - Companies

Who Uses Pyomo?

- Software Projects
  - TEMOA – Energy economy optimization models
  - Minpower – Power systems toolkit
  - Water Security Toolkit – Planning/Response for water contamination
  - SolverStudio – Excel plugin for optimization modeling

For More Information

See the Pyomo homepage

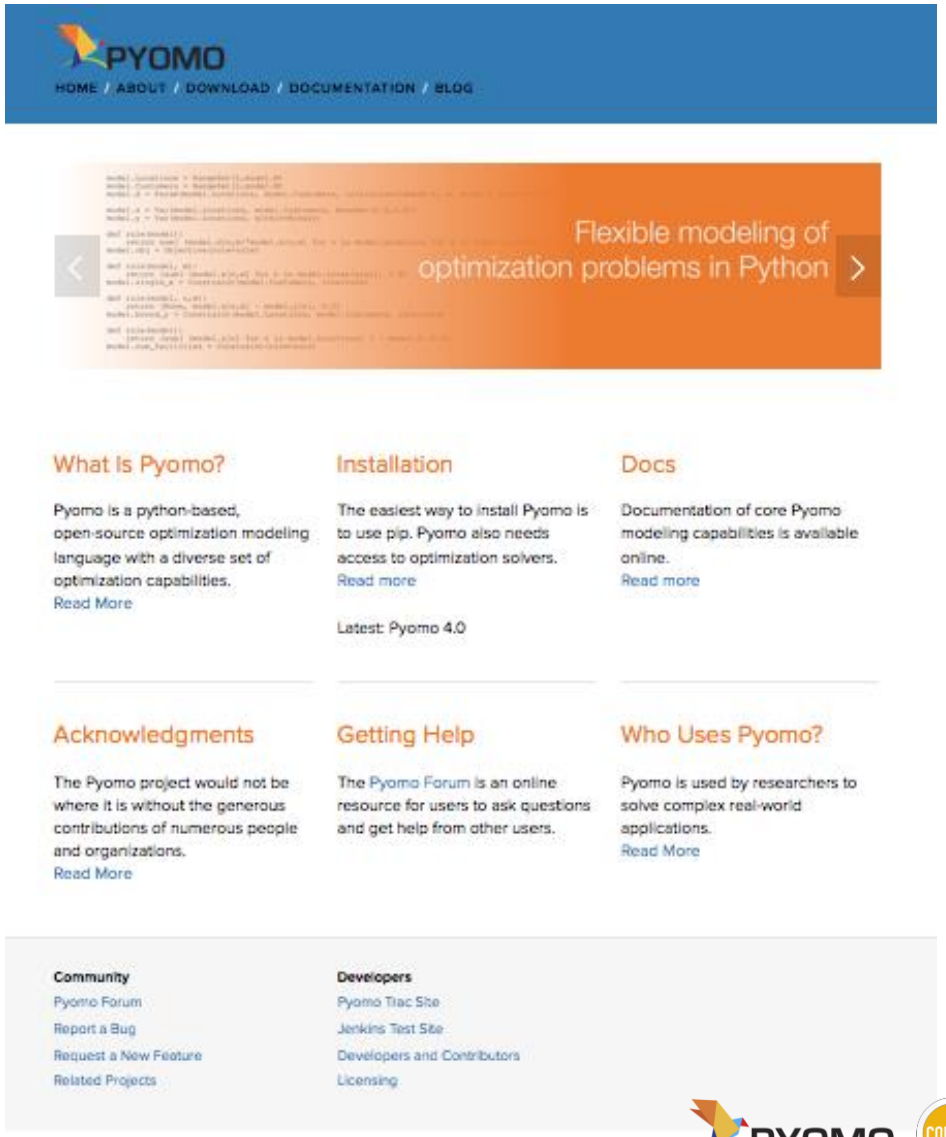
- www.pyomo.org

The Pyomo homepage provides a portal for:

- Online documentation
- Installation instructions
- Help information
- Developer links

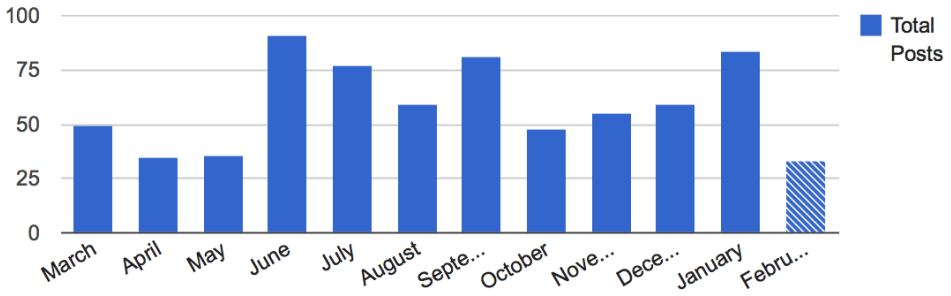
Coming soon:

- A gallery of simple examples

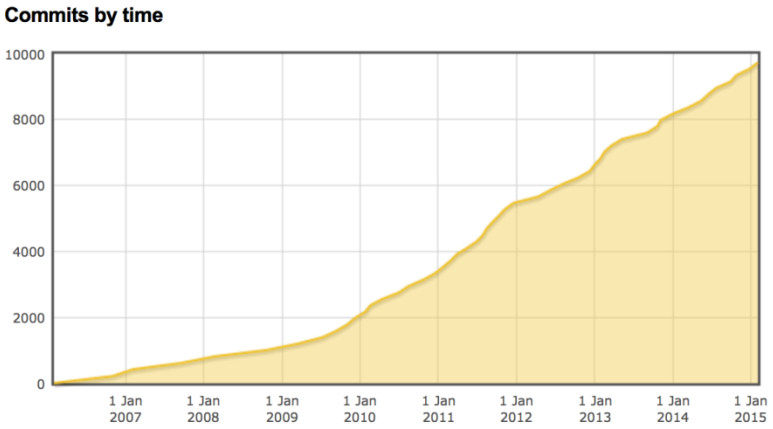


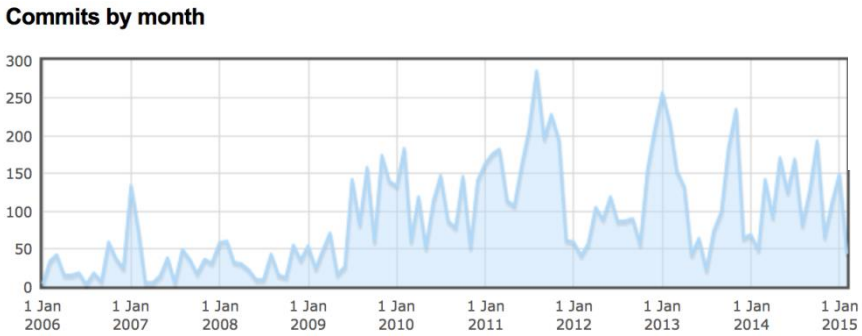
Development, Community Activity

- Pyomo Forum
  - Active discussion list



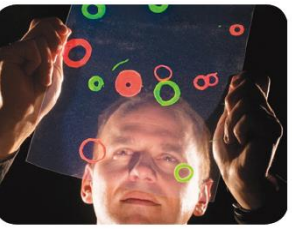
- Active developer community

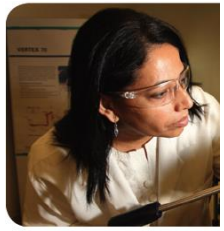




Acknowledgements

- Sandia National Laboratories
  - William Hart
  - Jean-Paul Watson
  - John Siirola
  - Francisco Munoz
- University of California, Davis
  - Prof. David L. Woodruff
  - Prof. Roger Wets
- Purdue University
  - Prof. Carl D. Laird
- Oregon State University
  - Gabe Hackebeil
- Carnegie Mellon University
  - Bethany Nicholson





Sandia National Laboratories

Exceptional service in the national interest

2. A Python Tutorial

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Why Python?

Sandia National Laboratories

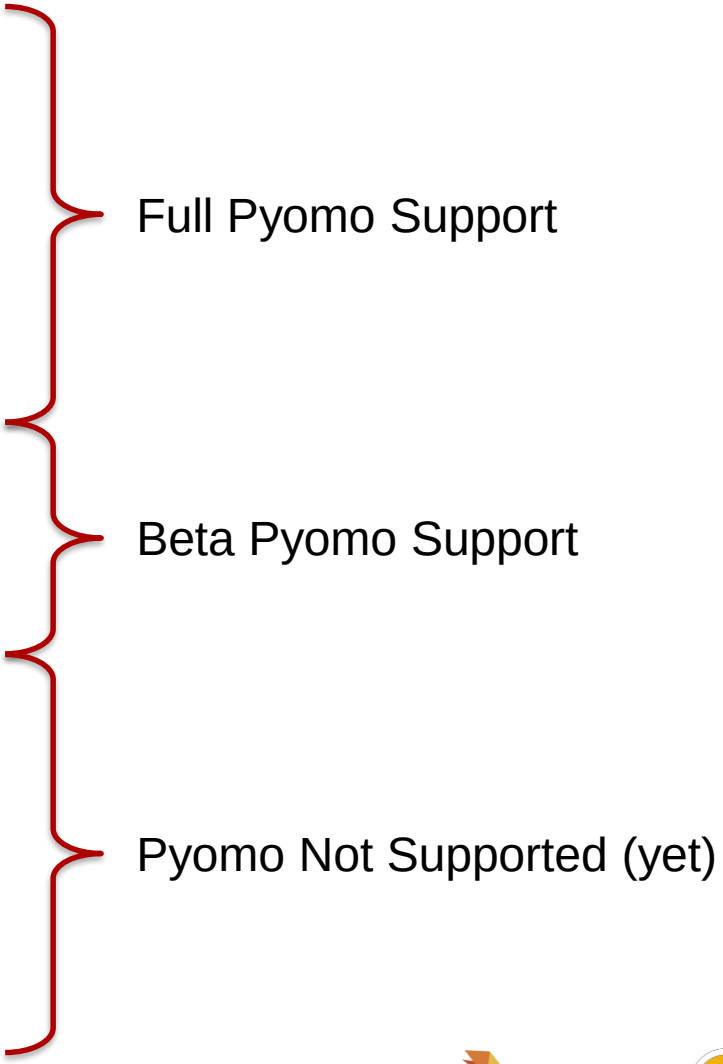
- Interpreted language
- Intuitive syntax
- Dynamic typing
- Lots of built-in libraries and third-party extensions
- Shallow learning curve
- Integration with C/Java
- Object-oriented
- Simple, but extremely powerful

CCR Center for Computing Research

PYOMO COIN-OR

Python Implementations

- Cpython
  - C Python interpreter
  - https://www.python.org/downloads/
  - SciPy Stack
    - http://www.scipy.org/install.html
    - Anaconda: Linux/MacOS/MS Windows
- PyPy
  - A Python interpreter written in Python
  - http://pypy.org/
- Jython
  - Java Python interpreter
  - http://www.jython.org/
- IronPython
  - .NET Python interpreter
  - http://ironpython.net/



Python Versions: 2.x *vs* 3.x

- Python 3.0 was released in 2008
  - Included significant backward incompatibilities
- Adoption of Python 3.x has been slow
  - Major Linux distributions are still including Python 2.x
  - Major Python packages have slowly transitioned
  - Some commercial packages still only have Python 2.x interfaces
- Status
  - Python 2.7.11
    - Very stable; patches have included package updates to support Python 3.x compatibility
  - Python 3.5.1
    - Very stable

We try to stick to "universal" syntax that will work in both 2.x and 3.x

Overview

- interactive "shell"
- basic types: numbers, strings
- container types: lists, dictionaries, tuples
- variables
- control structures
- functions & procedures
- classes & instances
- modules
- exceptions
- files & standard library

Interactive Shell

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules
- Two variations:
  - IDLE (GUI)
  - python (command line)
- Type statements or expressions at prompt:

```
>>> print( "Hello, world" )
Hello, world
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```

Python Program

- To write a program, put commands in a file

```
# hello.py
print( "Hello, world" )
x = 12**2
print( x )
```

- Execute on the command line

```
C:\Users\me> python hello.py
Hello, world
144
```

Python Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

```
greeting = 34.2
if friendly:
    greeting = "hello world"
else:
    greeting = 12**2
print( greeting )
```

- ***Everything*** is a "variable":
  - Even functions, classes, modules

Control Structures

```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements
```

```
while condition:
    statements

for var in sequence:
    statements

break
continue
```

Note: *Spacing matters!*
Control structure scope dictated by indentation

Grouping Indentation

Sandia National Laboratories

In Python:

```
for i in range(20):
    if i % 3 == 0:
        print(i)
        if i % 5 == 0:
            print("Bingo!“)
    print("---")
```

In C:

```
for (i = 0; i < 20; i++)
{
   if (i % 3 == 0) {
      printf("%d\n", i);
      if (i % 5 == 0) {
         printf("Bingo!\n");
      }
   }
   printf("---\n");
}
```

Numbers

- The usual suspects
  - `12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), 0<x<=5`
- C-style shifting & masking
  - `1<<16, x&0xff, x|1, ~x, x^y`
- Integer division truncates
  - Python 2.x
    - `1/2 → 0, 1./2. → 0.5, float(1)/2 → 0.5`
    - `from __future__ import division`
      - `» 1/2 → 0.5`
  - Python 3.x
    - `1/2 → 0.5`
- Long (arbitrary precision), complex
  - `2L**100 → 1267650600228229401496703205376L`
    - In Python 2.2 and beyond, 2**100 does the same thing
  - `1j**2 → (-1+0j)`

Strings

- `"hello"+"world”` `"helloworld”` `# concatenation`
- `"hello"*3` `"hellohellohello"` `# repetition`
- `"hello"[0]` `"h"` `# indexing`
- `"hello"[-1]` `"o"` `# (from end)`
- `"hello"[1:4]` `"ell"` `# slicing`
- `len("hello")` `5` `# size`
- `"hello" < "jello”` `True` `# comparison`
- `"e" in "hello”` `True` `# search`

- `"escapes: \n etc, \033 etc, \if etc"`
- `'single quotes'  """triple quotes"""  r"raw strings"`

Lists

- Flexible arrays, *not* linked lists
  - `a = [99, "bottles of beer", ["on", "the", "wall"]]`
- Same operators as for strings
  - `a+b, a*3, a[0], a[-1], a[1:], len(a)`
- Item and slice assignment
  - `a[0] = 98`
  - `a[1:2] = ["bottles", "of", "beer"]`

    ```
    # -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
    ```
  - `del a[-1]`

    ```
    # -> [98, "bottles", "of", "beer"]
    ```

List Operations

```
>>> a = range(5)                # [0,1,2,3,4]
>>> a.append(5)                 # [0,1,2,3,4,5]
>>> a.pop()                     # [0,1,2,3,4]
5
>>> a.insert(0, 42)             # [42,0,1,2,3,4]
>>> a.pop(0)                    # [0,1,2,3,4]
42
>>> a.reverse()                 # [4,3,2,1,0]
>>> a.sort()                    # [0,1,2,3,4]
```

Dictionaries

- Hash tables, "associative arrays"
  - `d = {"duck": "eend", "water": "water"}`
- Lookup:
  - `d["duck"]  # -> "eend"`
  - `d["back"]  # raises KeyError exception`
- Delete, insert, overwrite:
  - `del d["water"]  # {"duck": "eend", "back": "rug"}`
  - `d["back"] = "rug"  # {"duck": "eend", "back": "rug"}`
  - `d["duck"] = "duik"  # {"duck": "duik", "back": "rug"}`

Dictionary Operations

- Keys, values, items:
  - `d.keys()  -> ["duck", "back"]`
  - `d.values()  -> ["duik", "rug"]`
  - `d.items()  -> [("duck","duik"), ("back","rug")]`

Note: These actually return generators, not lists.

- Presence check:
  - `d.has_key("duck")  # -> 1; d.has_key("spam") -> 0`

- Values of any type; keys almost any

```
{ "name": "Guido",
  "age": 43,
  ("hello","world"): 1,
  42: "yes",
  "flag": ["red","white","blue"] }
```

Dictionary Details

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - keys are hashed (to ensure fast lookup)
  - lists or dictionaries cannot be used as keys
    - these objects can be changed "in place"
  - no restrictions on values

- Keys will be listed in **arbitrary order**
  - key hash values are in an arbitrary order
  - that numeric keys are returned sorted is an artifact of the implementation and *is not guaranteed*

Tuples

- `key = (lastname, firstname)`
- `point = x, y, z` `# parentheses optional`
- `x, y, z = point` `# unpack`
- `lastname = key[0]` `# index tuple values`
- `singleton = (1,)` `# trailing comma!!!`
  `#  (1) → integer!`
- `empty = ()` `# parentheses!`

- Tuples vs. lists
  - tuples immutable
  - lists mutable

Reference Semantics

- Assignment manipulates references
  - `x = y` **does not make a copy** of y
  - `x = y` makes x **reference** the object y references
- Reference values can be modified!

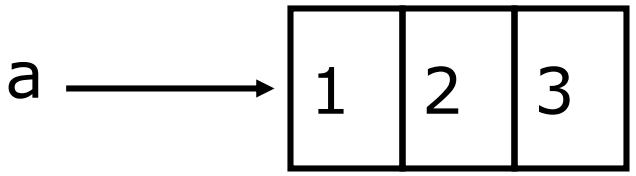
```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print(b)
[1, 2, 3, 4]
```

- Copied objects are distinct

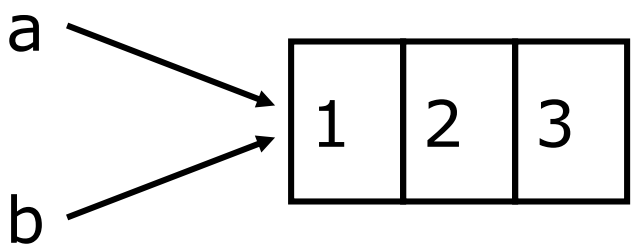
```
>>> import copy
>>> c = copy.copy(a)
>>> a.pop()
>>> print(c)
[1, 2, 3, 4]
```

Changing a Shared List

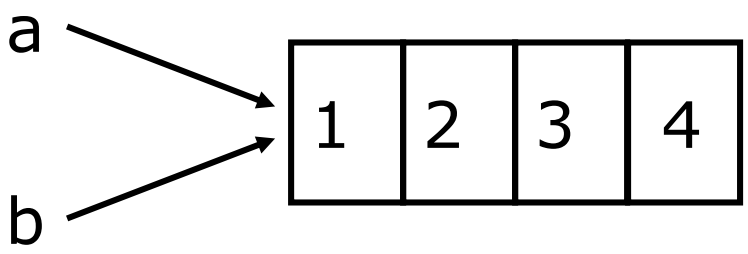
```
a = [1, 2, 3]
```




```
b = a
```

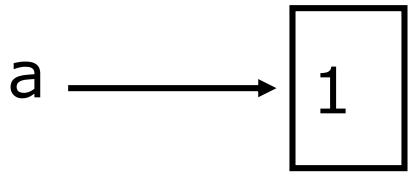



```
a.append(4)
```

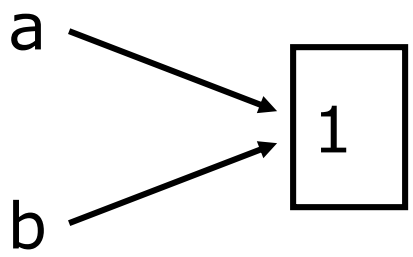


Changing an Integer

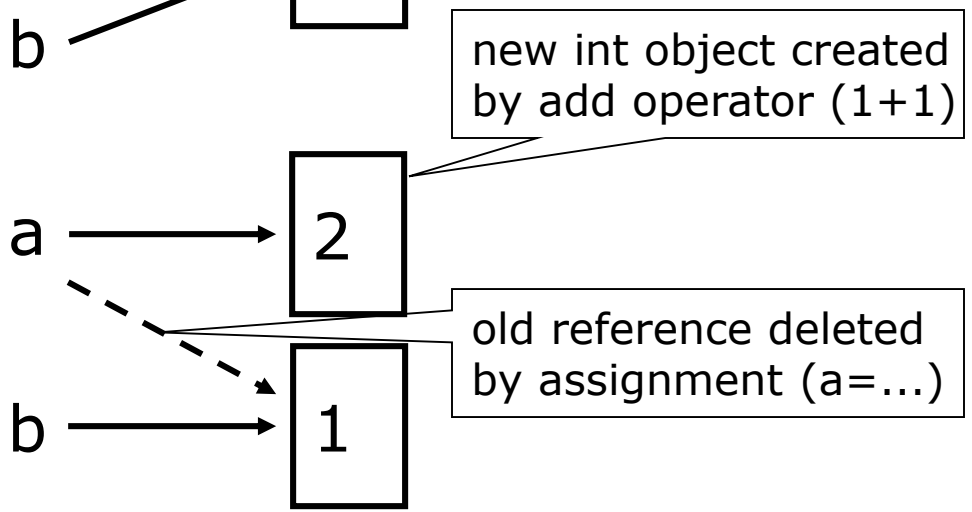
```
a = 1
```




```
b = a
```




```
a = a+1
```



Functions / Procedures

```
def name(arg1, arg2, ...):
    """documentation"""    # optional doc string
    statements

    return expression      # from function
    return                 # from procedure (returns None)
```

Example

```
def gcd(a, b):
    """greatest common divisor"""
    while a != 0:
        a, b = b%a, a   # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'

>>> gcd(12, 20)
4
```

Classes

```
class name(object):
    """documentation"""
    statements
```

Most, *statements* are method definitions:

```
    def name(self, arg1, arg2, ...):
        ...
```

May also be *class variable* assignments

Example

```
class Stack(object):
    """A well-known data structure..."""
    def __init__(self):             # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)        # the sky is the limit
    def pop(self):
        x = self.items[-1]          # what if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0     # Boolean result
```

Example (cont'd)

Sandia National Laboratories

- To create an instance, simply call the class object:

```
x = Stack()              # no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.empty()                # -> 1
x.push(1)                # [1]
x.empty()                # -> 0
x.push("hello")          # [1, "hello"]
x.pop()                  # -> "hello"   # [1]
```

- To inspect instance variables, use dot notation:

```
x.items                  # -> [1]
```

Class/Instance Variables

```
class Connection(object):
    verbose = 0                 # class variable

    def __init__(self, host):
        self.host = host        # instance variable

    def debug(self, v):
        self.verbose = v        # make instance variable!

    def connect(self):
        if self.verbose:        # class or instance variable?
            print("connecting to %s" % (self.host,))
```

Instance Variable Rules

- On use via instance (`self.x`), search order:
  - (1) instance, (2) class, (3) base classes
  - this also works for method lookup
- On assignment via instance (`self.x = ...`):
  - always makes an instance variable
- Class variables "default" for instance variables
- But...!
  - mutable *class* variable: one copy *shared* by all
  - mutable *instance* variable: each instance its own

Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables

- Importing modules:

```
import re
print( re.match("[a-z]+", s) )
from re import match
print( match("[a-z]+", s) )
```

- Import with rename:

```
import re as regex
from re import match as m
```

Catching Exceptions

```
def foo(x):
    return 1/x

def bar(x):
    try:
        print( foo(x) )
    except ZeroDivisionError as message:
        print("Can't divide by zero: %s" % message)

bar(0)
```

Try-Finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close()           # always executed
print("OK")             # executed on success only
```

Raising Exceptions

```
raise IndexError

raise IndexError("k out of range")

raise IndexError, "k out of range”
            # this only works in Python 2.x!

try:
    something
except:                          # catch everything
    print( "Oops" )
    raise                        # reraise
```

More on Exceptions

- User-defined exceptions
  - subclass `Exception` or any other standard exception
- Note: in older versions of Python exceptions can be strings
- Last caught exception info:
  - `sys.exc_info() == (exc_type, exc_value, exc_traceback)`
- Printing exceptions: traceback module

Major Python Packages

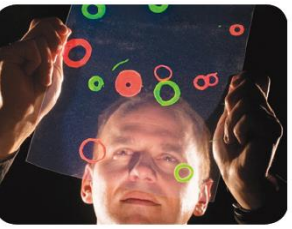
- SciPy
  - Scientific Python for mathematics and engineering
  - http://www.scipy.org
- Numpy
  - Numeric array package
  - http://www.numpy.org/
- Matplotlib
  - 2D plotting library
  - http://matplotlib.org/
- Pandas
  - Data structures and analysis
  - http://pandas.pydata.org/
- Ipython
  - Interactive Python shell
  - http://ipython.org/

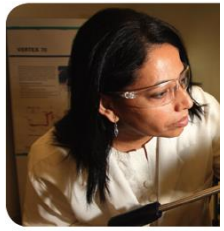
Resources

- Software Carpentry
  - http://software-carpentry.org/
- Python webpage
  - http://www.python.org
- Books
  - Python Essential Reference (4th Edition), David Beazley, 2009
  - Python in a Nutshell, Alex Martelli, 2003
  - Python Pocket Reference, 4th Edition, Mark Lutz, 2009
  - …

Acknowledgements

- William Hart
- Ted Ralphs
- John Siirola
- Dave Woodruff
- Guido van Rossum





Sandia National Laboratories

Exceptional service in the national interest

3. Pyomo Fundamentals

PYOMO

U.S. DEPARTMENT OF ENERGY

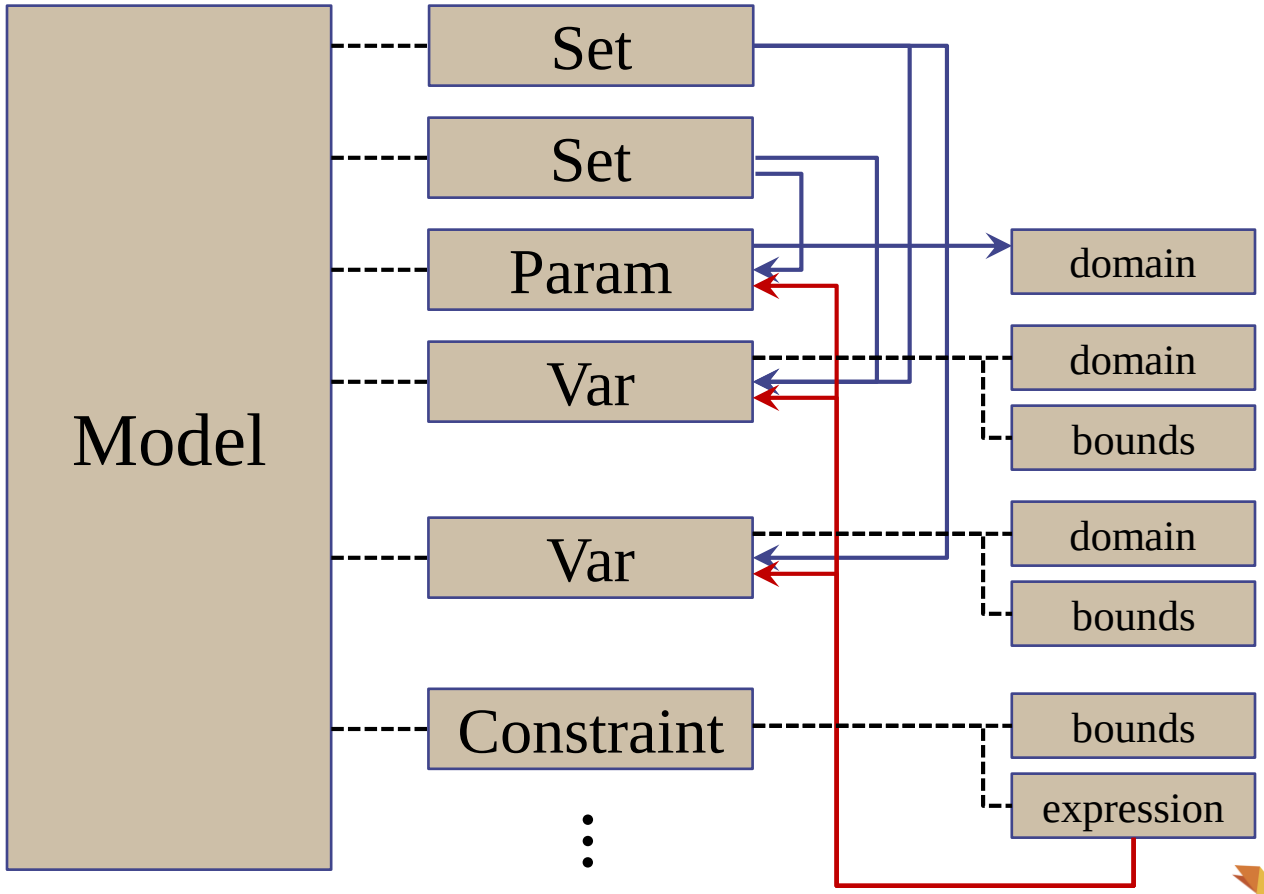
NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

3. Fundamental Pyomo Components

- Pyomo is an *object model* for describing optimization problems
  - The fundamental objects used to build models are *Components*



Cutting to the chase: a simple Pyomo model

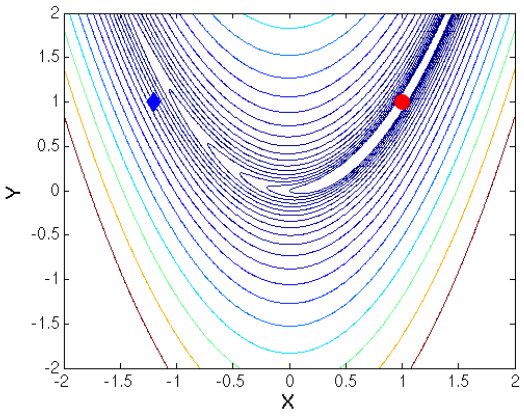
- rosenbrock.py:

```
from pyomo.environ import *

model = ConcreteModel()

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

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



Cutting to the chase: a simple Pyomo model

- Solve the model:
  - The pyomo command

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

========================================================
Solution Summary
========================================================

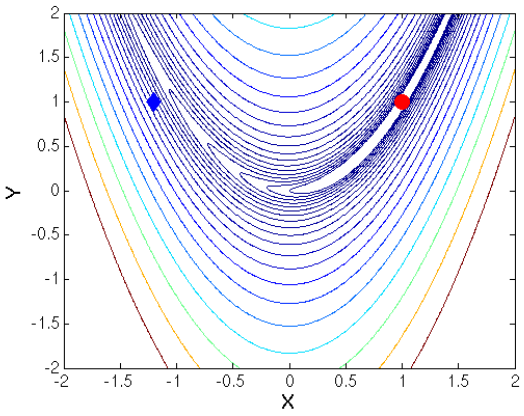
Model unknown

  Variables:
    Variable x : Size=1 Domain=Reals
      Value=0.999999994543
    Variable y : Size=1 Domain=Reals
      Value=0.999999989052

  Objectives:
    Objective obj : Size=1
      Value=2.98956421871e-17

  Constraints:
    None

[    0.03] Applying Pyomo postprocessing actions
[    0.03] Pyomo Finished
```



Regarding *namespaces*

- Pyomo objects exist within the `pyomo.environ` namespace:

```
import pyomo.environ
model = pyomo.environ.ConcreteModel()
```

- …but this gets verbose. To save typing, we will import the core Pyomo classes into the main namespace:

```
from pyomo.environ import *
model = ConcreteModel()
```

- To clarify Pyomo-specific syntax in this tutorial, we will highlight Pyomo symbols in green

Getting Started: the *Model*

```
from pyomo.environ import *

model = ConcreteModel()
```

Every Pyomo model starts with this; it tells Python to load the Pyomo Modeling Environment

Create an instance of a *Concrete* model

- Concrete models are immediately constructed
- Data must be present *at the time* components are defined

Local variable to hold the model we are about to construct

- While not required, by convention we use "`model`"
- If you choose to name your model something else, you will need to tell the Pyomo script the object name through the command line

Populating the Model: *Variables*

```
model.a_variable = Var(within = NonNegativeReals)
```

The name you assign the object to becomes the object's name, and must be unique in any given model.

"`within`" is optional and sets the variable domain ("`domain`" is an alias for "`within`")

Several pre-defined domains, e.g., "`Binary`"

```
model.a_variable = Var(bounds = (0, None))
```

Same as above: "`domain`" is assumed to be `Reals` if missing

Defining the *Objective*

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

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

If "sense" is omitted, Pyomo assumes minimization

Note that the Objective expression is not a *relational expression*

"expr" can be an expression, or any function-like object that returns an expression

Defining the Problem: *Constraints*

```
model.a = Var()
model.b = Var()
model.c = Var()
model.c1 = Constraint(
     expr = model.b + 5 * model.c <= model.a )
```

"expr" can be an expression, or any function-like object that returns an expression

```
model.c2 = Constraint(expr = (None, model.a + model.b, 1))
```

"expr" can also be a tuple:

- 3-tuple specifies (LB, expr, UB)
- 2-tuple specifies an equality constraint.

In general, we do not recommend this notation

Lists of Constraints

```
model.a = Var()
model.b = Var()
model.c = Var()
model.limits = ConstraintList()

model.limits.add(30*model.a + 15*model.b + 10*model.c <= 100)
model.limits.add(10*model.a + 25*model.b +  5*model.c <= 75)
model.limits.add(6*model.a  + 11*model.b +  3*model.c <= 30)
```

"add" adds a single new constraint to the list.
The constraints need not be related.

Higher-dimensional components

- (Almost) All Pyomo *components* can be *indexed*
  - *All* non-keyword arguments are assumed to be *indices*
  - Individual indices may be multi-dimensional (e.g., a list of pairs)

```
<Type>(<IDX1>, <IDX2>, […] <keyword>=<value>, …)
```

- Indexed variables

```
model.a_vector = Var(IDX)
model.a_matrix = Var(IDX_A, IDX_B)
```

The indexes are any iteratable object, e.g., list or Set

- ConstraintList is a special case with an implicit index
- *Note:* while indexed variables look like matrices, they are not.
  - In particular, we do not support matrix algebra (*yet…)*

Manipulating indices: list comprehensions

```
model.IDX = range(10)
model.a = Var()
model.b = Var(model.IDX)
model.c1 = Constraint(
    expr = sum(model.b[i] for i in model.IDX) <= model.a )
```

Python *list comprehensions* are very common for working over indexed variables and nicely parallel mathematical notation:

$$\sum_{i \in IDX} b_i \leq a$$

Concrete Modeling

Putting It All Together: *Concrete p-Median*

- Determine the set of P warehouses chosen from N candidates that minimizes the total cost of serving all customers M where $d_{n,m}$ is the cost of serving customer m from warehouse location n.

$$\min \sum_{n \in N, m \in M} d_{n,m} x_{n,m} \qquad \text{(minimize total cost)}$$

$$s.t. \quad \sum_{n} x_{n,m} = 1 \qquad \forall\ m \in M \qquad \text{(guarantee all customers served)}$$

$$x_{n,m} \leq y_n \qquad \forall\ n \in N, m \in M \qquad \text{(customer } n \text{ can only be served from warehouse } m \text{ if warehouse } m \text{ is selected)}$$

$$\sum_{n \in N} y_n = P \qquad \text{(select } P \text{ warehouses)}$$

$$0 \leq x \leq 1 \qquad y \in \{0,1\}^N$$

Concrete p-Median (1)

```
from pyomo.environ import *

N = 3
M = 4
P = 3
d = {(1, 1): 1.7, (1, 2): 7.2, (1, 3): 9.0, (1, 4): 8.3,
     (2, 1): 2.9, (2, 2): 6.3, (2, 3): 9.8, (2, 4): 0.7,
     (3, 1): 4.5, (3, 2): 4.8, (3, 3): 4.2, (3, 4): 9.3}

model = ConcreteModel()
model.Locations = range(N)
model.Customers = range(M)

model.x = Var( model.Locations, model.Customers,
               bounds=(0.0,1.0) )

model.y = Var( model.Locations, within=Binary )
```

Concrete p-Median (2)

```
model.obj = Objective( expr = sum( d[n,m]*model.x[n,m]
    for n in model.Locations for m in model.Customers ) )

model.single_x = ConstraintList()
for m in model.Customers:
    model.single_x.add(
        sum( model.x[n,m] for n in model.Locations ) == 1.0 )

model.bound_y = ConstraintList()
for n in model.Locations:
    for m in model.Customers:
        model.bound_y.add( model.x[n,m] <= model.y[n] )

model.num_facilities = Constraint(
    expr=sum( model.y[n] for n in model.Locations ) == P )
```

Solving models: *the* pyomo *command*

- pyomo (`pyomo.exe` on Windows):
  - Constructs model and passes it to an (external) solver

```
pyomo solve <model_file> [<data_file> …] [options]
```

  - Installed to:
    - `[PYTHONHOME]\Scripts` [Windows; `C:\Python27\Scripts`]
    - `[PYTHONHOME]/bin` [Linux; `/usr/bin`]

- Key options (*many* others; see `--help`)

`--help`	Get list of all options
`--help-solvers`	Get the list of all recognized solvers
`--solver=<solver_name>`	Set the solver that Pyomo will invoke
`--solver-options="key=value[ …]"`	Specify options to pass to the solver as a space-separated list of keyword-value pairs
`--stream-solver`	Display the solver output during the solve
`--summary`	Display a summary of the optimization result
`--report-timing`	Report additional timing information, including construction time for each model component

In Class Exercise: Concrete Knapsack

$$\max \quad \sum_{i=1}^{N} v_i x_i$$

$$s.t. \quad \sum_{i=1}^{N} w_i x_i \leq W_{\max}$$

$$x_i \in \{0,1\}$$

Item	Weight	Value
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11
Max weight:	14	

Syntax reminders:

```
from pyomo.environ import *
ConcreteModel()
Var( [index, …], [within=domain], [bounds=(lower,upper)] )
ConstraintList()
    c.add( expression )
Objective( sense={maximize|minimize},
           expr=expression )
```

Concrete Knapsack: *Solution*

```
from pyomo.environ import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model = ConcreteModel()
model.ITEMS = v.keys()
model.x = Var( model.ITEMS, within=Binary )

model.value = Objective(
    expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
    sense = maximize )

model.weight = Constraint(
    expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Abstract Modeling

Concrete *vs.* Abstract Models

- Concrete Models: data first, then model
  - 1-pass construction
  - All data must be present before Python starts processing the model
  - Pyomo will construct each component in order at the time it is declared
  - Straightforward logical process; easy to script.
  - Familiar to modelers with experience with GAMS
- Abstract Models: model first, then data
  - 2-pass construction
  - Pyomo stores the basic model declarations, but does not construct the actual objects
    - Details on how to construct the component hidden in functions, or *rules*
    - e.g., it will declare an indexed variable "x", but will not expand the indices or populate any of the individual variable values.
  - At "creation time", data is applied to the abstract declaration to create a concrete instance (components are still constructed in declaration order)
  - Encourages generic modeling and model reuse
    - e.g., model can be used for arbitrary-sized inputs
  - Familiar to modelers with experience with AMPL

Generating and Managing Indices: *Sets*

Sandia National Laboratories

- Any iterable object can be an index, e.g., lists:
  - `IDX_a = [1,2,5]`
  - ```
    DATA = {1: 10, 2: 21, 5:42};
    IDX_b = DATA.keys()
    ```

- Sets: objects for managing multidimensional indices
  - `model.IDX = Set( initialize = [1,2,5] )`

Note: capitalization matters:
Set = Pyomo class
set = native Python set

Like indices, Sets can be initialized from any iterable

  - `model.IDX = Set( [1,2,5] )`

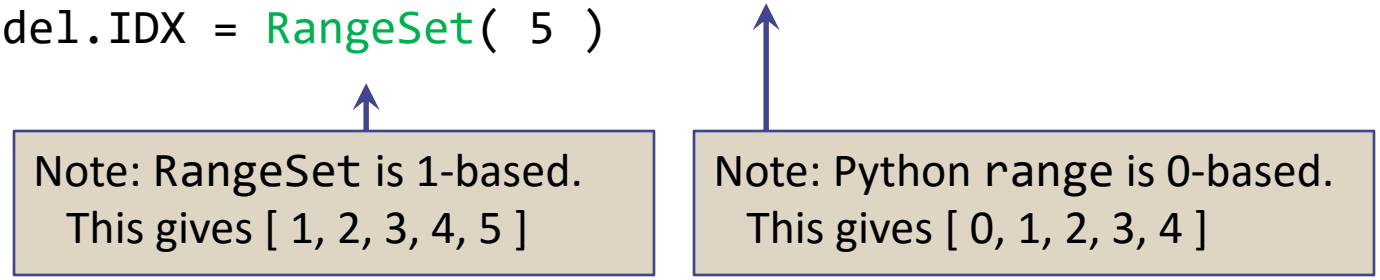
Note: This doesn't do what you want.
This creates a 3-member *indexed set*, where each set is *empty.*

CCR Center for Computing Research

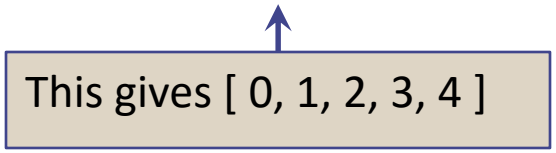
PYOMO COIN-OR
    ```

Sequential Indices: *RangeSet*

- Sets of sequential integers are common
  - `model.IDX = Set( initialize=range(5) )`
  - `model.IDX = RangeSet( 5 )`



- You can provide lower and upper bounds to RangeSet
  - `model.IDX = RangeSet( 0, 4 )`



Manipulating Sets

- Sets support efficient higher-dimensional indices

```
model.IDX = Set( initialize=[1,2,5] )
model.IDX2 = model.IDX * model.IDX
```

This creates a *virtual* 2-D "matrix" Set

Sets also support union (&), intersection (|), difference (-), symmetric difference (^)

- Creating sparse sets

```
model.IDX = Set( initialize=[1,2,5] )
def lower_tri_filter(model, i, j):
    return j <= i
model.LTRI = Set( initialize = model.IDX * model.IDX,
                  filter = lower_tri_filter )
```

The `filter` should return *True* if the element is in the set; *False* otherwise.

Deferred construction: *Rules*

- Abstract modeling constructs the model in two passes:
  - Python parses the model declaration
    - creating "empty" Pyomo components in the model
  - Pyomo loads and parses external data
- Components are constructed in declaration order
  - The instructions for *how* to construct the object are provided through a function, or *rule*
  - Pyomo calls the rule for each component index
  - *Rules* can be provided to virtually all Pyomo components (even when using Concrete models)
- Naming conventions
  - the component name prepended with "_" (`c4` → `_c4`)
  - the component name with "`_rule`" appended (`c4` → `c4_rule`)
  - each rule is called "`rule`" (Python implicitly overrides each declaration)

Indexed Constraints

```
model.IDX = Set( initialize=range(5) )
model.a = Var( model.IDX )
model.b = Var()
def c4_rule(model, i):
    return model.a[i] + model.b <= 1
model.c4 = Constraint( model.IDX, rule=c4_rule )
```

For indexed constraints, you provide a "rule" (function) that returns an expression (or tuple) for each index.

```
model.IDX2 = model.IDX * model.IDX
def c5_rule(model, i, j, k):
    return model.a[i] + model.a[j] + model.a[k] <= 1
model.c5 = Constraint( model.IDX2, model.IDX, rule=c5_rule )
```

Each dimension of each index is a separate argument to the rule

Importing Data: *Parameters*

- Scalar numeric values

```
model.a_parameter = Param( initialize = 42 )
```

Provide an (initial) value of 42 for the parameter

- Indexed numeric values

```
model.a_param_vec = Param( IDX,
                           initialize = data,
                           default = 0 )
```

Providing "`default`" allows the initialization data to only specify the "unusual" values

"`data`" *must* be a dictionary(*) of index keys to values because all sets are assumed to be *unordered*

(*) – actually, it must define __getitem__(), but that only really matters to Python geeks

Data Sources

- Data can be imported from "`.dat`" file
  - Format similar to AMPL style
  - Explicit data from "`param`" declarations
  - External data through "`load`" declarations:
    - Excel

```
load ABCD.xls range=ABCD : Z=[A, B, C] Y=D ;
```

    - Databases

```
load "DBQ=diet.mdb" using=pyodbc query="SELECT FOOD, cost,
  f_min, f_max from Food" : [FOOD] cost f_min f_max ;
```

- External data overrides "`initialize=`" declarations

Abstract p-Median (pmedian.py, 1)

```
from pyomo.environ import *

model = AbstractModel()

model.N = Param( within=PositiveIntegers )
model.P = Param( within=RangeSet( model.N ) )
model.M = Param( within=PositiveIntegers )

model.Locations = RangeSet( model.N )
model.Customers = RangeSet( model.M )

model.d = Param( model.Locations, model.Customers )

model.x = Var( model.Locations, model.Customers, bounds=(0.0, 1.0) )
model.y = Var( model.Locations, within=Binary )
```

Abstract p-Median (pmedian.py, 2)

```
def obj_rule(model):
    return sum( model.d[n,m]*model.x[n,m]
       for n in model.Locations for m in model.Customers )
model.obj = Objective( rule=obj_rule )

def single_x_rule(model, m):
    return sum( model.x[n,m] for n in model.Locations ) == 1.0
model.single_x = Constraint( model.Customers, rule=single_x_rule )

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

def num_facilities_rule(model):
    return sum( model.y[n] for n in model.Locations ) == model.P
model.num_facilities = Constraint( rule=num_facilities_rule )
```

Abstract p-Median (pmedian.dat)

```
param N := 3;

param M := 4;

param P := 2;

param d:  1     2     3     4 :=
     1   1.7   7.2   9.0   8.3
     2   2.9   6.3   9.8   0.7
     3   4.5   4.8   4.2   9.3 ;
```

In Class Exercise: Abstract Knapsack

$$\max \quad \sum_{i=1}^{N} v_i x_i$$

$$s.t. \quad \sum_{i=1}^{N} w_i x_i \le W_{\max}$$

$$x_i \in \{0,1\}$$

Item	Weight	Value
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11
Max weight:	14	

Syntax reminders:

```
AbstractModel()
Set( [index, …], [initialize=list/function] )
Param( [index, …], [within=domain], [initialize=dict/function] )
Var( [index, …], [within=domain], [bounds=(lower,upper)] )
Constraint( [index, …], [expr=expression|rule=function] )
Objective( sense={maximize|minimize},
           expr=expression|rule=function )
```

Abstract Knapsack: *Solution*

```
from pyomo.environ import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( rule=value_rule, sense=maximize )

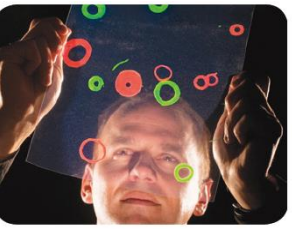
def weight_rule(model):
    return sum( model.w[i]*model.x[i] for i in model.ITEMS ) \
        <= model.W_max
model.weight = Constraint( rule=weight_rule )
```

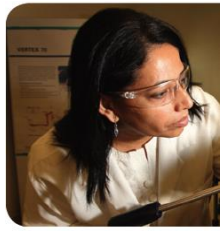
Abstract Knapsack: *Solution Data*

```
set ITEMS := hammer wrench screwdriver towel ;

param:  v w :=
    hammer        8 5
    wrench        3 7
    screwdriver   6 4
    towel        11 3;

param W_max := 14;
```





Sandia National Laboratories

Exceptional service in the national interest

4. Pyomo Idioms

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Pyomo and Python: Idioms and Efficiency

Sandia National Laboratories

- Being embedded in a high-level (and interpreted) programming language can present challenges
  - Inability to constrain syntax => users have many guns
  - Some of approaches may be *very* slow
- Some of the blame can be placed on Python
  - But a lot can be blamed on Pyomo

What are *reasonable* performance expectations?

- Python is a *byte-compiled scripting language*
  - and Pyomo is pure Python
  - …so expectations were not high
  - *…and raw speed has never been a goal!*
- Early experiences bore this out… in November, 2010:
  - *p*-median facility location
    - AMPL model construction time: ~4 seconds
    - Pyomo model construction time: >2000 seconds
  - Logistics disruption modeling
    - GAMS solution time: ~20 seconds
    - Pyomo solution time: >200 seconds
- ...but the gap is closing… in Coopr 3.2:
  - *p*-median facility location: ~45 seconds
  - Logistics disruption modeling: ~25 seconds

Managing edge cases (special sets *vs* rule logic)

- Linking time; consider:

$$T \in [0..T_{\max}]$$
$$x_t = cx_{t-1} \qquad \{t \mid t \in T, t \neq 0\}$$

Managing edge cases (special sets *vs* rule logic)

- Linking time; consider:

$$T \in [0..T_{\max}]$$
$$x_t = cx_{t-1} \qquad \{t \mid t \in T, t \neq 0\}$$

```
model.T = RangeSet(0, model.Tmax)

model.Tnot0 = model.T - [ 0 ]
def rule1(model, t):
      return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.Tnot0, rule=rule1)
```

Managing edge cases (special sets *vs* rule logic)

- Linking time; consider:

$$T \in [0..T_{\max}]$$

$$x_t = cx_{t-1} \qquad \{t \mid t \in T, t \neq 0\}$$

```
model.T = RangeSet(0, model.Tmax)
```

```
def rule2(model, t):
    if t == 0:
        return Constraint.Skip
      return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.T, rule=rule2)
```

Managing edge cases (special sets *vs* rule logic)

- Linking time; consider:

$$T \in [0..T_{\max}]$$
$$x_t = cx_{t-1} \qquad \{t \mid t \in T, t \neq 0\}$$

```
model.T = RangeSet(0, model.Tmax)

model.Tnot0 = model.T - [ 0 ]
def rule1(model, t):
      return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.Tnot0, rule=rule1)


def rule2(model, t):
    if t == 0:
        return Constraint.Skip
    return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.T, rule=rule2)
```

Managing edge cases (special sets *vs* rule logic)

- Linking time; consider:

$$T \in [0..T_{\max}]$$
$$x_t = cx_{t-1} \qquad \{t \mid t \in T, t \neq 0\}$$

```
model.T = RangeSet(0, model.Tmax)

model.Tnot0 = model.T - [ 0 ]
def rule1(model, t):
      return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.Tnot0, rule=rule1)


def rule2(model, t):
    if t == 0:
        return Constraint.Skip
    return model.x[t] == model.c * model.x[t-1]
model.link = Constraint( model.T, rule=rule2)
```

[Tmax = 1e5]

rule1: 4.35 sec

rule2: 4.10 sec

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
def rule1(model):
      ans = 0
      for n in model.Locations:
          for m in model.Customers:
              ans = ans + model.d[n,m]*model.x[n,m]
      return ans
```

```
model.obj = Objective( rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
def rule2(model):
      ans = 0
      for n in model.Locations:
          for m in model.Customers:
              ans += model.d[n,m]*model.x[n,m]
      return ans
```

```
model.obj = Objective( rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
def rule3(model):
      return sum( [ model.d[n,m]*model.x[n,m]
           for n in model.Locations for m in model.Customers ] )
```

```
model.obj = Objective( rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
def rule4(model):
      return sum( model.d[n,m]*model.x[n,m]
                  for n in model.Locations for m in model.Customers )

model.obj = Objective( rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Expression generation; consider:

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
def rule1(model):
      ans = 0
      for n in model.Locations:
          for m in model.Customers:
              ans = ans + model.d[n,m]*model.x[n,m]
      return ans

def rule2(model):
      ans = 0
      for n in model.Locations:
          for m in model.Customers:
              ans += model.d[n,m]*model.x[n,m]
      return ans

def rule3(model):
      return sum( [ model.d[n,m]*model.x[n,m]
           for n in model.Locations for m in model.Customers ] )

def rule4(model):
      return sum( model.d[n,m]*model.x[n,m]
                  for n in model.Locations for m in model.Customers )

model.obj = Objective( rule=ruleN )
```

[n = m = 1..640]

rule1: >>10000 sec
rule2: 5.0 sec
rule3: 7.4 sec
rule4: 5.0 sec

Managing performance (how not to shoot yourself)

- Sparse data; consider:

$$\sum_{m \in M} a_{n,m} x_m \leq b_n \quad \forall\, n \in N$$

```
model.a = Param( model.N, model.M, default=0, mutable=True )

def rule1(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M) <= model.b[n] )
```

For n = 1..10, m = 1..1e5, 4% nonzero,

1.9 seconds to generate the constraint
1e5 terms in the constraint (dense!!)

```
model.C = Constraint( model.N, rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Sparse data; consider:

$$\sum_{m \in M} a_{n,m} x_m \le b_n \quad \forall\, n \in N$$

```
model.a = Param( model.N, model.M, default=0, mutable=True )
```

```
def rule2(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m] != 0 ) <= model.b[n]
```

For n = 1..10, m = 1..1e5, 4% nonzero,

0.1 seconds *slower*, and still dense!

```
model.C = Constraint( model.N, rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Sparse data; consider:

$$\sum_{m \in M} a_{n,m} x_m \leq b_n \quad \forall\, n \in N$$

```
model.a = Param( model.N, model.M, default=0, mutable=True )
```

```
def rule3(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if value(model.a[n,m]) != 0 ) <= model.b[n]

def rule4(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m].value != 0 ) <= model.b[n]

model.C = Constraint( model.N, rule=ruleN )
```

Managing performance (how not to shoot yourself)

- Sparse data; consider:

$$\sum_{m \in M} a_{n,m} x_m \le b_n \quad \forall\, n \in N$$

```
model.a = Param( model.N, model.M, default=0, mutable=True )

def rule1(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M) <= model.b[n] )

def rule2(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m] != 0 ) <= model.b[n]

def rule3(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if value(model.a[n,m]) != 0 ) <= model.b[n]

def rule4(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m].value != 0 ) <= model.b[n]

model.C = Constraint( model.N, rule=ruleN )
```

[n = 1..10, m = 1..1e5, 4% fill]

rule1:	1.9 sec
rule2:	2.0 sec
rule3:	0.9 sec
rule4:	0.8 sec

Managing performance (how not to shoot yourself)

Sandia National Laboratories

- Mutable vs Immutable Params
  - Given a promise parameter value *will never change*, we can optimize
  - Immutable Params are now the default!

```
model.a = Param( model.N, model.M, default=0, mutable=False )

def rule1(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M) <= model.b[n] )

def rule2(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m] != 0 ) <= model.b[n]

def rule3(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if value(model.a[n,m]) != 0 ) <= model.b[n]

def rule4(model,n):
      return sum( model.a[n,m] * model.x[m] for m in model.M
                  if model.a[n,m].value != 0 ) <= model.b[n]

model.C = Constraint( model.N, rule=ruleN )
```

[n = 1..10, m = 1..1e5, 4% fill]

rule1:	1.2 sec
rule2:	0.5 sec
rule3:	0.5 sec
rule4:	*Exception!*

CCR Center for Computing Research

PYOMO COIN-OR

Managing performance (how not to shoot yourself)

- Sparse constraints; consider:

$$storage_{t,p,d} = storage_{t-1,p,d} + production_{t,p,d}$$

$$\forall\, t \in [dStart_d, dEnd_d],\ p \in Products,\ d \in Disruptions$$

Managing performance (how not to shoot yourself)

- Sparse constraints; consider:

$$storage_{t,p,d} = storage_{t-1,p,d} + production_{t,p,d}$$

$$\forall\, t \in [dStart_d, dEnd_d],\ p \in Products,\ d \in Disruptions$$

```
def rule1(model,t,d,p):
      if t < model.dStart[d] or t > model.dEnd[d]:
          return Constraint.Skip
      return model.storage[t,p,d] == model.storage[t-1,p,d] + model.production[t,p,d]
model.C1 = Constraint( model.TIME, model.DISRUPTIONS, model.PRODUCTS, rule=rule1 )
```

Managing performance (how not to shoot yourself)

- Sparse constraints; consider:

$$storage_{t,p,d} = storage_{t-1,p,d} + production_{t,p,d}$$

$$\forall\, t \in [dStart_d, dEnd_d],\ p \in Products,\ d \in Disruptions$$

```
def rule2(model,t,d,p):
      return model.storage[t,p,d] == model.storage[t-1,p,d] + model.production[t,p,d]

def _filter2(model,t,d,p):
      return t >= model.dStart[d] and t <= model.dEnd[d]
model.ACTIVE_DISRUPTIONS = Set( within=model.TIME * model.DISRUPTIONS * model.PRODUCTS,
                                filter=_filter2 )
model.C2 = Constraint( model.ACTIVE_DISRUPTIONS, rule=rule2 )
```

Managing performance (how not to shoot yourself)

- Sparse constraints; consider:

$$storage_{t,p,d} = storage_{t-1,p,d} + production_{t,p,d}$$

$$\forall\, t \in [dStart_d, dEnd_d],\ p \in Products,\ d \in Disruptions$$

```
def rule2(model,t,d,p):
      return model.storage[t,p,d] == model.storage[t-1,p,d] + model.production[t,p,d]
```

```
def _filter3(model,t,d):
      return t >= model.dStart[d] and t <= model.dEnd[d]
model.ACTIVE_DISRUPTIONS = Set( within=model.TIME * model.DISRUPTIONS, filter=_filter3 )
model.C3 = Constraint( model.ACTIVE_DISRUPTIONS, model.PRODUCTS, rule=rule2 )
```

Managing performance (how not to shoot yourself)

- Sparse constraints; consider:

$$storage_{t,p,d} = storage_{t-1,p,d} + production_{t,p,d}$$

$$\forall\, t \in [dStart_d, dEnd_d],\ p \in Products,\ d \in$$

[t = d = 1..250,
p =1..10,
t*d = 2% fill]

C1: 2.2 sec
C2: 2e-4 sec
C3: 2e-4 sec

```
def rule1(model,t,d,p):
      if t < model.dStart[d] or t > model.dEnd[d]:
          return Constraint.Skip
      return model.storage[t,p,d] == model.storage[t-1,p,d] + model.p
model.C1 = Constraint( model.TIME, model.DISRUPTIONS, model.PRODUCTS,

def rule2(model,t,d,p):
      return model.storage[t,p,d] == model.storage[t-1,p,d] + model.production[t,p,d]

def _filter2(model,t,d,p):
      return t >= model.dStart[d] and t <= model.dEnd[d]
model.ACTIVE_DISRUPTIONS = Set( within=model.TIME * model.DISRUPTIONS * model.PRODUCTS,
                                filter=_filter2 )
model.C2 = Constraint( model.ACTIVE_DISRUPTIONS, rule=rule2 )

def _filter3(model,t,d):
      return t >= model.dStart[d] and t <= model.dEnd[d]
model.ACTIVE_DISRUPTIONS = Set( within=model.TIME * model.DISRUPTIONS, filter=_filter3 )
model.C3 = Constraint( model.ACTIVE_DISRUPTIONS, model.PRODUCTS, rule=rule2 )
```

Managing performance (how not to shoot yourself)

Sandia National Laboratories

- Using Pyomo convenience functions

$$\min \sum_{n} \sum_{m} d_{n,m} x_{n.m}, \quad n \in Locations, m \in Customers$$

```
model.d1 = Param( model.Locations, model.Customers, mutable=True )

def rule1(model):
      return sum( model.d1[n,m]*model.x[n,m]
                  for n in model.Locations for m in model.Customers )

def rule2(model):
      return summation( model.d1, model.x )

model.d2 = Param( model.Locations, model.Customers, mutable=False )

def rule3(model):
      return sum( model.d2[n,m]*model.x[n,m]
                  for n in model.Locations for m in model.Customers )

def rule4(model):
      return sumation( model.d2, model.x )
```

[n = m = 1..640]

rule1:	5.0 sec
rule2:	7.9 sec
rule3:	3.5 sec
rule4:	5.6 sec

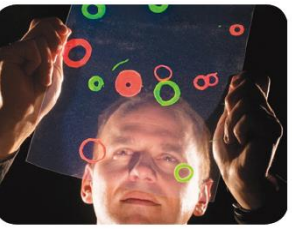
CCR Center for Computing Research

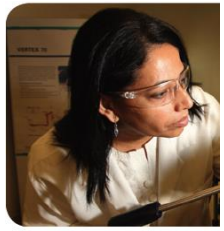
PYOMO COIN-OR

The Performance "Elephant": Memory

- Known issue …
  - Python uses a fairly heavy-weight object model
  - "Significant" recent improvements in low-level core components
    - Coopr 3.3 uses <50% of the memory of Coopr 3.0
  - But…
    - 640 x 640 *p*-median problem still consumes ~1 GB.

- Focus of efforts, with several more enhancements on the horizon.





Sandia National Laboratories

Exceptional service in the national interest

5. Nonlinear Problems

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Nonlinear problems are easy…

Nonlinear problems are easy…

… to write in Pyomo (correct formulation and solution is another story)

Nonlinear problems are easy…

… to write in Pyomo (correct formulation and solution is another story)

- Agenda:
  - Introduction
  - Rosenbrock Example
  - Introduction to Scripting
  - Introduction to IPOPT
  - Recommendations for Nonlinear Problems
  - Formulation Matters Example
  - Exercises

Nonlinear: Supported expressions

Operation	Operator	Example
multiplication	`*`	`expr = model.x * model.y`
division	`/`	`expr = model.x / model.y`
exponentiation	`**`	`expr = (model.x+2.0)**model.y`
in-place multiplication[1]	`*=`	`expr *= model.x`
in-place division[2]	`/=`	`expr /= model.x`
in-place exponentiation[3]	`**=`	`expr **= model.x`

```
model = ConcreteModel()
model.r = Var()
model.h = Var()

def surf_area_obj_rule(m):
    return 2 * pi * m.r * (m.r + m.h)
model.surf_area_obj = Objective(rule=surf_area_obj_rule)

def vol_con_rule(m):
    return pi * m.h * m.r**2 == 355
model.vol_con = Constraint(rule=vol_con_rule)
```

Nonlinear: Supported expressions

Operation	Function	Example
arccosine	`acos`	`expr = acos(model.x)`
hyperbolic arccosine	`acosh`	`expr = acosh(model.x)`
arcsine	`asin`	`expr = asin(model.x)`
hyperbolic arcsine	`asinh`	`expr = asinh(model.x)`
arctangent	`atan`	`expr = atan(model.x)`
hyperbolic arctangent	`atanh`	`expr = atanh(model.x)`
cosine	`cos`	`expr = cos(model.x)`
hyperbolic cosine	`cosh`	`expr = cosh(model.x)`
exponential	`exp`	`expr = exp(model.x)`
natural log	`log`	`expr = log(model.x)`
log base 10	`log10`	`expr = log10(model.x)`
sine	`sin`	`expr = sin(model.x)`
square root	`sqrt`	`expr = sqrt(model.x)`
hyperbolic sine	`sinh`	`expr = sinh(model.x)`
tangent	`tan`	`expr = tan(model.x)`
hyperbolic tangent	`tanh`	`expr = tanh(model.x)`

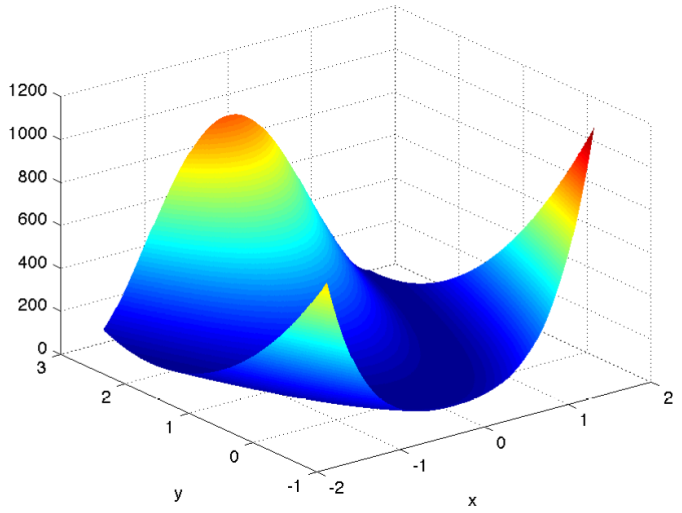
Caution: Always use the intrinsic functions that are part of the Pyomo package.

```
from pyomo.environ import * # imports, e.g., pyomo versions of exp, log, etc.)
from math import *          # overrides the pyomo versions with math versions
```

Example: Rosenbrock function

$$\min_{x,y} \; f(x,y) = (1-x)^2 + 100\left(y - x^2\right)^2$$

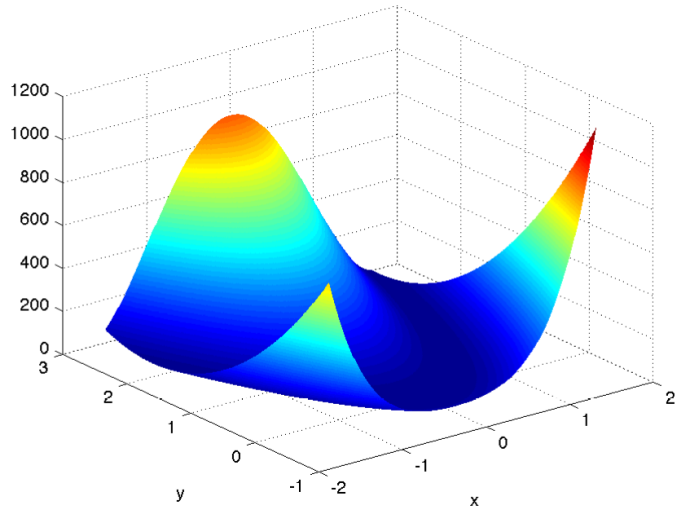
- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at x=1.5, y=1.5



Example: Rosenbrock function

$$\min_{x,y} \; f(x,y) = (1-x)^2 + 100\left(y - x^2\right)^2$$

- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at x=1.5, y=1.5




```
# rosenbrock.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)
```

Example: Rosenbrock function

```
# rosenbrock.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)
```

```
pyomo solve --solver=ipopt --summary --stream-solver rosenbrock.py
```

```
…
Variables:
  x : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
  y : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
…
```

Example: Rosenbrock function

```
…
Variables:
  x : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
  y : Size=1, Index=None, Domain=Reals
      Key  : Lower : Value : Upper : Fixed : Stale
      None :  None :   1.0 :  None : False : False
…
```

- How do I generate nicely formatted output?
- What if I want to solve this problem repeatedly with different initialization?
- What if I have data processing to do before hand?
- How can I use the power of Python to build optimization solutions?
- Write a Python script instead of using the “pyomo” command

Scripting brings the power of Python to Pyomo

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

Example: Scripting (Rosenbrock)

```
# rosenbrock_script.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

```
python rosenbrock_script.py
```

Exercise: Scripting (looping over initial value)

- Modify rosenbrock_script.py to solve the rosenbrock problem for different initial values and a table of output that shows the initial values and the solution for both x and y. (I.e., complete the following table)

x_init, y_init, x_soln, y_soln			
2.00	5.00	----	----
3.00	5.00	----	----
4.00	5.00	----	----
5.00	5.00	----	----

Example: Scripting (loop over initial value)

```
# rosenbrock_script_loop.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)


print('x_init, y_init, x_soln, y_soln')
y_init = 5.0
for x_init in range(2, 6):
    model.x = x_init
    model.y = 5.0

    solver = SolverFactory('ipopt')
    solver.solve(model)

    print("{0:6.2f}  {1:6.2f}  {2:6.2f}  {3:6.2f}".format(x_init, \
          y_init, value(model.x), value(model.y)))
```

Introduction to IPOPT

$$\min_{x} \quad f(x) \qquad \leftarrow \text{Objective Function}$$

$$\text{s.t.} \quad c(x) = 0 \qquad \leftarrow \text{Equality Constraints}$$

$$d^L \leq d(x) \leq d^U \qquad \leftarrow \text{Inequality Constraints}$$

$$x^L \leq x \leq x^U \qquad \leftarrow \text{Variable Bounds}$$

$$x \in \Re^n$$

$$f(x) : \Re^n \mapsto \Re$$

$$c(x) : \Re^n \mapsto \Re^m$$

$$d(x) : \Re^n \mapsto \Re^p$$

- f(x), c(x), d(x)
  - general nonlinear functions (non-convex?)
  - Smooth (C^2)
- The x variables are continuous
  - x(x-1)=0 for discrete conditions really doesn't work

Introduction to IPOPT

$$\min_{x} \quad f(x)$$
$$\text{s.t.} \quad c(x) = 0$$
$$d^L \leq d(x) \leq d^U$$
$$x^L \leq x \leq x^U$$

Cost/Profit, Measure of fit

Physics of the system

Physical, Performance, Safety Constraints

$$x \in \Re^n$$
$$f(x) : \Re^n \mapsto \Re$$
$$c(x) : \Re^n \mapsto \Re^m$$
$$d(x) : \Re^n \mapsto \Re^p$$

- f(x), c(x), d(x)
  - general nonlinear functions (non-convex?)
  - Smooth (C^2)
- The x variables are continuous
  - x(x-1)=0 for discrete conditions really doesn't work

Large Scale Optimization

- Gradient Based Solution Techniques

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \cdot \lambda - z &= 0 \\ c(x) &= 0 \\ X \cdot z &= 0 \\ (x \geq 0, z &\geq 0) \end{aligned}$$

Newton Step

$$\begin{bmatrix} W_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla f(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

$$\left(W_k = \nabla^2_{xx} \mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k) \lambda \right)$$

Active-set Strategy

Interior Point Methods

Original NLP

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

Barrier NLP

$$\begin{aligned} \min_x \quad & f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

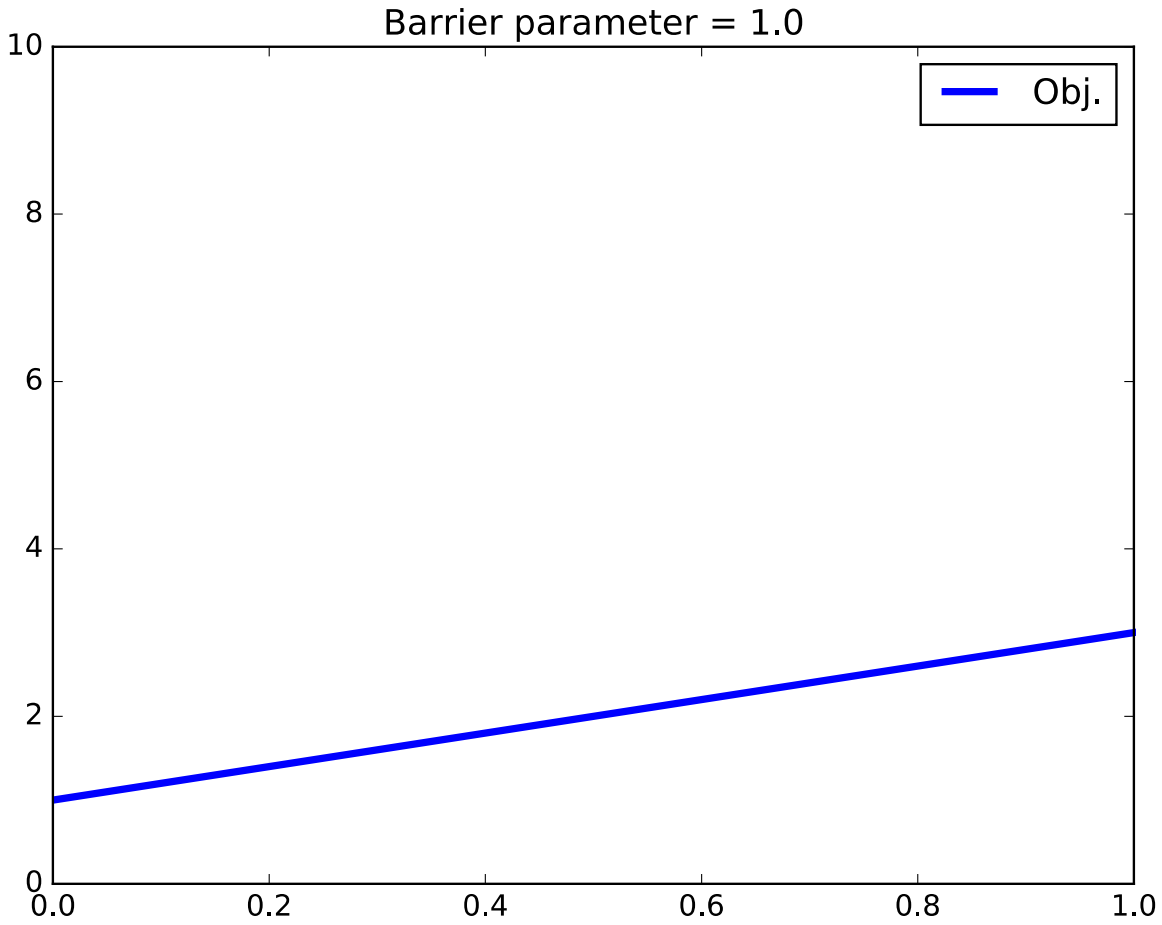
as $x \to 0, \quad ln(x) \to \infty$

as $\mu \to 0, \quad x^\star(\mu) \to x^\star$

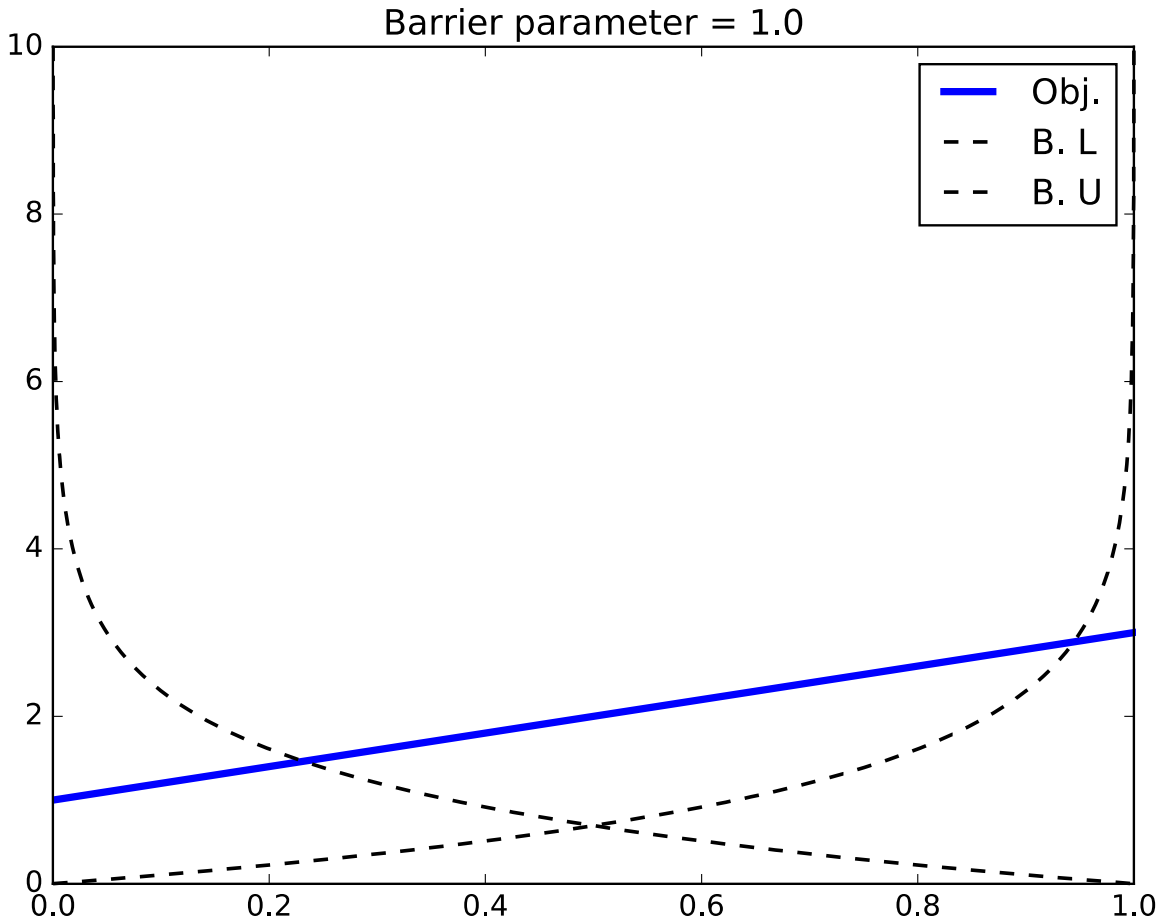
Fiacco & McCormick (1968)

- Initialize
  $x_0 > 0, \quad \mu_0 > 0, \quad \text{Set } l \leftarrow 0$
- Solve Barrier NLP for μ_l
- Decrease the barrier parameter
  $\mu_{l+1} < \mu_l$
- Increase $l \leftarrow l + 1$

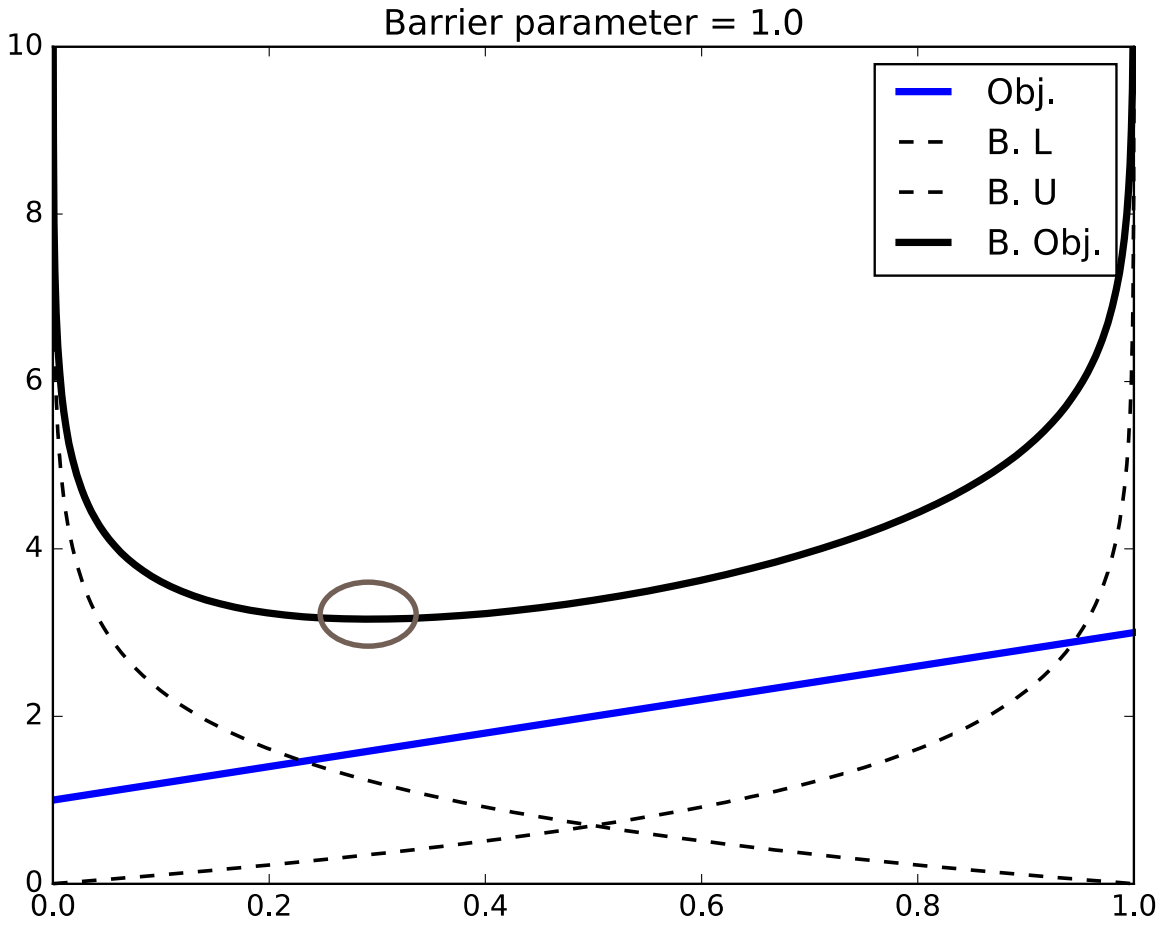
Effect of Barrier Term



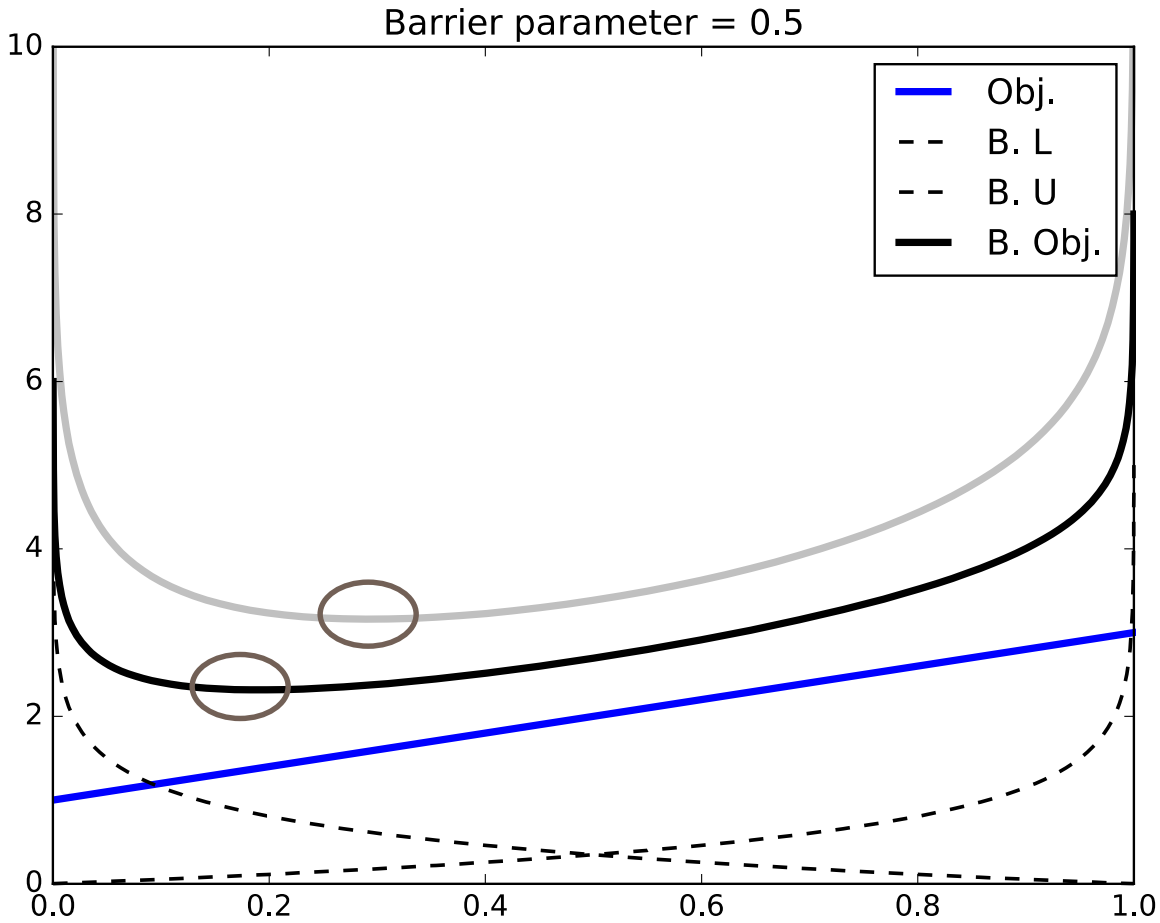
Effect of Barrier Term



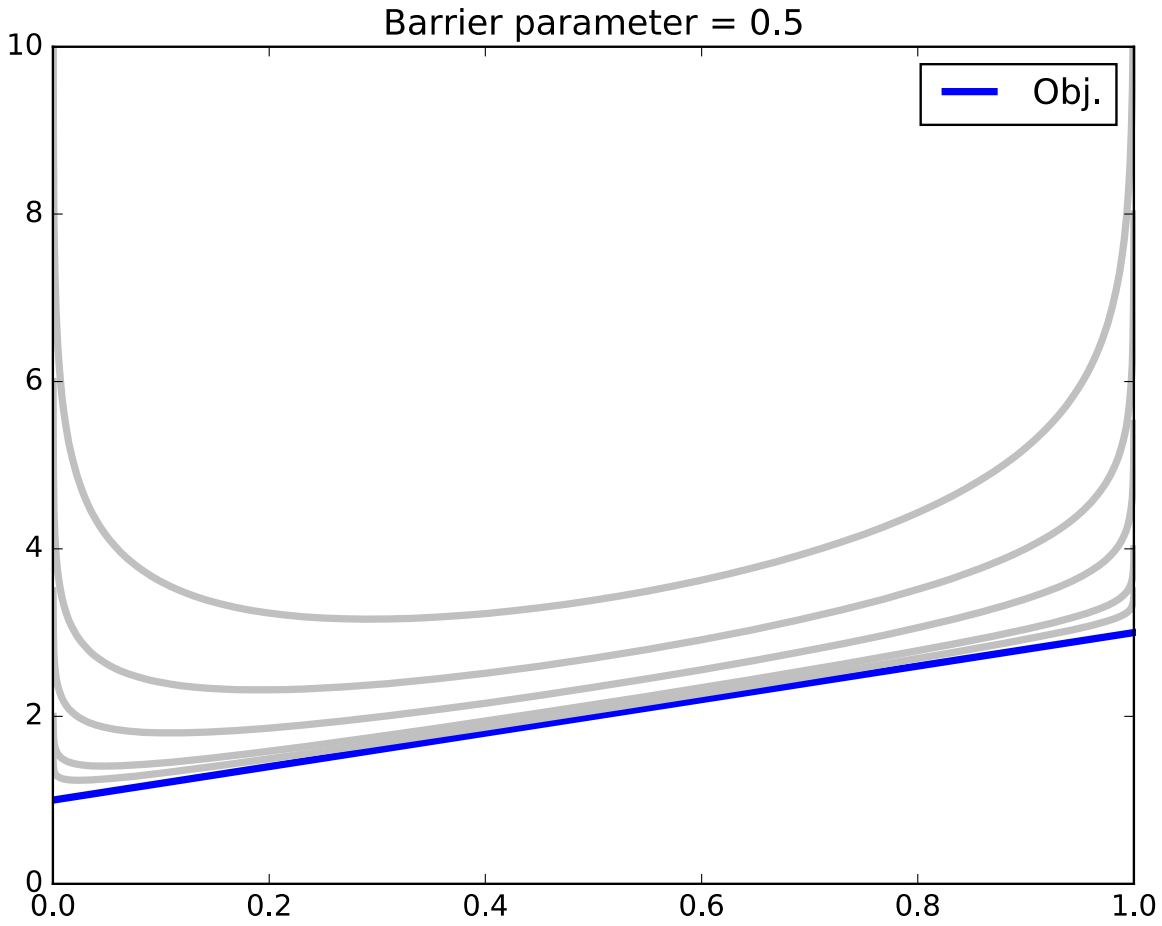
Effect of Barrier Term



Effect of Barrier Term



Effect of Barrier Term



Interior Point Methods

Original NLP

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

Barrier NLP

$$\begin{aligned} \min_x \quad & \varphi_\mu(x) = f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

$$\text{as} \quad x \to 0, \quad ln(x) \to \infty$$

$$\text{as} \quad \mu \to 0, \quad x^\star(\mu) \to x^\star$$

Fiacco & McCormick (1968)

- Initialize
  $x_0 > 0, \quad \mu_0 > 0, \quad \text{Set} \quad l \leftarrow 0$
- Solve Barrier NLP for μ_l
- Decrease the barrier parameter
  $\mu_{l+1} < \mu_l$
- Increase $l \leftarrow l + 1$

Solve Barrier NLP?
Barrier parameter update?
Globalization?

- KNITRO (Byrd, Nocedal, Hribar, Waltz)
- LOQO (Benson, Vanderbei, Shanno
- IPOPT (Waechter, Biegler)

Interior Point Methods

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t} \quad & c(x)=0 \\ & x \geq 0 \end{aligned}$$

$$\begin{aligned} \min_x \quad & f(x) - \mu \sum_i \ln(x_i) \\ \text{s.t} \quad & c(x)=0 \end{aligned}$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - z &= 0 \\ c(x) &= 0 \\ Xz &= \mu e \\ (x > 0, z > 0) & \end{aligned}$$

$z=\mu X^{-1} e$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - \mu X^{-1} e &= 0 \\ c(x) &= 0 \\ (x > 0) & \end{aligned}$$

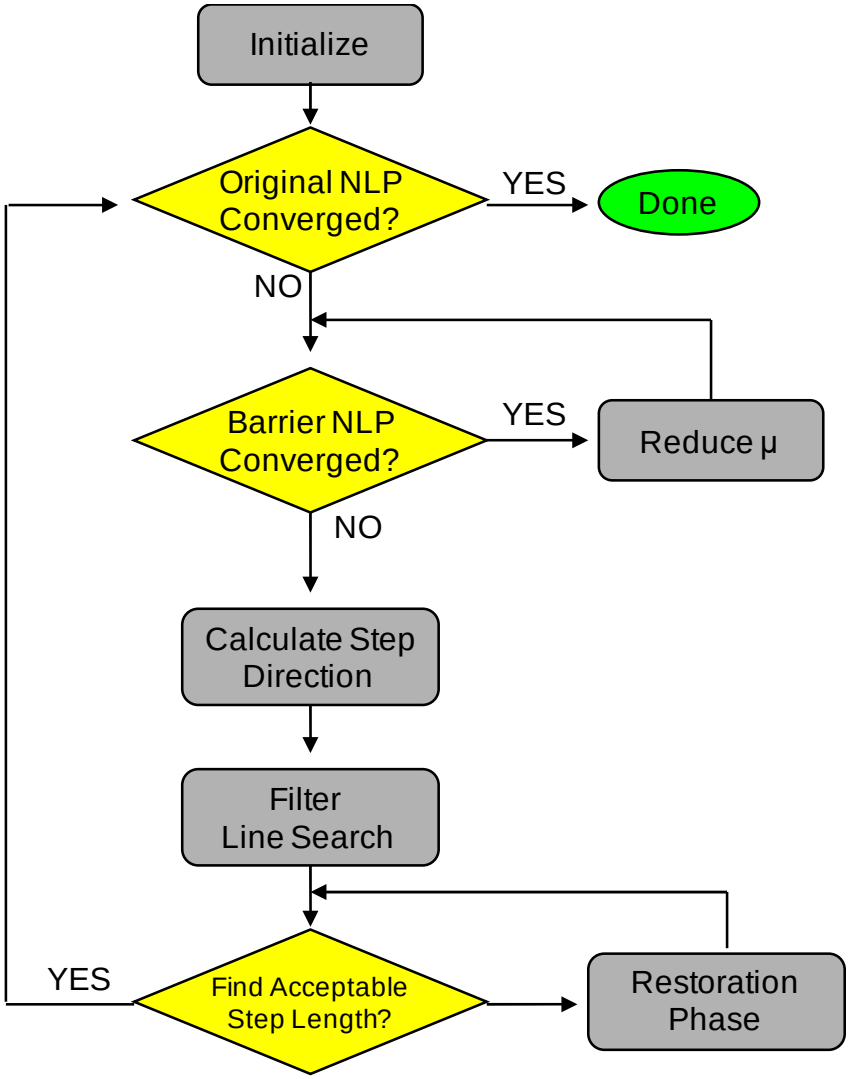
$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla \psi_\mu(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

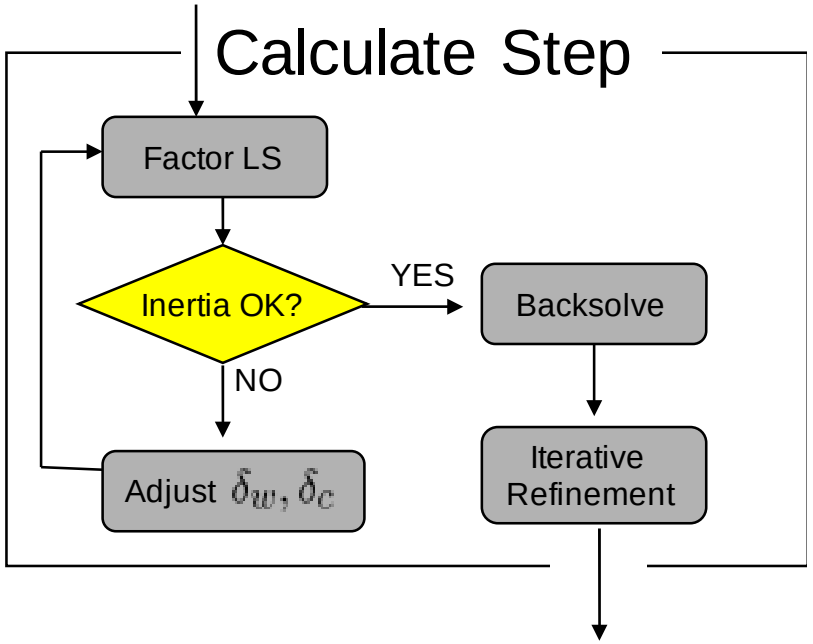
$$\left(W_k = \nabla^2_{xx} \mathcal{L},\ \Sigma_k = Z_k X_k^{-1}\right)$$

IPOPT: Other Considerations

- Regularization:
  - If certain convexity criteria are not satisfied at a current point, IPOPT may need to regularize. (This can be seen in the output.)
  - We do NOT want to see regularization at the final iteration (solution).
  - Can be an indicator of poor conditioning.
- Globalization:
  - IPOPT uses a filter-based line-search approach
  - Accepts the step if sufficient reduction is seen in objective or constraint violation
- Restoration Phase:
  - Minimize constraint violation
  - Regularized with distance from current point
  - Similar structure to original problem (reuse symbolic factorization)

IPOPT Algorithm Flowsheet





IPOPT Output

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.11.7, running with linear solver ma27.

Number of nonzeros in equality constraint Jacobian...:        2
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        1

Total number of variables............................:        2
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        1
                     variables with only upper bounds:        0
Total number of equality constraints.................:        1
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
```

IPOPT Output

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.4298076e-01 2.33e-01 7.67e-01  -1.0 5.20e-01    -  7.73e-01 9.52e-01h  1
   2  2.6898113e-02 7.23e-05 4.09e-04  -1.7 2.16e-01    -  1.00e+00 1.00e+00h  1
   3  1.8655807e-04 1.83e-04 7.86e-05  -3.8 2.68e-02    -  1.00e+00 9.97e-01f  1
   4  1.8250072e-06 1.23e-12 2.22e-16  -5.7 1.85e-04    -  1.00e+00 1.00e+00h  1
   5 -1.7494097e-08 8.48e-13 0.00e+00  -8.6 1.84e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:  -1.7494096510394117e-08   -1.7494096510394117e-08
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   8.4843243541854463e-13    8.4843243541854463e-13
Complementarity.........:   2.5050549017950606e-09    2.5050549017950606e-09
Overall NLP error.......:   2.5050549017950606e-09    2.5050549017950606e-09
```

- iter: iterations (codes)
- objective: objective
- Inf_pr: primal infeasibility (constraints satisfied? current constraint violation)
- Inf_du: dual infeasibility (am I optimal?)
- lg(mu): log of the barrier parameter, mu
- ||d||: length of the current stepsize
- lg(rg): log of the regularization parameter
- alpha_du: stepsize for dual variables
- alpha_pr: stepsize for primal variables
- ls: number of line-search steps

Exit Conditions

- Successful Exit
- Successful Exit with regularization at solution
- Infeasible
- Unbounded

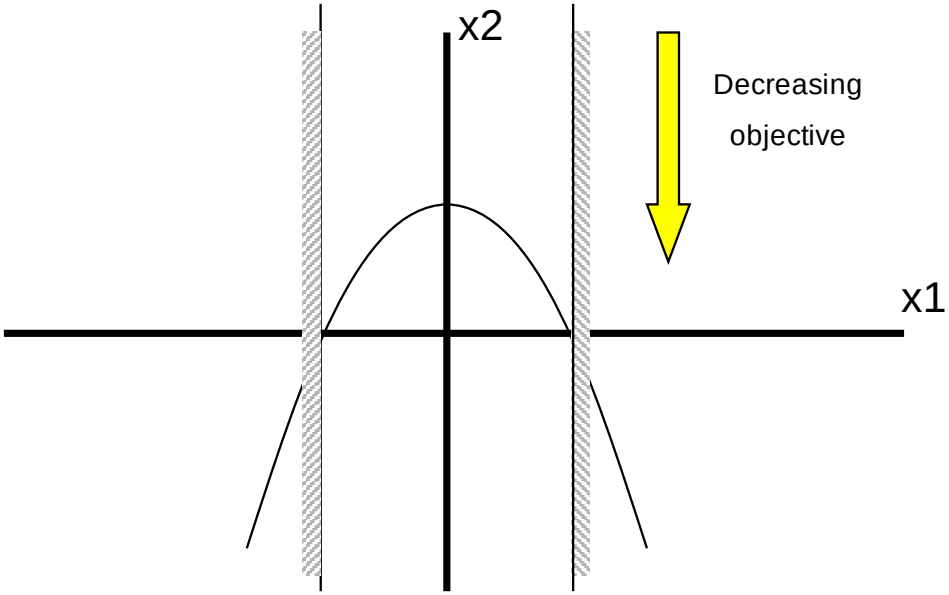
Exit Conditions: Successful

$$\min \quad x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

Initialize at (x1=0.5, x2=0.5)



Exit Conditions: Successful

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  5.0000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.4298076e-01 2.33e-01 7.67e-01  -1.0 5.20e-01    -  7.73e-01 9.52e-01h  1
   2  2.6898113e-02 7.23e-05 4.09e-04  -1.7 2.16e-01    -  1.00e+00 1.00e+00h  1
   3  1.8655807e-04 1.83e-04 7.86e-05  -3.8 2.68e-02    -  1.00e+00 9.97e-01f  1
   4  1.8250072e-06 1.23e-12 2.22e-16  -5.7 1.85e-04    -  1.00e+00 1.00e+00h  1
   5 -1.7494097e-08 8.48e-13 0.00e+00  -8.6 1.84e-06    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:  -1.7494096510367012e-08   -1.7494096510367012e-08
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   8.4843243541854463e-13    8.4843243541854463e-13
Complementarity.........:   2.5050549017950606e-09    2.5050549017950606e-09
Overall NLP error.......:   2.5050549017950606e-09    2.5050549017950606e-09

. . .

EXIT: Optimal Solution Found.

Ipopt 3.11.1: Optimal Solution Found

*** soln
x1 = 1.0
x2 = -1.7494096510367012e-08
```

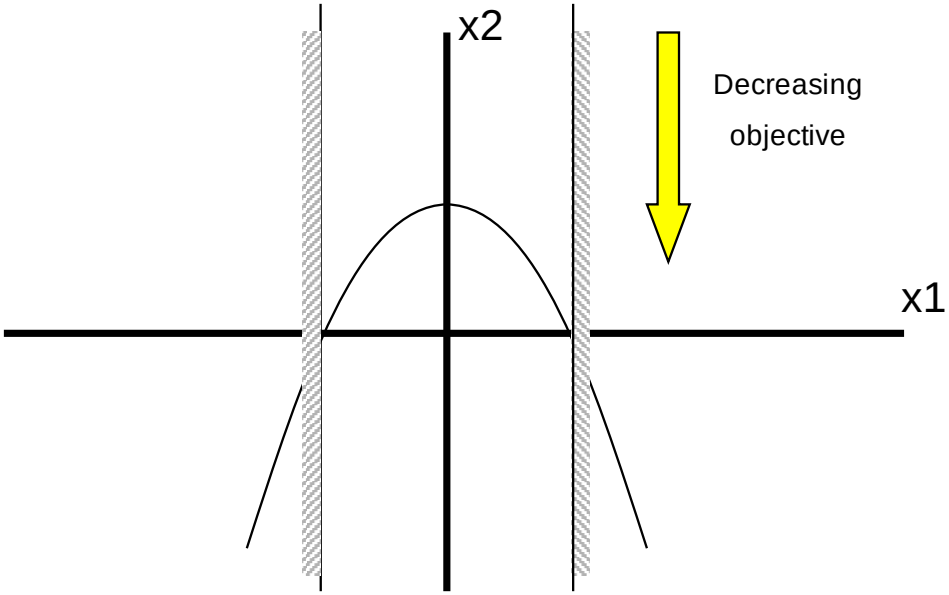
Exit Conditions: Successful w/ Regularization

$$\min \quad x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

Initialize at (x1=0.0, x2=2.0)



Exit Conditions: Successful w/ Regularization

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  2.0000000e+00 1.00e+00 0.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.0000000e+00 0.00e+00 1.00e-04  -1.7 1.00e+00  -4.0 1.00e+00 1.00e+00h  1
   2  1.0000000e+00 0.00e+00 0.00e+00  -3.8 0.00e+00   0.9 1.00e+00 1.00e+00   0
   3  1.0000000e+00 0.00e+00 0.00e+00  -5.7 0.00e+00   0.5 1.00e+00 1.00e+00T  0
   4  1.0000000e+00 0.00e+00 0.00e+00  -8.6 0.00e+00   0.9 1.00e+00 1.00e+00T  0

Number of Iterations....: 4

                                   (scaled)                 (unscaled)
Objective...............:   1.0000000000000000e+00    1.0000000000000000e+00
Dual infeasibility......:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5059035596800808e-09    2.5059035596800808e-09
Overall NLP error.......:   2.5059035596800808e-09    2.5059035596800808e-09

. . .

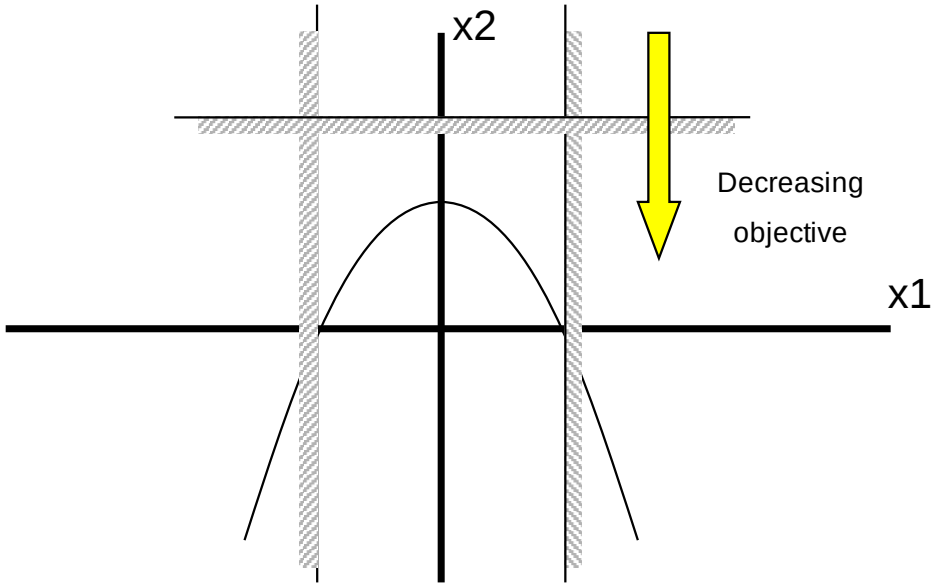
EXIT: Optimal Solution Found.

Ipopt 3.11.1: Optimal Solution Found

*** soln
x1 = 0.0
x2 = 1.0
```

Exit Conditions: Infeasible

$$
\begin{aligned}
\min \quad & -x_2 \\
\text{s.t.} \quad & -x_2 = x_1^2 - 1 \\
& -1 \leq x_1 \leq 1 \\
& x_2 \geq 2
\end{aligned}
$$



Exit Conditions: Infeasible

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
. . .
   5  2.0000016e+00 1.00e+00 4.74e+03  -1.0 3.47e+01    -  2.79e-02 4.16e-04h  7
   6r 2.0000016e+00 1.00e+00 1.00e+03   0.0 0.00e+00    -  0.00e+00 4.44e-07R  3
   7r 2.0010000e+00 1.01e+00 1.74e+02   0.0 8.73e-02    -  1.00e+00 1.00e+00f  1
   8r 2.0010010e+00 1.00e+00 1.32e-03   0.0 8.73e-02    -  1.00e+00 1.00e+00f  1
   9r 2.0000080e+00 1.00e+00 5.18e-03  -2.1 6.12e-03    -  9.94e-01 9.99e-01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10r 2.0000000e+00 1.00e+00 3.73e-06  -4.7 7.99e-06    -  1.00e+00 1.00e+00f  1
  11r 2.0000000e+00 1.00e+00 1.79e-07  -7.1 2.85e-08    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 11

                                   (scaled)                 (unscaled)
Objective...............:   1.9999999800009090e+00    1.9999999800009090e+00
Dual infeasibility......:   1.0000000002321485e+00    1.0000000002321485e+00
Constraint violation....:   9.9999998000090895e-01    9.9999998000090895e-01
Complementarity.........:   9.0909091652062654e-10    9.0909091652062654e-10
Overall NLP error.......:   9.9999998000090895e-01    1.0000000002321485e+00

. . .

EXIT: Converged to a point of local infeasibility. Problem may be infeasible.

Ipopt 3.11.1: Converged to a locally infeasible point. Problem may be infeasible.
WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt 3.11.1\x3a Converged
to a locally infeasible point. Problem may be infeasible.

*** soln
x1 = -6.353194883662875e-12
x2 = 2.0
```

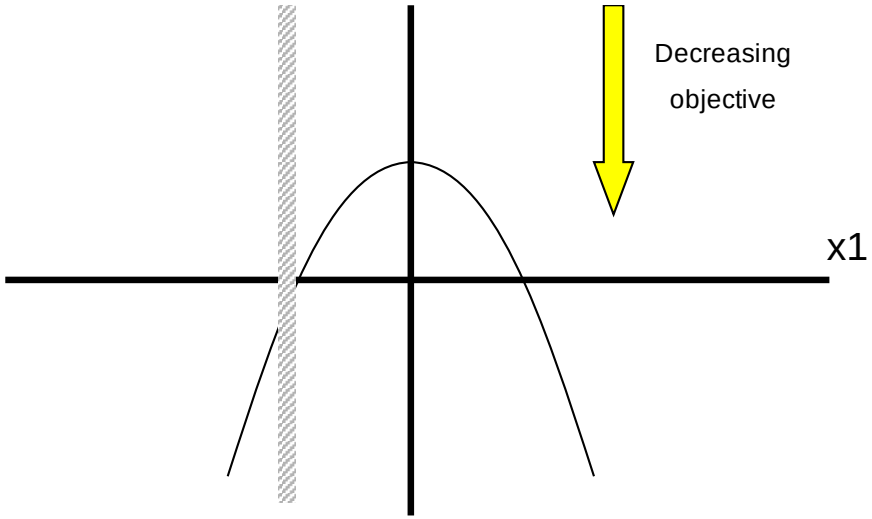
Exit Conditions: Unbounded

$$\min \quad -x_2$$

$$\text{s.t.} \quad -x_2 = x_1^2 - 1$$

$$-1 \leq x_1$$

Initialize at (x1=0.5, x2=0.5)



Exit Conditions: Unbounded

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
. . .
  45 -2.2420218e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 3.55e-08 7.11e-15f 48
  46 -2.2420225e+11 1.00e+04 1.00e+00  -1.7 3.75e+19 -19.6 1.20e-08 1.78e-15f 50
  47 -2.2420229e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 1.00e+00 3.55e-15f 49
  48 -3.7503956e+19 1.57e+27 8.36e+09  -1.7 3.75e+19 -19.6 1.18e-08 1.00e+00w  1
  49 -1.3750923e+20 3.92e+26 2.09e+09  -1.7 1.00e+20 -20.0 1.00e+00 1.00e+00w  1

Number of Iterations....: 49

                                   (scaled)                 (unscaled)
Objective...............:  -1.3750923074037683e+20   -1.3750923074037683e+20
Dual infeasibility......:   2.0888873315629249e+09    2.0888873315629249e+09
Constraint violation....:   3.9209747283936173e+26    3.9209747283936173e+26
Complementarity.........:   3.1115099971882619e+03    3.1115099971882619e+03
Overall NLP error.......:   3.9209747283936173e+26    3.9209747283936173e+26

EXIT: Iterates diverging; problem might be unbounded.

Ipopt 3.11.1: Iterates diverging; problem might be unbounded.
WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt
3.11.1\x3a Iterates diverging; problem might be unbounded.

*** soln
x1 = 0.5
x2 = 0.5
```

IPOPT Options

- Solver options can be set through scripts (and the pyomo command line)
- print_options_documentation yes
  - Outputs the complete set of IPOPT options (with documentation and their defaults)
- mu_init
  - Sets the initial value of the barrier parameter
  - Can be helpful to make this smaller when initial guesses are known to be good
- bounds_push
  - By default, IPOPT pushes the bounds a little further out.
  - This can be set to remove this behavior
  - E.g., sqrt(x), x>= 0
- linear_solver
  - Set the linear solver that will be used for the KKT system
  - Significantly better performance with HSL (MA27) over default MUMPS
- print_user_options
  - Print options set and whether or not they were used
  - Helpful to detect mismatched options

IPOPT Options

```
# rosenbrock_options.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.options['mu_init'] = 1e-4
solver.options['print_user_options'] = 'yes'
solver.options['ma27_pivtol'] = 1e-4
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

IPOPT Options

```
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001


List of user-set options:

                                    Name   Value                used
                         ma27_pivtol = 0.0001                     no
                            mu_init = 0.0001                     yes
                  print_user_options = yes                       yes

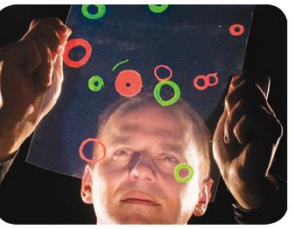
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

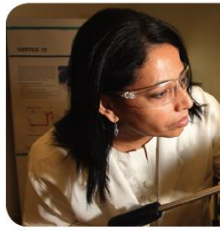
NOTE: You are using Ipopt by default with the MUMPS linear solver.
      Other linear solvers might be more efficient (see Ipopt documentation).


This is Ipopt version 3.11.1, running with linear solver mumps.
```

Modeling Tips

- Variable Initialization
  - Proper initialization of nonlinear problems can be critical for effective solution.
  - Strategies include:
    - Using understood physics or past successful solutions
    - Solving simpler problem(s) first, progressing to more difficult
- Undefined Evaluations
  - Many mathematical functions have a valid domain, and evaluation outside that domain causes errors
  - Add appropriate bounds to variables to keep them inside valid domain
  - Note that solvers use first and second derivatives. While sqrt(x) is valid at x=0, 1/sqrt(x) is not
- Problem Scaling
  - Scale model to avoid variables, constraints, derivatives with different scales.
- Formulation Matters!





Sandia National Laboratories

Exceptional service in the national interest

6. Structured Modeling & Transformations

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Is this an *optimization model*?

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

Models are for *Modelers*

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

- I would argue this is an *optimization problem!*
- So, what's a *model*?
  - A general representation of a class of problems
    - Data (instance) independent
  - Represents the modeler's understanding of the class of problems
    - Explicitly annotates and conveys the class structure
  - Incorporates assumptions and simplifications
  - Is both tractable and valid
    - (although these are often contradictory goals)

Models are for *Modelers*

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

- I would argue this is an *optimization problem!*
- So, what's a *model*?
  - A general representation of a class of problems
    - Data (instance) independent
  - Represents the modeler's understanding of the class of problems
    - Explicitly annotates and conveys the class structure
  - Incorporates assumptions and simplifications
  - Is both tractable and valid
    - (although these are often contradictory goals)

Optimization problems: Model instances

- We seldom have a single *problem* to solve
  - Rather we would like to write a *single model* for a *class of problems*
    - Key design feature of many AMLs (e.g. strongly encouraged by AMPL)
  - Why?
    - Test small, deploy big
    - Tomorrow's problem is different from today's
    - Data may be
      - Huge
      - Machine-generated
      - Stored externally (loaded from external tools, e.g. databases)

Models are for *Modelers*

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

- I would argue this is an *optimization problem!*
- So, what's a *model*?
  - A general representation of a class of problems
    - Data (instance) independent
  - Represents the modeler's understanding of the class of problems
    - Explicitly annotates and conveys the class structure
  - Incorporates assumptions and simplifications
  - Is both tractable and valid
    - (although these are often contradictory goals)

What is *model structure*?

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

- Unlike a solver, modelers don't think in terms of "A"
  - Rather, I think in terms of repeated (indexed) units
    - Sets (1-, 2-, n- dimensional)
    - Vectors or matrices of variables
    - Groups of related constraints (blocks)
- The model may not be "flat"
  - Block diagonal (e.g., scenarios in stochastic programming)
  - Graph-based (e.g., network flow)
  - Hierarchically defined (e.g., a model composed of sub-models)

Models are for *Modelers*

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \le b \\ & x \in \Re^n \end{aligned}$$

- I would argue this is an *optimization problem!*
- So, what's a *model*?
  - A general representation of a class of problems
    - Data (instance) independent
  - Represents the modeler's understanding of the class of problems
    - Explicitly annotates and conveys the class structure
  - Incorporates assumptions and simplifications
  - Is both tractable and valid
    - (although these are often contradictory goals)

Tractability / validity: The optimization tug-of-war

- The "highest fidelity" model of a system is rarely tractable
  - Delicate balance between the model we want to solve and the solver we want to use
  - What can we do?
    - Simplify (reduce the model scope)
    - Approximate (relax or recast constraints)
    - Iterate (solve a series of related problems to develop the solution to the original problem)
  - Optimization 101 ingrains this tension into us; consider:

$$\begin{aligned} \max \quad & abs(x-3) \\ s.t. \quad & [\ldots] \end{aligned}$$

“Modeling” absolute value

- This probably makes you cringe:
  - “Experienced modelers would never write `abs()`!”
- Instead, we write:

$$\begin{aligned} \max \quad & abs(x-3) \\ s.t. \quad & [\ldots] \end{aligned}$$

$$\begin{aligned} \max \quad & absX \\ s.t. \quad & absX = negX + posX \\ & negX \le My \\ & posX \le M(1-y) \\ & X - 3 = posX - negX \\ & posX \ge 0, negX \ge 0 \\ & y \in \{0,1\} \\ & [\ldots] \end{aligned}$$

"Modeling" absolute value

- This probably makes you cringe:
  - "Experienced modelers would never write `abs()`!"

$$\begin{aligned} \max \quad & abs(x-3) \\ s.t. \quad & [\ldots] \end{aligned}$$

- Instead, we write:

$$\begin{aligned} \max \quad & absX \\ s.t. \quad & absX = negX + posX \\ & negX \le My \\ & posX \le M(1-y) \\ & X - 3 = posX - negX \\ & posX \ge 0, negX \ge 0 \\ & y \in \{0,1\} \\ & [\ldots] \end{aligned}$$

- But what if "[…]" is a nonlinear model? Then,

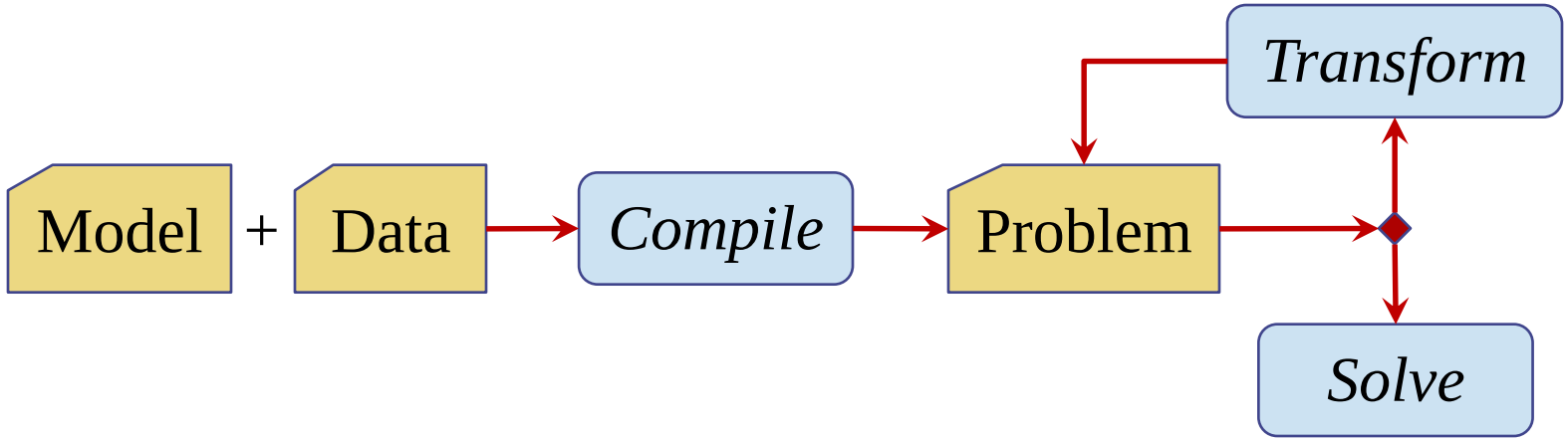
$$absX = \sqrt{x^2 + \varepsilon}$$

$$absX = \frac{2x}{1+e^{-x/h}} - x$$

- Does any of this really encode our understanding of the *class of problems*?
  - …or is this a reflection of our understanding of the *solver*?

Transformations: *Projecting problems to problems*

- Model Transformations
  - Project from one problem space to another
  - Standardize common reformulations or approximations
  - Convert “unoptimizable” modeling constructs into equivalent optimizable forms



Transformations are not entirely new

- LINGO's automatic linearization:

```
MODEL:
  MAX = @ABS( X-3 );
  X <= 2;
END
```

  - Generates the "usual" Big-M integer linear model:

```
MAX _C3
SUBJECT TO
  X <= 2
  - _C1 - _C2 + _C3 = 0
  _C1 – 100000 _C4 <= 0
  _C2 + 100000 _C4 <= 100000
  X - _C1 + _C2 = 3
END
INTE _C4
```

Cunningham and Schrage, "The LINGO Algebraic Modeling Language." In *Modeling Languages in Mathematical Optimization*, Josef Kallrath ed. Springer, 2004.

Why are we interested in transformations?

Sandia National Laboratories

- Separate model expression from how we intend to solve it
  - Defer decisions that improve tractability until solution time
  - Explore alternative reformulations or representations
  - Support *solver-specific* model customizations (e.g., `abs()`)
  - Support iterative methods that use different solvers requiring different representations (e.g., initializing NLP from MIP)
- Support "higher level" or non-algebraic modeling constructs
  - Express models that are closer to reality, e.g.:
    - Piecewise expressions
    - Disjunctive models (switching decisions & logic models)
    - Differential-algebraic models (dynamic models)
    - Bilevel models (game theory models)
- Reduce "mechanical" errors due to manual transformation

Structural transformations: Disjunctive programs

- Disjunctions: selectively enforce sets of constraints
  - Sequencing decisions: x ends before y or y ends before x
  - Switching decisions: a process unit is built or not
  - Alternative selection: selecting from a set of pricing policies

- Implementation: leverage Pyomo "blocks"
  - `Disjunct`:
    - Block of Pyomo components
      - (Var, Param, Constraint, etc.)
    - Boolean (binary) indicator variable determines if block is enforced
  - `Disjunction`:
    - Enforces logical XOR across a set of Disjunct indicator variables
  - (Logic constraints on indicator variables)

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \le o \\ c_k = \gamma_{ik} \end{bmatrix}$$
$$\Omega(Y) = true$$

Example: Task sequencing

- Prevent tasks colliding on a single piece of equipment
  - Derived from Raman & Grossmann (1994)
  - Given:
    - Tasks I processed on a sequence of machines (with no waiting)
    - Task i starts processing at time t_i with duration τ_{im} on machine m
    - $J(i)$ is the set of machines used by task i
    - C_{ik} is the set of machines used by both tasks i and j

$$\left[t_i + \sum_{\substack{m \in J(i) \\ m \le j}} \tau_{im} \le t_k + \sum_{\substack{m \in J(k) \\ m < j}} \tau_{km} \right]^{Y_{ik}} \vee \left[t_k + \sum_{\substack{m \in J(k) \\ m \le j}} \tau_{km} \le t_i + \sum_{\substack{m \in J(i) \\ m < j}} \tau_{im} \right]^{Y_{ki}}$$

$$\forall j \in C_{ik}, \forall i,k \in I, i < k$$

Example: Task sequencing in Pyomo

```
from pyomo.dae import *
def _NoCollision(disjunct, i, k, j, ik):
    model = disjunct.model()
    lhs = model.t[i] + sum(model.tau[i,m] for m in model.STAGES if m<j)
    rhs = model.t[k] + sum(model.tau[k,m] for m in model.STAGES if m<j)
    if ik:
        disjunct.c = Constraint( expr= lhs + model.tau[i,j] <= rhs )
    else:
        disjunct.c = Constraint( expr= rhs + model.tau[k,j] <= lhs )
model.NoCollision = Disjunct( model.L, [0,1], rule=_NoCollision )

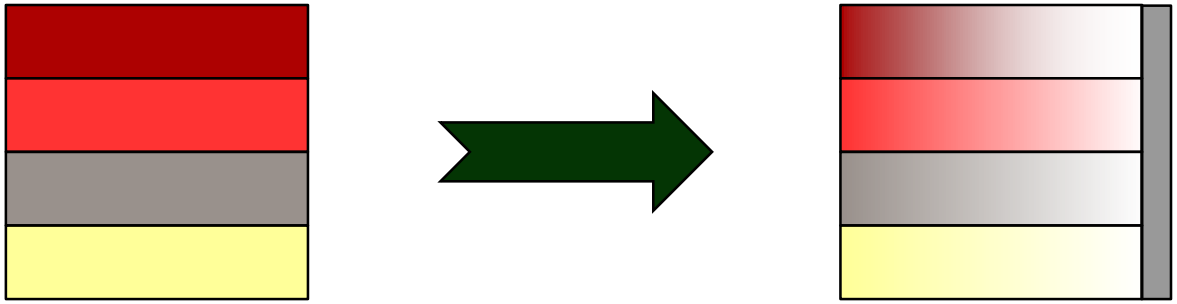
def _setSequence(model, i, k, j):
    return [ model.NoCollision[i,k,j,ik] for ik in [0,1] ]
model.setSequence = Disjunction(model.L, rule=_setSequence)
```

$$\left[t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} + \overset{Y_{ik}}{\tau_{ij}} \leq t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} \right] \vee \left[t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} + \overset{Y_{ki}}{\tau_{kj}} \leq t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} \right]$$

$$\forall j \in C_{ik}, \forall i,k \in I, i<k$$

Solving disjunctive models

- Few solvers "understand" disjunctive models
  - *Transform* model into standard math program
  - Big-M relaxation:
    - Convert logic variables to binary
    - Split equality constraints in disjuncts into pairs of inequality constraints
    - Relax all constraints in the disjuncts with "appropriate" M values



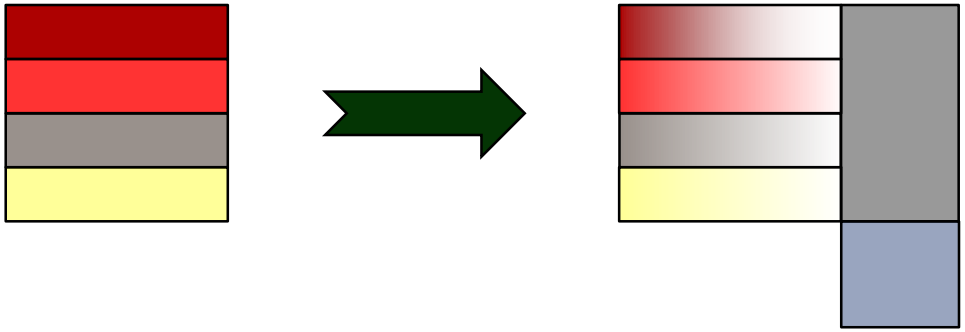
```
pyomo solve --solver cbc --transform=gdp.bigm jobshop.py jobshop.dat
```

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)

Why is the transformation interesting?

- Model preserves explicit disjunctive structure
- Automated transformation reduces errors
- Automatically identifies appropriate M values (for bounded linear)
- Big-M is not the only way to relax a disjunction!
  - Convex hull transformation (Balas, 1985; Lee and Grossmann, 2000)



```
pyomo solve --solver cbc --transform=gdp.chull jobshop.py jobshop.dat
```

  - Algorithmic approaches
    - e.g., Trespalacios and Grossmann (submitted 2013)
  - Prematurely choosing one relaxation makes trying others difficult

Expression transformations: MPEC

- Mathematical Programming with Equilibrium Constraints (MPEC)
  - Engineering design, economic equilibrium, multilevel games
  - Feasible region may be nonconvex and disconnected

- Equilibrium Constraints
  - Variational inequalities
  - Complementarity conditions
  - Optimality conditions (for bilevel problems)

MPEC formulations

- General MPEC models can be expressed as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0 \\ & a_i \leq w_i(x) \leq b_i \perp v_i(x) \quad i = 1 \ldots m \end{aligned}$$

- The last set of constraints are generalized mixed complementarity conditions (Ferris, Fourer, and Gay, ‘06), which have the form

$$\begin{aligned} \text{either } w_i(x) = a_i \quad & \text{and} \quad v_i(x) \geq 0 \\ \text{or } w_i(x) = b_i \quad & \text{and} \quad v_i(x) \leq 0 \\ \text{or } a_i < w_i(x) < b_i \quad & \text{and} \quad v_i(x) = 0 \end{aligned}$$

Modeling languages support MPECs

- AMPL
  - The `complements` keyword is used to denote complementarity between two constraints, expressions or variables
- GAMS
  - The `complements` keyword is used to denote complementarity between two constraints, expressions or variables
- AIMMS
  - Express mixed complementarity conditions by declaring complementarity variables along with associated constraints
- YALMIP
  - The `complements` function declares a constraint that reflects a mixed complementarity condition.

➢ Common challenge: lack of control over how the complementarity constraints are exposed to the solver

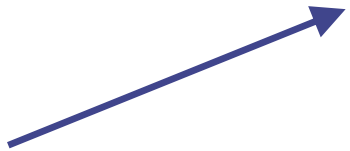
Complementarity conditions in Pyomo

Sandia National Laboratories

```
from pyomo.environ import *
from pyomo.mpec import Complementarity

M = ConcreteModel()
M.x = Var(bounds=(-1,2))
M.y = Var()

M.c3 = Complementarity(expr=(M.y - M.x**2 + 1 >= 0, M.y >= 0))
```



- The `Complementarity` component declares a complementarity condition
- The tuple argument specifies the two constraints, expressions, or variables in the complementarity condition.

This model definition is solver agnostic!

A simple nonlinear reformulation

$$
\begin{aligned}
\min \quad & f(x) \\
s.t. \quad & h(x) = 0 \\
& a_i \leq \omega_i \leq b_i && i = 1 \dots m \\
& \omega_i = w_i(x) && i = 1 \dots m \\
& (\omega_i - a_i) v_i(x) \leq 0 && i = 1 \dots m \\
& (\omega_i - b_i) v_i(x) \leq 0 && i = 1 \dots m
\end{aligned}
$$

- NOTE: There are serious difficulties with solving this formulation as standard stability assumptions are not met.
  - But other nonlinear transformations exist!

A simple disjunctive reformulation

$$
\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & h(x) = 0 \\
& \begin{bmatrix} y_{1,i} \\ w_i(x) = a_i \\ v_i(x) \geq 0 \end{bmatrix} \vee \begin{bmatrix} y_{2,i} \\ w_i(x) = b_i \\ v_i(x) \leq 0 \end{bmatrix} \vee \begin{bmatrix} y_{3,i} \\ a_i < w_i(x) < b_i \\ v_i(x) = 0 \end{bmatrix} && i = 1 \dots m \\
& y_{1,i} + y_{2,i} + y_{3,i} = 1 && i = 1 \dots m \\
& y_{1,i}, y_{2,i}, y_{3,i} \in \{0,1\} && i = 1 \dots m
\end{aligned}
$$

Back to our original example: ABS(x)

- Chaining transformations

$$f = abs(x) \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^+ \geq 0 \perp x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ \begin{bmatrix} Y \\ x^- = 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y \\ x^+ = 0 \end{bmatrix} \\ x^+ \geq 0, x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^- \leq My \\ x^- \leq M(1-y) \\ x^+ \geq 0, x^- \geq 0 \end{array}$$

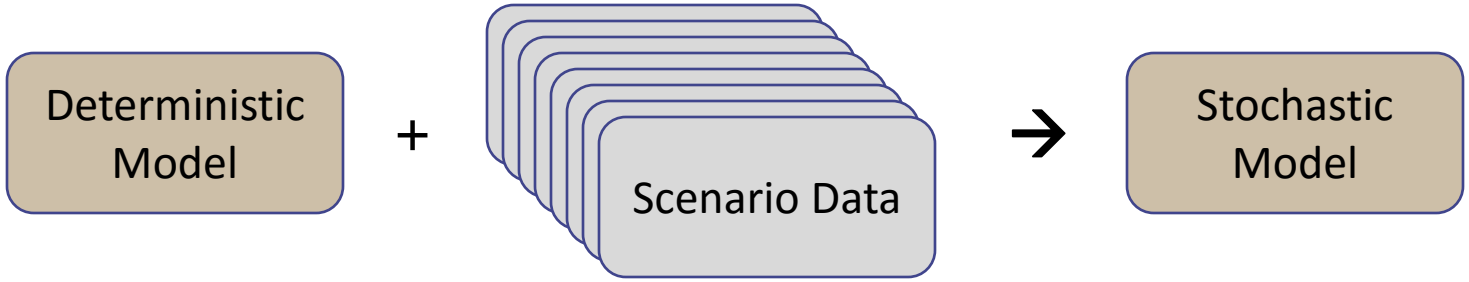
Notional transformation chain:

```
model = ConcreteModel()
# […]
TransformFactory("abs.complements").apply_to(model)
TransformFactory("mpec.simple_disjunction").apply_to(model)
TransformFactory("gdp.bigm").apply_to(model)
```

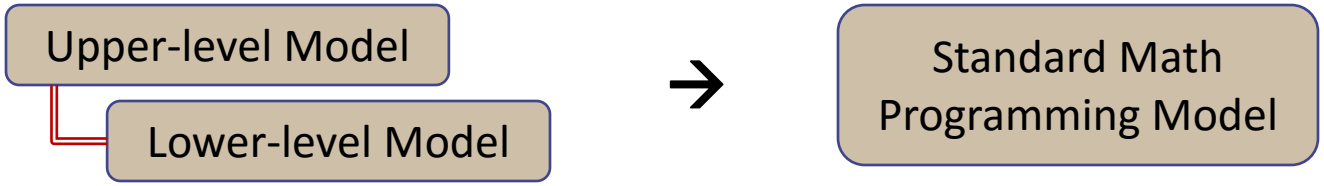
(Pyomo) Transformations for other domains

Sandia National Laboratories

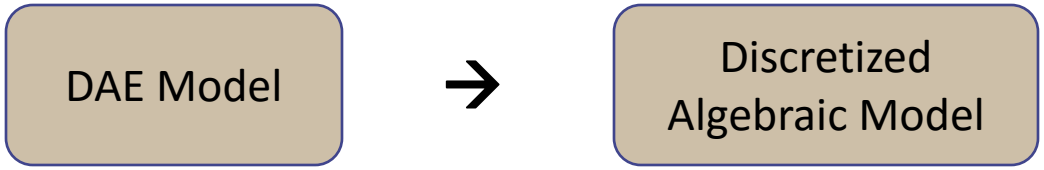
- Stochastic programming

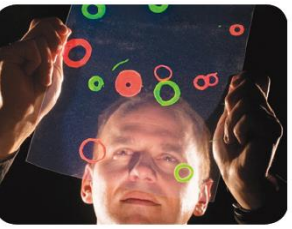


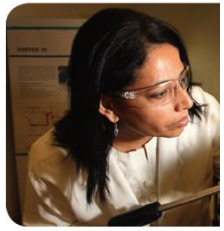
- Bilevel models



- DAE models







Sandia National Laboratories

Exceptional service in the national interest

7. Dynamic Systems

PYOMO

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Extending the Pyomo environment

- What about models that are not strictly math programs?
  - Dynamic systems
- Optimization of dynamic systems is *hard*.
  - In OR, think "multi-stage" problems
  - In "engineered systems", think differential equations
    - High fidelity *simulation* is difficult and expensive (e.g., HPC)
    - How to optimize?

Examples of Dynamic Optimization

- Parameter Estimation
- Nonlinear model predictive control
- Batch process operation
- Reactor design

$$\min \Psi(x(t), y(t), u(t))$$

$$\dot{x}(t) = f(x(t), y(t), u(t), t, p)$$

DAE model

$$0 = g(x(t), y(t), u(t), t, p)$$

$$x(t) \in \Re^n$$

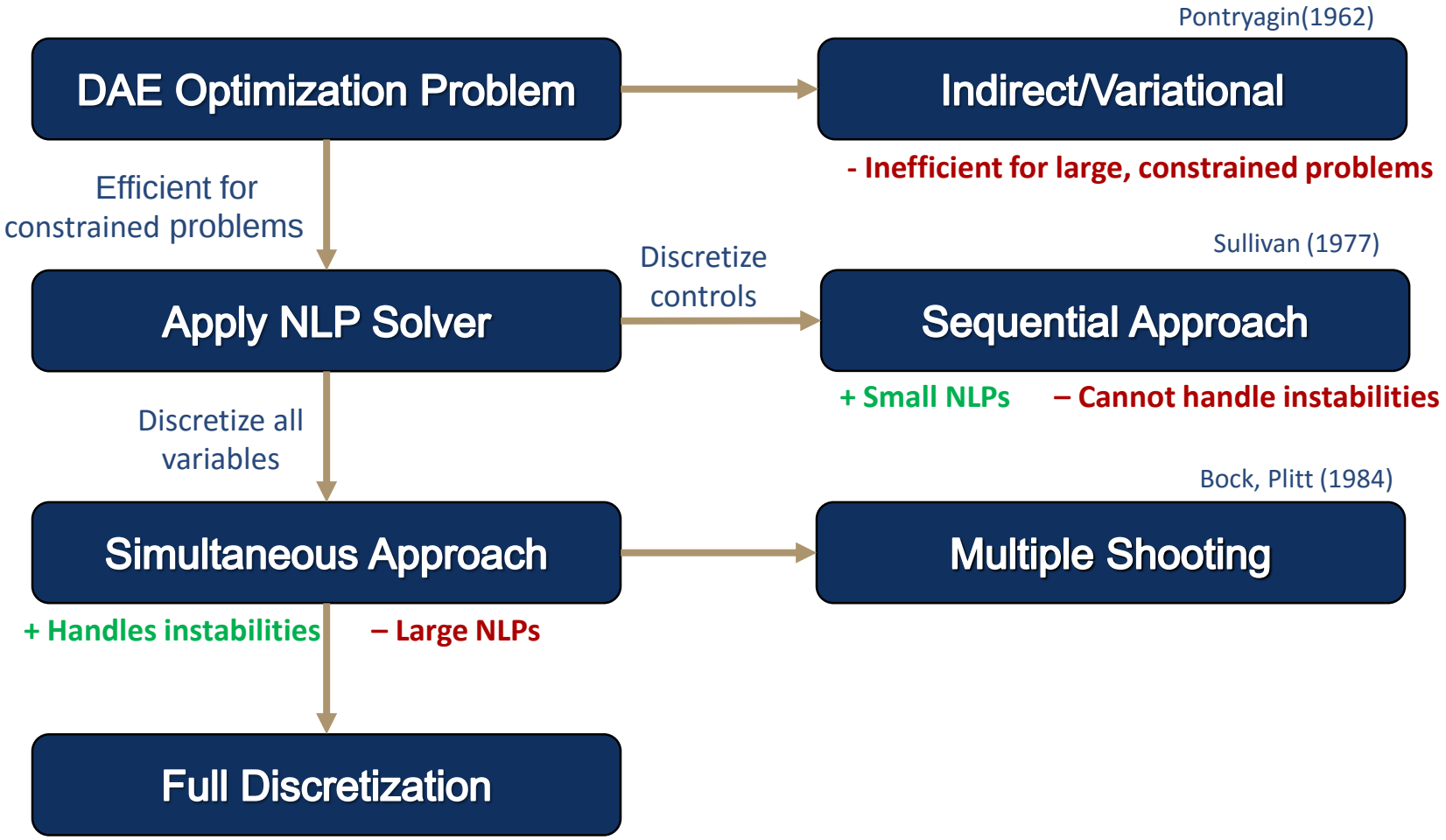
$$y(t) \in \Re^n$$

$$u(t) \in \Re^m$$

x: State (differential) variables
u: Control (input) variables
y: Algebraic variables

Solution Approaches

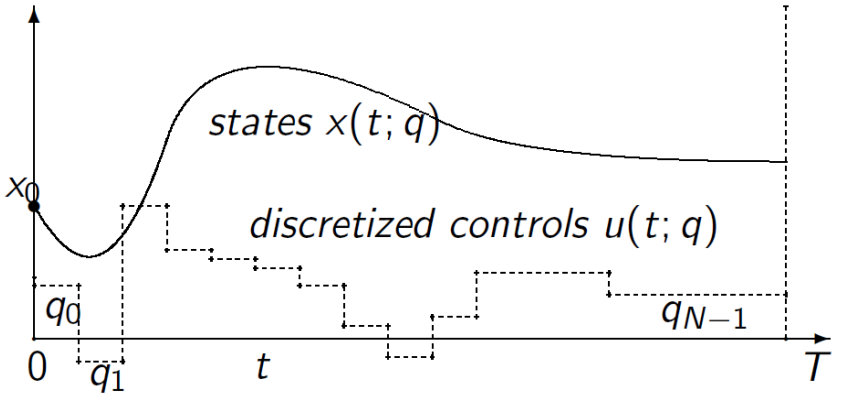
[Figure from L.T. Biegler (2007)]

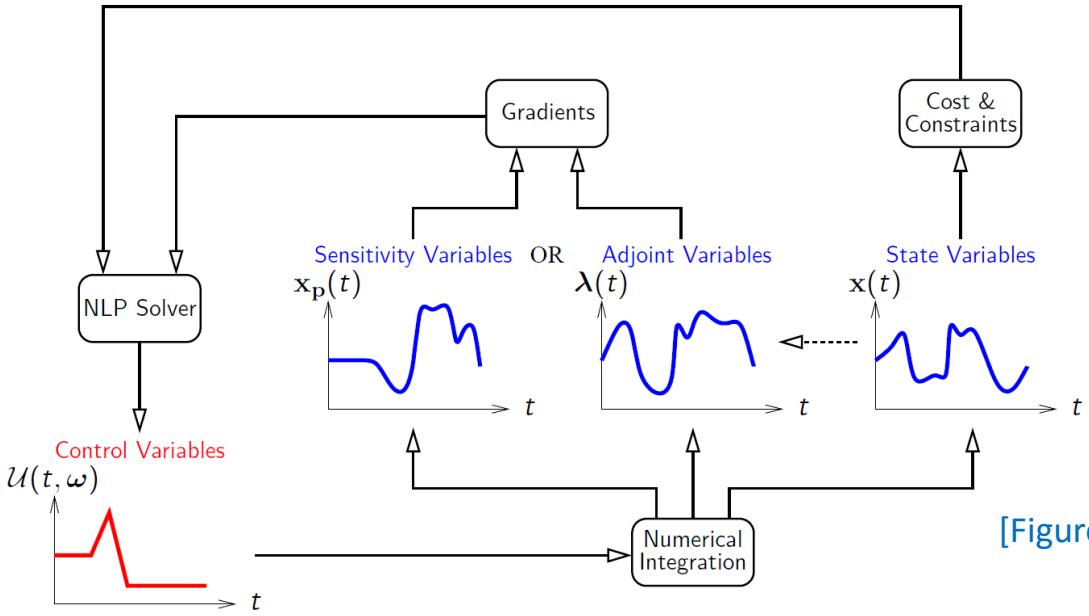


Sequential Approach

[Figure from M. Diehl]

- Single Shooting Method

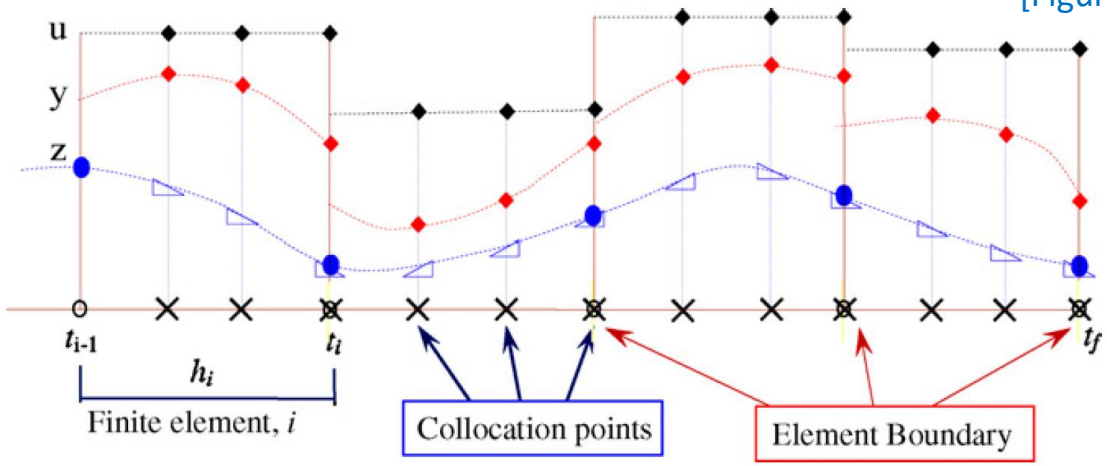




[Figure from B. Chachuat (2009)]

Simultaneous Approach

[Figure from L.T. Biegler (2007)]



- Multiple Shooting
  - Discretize controls and initial conditions for each finite element
  - ✓ Embeds DAE Solvers/Sensitivity
  - ✓ Can solve unstable systems
  - ✓ Good for problems with long time horizons and few dynamic state
  - X Dense sensitivity blocks
  - X Difficult to enforce path constraints
- Full Discretization
  - Discretize all variables
  - ✓ Can solve unstable systems
  - ✓ Good for problems with many dynamic states and degrees of freedom
  - ✓ Sparse NLP
  - X Large-scale NLP

Full Discretization

- Finite Difference Methods

$$\frac{df}{dt}(t) = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h}$$

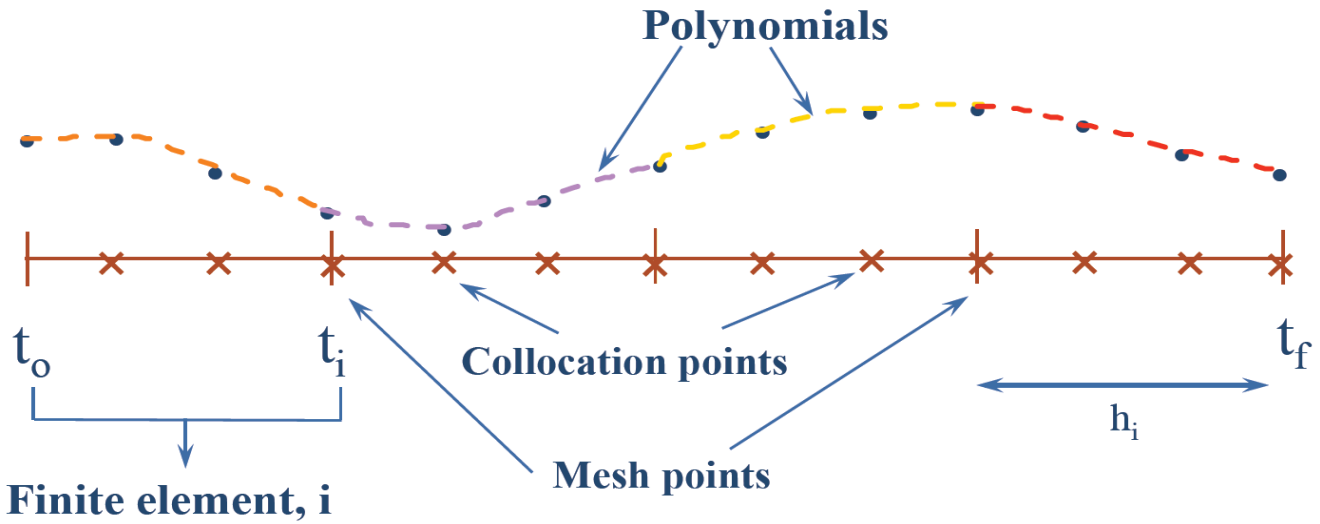
Forward Difference →

$$\frac{df}{dt}(t) = \frac{f(t+h) - f(t)}{h}$$

Backward Difference →

$$\frac{df}{dt}(t) = \frac{f(t) - f(t-h)}{h}$$

- Collocation over finite elements



Simple Example

$$\frac{dz}{dt} = z^2 - 2z + 1, \qquad z(0) = -3$$

- Exercise: Solve the differential equation using a backward finite difference scheme over the time interval $t \in [0,1]$

Backward Difference $\frac{df}{dt}(t) = \frac{f(t) - f(t-h)}{h}$

- Exercise: Plot the solution against the analytic solution

Analytic solution $z(t) = (4t - 3)/(4t + 1)$

Simple Example – Exercise Solution

- Solve the differential equation using a backward finite difference scheme

```
from pyomo.environ import *

numpoints = 10
model = m = ConcreteModel()
m.points   = RangeSet(0,numpoints)
m.h      = Param(initialize=1.0/numpoints)

m.z    = Var(m.points)
m.dzdt = Var(m.points)

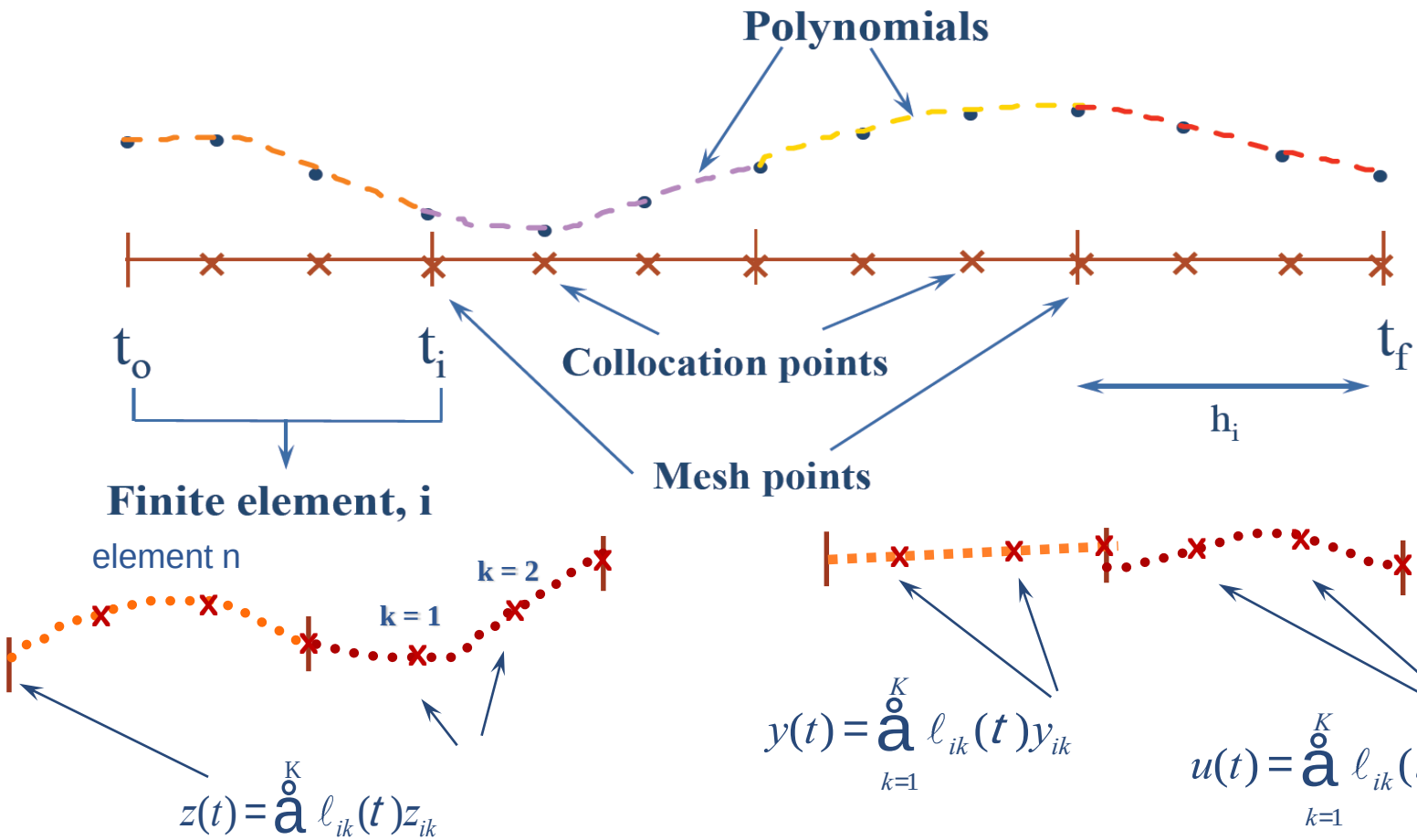
m.obj = Objective(expr=1) # Dummy Objective

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

def _back_diff(m,i):
    if i == 0:
        return Constraint.Skip
    return m.dzdt[i] == (m.z[i]-m.z[i-1])/m.h
m.back_diff = Constraint(m.points, rule=_back_diff)

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

Collocation over finite elements



Differential variables
Continuous

Algebraic and
Control variables
Discontinuous

Collocation over finite elements

Given: $$\frac{dz}{dt} = f(z(t), t), \quad z(0) = z_0,$$

Approximate z by Lagrange interpolation polynomials (order K+1) with interpolation points, t_k

$$\left.\begin{aligned} t &= t_{i-1} + h_i \tau, \\ z^K(t) &= \sum_{j=0}^{K} \ell_j(\tau) z_{ij}, \end{aligned}\right\} t \in [t_{i-1}, t_i], \quad \tau \in [0,1], \quad \ell_j(\tau) = \prod_{k=0, \neq j}^{K} \frac{(\tau - \tau_k)}{(\tau_j - \tau_k)},$$

Collocation Coefficients to be solved for

Evaluated at t not **τ**

Collocation Equations → $$\sum_{j=0}^{K} z_{ij} \frac{d\ell_j(\tau_k)}{d\tau} = h_i f(z_{ik}, t_{ik}), \quad k = 1, \ldots, K,$$

Known Collocation Points

Evaluated at the element boundary

Continuity Equations → $$z_{i+1,0} = \sum_{j=0}^{K} \ell_j(1) z_{ij}, \quad i = 1, \ldots, N-1$$

Collocation points

Degree K	Legendre Roots	Radau Roots
1	0.500000	1.000000
2	0.211325	0.333333
	0.788675	1.000000
3	0.112702	0.155051
	0.500000	0.644949
	0.887298	1.000000
4	0.069432	0.088588
	0.330009	0.409467
	0.669991	0.787659
	0.930568	1.000000
5	0.046910	0.057104
	0.230765	0.276843
	0.500000	0.583590
	0.769235	0.860240
	0.953090	1.000000

Table 10.1. *Shifted Gauss-Legendre and Radau roots as collocation points.*

Simple Example - Collocation

Example 10.2 (Demonstration of Orthogonal Collocation)

Consider a single differential equation:

$$\frac{dz}{dt} = z^2 - 2z + 1,\ z(0) = -3. \tag{10.16}$$

with $t \in [0, 1]$. This equation has an analytic solution given by $z(t) = (4t-3)/(4t+1)$. Using Lagrange interpolation and applying the collocation and continuity equations (10.7) and (10.14), respectively, with $K = 3$ collocation points, N elements and $h = 1/N$ leads to:

$$\sum_{j=0}^{3} z_{ij} \frac{d\ell_j(\tau_k)}{d\tau} = h(z_{ik}^2 - 2z_{ik} + 1),\ k = 1, \ldots, 3,\ i = 1, \ldots, N$$

Simple Example - Collocation

$$z_{i+1,0} = \sum_{j=0}^{3} \ell_j(1) z_{ij}, \quad i = 1, \ldots, N-1$$

$$z_f = \sum_{j=0}^{K} \ell_j(1) z_{Nj}, \; z_{1,0} = -3.$$

Using Radau collocation, we have $\tau_0 = 0$, $\tau_1 = 0.155051$, $\tau_2 = 0.644949$ and $\tau_3 = 1$.

$$\ell_0(\tau) = \frac{(\tau - \tau_1)(\tau - \tau_2)(\tau - \tau_3)}{(\tau_0 - \tau_1)(\tau_0 - \tau_2)(\tau_0 - \tau_3)} = a_3\tau^3 + a_2\tau^2 + a_1\tau + a_0$$

$$\ell_0(\tau) = -10\tau^3 + 18\tau^2 - 9\tau + 1$$

$$\dot{\ell}_0(\tau) = -30\tau^2 + 36\tau - 9$$

Other Lagrange polynomials found similarly

Simple Example - Collocation

For $N = 1$, and $z_0 = -3$ the collocation equations are given by:

$$\sum_{j=0}^{3} z_j \frac{d\ell_j(\tau_k)}{d\tau} = (z_k^2 - 2z_k + 1),\ k = 1, \ldots, 3,$$

which can be written out as:

$$\begin{aligned} &z_0(-30\tau_k^2 + 36\tau_k - 9) + z_1(46.7423\tau_k^2 - 51.2592\tau_k + 10.0488) \\ &+ z_2(-26.7423\tau_k^2 + 20.5925\tau_k - 1.38214) + z_3(10\tau_k^2 - \frac{16}{3}\tau_k + \frac{1}{3}) \\ &= (z_k^2 - 2z_k + 1),\ k = 1, \ldots, 3. \end{aligned}$$

Solving these three equations gives $z_1 = -1.65701$, $z_2 = 0.032053$, $z_3 = 0.207272$

Collocation Matrix

$$\sum_{j=0}^{3} z_j \frac{d\ell_j(\tau_k)}{d\tau} = (z_k^2 - 2z_k + 1),\ k = 1, \ldots, 3,$$

Constant

which can be written out as:

$$z_0(-30\tau_k^2 + 36\tau_k - 9) + z_1(46.7423\tau_k^2 - 51.2592\tau_k + 10.0488)$$
$$+ z_2(-26.7423\tau_k^2 + 20.5925\tau_k - 1.38214) + z_3(10\tau_k^2 - \frac{16}{3}\tau_k + \frac{1}{3})$$
$$= (z_k^2 - 2z_k + 1),\ k = 1, \ldots, 3.$$

- $adot(k,k) = \begin{bmatrix} \dot{\ell}_0(\tau_0) & \cdots & \dot{\ell}_0(\tau_k) \\ \vdots & \ddots & \vdots \\ \dot{\ell}_k(\tau_0) & \cdots & \dot{\ell}_k(\tau_k) \end{bmatrix}$

Python code for generating collocation matrix

```
import numpy
# Specify collocation points
cp = [0, 0.155051, 0.644949, 1]

a = []
for i in range(len(cp)):
    ptmp = []
    tmp = 0
    for j in range(len(cp)):
        if j != i:
            row = []
            row.insert(0,1/(cp[i]-cp[j]))
            row.insert(1,-cp[j]/(cp[i]-cp[j]))
            ptmp.insert(tmp,row)
            tmp += 1
    p=[1]
    for j in range(len(cp)-1):
        p = numpy.convolve(p,ptmp[j])
    pder = numpy.polyder(p,1)
    arow = []
    for j in range(len(cp)):
        arow.append(numpy.polyval(pder,cp[j]))
    a.append(arow)
    print(arow)
```

Simple Example - Collocation

$$\frac{dz}{dt} = z^2 - 2z + 1, \qquad z(0) = -3$$

- Exercise: Solve the differential equation using collocation over a single finite element with $t \in [0,1]$

$$\left.\begin{aligned} t &= t_{i-1} + h_i \tau, \\ z^K(t) &= \sum_{j=0}^{K} \ell_j(\tau) z_{ij}, \end{aligned}\right\} t \in [t_{i-1}, t_i], \quad \tau \in [0,1], \quad \ell_j(\tau) = \prod_{k=0, \neq j}^{K} \frac{(\tau - \tau_k)}{(\tau_j - \tau_k)},$$

Collocation Equations $\rightarrow$

$$\sum_{j=0}^{K} z_{ij} \frac{d\ell_j(\tau_k)}{d\tau} = h_i f(z_{ik}, t_{ik}), \quad k = 1, \ldots, K,$$

- Exercise: Plot the solution against the analytic solution

Analytic solution $z(t) = (4t - 3)/(4t + 1)$

Implementation is challenging!

- Common theme: significant effort to rework formulation
  - Time: first ~6 months of a grad student's research
  - Error prone: many ways to make subtle mistakes
  - Inflexible: formulation specific to selected solution approach

- Difficult to know apriori the best solution approach for a particular model

Expressing dynamical systems

[with J. Siirola, V. Zavala]

- Model dynamical systems in a natural form
  - Systems of Differential Algebraic Equations (DAE)
  - Extend the Pyomo component model
    - ContinuousSet: A virtual set over which you can take a derivative
    - DerivativeVar: The derivative of a Var with respect to a ContinuousSet

$$
\begin{aligned}
& \min \Psi(x(t), y(t), u(t)) \\
& \dot{x}(t) = f(x(t), y(t), u(t), t, p) \\
\text{DAE} \quad & 0 = g(x(t), y(t), u(t), t, p) \\
\text{model} \quad & x(t) \in \Re^n \\
& y(t) \in \Re^n \\
& u(t) \in \Re^m
\end{aligned}
$$

Simple Example – pyomo.dae

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

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

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

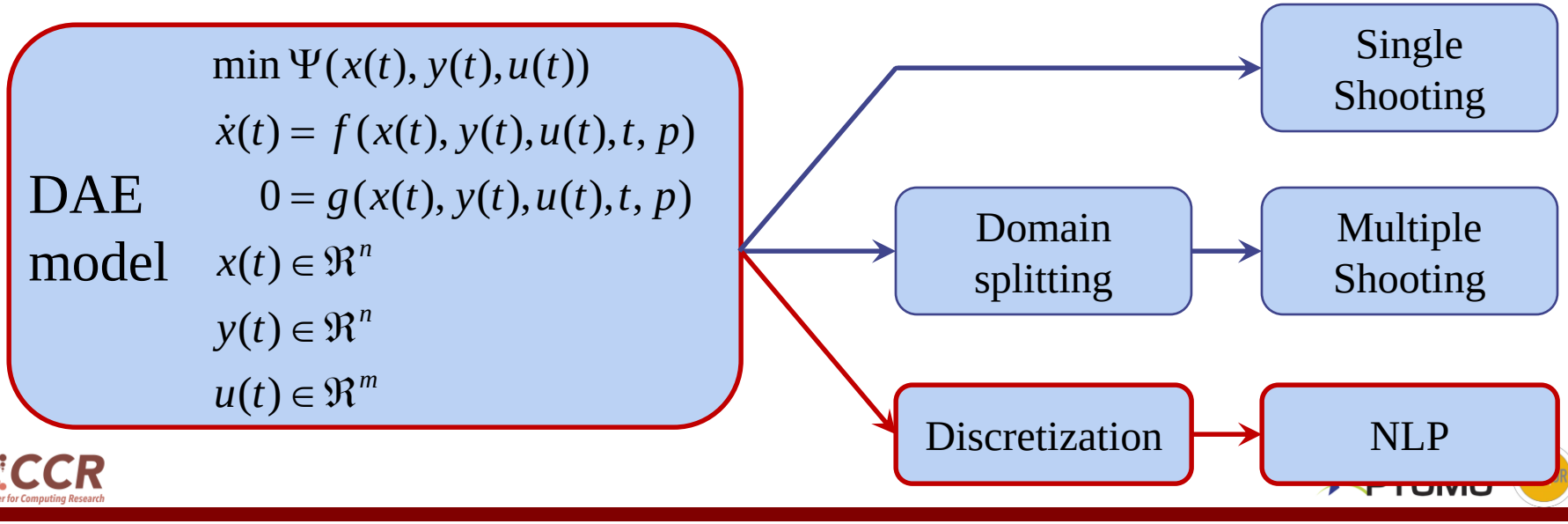
m.obj = Objective(expr=1) # Dummy Objective

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

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

Solving dynamical systems

- Given that we have a DAE model in Pyomo… now what?
  - How to optimize?
    - Simulation-based / Multiple shooting methods / Simultaneous
  - Common theme: significant effort to rework formulation
    - Time / Error prone / Inflexible
- Our approach: separate the *declaration* of dynamical models from the *solution approach* using (nearly) *automatic transformations*



Simple Example – pyomo.dae

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

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

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

m.obj = Objective(expr=1) # Dummy Objective

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

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

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

```
# Discretize model using radau collocation
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m,nfe=1,ncp=3,scheme='LAGRANGE-RADAU')
```

Small example: optimal control (1/8)

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

[Jacobson and Lele (1969)]

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

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

m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
m.x2    = Var(m.t)
m.x3    = Var(m.t)

m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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

Small example: optimal control (2/8)

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

[Jacobson and Lele (1969)]

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

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

m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
```


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


```
m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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

Small example: optimal control (3/8)

$$
\begin{aligned}
\min \quad & x_3(t_f) \\
s.t \quad & \dot{x}_1 = x_2 \\
& \dot{x}_2 = -x_2 + u \\
& \dot{x}_3 = x_1^2 + x_2^2 + 0.005 * u^2 \\
& x_2 - 8 * (t - 0.5)^2 + 0.5 \leq 0 \\
& x_1(0) = 0 \\
& x_2(0) = -1 \\
& x_3(0) = 0 \\
& t_f = 1
\end{aligned}
$$

[Jacobson and Lele (1969)]

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

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

m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
```


```
model = m = ConcreteModel()
m.tf    = Param(initialize = 1)
m.t     = ContinuousSet(bounds=(0, m.tf))
```


```
def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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

Small example: optimal control (4/8)

$$\min \; x_3(t_f)$$
$$s.t. \quad \dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_2 + u$$
$$\dot{x}_3 = x_1^2 + x_2^2 + 0.005 * u^2$$
$$x_2 - 8 * (t -$$
$$x_1($$
$$x_2(0$$
$$x_3($$
$$t_f$$

[Jacobson a

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

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

m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
m.x2    = Var(m.t)
m.x3    = Var(m.t)

m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
```


```
m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
m.x2    = Var(m.t)
m.x3    = Var(m.t)

m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)
```


```
    yield m.x2[0] == -1
    yield m.x3[0] == 0
m.init_conditions = ConstraintList(rule=_init)
```

Small example: optimal control (5/8)

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

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

[Jacobson and Lele (1969)]

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

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

m.u     = Var(m.t, initialize=0)
m.x1    = Var(m.t)
m.x2    = Var(m.t)
m.x3    = Var(m.t)

m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
```


```
m.obj = Objective(expr=m.x3[m.tf])
```


```
def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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

Small example: optimal control (6/8)

Sandia National Laboratories

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

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

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

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

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

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

$$t_f = 1$$

[Jacobson and Lele (1969)]

```
from pyomo.environ import *

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)
```

```
m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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

CCR Center for Computing Research

PYOMO COIN-OR

Small example: optimal control (7/8)

Sandia National Laboratories

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

[Jacobson and Lele (1969)]

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

model = m = ConcreteModel()

m.t = ContinuousSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)
m.tf = Param(initialize=1)

m.dx1dt = DerivativeVar(m.x1, wrt=m.t)
m.dx2dt = DerivativeVar(m.x2, wrt=m.t)
m.dx3dt = DerivativeVar(m.x3, wrt=m.t)

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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

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


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


CCR Center for Computing Research

PYOMO COIN-OR

Small example: optimal control (8/8)

Sandia National Laboratories

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

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

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

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

$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$

$$t_f = 1$$

[Jacobson and Lele (1969)]

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

model = m = ConcreteModel()

m.obj = Objective(expr=m.x3[m.tf])

def _x1dot(m, t):
    return m.dx1dt[t] == m.x2[t]
m.x1dot = Constraint(m.t, rule=_x1dot)

def _x2dot(m, t):
    return m.dx2dt[t] == -m.x2[t] + m.u[t]
m.x2dot = Constraint(m.t, rule=_x2dot)

def _x3dot(m, t):
    return m.dx3dt[t] == m.x1[t]**2 + \
        m.x2[t]**2 + 0.005*m.u[t]**2
m.x3dot = Constraint(m.t, rule=_x3dot)

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


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


CCR Center for Computing Research

Optimal Control Example - Script

```
from pyomo.environ import *
# Import dynamic model
from optimalControl import m

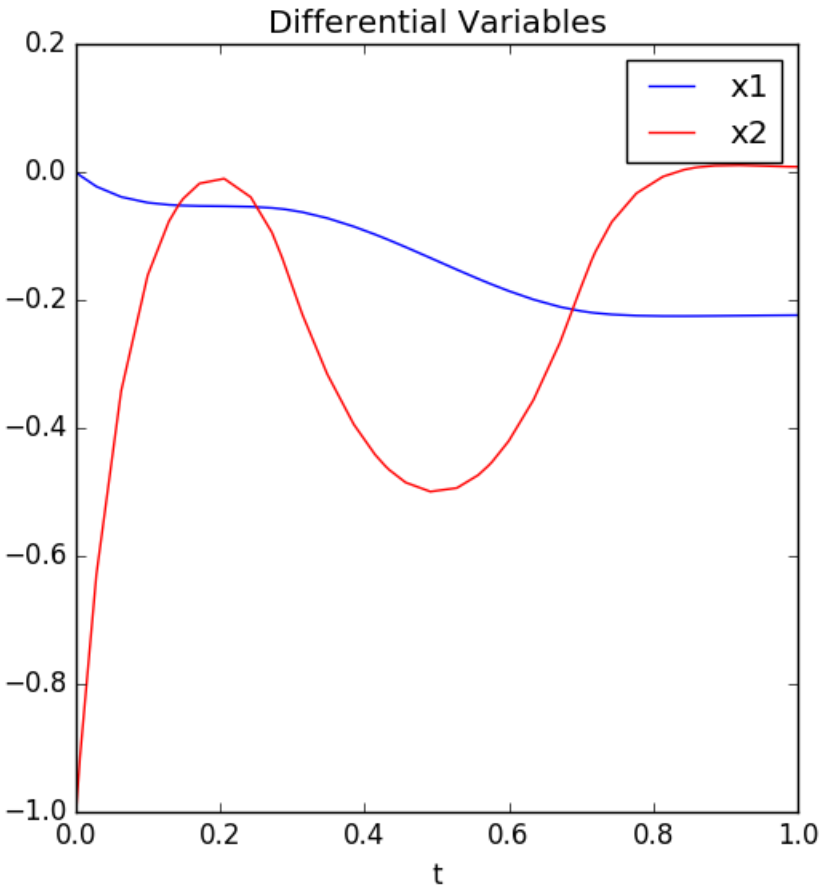
# Discretize model using radau collocation
TransformationFactory('dae.collocation').apply_to(
    m, nfe=7, ncp=6, scheme='LAGRANGE-RADAU' )

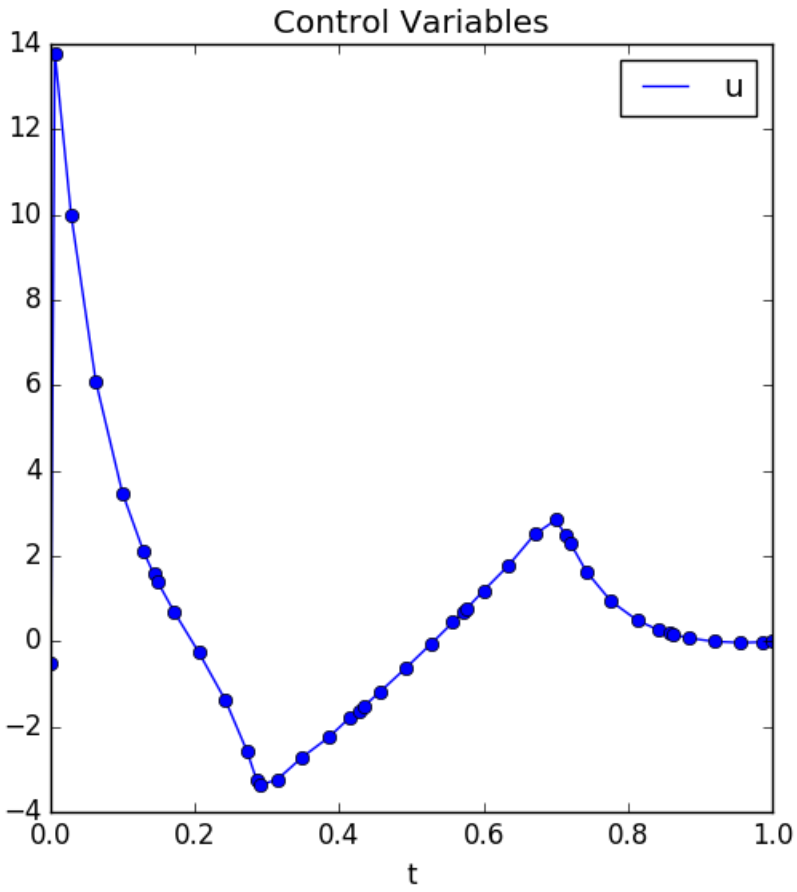
# Solve algebraic model
results = SolverFactory('ipopt').solve(m)

def plotter(subplot, x, *series, **kwds):
    plt.subplot(subplot)
    for i,y in enumerate(series):
        plt.plot(x, [value(y[t]) for t in x], 'brgcmk'[i%6]+kwds.get('points','')
    plt.title(kwds.get('title',''))
    plt.legend(tuple(y.cname() for y in series))
    plt.xlabel(x.cname())

import matplotlib.pyplot as plt
plotter(121, m.t, m.x1, m.x2, title='Differential Variables')
plotter(122, m.t, m.u, title='Control Variable', points='o')
plt.show()
```

Optimal Control Example - Results





Optimal Control Example - Script

```
from pyomo.environ import *
# Import dynamic model
from optimalControl import m

# Discretize model using radau collocation
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to( m, nfe=7, ncp=6, scheme='LAGRANGE-RADAU' )

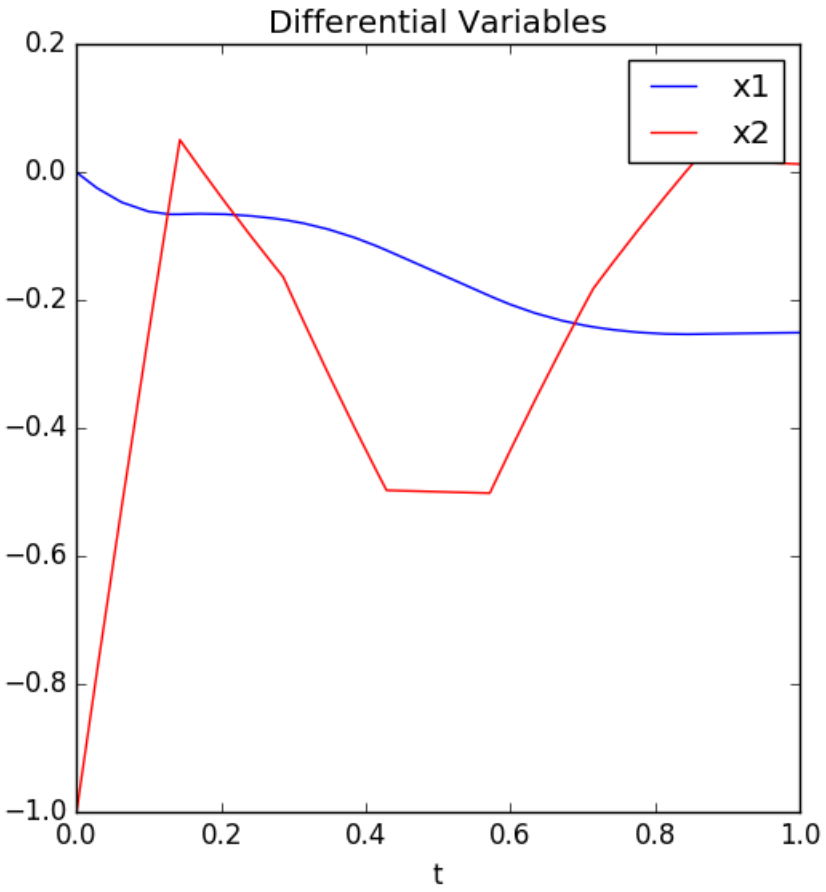
# Control variable u made constant over each finite element
discretizer.reduce_collocation_points(var=m.u, ncp=1, contset=m.t)

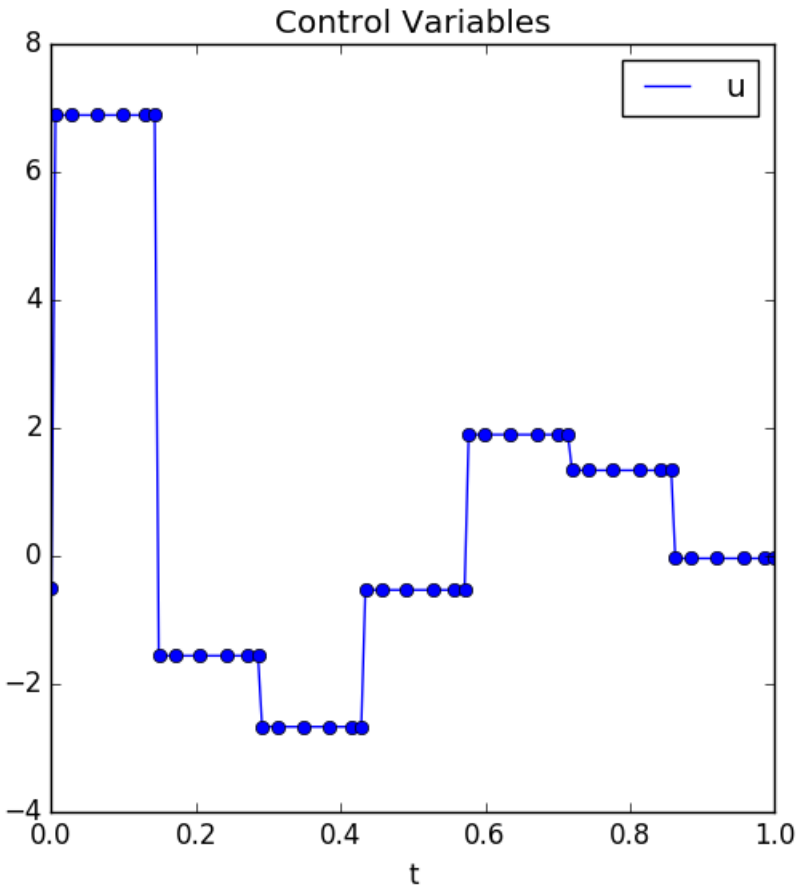
# Solve algebraic model
results = SolverFactory('ipopt').solve(m)

def plotter(subplot, x, *series, **kwds):
    plt.subplot(subplot)
    for i,y in enumerate(series):
        plt.plot(x, [value(y[t]) for t in x], 'brgcmk'[i%6]+kwds.get('points','')
    plt.title(kwds.get('title',''))
    plt.legend(tuple(y.cname() for y in series))
    plt.xlabel(x.cname())

import matplotlib.pyplot as plt
plotter(121, m.t, m.x1, m.x2, title='Differential Variables')
plotter(122, m.t, m.u, title='Control Variable', points='o')
plt.show()
```

Optimal Control Example - Results 2





Parameter Estimation Example 1

$$\min \sum_{t_i} \left(x_1(t_i) - x_{1_{meas}}(t_i) \right)^2$$

$$s.t. \quad \frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = 1 - 2x_2 - x_1$$

$$-1.5 \leq p_1, p_2 \leq 1.5$$

$$x_1(0) = p_1, \qquad x_2(0) = p_2$$

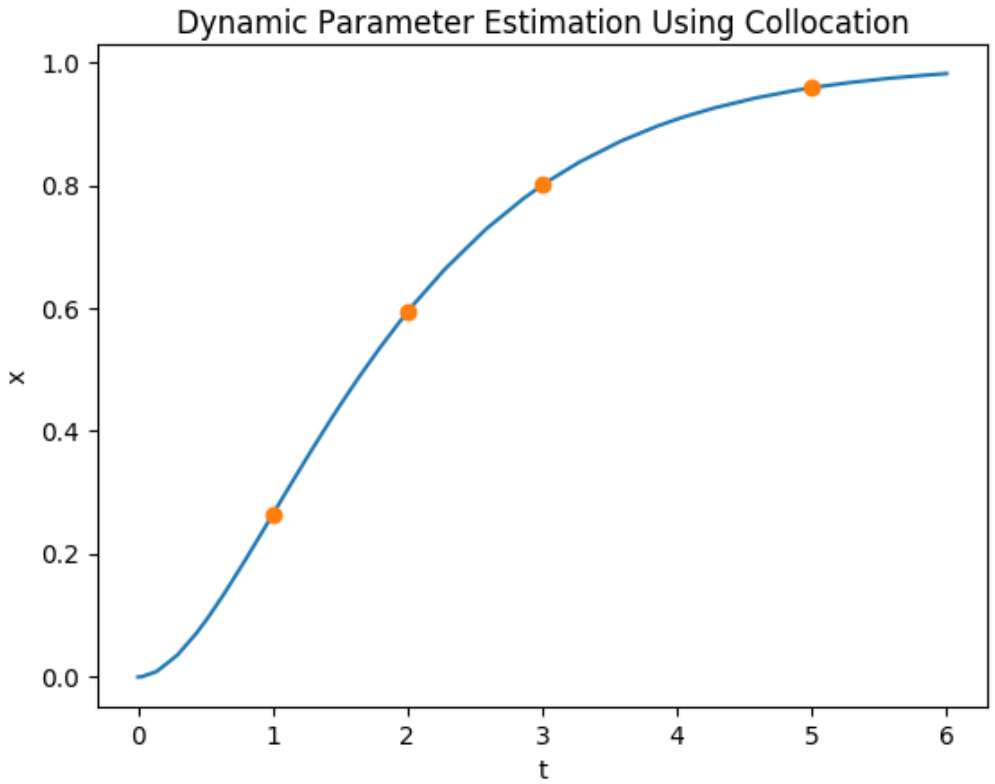
Time	1	2	3	5
$x_{1_{meas}}$	0.264	0.594	0.801	0.959

Parameter Estimation Example 1

Sandia National Laboratories

data file

```
set t := 0 1 2 3 5 6 ;
set MEAS_t := 1 2 3 5 ;
param x1_meas :=
1 0.264
2 0.594
3 0.801
5 0.959
;
```



CCR Center for Computing Research

PYOMO COIN-OR

Parameter Estimation Example 2

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$

$$\frac{dA}{dt} = -k_1 A$$

$$\frac{dB}{dt} = k_1 A - k_2 B$$

$$A(0) = 1, \qquad B(0) = 0$$

Time	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
A	0.606	0.368	0.223	0.135	0.082	0.050	0.030	0.018	0.011	0.007
B	0.373	0.564	0.647	0.669	0.656	0.624	0.583	0.539	0.494	0.451

- Exercise: Solve for k_1 and k_2 given the concentration measurements in the table

Disease Transmission Example

- Parameter estimation of a S-I-R disease transmission model[*]

$$
\begin{aligned}
\min \quad & \omega_M \sum_{i\in\mathcal{F}} (\ln(\varepsilon_{M_i}))^2 + \omega_Q \sum_{k\in\mathcal{T}} (\varepsilon_{Q_k})^2 \\
\text{s.t.} \quad & \frac{\mathrm{d}S}{\mathrm{d}t} = \frac{-\beta(y(t))S(t)I(t)}{N(t)} \cdot \varepsilon_M(t) + B(t), \\
& \frac{\mathrm{d}I}{\mathrm{d}t} = \frac{\beta(y(t))S(t)I(t)}{N(t)} \cdot \varepsilon_M(t) - \gamma I(t), \\
& \frac{\mathrm{d}Q}{\mathrm{d}t} = \frac{\beta(y(t))S(t)I(t)}{N(t)} \cdot \varepsilon_M(t), \\
& R_k^\star = \eta_k(Q_{i,k} - Q_{i,k-1}) + \varepsilon_{Q_k}, \\
& \bar{S} = \frac{\sum_{i\in\mathcal{F}} S_i}{\mathrm{len}(\mathcal{F})}, \\
& \bar{\beta} = \frac{\sum_{i\in\tau} \beta_i}{\mathrm{len}(\tau)}, \\
& 0 \le I(t), S(t) \le N(t) \\
\text{and} \quad & 0 \le \beta(y(t)), Q(t),
\end{aligned}
$$

Discretized problem:

- 3 differential equations
- 520 finite elements
- 3 collocation points

~ 10,500 variables

~ 10,000 constraints

[*] Word, Cummings, Burke, Iamsirithaworn, and Laird, *Journal of the Royal Society Interface*, 2012, pp. 1983-1997.

Performance impact

- Comparing the automated discretization to a manually discretized model implemented (and tuned) by a person

	Manual Discretization	Using pyomo.dae (Radau Collocation)
Model Creation Time (CPU secs)	1.79	5.29
Solve Time (CPU secs)	1.35	0.86
IPOPT Iterations	27	26
Objective ($x10^{-5}$)	1.4716	1.4716

- The bulk of the added model creation time is the model transformation that implements the discretization
- The solve time difference likely due to a more sophisticated initialization scheme in the manual discretization model

Distillation Example

```
model.S_TRAYS = Set()
model.S_RECTIFICATION = Set(within = model.S_TRAYS)
model.S_STRIPPING = Set(within = model.S_TRAYS)

model.t = ContinuousSet(initialize=range(1,52))
model.x = Var(model.S_TRAYS, model.t, initialize=x_init_rule)
model.dx = DerivativeVar(model.x)



def _diffeq(m,n,t):
    if t == 1:
        return Constraint.Skip
    if n == 1:
        return m.dx[n,t] == 1/m.acond*m.V[t]*(m.y[n+1,t]-m.x[n,t])
    elif n in m.S_RECTIFICATION:
        return m.dx[n,t] == 1/m.atray*(m.L[t]*(m.x[n-1,t]-m.x[n,t])-m.V[t]*(m.y[n,t]-m.y[n+1,t]))
    elif n == 17:
        return m.dx[n,t] == 1/m.atray*(m.Feed*m.x_Feed+m.L[t]*m.x[n-1,t]-m.FL[t]*m.x[n,t]- \
     m.V[t]*(m.y[n,t]-m.y[n+1,t]))
    elif n in m.S_STRIPPING:
        return m.dx[n,t] == 1/m.atray*(m.FL[t]*(m.x[n-1,t]-m.x[n,t])-m.V[t]*(m.y[n,t]-m.y[n+1,t]))
    else :
        return m.dx[n,t] == 1/m.areb*(m.FL[t]*m.x[n-1,t]-(m.Feed-m.D)*m.x[n,t]-m.V[t]*m.y[n,t])
model.diffeq = Constraint(model.S_TRAYS, model.t, rule=_diffeq)
```

PDE Example

- Illustrative example[1]
  - PDE

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

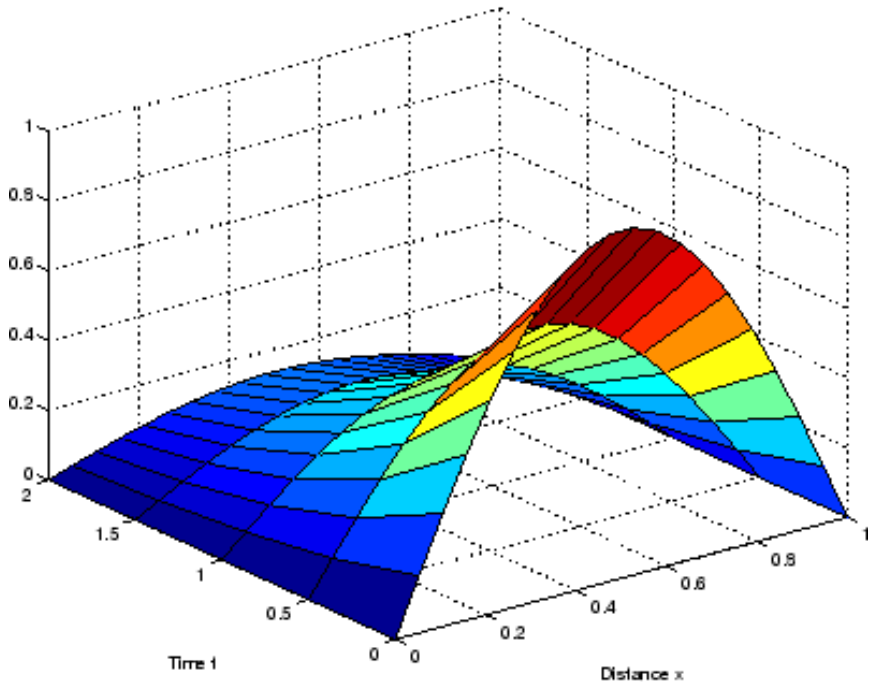
  - Initial Condition

$$u(x,0) = \sin(\pi x)$$

  - Boundary Conditions

$$u(0,t) = 0$$

$$\pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0$$



[1] Example 1 from http://www.mathworks.com/help/matlab/ref/pdepe.html

PDE Example

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

m = ConcreteModel()
m.pi = Param(initialize=3.1416)
m.t = ContinuousSet(bounds=(0,2))
m.x = ContinuousSet(bounds=(0,1))
m.u = Var(m.x,m.t)

# Declare derivatives in the model
m.dudx = DerivativeVar(m.u,wrt=m.x)
m.dudx2 = DerivativeVar(m.u,wrt=(m.x,m.x))
m.dudt = DerivativeVar(m.u,wrt=m.t)
```

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad u(x,0) = \sin(\pi x) \qquad u(0,t) = 0 \qquad \pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0$$

PDE Example

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right)$$

```
def _pde(m,i,j):
    if i == 0 or i == 1 or j == 0 :
        return Constraint.Skip
    return m.pi**2*m.dudt[i,j] == m.dudx2[i,j]
m.pde = Constraint(m.x,m.t,rule=_pde)
```

$$u(x,0) = \sin(\pi x)$$

```
def _initcon(m,i):
    if i == 0 or i == 1:
        return Constraint.Skip
    return m.u[i,0] == sin(m.pi*i)
m.initcon = Constraint(m.x,rule=_initcon)
```

$$u(0,t) = 0$$

```
def _lowerbound(m,j):
    return m.u[0,j] == 0
m.lowerbound = Constraint(m.t,rule=_lowerbound)
```

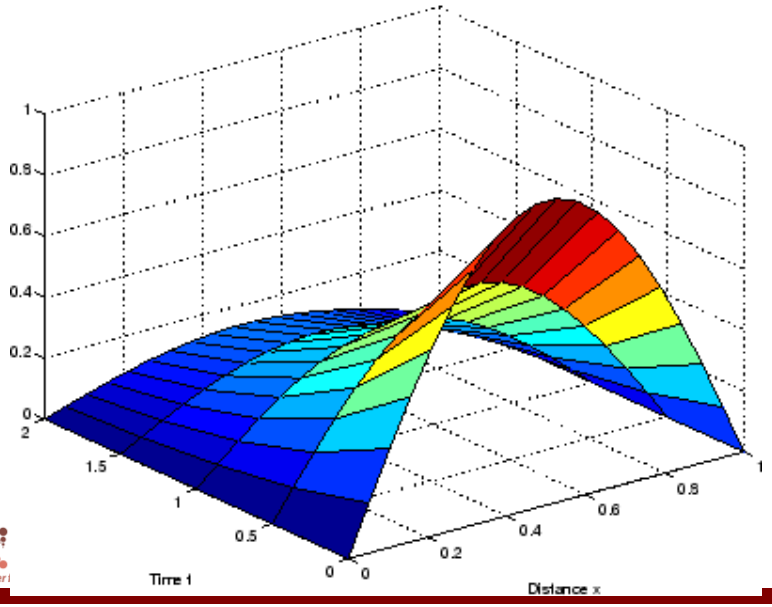
$$\pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0$$

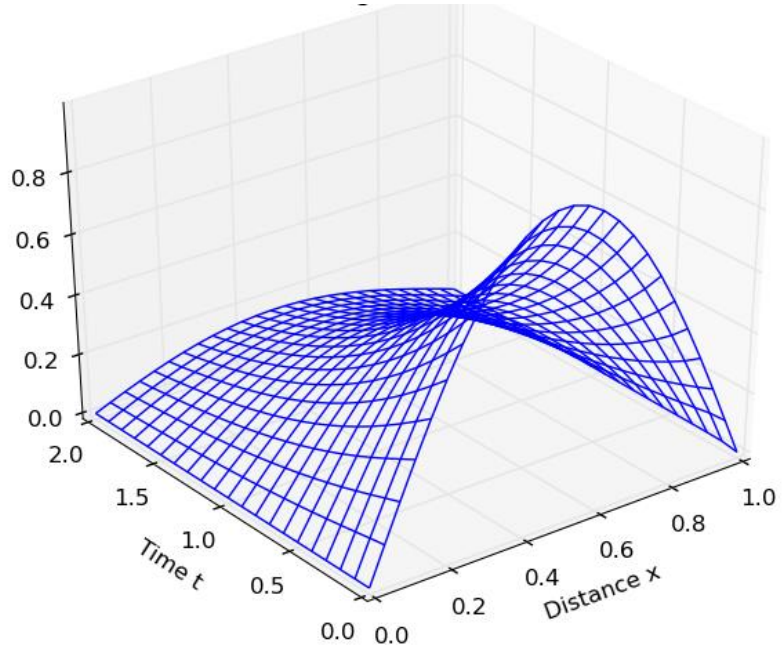
```
def _upperbound(m,j):
    return m.pi*exp(-j)+m.dudx[1,j] == 0
m.upperbound = Constraint(m.t,rule=_upperbound)
```

PDE Example

```
# Discretize using Finite Difference Method
discretizer = TransformationFactory('dae.finite_difference')
discretizer.apply_to(m,nfe=25,wrt=m.x,scheme='BACKWARD')
discretizer.apply_to(m,nfe=20,wrt=m.t,scheme='BACKWARD')

solver = SolverFactory('ipopt')
results = solver.solve(m,tee=True)
```





Upcoming Features

- Interface to integrator/DAE solver for:
  - Model simulation
  - Model initialization
  - Implementation of multiple shooting
- Interpolation tools for:
  - Model initialization

Building Expert Frameworks with pyomo.dae

- Nonlinear model predictive control
- Reaction kinetics

Nonlinear Model Predictive Control

Sandia National Laboratories

```
## NMPC
from iNMPC import *
CONTROLLER = IdealNMPC(model=m, model_time=m.t, prediction_horizon=50,
    control_horizon=25, simulation_periods=50, default_discretization=True)
#CONTROLLER.discretize_model(discretization_type='OrthogonalCollocation',
#                            ncp=5, wrt=m.t, scheme='LAGRANGE-LEGENDRE')
CONTROLLER.solve('ipopt', options=[('linear_solver', 'ma57')],
    file_name='test1', dynamic_plot=True)


## asNMPC
from asNMPC import *
CONTROLLER = AdvancedStepNMPC(model=m, model_time=m.t, prediction_horizon=50,
    control_horizon=25, simulation_periods=50, default_discretization=True)
CONTROLLER.solve('ipopt_sens', options=[('linear_solver', 'ma57'),
                ('run_sens','yes')], file_name='test1', real_plant = plant,
                dynamic_plot=True)
```

[Control frameworks developed by Federico Lozano Santamaría Universidad de los Andes, Bogotá, Colombia]

CCR Center for Computing Research

PYOMO COIN-OR

Reaction Kinetics

- Chemical reaction kinetics: $2A \xrightarrow{k_1} B; \quad B \xrightarrow{k_2} C$

- Reaction rate:

$$r_1 = k_1 c_A^2 \qquad k_1 = A_1 e^{\left(\frac{E_a}{R \cdot T}\right)}$$

$$r_2 = k_2 c_B \qquad k_2 = A_2 e^{\left(\frac{E_a}{R \cdot T}\right)}$$

- Stoichiometry

$$\frac{dc_A}{dx} = -2r_1 k_1$$

$$\frac{dc_B}{dx} = r_1 k_1 - r_2 k_2$$

$$\frac{dc_C}{dx} = r_2 k_2$$

Example from: http://www.comsol.com/blogs/general-introduction-chemical-kinetics-arrhenius-law/

General purpose kinetic model

```
def create_kinetic_model(rxnNet, time):
    model = ConcreteModel()
    model.SPECIES   = Set( initialize=rxnNet.species() )
    model.REACTIONS = Set( initialize=rxnNet.reactions.keys() )
    model.TIME      = ContinuousSet( bounds=(0,max(time)), initialize=time )
    model.c         = Var( model.TIME, model.SPECIES, bounds=(0,None) )
    model.dcdt      = DerivativeVar( model.c, wrt=model.TIME )
    model.k         = Var( model.REACTIONS, bounds=(0,None) )
    model.rate      = Var( model.TIME, model.REACTIONS )
    def reaction_rate(m, t, r):
        rhs = m.k[r]
        for s, coef in iteritems(m.rxnNetwork.reactions[r].reactants):
            rhs *= m.c[t,s]**coef
        return m.rate[t,r] == rhs
    model.reaction_rate = Constraint( model.TIME, model.REACTIONS, rule=reaction_rate )
    def stoichiometry(m, t, s):
        rhs = 0
        for r in m.REACTIONS:
            if s in m.rxnNetwork.reactions[r].reactants:
                rhs -= m.rate[t,r] * m.rxnNetwork.reactions[r].reactants[s]
            if s in m.rxnNetwork.reactions[r].products:
                rhs += m.rate[t,r] * m.rxnNetwork.reactions[r].products[s]
        return m.dcdt[t,s] == rhs
    model.stoichiometry = Constraint( model.TIME, model.SPECIES, rule=stoichiometry )
    return model
```

Step 1: simulation

```
fdiff = TransformationFactory("dae.finite_difference")
rxns = ReactionNetwork()
rxns.add( Reaction("AtoB", reactants=['2*A'], products=['B']) )
rxns.add( Reaction("BtoC", reactants=['B'], products=['C']) )

model = create_kinetic_model(rxns, 60*60)

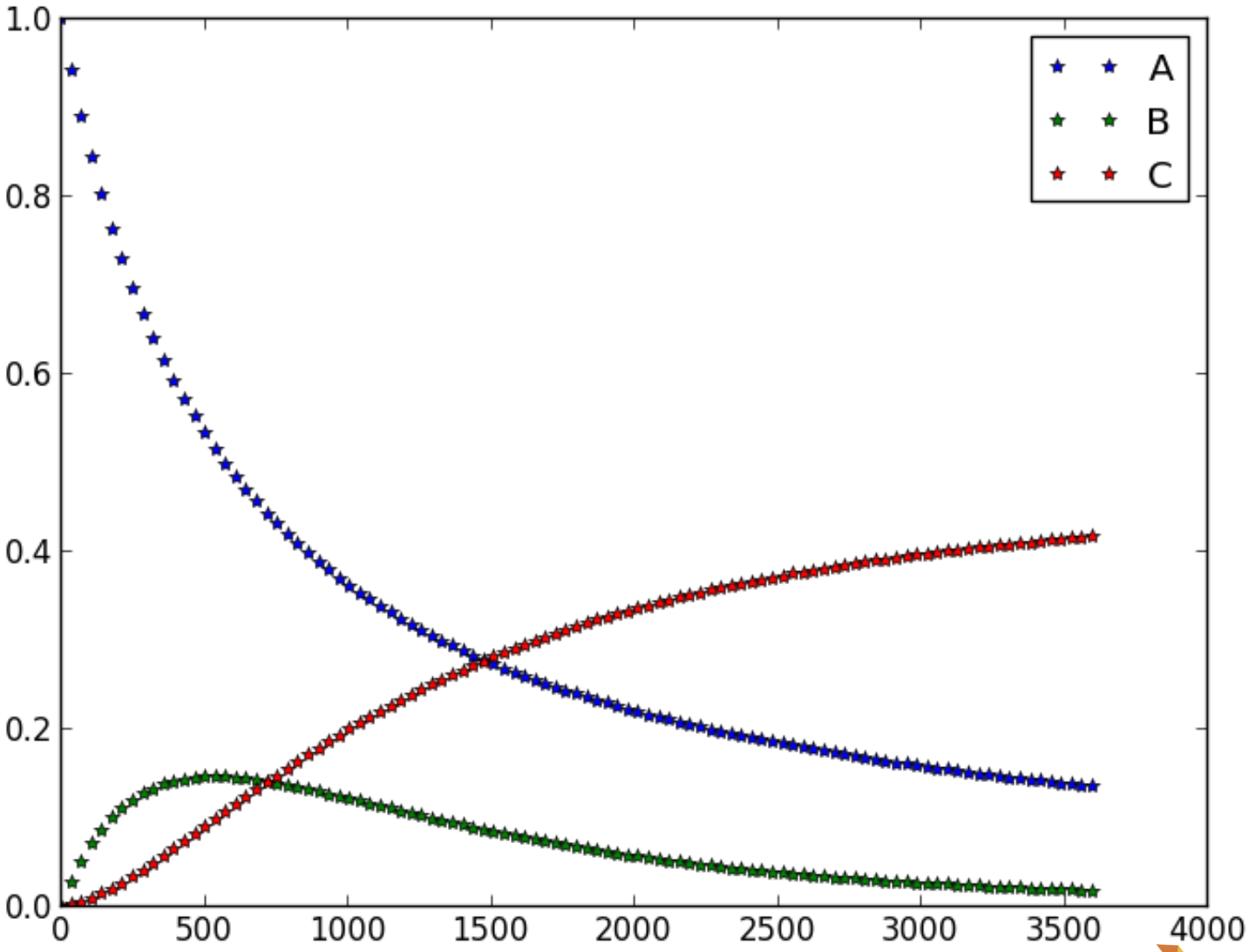
A1  = 1.32e19 # L / mol*s
A2  = 1.09e13 # 1/s
Ea1 = 140000  # J/mol
Ea2 = 100000  # J/mol
R   =  8.314  # J / K*mol
T   =    330  # K
model.k['AtoB'].fix( A1 * exp( -Ea1 / (R*T) ) )
model.k['BtoC'].fix( A2 * exp( -Ea2 / (R*T) ) )

model.c[0, 'A'].fix(1)
model.c[0, 'B'].fix(0)
model.c[0, 'C'].fix(0)

fdiff.apply_to(model, nfe=100)
solver.solve(model)
plot_results(model)
```

```
def plot_results(model):
    if plt is not None:
        _tmp = sorted(iteritems(model.c))
        for _i, _x in enumerate('ABC'):
            plt.plot([x[0][0] for x in _tmp if x[0][1] == _x],
                     [value(x[1]) for x in _tmp if x[0][1] == _x],
                     'bgr'[_i]+'*', label=_x)
        plt.legend()
        plt.show()
```

Step 1: results



Step 2: maximize the production of “B”

```
fdiff = TransformationFactory("dae.finite_difference")
rxns = ReactionNetwork()
rxns.add( Reaction("AtoB", reactants=['2*A'], products=['B']) )
rxns.add( Reaction("BtoC", reactants=['B'], products=['C']) )

model = create_kinetic_model(rxns, 60*60)

A1  = 1.32e19 # L / mol*s
A2  = 1.09e13 # 1/s
Ea1 = 140000  # J/mol
Ea2 = 100000  # J/mol
R   =  8.314  # J / K*mol

model.T   =    Var(bounds=(0,None), initialize=330)  # K
def compute_k(m):
    yield m.k['AtoB'] == A1 * exp( -Ea1 / (R*m.T) )
    yield m.k['BtoC'] == A2 * exp( -Ea2 / (R*m.T) )
model.compute_k = ConstraintList(rule=compute_k)

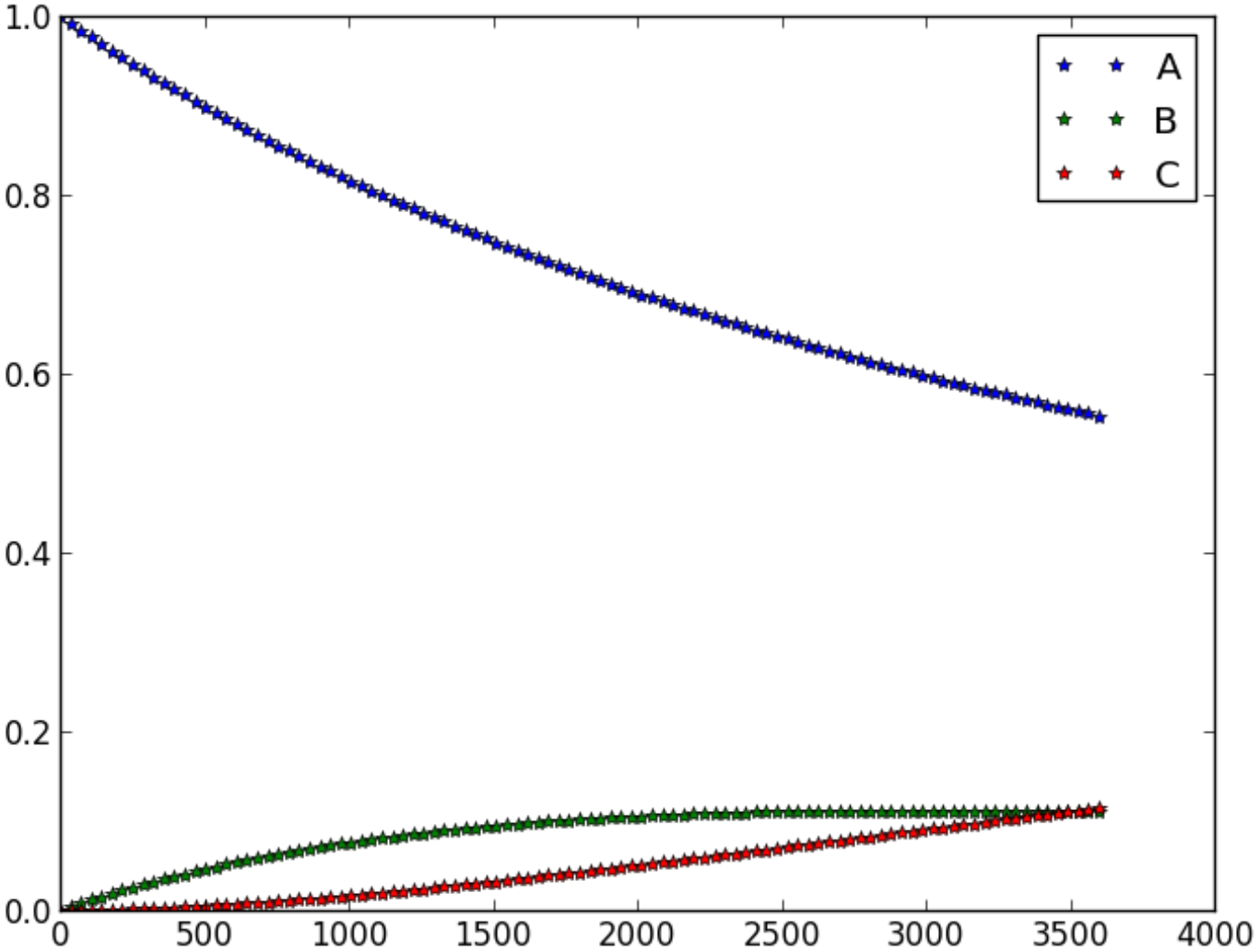
model.c[0, 'A'].fix(1)
model.c[0, 'B'].fix(0)
model.c[0, 'C'].fix(0)

fdiff.apply_to(model, nfe=100)

model.obj = Objective( sense=maximize, expr=model.c[max(model.TIME), 'B'])

solver.solve(model)
plot_results(model)
```

Step 2: maximize the production of “B”



Kinetic parameter regression

- (Assumed) Chemical reaction kinetics:

$$\mathrm{TG} + \mathrm{MeOH} \overset{k_1}{\leftrightarrow} \mathrm{DG} + \mathrm{FAME}$$

$$\mathrm{DG} + \mathrm{MeOH} \overset{k_2}{\leftrightarrow} \mathrm{MG} + \mathrm{FAME}$$

$$\mathrm{MG} + \mathrm{MeOH} \overset{k_3}{\leftrightarrow} \mathrm{Glycerol} + \mathrm{FAME}$$

- Given experimental data, estimate the reaction rate constants

Example from: http://www.doiserbia.nb.rs/img/doi/0367-598X/2014/0367-598X1300037A.pdf

Generate (sub)model for each experiment…

```
def create_regression_model(b, t):
    rxns = ReactionNetwork()
    rxns.add_reversible(
        Reaction( "k_1", reactants=['TG','MeOH'], products=['DG','FAME'] ) )
    rxns.add_reversible(
        Reaction( "k_2", reactants=['DG','MeOH'], products=['MG','FAME'] ) )
    rxns.add_reversible(
        Reaction( "k_3", reactants=['MG','MeOH'], products=['Glycerol','FAME'] ) )

    data = b.model().data[t]
    key = b.model().key

    model = create_kinetic_model(rxns, data.keys())

    model.error = Var(bounds=(0,None))

    model.compute_error = Constraint(
        expr= model.error == sum(
            (( model.c[t,key[i]] - x ) / max(data[_t][i] for _t in data) )**2
            for t in data for i,x in enumerate(data[t]) ) )

    return model
```

Assemble the regression model and solve

```
model = ConcreteModel()
model.key = key = ('MeOH','TG','DG','MG','FAME','Glycerol')
model.data = data = { 150: { 0:    (2.833,6.84E-02,0.00,0.00,0.00,0.00,),
                             256:  (2.807,4.75E-02,1.51E-02,3.71E-03,2.60E-02,8.18E-04,), # ...
                     } 210: { # ...
                     } }

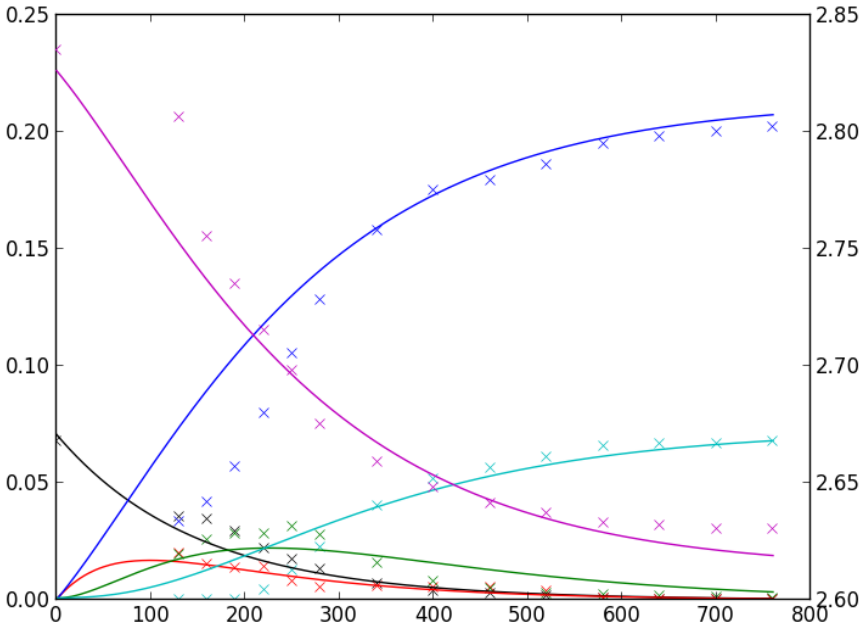
model.experiment = Block( data.keys(), rule=create_regression_model )
model.obj = Objective( sense=minimize,
                       expr=sum(b.error for b in model.experiment[:]) )

# initializations from the paper
for exp in model.experiment[:]:
    exp.k['k_1']   = 7.58e-7
    exp.k['k_1_r'] = 0
    exp.k['k_2']   = 2.20e-7
    exp.k['k_2_r'] = 0
    exp.k['k_3']   = 2.15e-7
    exp.k['k_3_r'] = 0

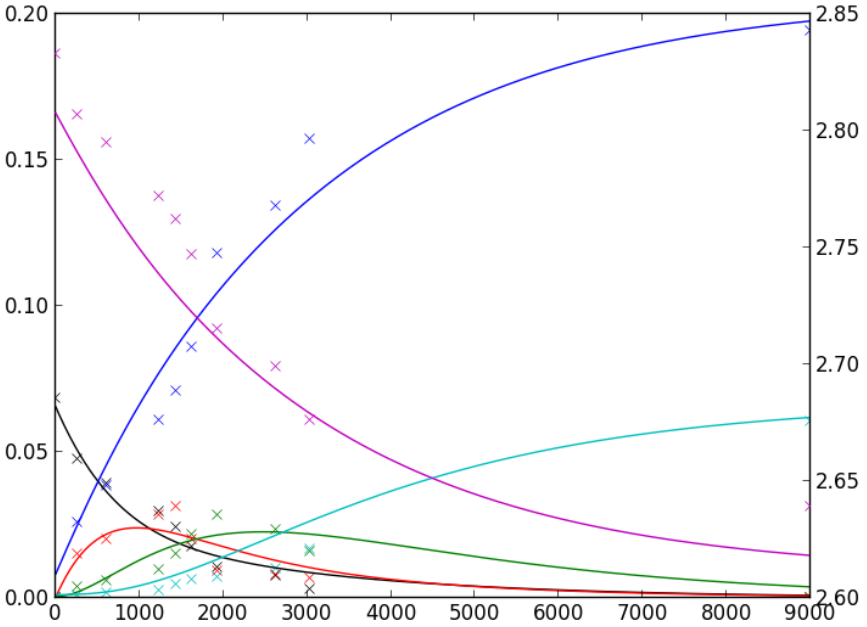
colloc.apply_to(model, nfe=100, ncp=3)
solver.solve(model, tee=True)
plot_regression_results(model)
```

Regression results

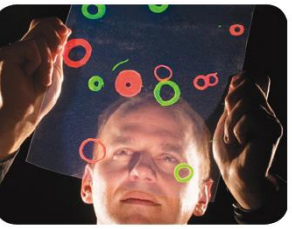
T=150

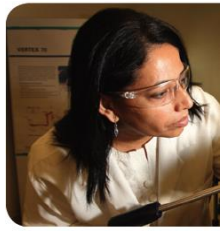


T=210



- 10850 variables, 10826 constraints
- Total time: 6.4 seconds
  - 2.8 seconds for model generation and processing
  - 3.6 seconds in solver (ipopt)





8. Solving Stochastic Programs

Sandia National Laboratories

Exceptional service in the national interest

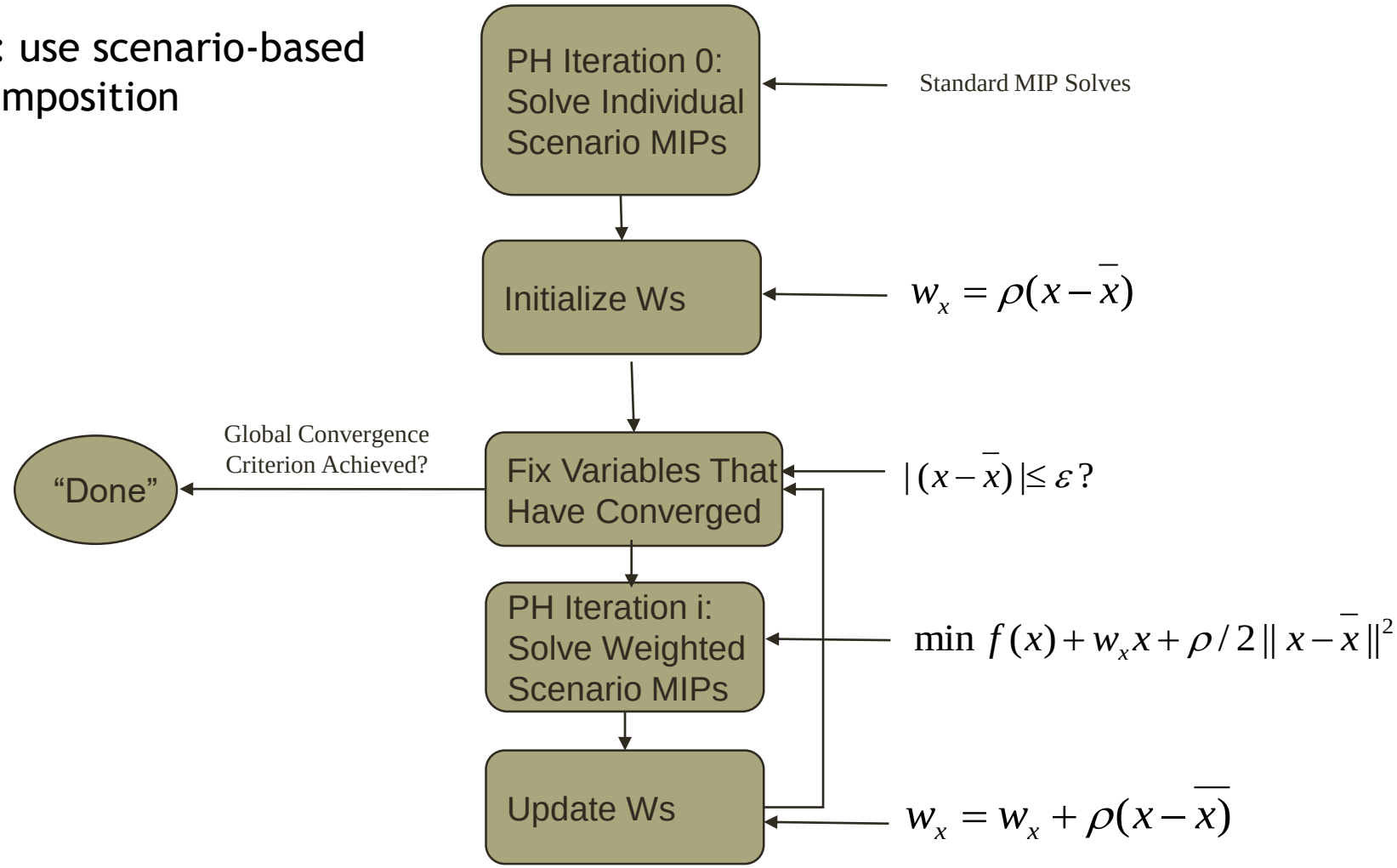
U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Progressive Hedging Algorithm

Idea: use scenario-based decomposition



Using PySP

1. Formulate the deterministic model
2. Specify the deterministic model data
3. Specify the scenario tree
4. Specify the scenario instance data

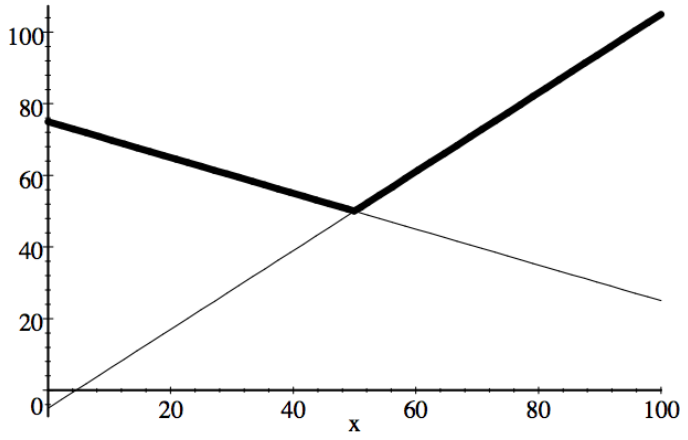
Example: Production Planning

Example from Alexander Shapiro and Andy Philpott, 2007

A company has decided to order a quantity *x* of a product to satisfy demand *d*. The per-unit cost of ordering is *c*, and if demand *d* is greater than *x*, then the back-order penalty is *b* per unit.

The objective is to minimize the total cost: max{ (c-b)x+bd, (c+h)x-hd}

For example: c=1, b=1.5, h=0.1, d=50



Deterministic Formulation

In general, the ordering decision is made before a realization of the demand is known.

The deterministic formulation corresponds to a single scenario taken with probability one.

$$\begin{aligned} \min_{x,t} \quad & t \\ \text{s.t.} \quad & t \geq (c-b)x + bd, \\ & t \geq (c+h)x - hd, \\ & x \geq 0. \end{aligned}$$

We can formulate a two-stage stochastic program where the first stage has zero cost and the second stage has cost t.

Pyomo Deterministic Formulation (1)

```
from pyomo.environ import *

c, b, h = 1.0, 1.5, 0.1

model = AbstractModel()
model.d = Param()
model.t = Var()
model.x = Var(within=NonNegativeReals, bounds=(0,100))

def c1_rule(model):
    return model.t >= (c-b)*model.x + b*model.d
model.c1 = Constraint(rule=c1_rule)

def c2_rule(model):
    return model.t >= (c+h)*model.x - h*model.d
model.c2 = Constraint(rule=c2_rule)
```

Pyomo Deterministic Formulation (2)

```
model.FirstStageCost = Var()
model.SecondStageCost = Var()
model.o = Objective(expr=model.FirstStageCost +
        model.SecondStageCost)

# 0*model.x
model.stage1 = Constraint(expr=model.FirstStageCost==0)
model.stage2 = Constraint(expr=model.SecondStageCost==model.t)
```

Deterministic Model Data

There is no deterministic model data for this example!

We used native Python data for the values of *c*, *b* and *h*.

Scenario Specification

The scenario specification is a Pyomo data file that provides meta-data about

1. The deterministic model
2. The scenario tree
3. The scenario data files

```
set Stages := FirstStage SecondStage ;

set Nodes := RootNode
             d1 d2 d3 d4 d5 ;

param NodeStage := RootNode FirstStage
                   d1       SecondStage
                   d2       SecondStage
                   d3       SecondStage
                   d4       SecondStage
                   d5       SecondStage
                   ;

set Children[RootNode] := d1 d2 d3 d4 d5 ;

param ConditionalProbability := RootNode          1.0
                                d1                0.2
                                d2                0.2
                                d3                0.2
                                d4                0.2
                                d5                0.2
                                ;

set Scenarios := s1 s2 s3 s4 s5 ;

param ScenarioLeafNode :=
                    s1 d1
                    s2 d2
                    s3 d3
                    s4 d4
                    s5 d5
                    ;

set StageVariables[FirstStage] :=  x;

set StageVariables[SecondStage] := t;

param StageCostVariable := FirstStage  FirstStageCost
                           SecondStage SecondStageCost ;
```

Scenario Data

Two methods are available to specify scenario-specific data

- Scenario-based
- Node-based

In the scenario-based approach, a single and complete .dat file is specified for each individual scenario

- Redundant, but straightforward if computer-generated

In the node-based approach, a single .dat file is specified for each node in the scenario tree

- Maximally compact, but requires some book-keeping

This example uses scenario-based data, with data files that simply define *d*.

Writing and Solving the Extensive Form (1)

In PySP, the runef script is provided to both write and solve the extensive form of a stochastic programming model

The basic command-line:

- `runef -m models -i scenarios --solve --solver=glpk`

NOTE: even commercial solvers often have difficulty solving EFs

Writing and Solving the Extensive Form (2)

After the solution, you get information about the tree

```
Tree Nodes:

        Name=RootNode
        Stage=FirstStage
        Parent=None
        Conditional probability=1.0000
        Children:
                d1
                d2
                d3
                d4
                d5
        Scenarios:
                s1
                s2
                s3
                s4
                s5
        Expected cost of (sub)tree rooted at node=   76.5000

        Name=d1
        Stage=SecondStage
        Parent=RootNode
        Conditional probability=0.2000
        Children:
                None
        Scenarios:
                s1
        Expected cost of (sub)tree rooted at node=   64.5000

        Name=d2
        Stage=SecondStage
        Parent=RootNode
        Conditional probability=0.2000
        Children:
                None
```

Progressive Hedging (1)

The execution of PH requires the specification of the penalty parameter (rho)

The global rho value can be easily specified:

- `runph -m models -i scenarios --default-rho=0.05`

The quadratic penalty term in PH is computationally problematic

- Quadratic MIP solvers can be 10x or slower than MIP solvers
- Open-source quadratic solvers are (almost) non-existent

PySP provides automatic, generic linearization mechanisms

- Requires specification of variable lower and upper bounds
- Specify number of breakpoints, distribution strategy
- `runph -m models -i scenarios --solver=glpk --default-rho=0.05 --linearize-nonbinary-penalty-terms=100`

Progressive Hedging (2)

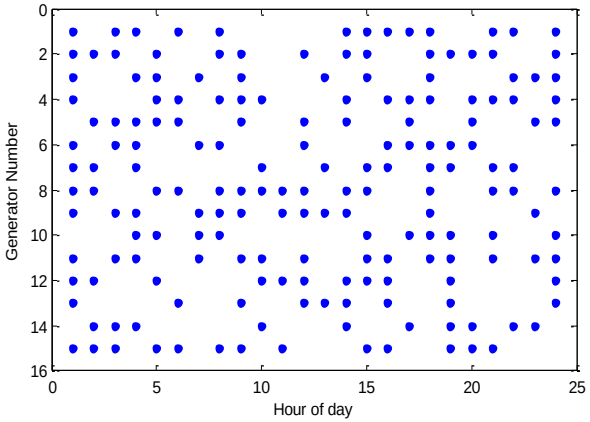
- In the presence of integers, PH is no longer guaranteed to converge
  - Cycling behavior
  - Stagnation behavior
- To facilitate PH convergence for mixed-integer stochastic programs, PySP provides various configurable mechanisms
  - "Watson-Woodruff" Extensions
    - Computational Management Science (2009)
  - Implemented via a generic plug-in callback framework
- Capabilities include:
  - Variable fixing
  - Cycle detection
  - Cycle breaking
  - Slamming

The End

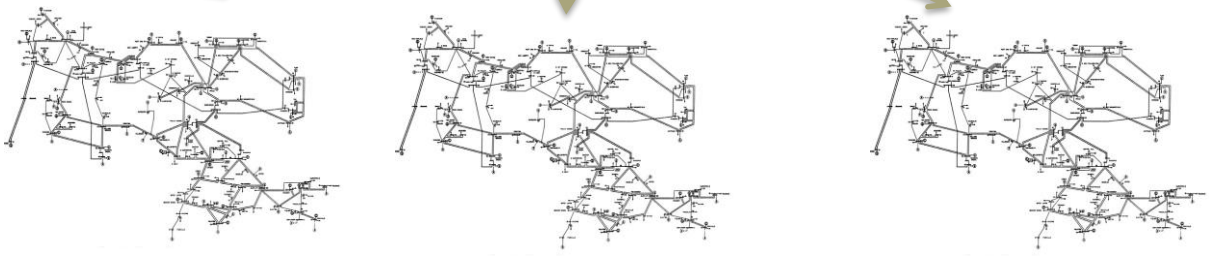
Questions?

Example: Stochastic Unit Commitment

Objective: Minimize expected cost



p_1 p_2 … p_N



Scenario 1 Scenario 2 … Scenario N

First stage variables:

- Unit On / Off

Nature resolves uncertainty

- Load
- Renewables output
- Forced outages

Second stage variables (*per time period*):

- Generation levels
- Power flows
- Voltage angles
- …

Solving with the Stochastic Extensive Form

- Reliability Unit Commitment (RUC) Test Instance: WECC-240++
- J.E. Price, Reduced Network Modeling of WECC as a Market Design Prototype, 2011 IEEE PES General Meeting
- Changes necessary to create viable RUC test case
  - Addition of realistic ramping rates and min up/down time constraints
- Results

Table 3 Solution quality statistics for the extensive form of the *WECC-240-r1* instance, given 4 hours of run time.

# Scenarios	Objective Value	MIP Lower Bound	Gap %	Run Time (s)
3	64278.20	63797.72	0.75	14491
5	62740.67	62180.86	0.89	14723
10	61563.10	60835.45	1.18	14630
25	61455.55	59963.78	2.36	14960
50	61911.74	59540.87	3.83	15480
100	62388.85	59548.23	4.51	16562

Parallelization and Bundling

- Progressive Hedging is, at least conceptually, easily parallelized
  - Scenario sub-problem solves are clearly independent
  - Advantage over Benders, in that “bloat” is distributed
    - Critical in low-memory-per-node cluster environments
  - Parallel efficiency drops rapidly as the number of processors increases
    - But: Relaxing barrier synchronization does not impact PH convergence
  - Bundling scenarios might help with parallel scaling
    - May increase number of iterations required

- PH can provide bounds!
  - Now comes with (rather tight) lower bounds
  - See “Obtaining Lower Bounds from the Progressive Hedging Algorithm for Stochastic Mixed-Integer Programs” (Under review)

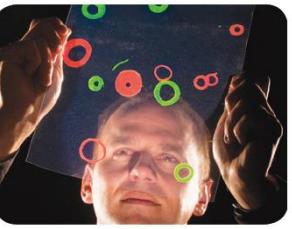
PH Results: Workstation and RedSky (HPC)

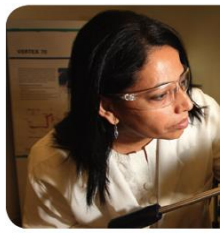
Table 4 Solve time (in seconds) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 1.0$, $\mu = 3$, and $\gamma = 0.03$

# Scenarios	Convergence Metric	Obj. Value	PH L.B.	# Vars Fx.	Time
		64-Core Workstation Results			
3	0.0 (in 23 iters)	64727.714	63188.709	4080	155
5	0.0 (in 26 iters)	62911.104	61609.576	4080	163
10	0.0 (in 26 iters)	61493.375	60347.220	4080	227
25	0.0 (in 27 iters)	60990.111	59875.661	4080	364
50	0.0 (in 17 iters)	60721.319	59527.252	4076	584
100	0.0 (in 23 iters)	61156.832	59880.559	4080	1218
		Red Sky Results			
50	0.0 (in 29 iters)	60676.383	59670.142	4062	514
100	0.0 (in 33 iters)	61122.781	60148.285	4073	672

Acknowledgements

- Jean-Paul Watson
- Roger Wets
- David Woodruff





9. Solving Bilevel Problems

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Overview of Bilevel Programming

A *bilevel program* is a mathematical program in which a subset of decision variables is constrained to take values associated with an optimal solution of a distinct, "lower" level mathematical program.

General formulation:

$$\begin{array}{ll} \min_{x \in X} & F(x, y) \\ \text{s.t.} & G(x, y) \leq 0 \\ & y \in P(x) \end{array}$$

Upper-level problem

where

$$P(x) = \begin{array}{ll} \operatorname{argmin}_{y \in Y} & f(x, y) \\ \text{s.t.} & g(x, y) \leq 0 \end{array}$$

Lower-level problem

Example: Modeling Security Problems

- Opponents must anticipate each other's moves
- Strategy should account for how opponent (best) responds

Extremely complex

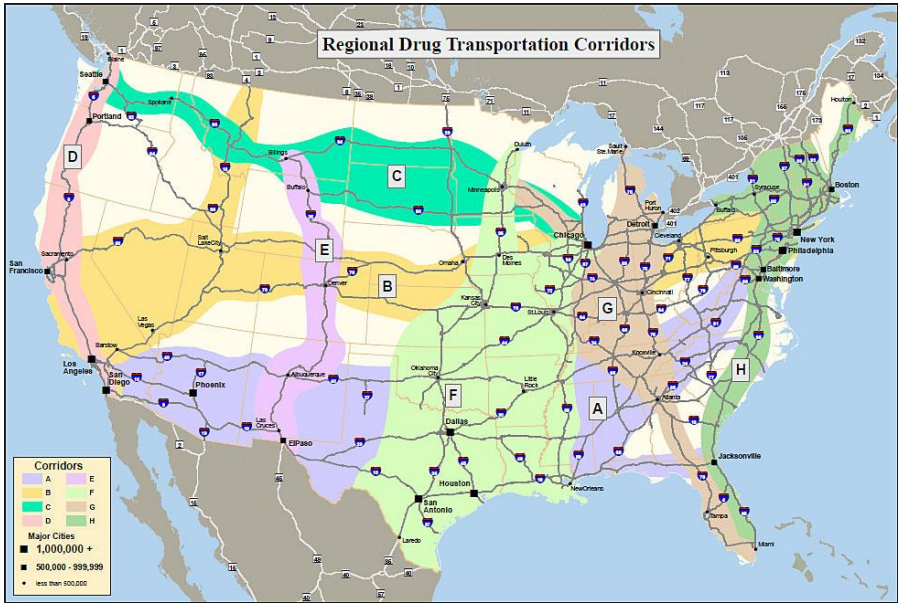
- Impossible to enumerate the set of all states in the game

Stackelberg games – bilevel programs

Example: Smuggling Interdiction

Interdictor minimizes the potential for a smuggler to evade detection

- Interdictor installs defenses (x) to minimize smuggler's evasion probability
- Smuggler traverses path (y) that maximizes the probability of evasion



Origin-destination nodes maybe unknown

- *F(x,y)* – evasion probability
- *X* – interdictor's constraints (e.g. resource, budget)
- *P(x)* – feasible paths given *x*

Modeling Bilevel Programs

No algebraic modeling language currently provides an *intuitive* syntax for expressing the structure of bilevel programs!

MPEC formulations are supported in several AMLs

- AMPL, AIMMS, GAMS, …
- MacMPEC includes bilevel programs that are reformulated as single-level programs using optimality conditions

Bilevel problems can be expressed in several modeling languages:

- GAMS, YALMIP
- Explicitly pass variables and constraints to a bilevel solver

A Simple Bilevel Example

Practical Bilevel Optimization: Algorithms and Applications

Jonathan Bard

Example 5.1.1

$$
\begin{array}{lll}
\min_{x \geq 0} & x - 4y & \\
\text{s.t.} & \min_{y \geq 0} & y \\
 & \text{s.t.} & -x - y \leq -3 \\
 & & -2x + y \leq 0 \\
 & & 2x + y \leq 12 \\
 & & -3x + 2y \leq -4
\end{array}
$$

Modeling Example 5.1.1 with YALMIP

```
sdpvar x, y;

OO = x - 4y;
CO = [x] >= 0;

OI = y;
CI = [[y] >= 0,
       -x -  y <= -3,
      -2x +  y <=  0,
       2x +  y <= 12,
      -3x + 2y <= -4];

solvebilevel(CO,OO,CI,OI,[y])
```

- Solve a bilevel problem using a simple branching strategy.
- Upper level problem defined by `CO` and `OO`
- Lower level problem defined by `CI` and `OI`, with decision variable `y`.

Note: This example adapted from the YALMIP bilevel documentation.

Modeling Example 5.1.1 with GAMS

```
positive variables x,y; variables objout,objin;
equations defout,defin,e1,e2,e3,e4;

defout.. objout =e= x - 4*y;
defin..  objin  =e= y;

e1..     - x -   y =l= -3;
e2..   -2*x +   y =l=  0;
e3..    2*x +   y =l= 12;
e4..    3*x - 2*y =l=  4;

model bard / all /;

$echo bilevel x min objin * defin e1 e2 e3 e4 > "%emp.info%"

solve bard us emp min objout;
```

Writes an "empinfo" file that tells the solver that this is a bilevel problem with a lower level problem that minimizes objective *objin* with variables *y* subject to the constraints (*defin*), *e1*, *e2*, *e3* and *e4*.

Modeling Example 5.1.1 with Pyomo

```
from pyomo.environ import *
from pyomo.bilevel import *

M = ConcreteModel()
M.x = Var(bounds=(0,None))
M.y = Var(bounds=(0,None))
M.o = Objective(expr=M.x - 4*M.y)

M.sub = SubModel(fixed=M.x)
M.sub.o = Objective(expr=M.y)
M.sub.c1 = Constraint(expr=-  M.x -   M.y <= -3)
M.sub.c2 = Constraint(expr=-2*M.x +   M.y <=  0)
M.sub.c3 = Constraint(expr= 2*M.x +   M.y <= 12)
M.sub.c4 = Constraint(expr=-3*M.x + 2*M.y <= -4)
```

- Lower level problem is declared with a *SubModel* component.
- The *var* argument indicates the lower level variables.
- Objectives, variables and constraints for the lower level problem are declared within this component.

Pyomo Extensions for Bilevel Programs

Modeling extensions

- Modeling components (pyomo.bilevel)

Model transformations

- Can be applied automatically

Custom solvers

- *Solvers tailored for specific classes of bilevel problems*

Solving Bilevel Problems

Goal: Enable solution of bilevel problems with standard solvers

Process:

- Model problem with SubModel components
- Transform the problem to a standard form
  - LP, MIP, etc
- Apply a suitable solver

Reformulations for linear bilevel programming (BLP)

A. BLP with continuous variables
B. Quadratic minimax with continuous lower-level variables

(A) BLP with Continuous Variables

Problem:

$$
\begin{aligned}
&\min_{x \geq 0} && c_1^T x + d_1^T y \\
&\text{s.t.} && A_1 x + B_1 y \leq b_1 \\
& && \min_{y \geq 0} && c_2^T x + d_2^T y \\
& && && A_2 x + B_2 y \leq b_2
\end{aligned}
$$

Reformulation: Replace lower-level problem with corresponding optimality conditions

$$
\begin{aligned}
&\min && c_1^T x + d_1^T y \\
&\text{s.t.} && A_1 x + B_1 y \leq b_1 \\
& && d_2 + B_2^T u - v = 0 \\
& && b_2 - A_2 x - B_2 y \geq 0 \perp u \geq 0 \\
& && y \geq 0 \perp v \geq 0 \\
& && x \geq 0, y \geq 0
\end{aligned}
$$

(A) BLP with Continuous Variables (cont'd)

Idea: Analyze the MPEC reformulation

Example:

- Use a custom solver that considers complementarity conditions (Bard, 1998)

Example:

- Chain reformulations: BLP -> MPEC -> GDP -> MIP
- Provide "BigM" values for unbounded variables
- Apply standard MIP solver (Fortuny-Amat and McCarl, 1981)

Example:

- Reformulate the complementarity conditions with nonlinear constraints (Ferris and Dirkse, 2005)

(B) Quadratic Min/Max

Problem:

- Upper level constraints do not constrain *y*

$$X = \{x | A_1 x \leq b_1, x \geq 0\}$$

- The upper decision variables may binary

$$\begin{aligned} \min_{x \in X} \quad \max_{y \geq 0} \quad & c_1^T x + d_1^T y + x^T Q y \\ \text{s.t.} \quad & A_2 x + B_2 y \leq b_2 \end{aligned}$$

Reformulation: Replace lower-level problem with the linear dual

$$\begin{aligned} \min \quad & c_1^T x + (b_2 - A_2 x)^T v \\ \text{s.t.} \quad & B_2^T v \geq d_1 + Q^T x \\ & A_1 x \leq b_1 \\ & x \geq 0, v \geq 0 \end{aligned}$$

(B) Quadratic Min/Max (cont'd)

Case 1:

- $A_2 \equiv 0$
- The reformulation is a simple LP (or a MIP if x are binary)

Case 2:

- The upper-level decision variables *x* are binary
- Reformulate the bilinear objective terms as disjunctions:

$$\begin{aligned} \min \quad & c_1^T x + b_2^T v - 1^T z \\ \text{s.t.} \quad & B_2^T v \geq d_1 + Q^T x \\ & \begin{bmatrix} x_i = 0 \\ z_i = 0 \end{bmatrix} \veebar \begin{bmatrix} x_i = 1 \\ z_i = A_2^T(i,*)v \end{bmatrix} \\ & x \in X, v \geq 0 \end{aligned}$$

- Reformulate this GDP -> MIP using "BigM" values

Solving Bilevel Programs in Pyomo

Python Script:

- Formulate the model
- Apply desired model reformulations
- Apply a suitable optimizer

  OR
- Directly analyze the model within Python
  - (e.g. using Pyomo's algebraic structure)

Pyomo Command:

- Execute a command that executes a Pyomo meta-solvers
  - Performs suitable reformulations
  - Applies a suitable optimizer
  - Maps the solution to the original problem

```
pyomo solve --solver=bilevel_ld model.py
```

Pyomo Capabilities

Relevant Pyomo Transformations

- core.linear_dual
- bilevel.linear_dual
- bilevel.linear_mpec
- gdp.bigm
- gdp.bilinear
- gdp.chull
- mpec.simple_disjunction
- mpec.simple_nonlinear

Relevant Pyomo Meta-Solvers

- bilevel_ld

Ongoing Work …

- Generalization and maturation of transformations
  - E.g. Working with general BLP models
- Automatic recognition of bilevel structure
  - Can we automate the application of reformulations?
- Parameterizing transformations
  - How can we flexibly specify transformation options?
  - E.g. Specifying big-M values for specific complementarity conditions
- Additional meta-solvers
  - E.g. bilevel_blp

Acknowledgements

- Richard L. Chen
- William E. Hart
- John D. Siirola
- Jean-Paul Watson

(A) BLP with Continuous Variables

Problem:

$$\begin{aligned} \min_{x \geq 0} \quad & c_1^T x + d_1^T y \\ \text{s.t.} \quad & A_1 x + B_1 y \leq b_1 \\ & \min_{y \geq 0} \quad c_2^T x + d_2^T y \\ & \qquad\qquad A_2 x + B_2 y \leq b_2 \end{aligned}$$

Reformulation: This is the MPEC model that eliminates the v variable in the reformulation on the earlier slide.

$$\begin{aligned} \min \quad & c_1^T x + d_1^T y \\ \text{s.t.} \quad & A_1 x + B_1 y \leq b_1 \\ & b_2 - A_2 x - B_2 y \geq 0 \perp u \geq 0 \\ & y \geq 0 \perp d_2 + B_2^T u \geq 0 \\ & x \geq 0, y \geq 0 \end{aligned}$$