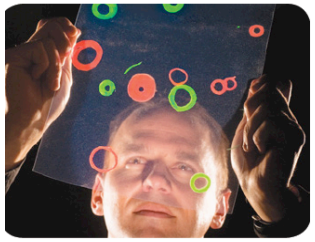
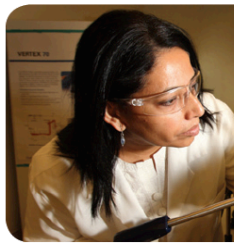




Sandia National Laboratories

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 multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Nonlinear problems are easy…

Nonlinear problems are easy…

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

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 * math.pi * m.r * (m.r + m.h)
model.surf_area_obj = Objective(rule=surf_area_obj_rule)

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

Nonlinear: Supported expressions

Sandia National Laboratories

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
```

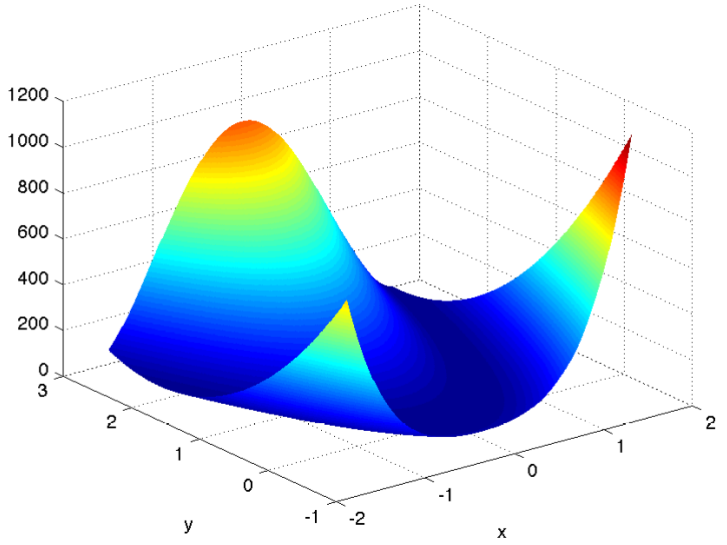
CCR Center for Computing Research

PYOMO COIN-OR

Example: Rosenbrock function

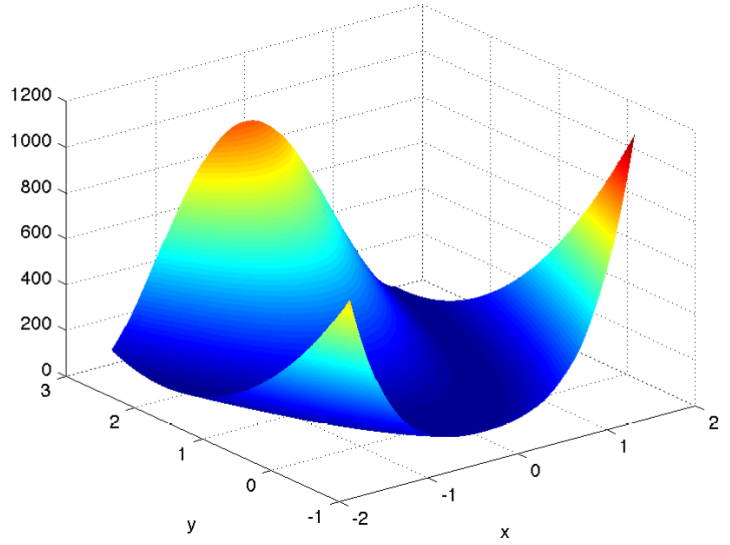
$$\min_{x,y} \; f(x,y) = (1-x)^2 + 100\left(y-x^2\right)^2$$

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



Example: Rosenbrock function

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



- 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(initialize=1.5)
model.y = Var(initialize=1.5)

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

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

Nonlinear: Exercises

1.1 Alternative Initialization: Effective initialization can be critical for solving nonlinear problems, since they can have several local solutions and numerical difficulties. Solve the Rosenbrock example using different initial values for the x variables. Write a loop that varies the initial value from 2.0 to 6.0, solves the problem, and prints the solution for each iteration of the loop. (A solution for this problem can be found in `rosenbrock_init_soln.py`.)

1.2 Evaluation errors: Consider the following problem with initial values x=5, y=5.

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad y = \sqrt{x - 1.0}$$

(a) Starting with `evaluation_error_incomplete.py`, formulate this Pyomo model and solve using IPOPT. You should get a list of errors from the solver. Add the IPOPT solver option `solver.options['halt_on_ampl_error']='yes'` to find the problem. (Hint: error output might be ordered strangely, look up in the console output.) What did you discover? How might you fix this? (A solution for this can be found in `evaluation_error_soln.py`.)

(b) Add bounds $x \geq 1$ to fix this problem. Resolve the problem. Comment on the number of iterations and the quality of solution. (Note: The problem still occurs because $x \geq 1$ is not enforced exactly, and small numerical values still cause the error.) (A solution for this can be found in `evaluation_error_bounds_soln.py`.)

(c) Think about other solutions for this problem. (e.g., $x \geq 1.001$). (A solution for this can be found in `evaluation_error_bounds2_soln.py`.)

Nonlinear: Exercises

1.3 Alternative Formulations: Consider the following problem with initial values x=5, y=5.

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x-1}{y} = 1$$

Note that the solution to this problem is x=1.005 and y=0.005. There are several ways that the problem above can be reformulated. Some examples are shown below. Which ones do you expect to be better? Why? Starting with `formulation_incomplete.py` finish the Pyomo model for each of the formulations and solve with IPOPT. Note the number of iterations and quality of solutions. What can you learn about problem formulation from these examples?

(a) (A solution can be found in `formulation_1_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x-1}{y} = 1$$

(b) (A solution for this can be found in `formulation_2_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad \frac{x}{y+1} = 1$$

(c) (A solution for this can be found in `formulation_3_soln.py`.)

$$\min_{x,y} \; f(x,y) = (x-1.01)^2 + y^2$$
$$\text{s.t.} \quad y = x - 1$$

(d) Bounds and initialization can be very helpful when solving nonlinear optimization problems. Starting with `formulation_incomplete.py` resolve the original problem, but add bounds, $y \geq 0$. Note the number of iterations and quality of solution, and compare with what you found in 1.2 (a). (A solution for this can be found in `formulation_4_soln.py`.)

Nonlinear: Exercises

1.4 Reactor design problem (Hart et al., 2017; Bequette, 2003): In this example, we will consider a chemical reactor designed to produce product B from reactant A using a reaction scheme known as the Van de Vusse reaction:

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C$$
$$2A \xrightarrow{k_3} D$$

Under appropriate assumptions, F is the volumetric flowrate through the tank. The concentation of component A in the feed is c_{Af}, and the concentrations in the reactor are equivalent to the concentrations of each component flowing out of the reactor, given by c_A, c_B, c_C, and c_D.

If the reactor is too small, we will not produce sufficient quantity of B, and if the reactor is too large, much of B will be further reacted to form the undesired product C. Therefore, our goal is to solve for the reactor volume that maximizes the outlet concentration for product B.

The steady-state mole balances for each of the four components are given by,

$$\begin{aligned}
0 &= \frac{F}{V}c_{Af} - \frac{F}{V}c_A - k_1 c_A - 2k_3 c_A^2 \\
0 &= -\frac{F}{V}c_B + k_1 c_A - k_2 c_B \\
0 &= -\frac{F}{V}c_C + k_2 c_B \\
0 &= -\frac{F}{V}c_D + k_3 c_A^2
\end{aligned}$$

The known parameters for the system are,

$$c_{Af} = 10\ \frac{\text{gmol}}{\text{m}^3} \quad k_1 = \frac{5}{6}\ \text{min}^{-1} \quad k_2 = \frac{5}{3}\ \text{min}^{-1} \quad k_3 = \frac{1}{6000}\ \frac{\text{m}^3}{\text{mol min}}.$$

Formulate and solve this optimization problem using Pyomo. Since the volumetric flowrate F always appears as the numerator over the reactor volume V, it is common to consider this ratio as a single variable, called the space-velocity SV. (A solution to this problem can be found in `reactor_design_soln.py`.)

OTHER MATERIAL

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

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

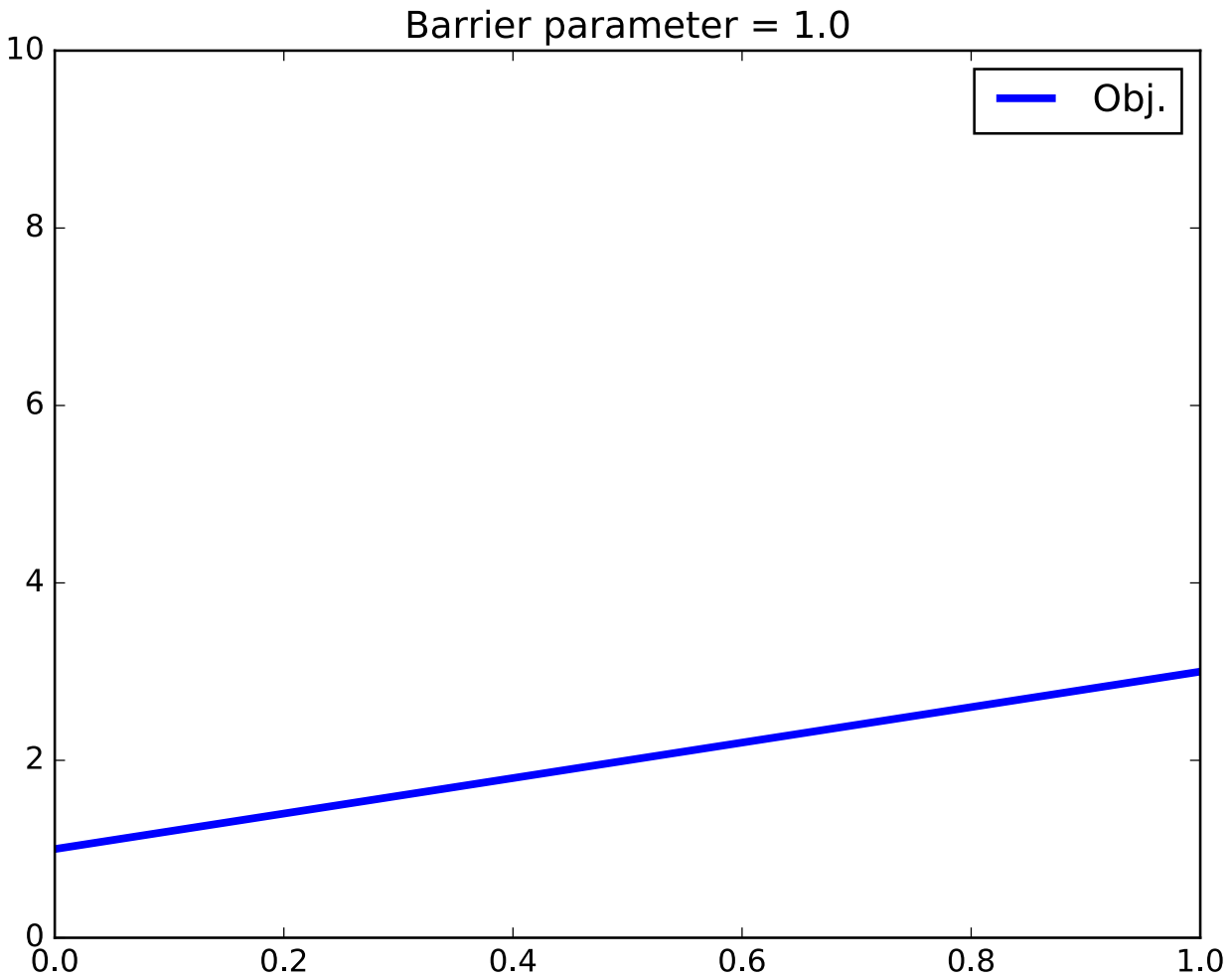
as $x \to 0, \quad ln(x) \to \infty$

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

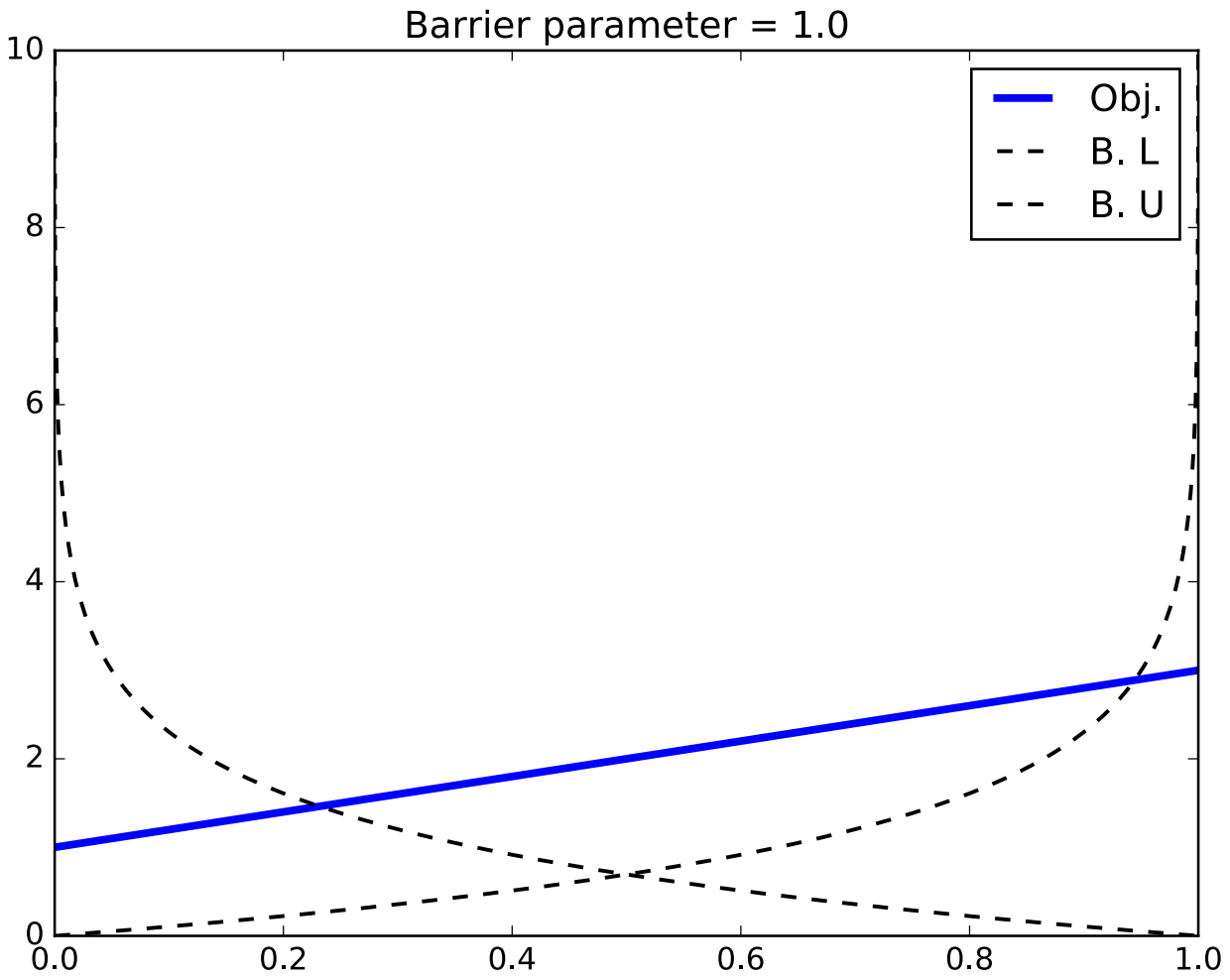
Fiacco & McCormick (1968)

- 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 $l \leftarrow l + 1$

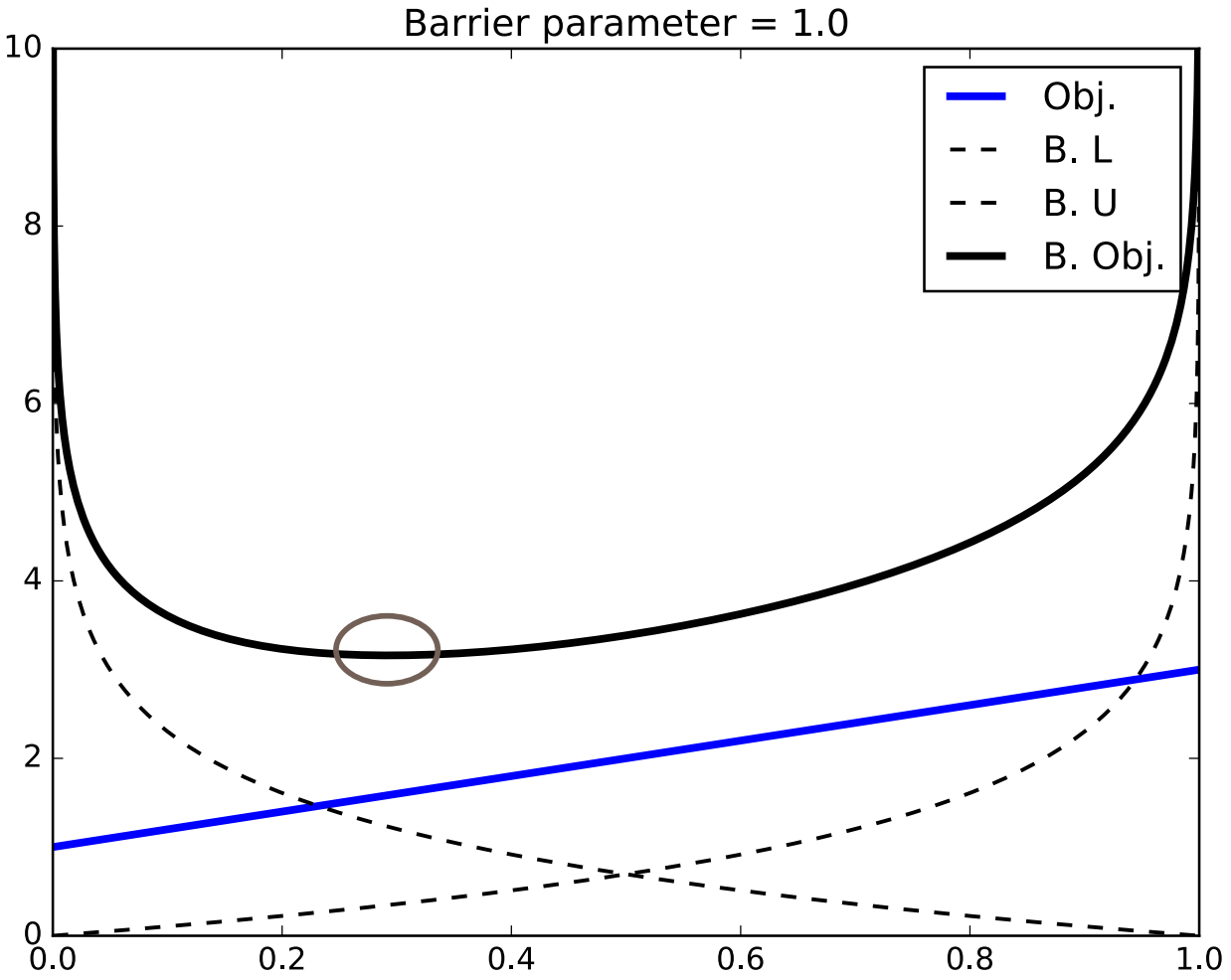
Effect of Barrier Term



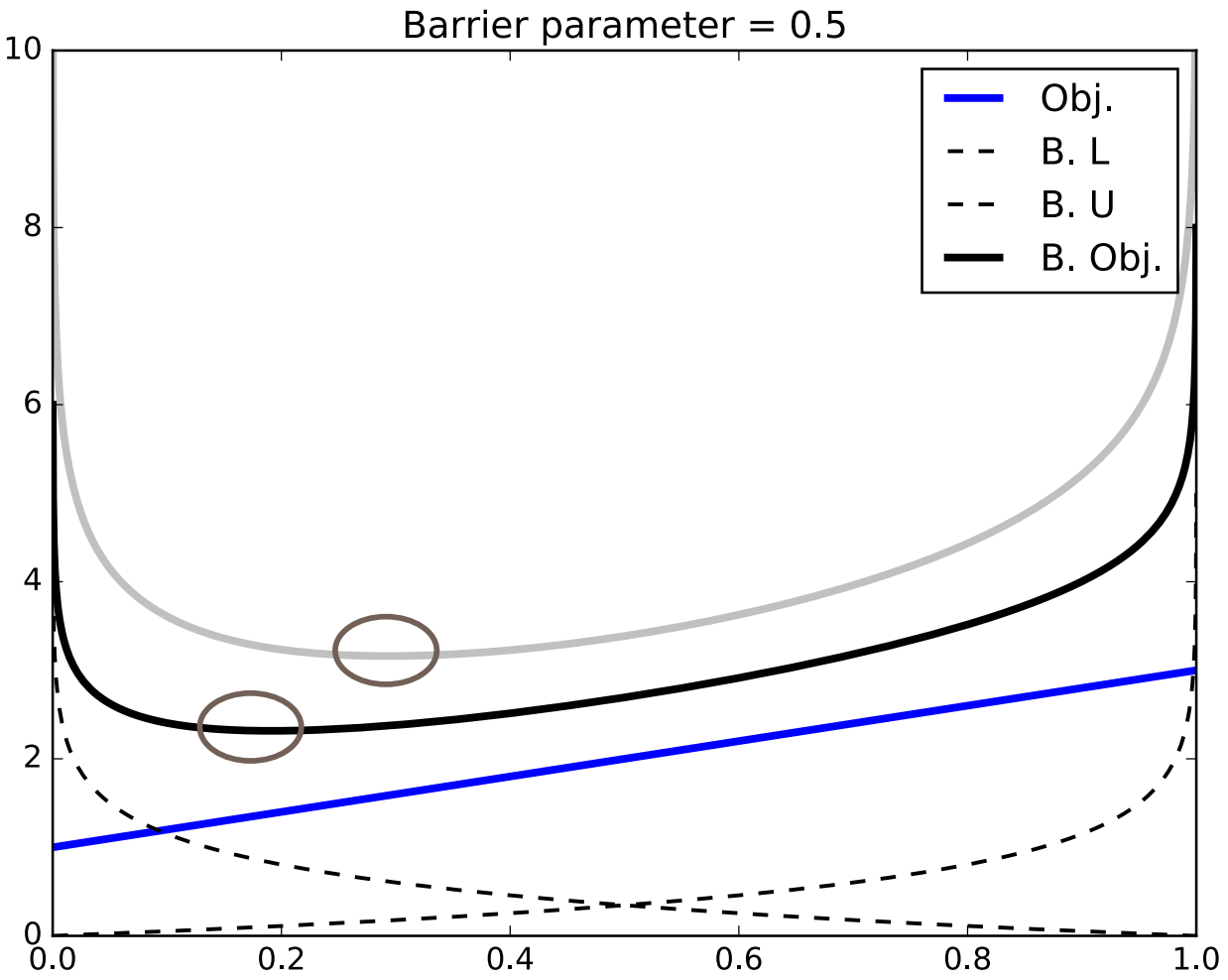
Effect of Barrier Term



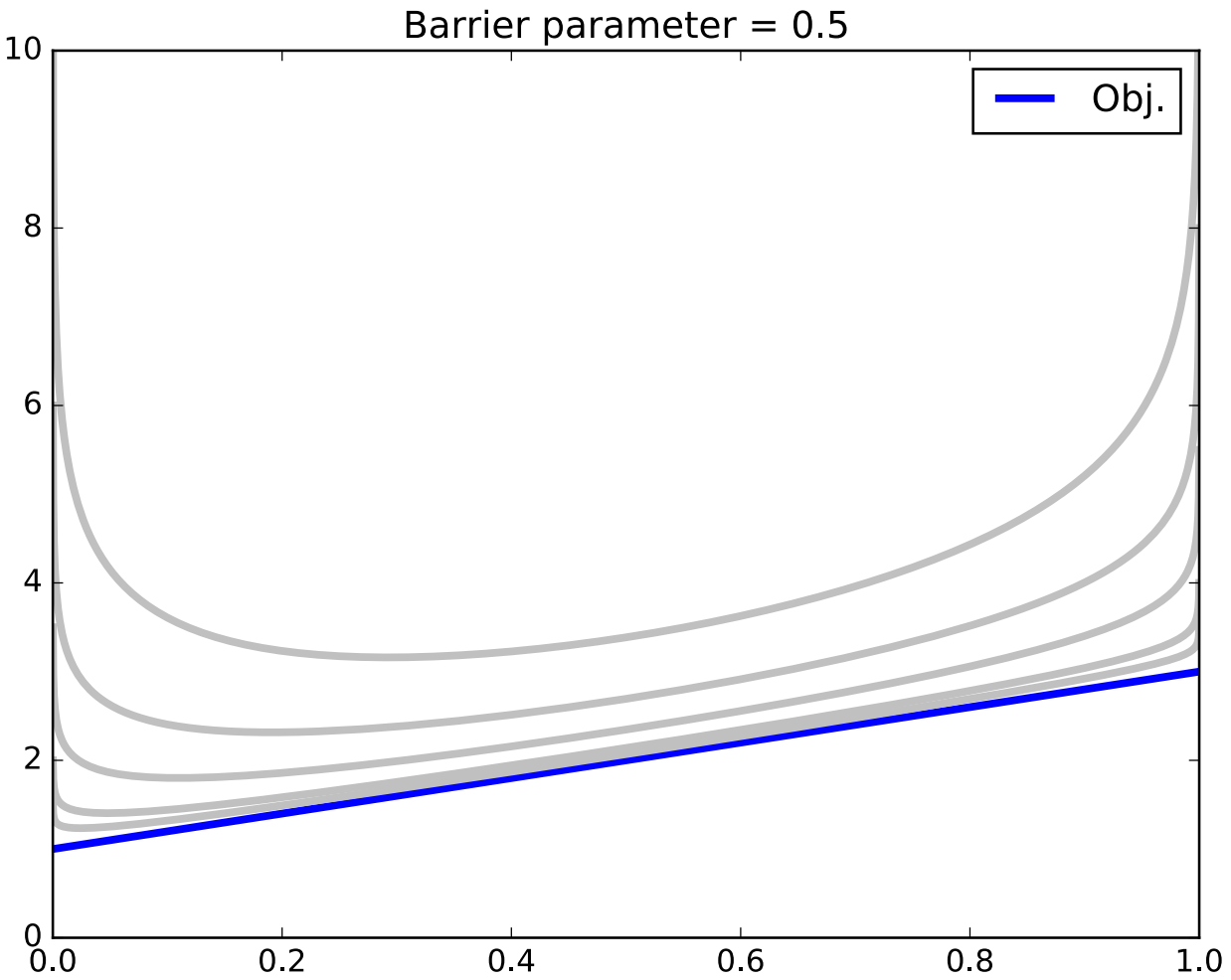
Effect of Barrier Term



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

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

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

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

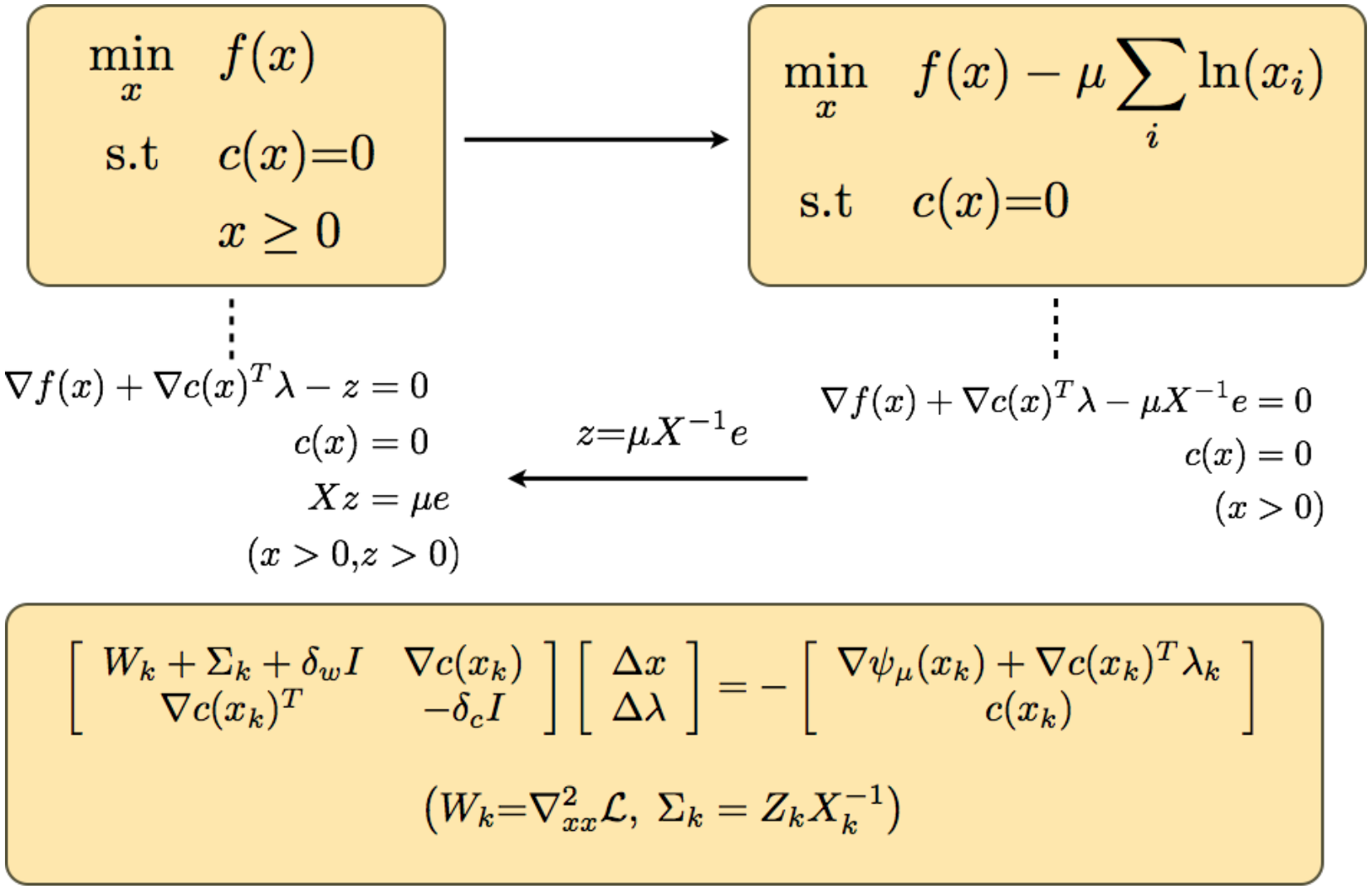
Fiacco & McCormick (1968)

- 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 $l \leftarrow l + 1$

Solve Barrier NLP?
Barrier parameter update?
Globalization?

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

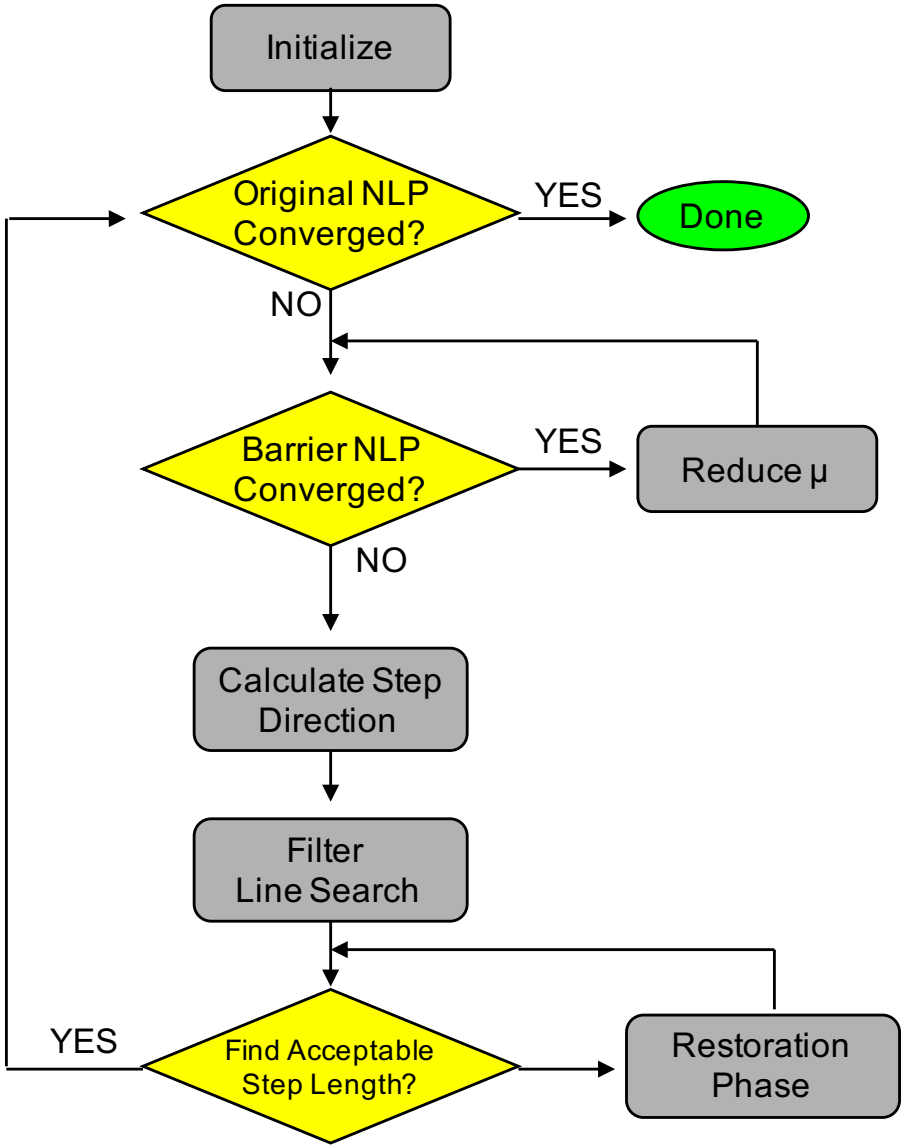
Interior Point Methods

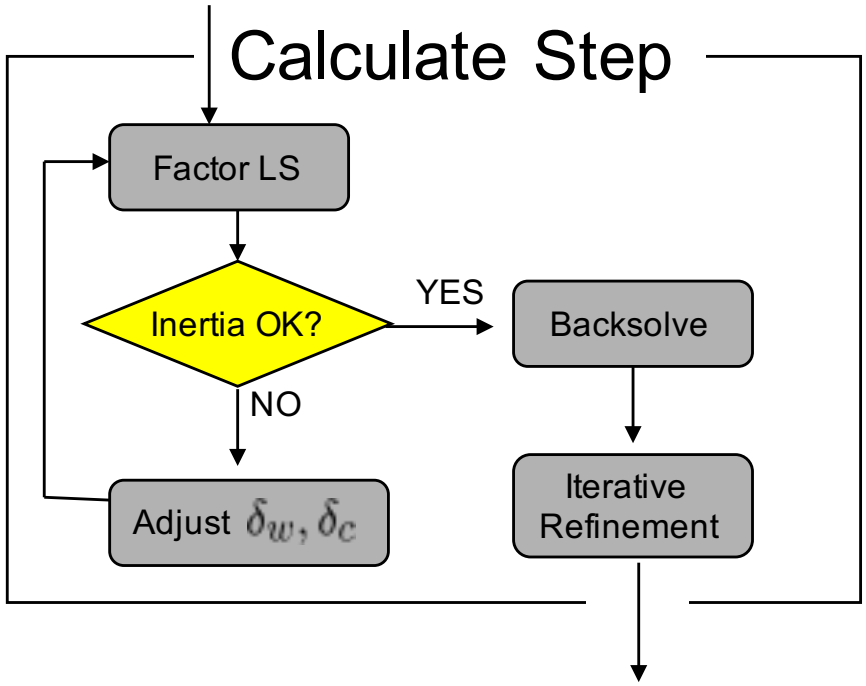


IPOPT: Other Considerations

- 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

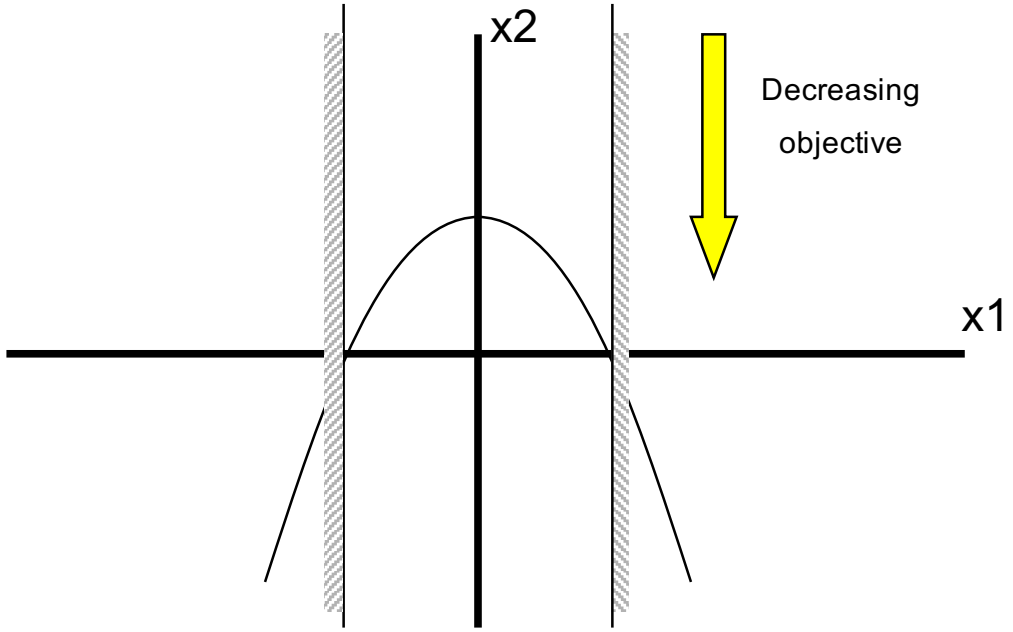
Exit Conditions: Successful

$$\min \quad x_2$$

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

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

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
```

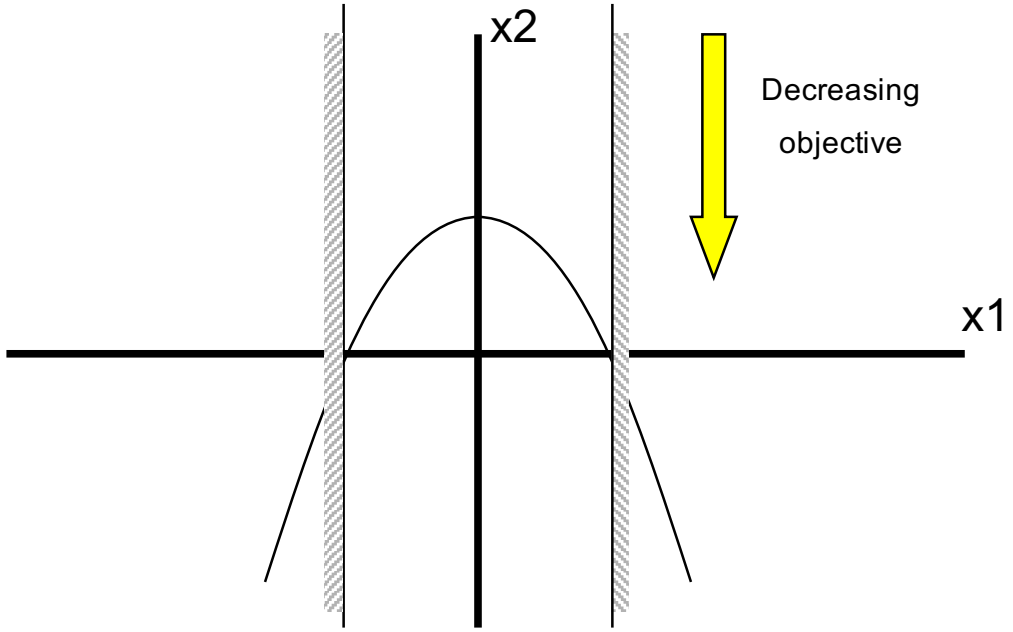
Exit Conditions: Successful w/ Regularization

$$\min \quad x_2$$

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

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

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

. . .

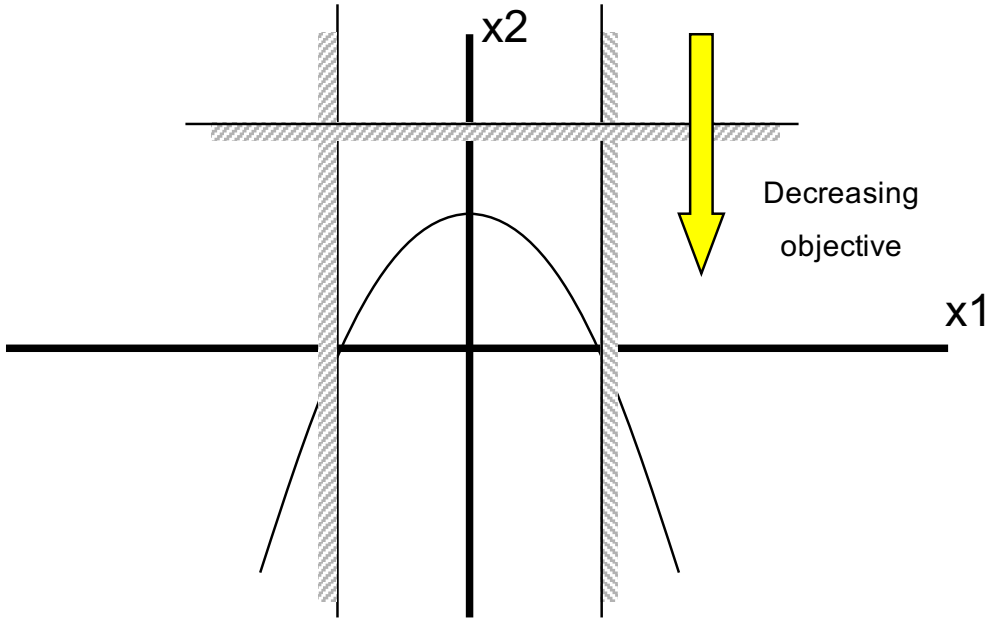
EXIT: Optimal Solution Found.

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.

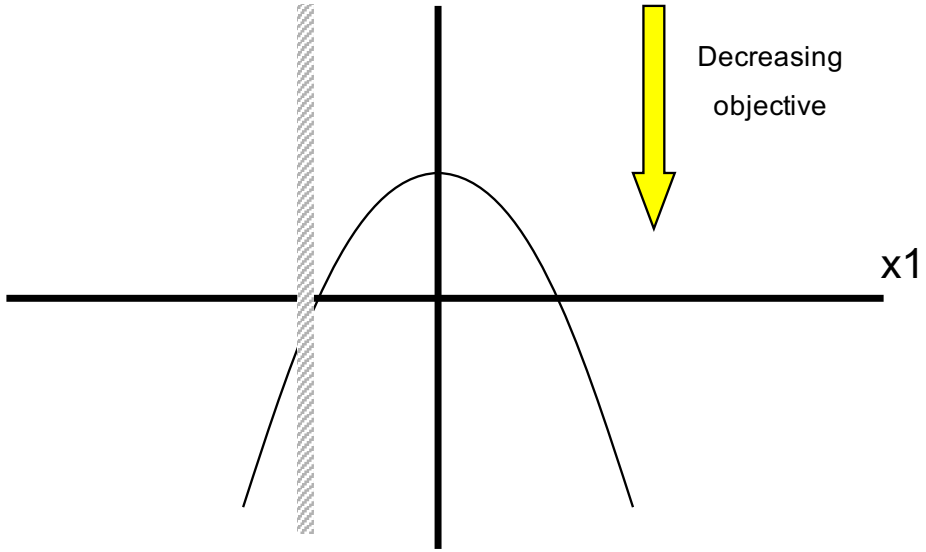
Ipopt 3.11.1: Converged to a locally infeasible point. Problem may be infeasible.
WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt
3.11.1\x3a Converged to a locally infeasible point. Problem may be infeasible.

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

Exit Conditions: Unbounded

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

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).

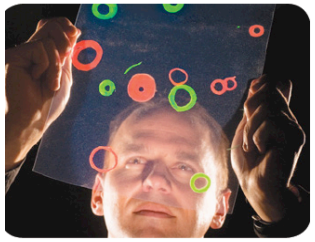

This is Ipopt version 3.11.1, running with linear solver mumps.
```

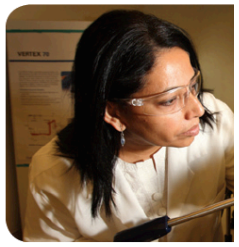
Other Considerations

- Linear Solvers
  - Performance of IPOPT is HIGHLY dependent on the linear solver selected
  - The version of IPOPT installed in this workshop uses a freely distributable linear solver, MUMPS.
  - Performance of IPOPT with the Harwell Subroutine Library suite can be significantly better
  - MA27 (HSL) is **free** for personal and commercial use, but not **distributable**
  - Recommend downloading and installing Coin-HSL Archive (MA27) from HSL
    - Web search: "HSL for IPOPT"
- 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

Other Considerations

- 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

Introduction to Blocks: Structured Modeling in Pyomo

PYOMO

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What's the problem with Math Programming?

Sandia National Laboratories

$$\text{Minimize}: \quad \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right) \tag{1}$$

$$\text{S.t.} \quad \theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t \tag{2}$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$

$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t \tag{3a}$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$

$$\forall\, n, \text{ generator contingency states } c, t \tag{3b}$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t \tag{4}$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t \tag{5a}$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t \tag{5b}$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t \tag{6}$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t \tag{7}$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\} \tag{8}$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\} \tag{9}$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t \tag{10}$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t \tag{11}$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t \tag{12}$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t \tag{13}$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t \tag{14}$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t \tag{15}$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t \tag{16}$$

CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

PYOMO COIN-OR

What's the problem with Math Programming?

Sandia National Laboratories

$$
\begin{aligned}
\text{Minimize}: \quad & \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt} \right) \\
\text{S.t.} \quad & \theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t \\
& \sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt}, \\
& \forall\, n, \quad c = 0, \text{ transmission contingency states } c, t \\
& \sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt}, \\
& \forall\, n, \text{ generator contingency states } c, t \\
& P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t \\
& B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t \\
& B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t \\
& P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t \\
& v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t \\
& \sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\} \\
& \sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\} \\
& P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t \\
& P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t \\
& P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t \\
& P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t \\
& 0 \le v_{g,t} \le 1, \quad \forall\, g, t \\
& 0 \le w_{g,t} \le 1, \quad \forall\, g, t \\
& u_{g,t} \in \{0, 1\}, \quad \forall\, g, t
\end{aligned}
$$

- The "solution" for Unit Commitment + Transmission Switching + N-1 reliability

CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

PYOMO COIN-OR

The challenge: MP is dense and subtle

Sandia National Laboratories

$$\text{Minimize}: \quad \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right)$$

S.t.

$$\theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$

$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$

$$\forall\, n, \text{ generator contingency states } c, t$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$$

To a first approximation:

- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

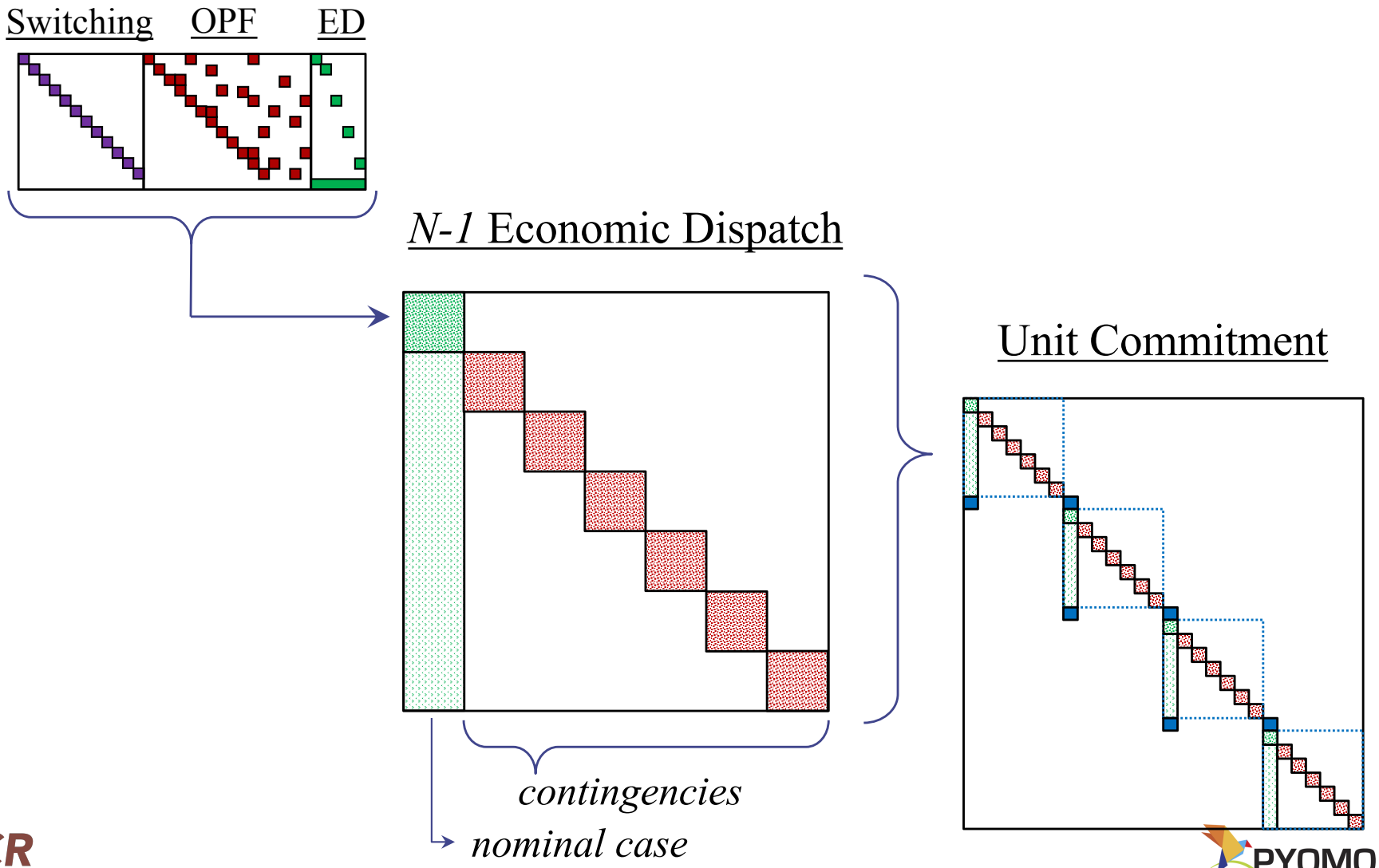
CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

PYOMO COIN-OR

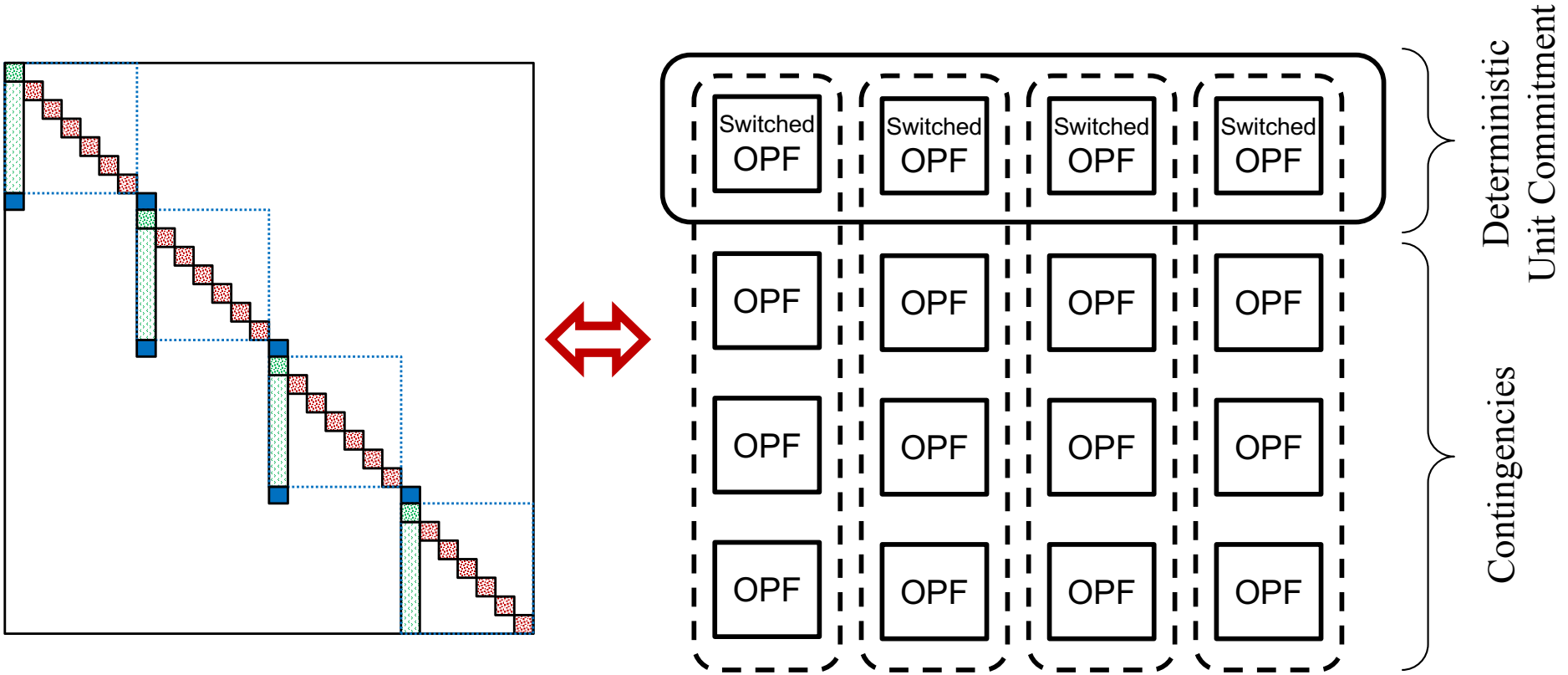
(Nonobvious) Inherent structure

(Sparsity pattern in model constraint matrix)



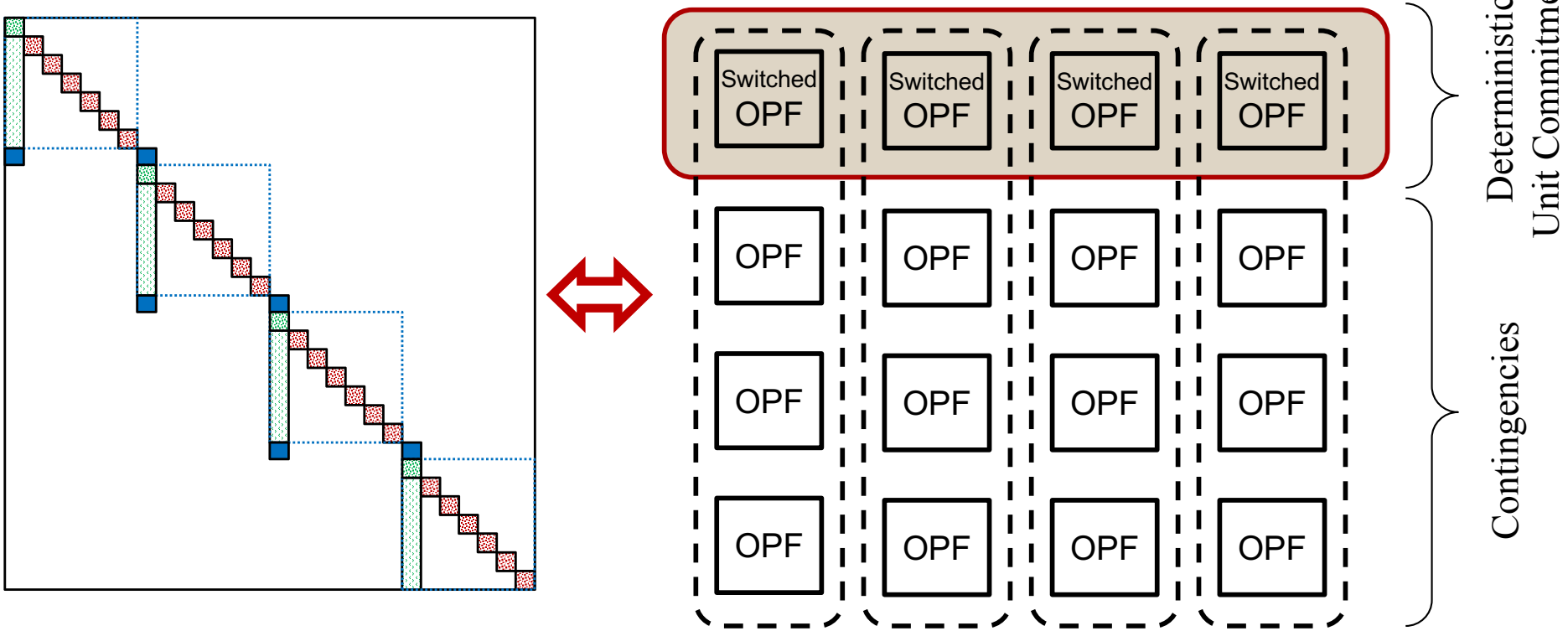
UC + N-1 + Switching block structure

- "2-D" grid of linked optimal power flow models

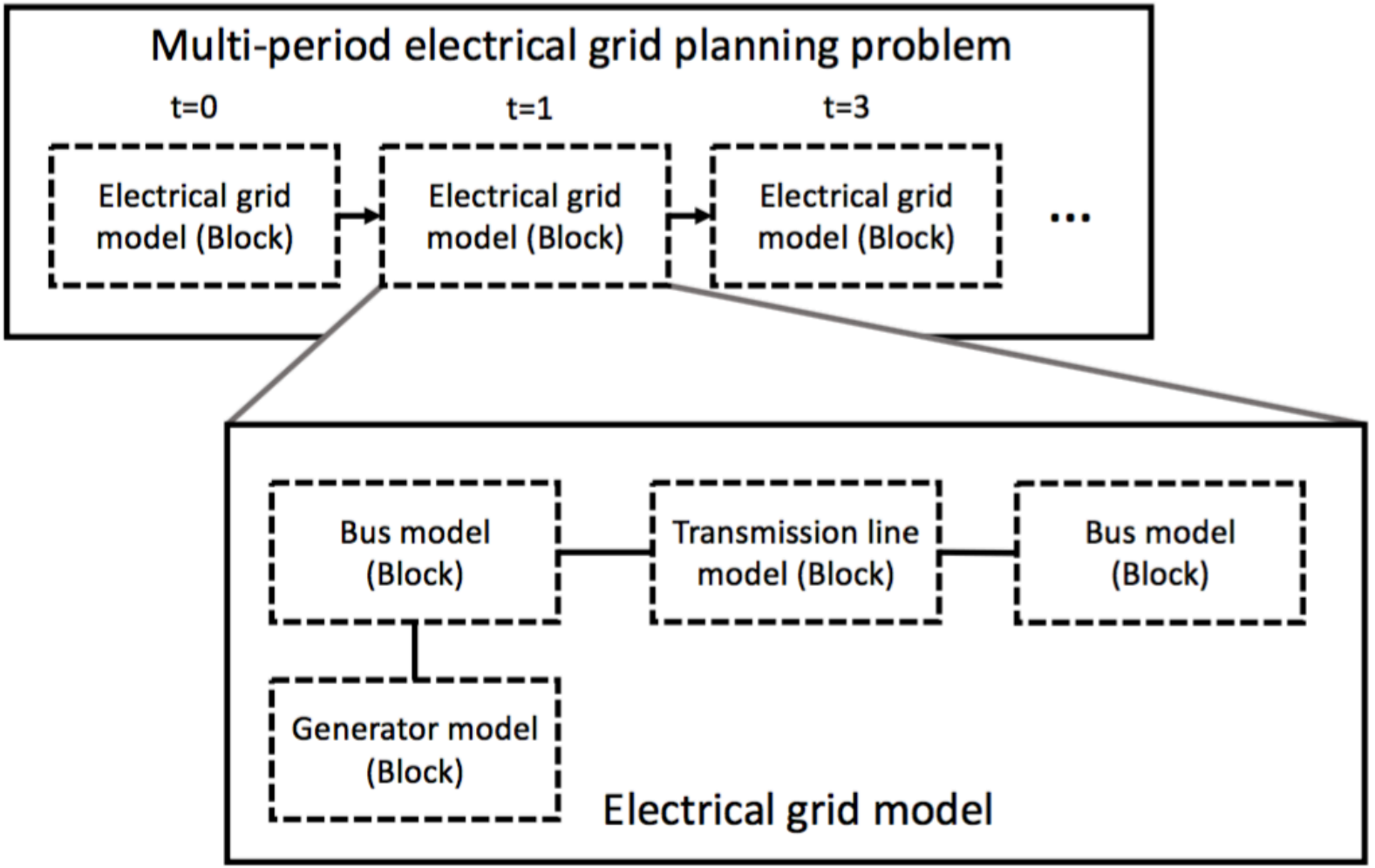


UC + N-1 + Switching block structure

- "2-D" grid of linked optimal power flow models



Object-oriented model construction



Pyomo Blocks

- The Pyomo ”Block” allows us to construct hierarchical models
- A Block can be treated in much the same way as a model
  - Modeling components can be added to blocks (e.g., Var, Constraint)
  - Blocks can be added to other Blocks

```
model = ConcreteModel()
model.x = Var()
model.P = Param(initialize=5)
model.S = RangeSet(model.P)
```

Blocks provide *namespacing*

- Each block defines its own namespace

```
print(model.x.local_name)       # x
print(model.x.name)             # x
print(model.b.x.local_name)     # x
print(model.b.x.name)           # b.x
print(model.b.b.x.local_name)   # x
print(model.b.b.x.name)         # b.b.x
```

- Blocks can be created (and "solved") and later added to a parent model

```
new_b = Block()
new_b.x = Var()
new_b.P = Param(initialize=5)
new_b.I = RangeSet(10)

model = ConcreteModel()
model.b = new_b
model.x = Var(model.b.I)
```

Indexed objects and blocks

- Blocks may be indexed (and defined by rules)

```
model = ConcreteModel()
model.P = Param(initialize=3)
model.T = RangeSet(model.P)

def xyb_rule(b, t):
    b.x = Var()
    b.I = RangeSet(t)
    b.y = Var(b.I)
    b.c = Constraint(expr = b.x == 1.0 - sum(b.y[i] for i \
        in b.I))
model.xyb = Block(model.T, rule=xyb_rule)
```

- Components within blocks may also be indexed

```
def xyb_rule(b, t):
    b.x = Var()
    b.I = RangeSet(t)
    b.y = Var(b.I, initialize=1.0)
    def _b_c_rule(_b):
        return _b.x == 1.0 - sum(_b.y[i] for i in _b.I)
    b.c = Constraint(rule=_b_c_rule)
model.xyb = Block(model.T, rule=xyb_rule)
```

Accessing objects within blocks

- Accessing components within Blocks: looping

```
for t in model.xyb:
    for i in model.xyb[t].y:
        print("%s %f" % (model.xyb[t].y[i], value(model.xyb[t].y[i])
```

- Slicing
  - Pyomo supports a special "slice-like" wildcard notation
    - ":" is a wildcard matching a single index
    - "..." is a wildcard that matches 0 or more indices

```
for v in model.xyb[:].y[:]:
    print("%s %f" % v, value(v))
```

Advanced slicing: a "weird" model

```
from pyomo.environ import *

m = ConcreteModel()
m.I = Set(initialize=[1,2,3])
@m.Block(m.I, m.I)
def block(b, i, j):
    b_id = 10*(3*i+j)

    @b.Set(dimen=None)
    def IDX(b):
        return [ tuple(
            x*10**y for y in range(j))
            for x in range(i) ]

    @b.Block(b.IDX)
    def subblock(b, *x):
        b.x = Var(initialize=b_id + sum(x))

for v in m.block[...].subblock[...].x:
    print("%-25s= %s" % (v, value(v)))
```

```
block[1,1].subblock[0].x = 40
block[1,2].subblock[0,0].x= 50
block[1,3].subblock[0,0,0].x= 60
block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201
block[3,1].subblock[2].x = 102
block[3,1].subblock[0].x = 100
block[3,1].subblock[1].x = 101
block[3,2].subblock[1,10].x= 121
block[3,2].subblock[0,0].x= 110
block[3,2].subblock[2,20].x= 132
block[3,3].subblock[2,20,200].x= 342
block[3,3].subblock[0,0,0].x= 120
block[3,3].subblock[1,10,100].x= 231
```

Advanced slicing: explicit indices

```
for v in m.block[2,...].subblock[...].x:
    print("%-25s= %s" % (v, value(v)))

block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201

for v in m.block[2,:].subblock[...].x:
    print("%-25s= %s" % (v, value(v)))

block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201
```

```
block[1,1].subblock[0].x = 40
block[1,2].subblock[0,0].x= 50
block[1,3].subblock[0,0,0].x= 60
block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201
block[3,1].subblock[2].x = 102
block[3,1].subblock[0].x = 100
block[3,1].subblock[1].x = 101
block[3,2].subblock[1,10].x= 121
block[3,2].subblock[0,0].x= 110
block[3,2].subblock[2,20].x= 132
block[3,3].subblock[2,20,200].x= 342
block[3,3].subblock[0,0,0].x= 120
block[3,3].subblock[1,10,100].x= 231
```

Advanced slicing: jagged sets

```
for v in m.block[...].subblock[1,...].x:
    print("%-25s= %s" % (v, value(v)))

block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,3].subblock[1,10,100].x= 201
block[3,1].subblock[1].x = 101
block[3,2].subblock[1,10].x= 121
block[3,3].subblock[1,10,100].x= 231

for v in m.block[...].subblock[1,:].x:
    print("%-25s= %s" % (v, value(v)))

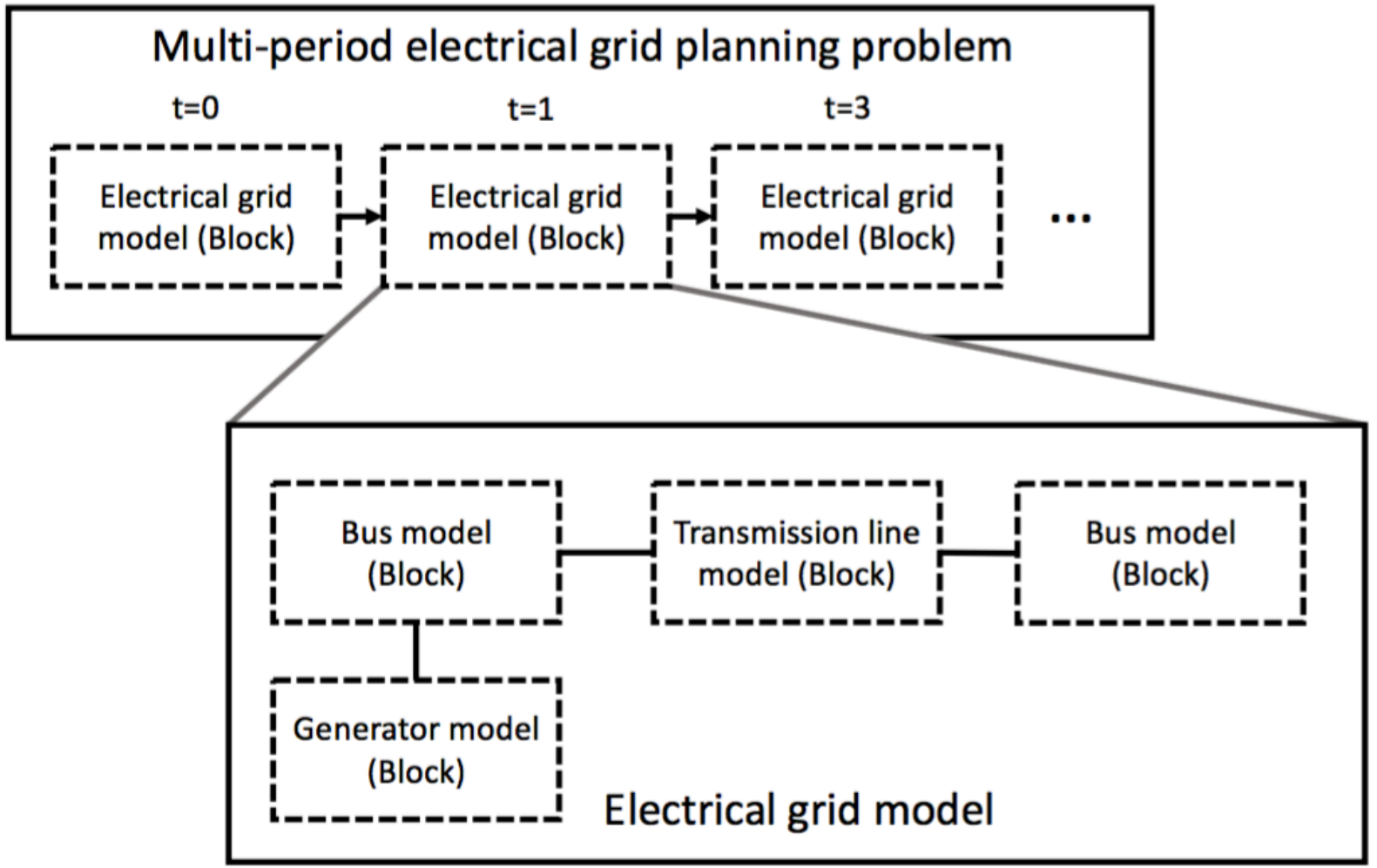
block[2,2].subblock[1,10].x= 91
block[3,2].subblock[1,10].x= 121

for v in m.block[...].subblock[...,1].x:
    print("%-25s= %s" % (v, value(v)))

block[2,2].subblock[1].x= 71
block[3,2].subblock[1].x= 101
```

```
block[1,1].subblock[0].x = 40
block[1,2].subblock[0,0].x= 50
block[1,3].subblock[0,0,0].x= 60
block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201
block[3,1].subblock[2].x = 102
block[3,1].subblock[0].x = 100
block[3,1].subblock[1].x = 101
block[3,2].subblock[1,10].x= 121
block[3,2].subblock[0,0].x= 110
block[3,2].subblock[2,20].x= 132
block[3,3].subblock[2,20,200].x= 342
block[3,3].subblock[0,0,0].x= 120
block[3,3].subblock[1,10,100].x= 231
```

Back to object-oriented modeling



(Complete!) Solution

```
import pyomo.environ as pe
import dcopf_decl as dcopf
from uc_example_data import data

model = pe.ConcreteModel()
model.T = pe.Set(initialize=range(2), ordered=True)
model.G = pe.Set(initialize=sorted(data['gens'].keys()), ordered=True)

# create blocks of the dcopf model for each time period
def period_rule(b, t):
    return dcopf.create_dcopf_model(data[t])
model.period = pe.Block(model.T, rule=period_rule)

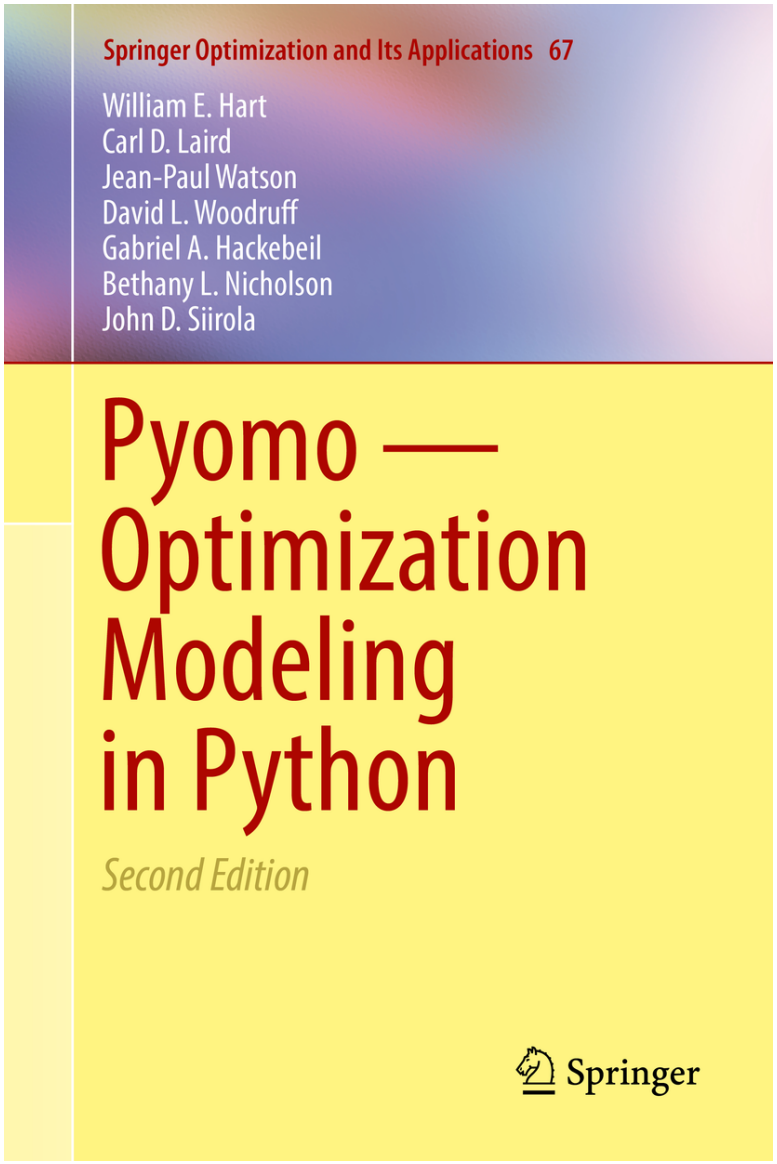
# create the ramping constraints between time periods
def ramp_con_rule(m, t, g):
    if t == m.T.first():
        return pe.Constraint.Skip
    return (-15.0, m.period[t-1].pg[g] - m.period[t].pg[g], 15.0)
model.ramp_con = pe.Constraint(model.T, model.G, rule=ramp_con_rule)

# create the new objective (sum over time)
model.period[:].objective.deactivate()
model.obj = pe.Objective(expr=sum(model.period[:].objective.expr))

# solve the new multi-period model
solver = pe.SolverFactory('ipopt')
solver.solve(model, tee=True)

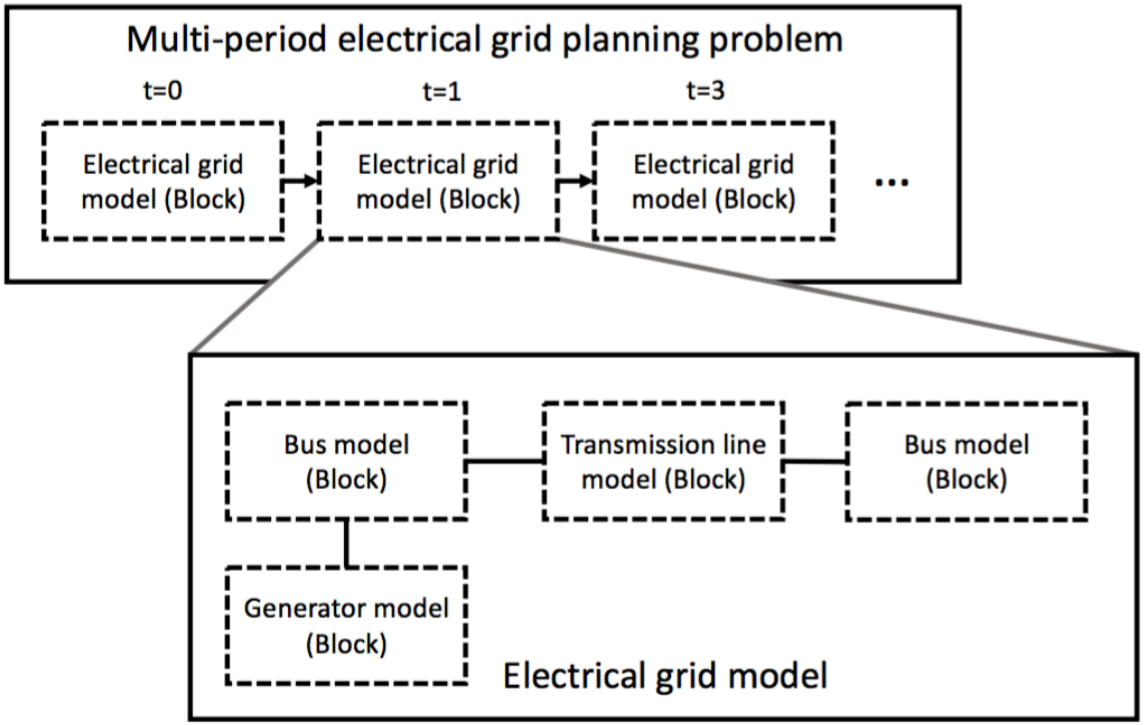
# print the solution
model.period[:].pg.display()
```

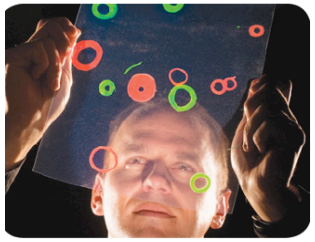
Other references

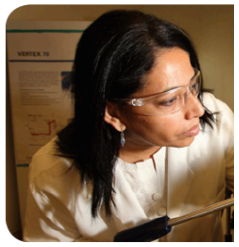


Blocks Summary

- Useful to create object-oriented models
  - (e.g, engineering equipment, physical concepts)
- Useful for other hierarchical structure (temporal, scenario)
- Basis for numerous other extensions *(more in later units)*







Sandia National Laboratories

Exceptional service in the national interest

7. Model Transformations

PYOMO

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Is this an *optimization model*?

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

Models are for *Modelers*

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

- 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*

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

- 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*

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

- 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*?

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

- 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*

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

- 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,

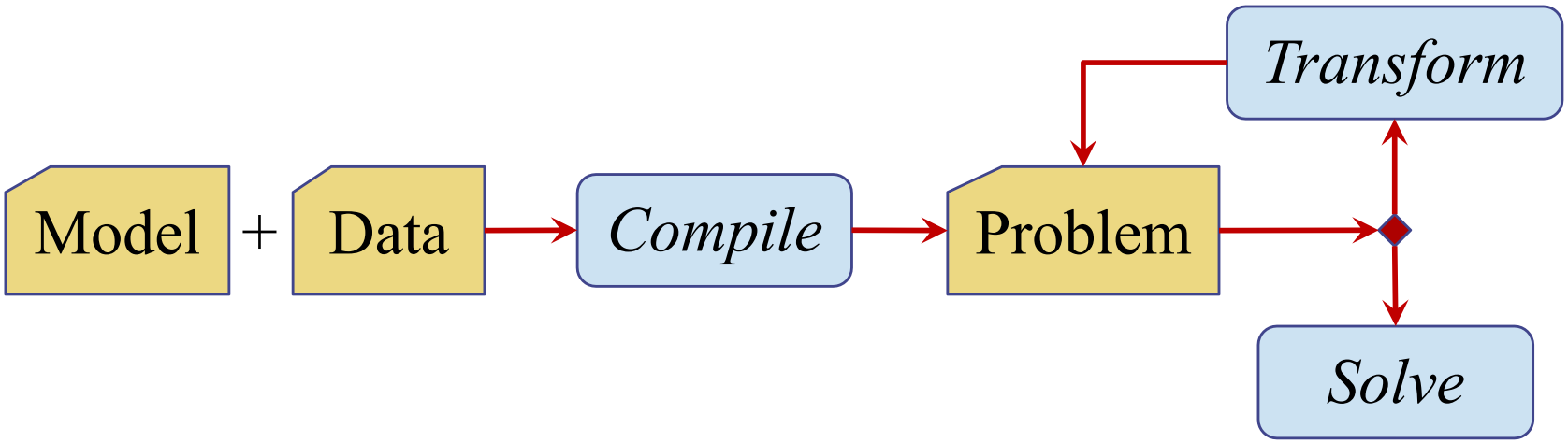
$$absX = \sqrt{x^2 + \varepsilon}$$

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

- 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

A growing library of transformations

- Bilevel optimization
  - Linear dual reformulation
  - Linear complementarity (KKT) reformulation
- Complementarity / Equilibrium constraints
  - Nonlinear relaxation
  - Disjunctive relaxation
  - “Standard” form relaxation
- Disjunctive programming
  - Big-M reformulation
  - Convex Hull reformulation
  - Cutting planes-based strengthened Big-M
  - Hybrid Basic-Step based algorithm
  - Fix disjuncts
- Dynamic systems
  - Collocation on finite elements
  - Finite difference discretization
- Structural transformations
  - Relax discrete variables
  - Standard linear form
  - Dual transformation
  - Fix discrete variables
  - Nonnegative variables
  - Expand connectors
  - Add slack variables
- Contributed transformations
  - Constraints to var bounds
  - Deactivate trivial constraints
  - Detect implicitly fixed vars
  - Variable initialization
  - Remove zero terms
  - Propagate var bounds, fixed flags

Applying transformations

- Transformations can be retrieved from a global registry

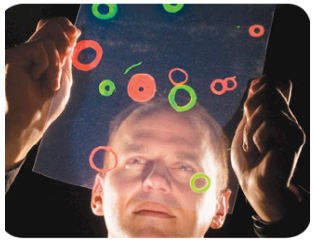
```
import pyomo.environ as pe
transform = pe.TransformationFactory('core.relax_integrality')
```

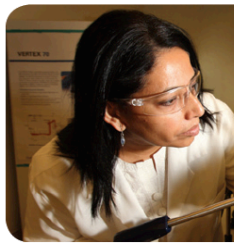
- Transformations can be applied
  - "in place" (modifying the model passed on)

```
transform.apply_to(model)
```

  - "out of place" (return a new model leaving original unchanged)

```
new_model = transform.create_using(model)
```





Sandia National Laboratories

Exceptional service in the national interest

Dynamic Systems

PYOMO

PYOMO DAE

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Examples of Dynamic Optimization

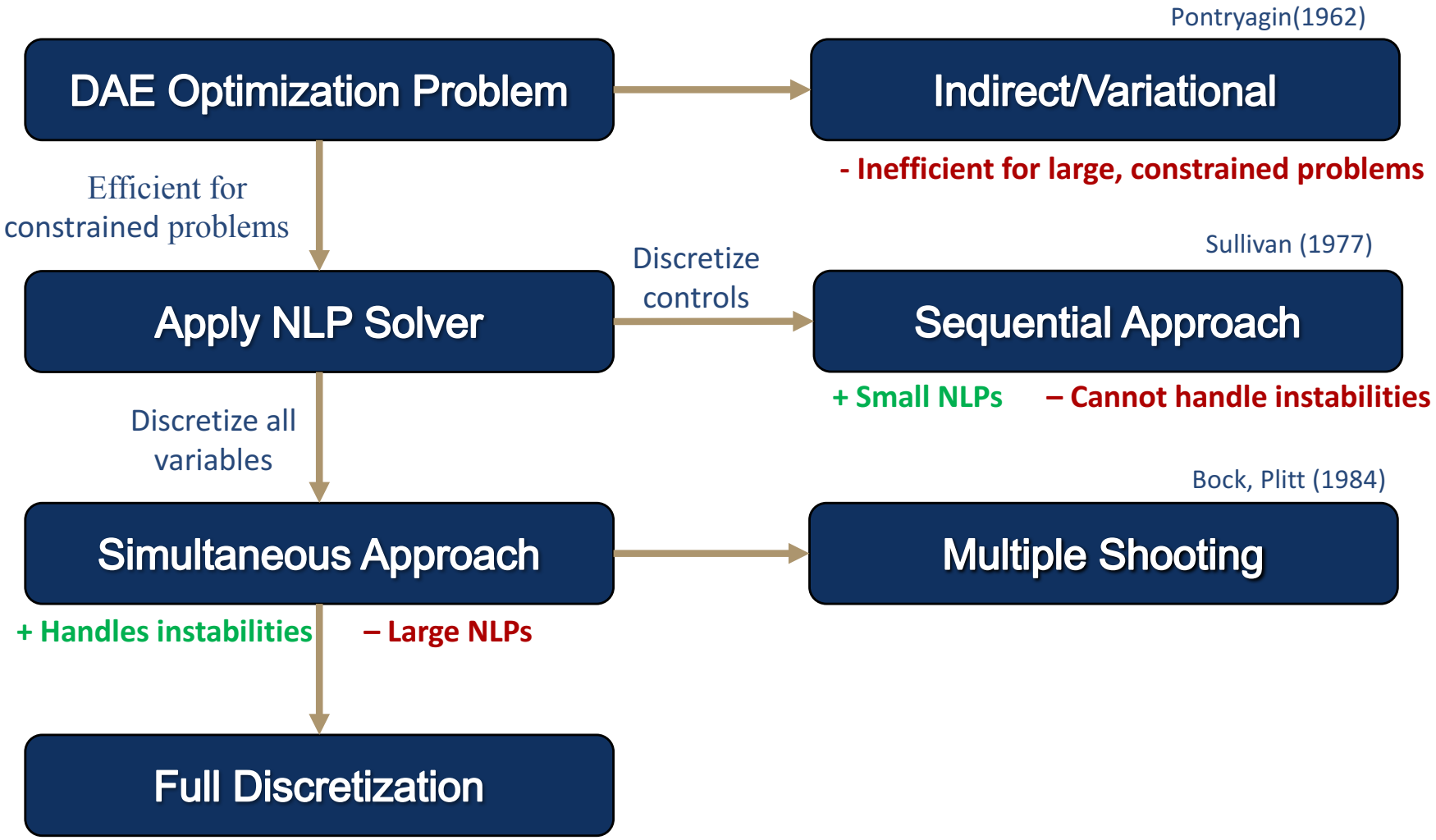
- Parameter Estimation
- Nonlinear model predictive control
- Batch process operation
- Reactor design

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

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

Solution Approaches

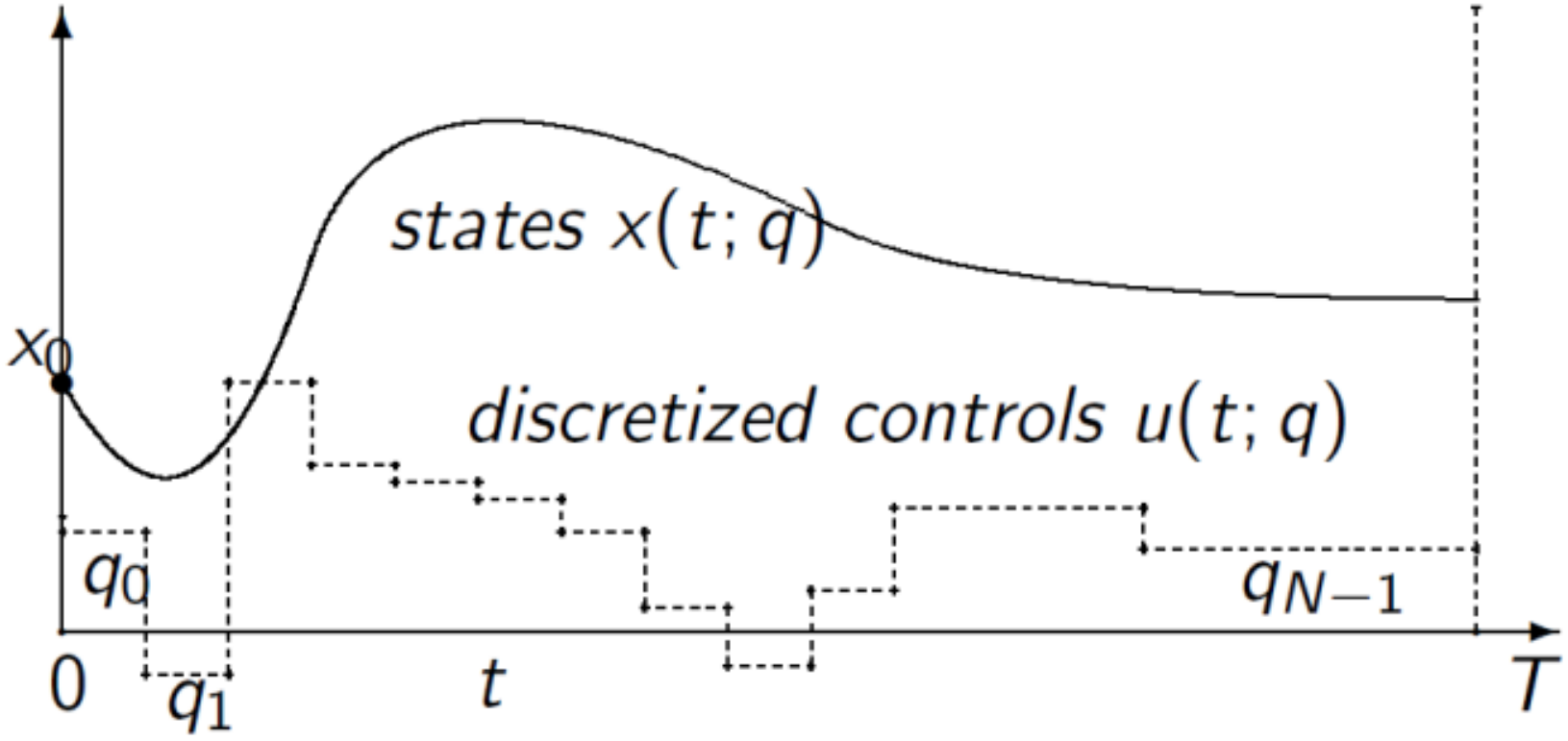
[Figure from L.T. Biegler (2007)]



Sequential Approach

[Figure from M. Diehl]

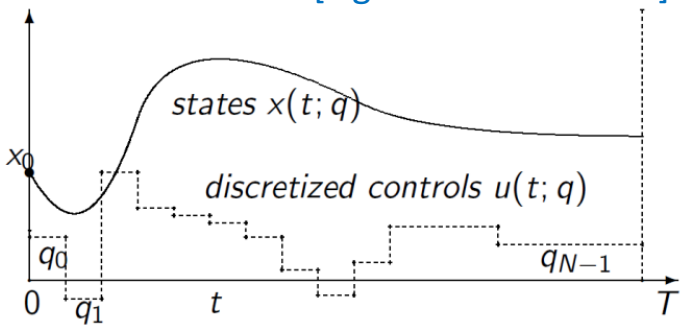
- Single Shooting Method

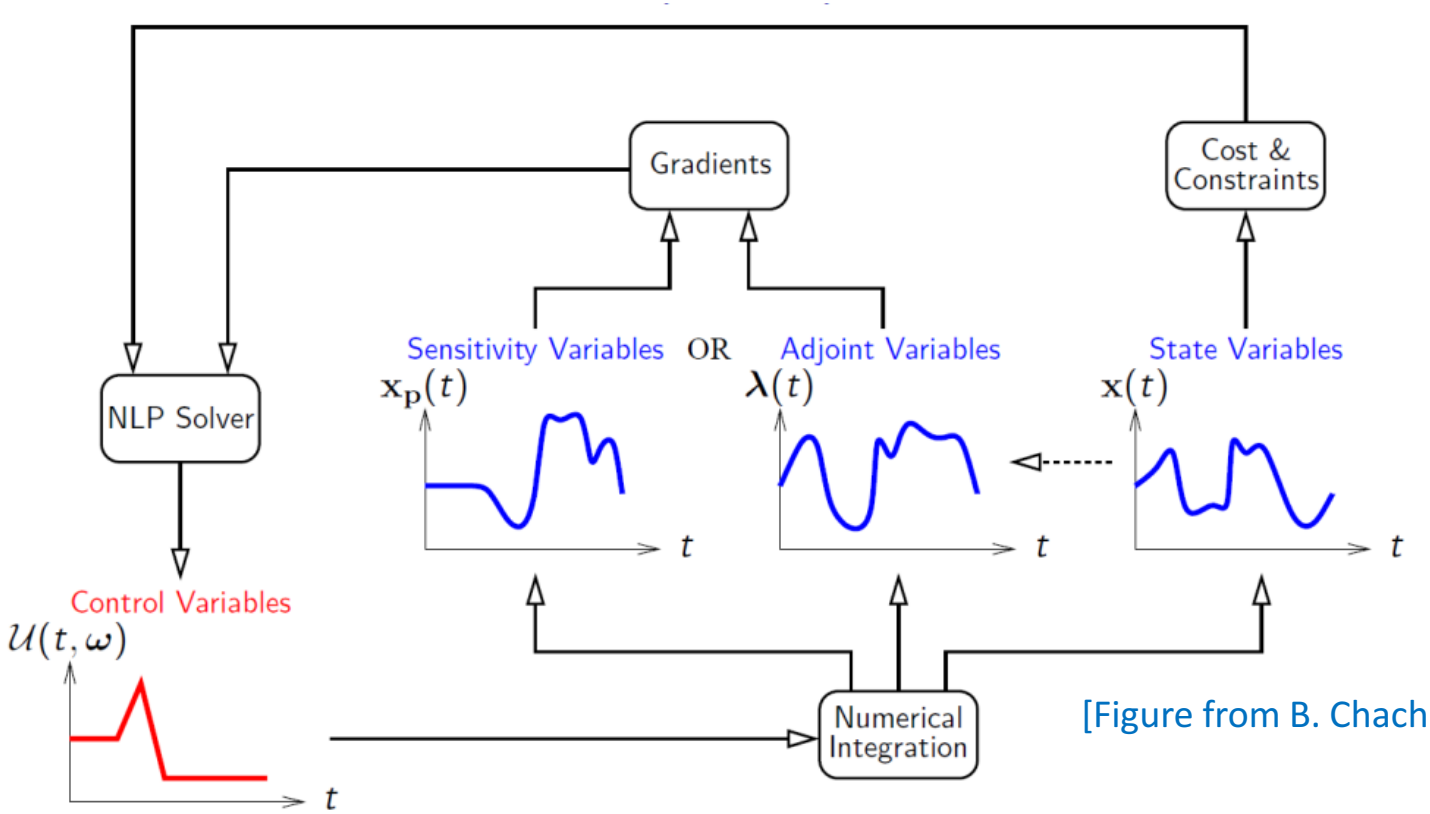


Sequential Approach

- Single Shooting Method

[Figure from M. Diehl]

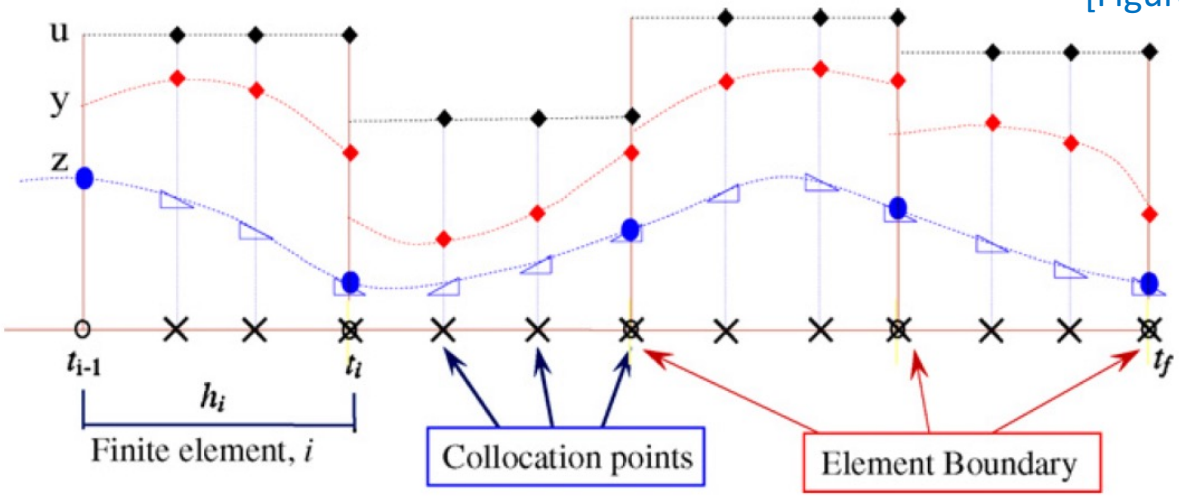




[Figure from B. Chachuat (2009)]

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 states
  - 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

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

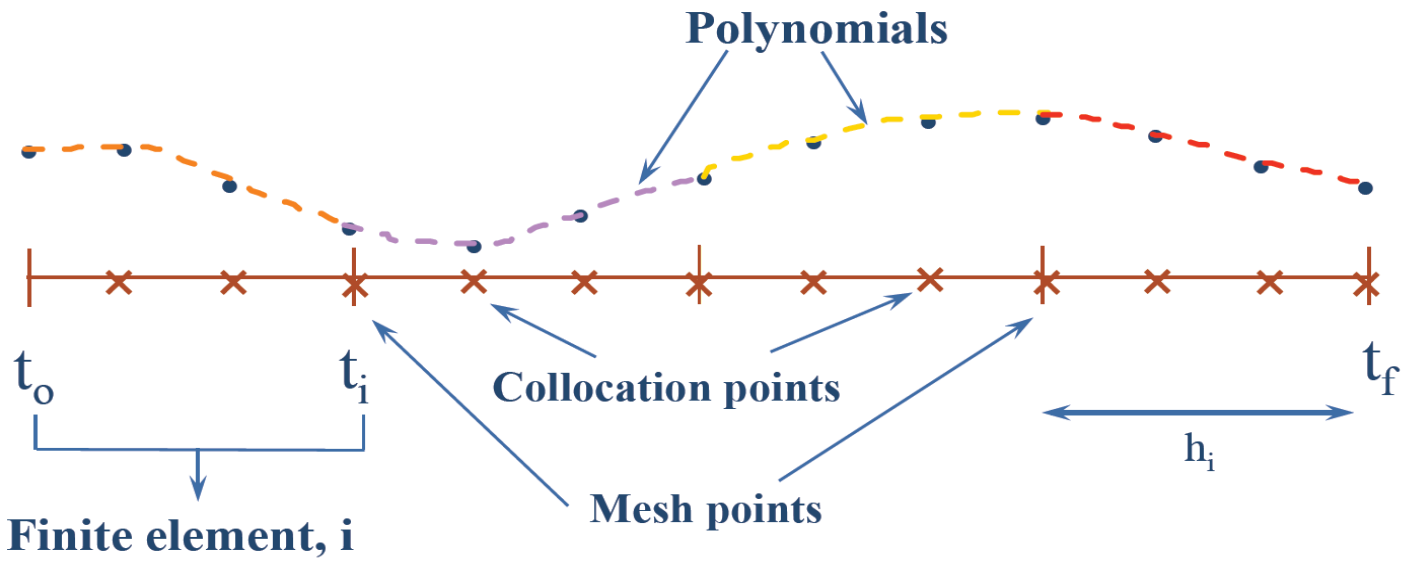
Forward Difference →

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

Backward Difference →

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

- Collocation over finite elements



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

Small Example – Exercise Solution

- Solve the differential equation using a backward finite difference scheme

```
from pyomo.environ import *

numpoints = 10
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)
```

Small Example – Exercise Solution

- Plotting

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

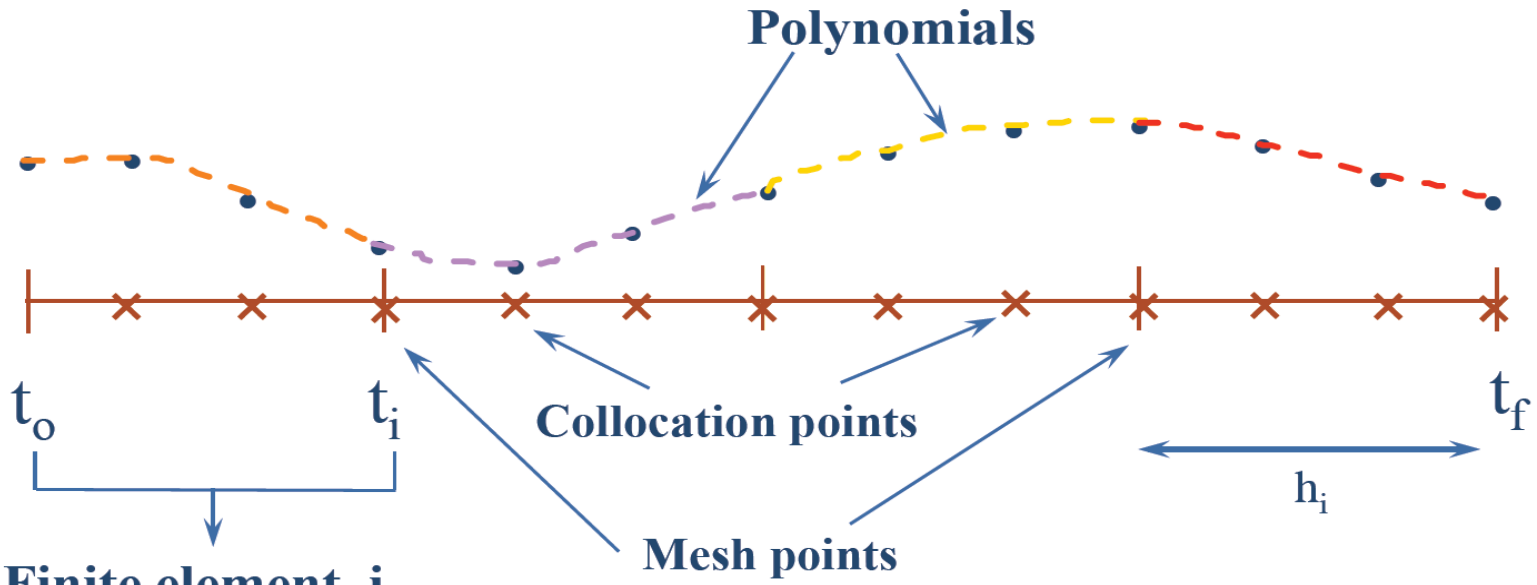
import matplotlib.pyplot as plt

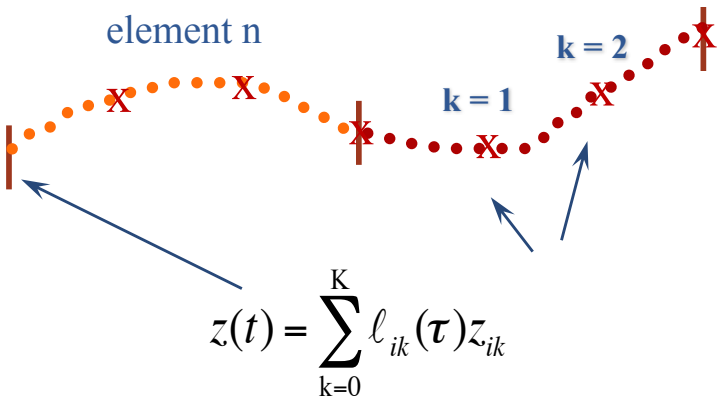
analytical_t = [0.01*i for i in range(0,101)]
analytical_z = [(4*t-3)/(4*t+1) for t in analytical_t]

findiff_t = [m.h*i for i in m.points]
findiff_z = [value(m.z[i]) for i in m.points]

plt.plot(analytical_t,analytical_z,'b',label='analytical solution‘)
plt.plot(findiff_t,findiff_z,'ro--',label='finite difference solution‘)
plt.legend(loc='best‘)
plt.xlabel('t‘)
plt.show()
```

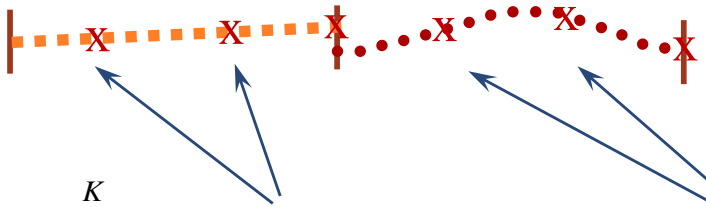
Collocation over finite elements





$$z(t) = \sum_{k=0}^{K} \ell_{ik}(\tau) z_{ik}$$

Differential variables
(Continuous)



$$y(t) = \sum_{k=1}^{K} \ell_{ik}(\tau) y_{ik}$$

$$u(t) = \sum_{k=1}^{K} \ell_{ik}(\tau) u_{ik}$$

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 0.788675	0.333333 1.000000
3	0.112702 0.500000 0.887298	0.155051 0.644949 1.000000
4	0.069432 0.330009 0.669991 0.930568	0.088588 0.409467 0.787659 1.000000
5	0.046910 0.230765 0.500000 0.769235 0.953090	0.057104 0.276843 0.583590 0.860240 1.000000

Shifted Gauss-Legendre and Radau roots as collocation points

Small Example - Collocation

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

- Solve the differential equation using collocation with a single finite element and 3 radau collocation points over the time interval $t \in [0,1]$

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

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

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

Small Example - Collocation

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

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

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

Small 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

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

which can be written out as:

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

Constant

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

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)
```

Small 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)

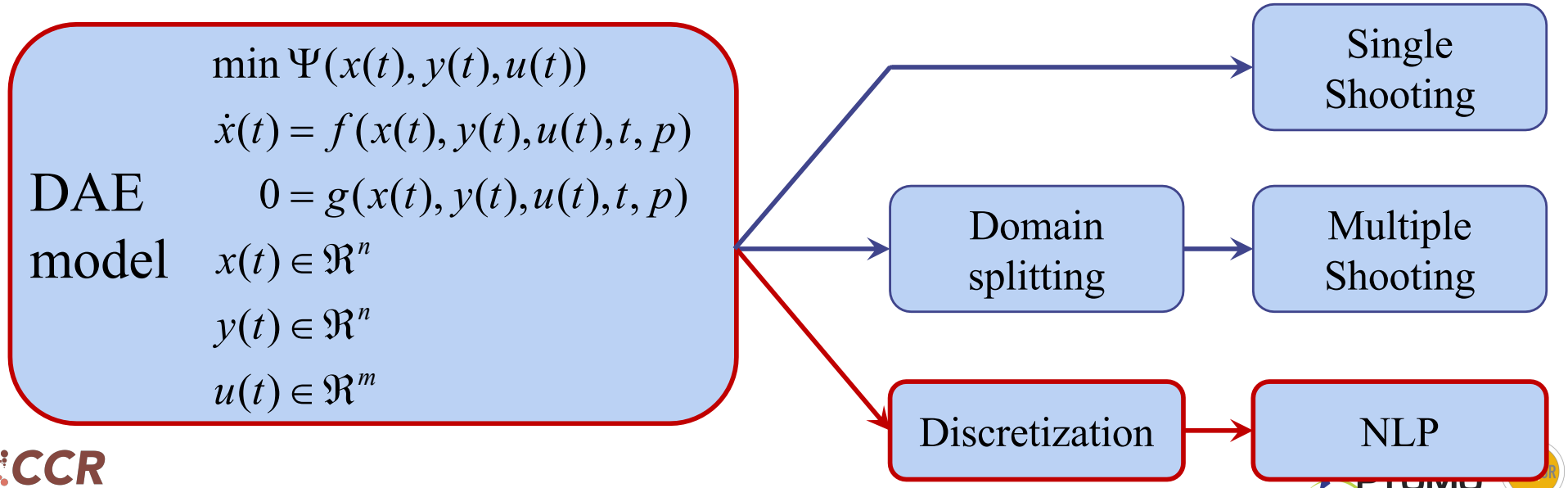
def _init_con(m):
    return m.z[0] == -3
m.init_con = Constraint(rule=_init_con)
```

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

$$z(0) = -3$$

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)

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

[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)

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

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

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

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

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

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

$$t_f = 1$$

[Jacobson and Lele (1969)]

```
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)
```

```
from pyomo.environ import *

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 (7/8)

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

[Jacobson and Lele (1969)]

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

model = m = ConcreteModel()

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

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 (8/8)

```
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)
```

Optimal Control Example - Script

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

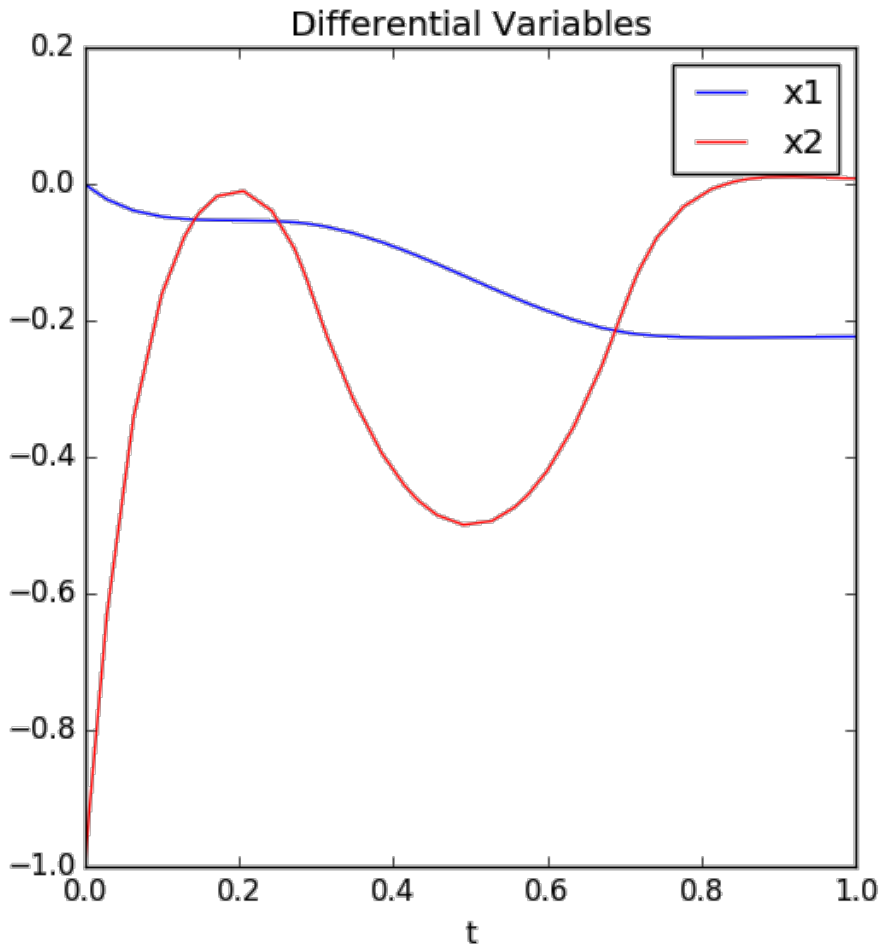
# Discretize model using radau collocation
TransformationFactory('dae.collocation').apply_to(m, nfe=7, ncp=6,
                                                  scheme='LAGRANGE-RADAU' )

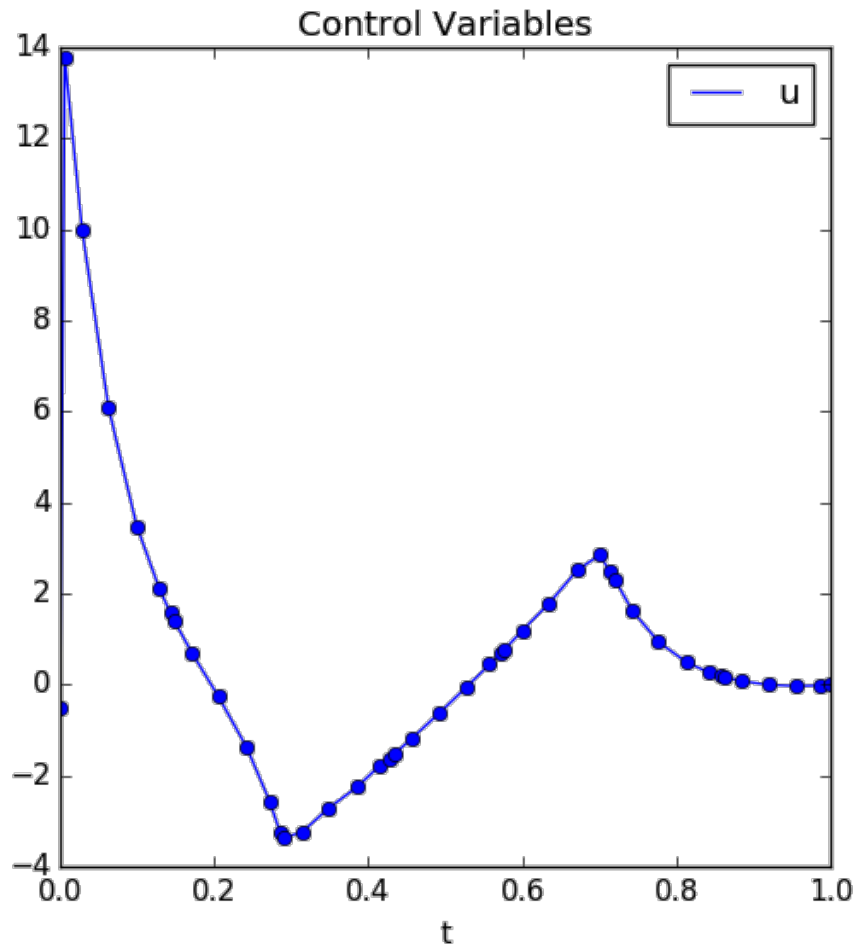
# Solve algebraic model
results = SolverFactory('ipopt').solve(m)

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' )

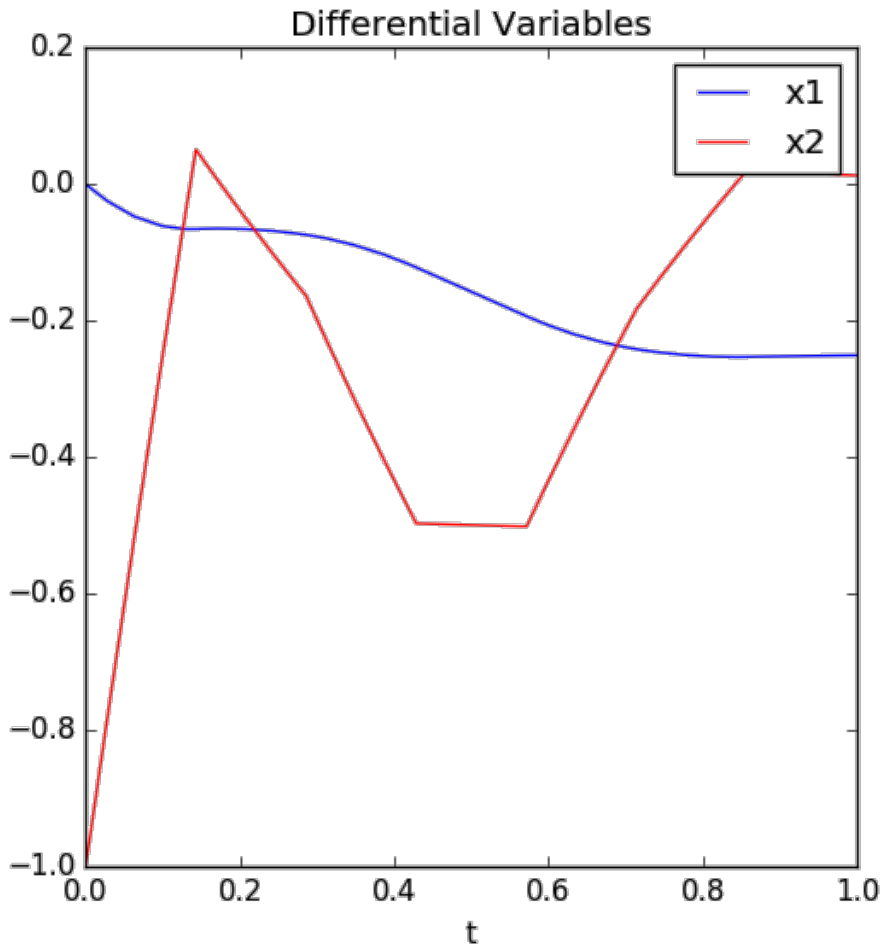
# Control variable u made constant over each finite element
discretizer.reduce_collocation_points(var=m.u, ncp=1, contset=m.t)

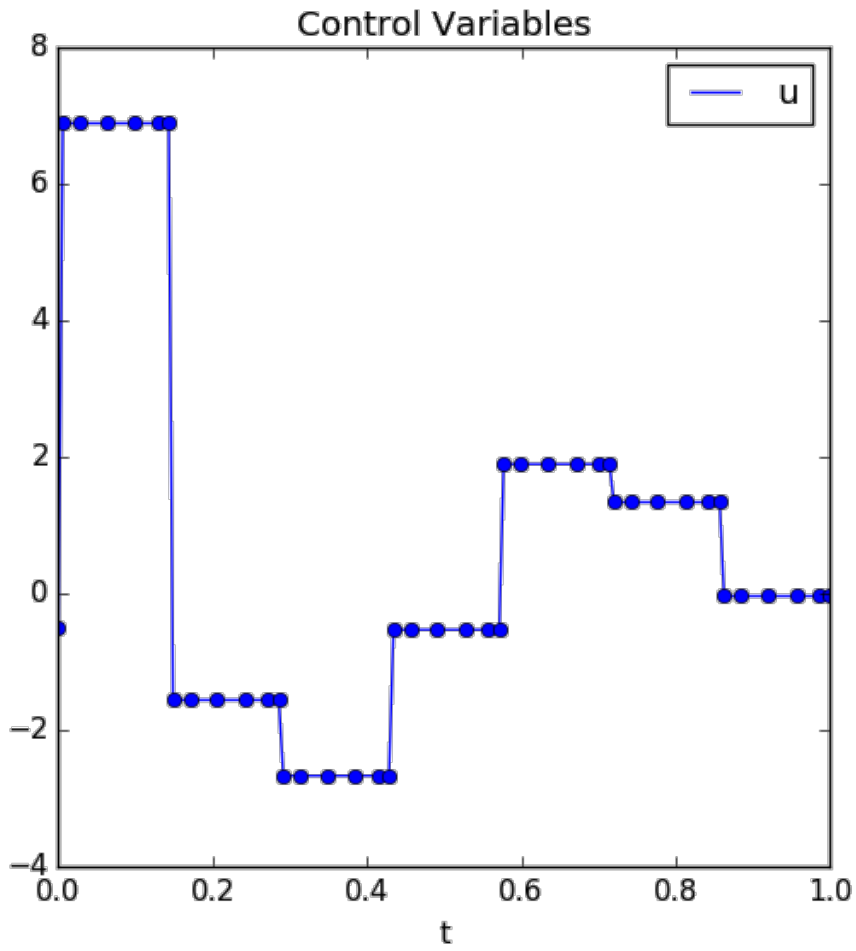
# Solve algebraic model
results = SolverFactory('ipopt').solve(m)

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

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

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

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

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

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

Want to ensure that these time points are included as discretization points

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

Initialize with these points:

```
measT = [1, 2, 3, 5]
m = ConcreteModel()
m.t = ContinuousSet(bounds=(0, 6),
                    initialize=measT)
```

Parameter Estimation Example 1

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

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

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

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

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

```
measurements = {1:0.264, 2:0.594, 3:0.801, 5:0.959}

model = ConcreteModel()
model.t = ContinuousSet(initialize=measurements.keys(), bounds=(0, 6))

model.x1 = Var(model.t)
model.x2 = Var(model.t)
model.p1 = Var(bounds=(-1.5,1.5))
model.p2 = Var(bounds=(-1.5,1.5))

model.x1dot = DerivativeVar(model.x1,wrt=model.t)
model.x2dot = DerivativeVar(model.x2)

def _init_conditions(model):
      yield model.x1[0] == model.p1
      yield model.x2[0] == model.p2
model.init_conditions = ConstraintList(rule=_init_conditions)

def _x1dot(model,i):
      return model.x1dot[i] == model.x2[i]
model.x1dotcon = Constraint(model.t, rule=_x1dot)

def _x2dot(model,i):
      return model.x2dot[i] == 1-2*model.x2[i]-model.x1[i]
model.x2dotcon = Constraint(model.t, rule=_x2dot)

def _obj(model):
      return sum((model.x1[i]-measurements[i])**2 for i in measurements.keys())
model.obj = Objective(rule=_obj)

# Discretize model using Orthogonal Collocation
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(model,nfe=8,ncp=5)

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

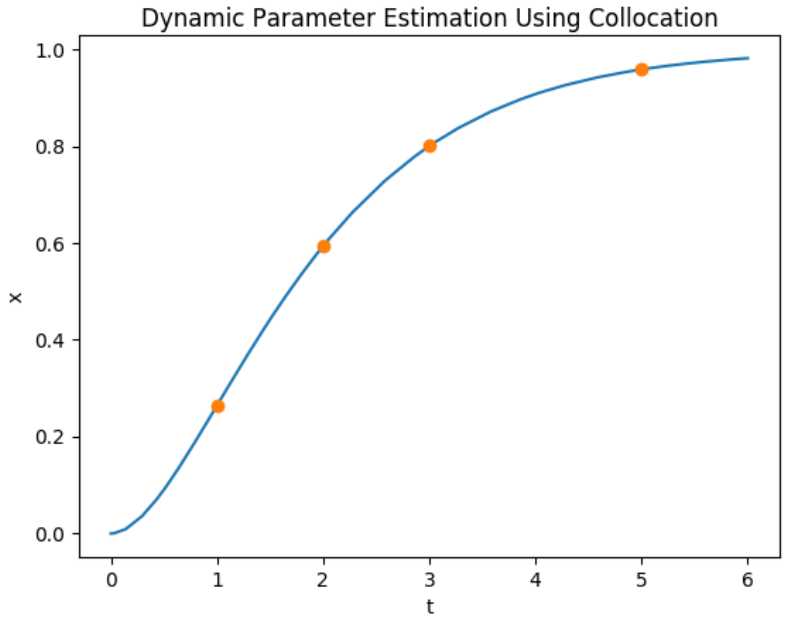
Parameter Estimation Example 1

```
t_meas = sorted(list(measurements.keys()))
x1_meas = [value(measurements[i]) for i in sorted(measurements.keys())]

t = list(model.t)
x1 = [value(model.x1[i]) for i in model.t]

import matplotlib.pyplot as plt

plt.plot(t,x1)
plt.plot(t_meas,x1_meas,'o')
plt.xlabel('t')
plt.ylabel('x')
plt.title('Dynamic Parameter Estimation Using Collocation')
plt.show()
```



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. Start with `start_param_est.py`

Solution to this problem is k1 = 5, k2 = 1

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

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 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

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

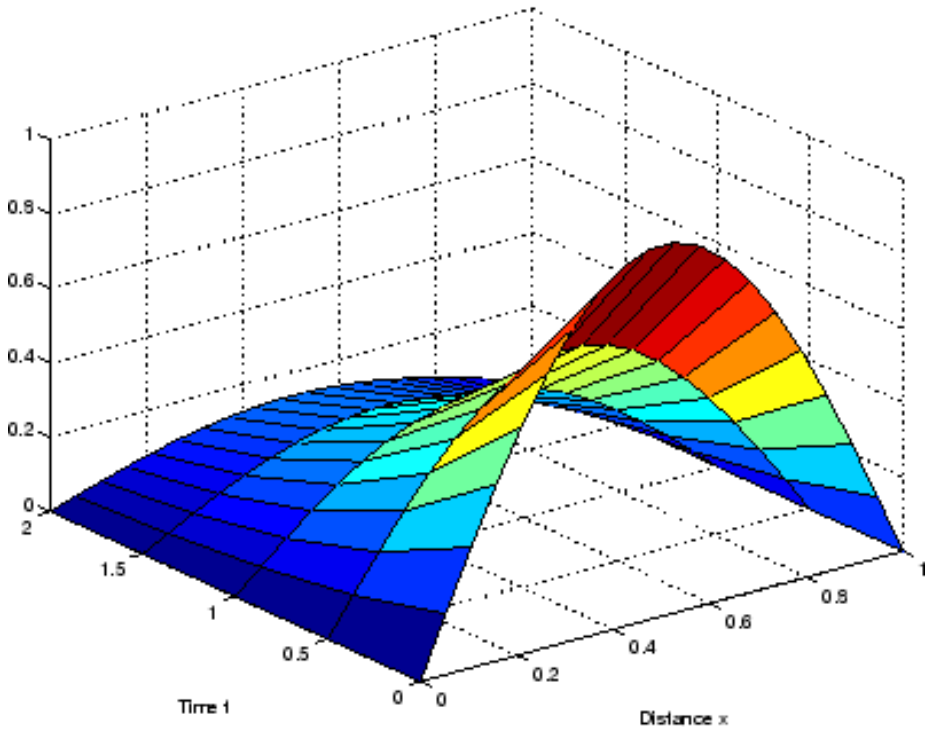
  - Initial Condition

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

  - 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)
```

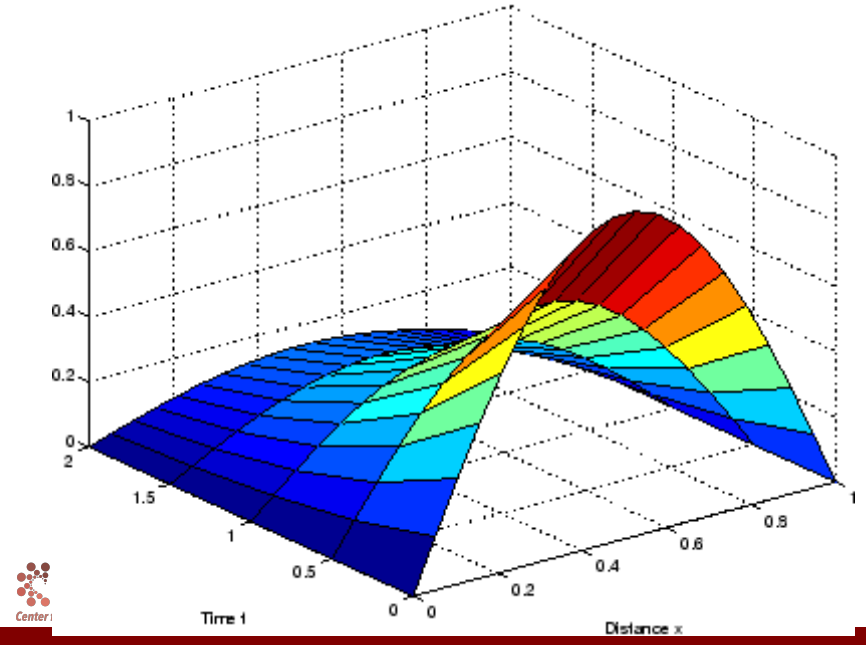
$$\pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0$$

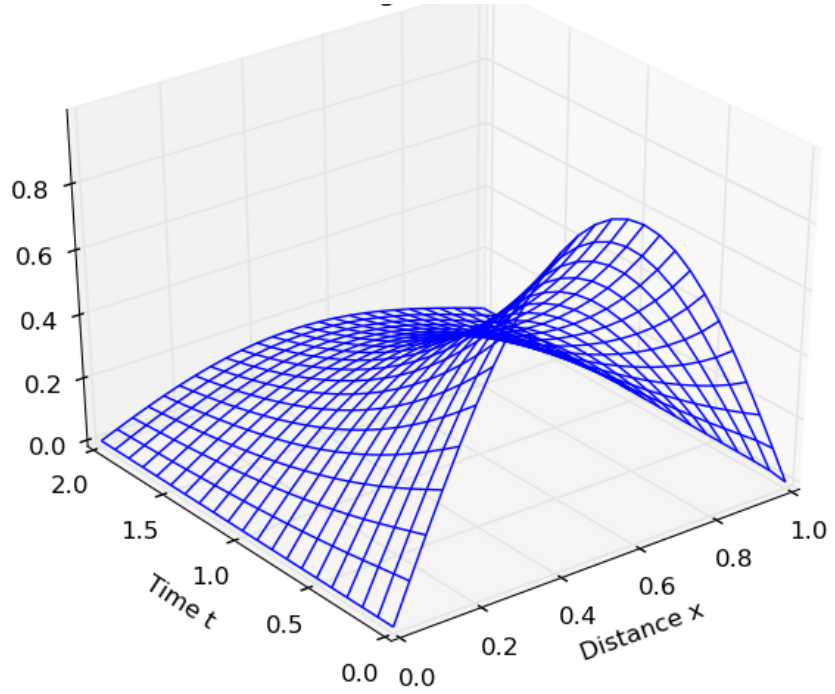
```
def _upperbound(m,j):
    return m.pi*exp(-j)+m.dudx[1,j] == 0
m.upperbound = Constraint(m.t,rule=_upperbound)
```

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)
```





Interfacing with ODE/DAE integrators

$$\frac{d\theta}{dt} = \omega$$

$$\frac{d\omega}{dt} = -b * \omega - c * \sin\theta$$

$$\theta(0) = \pi - 0.1$$

$$\omega(0) = 0$$

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

m = model = ConcreteModel()

m.t = ContinuousSet(bounds=(0.0, 10.0))
m.b = Param(initialize=0.25)
m.c = Param(initialize=5.0)

m.theta = Var(m.t)
m.omega = Var(m.t)
m.dthetadt = DerivativeVar(m.theta, wrt=m.t)
m.domegadt = DerivativeVar(m.omega, wrt=m.t)

m.theta[0].fix(3.14 - 0.1)
m.omega[0].fix(0.0)

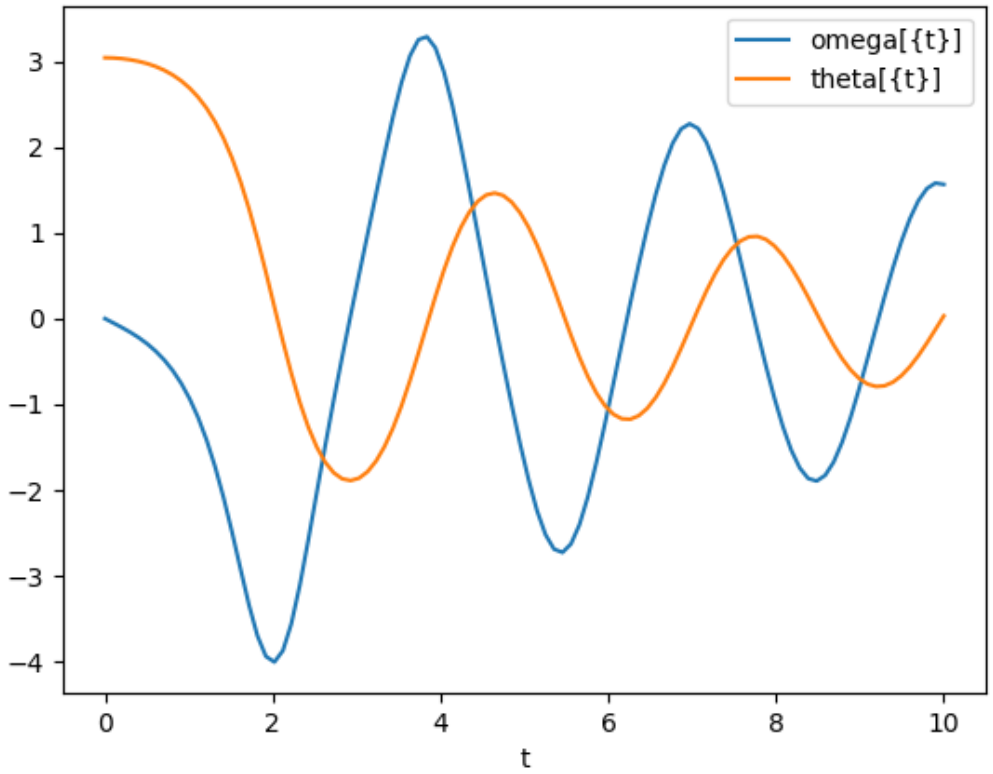
def _diffeq1(m,t):
    return m.dthetadt[t] == m.omega[t]
m.diffeq1 = Constraint(m.t, rule=_diffeq1)

def _diffeq2(m,t):
    return m.domegadt[t] == -m.b*m.omega[t] \
           - m.c*sin(m.theta[t])
m.diffeq2 = Constraint(m.t, rule=_diffeq2)
```

Interface with ODE/DAE integrators

- ODE model simulation

```
mysim = Simulator(model, package='scipy')
tsim, profiles = mysim.simulate(integrator='vode', numpoints=100)
varorder = mysim.get_variable_order()
for idx, v in enumerate(varorder):
    plt.plot(tsim, profiles[:, idx], label=v)
```

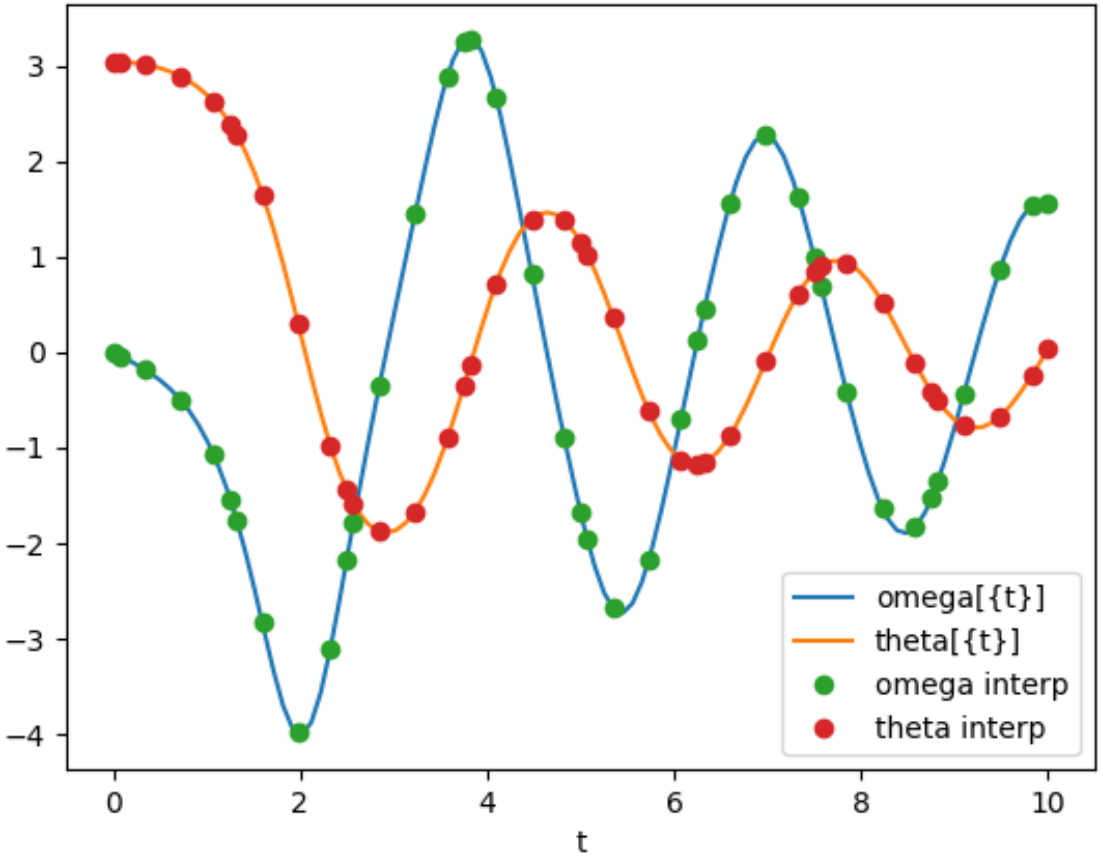


Interface with ODE/DAE integrators

- Model initialization

```
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(model, nfe=8, ncp=5)

mysim.initialize_model()
```



Use the same Pyomo model for simulation and optimization!

Simulator Exercise

- Simulate the following kinetic model using Scipy and plot the resulting profiles

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

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

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

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

$$k_1 = 5, k_2 = 1$$

- Reminder:

```
mysim = Simulator(model, package='scipy')
tsim, profiles = mysim.simulate(integrator='vode', numpoints=100)
varorder = mysim.get_variable_order()
for idx, v in enumerate(varorder):
    plt.plot(tsim, profiles[:, idx], label=v)
```

OTHER MATERIAL

Framework for Modeling 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)

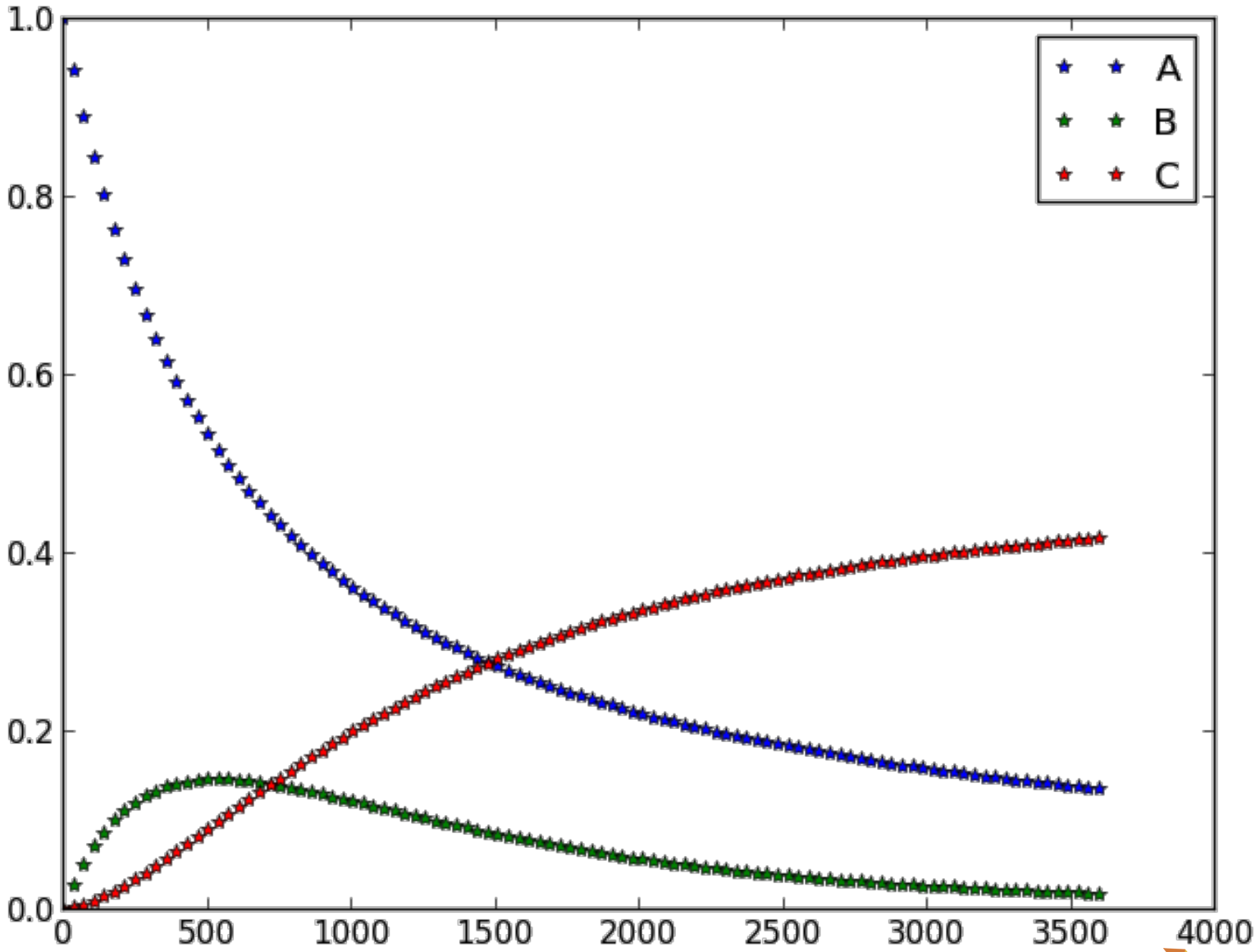
A1  = 1.32e19 # L / mol*s
A2  = 1.09e13 # 1/s
Ea1 = 140000  # J/mol
Ea2 = 100000  # J/mol
R   =  8.314  # J / K*mol
T   =    330  # K
model.k['AtoB'].fix( A1 * exp( -Ea1 / (R*T) ) )
model.k['BtoC'].fix( A2 * exp( -Ea2 / (R*T) ) )

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)

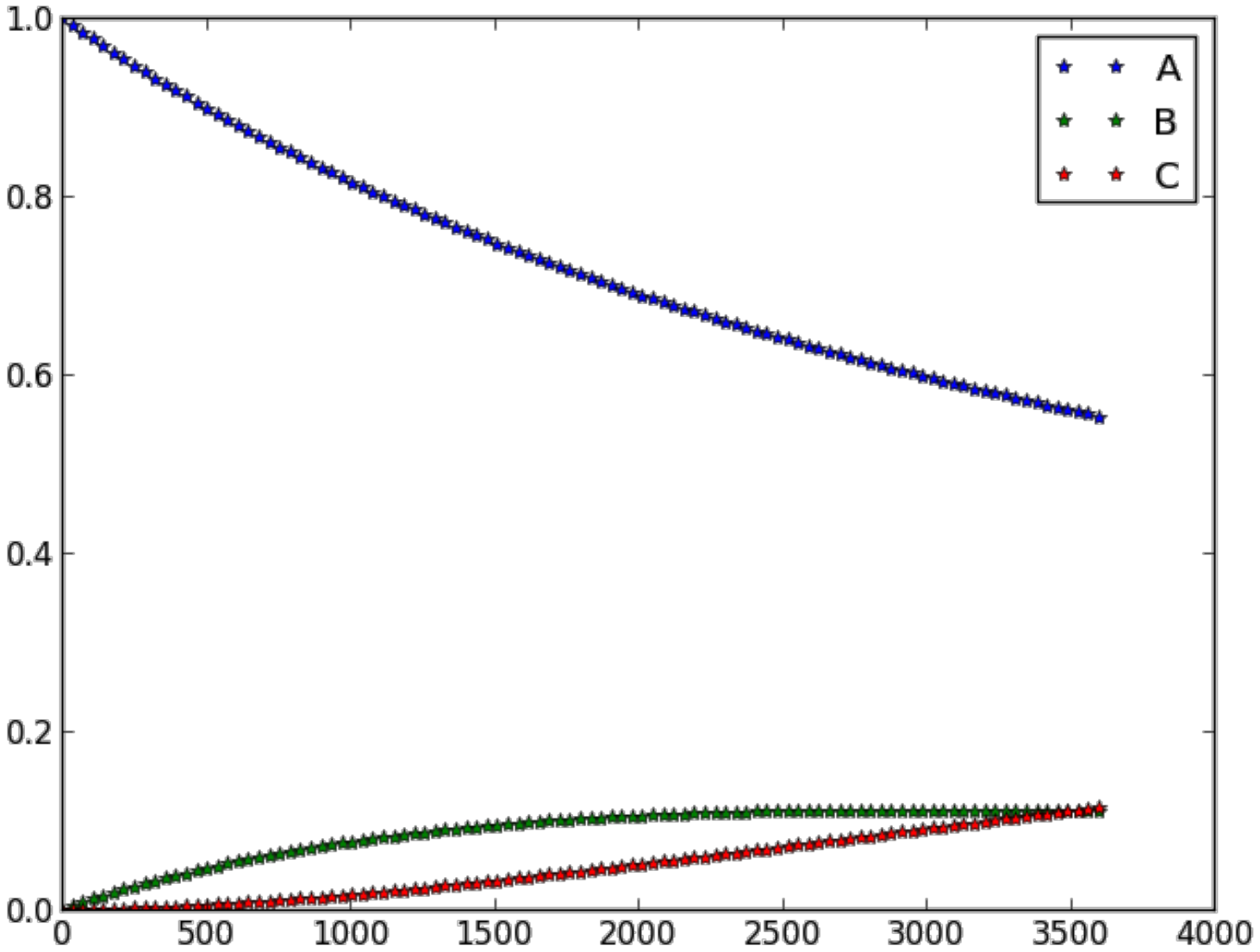
model.c[0, 'A'].fix(1)
model.c[0, 'B'].fix(0)
model.c[0, 'C'].fix(0)

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: { # ...
                     } }

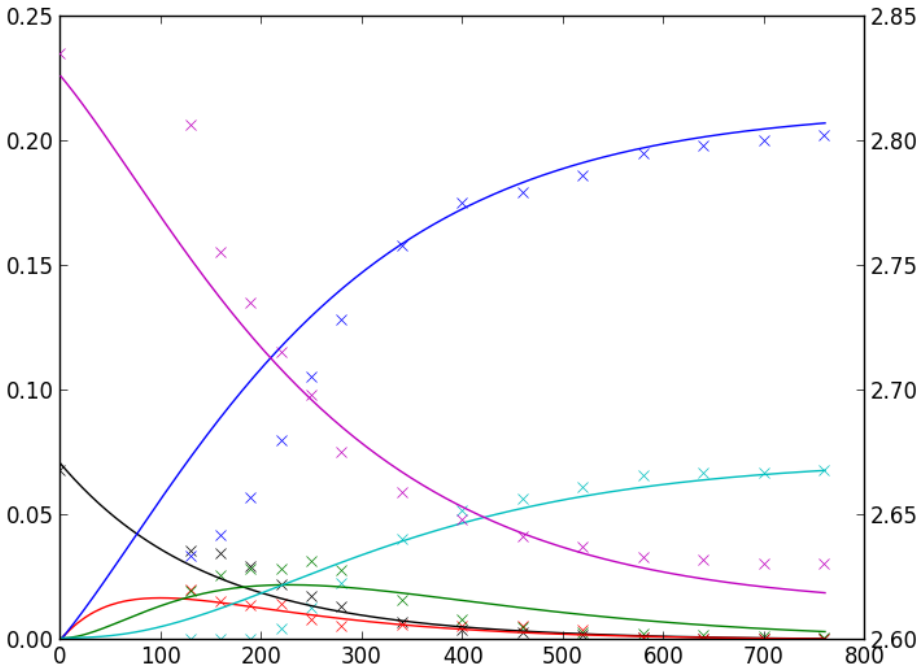
model.experiment = Block( data.keys(), rule=create_regression_model )
model.obj = Objective( sense=minimize,
                       expr=sum(b.error for b in model.experiment[:]) )

# 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

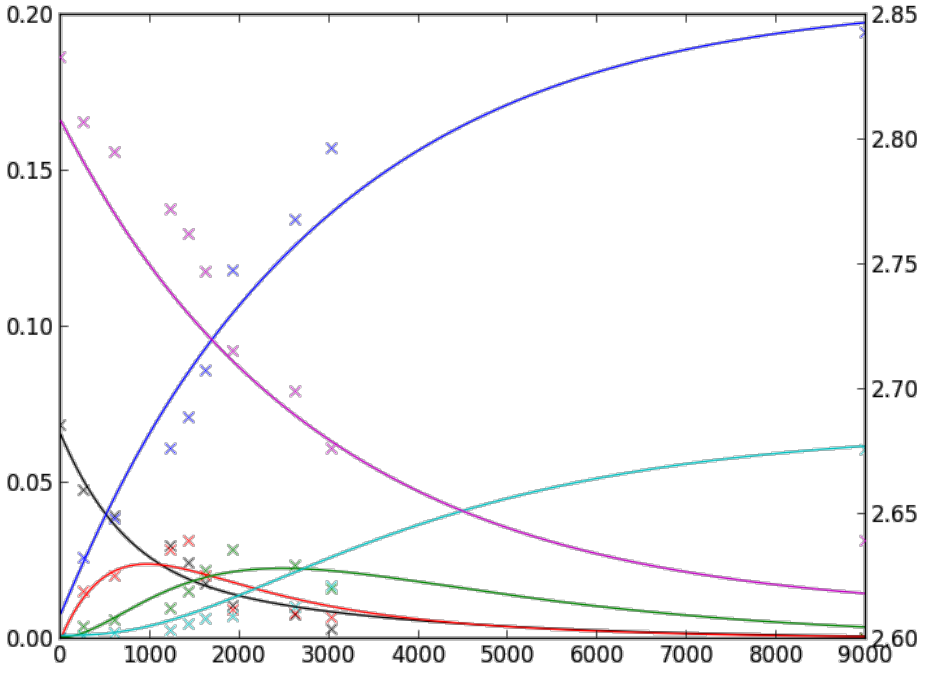
colloc.apply_to(model, nfe=100, ncp=3)
solver.solve(model, tee=True)
plot_regression_results(model)
```

Regression results

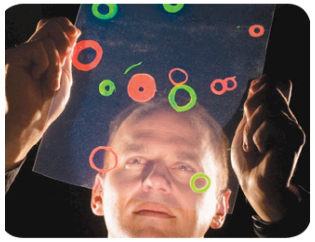
T=150

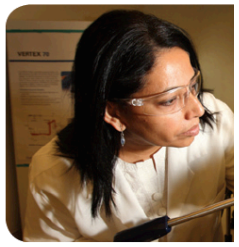


T=210



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





Sandia National Laboratories

Exceptional service in the national interest

10. Generalized Disjunctive Programming

PYOMO

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

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

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \leq 0 \\ c_k = \gamma_{ik} \end{bmatrix}$$

$$\Omega(Y) = True$$

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

- Disjunctions enforces a logical X*OR(*)* relationship
- Additional logical constraints on the indicator variables

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \leq 0 \\ c_k = \gamma_{ik} \end{bmatrix}$$

$$\Omega(Y) = True$$

- Boolean "indicator variable"
- Constraints enforced when indicator variable is True
- Parameter values set when the indicator variable is True

* Strictly speaking, this is an OR and the XOR is enforced by $\Omega(x) = True$, but in every model we have done, the effective relationship is an XOR

Implementing disjunctive programs in Pyomo

- Modeling components implemented in pyomo.gdp
  - `Disjunct`:
    - Block of Pyomo components
    - Implicit Boolean (binary) "`indicator_var`" determines if block is enforced
  - `Disjunction`:
    - Enforces logical OR / XOR across a set of Disjunct indicator variables
  - (Logic constraints on indicator variables)

- Import with
  - ```
    from pyomo.gdp import (
        Disjunct, Disjunction
    )
    ```

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \leq 0 \\ c_k = \gamma_{ik} \end{bmatrix}$$

$$\Omega(Y) = True$$
    ```

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

$$\begin{bmatrix} Y_{ik} \\ t_i + \sum_{\substack{m \in J(i) \\ m \leq j}} \tau_{im} \leq t_k + \sum_{\substack{m \in J(k) \\ m < j}} \tau_{km} \end{bmatrix} \vee \begin{bmatrix} Y_{ki} \\ t_k + \sum_{\substack{m \in J(k) \\ m \leq j}} \tau_{km} \leq t_i + \sum_{\substack{m \in J(i) \\ m < j}} \tau_{im} \end{bmatrix}$$

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

Example: Task sequencing in Pyomo (1)

```
import pyomo.environ as pe
from pyomo.gdp import Disjunct, Disjunction

model = pe.ConcreteModel()
model.JOBS = pe.Set()
model.STAGES = pe.Set()
model.IK = pe.RangeSet(0,1)

model.tau = pe.Param(model.JOBS, model.STAGES, default=0)

def _L_filter(m, val):
    i, k, j = val
    return i < k and m.tau[i,j] and m.tau[k,j]
model.L = pe.Set( initialize=model.JOBS * model.JOBS * model.STAGES,
                  filter=_L_filter )

def t_bounds(m, i):
    return (0, sum(m.tau[idx] for idx in m.tau))
model.t = pe.Var( model.JOBS, within=pe.NonNegativeReals, bounds=t_bounds )
```

Example: Task sequencing in Pyomo (2)

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

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

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

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

Simplified Disjunction syntax

- The most common Disjunctions contain *only* constraints
  - Disjunction rules can return
    - a list of relational expressions
    - a list of lists of relational expressions
  - The Disjunction will *implicitly* create the necessary Disjunct objects

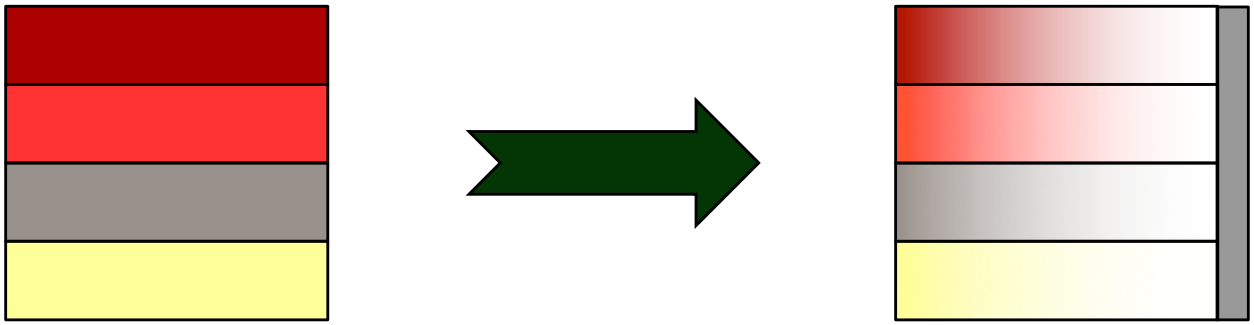
```
@model.Disjunction(model.L)
def NoCollision(m, i, k, j):
    lhs = m.t[i] + sum(m.tau[i,m] for m in m.STAGES if m<j)
    rhs = m.t[k] + sum(m.tau[k,m] for m in m.STAGES if m<j)
    return [ lhs + m.tau[i,j] <= rhs, rhs + m.tau[k,j] <= lhs ]
```

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

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

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



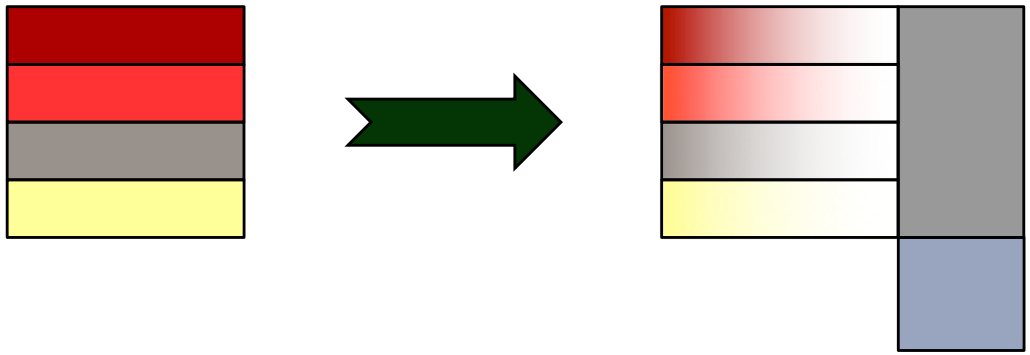
```
pe.TransformationFactory('gdp.bigm').apply_to(model)
```

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)



```
pe.TransformationFactory('gdp.chull').apply_to(model)
```

  - Algorithmic approaches
    - e.g., Trespalacios and Grossmann (2014)
  - Prematurely choosing one relaxation makes trying others difficult

Exercise: strip packing

- Given a set of rectangles, and a strip with a fixed width, determine the minimal strip length required such that the rectangles can all be cut out of the single strip (without overlapping).
  - Strip width: 10
  - Rectangles (W*L): [6, 6], [3, 8], [4, 5], [2, 3]
  - "Non overlap":

$$\begin{bmatrix} Y_L \\ x_i + l_i \leq x_j \end{bmatrix} \vee \begin{bmatrix} Y_R \\ x_j + l_j \leq x_i \end{bmatrix} \vee \begin{bmatrix} Y_U \\ y_i + w_i \leq y_j \end{bmatrix} \vee \begin{bmatrix} Y_D \\ y_j + w_j \leq y_i \end{bmatrix}$$