SAND2014-2784C

Specification and Automatic Discretization of ODE and DAE Systems in an AML

Jean-Paul Watson – Sandia National Laboratories

John D. Siirola – Sandia National Laboratories

Victor Zavala – Argonne National Laboratory

Bethany Nicholson – Carnegie Mellon University

April 9, 2014

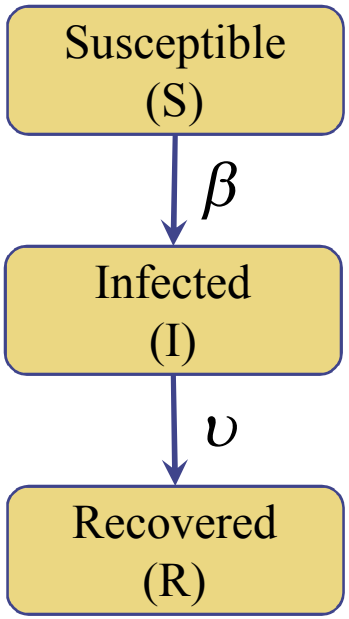
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

A bit of history / context

- Many of the ideas here originated at APMOD 2012
  - In particular, discussions with Hans Pirnay (Aachen)
- Optimization of dynamic systems is *hard*.
  - In OR, think “multi-stage” problems
  - In “engineered systems”, think differential equations
    - High fidelity *simulation* is difficult and expensive (e.g., HPC)
    - How to optimize?
      - Simulation-based optimization (*single shooting*)
      - Multiple shooting methods
      - Discretization (*collocation methods*)
    - 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
- Our idea: separate the *declaration* of dynamical models from the *solution approach* using (nearly) *automatic transformations*

Example: Disease transmission[1]

- Estimate (seasonal) transmission parameters (β) from historical data
- SIR model:

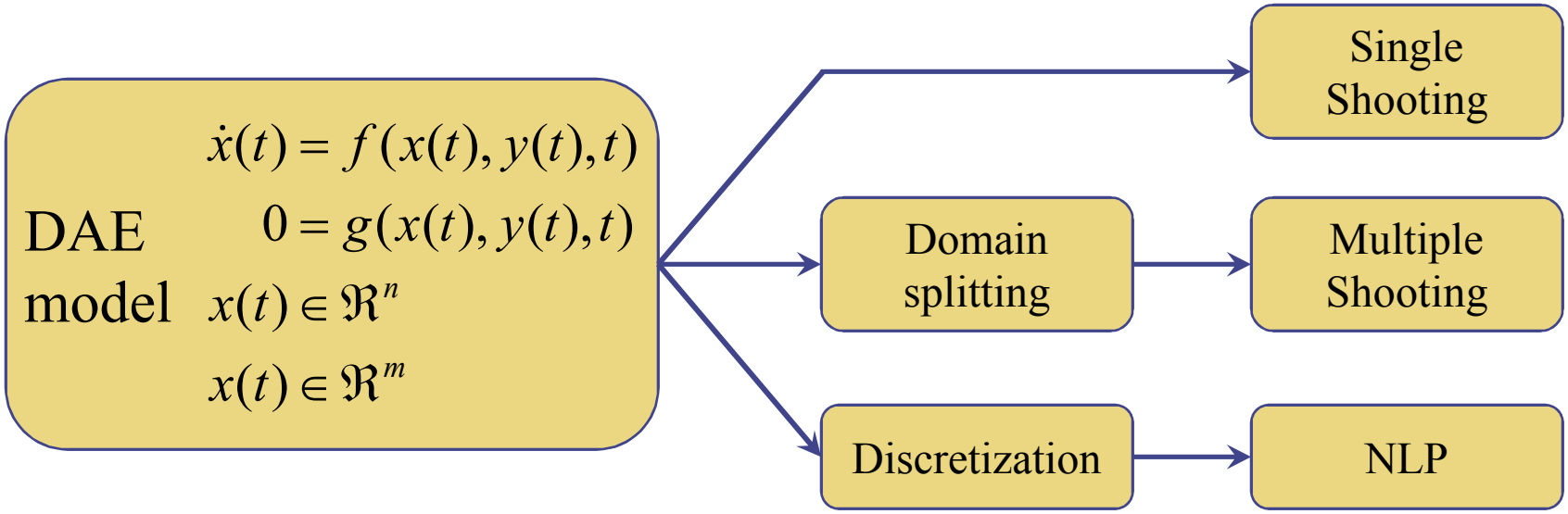


$$
\begin{aligned}
\min \quad & \omega_M \sum_{i \in \mathcal{F}} (\ln(\varepsilon_{M_i}))^2 + \omega_Q \sum_{k \in \mathcal{T}} (\varepsilon_{Q_k})^2 \\
\text{s.t.} \quad & \frac{\mathrm{d}S}{\mathrm{d}t} = \frac{-\beta(y(t))S(t)I(t)}{N(t)} \cdot \varepsilon_M(t) + B(t), \\
& \frac{\mathrm{d}I}{\mathrm{d}t}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

[1] Word, Cummings, Burke, Iamsirithaworn, and Laird. "A Nonlinear Programming Approach for Estimation of Transmission Parameters in Childhood Infectious Disease Using a Continuous Time Model", *Journal of the Royal Society Interface*. (9). August 2012.

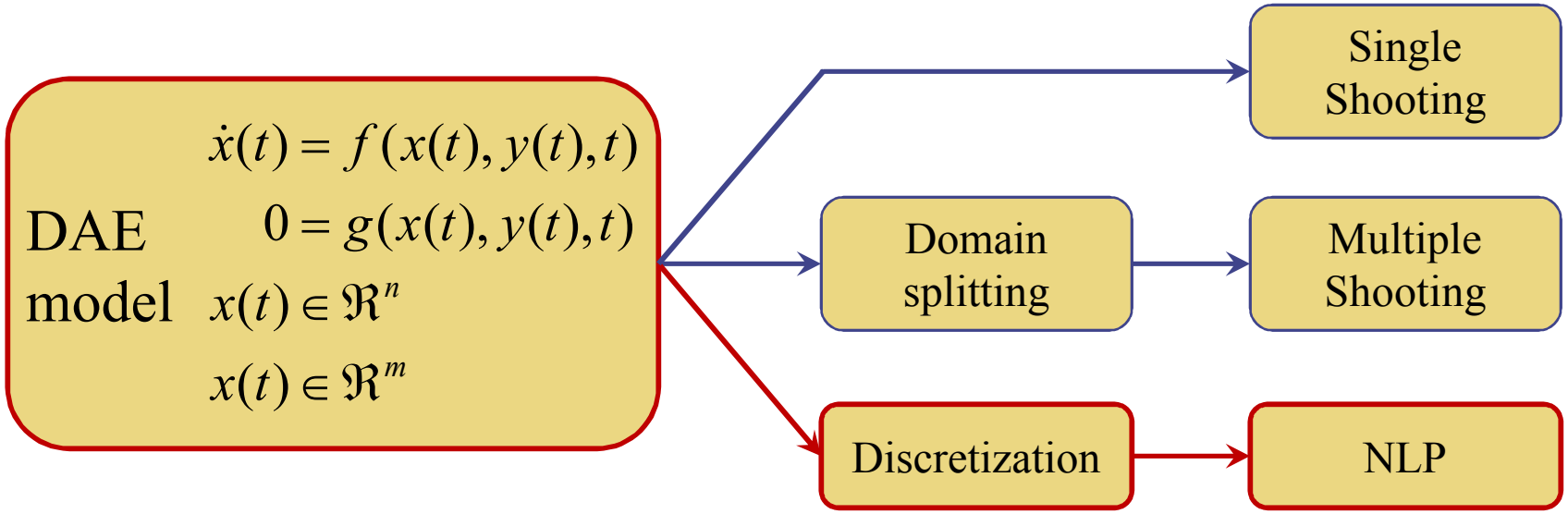
Vision: optimization of DAE systems

- Model dynamical systems in a natural form
  - Systems of Differential Algebraic Equations (DAE)
- *Transform* the DAE model into suitable optimization form
  - Retain user control, e.g. for scripting the solution approach



Vision: optimization of DAE systems

- Model dynamical systems in a natural form
  - Systems of Differential Algebraic Equations (DAE)
- *Transform* the DAE model into suitable optimization form
  - Retain user control, e.g. for scripting the solution approach
  - Initial focus: *discretization path*



Modeling DAEs in Coopr

- Extend the Pyomo object model
  - DifferentialSet: represent a continuous set that will be discretized
  - Differential: represent a differential equation of the form:

$$\frac{dx}{dt} = f(x, t, \ldots)$$

- General Model transformations
  - Standardized approach for transforming DAE to (N)LP
  - Implemented two example transformations
    - Implicit Euler Method (Backward Euler)
    - Radau Orthogonal Collocation

General Syntax

DifferentialSet

```
m.t = DifferentialSet(bounds=(0,10))
m.t = DifferentialSet(initialize=[1,2,3,4])
m.t = DifferentialSet() # Initialize using data
```

Differential

```
m.xdot = Differential(
        dvar=m.x,
        rule=_xdot,
        dset=m.t,
        bounds=(0,10),
        initialize=_init_xdot)
```

Discretization Transformation

- Transformation Framework
  1. Set Discretization Options
  2. Discretize Differential Set
  3. Update Components
  4. Add Discretization Equations
  5. Return Discretized Model

- Advanced Functionality
  - Differential sets can be discretized differently
  - Finite elements can be supplied by the user or generated
  - Discretized model can be further manipulated after transformation

Discretization Schemes

- Implicit Euler

$$\frac{dz}{dt} = f(z,t), \quad z_k = z(t_0 + kh), \quad z_{k+1} = z_k + hf(z_{k+1}, t_{k+1})$$

- Collocation

Given: dz/dt = f(z,t), z(0)

Approximate: dz/dt by Lagrange interpolation polynomials with K interpolation points, τ_k, in each finite element *i*

$$z_{ik} = z_{i-1,0} + h_i \sum_{j=1}^{K} \Omega_j(\tau_k)\dot{z}_{ij}\,, \qquad \Omega_j(\tau) = \int_0^{\tau} \ell_j(\tau')dt'\,, \qquad \ell_j(\tau) = \prod_{\substack{k=1 \\ k\neq j}}^{K} \frac{(\tau - \tau_k)}{(\tau_j - \tau_k)}, \qquad t_{ij} = t_{i-1} + \tau_j h_i$$

$$z_{i,0} = z_{i-1,0} + h_i \sum_{j=1}^{K} \Omega_j(1)\,\dot{z}_{ij}, \qquad z_f = z_{N-1,0} + h_N \sum_{j=1}^{K} \Omega_j(1)\,\dot{z}_{Nj}$$

Collocation Transformation

- Using the python library NumPy, collocation points and coefficients can be calculated for arbitrary order
- Collocation continuity equations implicitly enforced
- User can supply the finite element locations, but not collocation points

Small example: Model (1/7)

Optimal Control Problem[2]

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

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

m = ConcreteModel()

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

m.obj = Objective(expr=m.x3[1])

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

def _x1dot(m, t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

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

def _x3dot(m, t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

[2] Jacobson and Lele (1969)

Small example: Model (2/7)

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

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

m = ConcreteModel()
```


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


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

def _x1dot(m, t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

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

def _x3dot(m, t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

Small example: Model (3/7)

$$\min \; x_3(t_f)$$
$$s.t. \quad \dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_2 + u$$
$$\dot{x}_3 = x_1^2 + x_2^2 + 0.00$$
$$x_2 - 8 * (t - 0.5)^2 +$$
$$x_1(0) = 0$$
$$x_2(0) = -1$$
$$x_3(0) = 0$$
$$t_f = 1$$

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

m = ConcreteModel()

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

m.obj = Objective(expr=m.x3[1])
```

```
m = ConcreteModel()

m.t = DifferentialSet(bounds=(0,1))
m.x1 = Var(m.t)
m.x2 = Var(m.t)
m.x3 = Var(m.t)
m.u = Var(m.t)
```

```
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

Small example: Model (4/7)

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

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

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

m = ConcreteModel()

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

m.obj = Objective(expr=m.x3[1])

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


```
m.obj = Objective(expr=m.x3[1])
```


```
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

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

def _x3dot(m, t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

Small example: Model (5/7)

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

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

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

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

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

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

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

m = ConcreteModel()

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

m.obj = Objective(expr=m.x3[1])

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

def _x1dot(m, t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

def _x2dot(m, t):
    return -m.x2[t]+m.u[t]
```

```
def _x1dot(m,t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, dset=m.t,
      rule=_x1dot)
```

Small example: Model (6/7)

$$\min \quad \int_0^1 x_3(t)\,dt$$

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

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

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

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

$$x_1(0) = 0$$

$$x_2(0) = -1$$

$$x_3(0) = 0$$

$$t_f = 1$$

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

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

m.obj = Objective(expr=m.x3[1])

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

def _x1dot(m, t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

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

def _x3dot(m, t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

Small example: Model (7/7)

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

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

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

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

$$t_f = 1$$

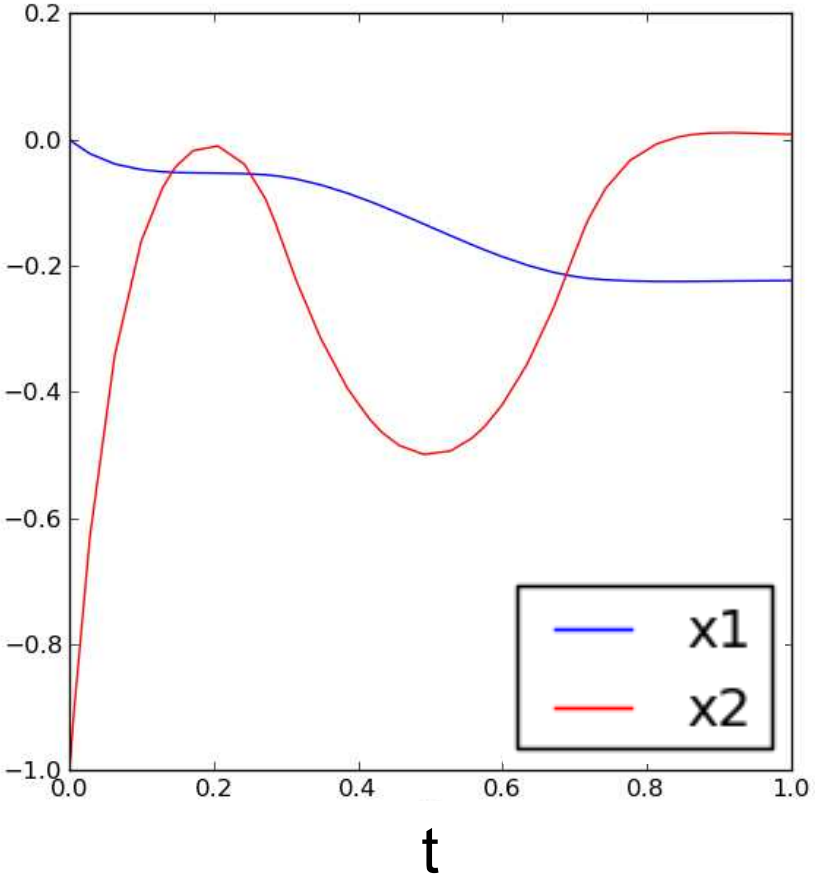
```
def _init(m):
    yield m.x1[0] == 0
    yield m.x2[0] == -1
    yield m.x3[0] == 0
    yield ConstraintList.End
m.init_conditions = ConstraintList(rule=_init)

def _x1dot(m, t):
    return m.x2[t]
m.x1dot = Differential(dv=m.x1, rule=_x1dot)

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

def _x3dot(m, t):
    return m.x1[t]**2+m.x2[t]**2+0.005*m.u[t]**2
m.x3dot = Differential(dv=m.x3, 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)
```

Small example: Script

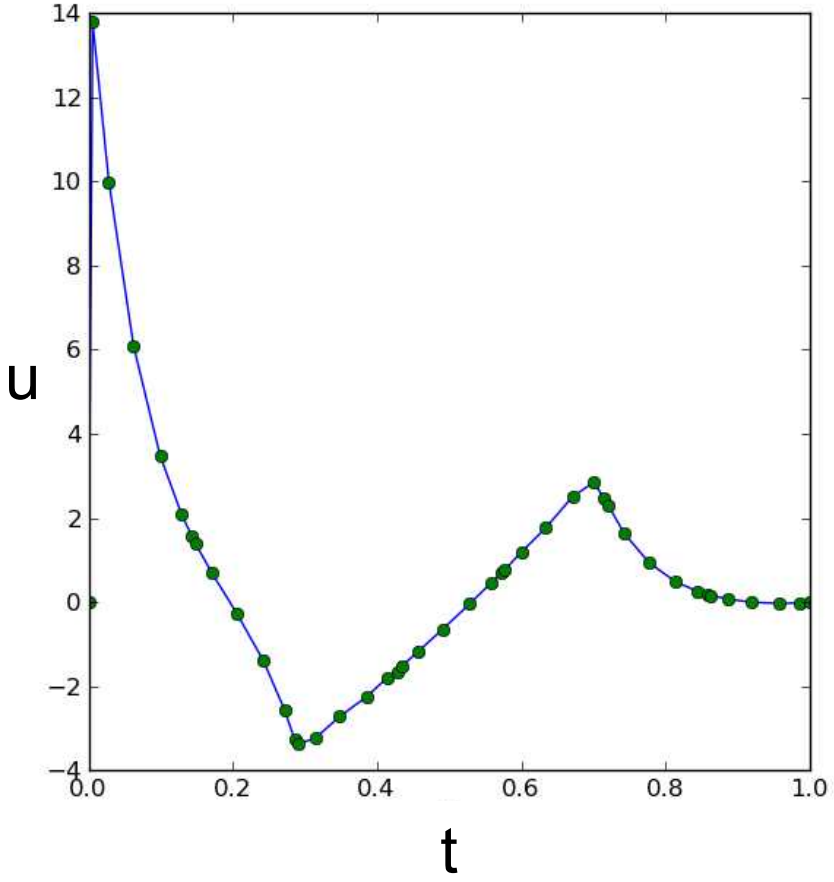
```
import coopr.environ
from coopr.opt import SolverFactory

# Create model from file
from smallExample import model
instance = model.create()

# Discretize model using radau collocation
disc_instance = instance.transform(
                        “dae.collocation_discretization”,
                        nfe=7, ncp=6)

# Solve discrete model
opt = SolverFactory(‘ipopt’)
results = opt.solve(disc_instance)
disc_instance.load(results)
```

Small example: Results

Differential Variables



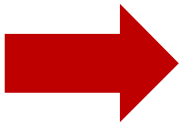
Control Variable



Sandia National Laboratories

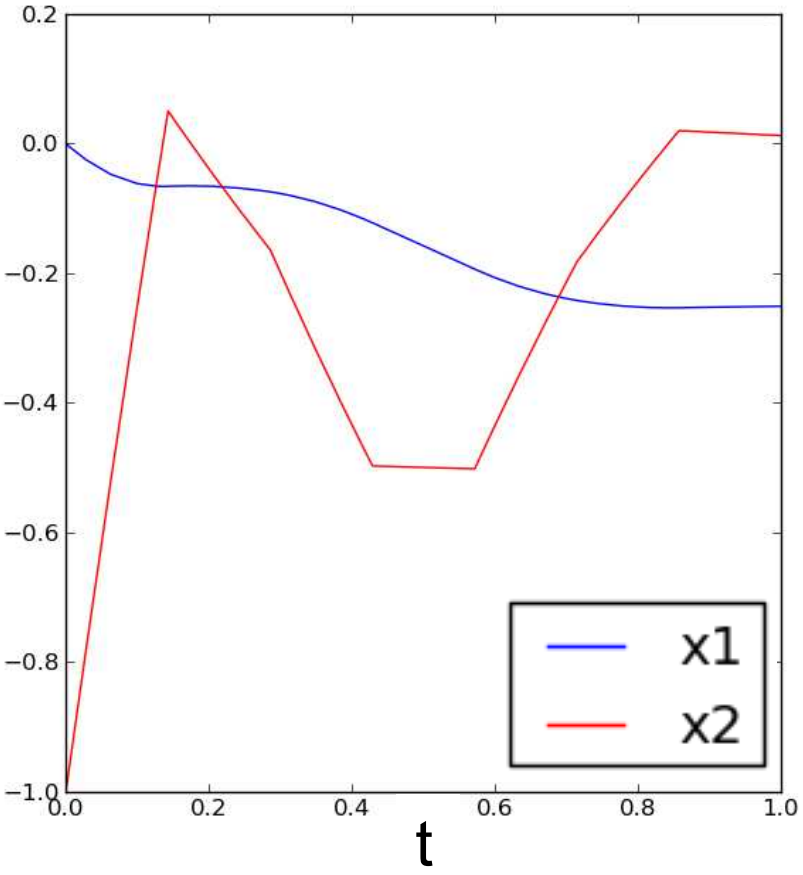
Small example: Script 2

```
import coopr.environ
from coopr.opt import SolverFactory
from coopr.pyomo import TransformationFactory

# Create model from file
from smallExample import model
instance = model.create()

# Discretize model using radau collocation
transform =
     TransformationFactory('dae.collocation_discretization')
disc_instance = transform.apply(instance, nfe=7, ncp=6)

# Control variable u made constant over each finite element
disc_instance = transform.reduce_collocation_points(
                          var=instance.u, ncp=1,
                          diffset=instance.t)

opt = SolverFactory('ipopt')
results = opt.solve(disc_instance)
disc_instance.load(results)
```

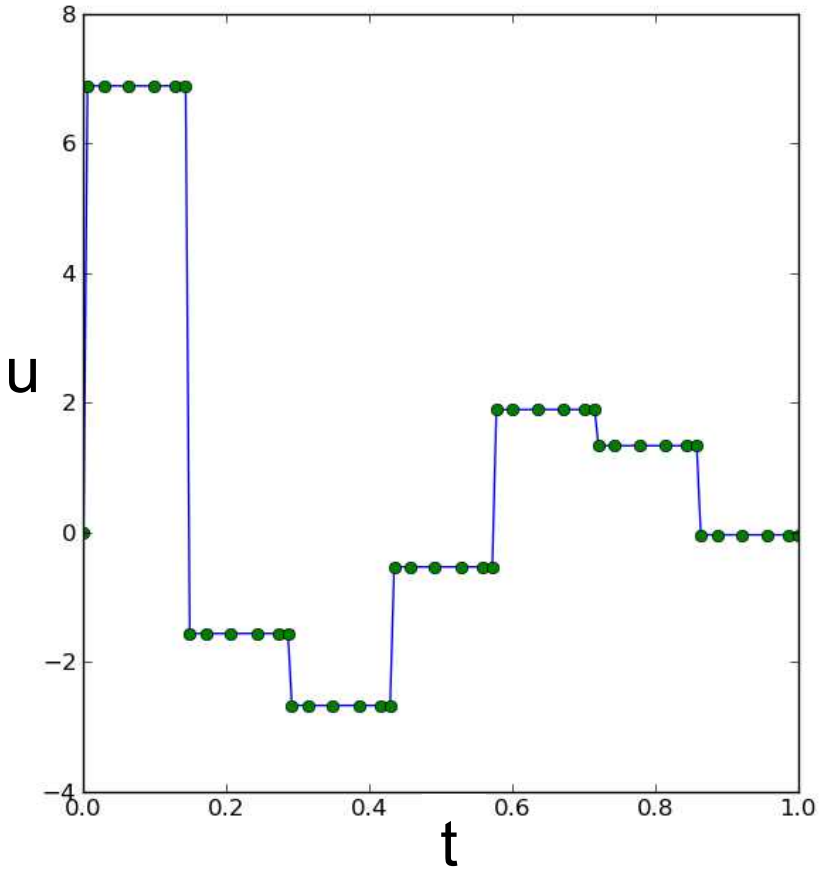


Small example: Results 2

Differential Variables



Control Variable



Disease model parameter estimation

- SIR DAE model shown earlier taken from [1]

- Parameter estimation problem with:
  - ~ 10,500 variables
  - ~ 10,000 constraints
  - 3 differential equations
  - 520 finite elements
  - 3 collocation points

[1] Word, Cummings, Burke, Iamsirithaworn, and Laird, "A Nonlinear Programming Approach for Estimation of Transmission Parameters in Childhood Infectious Disease Using a Continuous Time Model", Journal of the Royal Society Interface. vol 9, August 2012, pp. 1983-1997.

Converting the model: Indexing sets

- Integer index to float index

```
m.x[1,1] → m.x[1.088588]
```

- Iterating over all time points

$$\sum_{i \in FE} \sum_{j \in CP} x_{i,j}$$ Original Model

$$\sum_{t_i \in t} x_{t_i}$$ Implementation with coopr.dae (simplified, but must be declared after iscretization)

Converting the model: Scaling

$$z_{ik} = z_{i-1} + h_i \sum_{j=1}^{K} \Omega_j(\tau_k)\dot{z}_{ij}$$

→Enforce scaling in Differential Set

$$z_{ik} = z_{i-1} + h_i \sum_{j=1}^{K} \Omega_j(\tau_k)\dot{z}_{ij}$$

→Enforce scaling in Differential

Computational results: Disease model

Radau Collocation Implementation	By Hand	Using coopr.dae Scaling in DifferentialSet	Using coopr.dae Scaling in Differential
Creation Time (CPU secs)	2.7	136.13	76.25
Solve Time (CPU secs)	2.468	5.123	6.791
IPOPT Iterations	27	43	50
Objective ($x10^{-5}$)	1.4716	1.4716	1.4716

Future work

- Extending current modeling components
  - Support for general forms (e.g., $0 = F(x(t), \dot{x}(t), y(t))$)
  - Explicit characterization of state / control variables
- New transformations
  - Alternative collocation points
  - Transformations for multiple shooting methods
- Translating solutions from the discretized / decomposed models back to the original model space
  - Exact
  - Approximate
- Cleaner support for auxiliary functionality
  - Operations over DifferentialSets (e.g., sums over time)
  - Manipulating / reducing the discretized model

Acknowledgements

- Bethany Nicholson - Carnegie Mellon University
- Carl Laird – Purdue University
- Gabriel Hackebeil – Texas A&M University
- Larry Biegler – Carnegie Mellon University

Sandia National Laboratories' Laboratory-Directed Research and Development Program and the U.S. Department of Energy's Office of Science (Advanced Scientific Computing Research program) funded portions of this work.