SAND2021-14194C

Diagnostic Tools for Dynamic Models of Chemical Process Systems in Pyomo.DAE

Robert Parker [1], Bethany Nicholson [2], John Siirola [2], Lorenz Biegler [1]

[1] Carnegie Mellon University

[2] Sandia National Laboratories

November 7, 2021

AIChE Annual Meeting

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.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

Motivation: Dynamic optimization is not easy

Have a working steady state model

Add differential equations to make the model dynamic

Fix initial conditions and inputs

Solve square problem (simulation) as initialization

Unfix inputs, add objective function

Dynamic optimization problem fails to solve

We would like to know if we have made a mistake

A dynamic model is one in which many variables and constraints are indexed by time

Pyomo DAE provides:

- `DerivativeVar`
- `ContinuousSet` (time)
- Discretization equations

$$\frac{dC_A}{dt} = F_{\text{in}}C_{A,\text{in}} - F_{\text{out}}C_A - Vr_A$$

Many variables and equations are indexed by time

```
model.temperature[t]
model.concentration[t, "CO2"]
```

Fundamental operations:

- Get all variables and constraints at a point in time
- Find "the same" variable at different points in time

```
model.concentration[t, "CO2"] = model.concentration[t0, "CO2"]
```

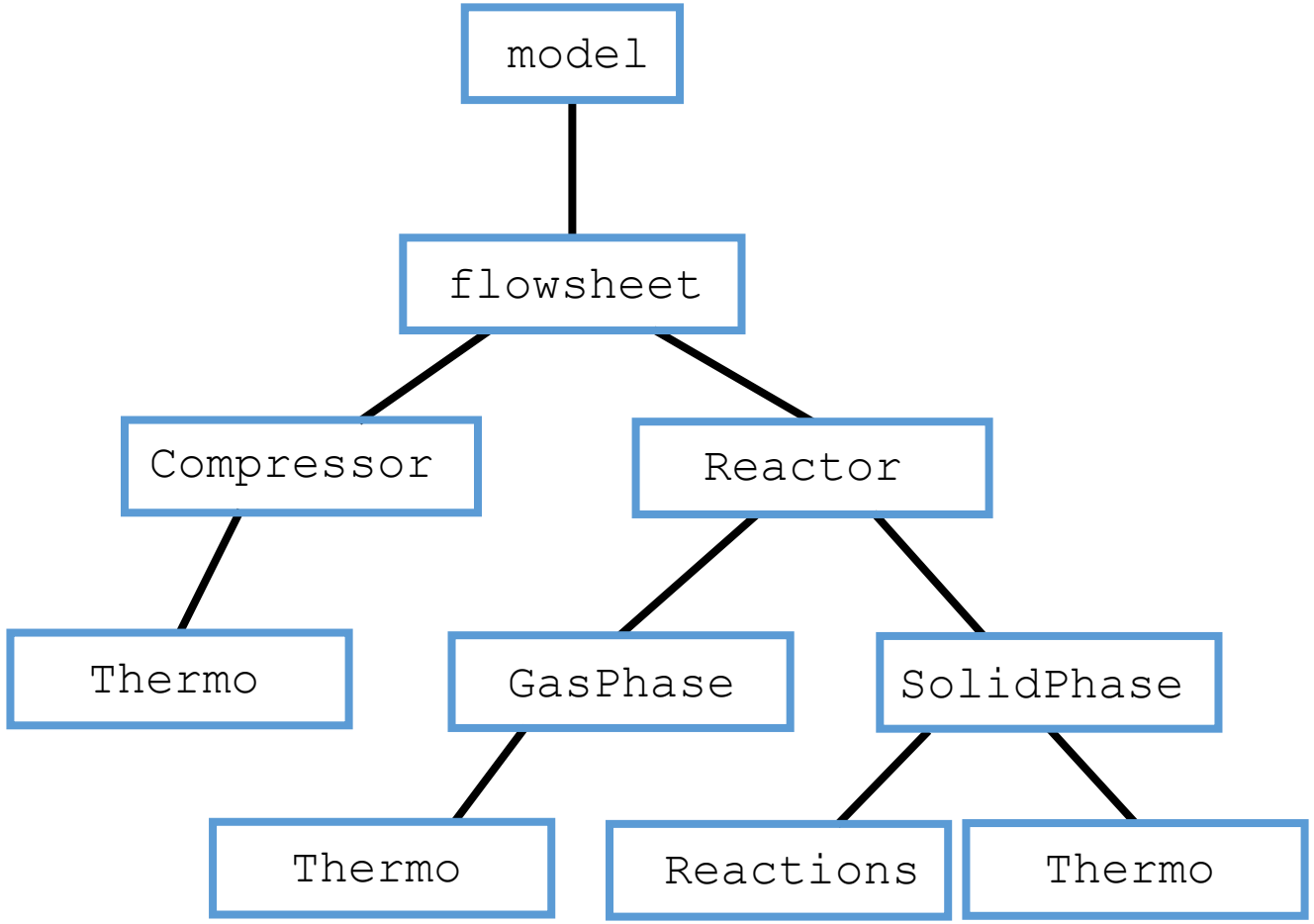
Pyomo models are trees of blocks

Blocks can represent:

- Unit models
- Thermodynamic models
- Scenarios
- Time periods

Blocks are useful for:

- Abstraction
- Namespace organization
- Modular model construction
- Programmatic model construction




```
model.flowsheet.Compressor.Thermo
```

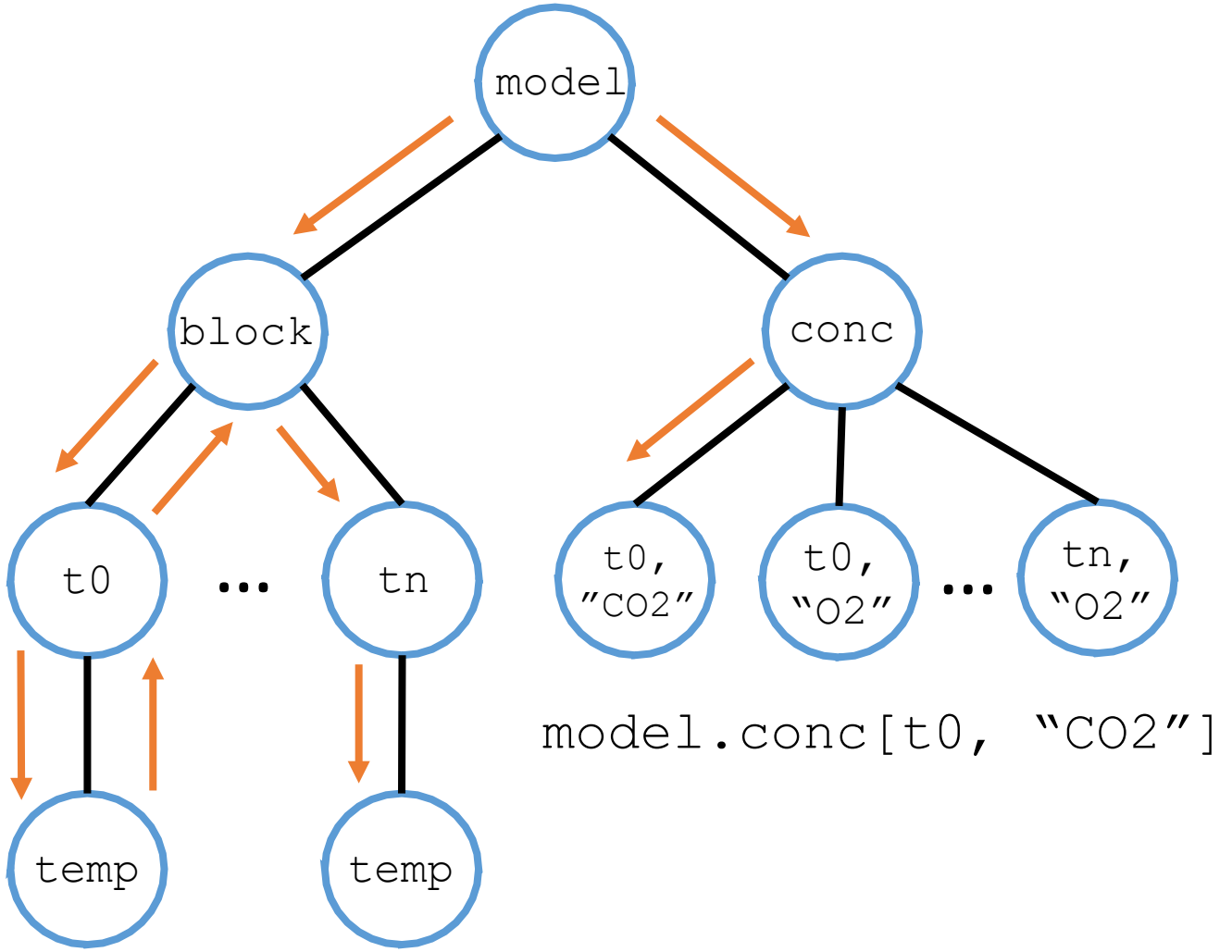
Time index may appear at any level of the model tree

To identify variables/constraints at a point in time:

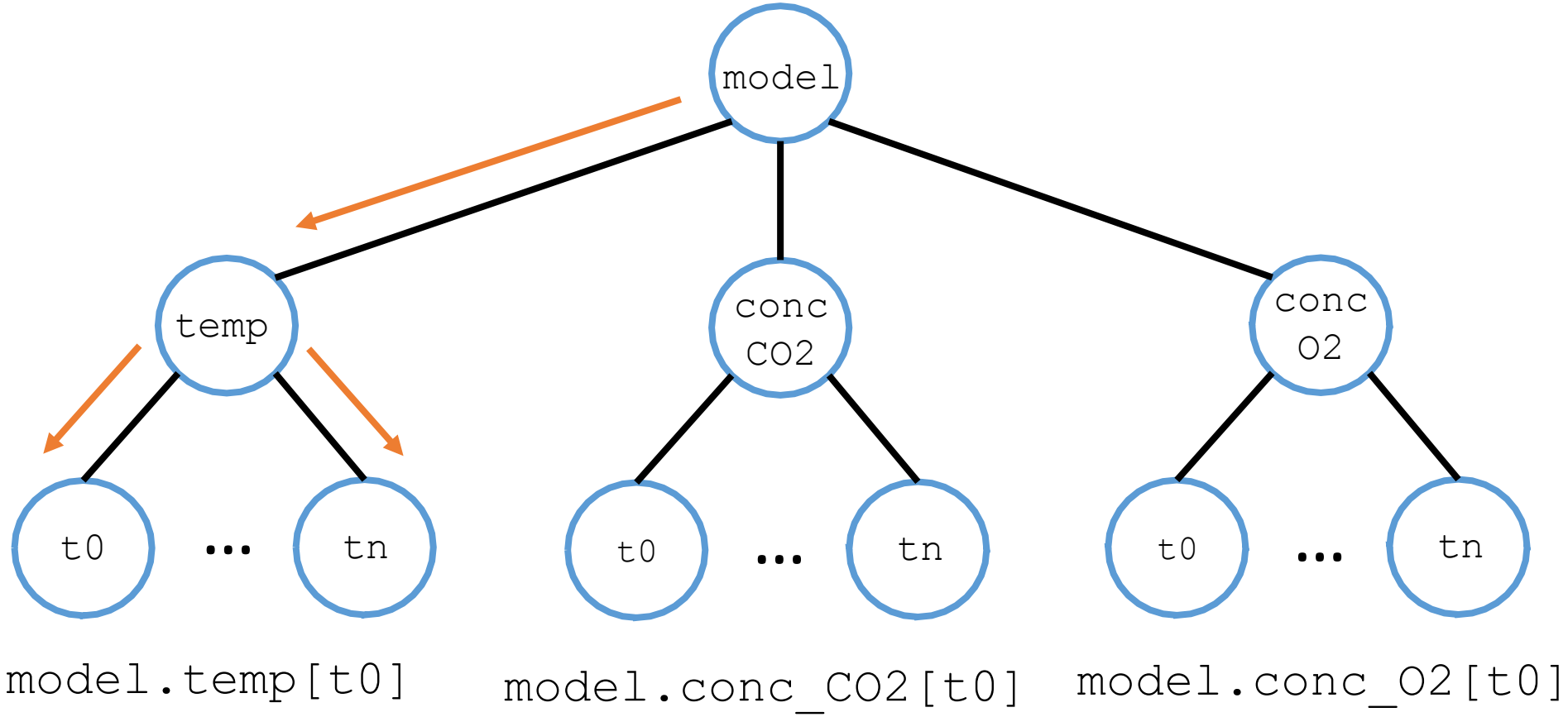
- Identify time-indexed variables
- Identify time-indexed blocks
- Process each indexing set

To identify the same variable/constraint at different time points

- Descend into time-indexed blocks
- Identify variables
- Ascend back to time-indexed block
- Re-descend at different time point
- Process each indexing set



A flattened structure makes accessing time-indices easier



- Don’t have to descend/ascend into/out of blocks
- Don’t have to process indexing sets

`flatten_dae_components` reshapes a model into this structure

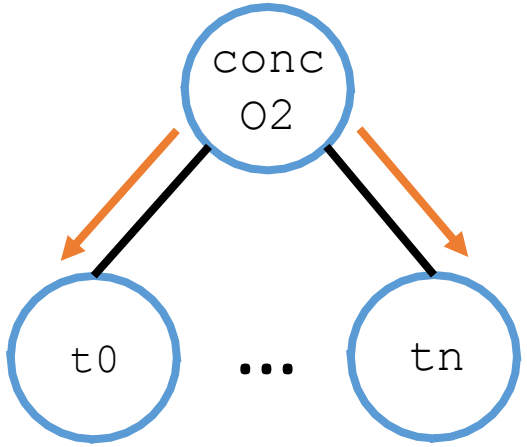
```
scalar_vars, dae_vars = flatten_dae_components(model, time, Var)
```

`scalar_vars`

- All non-time-indexed variables in model

`dae_vars`

- All time-indexed variables in model
- Indexed only by time



We have:

- Partitioned variables into time-indexed and non-time-indexed
- Partitioned time-indexed variables into only-time-indexed variables

Now easy to:

- Identify all variables at a particular point in time
- Identify the same variable at different points in time

A model partitioned by time allows targeted debugging

`flatten_dae_components` allows identification of the subsystem at each time point

`DerivativeVar` and discretization equations allow identification of differential and algebraic subsystems at each point in time

In a well-posed DAE:

- The independent subsystem at any point in time should be nonsingular

$$F(x, y, \dot{x}) = 0 \qquad \begin{bmatrix} \nabla_y F & \nabla_{\dot{x}} F \end{bmatrix}$$

In an index-1 DAE:

- The algebraic subsystem at any point in time should be nonsingular

$$\begin{aligned} \dot{x} &= f(x, y) \\ 0 &= g(x, y) \end{aligned} \qquad \begin{bmatrix} I & \nabla_y f \\ & \nabla_y g \end{bmatrix}$$

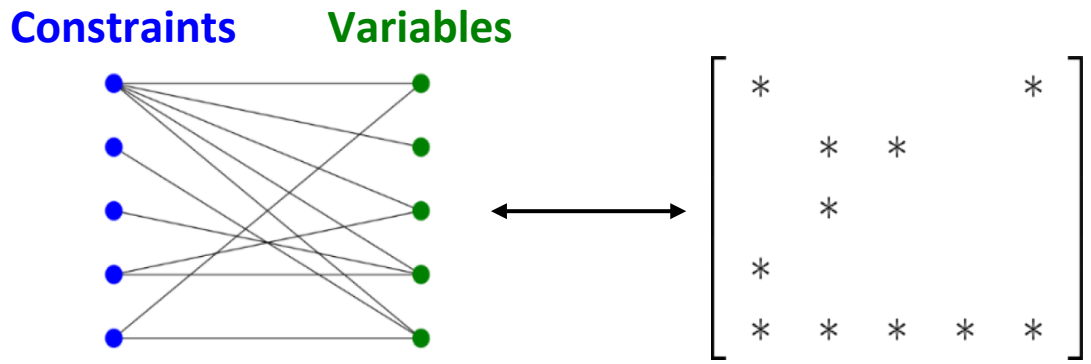
Debugging smaller systems is easier than debugging the entire model

We have implemented graph-based tools for debugging singular systems

Model is a bipartite graph of variables and constraints

Bipartite graph has an associated incidence matrix

- Rows = constraints
- Columns = variables



We implement the following algorithms

1. Maximum matching

2. Dulmage-Mendelsohn Partition

3. Block triangularization

NetworkX Subroutines

i. Maximum matching

i. Maximum matching
ii. Breadth-first search

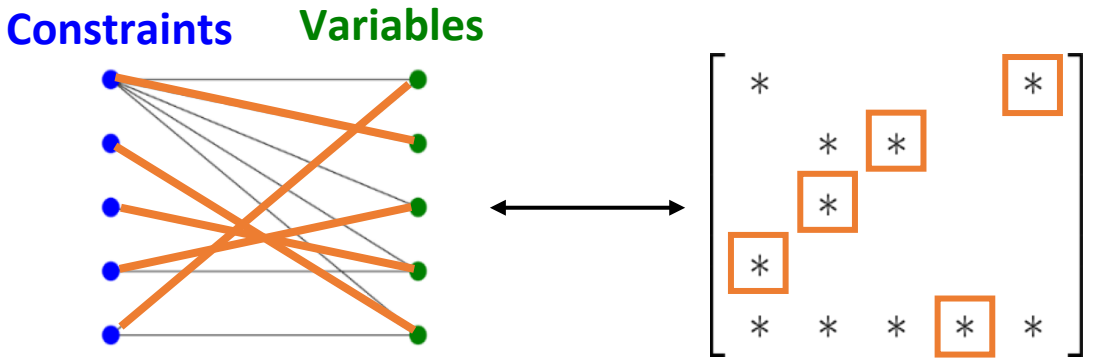
i. Maximum matching
ii. Strongly connected components
iii. Topological sort

NetworkX
Network Analysis in Python

A maximum matching identifies structural singularity

A **maximum matching** in a bipartite graph:

- Is the largest set of disjoint pairs of adjacent nodes
- Corresponds to the maximum number of matrix entries on the diagonal
- Is a **perfect matching** if every node is matched

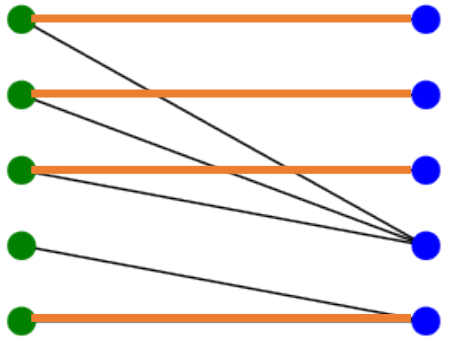


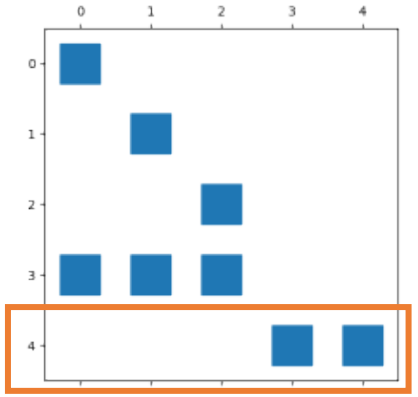
If a system does not have a perfect matching, its Jacobian is guaranteed to be singular

Example: system of 5 variables and 5 equations

- Variables: x_1, x_2, x_3, v, F
- Parameters: $\rho_1, \rho_2, \rho_3, \rho$

$$x_1\rho = \rho_1$$
$$x_2\rho = \rho_2$$
$$x_3\rho = \rho_3$$
$$\sum_i x_i = 1$$
$$v\rho = F$$





Size of maximal matching = 4

The Dulmage-Mendelsohn partition identifies square, underconstrained, and overconstrained subsystems

The Dulmage-Mendelsohn partition:

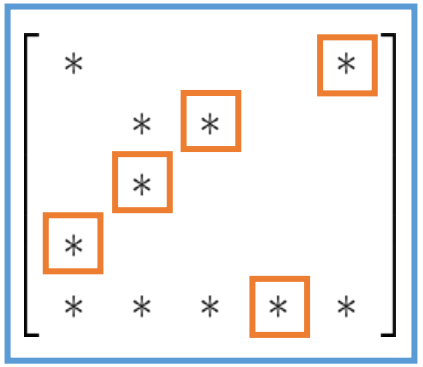
- Tells us which nodes can possibly be unmatched

Unmatched variables = underconstrained subsystem

Unmatched constraints = overconstrained subsystem

Perfect matching = square subsystem

Square subsystem is the entire system



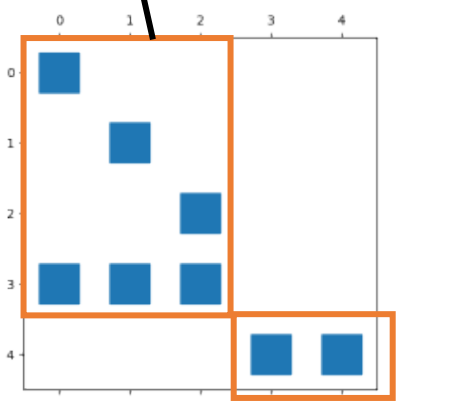
Overconstrained

$$x_1\rho = \rho_1$$
$$x_2\rho = \rho_2$$
$$x_3\rho = \rho_3$$
$$\sum_i x_i = 1$$

Underconstrained

$$v\rho = F$$

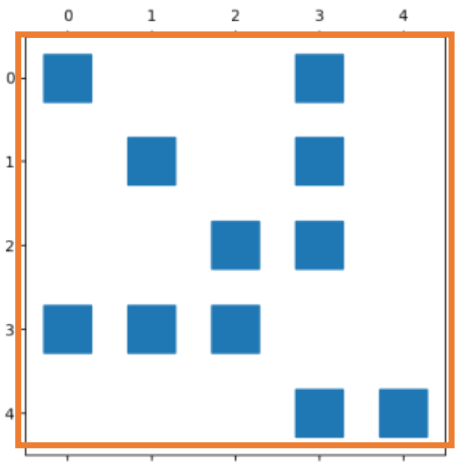
Overconstrained



Underconstrained

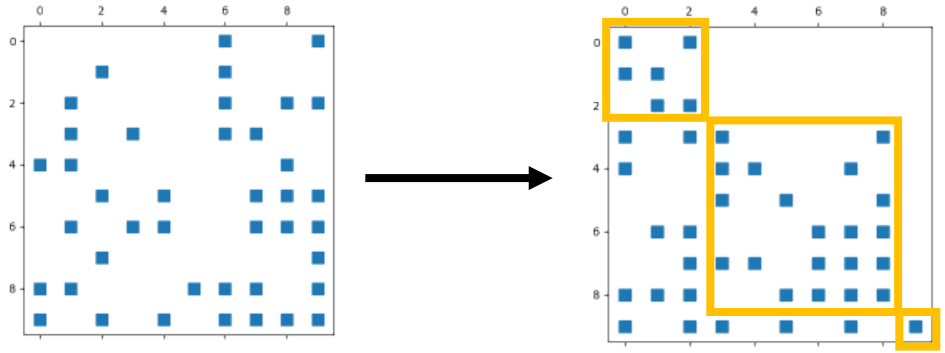
Fix: Make v or F a parameter instead of ρ

Square



Block triangularization decomposes a system into subsystems that can be solved independently

- Performed by identifying strongly connected components
- Narrows down the source of numeric singularity or poor conditioning
- Only possible when the system already has a perfect matching



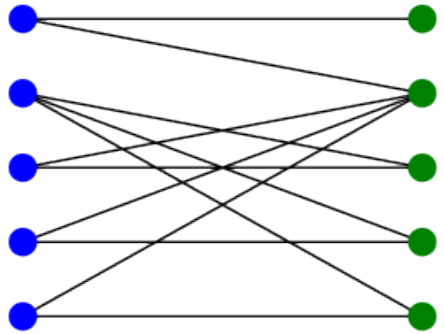
Factorize/solve each block independently

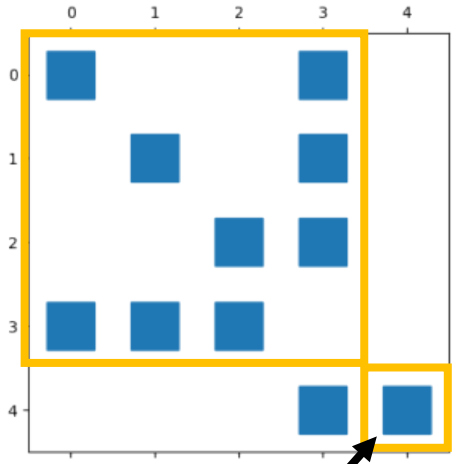
Example: system of 5 variables and 5 equations

- Variables: x_1, x_2, x_3, F, ρ
- Parameters: $\rho_1, \rho_2, \rho_3, v$

Fix this system: make v a parameter instead of ρ

$$x_1\rho = \rho_1$$
$$x_2\rho = \rho_2$$
$$x_3\rho = \rho_3$$
$$\sum_i x_i = 1$$
$$v\rho = F$$

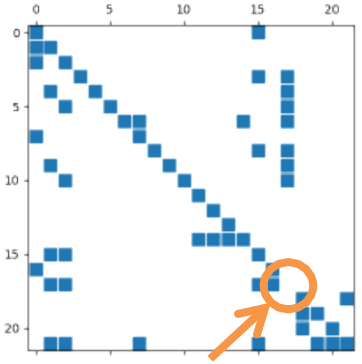




Singular as $v \to 0$

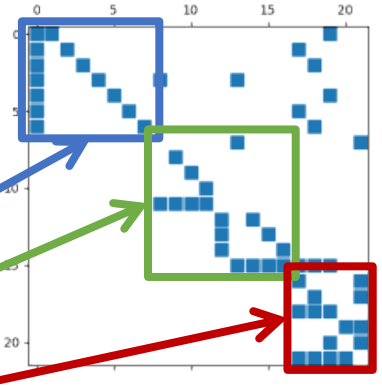
Example on a subsystem from a solid phase thermodynamic property package

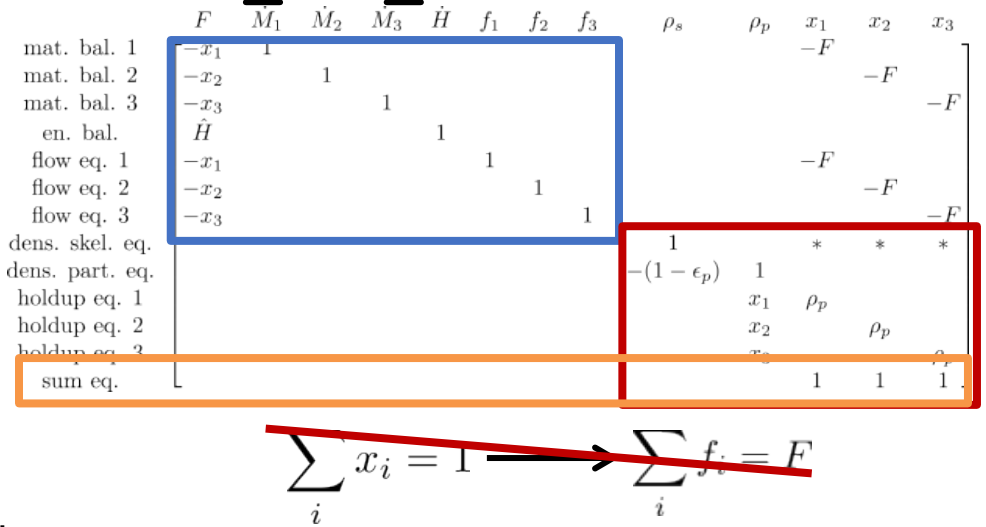
Small subsystem extracted from 60,000-variable model using `flatten_dae_components`



Missing diagonal entry

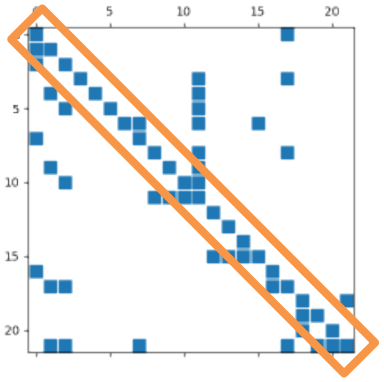
1. Make sure a maximum matching contains all constraints and variables
2. Dulmage-Mendelsohn permutation gives a useful partition:
   - Underconstrained
   - Square
   - Overconstrained



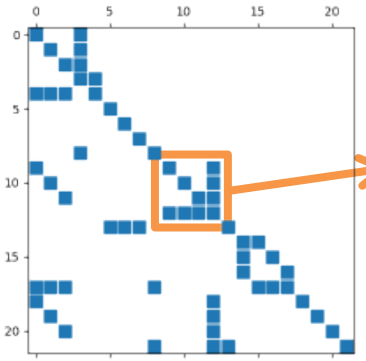


$$\sum_i x_i = 1 \quad \sum_i f_i = F$$

Complete diagonal



3. Check numerical singularity (e.g. via SVD)
4. Block triangularization helps identify the source of a numeric singularity

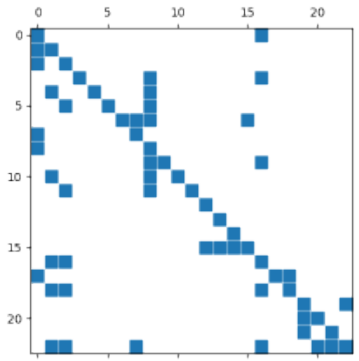


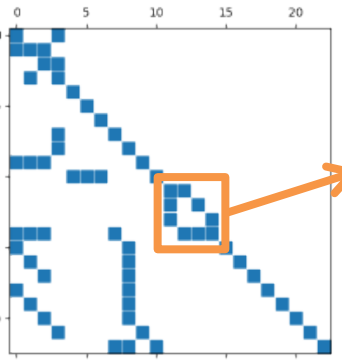
$$\begin{bmatrix} 1 & & & -x_1 \\ & 1 & & -x_2 \\ & & 1 & -x_3 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

when $x_1 + x_2 + x_3 = 1$, matrix is singular

Our fix:

- Make particle porosity a variable
- Add equation to calculate flow rate from density





$$\begin{bmatrix} F & & & x_1 \\ & F & & x_2 \\ & & F & x_3 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

We have implemented tools to decompose and debug large-scale DAE models

`flatten_dae_components` walks the model tree and processes indexing sets to return convenient data structures

A maximum matching identifies structural singularity in subsystems at each point in time

The Dulmage-Mendelsohn partition helps identify the source of structural singularity

Block triangularization helps identify the source of numeric singularity or poor conditioning

Code is available:

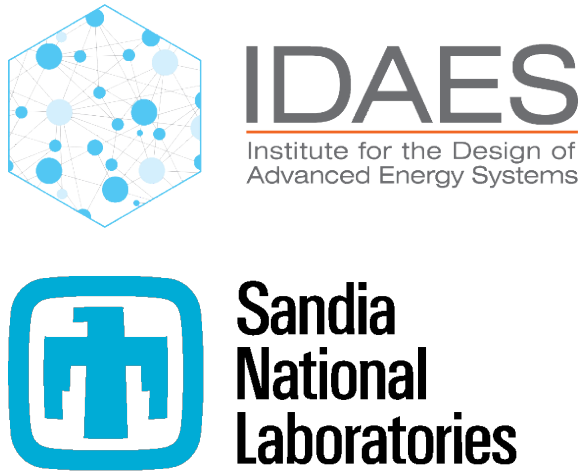
`pyomo.dae.flatten` (Pyomo v5.7)

`pyomo.contrib.incidence_analysis` (Pyomo v6.0)

Acknowledgements

Thanks to:

- Larry Biegler and all our group members
- Bethany Nicholson and John Siirola
- Sandia
- IDAES

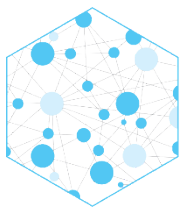


Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

CAPD CENTER

Carnegie Mellon University

Questions?



IDAES
Institute for the Design of Advanced Energy Systems