

# Transformation of Differential Algebraic Array Equations to Index One Form

Martin Otter<sup>1</sup>    Hilding Elmqvist<sup>2</sup>

<sup>1</sup>DLR Institute of System Dynamics and Control, Oberpaffenhofen, Germany

<sup>2</sup>Mogram AB, Lund, Sweden

Modelica'2017 Conference, Prague, May 15-17, 2017

## Goal: Model and Simulate Large Systems (upto $10^4..10^6$ differential equations)

- Large multi-body systems with elastic bodies and contacts
- Large electrical circuits and power electronics
- Large thermo-fluid systems

Available algorithms for Modelica tools  
reaching their limits



New algorithms are needed that move the limits!

# How to reach the Goal?

- **No scalarization of array equations**

→ more efficient code generation + simulation.

- **No transformation to ODE form**

Today: Transform ODAE (Overdetermined Differential Algebraic Equation System) from Pantelides algorithm to ODE form:  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ .

Drawbacks:

- Every evaluation of  $\mathbf{f}(\cdot)$  may require solution of algebraic equations  
→ bad for implicit integrators (which solve algebraic equations in  $\mathbf{f}(\cdot)$ ).
- Sparsity of original equations might get lost  
→ bad for large systems.

- **How to solve ODAE, without transformation to ODE form?**

→ see next slides

- Start with solving the base problems above, afterwards more need to be done.

## Differential Algebraic Array Equations

e.g.  $\mathbf{r}_2 = \mathbf{r}_1 + \mathbf{n}s$

new

Removal of  
non-structural singularities

new

Transformation algorithms on  
array equations (Pantelides, BLT)

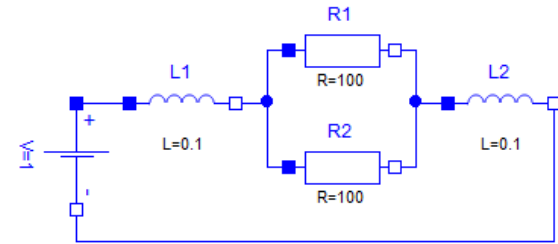
$$\begin{aligned}\mathbf{r}_2 &= \mathbf{r}_1 + \mathbf{n}s \\ \text{e.g. } \dot{\mathbf{r}}_2 &= \dot{\mathbf{r}}_1 + \dot{\mathbf{n}}\dot{s} \\ \ddot{\mathbf{r}}_2 &= \ddot{\mathbf{r}}_1 + \ddot{\mathbf{n}}\ddot{s}\end{aligned}$$

new

Transformation to special index 1 form

$$\begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) \\ \mathbf{f}_c(\mathbf{x}, t) \end{bmatrix} = \mathbf{0} \quad \begin{bmatrix} \frac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} \\ \frac{\partial \mathbf{f}_c}{\partial \dot{\mathbf{x}}} \end{bmatrix} \text{ is regular}$$

DAE integrator with sparse matrix support



- remove equation:  $-L2.n.i - V.n.i = 0$  (redundant)
- add equation :  $L2.n.v = 0$  (arbitrary value)
- replace equation:  $-R1.p.i - R2.p.i - L1.n.i = 0$   
with:  $-L1.p.i + L2.p.i = 0$   
to make constraint structural
- details: see paper
- no scalarization of equations
- no algebraic equations solved
- no dynamic state selection  
(partial static state selection)
- sparseness is kept
- BDF iteration matrix can be scaled,  
so that it is regular for  $h \rightarrow 0$
- algebraic equations only solved by  
DAE integrator (not in model, as of today)
- array equations intact (no scalarization)
- sparse matrix handling

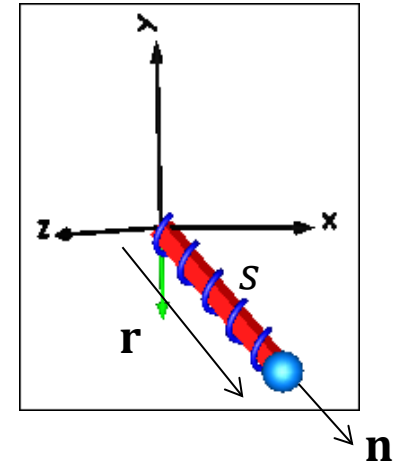
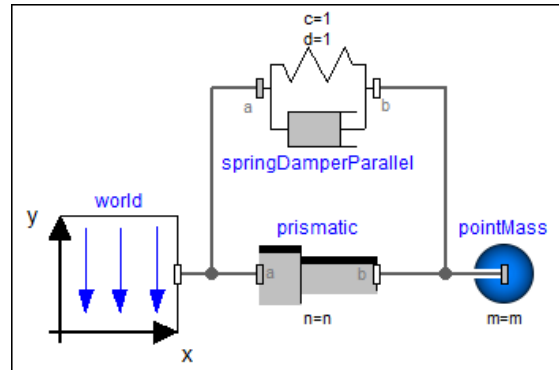
new algorithm

# Symbolic Transformation of Array Equations

# Example: Array equations of a sliding mass

parameters:  $c, d, m, \mathbf{n}, \mathbf{g}$

unknowns:  $s, \mathbf{r}, \mathbf{v}, \mathbf{f}, \mathbf{u}$



$$\mathbf{r} = \mathbf{n}s$$

$$\mathbf{v} = \dot{\mathbf{r}}$$

$$m\dot{\mathbf{v}} = \mathbf{f} + m\mathbf{g} + \mathbf{u}$$

$$0 = \mathbf{n} \cdot \mathbf{f}$$

$$\mathbf{u} = -(cs + d\dot{s})\mathbf{n}$$

Pantelides algorithm:

Determine, how often every equation must be differentiated until the highest derivative variables can be uniquely assigned to the highest derivative equations

Idea: Assign array variables to array equations

**No scalar equation** contains **scalar**  $s \rightarrow$  array equation must be scalarized

**Scalar equation** has **no scalar** unknown  $\rightarrow$  array unknown  $\mathbf{f}$  must be scalarized

$\rightarrow$  **Idea does not work**

1. (Conceptually) scalarize using **incidence of original array equations**

2. Apply **Pantelides**

*scalarized, highest  
derivative equations*      *unknowns assigned  
(incidence) variables*

...	...	...
$u_1 = -(cs + d\dot{s}) n_1$	$u_1, u_2, u_3$	$u_1$
$u_2 = -(cs + d\dot{s}) n_2$	$u_1, u_2, u_3$	$u_2$
$u_3 = -(cs + d\dot{s}) n_3$	$u_1, u_2, u_3$	$u_3$

3. **Sort** highest derivative equations (**BLT**)

→ **array elements** are in the same algebraic loop (= **BLT block**)

*BLT block*      *unknowns*

$u_1 = -(cs + d\dot{s}) n_1$ $u_2 = -(cs + d\dot{s}) n_2$ $u_3 = -(cs + d\dot{s}) n_3$	$u_1, u_2, u_3$
...	...

4. (Conceptually) transform **back to arrays**

<i>BLT block</i>	<i>solve for</i>
$\mathbf{u} = -(cs + d\dot{s})\mathbf{n}$	$\mathbf{u}$
$\ddot{\mathbf{r}} = \mathbf{n}\ddot{s}$ $\dot{\mathbf{v}} = \ddot{\mathbf{r}}$ $m\dot{\mathbf{v}} = \mathbf{f} + m\mathbf{g} + \mathbf{u}$ $0 = \mathbf{n} \cdot \mathbf{f}$	$\ddot{s}, \ddot{\mathbf{r}}, \dot{\mathbf{v}}, \mathbf{f}$

5. Analytically differentiate array equations

new algorithm

# Transformation to Special Index 1 DAE Form



# Example: Multi-Body Systems

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{v} \\ \mathbf{M}(\mathbf{q}, t)\dot{\mathbf{v}} + \mathbf{G}^T(\mathbf{q}, t)\boldsymbol{\lambda} &= \mathbf{h}(\mathbf{q}, \mathbf{v}, t) \\ \mathbf{0} &= \mathbf{g}(\mathbf{q}, t)\end{aligned}$$

$$\mathbf{G} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}}, \mathbf{M} = \mathbf{M}^T > \mathbf{0}$$

Pantelides algorithm

$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{v} \\ \mathbf{M}\dot{\mathbf{v}} + \mathbf{G}^T\boldsymbol{\lambda} &= \mathbf{h}(\mathbf{q}, \mathbf{v}, t) \\ \mathbf{0} &= \mathbf{g}(\mathbf{q}, t) \\ \mathbf{0} &= \dot{\mathbf{g}} = \mathbf{G}\dot{\mathbf{q}} + \mathbf{g}^{(1)}(\mathbf{q}, t) \\ \mathbf{0} &= \ddot{\mathbf{g}} = \mathbf{G}\ddot{\mathbf{q}} + \mathbf{g}^{(2)}(\mathbf{q}, \dot{\mathbf{q}}, t) \\ \ddot{\mathbf{q}} &= \dot{\mathbf{v}}\end{aligned}$$

$$\mathbf{g}^{(1)} = \frac{\partial \mathbf{g}}{\partial t}, \mathbf{g}^{(2)} = \dot{\mathbf{G}}\dot{\mathbf{q}} + \dot{\mathbf{g}}^{(1)}$$

Target equations

$$\begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) \\ \mathbf{f}_c(\mathbf{x}, t) \end{bmatrix} = \mathbf{0} \quad \left[ \frac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} \right] \text{ is regular}$$

$$\begin{aligned}\mathbf{0} &= \begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) \\ \mathbf{f}_c(\mathbf{x}, t) \end{bmatrix} \\ &= \begin{bmatrix} \dot{\mathbf{q}} - \mathbf{v} + \mathbf{G}^T \boldsymbol{\mu}_{int} \\ \frac{\mathbf{M}\dot{\mathbf{v}} + \mathbf{G}^T \boldsymbol{\lambda}_{int} - \mathbf{h}(\mathbf{q}, \mathbf{v}, t)}{\mathbf{g}(\mathbf{q}, t)} \\ \dot{\mathbf{g}} = \mathbf{G}\dot{\mathbf{v}} + \mathbf{g}^{(1)}(\mathbf{q}, t) \end{bmatrix}\end{aligned}$$

$$\mathbf{x} = [\mathbf{q}; \mathbf{v}; \boldsymbol{\lambda}_{int}; \boldsymbol{\mu}_{int}], \boldsymbol{\lambda} := \boldsymbol{\lambda}_{int}$$

(Gear/Gupta/Leimkuhler 1985; Gear 1988)

Can be generalized to any DAE (where Pantelides algorithm can be applied, details see paper)

# Variant of dummy derivative method → Index 1 DAE (Mattsson/Söderlind 1993)

Example:

Result of Pantelides algorithm:

BLT blocks	solve for
$\mathbf{u} = -(cs + d\dot{s})\mathbf{n}$	$\mathbf{u}$
$\ddot{\mathbf{r}} = \mathbf{n}\ddot{s}$ $\dot{\mathbf{v}} = \ddot{\mathbf{r}}$ $m\dot{\mathbf{v}} = \mathbf{f} + m\mathbf{g} + \mathbf{u}$ $\mathbf{0} = \mathbf{n} \cdot \mathbf{f}$	$\ddot{s}, \ddot{\mathbf{r}}, \dot{\mathbf{v}}, \mathbf{f}$

BLT Block 1	solve for
$\mathbf{u} = -(cs + d\dot{s})\mathbf{n}$	$\mathbf{u}$
BLT Block 2	
BLT Block 2.1	
$\mathbf{r} = \mathbf{n}s$	$s, \mathbf{r}$
BLT Block 2.2	
$\dot{\mathbf{r}} = \mathbf{n}\dot{s}$ $\mathbf{v} = \dot{\mathbf{r}}$	$\dot{s}, \dot{\mathbf{r}}, \mathbf{v}$
BLT Block 2.3	
$\ddot{\mathbf{r}} = \mathbf{n}\ddot{s}$ $\dot{\mathbf{v}} = \ddot{\mathbf{r}}$ $m\dot{\mathbf{v}} = \mathbf{f} + m\mathbf{g} + \mathbf{u}$ $\mathbf{0} = \mathbf{n} \cdot \mathbf{f}$	$\ddot{s}, \ddot{\mathbf{r}}, \dot{\mathbf{v}}, \mathbf{f}$

$$\begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) \\ \mathbf{f}_c(\mathbf{x}, t) \end{bmatrix}$$

$$\mathbf{x} = [s; \dot{s}]$$

$$\mathbf{r} := \mathbf{n}s$$

$$\dot{\mathbf{r}} := \mathbf{n}\dot{s}$$

$$\mathbf{v} := \dot{\mathbf{r}}$$

$$\ddot{\mathbf{r}} := \mathbf{n}\ddot{s}$$

$$\dot{\mathbf{v}} := \ddot{\mathbf{r}}$$

$$\mathbf{u} := -(cs + d\dot{s})\mathbf{n}$$

$$\mathbf{f} := m\dot{\mathbf{v}} - m\mathbf{g} - \mathbf{u}$$

With **tearing** on every BLT block, constraint equations are explicitly solved (here) and are local equations:

new algorithm

# Tearing

# Tearing with retained solution space

(Elmqvist/Otter 1999 (unpublished), Bender/Fineman/Gilbert/Tarjan 2016)

$$\mathbf{0} = \mathbf{g}(\mathbf{z}) \xrightarrow{\text{solve explicitly}}$$

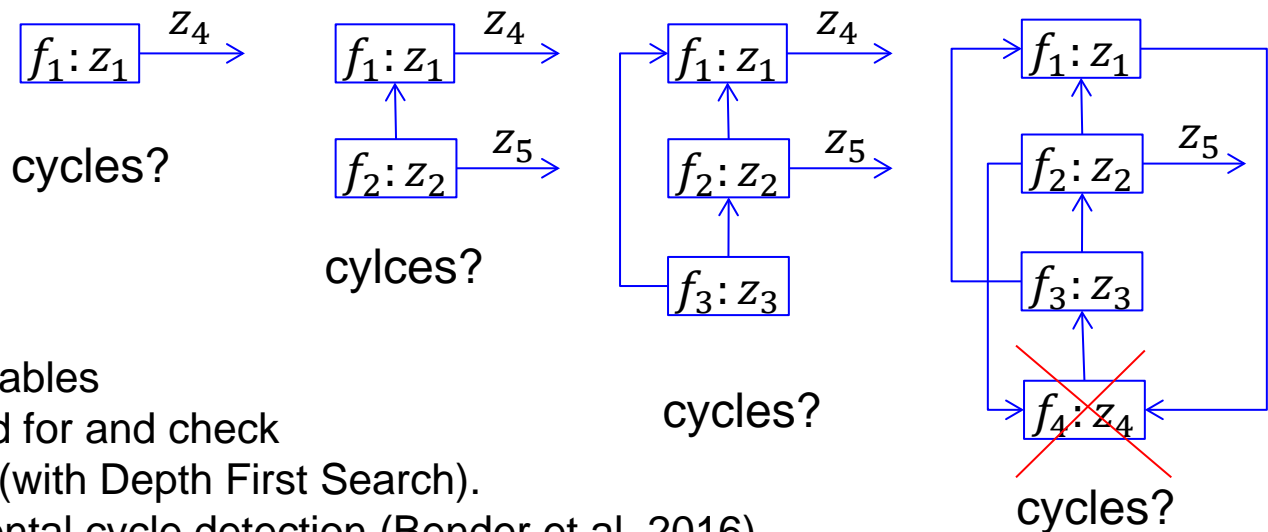
$$\begin{aligned} \mathbf{z}_e &:= \mathbf{g}_e(\mathbf{z}_e, \mathbf{z}_t) \\ \mathbf{0} &= \mathbf{g}_r(\mathbf{z}_e, \mathbf{z}_t) \end{aligned}$$

Observation:

$(z_{e,i}, g_{e,i})$  nodes of a Directed Acyclic Graph, so **no cycles**

Example:

$$\begin{aligned} z_1 &= f_1(z_4) \\ z_2 &= f_2(z_1, z_5) \\ z_3 &= f_3(z_2, z_1) \\ z_4 &= f_4(z_3, z_2) \end{aligned}$$



Try all combinations of variables that can be explicitly solved for and check whether a cycle is present (with Depth First Search).

Faster search with incremental cycle detection (Bender et al. 2016)

## Conclusions and Future Work

- Several new algorithms developed to improve symbolic processing of Modelica tools (should allow to support larger Modelica models)
- Test implementation and evaluation of the algorithms in Modia  
(= domain specific extension of the Julia programming language;  
see companion paper „Innovations for future Modelica“).
- Tests/evaluation with large, difficult models not yet done.  
Will be performed in the near future.
- Initialization of index 1 DAEs not discussed:  
Could be performed with all equations from Pantelides algorithm (as today).  
Evaluation of a new method to handle Dirac impulses in any DAE:  
 $\mathbf{x}(t_0 - \epsilon) \rightarrow$  changes discontinuously to  $\mathbf{x}(t_0 + \epsilon) \rightarrow$  Dirac impulses in  $\dot{\mathbf{x}}(t_0)$
- Modia, including the implementations of the algorithms, to become available from <https://github.com/modiasim> under MIT license.

