Transformation of Differential Algebraic Array Equations to Index One Form

Martin Otter[1] Hilding Elmqvist[2]

[1]Institute of System Dynamics and Control, DLR, Germany, Martin.Otter@dlr.de
[2]Mogram AB, Sweden, Hilding.Elmqvist@Mogram.net

Abstract

Several new algorithms are proposed that in effect transform DAEs (Differential Algebraic Equations) to a special index one form that can be simulated with standard DAE integrators. The transformation to this form is performed without solving linear and/or nonlinear equation systems, the sparsity of the equations is kept, and array equations remain array equations or differentiated versions of them. Furthermore, certain DAEs can be handled where structural index reduction methods fail. It is expected that these new algorithms will help to treat large Modelica models of any index in a better way as it is currently possible. The algorithms have been evaluated and tested in the experimental simulation environment Modia that is implemented with the Julia programming language.

Keywords: Modelica, Modia, Julia, DAE, sparse DAE, large DAE, Pantelides algorithm, Dummy Derivative Method.

1 Introduction

The objective is to handle larger Modelica models as it is practically possible today. For this purpose new algorithms have been developed: equations as well as variables are not scalarized but keep their original array types, even if differentiated. This gives a more compact code which is a benefit with regards to code cache behavior. Furthermore, there is a possibility to utilize vector instructions of modern processors. With respect to current Modelica tools, equation systems are not solved locally in the model code but by a DAE solver where the sparsity of the Jacobian is taken into account and in the model code single (array) equations are either explicitly solved if this is possible or residues for implicit equations are computed to be solved by the DAE solver. The new algorithms have been evaluated and tested in the prototype Modia (*Elmqvist et al., 2016, Elmqvist et al., 2017*) which is implemented with the Julia programming language[1] (*Bezanson et al., 2017*) and takes advantage of this very promising language effort with focus on scientific computing. Modia is available from https://github.com/ModiaSim.

[1] http://julialang.org/

2 Special Index One DAE Form

The goal is to simulate physical models that are described by a modeling language such as Modelica and mapped to a DAE in the form

$$\mathbf{f}_0(\dot{\mathbf{x}}_{d0}, \mathbf{x}_{d0}, \mathbf{x}_{a0}, t) = \mathbf{0} \tag{1}$$
$$\mathbf{f}_0 \in \mathbb{R}^{nd_0} \times \mathbb{R}^{nd_0} \times \mathbb{R}^{na_0} \times \mathbb{R} \rightarrow \mathbb{R}^{nd_0+na_0}$$

where $\mathbf{x}_{d0}(t)$ are variables that appear differentiated and $\mathbf{x}_{a0}(t)$ are variables that do not appear differentiated. Furthermore, it is assumed that a unique solution of this DAE exists if consistent initial conditions of $\dot{\mathbf{x}}_{d0}, \mathbf{x}_{d0}, \mathbf{x}_{a0}$ are given, and that the equations and variables are smoothly differentiable sufficiently often. Typically, Modelica tools transform (1) into ODE (Ordinary Differential Equation) form and use ODE or DAE integration methods for the solution. In this paper, this is not done because (a) a transformation to ODE form may destroy the sparsity structure of the equations and (b) requires in general solving linear and/or nonlinear algebraic equation systems and an implicit integration method will in turn solve nonlinear algebraic equation systems as well (so a nonlinear solver is called within a nonlinear solver). For certain classes of physical models, such as large 3-dimensional mechanical systems, this approach might not be efficient and not reliable. Instead a new approach is proposed to transform (1) to the following *special index one DAE*

$$\begin{aligned}\mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) &= \mathbf{0} \\ \mathbf{f}_c(\mathbf{x}, t) &= \mathbf{0}\end{aligned} \quad (2a) \qquad \mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} \\ \dfrac{\partial \mathbf{f}_c}{\partial \mathbf{x}} \end{bmatrix} \text{ is regular} \quad (2b)$$

without solving equation systems. DAE (2) shall have an identical solution space as DAE (1) and $\mathbf{x}_{d0}, \mathbf{x}_{a0}$ shall be part of $\mathbf{x}$. Note, when differentiating $\mathbf{f}_c(..)$,

$$\begin{aligned}\mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) &= \mathbf{0} \\ \frac{\partial \mathbf{f}_c}{\partial \mathbf{x}}\dot{\mathbf{x}} + \frac{\partial \mathbf{f}_c}{\partial t} &= \mathbf{0}\end{aligned} \tag{3}$$

can be solved for $\dot{\mathbf{x}}$ because the matrix of partial derivatives of (3) with respect to $\dot{\mathbf{x}}$ is the Jacobian $\mathbf{J}$ of (2b) which is regular. This shows that (2) has index 1.

A number of methods exist for solving system (2) numerically. In particular, under mild conditions BDF

(Backward Differentiation Formula) methods with order k ($k < 7$) and fixed or variable step-size h converge with $O(h^k)$, see (*Brenan et al. 1996*) page 51-54. This means that a DAE integrator like Sundials IDA (*Hindmarsh et al. 2005*) can solve such systems.

On the other hand, solving (2) with a BDF-method requires to solve a nonlinear equation system where the inverse of the iteration matrix becomes singular for $h \to 0$. (*Petzold and Lötstedt, 1986*) point out that step size selection is difficult if reducing the step size makes the iteration matrix ill-conditioned and it is proposed to scale elements of the iteration matrix with h so that this effect does not occur. In (*Arnold, 2016*) it is shown how this technique can be applied for multibody systems. For system (2) scaling with h is particularly simple, resulting in two possible ways:

$$\begin{aligned} h\mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) &= \mathbf{0} \\ \mathbf{f}_c(\mathbf{x}, t) &= \mathbf{0} \end{aligned} \quad (4a) \qquad \begin{aligned} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) &= \mathbf{0} \\ \frac{1}{h}\mathbf{f}_c(\mathbf{x}, t) &= \mathbf{0} \end{aligned} \quad (4b)$$

Assume that these systems are solved with a BDF method of order k. This means that the derivatives $\dot{\mathbf{x}}$ at step i are approximated as:

$$\dot{\mathbf{x}}_i \approx \frac{\alpha_{k0}}{h}\mathbf{x}_i + \frac{1}{h}\sum_{j=1}^{j=k} \alpha_{kj}\,\mathbf{x}_{i-j} \qquad (5)$$

where α_{kj} are constant coefficients depending on the order of the method, $h = t_i - t_{i-1}$ is the step size and the sum $\sum \alpha_{kj}\mathbf{x}_{i-j}$ is a known term computed from values of $\mathbf{x}$ at previous time instants. Inserting (5) in (4a) results in a nonlinear system of equations for $\mathbf{x}_i$:

$$\begin{aligned} h\mathbf{f}_d\left(\frac{\alpha_{k0}}{h}\mathbf{x}_i, \mathbf{x}_i, t_i\right) &= \mathbf{0} \\ \mathbf{f}_c(\mathbf{x}_i, t_i) &= \mathbf{0} \end{aligned} \qquad (6)$$

Assume that $\mathbf{x}, \mathbf{f}_d, \mathbf{f}_c$ are sufficiently smooth and bounded and that $\mathbf{x}_{i-1}$ solves (6) at the previous time instant t_{i-1}. According to the implicit function theorem, (6) has a *unique solution* at time instant $t_{i-1} + h$ if the inverse of the Jacobian of this system

$$\mathbf{J}_i = \begin{bmatrix} \dfrac{\partial h\mathbf{f}_d}{\partial \mathbf{x}_i} \\ \dfrac{\partial \mathbf{f}_c}{\partial \mathbf{x}_i} \end{bmatrix} = \begin{bmatrix} \alpha_{k0}\dfrac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} + h\dfrac{\partial \mathbf{f}_d}{\partial \mathbf{x}_i} \\ \dfrac{\partial \mathbf{f}_c}{\partial \mathbf{x}_i} \end{bmatrix} \qquad (7)$$

exists for small h. This is indeed the case, since for $h \to 0$ the Jacobian (7) converges to the regular Jacobian (2b), when dividing the upper equation by the constant α_{k0}. A similar result can be derived for (4b).

To summarize, DAE (2) can be solved by standard (index one) DAE integrators and a reliable numerical solution with a BDF method can be expected even for small step sizes when one of the h scaling methods of (4) is used. In the remaining part of this paper it is shown, how a large class of DAEs in the form (1) can be transformed to (2). This transformation is performed in several steps that are discussed now in sequence.

3 Index Reduction of DAEs that have Array Equations

3.1 Algorithms for Index Reduction

In order to reduce a DAE (1) to ODE or index one form, equations of (1) might need to be differentiated. There are in principle many algorithms to perform this index reduction based on the *structure of the equations*, that is by the information which variable is present in which equation. The essential idea and the key algorithm are from (*Pantelides, 1988*): The structure of the equations is described by a bipartite graph of equations and variables. The equations are differentiated until a *complete assignment* of the highest derivative equations is possible with respect to the highest derivative variables (which include algebraic variables that are not differentiated).

Since the goal is to achieve complete assignment in a bipartite graph, *any* matching algorithm for a bipartite graph can be used as basis. In (*Pantelides, 1988*), the matching algorithm of (*Duff, 1981*) is utilized which results in a very simple and elegant implementation. It results in a worst time complexity of $O(n \cdot m)$ where n is the number of equations in the final system (= original and all differentiated equations) and m is the number of entries (incidences) in the final bipartite graph.

In (*Duff et al., 2011*) eight matching algorithms and various additional heuristics are compared. Most of them have the same worst time complexity as (*Duff, 1981*), a few have $O(\sqrt{n} \cdot m)$. On average the $O(n \cdot m)$ algorithm PF+ described in this article had the best performance on the test matrices. In (*Frenkel et al., 2012*) nine matching algorithms and different implementations for structural index reductions are compared for multibody examples with varying number of bodies. This evaluation indicates that PF+ has on average the best performance for index reduction with the Pantelides algorithm.

It is well-known that complete matching in a bipartite graph is equivalent to the network flow problem where the maximum amount of flow shall be determined that can be sent between two given vertices of a graph, see for example (*Skiena, 2008, page 217*). It is also well-known that both the network flow problem and the bipartite matching problem can be formulated as a special linear programming problem, see for example (*Edmond,1965; Cook and Rohe, 1999; Skiena, 2008, pp. 509-510*). For all these problem classes solution algorithms are available and can be used for index reduction. For example, (*Pryce, 2001*) describes an index reduction method based on the special linear programming problem.

All above algorithms for index reduction are iterative and the question is when the iteration stops. (*Pantelides, 1988*) provides an elegant method to test beforehand whether the (structural) index is finite:

Adding the relationship $0 = h_i(\dot{x}_{d0,i}, x_{d0,i})$ structurally for all differentiated variables in (1) gives an *extended system* which can be interpreted as solving (1) with an implicit integration method. If this system has a *complete matching*, the (structural) index is finite and structural index reduction algorithms will converge.

All the algorithms mentioned above have the disadvantage that only structural properties are utilized and therefore will fail if state constraints are not structurally visible. In (*Chowdhry et al., 2004*) a symbolic/numeric index reduction procedure is proposed. It is based on a symbolic numerical algebra for pattern manipulations of the DAE (1) and reduces the index of linear constant coefficient DAEs *numerically* by LU decompositions and the index of nonlinear parts by a *structural* index reduction algorithm.

In section 5 a new method is proposed to exactly handle singularities in the connection graph of a model, both to treat (consistently) underdetermined and overdetermined equation systems, as well as state constraints that cannot be handled by a structural index reduction algorithm. This technique transforms a DAE (1) in a DAE (1) and is therefore a pre-processing step for the transformations of section 3 and 4.

3.2 Index Reduction on Array Equations

The Pantelides algorithm and other structural index reduction algorithms are designed for *scalar* variables and equations. So Modelica tools typically symbolically expand array equations into a set of scalar equations involving the array elements and the description with array equations is lost. This has significant drawbacks for large array equations. In (*Schuchart, et al., 2015*) it is shown how special for-loops can be handled so that they need not to be expanded for the Pantelides algorithm and are retained in the generated code.

Below, a new technique is proposed to handle any kind of array equations. Hereby the (conceptual) expansion of array equations is performed *only* in the bipartite graph to perform structural index reduction and in a BLT (Block Lower Triangular) transformation on the highest derivative equations. In order to illustrate this technique, a tiny multibody example will be used.

Consider the following model of a *sliding mass*. It is a one degree-of-freedom model, with scalar parameters c,d,m, vector parameters $\mathbf{n}, \mathbf{g}$, scalar unknown s and vector unknowns $\mathbf{r}, \mathbf{v}, \mathbf{f}, \mathbf{u},$ that is described by the following equations:

$$\begin{aligned} \mathbf{r} &= \mathbf{n}s \\ \mathbf{v} &= \dot{\mathbf{r}} \\ m\dot{\mathbf{v}} &= \mathbf{f} + m\mathbf{g} + \mathbf{u} \\ 0 &= \mathbf{n}\cdot\mathbf{f} \\ \mathbf{u} &= -(cs + d\dot{s})\mathbf{n} \end{aligned} \tag{8}$$

(8) is a DAE (1) with $4 \cdot 3 + 1 = 13$ equations in the 13 variables $\mathbf{x}_{d0} = [s; \mathbf{r}; \mathbf{v}], \mathbf{x}_{a0} = [\mathbf{f}; \mathbf{u}]$.

With the Pantelides algorithm it is determined how often every equation would have to be differentiated until the highest derivatives variables can be uniquely assigned to the highest derivative equations. Since we want to keep array equations intact, it is natural to assign array variables to array equations, provided they have the same type and the same dimensions. See also (*Stavåker, 2015)* chapter 9. However, this does not work for the sliding mass example above and any other multibody system where bodies are connected by joints.

The scalar variable s appears only in vector equations, so s or a higher derivative of it can only be assigned to an element of these equation (or a derivative of them), which means that a vector equation must be expanded in scalar equations. Furthermore, the vector variable $\mathbf{f}$ appears as only variable in a scalar equation ($0 = \mathbf{n}\cdot\mathbf{f}$) and therefore one element of $\mathbf{f}$ must be assigned to this scalar equation. As a result, the two other elements of $\mathbf{f}$ have to be assigned in other equations, which then must be expanded as well. In the end, all equations must be expanded to scalar equations in order that an assignment of all variables is possible.

After expanding all equation graphs, the Pantelides algorithm determines that the first vector equations must be differentiated twice and the second one time leading to the following assignments:

assigned	*highest derivative equations*	*diff. order*
$\ddot{s}$	$\ddot{r}_1 = n_1\ddot{s}$	2
$\ddot{r}_2$	$\ddot{r}_2 = n_2\ddot{s}$	2
$\ddot{r}_3$	$\ddot{r}_3 = n_3\ddot{s}$	2
$\ddot{r}_1$	$\dot{v}_1 = \ddot{r}_1$	1
$\dot{v}_2$	$\dot{v}_2 = \ddot{r}_2$	1
$\dot{v}_3$	$\dot{v}_3 = \ddot{r}_3$	1
$\dot{v}_1$	$m\dot{v}_1 = f_1 + mg_1 + u_1$	0
f_2	$m\dot{v}_2 = f_2 + mg_2 + u_2$	0
f_3	$m\dot{v}_3 = f_3 + mg_3 + u_3$	0
f_1	$0 = n_1f_1 + n_2f_2 + n_3f_3$	0
u_1	$u_1 = -(cs + d\dot{s})\,n_1$	0
u_2	$u_2 = -(cs + d\dot{s})\,n_2$	0
u_3	$u_3 = -(cs + d\dot{s})\,n_3$	0

As will become clear in section 4, the set of highest derivative equations must be sorted, so a BLT (Block Lower Triangular) transformation must be applied that identifies the order of evaluation, as well as the algebraic loops under the assumption that the lower-order derivative variables are known. Furthermore, the assumption is used that array equations have full incidence (see Assumption 1 below).

highest derivative equations	*solve for*
$u_1 = -(cs + d\dot{s})\, n_1$ $u_2 = -(cs + d\dot{s})\, n_2$ $u_3 = -(cs + d\dot{s})\, n_3$	u_1, u_2, u_3
$\ddot{r}_1 = n_1\ddot{s}$ $\ddot{r}_2 = n_2\ddot{s}$ $\ddot{r}_3 = n_3\ddot{s}$ $\dot{v}_1 = \ddot{r}_1$ $\dot{v}_2 = \ddot{r}_2$ $\dot{v}_3 = \ddot{r}_3$ $m\dot{v}_1 = f_1 + mg_1 + u_1$ $m\dot{v}_2 = f_2 + mg_2 + u_2$ $m\dot{v}_3 = f_3 + mg_3 + u_3$ $0 = n_1 f_1 + n_2 f_2 + n_3 f_3$	$\ddot{s}, \ddot{r}_1, \ddot{r}_2, \ddot{r}_3,$ $\dot{v}_1, \dot{v}_2, \dot{v}_3,$ f_1, f_2, f_3

The result is that there is an algebraic system with 3 equations that has to be solved for 3 unknowns. Afterwards an algebraic equation system with 10 equations has to be solved for 10 unknowns. Note, whenever an algebraic loop is encountered, the previous assignment information *on equation level* is no longer relevant (which is anyway not unique in such a case) but only the *set of unknown variables* of the respective algebraic loop.

Although the Pantelides algorithm must be performed on (conceptually) expanded scalar equations and scalar variables, *the BLT transformed highest derivative equations* can be contracted to array equations again:

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

The original Pantelides algorithm and the BLT transformation are graph based algorithms that assume nodes and vertices correspond to scalar real variables and equations. These algorithms can be generalized to work directly on the array variables and equations.

3.3 Properties of Array Equations

The presented approach relies on the fact that all scalarized equations/variables appear in the same algebraic equation system (or more precisely in the same strongly connected component of a directed graph). This sub-section contains the assumption under which this property holds and the proof of the property. Multi-dimensional arrays are treated as vectors with length being the total number of elements.

Assumption 1: *When expanding the bipartite graph of DAE (1) regarding array variables and array equations, it is assumed that they have full incidence.*

Example: For the equation $\mathbf{e}$: $\mathbf{w} = \mathbf{f}(\mathbf{u}, \mathbf{v})$ with all variables being vectors of length 2, the incidence structure is assumed to be:

	w_1	w_2	u_1	u_2	v_1	v_2
e_1	x	x	x	x	x	x
e_2	x	x	x	x	x	x

Theorem 1: *If one element of an array equation needs to be differentiated, all elements of* _all time varying variables_ *in the equation need to be differentiated.*

Example: For the equation $\mathbf{e}$: $\mathbf{w} = \mathbf{f}(\mathbf{v})$ with all variables being vectors of length 2, the incidence structure of the differentiated equation is:

	w_1	w_2	$\dot{w}_1$	$\dot{w}_2$	v_1	v_2	$\dot{v}_1$	$\dot{v}_2$
$\dot{e}_1$	x	x	x	x	x	x	x	x
$\dot{e}_2$	x	x	x	x	x	x	x	x

Proof: This follows since if an array variable has full incidence, so has its time derivative. This means that derivatives of all time varying array variable elements will appear in the differentiated element of the array equation. ■

This is consistent with the Pantelides algorithm because, if the function augmentPath returns false, all variable elements with incidence are marked as colored. All colored V-nodes to be differentiated are then marked in the A vector.

Theorem 2: *If one element of an array equation needs to be differentiated, all other elements of the* _equation_ *need to be differentiated.*

Proof: According to Theorem 3, all elements of an array equation will appear in the same strongly connected component. It means that there is a mutual dependency between all array equation elements. This means that there is also a mutual dependency between all differentiated array variable elements. In order to be able to solve for all derivatives, an equal number of array equation elements are needed, that is all array equation elements must be differentiated. ■

Function augmentPath colors all equations visited. If assignment is not possible, augmentPath tries to reassign. Due to the mutual dependency, all of the elements of an array equation are visited. In the B-vector of the Pantelides algorithm it is marked that all colored equation nodes should be differentiated.

Theorem 3: *If the highest derivative equations are structurally non-singular with respect to the highest derivative variables, all elements of an array equation will appear in the same strongly connected component.*

Proof: The incidence matrix of an array equation consists of n identical rows with n being the number of scalar elements of the left and right hand side. Assume that these elements of the array equation (incidence matrix rows) appear in different strongly connected components. This would mean that some of these elements of the array equation could be solved without the others. Since they have the same incidence, it would mean that the other elements of the array equation would be structurally over-constrained. This contradicts the assumption of structural non-singularity. ■

Limitations

It is worth noting that there are cases when symbolic expansion of array equations is beneficial. For example, (*Elmqvist and Mattsson, 2016)* discusses detection and handling of planar loops of multibody systems. In such cases, certain elements of position vectors are overdetermined and certain elements of force vectors are underdetermined. In order to reveal this, the zeros in axis of rotation and translation vectors must be utilized, that is, certain equations must be expanded.

When discretized partial differential equations are handled, the boundary elements may give rise to issues with Assumption 1. In such a case, array equations for the inner elements might be formulated and scalar equations for the boundary elements added separately.

3.4 Implementation Notes

The Pantelides and BLT algorithms have as input the incidence graph for the array variables and equations, that is, *non-expanded* equation and array structure. Additionally, a vector of lengths of each variable, that is the number of scalar elements, and a vector of the lengths of equations are given as inputs.

All for-loops over variables and equations in the original algorithms are replaced with nested for loops also looping over the lengths.

The usual indices in the assignment, lowlink and number vectors and stack are replaced by tuples denoting which array and which array element is referred to. However, due to Theorem 1, the **A** vector (see below) which tells which variables are differentiated is still just a vector over array variables. Similarly, due to Theorem 2, the **B** vector (see below) for differentiated equations does not need the tuple indexing. The strongly connected component representation is only referring to array equations due to Theorem 3.

3.5 Result of Structural Index Reduction and BLT

For the further processing, the result of the structural index reduction and BLT transformation of array equations is summarized formally. For this, the following notation is used

- All symbols are collected in a variable vector **v** and v_j is symbol j. A symbol may represent a scalar or an array. $V_{length,j}$ gives the number of elements of symbol j.
- All equations are collected in an equation vector **e** and e_i is equation i. An equation may be a scalar or an array equation. $E_{length,i}$ gives the number of elements of equation i.
- The relationship between the symbols is defined by the variable association vector **A**, such that:
  $A_j = \textbf{if } \dot{v}_j = v_k \textbf{ then } k \textbf{ else } 0.$
  Also the inverse relationship is needed below:
  $A_{inv,k} = \textbf{if } \dot{v}_j = v_k \textbf{ then } j \textbf{ else } 0.$
- The relationship between the equations is defined by the equation association vector **B**, such that:
  $B_i = \textbf{if } \dot{e}_i = e_k \textbf{ then } k \textbf{ else } 0.$
  Also the inverse relationship is needed below:
  $B_{inv,k} = \textbf{if } \dot{e}_i = e_k \textbf{ then } i \textbf{ else } 0.$

Starting point is DAE (1) that can be defined with $\mathbf{v}_0 = [\dot{\mathbf{x}}_{d0}; \mathbf{x}_{d0}; \mathbf{x}_{a0}]$ and the n_{e0} array equations:

$$\mathbf{e}_0(\mathbf{v}_0, t) = \mathbf{0} \tag{9}$$

Structural index reduction determines the minimal number of differentiations of (9) such that the following conditions are fulfilled by the final system:

$$e_i(v_j, t) = 0; \quad B_i = 0; \quad A_j \geq 0 \tag{10a}$$

$$e_k(v_j, t) = 0; \quad B_k > 0; \quad A_j > 0 \tag{10b}$$

$$\frac{\partial e_i}{\partial v_j} \text{ structurally regular for } A_j = 0 \tag{10c}$$

(10a) are n_{e0} array equations (the highest derivative equations) in n_{e0} unknown arrays (the highest derivative variables v_j with $A_j = 0$). The matrix (10c) of partial derivatives of the highest derivative equations with respect to the highest derivative variables is structurally regular. (10b) are $n_e - n_{e0}$ array equations describing the constraints between the variables appearing differentiated. The highest derivative variables (that is v_j with $A_j = 0$), do *not* appear in these equations.

4 Transformation to Index One Form

4.1 Overview

The transformation from (1) to index one form (2) using (10) is made in three steps: In a first step, the solution is sketched for multibody system equations. In a second step this approach is generalized and in a final step the transformation is made more efficient by partial static state selection.

In the field of multibody systems, constraints appear in nearly every model and hence multibody programs need to inherently cope with the special constraints appearing in 3-dimensional mechanical systems. It is therefore natural to inspect the many solution methods developed for multibody systems and try to generalize one or more of them to general DAEs (1). In the recent report (*Arnold, 2016*), a very broad and nice overview of the current state of the art for simulation of multibody systems is given and used as basis of this section.

One solution method is to integrate the highest derivative equations (10a) and when the violation of the constraints (10b) becomes too large project on the constraint manifold, see for example (*Ascher and Petzold, 1991*; *Eich, 1993*). Such methods could be implemented by utilizing an implicit one-step DAE integrator and after every step project the solution on the constraint manifold.

Many industrial applications of Modelica models are best solved with an implicit multistep method. Unfortunately, multistep methods require non-trivial modifications to embed a projection method because the solution is interpolated smoothly over several steps and (potentially) in every step the solution is modified in a discontinuous way by a projection. It seems that the stabilized index-2 formulation according to Gear, Gupta, Leimkuhler (*Gear et al., 1985*) is a more attractive starting point especially for multistep methods. This method is analyzed in the sequel in some detail. The presentation is made in such a way that a generalization is straightforward.

4.2 Transformation of Multibody Equations

Starting point is the following set of equations for a multibody system:

$$\dot{\boldsymbol{q}} = \boldsymbol{v} \tag{11a}$$
$$\mathbf{M}(\boldsymbol{q},t)\dot{\boldsymbol{v}} + \mathbf{G}^T(\boldsymbol{q},t)\boldsymbol{\lambda} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v},t) \tag{11b}$$
$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{q},t) \tag{11c}$$

with

$$\mathbf{G} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}}, \qquad \mathbf{M} = \mathbf{M}^T > \mathbf{0} \tag{11d}$$

It is assumed that $\mathbf{G}$ has full row rank that is the constraints equations (11c) are not redundant. (11) has $n_q + n_v + n_\lambda$ real unknowns $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}, \boldsymbol{\lambda}$ for the same number of equations. With the new array version of the Pantelides algorithm, equation (11a) is differentiated once and equation (11c) twice in order to arrive at the following equation system that needs to be fulfilled by consistent initial values $\boldsymbol{q}_0, \boldsymbol{v}_0, \boldsymbol{\lambda}_0, \dot{\boldsymbol{q}}_0, \ddot{\boldsymbol{q}}_0, \dot{\boldsymbol{v}}_0$:

$$\dot{\boldsymbol{q}} = \boldsymbol{v} \tag{12a}$$
$$\mathbf{M}\dot{\boldsymbol{v}} + \mathbf{G}^T\boldsymbol{\lambda} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v},t) \tag{12b}$$
$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{q},t) \tag{12c}$$
$$\mathbf{0} = \mathbf{G}\,\dot{\boldsymbol{q}} + \boldsymbol{g}^{(1)}(\boldsymbol{q},t) \tag{12d}$$
$$\mathbf{0} = \mathbf{G}\,\ddot{\boldsymbol{q}} + \boldsymbol{g}^{(2)}(\boldsymbol{q},\dot{\boldsymbol{q}},t) \tag{12e}$$
$$\ddot{\boldsymbol{q}} = \dot{\boldsymbol{v}} \tag{12f}$$

with

$$\boldsymbol{g}^{(1)} = \frac{\partial \mathbf{g}}{\partial \mathrm{t}}, \quad \boldsymbol{g}^{(2)} = \dot{\mathbf{G}}\,\dot{\boldsymbol{q}} + \dot{\boldsymbol{g}}^{(1)} \tag{12g}$$

This is an ODAE (Overdetermined Differential Algebraic Equation) with $2n_q + n_v + 3n_\lambda$ equations for the $2n_q + n_v + n_\lambda$ unknowns $\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \dot{\boldsymbol{v}}, \boldsymbol{\lambda}$. In order to arrive at an equation system with the same number of unknowns and equations that are consistent (so locally a unique solution exists), the following approach of (*Gear et al., 1985*) is used:

$$\mathbf{0} = \dot{\boldsymbol{q}} - \boldsymbol{v} + \mathbf{G}^T\boldsymbol{\mu} \tag{13a}$$
$$\mathbf{0} = \mathbf{M}\,\dot{\boldsymbol{v}} + \mathbf{G}^T\,\boldsymbol{\lambda} - \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v},t) \tag{13b}$$
$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{q},t) \tag{13c}$$
$$\mathbf{0} = \mathbf{G}\,\boldsymbol{v} + \boldsymbol{g}^{(1)}(\boldsymbol{q},t) \tag{13d}$$

These are $n_q + n_v + 2n_\lambda$ residue equations for the $n_q + n_v + 2n_\lambda$ unknowns $\dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}, \boldsymbol{\lambda}, \boldsymbol{\mu}$, so the number of equations and number of unknowns is the same. (13) has a *differential index of two* and has the same solution as (12) because it can be shown that $\boldsymbol{\mu} = \mathbf{0}$: Inserting equation (13a) in equation (13d):

$$\mathbf{0} = \mathbf{G}\,(\dot{\boldsymbol{q}} + \mathbf{G}^T\,\boldsymbol{\mu}) + \boldsymbol{g}^{(1)}(\boldsymbol{q},t)$$

and subtracting the derivative of (13c) results in the equation $\mathbf{0} = \mathbf{G}\,\mathbf{G}^T\boldsymbol{\mu}$. Provided $\mathbf{G}$ has full row rank, $\mathbf{G}\,\mathbf{G}^T$ is regular and therefore $\boldsymbol{\mu} = \mathbf{0}$. ■

This scheme can be easily generalized. For example assume that (say due to an inverse model) the third derivative of (12c) is needed. Then, new $\boldsymbol{\mu}_2$ variables and corresponding dummy derivatives are introduced in combination with the second derivatives of the constraints:

$$\boldsymbol{w} = \dot{\boldsymbol{v}} + \mathbf{G}^T\,\boldsymbol{\mu}_2$$
$$\mathbf{0} = \mathbf{G}\boldsymbol{w} + \boldsymbol{g}^{(2)}(\boldsymbol{q},\boldsymbol{v},t)$$

With the same argument as before it can be shown that $\boldsymbol{\mu}_2 = \mathbf{0}$: Inserting $\boldsymbol{w}$ in the second equation and subtracting the differentiated equation (13d) results in $\mathbf{0} = \mathbf{G}\,\mathbf{G}^T\boldsymbol{\mu}_2$ and therefore $\boldsymbol{\mu}_2 = \mathbf{0}$. ■

In (*Gear et al., 1985*) it is shown that variable-step and variable order BDF (Backward Differentiation Formula) methods converge for this index-2 DAE. However, (13) is not yet in the desired form (2). In particular, the BDF iteration matrix becomes singular for a small step size. (13) can be transformed to (2) by using the substitution (*Gear, 1988*):

$$\boldsymbol{\mu} = \dot{\boldsymbol{\mu}}_{int}, \qquad \boldsymbol{\lambda} = \dot{\boldsymbol{\lambda}}_{int} \tag{14}$$

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

$$\mathbf{0} = \begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}},\mathbf{x},t) \\ \mathbf{f}_c(\mathbf{x},t) \end{bmatrix} = \left[\begin{array}{c} \dot{\mathbf{q}} - \mathbf{v} + \mathbf{G}^{\mathrm{T}}\dot{\boldsymbol{\mu}}_{int} \\ \mathbf{M}\,\dot{\mathbf{v}} + \mathbf{G}^{\mathrm{T}}\,\dot{\boldsymbol{\lambda}}_{int} - \mathbf{h}(\mathbf{q},\mathbf{v},t) \\ \hline \mathbf{g}(\mathbf{q},t) \\ \mathbf{G}\,\mathbf{v} + \mathbf{g}^{(1)}(\mathbf{q},t) \end{array}\right] \tag{15}$$

Note, the Jacobian (2b) of (15) is regular ($\mathbf{P}$ in (16) is a permutation matrix to exchange the third and the fourth equation of (15) in order that the regularity is at once visible):

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} \\ \frac{\partial \mathbf{f}_c}{\partial \mathbf{x}} \end{bmatrix} = \mathbf{P}\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{G}^T \\ \mathbf{0} & \mathbf{M} & \mathbf{G}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{G} & \mathbf{0} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \text{ is regular} \tag{16}$$

4.3 Transformation of general DAEs

The goal is to transform the ODAE (10) to (2). In a first step the elements of vector $\mathbf{x}$ are identified:

1. All variables v_j that *appear differentiated* are collected together with their derivatives in vector $\mathbf{x}_d$ with exception of their highest derivatives (so $x_{d,j}$ are all variables v_j with $A_j > 0$).
2. All variables v_j that do *not appear differentiated* (so v_j with $A_j = 0$ and $A_{inv,j} = 0$) are collected either in vector $\mathbf{x}_a$ or vector $\boldsymbol{\lambda} = \dot{\boldsymbol{\lambda}}_{int}$ in such a way that (a) all assigned variables of every BLT block are either only differential variables ($\dot{\mathbf{x}}_d; \dot{\boldsymbol{\lambda}}_{int}$) or only algebraic ($\mathbf{x}_a$) variables and (b) a BLT block with assigned $\mathbf{x}_a$ variables does not contain any differential variables ($\dot{\mathbf{x}}_d; \dot{\boldsymbol{\lambda}}_{int}$).
3. New n_μ unknown variables $\boldsymbol{\mu}_{int}$ are introduced (n_μ is defined below).

The index-one DAE (2) can now be defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_a \\ \boldsymbol{\lambda}_{int} \\ \boldsymbol{\mu}_{int} \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_d \\ \dot{\mathbf{x}}_a \\ \boldsymbol{\lambda} \\ \dot{\boldsymbol{\mu}}_{int} \end{bmatrix} \tag{17a}$$

$$\mathbf{0} = \begin{bmatrix} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t) \\ \mathbf{f}_c(\mathbf{x}, t) \end{bmatrix} = \left[\begin{array}{c} \dot{\mathbf{x}}_{der(0:n-2)} - \mathbf{x}_{der(1:n-1)} + \mathbf{G}^T \dot{\boldsymbol{\mu}}_{int} \\ \mathbf{r}_{0,d} \\ \hline \mathbf{r}_{0,a} \\ \mathbf{r}_{DER(0)} \\ \mathbf{r}_{DER(1:n-1)} = \mathbf{G}\mathbf{x}_{der(1:n-1)} + \mathbf{g} \end{array}\right] \tag{17b}$$

The variables in **gray color**, that is $\boldsymbol{\lambda}_{int}, \boldsymbol{\mu}_{int}, \dot{\mathbf{x}}_a$, are not used in equations (17b) and are variables needed by the integrator. The different parts of the equations are:

1. The *index vectors* $der(0{:}n-2), der(1{:}n-1)$ are defined in such a form that (for $\boldsymbol{\mu}_{int} = \mathbf{0}$): $$\frac{d(\mathbf{x}_{der(0:n-2)})}{dt} = \mathbf{x}_{der(1:n-1)}$$
2. $\mathbf{r}_{0,d}$ are *non-differentiated* equations of (10a), so equations e_i with $B_i = 0$ and $B_{inv,i} = 0$, that have only differentiated variables ($\dot{\mathbf{x}}_d; \dot{\boldsymbol{\lambda}}_{int}$) as assigned variables of the respective BLT block.
3. $\mathbf{r}_{0,a}$ are *non-differentiated* equations of (10a), so equations e_i with $B_i = 0$ and $B_{inv,i} = 0$, that have only algebraic variables ($\mathbf{x}_a$) as assigned variables of the respective BLT block.
4. $\mathbf{r}_{DER(0)}$ are constraint equations (10b) that are not differentiated, so equations e_i with $B_i = 0$ and $B_{inv,i} = 0$.
5. $\mathbf{r}_{DER(1:n-1)} = \mathbf{G}(\mathbf{x}_{der(0:n-2)}, t)\mathbf{x}_{der(1:n-1)} + \mathbf{g}(\mathbf{x}_{der(0:n-2)}, t)$ are constraint equations (10b) that are differentiated at least once, but not the highest derivative equations, so equations e_i with $B_i > 0$ and $B_{inv,i} > 0$. The number of additionally introduces variables n_μ is equal to the number of equations of $\mathbf{r}_{DER(1:n-1)}$.
6. Matrix $\mathbf{G}$ collects the linear factors of the equations $r_{DER(1:n-1),i}$ with respect to the highest derivatives $\mathbf{x}_{der(1:n-1),i}$ appearing in the resp. equation e_i. Note, as recognized in (*Führer, 1988*) in a similar context, $\mathbf{G}$ is part of the iteration matrix (Jacobian) of a BDF integrator and therefore if the iteration matrix is computed numerically, $\mathbf{G}$ is determined *without additional effort*, see also (*Arnold, 2016*).

With this structuring we can now prove the following theorems:

Theorem 4: *(17) is a DAE (2) under the assumption that $\frac{\partial e_i}{\partial v_j}$ for $A_j = 0$ in (10a) is regular (and not just structurally regular) and **G** has full row rank (= the constraints are not redundant).*

Proof: (a) Due to the construction, the upper two equations of (17b) are a function of $\dot{\mathbf{x}}, \mathbf{x}, t$ and the lower three equations are not functions of $\dot{\mathbf{x}}$, so (17b) has the functional dependency as required by (2a).

(b) $d\boldsymbol{r}_{DER(0:n-2)}/dt = \mathbf{r}_{DER(1:n-1)} \rightarrow$

$$\mathbf{0} = \mathbf{G}(\dot{\mathbf{x}}_{der(0:n-2)} - \mathbf{x}_{der(1:n-1)}) = \mathbf{G}\mathbf{G}^T \dot{\boldsymbol{\mu}}_{int} \quad \rightarrow \quad \dot{\boldsymbol{\mu}}_{int} = \mathbf{0}$$

(c) If the highest order constraint equations in the lower part of (17b) are differentiated once, then these differentiated equations, together with the second and third equation of (17b) are the highest order derivative equations of (10a) which can be solved for the highest order derivatives (so for $\dot{\mathbf{x}}$, since $\dot{\boldsymbol{\mu}}_{int} = \mathbf{0}$) due to the assumption and therefore (2b) holds.■

Theorem 5: *(17) and (9) have the same solution space.*

Proof: Since $\dot{\boldsymbol{\mu}}_{\text{int}} = \mathbf{0}$, (17) are equations (9) and differentiated equations of (9). ■

To summarize, every DAE (1) can be transformed to DAE (17) *without* solving linear or nonlinear algebraic equation systems provided the Pantelides algorithm or an equivalent structural index reduction algorithm can be applied to it. (17) is an index one DAE (2).

4.4 Example

(17) is demonstrated with the following example from (*Mattsson and Söderlind, 1993*) that has been extended with additional equations and unknowns to include several special cases on the basis of a simple DAE:

$$0 = u_1(t) + x_1 - x_2 \tag{18a}$$
$$0 = u_2(t) + x_1 + x_2 - x_3 + \dot{x}_6 \tag{18b}$$
$$0 = u_3(t) + x_1 + \dot{x}_3 - x_4 \tag{18c}$$
$$0 = u_4(t) + 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + \dddot{x}_6 \tag{18d}$$
$$0 = u_5(t) + 3\ddot{x}_1 + 2\ddot{x}_2 + x_5 + 0.1x_8 \tag{18e}$$
$$0 = u_6(t) + 2x_6 + x_7 \tag{18f}$$
$$0 = u_7(t) + 3x_6 + 4x_7 \tag{18g}$$
$$0 = u_8(t) + x_8 - \sin(x_8) \tag{18h}$$

The $u_i(t)$ are known forcing functions. Applying the Pantelides algorithm (*Pantelides, 1988*)[2] and sorting the equations on highest derivative level results in the following three BLT components:

BLT component 1 (unknowns: x_8)

$$0 = u_8(t) + x_8 - \sin(x_8) \qquad \text{(19h)}$$

BLT component 2 (unknowns: $\dddot{x}_6, \dddot{x}_7$)

$$0 = \dddot{u}_6(t) + 2\dddot{x}_6 + \dddot{x}_7 \qquad (19\dddot{f})$$
$$0 = \dddot{u}_7(t) + 3\dddot{x}_6 + 4\dddot{x}_7 \qquad (19\dddot{g})$$

BLT component 3 (unknowns: $\ddot{x}_1, \ddot{x}_2, \ddot{x}_3, \dot{x}_4, x_5$)

$$0 = \ddot{u}_1(t) + \ddot{x}_1 - \ddot{x}_2 \qquad (19\ddot{a})$$
$$0 = \ddot{u}_2(t) + \ddot{x}_1 + \ddot{x}_2 - \ddot{x}_3 + \dddot{x}_6 \qquad (19\ddot{b})$$
$$0 = \dot{u}_3(t) + \dot{x}_1 + \ddot{x}_3 - \dot{x}_4 \qquad (19\dot{c})$$
$$0 = u_4(t) + 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + \dddot{x}_6 \qquad (19d)$$
$$0 = u_5(t) + 3\ddot{x}_1 + 2\ddot{x}_2 + x_5 + 0.1x_8 \qquad (19e)$$

Transformation to the index one DAE (17) results in:

$$\mathbf{x}_d = [x_1;\ x_2;\ x_3;\ x_4;\ x_6;\ x_7; \dot{x}_1; \dot{x}_2; \dot{x}_3; \dot{x}_6; \dot{x}_7; \ddot{x}_6; \ddot{x}_7]$$
$$\mathbf{x}_a = [x_8]$$
$$\dot{\boldsymbol{\lambda}}_{int} = [x_5]$$
$$\mathbf{r}_{0,d} = \begin{bmatrix} u_4(t) + 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + \dddot{x}_6 \\ u_5(t) + 3\ddot{x}_1 + 2\ddot{x}_2 + x_5 + 0.1x_8 \end{bmatrix}$$
$$\mathbf{r}_{0,a} = [u_8(t) + x_8 - \sin(x_8)]$$
$$\mathbf{r}_{DER(0)} = \begin{bmatrix} u_1(t) + x_1 - x_2 \\ u_2(t) + x_1 + x_2 - x_3 + \dot{x}_6 \\ u_3(t) + x_1 + \dot{x}_3 - x_4 \\ u_6(t) + 2x_6 + x_7 \\ u_7(t) + 3x_6 + 4x_7 \end{bmatrix}$$
$$\mathbf{r}_{DER(1:n-1)} = \begin{bmatrix} \dot{u}_1(t) + \dot{x}_1 - \dot{x}_2 \\ \dot{u}_2(t) + \dot{x}_1 + \dot{x}_2 - \dot{x}_3 + \ddot{x}_6 \\ \dot{u}_6(t) + 2\dot{x}_6 + \dot{x}_7 \\ \dot{u}_7(t) + 3\dot{x}_6 + 4\dot{x}_7 \\ \ddot{u}_6(t) + 2\ddot{x}_6 + \ddot{x}_7 \\ \ddot{u}_7(t) + 3\ddot{x}_6 + 4\ddot{x}_7 \end{bmatrix}'$$
$$\mathbf{G} = \begin{bmatrix} 1 & -2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 3 & 4 \end{bmatrix}$$
$$der(0{:}n-2) = [1; 2; 3; 5; 6; 10; 11]$$
$$der(1{:}n-1) = [7; 8; 9; 10; 11; 12; 13]$$

The result is a DAE with 21 equations that can be solved with an index one DAE integrator.

[2] For simplicity of the example, the equations contain higher derivatives. The Pantelides algorithm can be easily generalized to this case by providing a corresponding **A** vector.

4.5 Structuring the Constraint Sets

The direct mapping to (17) results often in unnecessarily large DAEs. We will therefore now improve the mapping by utilizing (partial) *static state selection*. As a first step, the inherent structure of the constraint set (10b) is (algorithmically) determined. This can be performed elegantly by utilizing results from the dummy derivative method of (*Mattsson and Söderlind, 1993*). The "lower derivative" equations (10b) determined by the Pantelides algorithm are ignored in the sequel and they are instead newly derived from the sorted highest derivative equations (10a) by inspecting every BLT component in sequence and for component k the follow actions are performed:

Step 1: The differentiation order of an equation in component k is reduced by one if it is a differentiated equation. The resulting set of equations forms an independent constraint set.

Step 2: The differentiation order of an unknown (= assigned variable) in component k is reduced by one if it is a differentiated variable. The resulting set of variables contains the unknowns of the derived constraint set.

Step 3: Goto Step 1, if there are still differentiated equations in the derived constraint set and apply Step 1-3 on it. Otherwise, go to Step 4.

Step 4: Order the components in such a way that within a BLT component first the lowest order derivative constraints are placed, then the constraints with one differentiation order higher and so on. The order of the BLT components is not changed.

Applying this procedure to (19) results in the following sorted ODAE:

BLT component 1 (unknowns: x_8)

$$0 = u_8(t) + x_8 - \sin(x_8)$$

BLT component 2

BLT component 2.1 (unknowns: x_6, x_7)

$$0 = u_6(t) + 2x_6 + x_7$$
$$0 = u_7(t) + 3x_6 + 4x_7$$

BLT component 2.2 (unknowns: $\dot{x}_6, \dot{x}_7$)

$$0 = \dot{u}_6(t) + 2\dot{x}_6 + \dot{x}_7$$
$$0 = \dot{u}_7(t) + 3\dot{x}_6 + 4\dot{x}_7$$

BLT component 2.3 (unknowns: $\ddot{x}_6, \ddot{x}_7$) (20)

$$0 = \ddot{u}_6(t) + 2\ddot{x}_6 + \ddot{x}_7$$
$$0 = \ddot{u}_7(t) + 3\ddot{x}_6 + 4\ddot{x}_7$$

BLT component 2.4 (unknowns: $\dddot{x}_6, \dddot{x}_7$)

$$0 = \dddot{u}_6(t) + 2\dddot{x}_6 + \dddot{x}_7$$
$$0 = \dddot{u}_7(t) + 3\dddot{x}_6 + 4\dddot{x}_7$$

BLT component 3

BLT component 3.1 (unknowns: x_1, x_2, x_3)

$$0 = u_1(t) + x_1 - x_2$$
$$0 = u_2(t) + x_1 + x_2 - x_3 + \dot{x}_6$$

BLT component 3.2 (unknowns: $\dot{x}_1, \dot{x}_2, \dot{x}_3, x_4$)

$$0 = \dot{u}_1(t) + \dot{x}_1 - \dot{x}_2$$
$$0 = \dot{u}_2(t) + \dot{x}_1 + \dot{x}_2 - \dot{x}_3 + \ddot{x}_6$$
$$0 = u_3(t) + x_1 + \dot{x}_3 - x_4$$

BLT component 3.3 (unknowns: $\ddot{x}_1, \ddot{x}_2, \ddot{x}_3, \dot{x}_4, x_5$)

$$0 = \ddot{u}_1(t) + \ddot{x}_1 - \ddot{x}_2$$
$$0 = \ddot{u}_2(t) + \ddot{x}_1 + \ddot{x}_2 - \ddot{x}_3 + \dddot{x}_6$$
$$0 = \dot{u}_3(t) + \dot{x}_1 + \ddot{x}_3 - \dot{x}_4$$
$$0 = u_4(t) + 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + \ddot{x}_6$$
$$0 = u_5(t) + 3\ddot{x}_1 + 2\ddot{x}_2 + x_5 + 0.1x_8$$

Due to the construction, a variable like $\ddot{x}_6$ is computed in a BLT sub-component (here: BLT component 2.3) and used only in later BLT components (here BLT component 3.2).

In the next step the number of constraint equations and unknowns shall be reduced statically. In principal this is just a variant of the dummy derivative method. However, (*Mattsson and Söderlind, 1993*) describe a (very useful) conceptual algorithm, but not how to implement it practically for nonlinear systems which requires non-trivial extensions. In order to do this, an auxiliary algorithm is needed that is described in the next section.

4.6 Tearing with retained solution space

Starting point is a nonlinear algebraic equation system

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{z}), \qquad \boldsymbol{z} \in \mathbb{R}^{nz}, \boldsymbol{g} \in \mathbb{R}^{ng}, ng \le nz \qquad (21)$$

where the number of equations ng is at most the same as the number of unknowns nz, but it may be less. The goal is to split this equation system in an explicitly solvable part and an implicit part:

$$\boldsymbol{z}_e := \boldsymbol{g}_e(\boldsymbol{z}_e, \boldsymbol{z}_t) \qquad (22a)$$
$$0 = \boldsymbol{g}_r(\boldsymbol{z}_e, \boldsymbol{z}_t) \qquad (22b)$$

where $\boldsymbol{z}_e$ can be solved recursively from (22a) by utilizing only already computed elements of $\boldsymbol{z}_e$ when calculating a new element of $\boldsymbol{z}_e$. $\boldsymbol{z}_e$ are called the explicitly solvable variables of $\boldsymbol{z}$, $\boldsymbol{z}_t$ the tearing variables of $\boldsymbol{z}$ and $\boldsymbol{g}_r$ the residue equations. The interpretation is that when $\boldsymbol{z}_t$ is given, $\boldsymbol{z}_e$ can be explicitly computed and $\boldsymbol{z}_t$ must be provided in such a way that the residues equations are fulfilled.

In order that this transformation is practically useful, the solution space of (22) must be identical to the solution space of (21). In the following an algorithm is derived to automatically deduce (22) from (21) such that (22) has the same solution space as (21).

Tearing is a well-known technique and was probably introduced by (*Kron, 1962*). A recent extensive literature survey is given in (*Bahainv et al., 2016a*). In (*Bahainv et al., 2016b*) a novel tearing technique is proposed based on an integer programming formulation with a custom branch and bound algorithm. Tearing was used in object-oriented modeling, for example in (*Elmqvist and Otter, 1994*) and in (*Carpanzano et al., 1997*). The following algorithm sketch for automatic tearing is due to a development of the authors of this paper in 1999. Some results of it have been reported in (*Otter, 1999*):

Equation (22a) can be interpreted as a DAG (Directed Acyclic Graph) where the nodes are equations $g_{e,i}$ together with the explicitly solved variables $z_{e,i}$ of $g_{e,i}$, and the edges of a node i are directed to the nodes of the remaining variables $z_{e,i}$ appearing in $g_{e,i}$. The goal of the algorithm is to construct such a DAG using equations and unknowns from (21). Initially, the DAG is empty and is constructed with the following steps:

Step 1: Select an array equation i from (21) that is not yet in the DAG (in a first step equations are selected according to their initial ordering; later, heuristics for the selection are added based on additional information).

Step 2: Select an array variable j from equation i that is (a) not assigned to equation i, (b) not yet selected before in this equation, and (c) can be explicitly solved from equation i (so the array size of variable j and of the array equation i must agree) without changing the solution space of this equation (e.g. using only variables z_j as candidates that are within linear factors $c \cdot z_j$ where c is a constant with $c \neq 0$; see also the discussion about heuristics below and (*Otter, 1999*)).

Step 3: Add equation i and the selected variable j from Step 2 as node to the potential DAG.

Step 4: Traverse the potential DAG *starting* from the added equation node and use a standard DFS (Depth First Search) to determine if there is a cycle for this equation node. If there is a cycle, remove the last added equation from it and if not all variables of equation i have been inspected, go to Step 2. If no cycle is present, continue with the next step.

Step 5: If not all equations have been inspected, go to Step 1. Otherwise, stop (equations (22a) are the equations in the DAG, $\boldsymbol{z}_e$ are the assigned variables in the DAG).

With m the number of variable incidences in the system of equations, the worst time complexity of this algorithm to find the tearing variables and residue equations is $O(m^2)$ because every DFS from every inserted node has a worst-time complexity of $O(m)$ and this operation is executed potentially m times (since in the worst case a DFS is performed on every variable in every equation).

In the last decade, several new algorithms have been developed to perform *incremental cycle detection* when inserting vertices and edges to an existing DAG. The algorithm of (*Bender et al., 2016*) has the currently best worst case performance for sparse DAGs with $O(\min(m^{1/2}, n^{2/3}) \cdot m)$, where n is the number of vertices and m the number of edges. For comparison of such algorithms, see (*Sigurðsson, 2016*).

The implementation in Modia/Julia uses currently the simple *Algorithm N* of (*Bender et al., 2016*) that has a worst time complexity of $O(n \cdot m)$. For example, when an equation system of the following form is present

$$\begin{aligned} 0 &= f_1(z_1, z_n) \\ 0 &= f_2(z_2, z_1) \\ 0 &= f_3(z_3, z_2) \\ &\dots \\ 0 &= f_n(z_n, z_{n-1}) \end{aligned} \tag{23}$$

and the equations are added in the order 1,2,..., n (or also in the order n,n-1,...,1), this tearing algorithm has a complexity of $O(n)$ to find the single tearing variable. On a standard notebook, this takes about 2 s for $n = 10^6$.

The result of the tearing algorithm depends on the order in which equations are added to the DAG. There are at least two useful heuristics: (a) If the user explicitly requires to solve equations for particular variables (for example using the operator ":=" instead of "=" in the Modia prototype, or the algorithm section or a function call in Modelica), then these equations are *inspected first*. All equations belonging to the connection graph (especially all linear equations with only Integer coefficients, see section 5) are inspected *last*, because it seems most natural for a physical system to cut an algebraic loop along the connection graph, and not within a component.

Tearing can have a significant influence on the reliability of the numerical solution and therefore it is not always clear whether it is useful to apply tearing to solve algebraic loops. For this reason, more heuristics need to be added, for example, arrays might be solved in Step 2 above only, if they appear as linear term and the linear factors are the scalars +1 or -1, in order to avoid a potential division by a small value, or if the linear term is an orthogonal matrix so that inversion is reliable.

4.7 Partial state selection

The tearing algorithm from the last section shall now be used to partially solve the constraint equations and thereby identify states and dummy states. The constraint equation sets are derived with the approach of section 4.5 and all these sets have the following structure:

$$\begin{gathered} \mathbf{0} = \boldsymbol{g}(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ \boldsymbol{x}_1 \in \mathbb{R}^{nx1}, \boldsymbol{x}_2 \in \mathbb{R}^{nx2} \boldsymbol{g} \in \mathbb{R}^{ng}, ng \le nx1 \end{gathered} \tag{24}$$

where $\boldsymbol{x}_1$ are potential states, $\boldsymbol{x}_2$ are potential states that have been already handled in a previous BLT sub-component (so can be treated as known variables) and $\boldsymbol{g}(..)$ is a set of algebraic constraint equations on $\boldsymbol{x}_1$. Via tearing the constraint equation set (24) can be split in an explicitly solvable part $\boldsymbol{g}_e(..)$ and in an implicit part $\boldsymbol{g}_r(..)$:

$$\begin{aligned} \mathbf{x}_{1e} &:= \boldsymbol{g}_e(\boldsymbol{x}_{1e}, \boldsymbol{x}_{1t}, \boldsymbol{x}_2) \\ 0 &= \boldsymbol{g}_r(\boldsymbol{x}_{1e}, \boldsymbol{x}_{1t}, \boldsymbol{x}_2) \end{aligned} \tag{25}$$

The explicitly solvable variables $\mathbf{x}_{1e}$ are dummy states according to the dummy derivative method of (*Mattsson and Söderlind, 1993*). The tearing variables $\boldsymbol{x}_{1t}$ remain potential states. If no residue equations $\boldsymbol{g}_r(..)$ are present, the full set of states has been identified. If the equations are *linear* in $\boldsymbol{x}_{1e}, \boldsymbol{x}_{1t}$, then the residue equations can be transformed to a linear equation in the tearing variables, see for example (*Elmqvist and Otter, 1994*). For *constant coefficient linear systems*, this equation system can be at once solved. For *variable coefficient linear systems*, an inline solution of the linear system might be used, at least for systems with up to three unknowns. In all these cases the full set of states has been identified as well.

The state selection with tearing is applied on all constraint sets starting from the lower to the higher derivative constraint sets ec[1]...ec[end] of every BLT component. Since by construction ec[i] is a superset of ec[i-1], and the unknowns $\dot{\boldsymbol{v}}_i$ of ec[i] are a differentiated superset of the unknowns $\boldsymbol{v}_{i-1}$, all explicitly solvable equations of ec[i-1] are also explicitly solvable equations of ec[i] and therefore tearing need only to be performed *additionally* on equations that are *added* at a higher level.

Applying the partial state selection for example on BLT sub-component 3.1 of (20) identifies x_2 as state and computes the dummy states from:

$$\begin{aligned} x_1 &:= -u_1(t) + x_2 \\ x_3 &:= -u_2(t) - x_1 + x_2 - \dot{x}_6 \end{aligned}$$

The same analysis holds for BLT sub-component 3.2, so $\dot{x}_2$ is also a state and the following equations are directly deduced from the previous equations:

$$\begin{aligned} \dot{x}_1 &:= -\dot{u}_1(t) + \dot{x}_2 \\ \dot{x}_3 &:= -\dot{u}_2(t) - \dot{x}_1 + \dot{x}_2 - \ddot{x}_6 \end{aligned}$$

Tearing must therefore only be applied for the additional equation of this sub-component leading to:

$$x_4 := u_3(t) + x_1 + \dot{x}_3$$

Applying the partial state selection on BLT sub-component 2.1 of (20) identifies x_6 as state and computes the dummy state from

$$x_7 := -u_6(t) - 2x_6$$

The residue equation is linear in x_6 and can then be solved for:

$$x_6 := (-u_7(t) - 4u_6(t))/5$$

Once partial state selection has been applied on all constraint sets, all the dummy states and their derivatives, *up to the highest derivatives of the dummy states*, are removed from $\mathbf{x}$ and $\dot{\boldsymbol{x}}$ and are computed *locally* from the remaining $\mathbf{x}$ and $\dot{\mathbf{x}}$. The remaining equations can be transformed to DAE (17). Hereby, the sorting order of the locally solved equations matters and therefore the ordering has to be performed according to the BLT ordering and the corresponding ordering of the constraint sets.

Applying partial state selection with tearing on example (20) results in the following DAE (17):

$$
\begin{aligned}
\mathbf{x}_d &= [x_2;\ \dot{x}_2] \\
\mathbf{x}_a &= [x_8] \\
\boldsymbol{\lambda}_{int} &= [\quad] \\
\dot{\boldsymbol{\mu}}_{int} &= [\quad] \\
x_6 &:= \left(-u_7(t) - 4u_6(t)\right)/5 \\
x_7 &:= -u_6(t) - 2x_6 \\
\dot{x}_6 &:= \left(-\dot{u}_7(t) - 4\dot{u}_6(t)\right)/5 \\
\dot{x}_7 &:= -\dot{u}_6(t) - 2\dot{x}_6 \\
\ddot{x}_6 &:= \left(-\ddot{u}_7(t) - 4\ddot{u}_6(t)\right)/5 \\
\ddot{x}_7 &:= -\ddot{u}_6(t) - 2\ddot{x}_6 \\
\dddot{x}_6 &:= \left(-\dddot{u}_7(t) - 4\dddot{u}_6(t)\right)/5 \\
\dddot{x}_7 &:= -\dddot{u}_6(t) - 2\dddot{x}_6 \\
x_1 &:= -u_1(t) + x_2 \\
x_3 &:= -u_2(t) - x_1 + x_2 - \dot{x}_6 \\
\dot{x}_1 &:= -\dot{u}_1(t) + \dot{x}_2 \\
\dot{x}_3 &:= -\dot{u}_2(t) - \dot{x}_1 + \dot{x}_2 - \ddot{x}_6 \\
x_4 &:= u_3(t) + x_1 + \dot{x}_3 \\
\ddot{x}_1 &:= -\ddot{u}_1(t) + \ddot{x}_2 \\
\ddot{x}_3 &:= -\ddot{u}_2(t) - \ddot{x}_1 + \ddot{x}_2 - \dddot{x}_6 \\
\dot{x}_4 &:= \dot{u}_3(t) + \dot{x}_1 + \ddot{x}_3 \\
x_5 &:= -u_5(t) - 3\ddot{x}_1 - 2\ddot{x}_2 - 0.1x_8 \\
\mathbf{r}_{0,d} &= [u_4(t) + 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + \ddot{x}_6] \\
\mathbf{r}_{0,a} &= [u_8(t) + x_8 - \sin(x_8)] \\
\mathbf{G} &= [\,] \\
der(0{:}n-2) &= [1] \\
der(1{:}n-1) &= [2] \quad (\rightarrow\ \dot{x}_{d,1} = x_{d,2})
\end{aligned}
$$

The result is a DAE with 3 equations (without partial state selection, it had been 21 equations).

Using tearing for the constraint equations seems to be always a useful approach because this can significantly reduce the number of variables that need to be discretized by the integrator. For example, assume a tree-structured multi-body system is modeled with the Modelica.Mechanics.MultiBody library and has n bodies that are connected together by revolute joints. Without partial state selection, DAE (17) consists of $(6 \cdot 3 + 4 \cdot 9 + 2)n = 56n$ equations[3]. After partial state selection with tearing the DAE consists of the $2n$ equations (15), provided the simple heuristics from section 4.6 are used.

However, it is less clear whether tearing is also useful when applied on the highest order derivative equations that are no derivatives of constraints. For example, discretized partial differential equations typically lead to structures where tearing cannot reduce the equation size much but will completely destroy the sparseness and may be numerically less reliable. In such cases it is much better to not apply tearing and rely on the sparse matrix handling used by the integrator. More investigations are needed here.

[3] $\mathbf{x} = \left[\mathbf{r}_i; \dot{\mathbf{r}}_i; \mathbf{T}_i; \dot{\mathbf{T}}_i; \boldsymbol{\omega}_i; \mathbf{r}_i^{CM}; \dot{\mathbf{r}}_i^{CM}; \mathbf{T}_i^{CM}; \dot{\mathbf{T}}_i^{CM}; \boldsymbol{\omega}_i^{CM}; \varphi_i; \dot{\varphi}_i\right]$
$i = 1,2,\ldots n$

5 Exact Removal of Singularities

5.1 Overview

In this section a new method is proposed to exactly remove certain types of singularities of a physical system model provided as DAE (1). The result is again a DAE in form (1). A typical example is shown in Figure 1.

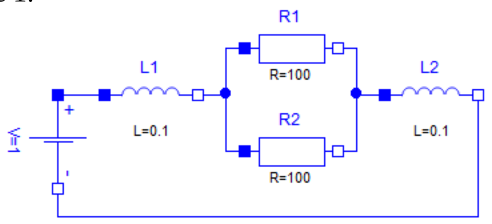

Figure 1. Modelica model of an electrical circuit that is difficult to simulate. It can be automatically handled with the method of this section.

Modelica tools transform DAEs (1) with structural symbolic algorithms. These algorithms fail for the circuit in Figure 1 (as well as other useful application models). Since this electrical circuit is not grounded, the potentials of the electrical Pins can float, that is, the system equations are underdetermined. On the other hand, the equations are overdetermined regarding currents. An analysis, literature survey, and a solution based on exploitation of the connection graph is presented in (*Elmqvist and Mattsson, 2016*).

Additionally, the currents i_{L1}, i_{L2} appear differentiated in the inductors L_1, L_2 and are therefore assumed to be known. The two inductors are connected by two resistors in parallel leading to the following connection equations for the currents:

$$
\begin{aligned}
i_{L1} &= i_{R1} + i_{R2} \\
i_{L2} &= i_{R1} + i_{R2}
\end{aligned}
\tag{26}
$$

where i_{R1}, i_{R2} are the currents through the resistances R_1, R_2. Structurally, (26) are two equations for two unknown algebraic variables i_{R1}, i_{R2} since the potential states i_{L1}, i_{L2} are assumed to be known. *Therefore, structural algorithms assume that* i_{R1}, i_{R2} *can be determined from* (26). However, when subtracting the two equations $i_{L1} + i_{L2} = 0$, that is an equation with only known variables is obtained, which means that one of the two variables cannot be a state. As a result structural index reduction algorithms, as discussed in section 3, will fail on this circuit.

The method below is based on the observation that object-oriented models have a particular structure: Zero-sum equations of flow variables i in connectors

c_k have the form $\sum c_k.i = 0$. After alias elimination, relative potential variables in a component have the form $u_{rel} = c_k.v - c_j.v$. All these equations have the common property that they are linear and the coefficients are integer (even +1 or -1). Due to the integer coefficients *exact analysis* is possible. In particular, singularities and state constraints in linear equations with integer coefficients are identified and if possible removed. The latter *cannot* be achieved with methods based on connection graphs, such as (*Elmqvist and Mattsson, 2016*) or similar techniques.

When applied to the circuit in Figure 1, the method in section 5.3 gives the result that the following equation shall be removed since redundant:

```
-L2.n.i - V.n.i = 0
```

In order to make all potentials well-defined, the following equation is added:

```
L2.n.v = 0
```

In order to make the state constraints structurally visible, the equation

```
-R1.p.i - R2.p.i - L1.n.i = 0
```

is replaced by

```
-L1.p.i + L2.p.i = 0
```

5.2 Transformation to upper trapezoidal form

The new approach is based on a utility algorithm to perform a fraction-free Gaussian elimination of linear algebraic equations with integer coefficients. The algorithm is a slight generalization of (*Bareiss, 1968*), see also (*Turner 1995*). Starting point is a linear algebraic equation system

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B}, \qquad \mathbf{A} \in \mathbb{Z}^{na1\ x\ na2}, \mathbf{B} \in \mathbb{Z}^{na1\ x\ nb2} \qquad (27)$$

where **A** and **B** are sparse, rectangular integer matrices. The goal is to use fraction-free Gaussian elimination to transform (27) to upper trapezoidal form:

$$\begin{bmatrix} \mathbf{A}_{u11} & \mathbf{A}_{u12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{X}_{u1} \\ \mathbf{X}_{u2} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{u1} \\ \mathbf{B}_{u2} \end{bmatrix} \qquad (28)$$

where $\mathbf{A}_{u11}, \mathbf{A}_{u12}, \mathbf{B}_{u1}, \mathbf{B}_{u2}$ are integer matrices and $\mathbf{A}_{u11}$ is quadratic, regular, and upper triangular with non-zeros on the diagonal, that is $\text{rank}(\mathbf{A}) = \text{size}(\mathbf{A}_{u11}, 1)$. Additionally, permutation vector p_1 describes the accumulated row interchanges of $\mathbf{A}, \mathbf{B}$ and permutation vector p_2 describes the accumulated column interchanges of **A** and row interchanges of **X** such that $\mathbf{X}_u = \mathbf{X}[p_2, :]$. Permutation vector p_2 is selected such that if possible the "upper" part of **X** is utilized in $\mathbf{X}_{u1}$ (this is used in section 5.3).

Algorithm 1 is a straightforward implementation of Gaussian elimination with full pivoting for sparse matrices. The key point are the two equations at the end with the "**div**(a,b)" operator where equation i is subtracted from equation k with a fraction free operation. Here, the non-trivial to derive property is used that the integer division **div**(a,b)=a/b has no remainder in this case. For details see (*Bareiss,1968*).

Algorithm 1

```
(Au,Bu,rk,p1,p2) = upperTrapezoidal(A,B)
# Transform the rectangular linear system A*X=B
# to upper trapezoidal form Au*X[p2,:]=Bu
# A,B,Au,Bu are integer matrices

# initialize variables
(na1,na2) = size(A); nb2 = size(B,2); p1=1:na1; p2=1:na2
Au = copy(A); Bu = copy(B);
oldPivot  = 1

# inspect all rows of Au
for k = 1:na1
  # search column wise for a pivot in Au[k:,min(k,na2):]
  pivotFound = false
  for k2 = k:na2
    for (k1,pivot) in < row indices k1 and values pivot of
                        non-zero entries of column k2 >
      if k1 >= k && pivot != 0
        pivotFound = true
        p1k = k1
        p2k = k2
        break
      end
    end
    if pivotFound; break; end
  end

  # exchange rows/columns such that Au[k,min(k,na2)]≠0
  if pivotFound
    <exchange rows k and p1k of Au, Bu, p1, and
     exchange columns k and p2k of Au and row p2k of p2 >
  else  # submatrix Au[k:na1,:] has only zeros
    rk = k-1
    return (Au, Bu, rk, p1, p2)
  end

  # Subtract row k from rows k+1:na1
  k1 = k+1
  j = k1:na2
  for (i,val) in < row indices i and values val of non-zero
                   entries of column k >
    if i >= k1
      Bu[i,:] = div(pivot*Bu[i,:] – val*Bu[k,:], oldPivot)
      Au[i,j] = div(pivot*Au[i,j] – val*Au[k,j], oldPivot)
      Au[i,k] = 0
    end
  end
  oldPivot = pivot
end
return (Au, Bu, rk, p1, p2)
```

5.3 Identifying singularities in the model

Starting point is the largest subset of equations of DAE (1) that is described by a linear algebraic system

$$\mathbf{A}_x \boldsymbol{v}_x + \mathbf{A}_y \boldsymbol{v}_y + \mathbf{A}_c \boldsymbol{v}_c + \boldsymbol{A}_r \boldsymbol{v}_r = \mathbf{0} \qquad (29)$$

where $\mathbf{A}_x, \mathbf{A}_y, \mathbf{A}_c, \mathbf{A}_r$ are sparse matrices with (scalar) integer elements of appropriate dimensions and $\boldsymbol{v}_x, \boldsymbol{v}_y, \boldsymbol{v}_c, \boldsymbol{v}_r$ are vectors of variables of (1). An element of these vectors may be a scalar, an array, or an instance of any data structure for which the operators "+", "-", "*" are defined (overloaded) as operations between instances of the same type and between an instance of the type and a scalar integer. Furthermore, it is assumed that *within one equation* the variables are all of the same type and arrays have the same dimension sizes (so, one equation may state a relationship between [3,3] matrices, whereas another equation between [6] vectors, and yet another one between scalars). Since an equation can only depend on variables of the same type, equations (29) are basically a disjunct set of equations for different types that are analyzed conceptually in an independent way from each other. The various vectors have the following meaning:

$\boldsymbol{v}_x$	Variables used in the derivative operator, so $\mathbf{der}(v_{x,i})$ appears in the model.
$\boldsymbol{v}_y$	After removing equations (29) from (1), variables $\boldsymbol{v}_y$ are no longer present in (1). Therefore, these variables *must be computed from* (29). For example, if $v_{rel} = p.v - n.v$ and the connector variables $p.v$ and $n.v$ are not used otherwise in the model, they are part of $\boldsymbol{v}_y$.
$\boldsymbol{v}_c$	Variables defined by a parameter expression. For example, if $v_0 = 5$ and v_0 is utilized in other linear equations with integer coefficients, then v_0 is part of $\boldsymbol{v}_c$.
$\boldsymbol{v}_r$	All remaining variables that do not belong to one of the other three categories above.

In a first step, equations (29) are restructured to:

$$\mathbf{A}_{yr}\boldsymbol{v}_{yr} = \mathbf{B}_{xc}(-\boldsymbol{v}_{xc}) \tag{30}$$

with

$$\begin{aligned}\mathbf{A}_{yr} &= [\mathbf{A}_y \quad \mathbf{A}_r],\ \boldsymbol{v}_{yr} = \begin{bmatrix}\boldsymbol{v}_y\\ \boldsymbol{v}_r\end{bmatrix}\\ \mathbf{B}_{xc} &= [\mathbf{A}_x \quad \mathbf{A}_c],\ \boldsymbol{v}_{xc} = \begin{bmatrix}\boldsymbol{v}_x\\ \boldsymbol{v}_c\end{bmatrix}\end{aligned} \tag{31}$$

With Algorithm 1 from section 5.2

$$(\mathbf{A}_u, \mathbf{B}_u, rk, p_1, p_2) = \text{upperTrapezoidal}(\mathbf{A}_{yr}, \mathbf{B}_{xc})$$

(30) can be transformed to upper trapezoidal form:

$$\begin{bmatrix}\mathbf{A}_{u,11} & \mathbf{A}_{u,12}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{v}_{u1}\\ \boldsymbol{v}_{u2}\end{bmatrix} = \begin{bmatrix}\mathbf{B}_{u1}\\ \mathbf{B}_{u2}\end{bmatrix} \cdot (-\boldsymbol{v}_{xc}) \tag{32}$$

where $\mathbf{A}_{u,11}$ is a quadratic, regular, upper triangular integer matrix of size $[rk, rk]$ with non-zeros on the diagonal, $\boldsymbol{v}_{u1} = \boldsymbol{v}_{yr}[p_2[1{:}rk]]$ are rk elements of $\boldsymbol{v}_{yr}$ and $\boldsymbol{v}_{u2} = \boldsymbol{v}_{yr}[p_2[rk+1{:}]]$ are the remaining elements of $\boldsymbol{v}_{yr}$. With $\mathbf{B}_{u2} = [\mathbf{B}_{u2,1} \quad \mathbf{B}_{u2,2}]$, the lower part of (32) can be stated as:

$$\mathbf{B}_{u2,1}\boldsymbol{v}_x = \mathbf{B}_{u2,2}(-\boldsymbol{v}_c) \tag{33}$$

Using Algorithm 1 again:

$$(\mathbf{A}_{ux}, \mathbf{B}_{ux}, rk_x, p_{x1}, p_{x2}) = \text{upperTrapezoidal}(\mathbf{B}_{u2,1}, \mathbf{B}_{u2,2})$$

(33) can be transformed to upper trapezoidal form:

$$\begin{bmatrix}\mathbf{A}_{ux11} & \mathbf{A}_{ux,12}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{v}_{ux1}\\ \boldsymbol{v}_{ux2}\end{bmatrix} = \begin{bmatrix}\mathbf{B}_{ux1}\\ \mathbf{B}_{ux2}\end{bmatrix} \cdot (-\boldsymbol{v}_c) \tag{34}$$

where $\mathbf{A}_{ux,11}$ is a quadratic, regular, upper triangular integer matrix of size $[rk_x, rk_x]$ with non-zeros on the diagonal, $\boldsymbol{v}_{ux1} = \boldsymbol{v}_x[p_{x2}[1{:}rk_x]]$ are rk_x elements of $\boldsymbol{v}_x$ and $\boldsymbol{v}_{ux2} = \boldsymbol{v}_x[p_{x2}[rk_x+1{:}]]$ are the remaining elements of $\boldsymbol{v}_x$.

From (32) and (34) the following conclusions can be drawn regarding singularities in the model equations:

- If $\mathbf{B}_{ux2}$ is not the zero matrix, then there are constraints $\mathbf{B}_{ux2}\boldsymbol{v}_c = \mathbf{0}$ between the parameter expressions $\boldsymbol{v}_c$. A tool may reject such a model. In the following it is assumed that $\mathbf{B}_{ux2} = \mathbf{0}$.
- If $rk + rk_x < \text{length}(\boldsymbol{v}_{yr})$, then the lower part of (34) are zero-equations and represent redundant equations. As a result, the original equations with row indices $p_1[rk + p_{1x}[rk_x + 1, :]]$ can be removed since they can be expressed as a linear combination of the other integer equations. A tool may just remove these equations and print an information message that it removed them.
- If $\boldsymbol{v}_{u2}$ contains elements of $\boldsymbol{v}_y$, then these variables can have an arbitrary value. For example assume that $\boldsymbol{v}_{yr}$ are scalar real variables, then (32) can be solved for $\boldsymbol{v}_{u1}$:
  $$\boldsymbol{v}_{u1} = -\mathbf{A}_{u,11}^{-1}(\mathbf{A}_{u,12}\boldsymbol{v}_{u2} + \mathbf{B}_{u1}\boldsymbol{v}_{xc})$$
  Therefore, for given states $\boldsymbol{v}_{xc}$ and given values of $\boldsymbol{v}_{u2}$, variables $\boldsymbol{v}_{u1}$ can be uniquely computed. Note, since variables $\boldsymbol{v}_y$ *must* be computed from (29) they can be arbitrarily set, if part of vector $\boldsymbol{v}_{u2}$. Since variables $\boldsymbol{v}_r$ appear also in the remaining model equations, they need to be computed in these remaining model equations. A tool may either reject a model where $\boldsymbol{v}_{u2}$ contains elements of $\boldsymbol{v}_y$, or may set arbitrary values (say the null-element of the respective type) and print an information message.
- The upper part of (34) can be formulated as:
  $$\mathbf{A}_{ux11}\boldsymbol{v}_{ux1} = -\mathbf{A}_{ux,12}\boldsymbol{v}_{ux2} - \mathbf{B}_{ux1}\boldsymbol{v}_c \tag{35}$$
  Since $\mathbf{A}_{ux11}$ is regular this means that $\boldsymbol{v}_{ux1}$ can be computed from $\boldsymbol{v}_{ux2}$ and $\boldsymbol{v}_c$ and therefore $\boldsymbol{v}_{ux1}$ are (dependent) dummy states. A tool can either utilize this information directly and transform the model equations with this information, or the equations leading to (35), that is the equations of (29) with row indices $p_1[rk + p_{1x}[1{:}rk_x]]$, are replaced by equations (35). Since $\mathbf{A}_{ux11}$ is upper triangular, the constraints between the states are *structurally* visible and therefore index reduction with structural algorithms (see section 3) is possible.

To summarize, with the transformation of the integer equations (29) of DAE (1) to upper trapezoidal form (32),(34) an important set of singularities can be exactly identified and *if possible and desired removed*.

6 Outlook

With this paper new algorithms are provided to start from a high level modeling language like Modelica or Modia and generate code for standard Index-1 DAE integrators. The algorithms are designed to keep array data structures intact from the model language until the generated code. Furthermore, no equation systems are solved to transform to index 1 form and therefore the sparsity of the model equations is kept. As a consequence, sparse matrix methods can be utilized in the DAE integrator.

In the paper it was not discussed how to initialize the index-1 DAEs. In principal similar techniques can be used as for Modelica models. Currently, in Modia it is experimented with a new form of initialization where start values $\mathbf{x}_0(t_0^-)$ can be provided that do *not* fulfill the constraints of (2), so $\mathbf{f}_c(\mathbf{x}_0(t_0^-), t_0^-) \neq 0$. Via Dirac impulses of the derivatives of the discontinuous start values, consistent start values $\mathbf{x}_0(t_0^+)$ are computed with the new technique of impulse handling for DAEs (2), developed in (*Benveniste et al., 2017*).

In industrial applications often steady-state initialization is required. This is still a difficult topic and not yet satisfactorily solved. Typically, reliable steady-state initialization requires the use of a *probability one* homotopy method; see for example (*Melville et. al., 1993*; *Sielemann, 2012*). It is an open question how to restrict the special index-one DAEs (2) so that probability one homotopy methods can be applied.

The goal is to further extend the algorithms and the Modia prototype in order to be able to simulate multi-mode systems, where the number of equations and unknowns can change during simulation (for example to simulate drastic failure cases or perform end-to-end simulations of complicated scenarios).

Acknowledgements

The authors would like to thank Martin Arnold from University Halle-Wittenberg for discussions to understand the fine details of multibody integrators.

References

M. Arnold (2016): *DAE aspects of multibody systems.* In A. Ilchmann, T. Reis (eds.): Surveys in Differential-Algebraic Equations IV. - Springer, 2017 (in print). - A preliminary version of this material was published as Technical Report 01-2016, Martin Luther University Halle-Wittenberg, Report No. 01, 2016. http://sim.mathematik.uni-halle.de/reports/sources/2016/01-2016.pdf

U.M. Ascher, L.R. Petzold (1991): *Projected Implicit Runge–Kutta Methods for Differential-Algebraic Equations*. SIAM J. Numer. Anal., 28(4), pp. 1097–1120.

A. Bahainv, H. Schichl, A. Neumaier (2016a): *Tearing systems of nonlinear equations. I. A survey*. http://www.mat.univie.ac.at/~neum/ms/tearing_survey.pdf

A. Bahainv, H. Schichl, A. Neumaier (2016b): *Tearing systems of nonlinear equations. II. A practical exact algorithm*. http://www.mat.univie.ac.at/~neum/ms/tearing_exact_algorithm.pdf

E.H. Bareiss (1968): *Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination*. Math. Comp. 22, pp. 565-578.

M.A. Bender, J.T. Fineman, S. Gilbert, R.E. Tarjan (2016): *A New Approach to Incremental Cycle Detection and Related Problems*. ACM Transactions on Algorithms, Volume 12, Issue 2.

A. Benveniste, B. Caillaud, H. Elmqvist, K. Ghorbal, M. Otter, and Marc Pouzet (2017): *Multi-Mode DAE Models Challenges, Theory and Implementation*. Lecture Notes on Computer Science, submitted for review.

J. Bezanson, A. Edelman, S. Karpinski and V.B. Shah (2017): *Julia: A Fresh Approach to Numerical Computing*. SIAM Review, Vol. 59, No. 1, pp. 65-98. http://julialang.org/publications/julia-fresh-approach-BEKS.pdf; see also: http://julialang.org/

K.E. Brenan, S.L. Campbell, and L.R. Petzold (1996): *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*. SIAM.

E. Carpanzano, R. Girelli (1997): *The Tearing Problem: Definition, Algorithm and Application to Generate Efficient Computational Code from DAE Systems*. Proceedings of 2nd Mathmod Vienna, IMACS Symposium on Mathematical Modelling, Wien.

S. Chowdhry, H. Krendl, and A.A. Linninger (2004): *Symbolic numeric index analysis algorithm for differential algebraic equations*. Industrial and Engineering Chemistry Research. Vol. 43, Issue 14, pp. 3886-3894.

W. Cook, and A. Rohe (1999): *Computing Minimum-Weight Perfect Matchings*. INFORMS Journal of Computing, Vol. 11. www.math.uwaterloo.ca/~bico/papers/match_ijoc.pdf

I.S. Duff (1981): *On algorithms for obtaining a maximum transversal*. ACM Trans. Math. Software, Vol. 7, Issue 3.

I.S. Duff, K. Kaya, and B. Ucar (2011): *Design, Implementation, and Analysis of Maximum Transversal Algorithms*. ACM Trans. Math. Software, Vol. 38, Issue 2.

J. Edmonds (1965): *Paths, Trees , and Flowers*. Canadian Journal of Mathematics. Vol. 17, pp. 449-467. https://cms.math.ca/openaccess/cjm/v17/cjm1965v17.0449-0467.pdf

E. Eich (1993): *Convergence Results for a Coordinate Projecion Method Applied To Mechanical Systems with Algebraic Constraints*. SIAM J. Numer. Anal. Vol. 30, No. 5, pp. 1467-1482.

H. Elmqvist, M. Otter (1994): *Methods for Tearing Systems of Equations in Object-Oriented Modeling*. Proceedings ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1–3, pp. 326–332.

H. Elmqvist, T. Henningsson, M. Otter (2016): *System Modeling and Programming in a Unified Environment based on Julia*. Proceedings of ISoLA 2016 Conference Oct. 10-14, T. Margaria and B. Steffen (Eds.), Part II, LNCS 9953, pp. 198-217. http://www.isola-conference.org/isola2016/proceedings.html

H. Elmqvist, S.E. Mattsson (2016): *Exploiting Model Graph Analysis for Simplified Modeling and Improved Diagnostics*. Proceedings EOOLT '16, April 18, Milano, Italy.

H. Elmqvist, T. Henningsson, M. Otter (2017): *Innovations for future Modelica*. Modelica Conference 2017, Prague.

J. Frenkel, G. Kunze, P. Fritzson (2012): *Survey of appropriate matching algorithms for large scale systems of differential algebraic equations*. Proceedings of the 9th International Modelica Conference, Munich. http://www.ep.liu.se/ecp/076/045/ecp12076045.pdf

C. Führer (1988): *Differential-algebraische Gleichungssys-teme in mechanischen Mehrkörpersystemen. Theorie, numerische Ansätze und Anwendungen*. PhD thesis, TU München, Mathematisches Institut und Institut für Informatik.

C.W. Gear (1988): *Differential-Algebraic Equation Index Transformations*. SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, pp. 39-47.

C.W. Gear, G.K. Gupta and B. Leimkuhler (1985): *Automatic integration of Euler–Lagrange equations with constraints*. J. Comp. Appl. Math., 12&13, pp. 77–90.

A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C. S. Woodward (2005): *SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers*. ACM Transactions on Mathematical Software, Vol. 31, No. 3, pp. 363–396. http://computation.llnl.gov/projects/sundials/toms_sundials.pdf

G. Kron (1962): *Diakoptics – The piecewise Solution of Large-Scale Systems*. MacDonald & Co., London.

S.E. Mattsson and G. Söderlind (1993): *Index Reduction in Differential-Algebraic Equations using Dummy Derivatives*. SIAM Journal of Scientific Computing. 14(3), pp. 677-692.

S.E. Mattsson, H. Olsson and H. Elmqvist (2000): *Dynamic Selection of States in Dymola*. Modelica Workshop 2000, Lund, Sweden, pp. 61-67. https://www.modelica.org/events/workshop2000/proceedings/old/Mattsson.pdf

R.C. Melville, L. Trajkovic,S.-C. Fang, L.T. Watson (1993): *Artifical parameter homotopy methods for the DC operating point problem*," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 12, pp. 861-877.

M. Otter (1999): *Objektorientierte Modellierung Physi-kalischer Systeme, Teil 4: Transformationsalgorithmen*. at Automatisierungstechnik, 47, 3, pp. A9-A12.

C.C. Pantelides (1988): *The Consistent Initialization of Differential-Algebraic Systems*. SIAM Journal on Scientific and Statistical Computing, 9(2):213–231.

L. Petzold and P. Lötstedt (1986): *Numerical Solution of Nonlinear Differential Equations with Algebraic Constraints II: Practical Implications*. SIAM J. Sci. Stat. Comput. Vol. 7, No. 3, pp. 720-733.

J.D. Pryce (2001). *A Simple Structural Analysis Method for DAEs*. BIT Numerical Mathematics, Vol. 41, Issue 2, pp. 364-394.

J. Schuchart, V. Waurich, M. Flehmig, M. Walther, W.E. Nagel, I. Gubsch (2015): *Exploiting Repeated Structures and Vectorization in Modelica*. Proc. of the 11th Int. Modelica Conference, Versailles. www.ep.liu.se/ecp/118/028/ecp15118265.pdf

M. Sielemann (2012): *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. PhD-Dissertation. Technische Universität Hamburg-Harburg. http://www.dr.hut-verlag.de/9783843905046.html

R.L. Sigurðsson (2016): *Practical performance of incremental topological sorting and cycle detection algorithms*. Chalmers University of Technology. Master Thesis. http://publications.lib.chalmers.se/records/fulltext/248308/248308.pdf

S.S. Skiena (2008): *The Algorithm Design Manual*. Second edition. Springer.

K. Stavaker: *Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures*. PhD thesis. Linköping University. http://liu.diva-portal.org/smash/get/diva2:806837/FULLTEXT01.pdf

P.R. Turner (1995): *A simplified fraction-free Integer Gauss Elimination Algorithm*. REPORT NO: NAWCADPAX-96-196-TR. Office of Naval Research. http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA313755