Symbolic elimination in dynamic optimization based on block-triangular ordering

Journal:	*Optimization Methods and Software*
Manuscript ID	GOMS-2016-0147
Manuscript Type:	Original Article
Date Submitted by the Author:	07-Jul-2016
Complete List of Authors:	Magnusson, Fredrik; Lund University, Department of Automatic Control Åkesson, Johan; Modelon AB,
Keywords:	dynamic optimization, differential-algebraic equations, block-triangular ordering, tearing, sparsity preservation, nonlinear programming, Modelica
2010 Mathematics Subject Classification:	34A09, 49M37

For Peer Review Only

SCHOLARONE™
Manuscripts

To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–30

Symbolic elimination in dynamic optimization based on block-triangular ordering

Fredrik Magnusson[a]* and Johan Åkesson[b]

[a]*Department of Automatic Control, Lund University, SE-221 00 Lund, Sweden*;
[b]*Modelon AB, Ideon Science Park, SE-223 70 Lund, Sweden*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

We consider dynamic optimization problems for systems described by differential-algebraic equations (DAEs). Such problems are usually solved by discretizing the full DAE. We propose techniques to symbolically eliminate many of the algebraic variables in a preprocessing step before discretization. These techniques are inspired by the causalization and tearing techniques often used when solving DAE initial value problems. Since sparsity is crucial for some dynamic optimization methods, we also propose a novel approach to preserving sparsity during this procedure.

The proposed methods have been implemented in the open-source JModelica.org platform. We evaluate the performance of the methods on a suite of optimal control problems solved using direct collocation. We consider both computational time and probability of solving the problem in a timely manner. We demonstrate that the proposed methods often are an order of magnitude faster than the standard way of discretizing the full DAE, and also significantly increase probability of successful convergence.

Keywords: dynamic optimization; differential-algebraic equations; block-triangular ordering; tearing; sparsity preservation; nonlinear programming; Modelica

AMS Subject Classification: 34A09; 49M37

1. Introduction

Dynamic optimization refers to optimization problems with differential equations as constraints. These problems occur in many different fields and contexts, including optimal control, parameter and state estimation, and design optimization. Examples of applications are minimization of material and energy consumption during setpoint transitions in power plants [42] and chemical processes [57], controlling kites for wind power generation [39], estimating occupancy and ambient air flow in buildings [66], and optimal experimental design for estimation of kinetic parameters in fed-batch processes [10].

There are many effective methods available for numerical solution of dynamic optimization problems [14, 58]. Most of these are applicable not only to dynamic systems described by explicit ordinary differential equations (ODE), but also to differential-algebraic equations (DAE). When dynamically simulating DAE systems—that is, solving initial value problems—one can either use specialized DAE solvers [18, 35], or transform the DAE to an equivalent ODE and then apply run-of-the-mill ODE solvers [22]. Key techniques in such transformations are block-triangular orderings to decompose the problem and tearing to reduce the size of the equation systems solved iteratively. While such transformation techniques are commonly applied when simulating DAEs, the conventional approach to solving DAE-constrained optimization problems

*Corresponding author. Email: fredrik.magnusson@control.lth.se

For Peer Review Only

is to discretize the full DAE [15, 16]. In this paper we consider the use of ODE transformation techniques on dynamic optimization problems. But rather than completing the transformation all the way to an explicit ODE, we will try to find the middle ground between the full implicit DAE and the underlying explicit ODE that maximizes computational efficiency and convergence robustness. We build upon previous work by the authors [47] and extend it with tearing, sparsity preservation and a thorough performance benchmark.

While these techniques can be applied in combination with any of the standard numerical methods for solving DAE-constrained optimization problems, the performance of some methods will be affected more than others. In this paper we focus on the use of direct collocation and how it is affected by these transformation techniques. The effect of these techniques is also dependent on the considered DAE. We will show the effect of symbolic transformation techniques on computational speed and convergence robustness for six different large-scale benchmark problems with different properties from various physical domains. The presented methods have been implemented in JModelica.org[1] [3], which is an open-source tool for analysis of dynamic systems described by the object-oriented, equation-based modelling language Modelica [32]. Modelica models tend to be large, sparse, and have considerably more algebraic than differential variables, making them a prime target for the considered methods. We will however show that the presented methods hold merit also for more typical textbook DAEs.

The contributions of this paper are the adaptation of causalization and tearing techniques to direct collocation for dynamic optimization and the development of a novel sparsity preservation technique. We also present a thorough benchmark of these techniques, showing how they improve both computational speed and convergence robustness.

The outline of the paper is as follows. Section 2 goes through a simple and illuminating example of how the ideas described in the paper are applied. Section 3 presents the theoretical background of dynamic optimization and the symbolic techniques we will employ, as well as JModelica.org and other related work. Section 4 presents how these techniques can be adapted to symbolically eliminate variables in dynamic optimization problems while preserving sparsity. Section 5 compares the effects of using and not using these techniques on six benchmark problems. Finally, Section 6 summarizes the paper and discusses potential future work.

Regarding notation, scalars and scalar-valued functions are denoted by regular italic letters x. Vectors and vector-valued functions are denoted by bold italic letters $\boldsymbol{x}$. The dimension of $\boldsymbol{x}$ is denoted by n_x and component k of $\boldsymbol{x}$ is denoted by x_k. Matrices are denoted by bold Roman letters $\mathbf{A}$ and element (i, j) of $\mathbf{A}$ is denoted by $A_{i,j}$. Row i and column j of $\mathbf{A}$ are denoted by $\boldsymbol{A}_{i,:}$ and $\boldsymbol{A}_{:,j}$, respectively. Systems of equations $\boldsymbol{f} = \mathbf{0}$ that are to be considered as being parametrized by $\boldsymbol{p}$ and having $\boldsymbol{x}$ as unknowns are denoted by $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}) = \mathbf{0}$.

2. Illustrative example

To demonstrate the main ideas of the symbolic eliminations we will employ, we will go through a simple example. If some steps are unclear to the reader, they should be revisited after Section 3.

[1] http://www.jmodelica.org

For Peer Review Only

Consider the simple time-invariant DAE

$$\dot{x} + y_1 + y_2 - y_3 = 0, \tag{1a}$$
$$xy_3 + y_2 - \sqrt{x} - 2 = 0, \tag{1b}$$
$$2y_1y_2y_4 - \sqrt{x} = 0, \tag{1c}$$
$$y_1y_4 + \sqrt{y_3} - x - y_4 = 0, \tag{1d}$$
$$y_4 - \sqrt{y_5} = 0, \tag{1e}$$
$$y_5^2 - x = 0. \tag{1f}$$

A structural incidence matrix for (1) is (2).

	$\dot{x}$	y_1	y_2	y_3	y_4	y_5
(1a)	1	1	1	1	0	0
(1b)	0	0	1	1	0	0
(1c)	0	$*$	$*$	0	$*$	0
(1d)	0	$*$	0	$*$	$*$	0
(1e)	0	0	0	0	1	$*$
(1f)	0	0	0	0	0	$*$

(2)

where zeroes indicate independence, ones indicate linear dependence, and asterisks indicate nonlinear dependence. Inspection of (1) reveals that we can eliminate y_4 using (1e). That is, we use

$$y_4 = \sqrt{y_5} \tag{3}$$

to substitute y_4 with $\sqrt{y_5}$, yielding

$$\dot{x} + y_1 + y_2 - y_3 = 0, \tag{4a}$$
$$xy_3 + y_2 - \sqrt{x} - 2 = 0, \tag{4b}$$
$$2y_1y_2\sqrt{y_5} - \sqrt{x} = 0, \tag{4c}$$
$$y_1\sqrt{y_5} + \sqrt{y_3} - x - \sqrt{y_5} = 0, \tag{4d}$$
$$y_5^2 - x = 0. \tag{4e}$$

Such eliminations can be identified by permuting (2) to a block-triangular matrix with blocks along the diagonal—henceforth referred to as diagonal blocks, despite them not being diagonal in general—of minimal size. Permuting (2) to such a form yields (5).

	y_5	y_4	y_1	y_2	y_3	$\dot{x}$
(1f)	$*$	0	0	0	0	0
(1e)	$*$	1	0	0	0	0
(1b)	0	0	0	1	1	0
(1c)	0	$*$	$*$	$*$	0	0
(1d)	0	$*$	1	0	$*$	0
(1a)	0	0	$*$	$*$	$*$	1

(5)

Note that some previously linear incidences have been reclassified as nonlinear and vice versa. We are no longer interested in the linearity of incidences outside of the diagonal blocks, and

For Peer Review Only

thus mark them all with asterisks. Within the diagonal blocks, we only care about linearity with respect to the variables in the block. The incidence of the equation-variable pair $((1d), y_1)$ that was previously considered to be nonlinear is thus now considered to be linear. An elimination such as (3) can be found by identifying the diagonal blocks that are scalar and linear. The only algebraic variable that can be eliminated with this approach in this example is y_4.

We can however go further in our elimination procedure by eliminating y_1 and y_2 from (1d) and (1b), respectively. That is, we identify

$$y_1 = \frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}}, \tag{6a}$$

$$y_2 = 2 + \sqrt{x} - xy_3, \tag{6b}$$

and through substitution obtain the DAE

$$\dot{x} + \frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}} + 2 + \sqrt{x} - xy_3 - y_3 = 0, \tag{7a}$$

$$2\frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}}(2 + \sqrt{x} - xy_3)\sqrt{y_5} - \sqrt{x} = 0, \tag{7b}$$

$$y_5^2 - x = 0. \tag{7c}$$

The eliminations (6) can be found by tearing the 3×3 diagonal block. Selecting y_3 as tearing variable and (1c) as tearing residual, we get the torn incidence matrix (8).

	y_5	y_4	y_1	y_2	y_3	$\dot{x}$
(1f)	*	0	0	0	0	0
(1e)	*	1	0	0	0	0
(1d)	0	*	1	0	*	0
(1b)	0	0	0	1	1	0
(1c)	0	*	*	*	0	0
(1a)	0	0	*	*	*	1

(8)

The eliminations (6) are then identified as the diagonal incidences in the upper left subblock of the torn block, which are feasible since this subblock is triangular (in this case it is even diagonal) and linear along the diagonal. Comparing (7) with (1), we have only 2 instead of 5 algebraic variables, but the residuals are more complicated.

3. Background

In this section we first present the class of DAEs and dynamic optimization problems that we consider. We then describe the open-source tool JModelica.org and in particular how it solves these problems. We review how ODE transformation techniques are employed when solving DAE simulation problems, in the context of which they are often referred to as causalization. We also describe well-established techniques for selecting pivot elements in direct, sparse linear solvers, which we later will utilize for sparsity preservation. Finally, we discuss related work.

For Peer Review Only

3.1 *Differential-algebraic equations*

We consider systems whose dynamics are described by an implicit DAE system. Specifically, we consider DAE systems of the form

$$\boldsymbol{\Phi}(t, \dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{\upsilon}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, \tag{9}$$

where t is time—the sole independent variable—$\boldsymbol{\xi}$ is the differential variable, $\boldsymbol{\upsilon}$ is the algebraic variable, $\boldsymbol{u}$ is the control variable, and $\boldsymbol{p}$ is the parameters to be optimized. We assume that the system is balanced; that is, that the dimension of the codomain of $\boldsymbol{\Phi}$ equals the sum of the dimensions of $\boldsymbol{\xi}$ and $\boldsymbol{\upsilon}$.

To deal with high index DAEs, we utilize JModelica.org's index reduction, which is based on the method of dummy derivatives [50] and reduces the index of the DAE to at most 1. Index reduction can be done in different ways, usually involving the differentiation of the high-index (index 2 or higher) algebraic equations, which can be identified by Pantelides's algorithm [54]. This differentiation can lead to drift during numerical integration, which can be remedied through the use of for example constraint stabilization [11] or projections [20]. Such techniques are however typically designed for special classes of DAEs. JModelica.org instead uses the more widely applicable method of dummy derivatives [50] to avoid drift, which retains both the original and differentiated equations and introduces new variables called dummy derivatives to obtain a balanced system. This method however requires the selection of state variables, which is difficult to automate. In general it is not even possible to preserve solvability with a static choice of state variables [49], which however is out of scope for this paper.

We thus assume that there exists a static choice of state variables $\boldsymbol{x}$ out of the differential variables $\boldsymbol{\xi}$ that does not give rise to solvability issues, in particular implying that the local index of the DAE [18] is constant along all feasible trajectories. The index reduction then creates an equivalent—in the sense that there exists a bijection between the solution sets—low-index DAE

$$\boldsymbol{F}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, \tag{10}$$

where $\boldsymbol{y}$ is the algebraic variables, consisting of the original algebraic variables $\boldsymbol{\upsilon}$, those differential variables $\boldsymbol{\xi}$ not selected as states, and the newly created dummy derivatives. We also assume that (feasible) initial conditions $\boldsymbol{x}(0) = \boldsymbol{x}_0$ are given and that $\boldsymbol{F}$ is twice continuously differentiable with respect to all of its arguments except the first.

3.2 *Dynamic optimization*

The problem considered throughout the paper is to

$$\begin{aligned}
&\text{minimize} && \phi(t_f, \boldsymbol{\xi}(t_f), \boldsymbol{\upsilon}(t_f), \boldsymbol{p}) + \int_0^{t_f} \ell(\boldsymbol{\xi}(t), \boldsymbol{\upsilon}(t), \boldsymbol{u}(t))\,\mathrm{d}t, && \text{(11a)}\\
&\text{with respect to} && \boldsymbol{\xi} : [0, t_f] \to \mathbb{R}^{n_\xi}, \quad \boldsymbol{\upsilon} : [0, t_f] \to \mathbb{R}^{n_\upsilon}, \quad \boldsymbol{u} : [0, t_f] \to \mathbb{R}^{n_u},\\
& && t_f \in \mathbb{R}, \quad \boldsymbol{p} \in \mathbb{R}^{n_p},\\
&\text{subject to} && \boldsymbol{\Phi}(t, \dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{\upsilon}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, && \text{(11b)}\\
& && \boldsymbol{L} \le (\dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{\upsilon}(t), \boldsymbol{u}(t), \boldsymbol{p}) \le \boldsymbol{U}, \quad \boldsymbol{g}(t, \dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{\upsilon}(t), \boldsymbol{u}(t), \boldsymbol{p}) \le \boldsymbol{0}, && \text{(11c)}\\
& && \boldsymbol{G}(\boldsymbol{\xi}(t_f), \boldsymbol{\upsilon}(t_f), \boldsymbol{p}) \le \boldsymbol{0}, \quad 0 \le t_{f,L} \le t_f \le t_{f,U}, && \text{(11d)}\\
& && \forall t \in [0, t_f],
\end{aligned}$$

For Peer Review Only

where the objective (11a) comprises the Mayer term ϕ and Lagrange integrand ℓ, (11b) is the system dynamics, (11c) is inequality constraints (where box constraints $\boldsymbol{L}$ and $\boldsymbol{U}$ are separated from the general nonlinear equalities $\boldsymbol{g}$), and (11d) is the terminal constraints $\boldsymbol{G}$ and nonnegative bounds on t_f. The bounds $\boldsymbol{L}$ and $\boldsymbol{U}$ can be infinite.

The problem will in general be nonconvex and we will not endeavour to find a global optimum. We will instead rely on first-order necessary optimality conditions to find a local optimum. Just like for $\boldsymbol{F}$, we assume that ϕ, ℓ, $\boldsymbol{g}$, and $\boldsymbol{G}$ are twice continuously differentiable in order to apply techniques based on Newton's method to find a solution to first-order optimality conditions.

The methods and software framework we present in this paper can be applied on the wider class of problems that is considered by Magnusson and Åkesson [46]. However, (11) captures the essentials and the generalization of the presented methods to the wider class is straightforward, and so we restrict ourselves to (11) for the sake of brevity.

We mainly consider the use of direct collocation to solve (11). The main idea of direct collocation is to discretize (11) by dividing the time horizon into a finite number of elements and then within each element approximate the system trajectories by polynomials defined by collocation points [15, 16, 46]. The result is a nonlinear program (NLP) whose solution approximates the solution of (11). This approach leads to the full DAE being exposed to the NLP solver. This is in contrast to sequential approaches based on shooting. In the case of single shooting, only the degrees of freedom, $\boldsymbol{u}$ and $\boldsymbol{p}$, are exposed to the solver and the DAE as well as $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$ are hidden in embedded calls to numerical integrators in the constraints. Multiple shooting also introduces $\boldsymbol{x}$ as variables in the shooting nodes, and possibly also $\boldsymbol{y}$, but the DAE (or at least parts of it) are still hidden in embedded integrator calls. The methods discussed in this paper are thus more relevant for simultaneous approaches, such as direct collocation, where they have a major impact on the size and structure of the NLP, whereas they often only affect the NLP function evaluation times when using sequential approaches. Nevertheless, these methods certainly have the potential to be beneficial also when using sequential approaches, since the DAE simulation in shooting methods is often the computational bottleneck and a source of numerical problems.

3.3 *Object-oriented acausal modelling*

The classical equation-based approach to modelling dynamic systems is that of causal, block-oriented modelling, which is used in general-purposes tools such as Simulink. The mathematical formalism is that of explicit ordinary differential equations, which facilitates efficient numerical solution, but may be inconvenient for modelling systems that are more naturally described by DAEs. Furthermore, causal modelling inhibits model reuse due to the need of defining causal input-output connections between components.

A more modern—but by now well-established—approach to modelling is the use of acausal, declarative relations, which is used in tools such as SPICE [52] and ASCEND [56]. However, most of these tools are domain-specific, focusing on e.g. electrical circuits or chemical processes. A more general-purpose language for acausal modelling is Modelica [32], which is a standardized language for which several different tools exist. Modelica is a textual language, but the most common use of it is through graphical connection of library components. An example of this is shown in Figure 1, which is the block diagram for a model of a four-bar system, which is the basis of one of the benchmark problems in Section 5.

While acausal modelling is convenient for the modeller, the resulting mathematical model is often unwieldy in its raw form and requires symbolic processing before it can be simulated efficiently using numerical methods. This holds especially true when the model is the result of object-oriented aggregation of many subcomponents, which gives rise to a multitude of variables and equations through connection equations. This challenge is further exacerbated when the used

For Peer Review Only

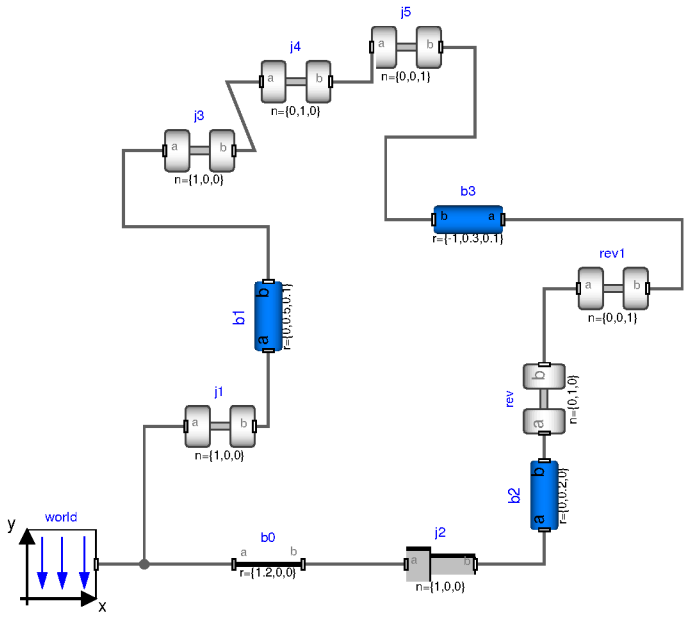


Figure 1. Object diagram of a four-bar system in Modelica. The kinematic loop is easily constructed by graphically connecting components from the Modelica Standard Library for rigid bodies as well as revolute and prismatic joints.

components have been developed for general purposes, requiring the computation of quantities which may hold no relevance for a particular application.

In Section 3.5 we review techniques that often are used to alleviate these issues. While these techniques often are crucial for handling object-oriented, acausal models, we will in Section 5 see that they hold merit also when applied to examples of specialized, non-hierarchical models in the context of dynamic optimization.

3.4 *JModelica.org*

JModelica.org [3] is one of the tools designed for the Modelica language. It is open source and historically has a strong focus on dynamic optimization with its support for the Modelica extension Optimica [2] and an efficient dynamic optimization framework [46]. Today it also has a strong support for FMI [17], allowing inter-tool exchange of models for dynamic simulation. The dynamic optimization framework in JModelica.org leverages the JModelica.org compiler to first perform structural analysis and some symbolic transformations and then transfer the resulting flattened problem to CasADi Interface [44]. The symbolic representation of CasADi Interface—which is based on CasADi [4] for efficient computation of derivatives using algorithmic differentiation—is then connected to algorithms based on direct collocation for solving dynamic optimization problems by transcribing them into NLPs. The NLPs are then finally solved by IPOPT [65] or WORHP [19].

It is further worth noting that while the overarching ideas of this paper are widely applicable, some of the detailed algorithmic choices discussed in Section 4 have been adapted to the dynamic optimization framework of JModelica.org, that is, the use of IPOPT, CasADi and the collocation algorithm in JModelica.org [46].

3.5 *Causalization*

The process of transforming an implicit DAE

$$\boldsymbol{\Phi}(t, \dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0} \tag{12}$$

to an equivalent explicit ODE

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) \tag{13}$$

is called causalization [22]. We divide this procedure into the following steps.

(1) Alias elimination
(2) Variability propagation
(3) Index reduction
(4) Matching
(5) Block-lower triangular ordering
(6) Tearing

Alias elimination and variability propagation are conceptually simple and serve to eliminate algebraic variables described by trivial equations. Alias elimination identifies variables occurring in equations of the form $x \pm y = 0$, which are ubiquitous in object-oriented acausal modelling, and eliminates one of them. Variability propagation identifies algebraic equations which are independent of time, such as $y = p + 1$, where p is a parameter, allowing the corresponding algebraic variables to be eliminated by solving the static equations.

Conceptually, an explicit ODE allows the computation of $\dot{\boldsymbol{x}}(t)$ given the current known variables $t, \boldsymbol{x}(t), \boldsymbol{u}(t)$, and $\boldsymbol{p}$. After having performed index reduction as discussed in Section 3.1, the remaining steps of the ODE transformation problem can thus be considered equivalent to solving the square system

$$\boldsymbol{F}(\boldsymbol{z}; t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) = 0, \tag{14}$$

where $\boldsymbol{z} = (\dot{\boldsymbol{x}}, \boldsymbol{y})$. Note that we have now dropped the dependence on time for $\dot{\boldsymbol{x}}, \boldsymbol{x}$, and $\boldsymbol{u}$, since we with this perspective only consider them as elements of $\mathbb{R}^n$ rather than functions.

While solving (14) can be done in a straightforward numerical manner by applying e.g. Newton's method[2], such an approach may be inefficient and may also be practically challenging due to the need of a having a sufficiently good initial guess of the solution for convergence. The idea of causalization and this paper is to apply symbolic transformations to the problem to allow for more efficient subsequent numerical solution.

The next step is to find a perfect matching between each scalar component of $\boldsymbol{F}$ and each scalar component of $\boldsymbol{z}$, meaning that each equation is matched to a single variable and vice versa. Such a matching exists if and only if the DAE is low index. Finding perfect matchings is a well-studied graph theoretical problem with several available efficient algorithms. JModelica.org uses the Hopcroft-Karp [37] algorithm, which has the best known worst case performance and often performs well in practice.

The two final steps, block-lower triangular (BLT) ordering and tearing, are discussed below.

[2]It is dubious whether the result really can be considered to be an explicit ODE, since a closed-form expression for $\boldsymbol{f}$ has not been found (in fact, one may not even exist). However, typical numerical ODE solvers will not be able to tell the difference as long as $\boldsymbol{f}$ can be evaluated.

For Peer Review Only

3.5.1 *Block-lower triangular ordering*

Consider the structural incidence matrix of the DAE, whose elements are defined by

$$\operatorname{struct} F_{i,j} = \begin{cases} 0 \text{ or } *, & \text{if } \nabla_{z_j} F_i \equiv 0, \\ 1 \text{ or } *, & \text{if } \nabla_{z_j} F_i \not\equiv 0 \text{ and } \nabla^2_{z,z_j} F_i \equiv \mathbf{0}, \\ *, & \text{otherwise,} \end{cases} \tag{15}$$

that is, 0 denotes incidences that do not depend on unknowns, 1 denotes incidences that depend affinely—henceforth called linearly—on unknowns, and $*$ denotes any kind of incidence. The main step of the causalization procedure is to permute the DAE structural incidence matrix to a BLT form. The result is that the equations and variables of the DAE have been sorted so that the DAE can be described by

$$\boldsymbol{F}^1(z^1; \boldsymbol{v}^1) = \mathbf{0}, \tag{16a}$$
$$\boldsymbol{F}^2(z^2; \boldsymbol{v}^2) = \mathbf{0}, \tag{16b}$$
$$\vdots$$
$$\boldsymbol{F}^m(z^m; \boldsymbol{v}^m) = \mathbf{0}, \tag{16c}$$

where $\boldsymbol{F}^i$ corresponds to a diagonal block (colloquially known as an algebraic loop if it is not scalar valued), m is the number of such blocks, z^i is the unknown variables of $\boldsymbol{F}^i$, and

$$\boldsymbol{v}^i := (z^1, z^2, \ldots, z^{i-1}, t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \tag{17}$$

is the known variables and unknown variables of preceding blocks. In other words, we have found permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ such that

$$\mathbf{P} \operatorname{struct}(\boldsymbol{F}) \mathbf{Q} = \begin{bmatrix} \mathbf{F}^{1,1} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{F}^{2,1} & \mathbf{F}^{2,2} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{F}^{3,1} & \mathbf{F}^{3,2} & \mathbf{F}^{3,3} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{F}^{m,1} & \mathbf{F}^{m,2} & \mathbf{F}^{m,3} & \cdots & \mathbf{F}^{m,m} \end{bmatrix}, \tag{18}$$

where $\mathbf{F}^{i,j} := \operatorname{struct} \boldsymbol{F}^i(z^j; z^1, \ldots, z^{j-1}, z^{j+1}, \ldots, z^i, t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$.

This form allows the sequential treatment of each diagonal block, allowing us to solve multiple small systems rather than a single large. Furthermore, it enables specialized treatment of each diagonal block, allowing the exploitation of equation structure that may exist within a diagonal block but not in the full system $\boldsymbol{F}$, such as linearity.

We are thus interested in the BLT form that has the maximal number of diagonal blocks, or equivalently, diagonal blocks of minimal size. This form turns out to be unique in the sense that the number of diagonal blocks and their respective sizes and the variables and equations within a diagonal block are unique, but the ordering of diagonal blocks and orderings of variables and equations within the blocks are in general not unique. A trivial consequence of this form is that the diagonal blocks are irreducible.

Finding a BLT ordering that is optimal in this sense is equivalent to finding the strongly connected components of the directed graph defined by the matching found in step 4 as follows. For each matched equation-variable pair, create a vertex. For each non-zero struct $F_{i,j}$, create an

For Peer Review Only

edge from the vertex corresponding to equation i to the vertex corresponding to variable j. This will create loops on all of the vertices, which may be removed.

Tarjan's algorithm [61] is widely regarded as the most efficient algorithm for finding the strongly connected components of a directed graph, with linear complexity in the number of vertices and edges. It also has the added benefit of not only identifying the strongly connected components, but also topologically sorting them so that all edges out of a component lead to preceding components, giving us the sought BLT ordering.

Block-triangular orderings are useful in contexts other than DAE systems, such as solving sparse, unsymmetric linear equations [28].

3.5.2 *Tearing*

All that remains in the computation of $\dot{\boldsymbol{x}}$ is the solution of each $\boldsymbol{F}^i$. In general this will require iterative numerical methods. In this paper we will however not proceed down this route at this stage. We will instead only rely on symbolic techniques, without involving iterative methods, in our computation of $\dot{\boldsymbol{x}}$. Consequently, we will not go all the way to an explicit ODE. The details of this are discussed in Section 4. An important part of this is the use of tearing [8, 27, 29, 41]. Tearing is a method for solving sparse systems of equations that lack significant structure, such as being triangular or block diagonal, by ordering the variables and equations to get a partitioning of the system

$$\mathbf{0} = \boldsymbol{F}^i(\boldsymbol{z}^i; \boldsymbol{v}^i) = \begin{bmatrix} \bar{\boldsymbol{F}}^i(\bar{\boldsymbol{z}}^i; \hat{\boldsymbol{z}}^i, \boldsymbol{v}^i) \\ \hat{\boldsymbol{F}}^i(\hat{\boldsymbol{z}}^i; \bar{\boldsymbol{z}}^i, \boldsymbol{v}^i) \end{bmatrix} \tag{19}$$

such that the first partition $\bar{\boldsymbol{F}}^i$ is highly structured, allowing this part of the system to be solved efficiently on its own and then utilized in the solution of the full system by block elimination.

Tearing can be done in different ways, for different applications, with different goals in mind. In this paper we apply tearing to the diagonal blocks of the BLT form. We seek a partitioning such that one part of the diagonal block is triangular and linear along the diagonal. That is, we find permutations $\mathbf{P}^i$ and $\mathbf{Q}^i$ so that

$$\mathbf{P}^i \operatorname{struct}\left(\boldsymbol{F}^i\right) \mathbf{Q}^i = \begin{array}{c} \begin{matrix} \bar{\boldsymbol{z}}^i & \hat{\boldsymbol{z}}^i \end{matrix} \\ \left[\begin{array}{cc|c} 1 & & \\ & \mathbf{0} & * \\ * & \ddots & \\ & 1 & \\ \hline * & & * \end{array}\right] \end{array} \begin{matrix} \\ \\ \\ \\ \\ \bar{\boldsymbol{F}}^i \\ \hat{\boldsymbol{F}}^i \end{matrix}, \tag{20}$$

where $\bar{\boldsymbol{z}}^i$ is called the causalized variables, $\bar{\boldsymbol{F}}^i$ the causalized equations (due to the lower-triangular structure), $\hat{\boldsymbol{z}}^i$ the tearing variables, and $\hat{\boldsymbol{F}}^i$ the tearing residuals. This form is appealing because it allows for symbolic elimination of the causalized variables in terms of the tearing variables; that is,

$$\bar{\boldsymbol{z}}^i = \left(\bar{\boldsymbol{F}}^i\right)^{-1}(\mathbf{0}; \hat{\boldsymbol{z}}^i, \boldsymbol{v}^i) =: \bar{\boldsymbol{H}}^i(\hat{\boldsymbol{z}}^i, \boldsymbol{v}^i). \tag{21}$$

For Peer Review Only

The fact that $\bar{\boldsymbol{F}}^i$ is triangular and linear along the diagonal allows for efficient and numerically stable computation of $\left(\bar{\boldsymbol{F}}^i\right)^{-1}$ through forward substitution. The remaining equations to be solved for $\hat{\boldsymbol{z}}^i$ numerically, typically using Newton's method, are given by

$$\bar{\boldsymbol{H}}^i(\hat{\boldsymbol{z}}^i) := \hat{\boldsymbol{F}}^i\left(\hat{\boldsymbol{z}}^i; \bar{\boldsymbol{H}}(\hat{\boldsymbol{z}}^i, \boldsymbol{v}^i), \boldsymbol{v}^i\right) = \boldsymbol{0}. \tag{22}$$

The benefits of this approach over solving the full diagonal block (19) numerically is that (22) has fewer variables and equations, equal to the number of tearing variables, often allowing it to be solved more efficiently. But a perhaps more important benefit is that a sufficiently good initial guess for convergence is only needed for $\hat{\boldsymbol{z}}^i$, whereas solving the full system would also require a sufficiently good guess for $\bar{\boldsymbol{z}}^i$. For these reasons, it is desirable to find a partitioning that minimizes the number of tearing variables and residuals. Unfortunately, this problem is NP-hard [8]. While in many cases it is tractable to solve the problem to optimality, it is common to instead apply heuristics to find near-optimal partitionings. The tearing algorithm used in JModelica.org, and in this paper, is based on the heuristics described by Meijer [51].

The use of tearing is however a double-edged sword. While the symbolic solution of (21) and numerical solution of (22) are numerically stable, the tearing residuals (22) may be ill-conditioned even if the full system (19) is not. Furthermore, even if the tearing residuals and full system are well-conditioned, the numerical computation of $\bar{\boldsymbol{z}}^i$ through (21) after having computed the solution of (22) may be numerically unstable. Another potential drawback is that (22) is significantly more dense than (19), which may cause it to actually be more expensive to solve if sparsity is exploited in the computations. This topic is discussed further below. In conclusion, there is more to the choice of tearing variables and residuals than just minimizing their number.

3.6 *Pivot selection in direct, sparse linear solvers*

When solving a nonlinear system of equations with Newton's method, the symbolic elimination of variables is computationally equivalent to a priori selection of pivot elements in the linear equation solver in each Newton iteration. Since pivots are selected based on numerical values to prevent numerical instability (caused by error growth due to numerical round-off), a priori selection based entirely on structural, as opposed to numerical, information can lead to numerical instability, as mentioned in Section 3.5.2 and discussed by Duff et al. [27].

Another potential issue of blindly eliminating variables is that the reduced system may be significantly more dense. As opposed to dense direct linear solvers, sparse direct linear solvers do not only select pivots to minimize error growth, but also to minimize fill-in. The typical approach is to select as pivot element the element that causes the least amount of estimated fill-in in the matrix factors while also being bigger (in magnitude) than a fraction of the largest element in the same column (partial pivoting). The reason that only an estimate of the fill-in is used is due to the computational intractability of computing the fill-in caused on a global level (the final matrix factors). The most widely used estimate of fill-in is the Markowitz criterion [48]. When LU-factorizing a (sub)matrix $\mathbf{M}$, the Markowitz criterion selects as the next pivot the element $M_{i,j}$ that minimizes

$$(\operatorname{nnz} \boldsymbol{M}_{i,:} - 1) \cdot (\operatorname{nnz} \boldsymbol{M}_{:,j} - 1), \tag{23}$$

typically out of those elements satisfying

$$|M_{i,j}| \geq u \max_l |M_{l,j}|, \tag{24}$$

For Peer Review Only

where $0 < u \leq 1$ is the pivot tolerance. Another useful estimate is that of local minimum fill-in, also proposed by Markowitz [48], which is a better but more expensive estimate. The difference is that local minimum fill-in computes the actual fill-in in each stage, whereas the Markowitz criterion estimates the fill-in by the amount of incidences in the pivot row and column, i.e., it does not take into account that some of the incidences already occur in the remaining equations and thus do not cause additional fill-in. The sparsity-preservation techniques of Section 4.3 are related to these ideas.

3.7 *Related work*

The standard approach of discretizing the full DAE relies on fill-reducing orderings, such as nested dissection [33], for efficient numerical linear algebra and numerical pivoting for stability. While our proposed approach of using causalization techniques has been used in the context of dynamic optimization before [7, 31], in this paper we consider the effects they have on the solution procedure. We also do not complete the causalization all the way to an explicit ODE, but rather choose to keep some algebraic variables and implicit equations for efficiency.

Safdarnejad et. al [60] consider the use of BLT decompositions for the purpose of more efficiently generating initial guesses to (11) by solving square problems (with fixed degrees of freedom), and in particular identifying infeasibilities. They do however not consider BLT decompositions for the actual solution of (11). Fletcher [30] considers the use of block-triangular ordering and tearing for efficient and sparsity-preserving orderings for implicit LU factorization tailored for linear programming.

The main purpose of causalization is the efficient treatment of algebraic equations. There are other approaches to achieving this. One technique that is common when using direct multiple shooting is the elimination of all algebraic variables in the shooting nodes by linearizing the consistency conditions (algebraic equations) in each iteration, which is possible for semi-explicit index-one DAEs [25]. Another related method is the use of reduced space methods [23] which primarily work in the null space of the NLP equality constraints, which can be beneficial when there are few degrees of freedom, as is typically the case in dynamic optimization problems. Since these techniques are applied to the NLP rather than the dynamic problem (11), they could be combined with the ideas presented in this paper with potential benefits, but such possibilities are not considered further in this paper.

Another possibility when employing direct local collocation is the use of parallelization by exploiting the arrowhead structure of the KKT system that arises due to the temporal decoupling of the finite elements [64]. This has great potential speedups when there are significantly more algebraic than differential variables, a sufficiently large amount of finite elements is used, and a large number of processor cores are available. Unlike the other approaches outlined in this section, this approach is ill-suited to be combined with the ideas of this paper, as its efficiency is dependent on there being a large amount of algebraic variables exposed to the collocation discretization.

4. Symbolic elimination for dynamic optimization

In Section 3.5 we reviewed how implicit DAEs can be transformed to explicit ODEs. While almost all of the steps were symbolic in nature, in the end we were left with the algebraic loops $\boldsymbol{F}^i = 0$ (which can be further symbolically reduced through the use of tearing), which in general will require numerical iterative methods to solve. However, going all the way to an explicit ODE is not necessary when applying direct collocation in the context of dynamic optimization, as collocation methods are perfectly capable of dealing with low-index (and to some extent also

For Peer Review Only

high-index) implicit DAEs. In this section we describe how the causalization techniques can be applied to (11) to symbolically eliminate many of the algebraic variables.

We will describe various approaches to symbolic elimination which can be combined in different ways, giving rise to different *schemes* of symbolic elimination. In Section 5 we will compare the performance of these schemes to each other, and also to the trivial elimination scheme of not performing any eliminations at all, which can be considered to be the standard approach when employing direct collocation to dynamic optimization problems.

SCHEME 0 *Do not eliminate any algebraic variables.*

4.1 *Elimination in scalar, linear diagonal blocks*

We start by applying the first 5 steps of the causalization procedure of Section 3.5 to the DAE in (11b) in the standard manner, and postpone the discussion on the use of tearing. We next identify the diagonal blocks $\boldsymbol{F}^i$ of the BLT form that are both scalar and linear; that is, struct $\boldsymbol{F}^i = 1$. The corresponding variables $\boldsymbol{z}^i$ of these blocks, which are scalar, are considered for elimination. By elimination we mean that all occurrences of $\boldsymbol{z}^i$ in (11) are substituted by $\left(\boldsymbol{F}^i\right)^{-1}(\boldsymbol{0}; \boldsymbol{v}^i)$ using (16). Note that the computation of $\left(\boldsymbol{F}^i\right)^{-1}$ in this case is just division by a single, scalar closed-form expression[3]. After this elimination step, the variable $\boldsymbol{z}^i$ and the equation to which it has been matched are removed from the problem.

In a first effort, we choose to eliminate all such variables (whose matched equations are scalar and linear with respect to the variable) that are unbounded, that is, those for which the corresponding elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ are $-\infty$ and $+\infty$, respectively. No other variables are eliminated.

SCHEME 1 *Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, and whose incidence in the diagonal block is linear.*

A consequence of eliminating bounded variables is that the linear bound is transformed into a nonlinear inequality constraint (it is moved from the left part of (11c) to the right part). When using IPOPT to solve the discretized version of (11), there is no obvious benefit in eliminating bounded variables. This is because IPOPT transforms each nonlinear inequality constraint into a nonlinear equality constraint and a variable bound, through the introduction of a slack variable. So even if we eliminate a bounded variable, IPOPT would essentially undo the elimination, leading to no difference in the dimension of the KKT system that is solved. But while the size of the KKT system is unaffected, the structure is not. One situation when the change in structure is potentially harmful is when the NLP functions are undefined if the algebraic variable bound is not satisfied. Since IPOPT iterates always are feasible with respect to the variable bounds, but not necessarily the nonlinear inequality constraints, eliminating bounded algebraic variables can thus cause evaluation errors which otherwise would not have occurred. For these reasons, we do not consider the elimination of bounded variables.

There is a problem with the approach of only eliminating unbounded variables, since in Optimica, many of the variables are bounded. This is because bounds are defined by the `min` and `max` Modelica attributes, which variables inherit from their types. For example, all variables declared as temperatures are bounded below by 0 [K], even though these bounds will never be active at feasible points. Because of this, we force the user to specify which variables are expected to be actively bounded, and consider all other variables as unbounded for the purpose of elimination, even if they have finite `min` and `max` values.

[3]In the general case, care is needed to ensure that this scalar expression is non-zero [9, 51]. However, since this is always the case for the problems we consider in Section 5, we henceforth ignore this issue for the sake of simplicity.

For Peer Review Only

We only consider the elimination of algebraic variables, and not differential variables. While the techniques we use to find closed-form expressions for computing algebraic variables also can be used to find closed-form expressions for the derivatives of differential variables (which indeed is what is done when doing full causalization to obtain an explicit ODE), finding these expressions is not sufficient for the elimination of the derivatives of differential variables. In order to eliminate the derivatives of differential variables, we would also have to eliminate the corresponding differential variables, which would require the solution of differential equations. Finding closed-form solutions to differential equations is rarely possible for the cases of interest, so this possibility is not considered further in this paper.

There is however another potential use of the closed-form expressions for the derivatives of the differential variables without eliminating them. All occurrences of a differential variable derivative, except the ones in the matched diagonal block equations, can be substituted by the closed-form expression. While this will never affect NLP size, it will affect the NLP sparsity in a way that depends on the simultaneous discretization method. The collocation algorithm in JModelica.org creates NLP variables for all differential variable derivatives. So from a sparsity perspective it is never good to substitute the differential variable derivatives in JModelica.org. However, another common approach to direct collocation is to eliminate the differential variable derivatives by using the collocation equations. In this case it is not obvious whether such substitutions would be beneficial. Comparing the number of incidences from a sparsity perspective would be easy enough, but since the closed-form expressions found via the causalization tend to be highly nonlinear whereas the collocation equations are linear, a pure sparsity perspective is probably too narrow to be useful. But since this is a non-issue in JModelica.org, we do not consider this further.

4.2 *Elimination in non-scalar diagonal blocks*

We next consider the elimination of variables in non-scalar diagonal blocks. If the diagonal blocks are linear—that is, struct $\boldsymbol{F}^i \in \{0,1\}^{n^i \times n^i}$, where n^i is the dimension of $\boldsymbol{z}^i$—a conceivable approach would be to invoke a numerical factorization algorithm to solve for $\boldsymbol{z}^i$. This would however lead to crippling inefficiencies in the current dynamic optimization framework in JModelica.org[4], and is thus not considered further in this paper.

Another possibility of treating linear, non-scalar blocks is symbolic factorization. While this is readily supported by JModelica.org, experiments on the problems in Section 5.1 have shown that symbolic QR factorization all too often leads to numerical issues caused by instability due to the lack of numerical pivoting for practical use, and is thus not considered further in this paper.

There is however another approach that allows us to eliminate some of the variables in non-scalar diagonal blocks, which even extends to nonlinear blocks: tearing. By tearing the diagonal blocks as described in Section 3.5.2, we can then symbolically eliminate the causalized variables in the torn blocks by forward substitution.

There are two issues with the choice of causalized variables by the JModelica.org compiler. The first is that it can choose to causalize differential variable derivatives, which is useful when the causalization goal is to get an explicit ODE. But as per the discussion in Section 4.1, we are not interested in eliminating differential variable derivatives. Likewise, we are not interested in eliminating (actively) bounded variables. Our approach is thus to besides using the tearing variables and residuals selected by the compiler, we additionally add all differential variable

[4]The current dynamic optimization framework in JModelica.org symbolically represents the DAE using CasADi MX (Matrix eXpression) or SX (Scalar eXpression) [4] graphs. The embedding of numerical factorization algorithms requires the use of MX graphs. Since representing the full DAE using MX graphs is highly inefficient in terms of computational speed, an efficient implementation of this would require a sophisticated mixture of SX and MX graphs to represent the DAE in JModelica.org, or an extension of CasADi's SX graphs to allow function calls. Both of these possibilities are beyond the scope of this paper.

For Peer Review Only

derivatives and bounded variables as tearing variables and their respectively matched equations as tearing residuals. It would have been better to force the compiler to make these choices, allowing it to potentially make better choices in choosing the remaining tearing variables and residuals, but this has not been implemented.

SCHEME 2 *Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, and whose incidence in the diagonal block is linear. Also eliminate causalized, unbounded algebraic variables in torn, non-scalar diagonal blocks.*

4.3 *Sparsity-preserving elimination*

In our attempt to eliminate as many variables as possible using only symbolic techniques, we may actually end up eliminating too many from a computational efficiency perspective. As briefly mentioned before, although the elimination of variables leads to smaller systems, they also tend to be more dense. The increased density can cause crippling slowdowns when performing sparse computations, in particular when solving sparse linear systems (the linearized KKT system in each iteration of an NLP solver).

We thus propose to avoid some of the eliminations performed by SCHEME 1 & 2, by analysing the effect each elimination has on the DAE structural incidence matrix in a seventh step of the causalization procedure presented in Section 3.5. We define a density measure μ of a causalized block variable[5] $\bar{z}_j^i$ which measures how much denser the resulting NLP will become if it is eliminated using (21). Unfortunately, any useful measure will depend on which other eliminations are performed. This makes it intractable to consider all possible combinations, just like it is not tractable for direct, sparse linear solvers to find the pivot sequence with minimal fill-in. However, due to the causalization, we can consider the blocks and the variables within the blocks in sequence to devise a greedy algorithm which only considers the current block. Thus, for a given block, whether to perform the eliminations in preceding blocks has already been decided, and we only consider the situation in which no eliminations are performed in succeeding blocks, consequently disregarding how eliminations in succeeding blocks will be affected by the eliminations in the current block. This simplified approach is analogous to how linear solvers use the Markowitz criterion or local minimum fill-in as discussed in Section 3.6.

We thus want to estimate how much the number of nonzero elements in the KKT system will increase if $\bar{z}_j^i$ is eliminated. We perform the elimination if

$$\mu(i, j) \leq \mu_{\text{tol}}, \tag{25}$$

where μ_{tol} is a user-provided tolerance. Rather than considering the sparsity of the KKT matrix, we instead only consider the number of nonzeros in the structural incidence matrix of the DAE—which is the dominant part of the KKT system of a typical dynamic optimization problem—for simplicity and computational efficiency during symbolic preprocessing. An estimative measure for this is

$$\mu(i, j) = \left(-2 + \sum_{\alpha \in \{\dot{x}, x, y, u, p\}} \sum_{k=1}^{n_\alpha} \mathcal{I}(\nabla_{\alpha_k} \bar{F}_j^i) \cdot d(\alpha_k)\right) \cdot \left(-1 + \operatorname{nnz} \nabla_{\bar{z}_j^i} \boldsymbol{F}\right), \tag{26}$$

where $\mathcal{I}$ is the indicator function (mapping zero functions to 0 and all other functions to 1), $d(\alpha_k)$ is the number of post-elimination dependencies of α_k, and $\operatorname{nnz} \nabla_{\bar{z}_j^i} \boldsymbol{F}$ is the number of nonzero

[5]To allow uniform treatment of scalar, linear and torn diagonal blocks, we consider a scalar, linear diagonal block equation and its corresponding variable to be causalized.

For Peer Review Only

incidences of $\bar{z}^i_j$ in the DAE. A recursive expression for d is

$$d(\alpha_k) = \begin{cases} 1, & \text{if } \alpha_k \text{ is not eliminated,} \\ \sum_{\beta \in \{\dot{x},x,y,u,p\}} \sum_{l=1}^{n_\beta} \mathcal{I}(\nabla_{\beta_l} \bar{h}_{\alpha_k}) \cdot d(\beta_l), & \text{otherwise,} \end{cases} \tag{27}$$

where $\bar{h}_{\bar{z}^i_j} = \bar{H}^i_j$ (see (21)). Note that $d(\alpha_k)$ not only depends on α_k but also which variables have been chosen for elimination, which changes over the course of the sparsity analysis. The measure (26) is similar to the Markowitz criterion (23), with some differences. It considers the whole system simultaneously rather than a single stage of Gaussian elimination, and also takes into account that the full system not only depends on $\boldsymbol{z}$ but also $\boldsymbol{\nu}$. Another important difference is that the Markowitz criterion estimates fill-in, whereas (26) estimates the increase in number of nonzeros. For example, the elimination of an alias variable can cause a lot of fill-in, but will not increase the number of nonzeros in the DAE. Thus, the value given by (23) of an alias variables will be equal to the number of incidences in the subsequent equations, whereas the value given by (26) will be 0.

To demonstrate (25)–(27), consider the torn incidence (8) of the previous example with $\mu_{\text{tol}} = 3$. The variables that are eligible for elimination are y_4, y_1, and y_2. At the start of the procedure, we have $d(\alpha) = 1$ for all α. Since the first block $\boldsymbol{F}^1$ has no eligible variables, we move on to $\boldsymbol{F}^2$ where we find the pair $((1e), y_4)$—noting that $\bar{F}^2_1$ is the residual for (1e)—for which we compute

$$\begin{aligned} \mu(2,1) &= \left(-2 + \sum_{\alpha \in \{\dot{x},x,y\}} \sum_{k=1}^{n_\alpha} \mathcal{I}(\nabla_{\alpha_k} \bar{F}^2_1) \cdot d(\alpha_k)\right) \cdot \left(-1 + \operatorname{nnz} \nabla_{\bar{z}^2_1} \boldsymbol{F}\right) \\ &= \left(-2 + \sum_{\alpha \in \{\dot{x}_1,x_1,y_1,y_2,y_3,y_4,y_5\}} \mathcal{I}(\nabla_\alpha \bar{F}^2_1) \cdot d(\alpha)\right) \cdot \left(-1 + \operatorname{nnz} \nabla_{y_4} \boldsymbol{F}\right) \\ &= \left(-2 + \mathcal{I}(1) \cdot d(y_4) + \mathcal{I}\left(-\frac{1}{2\sqrt{y_5}}\right) \cdot d(y_5)\right) \cdot (-2 + \operatorname{nnz}(0, 0, 2y_1y_2, y_1 - 1, 1, 0)), \\ &= (-2 + 2) \cdot (-1 + 3) = 0 \le \mu_{\text{tol}} = 3, \end{aligned} \tag{28}$$

and thus proceed to eliminate y_4 and compute $d(y_4) = 1$. In the next block we find the eligible pairs $((1d), y_1)$ and $((1b), y_2)$. For the first pair we compute

$$\mu(3,1) = (-2 + 4) \cdot (3 - 1) = 4 > \mu_{\text{tol}}, \tag{29}$$

and thus do not eliminate y_1. Moving on to the second and final pair, we compute

$$\mu(3,2) = (-2 + 3) \cdot (3 - 1) = 2 \le \mu_{\text{tol}}, \tag{30}$$

and thus eliminate y_2, for which we compute $d(y_2) = 2$.

The measure (26) is locally suboptimal in the same sense as the Markowitz criterion: It does not take into account that not all incidences will cause fill-in; it is a worst case estimate. To address this, we can instead of (26) use the measure

$$\mu(i,j) = \sum_{\imath=i}^{m} \sum_{J=\begin{cases} j+1, & \text{if } \imath = i, \\ 1, & \text{otherwise} \end{cases}}^{n^\imath} \mathcal{I}(\nabla_{\bar{z}^i_j} F^\imath_J) \left(-1 + \sum_{\alpha \in \{\dot{x},x,y,u,p\}} \sum_{k=1}^{n_\alpha} (1 - \mathcal{I}(\nabla_{\alpha_k} F^\imath_J)) \mathcal{I}(\nabla_{\alpha_k} F^i_j) d(\alpha_k)\right), \tag{31}$$

For Peer Review Only

where F^i_j is element j in block i with the ordering obtained after tearing; that is,

$$F^i_j = \begin{cases} \bar{F}^i_j, & \text{if } j \leq \bar{n}_i, \\ \hat{F}^i_{j-\bar{n}_i}, & \text{if } j > \bar{n}_i. \end{cases} \tag{32}$$

This measure is analogous to local minimum fill-in in the same way as (26) is analogous to (23).

As discussed in Section 3.6, Markowitz is generally regarded as the best criterion for general-purpose pivot selection, with local minimum fill-in being a lot more computationally expensive while usually only yielding slightly better results. However, for our purposes, we only perform the sparsity-preservation analysis once offline. Furthermore, the difference in computation times between using (26) and (31) is usually negligible compared to the other offline computations (model compilation, BLT analysis, collocation discretization, algorithmic differentiation graph construction, and so on). So while both measures have been implemented in JModelica.org, we will henceforth only consider the usage of (31) due to its slightly better expected performance.

The usage of sparsity preservation can be used both with and without tearing, yielding two new families of schemes, parametrized by the density tolerance μ_{tol}.

SCHEME $3_{\mu_{\text{tol}}}$ *Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, whose incidence in the diagonal block is linear, and whose density measure is smaller than or equal to* μ_{tol}.

SCHEME $4_{\mu_{\text{tol}}}$ *Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, whose incidence in the diagonal block is linear, and whose density measure is smaller than or equal to* μ_{tol}. *Also eliminate causalized, unbounded algebraic variables in torn, non-scalar diagonal blocks whose density measure is smaller than or equal to* μ_{tol}.

The choice of density tolerance is important and non-obvious. A tolerance of 0 means that we only perform eliminations that do not increase the number of nonzeros in the other equations. Such variables are similar to alias variables. A tolerance of $-\infty$ means that no eliminations are performed at all, and we thus preserve the original DAE. A tolerance of ∞ means that we eliminate all causalized variables. We thus get SCHEME 0 = SCHEME $3_{-\infty}$ = SCHEME $4_{-\infty}$, SCHEME 1 = SCHEME 3_{∞}, and SCHEME 2 = SCHEME 4_{∞}.

The schemes to be evaluated in the next section are summarized in Table 1.

Table 1. Summary of techniques used in schemes.

SCHEME	BLT	Tearing	Sparsity preservation
0			
1	✓		
2	✓	✓	
$3_{\mu_{\text{tol}}}$	✓		✓
$4_{\mu_{\text{tol}}}$	✓	✓	✓

5. Benchmark

To evaluate the various proposed schemes, we will consider six different optimal control problems. Since six problems is too small a test suite to draw any general conclusions regarding computational speed, and in particular convergence robustness, we will generate a larger number of problem instances by randomizing the initial state. We will primarily focus on local collocation, but will in the end also present a small benchmark on how the schemes perform when using global collocation.

For Peer Review Only

5.1 *Problems*

The systems considered in the six problems are a car, a combined-cycle power plant (CCPP), a double pendulum, a kinematic loop with four bars, a distillation column, and a heat recovery steam generator (HRSG). All of the problems are encoded in Modelica and Optimica, and most of them are available as a part of JModelica.org's suite of examples. The problems together span a large part of the domain of interesting dynamic optimization problems.

5.1.1 Car

The problem is to find the time-minimal manoeuvre for a car in a 90-degree turn with a high initial velocity, making it challenging to simply stay on the road. A single-track chassis model is used, where the two wheels on each axle are lumped together. The model has three degrees of freedom: two translational and one rotational. The tire forces under pure slip conditions are computed with the 'Magic formula' [53], which then are used to compute combined slip forces using weighting functions. This problem, and several other similar ones, has been developed and studied by Berntorp et al. [12, 13, 47].

Unlike most of the other problems, the Modelica code for this problem is atypical in its being flat, rather than hierarchically object-oriented, leading to it consisting of relatively few lines of code and not very many algebraic variables and no trivial algebraic equations resulting from component connections. It thus serves to demonstrate the effects of symbolic elimination on more typical representations of DAEs, rather than the verbose equations that result when using Modelica model libraries. It is also the only problem we consider that has a free time horizon.

5.1.2 CCPP

The next case considers warm startup of combined-cycle power plants (CCPP). This problem has become highly industrially relevant during the last decade, due to an increasing need to improve power-generation flexibility due to the unpredictable output of renewable energy sources such as solar and wind power. The model was first developed by Casella et al. [21] but has since been further developed. The objective is a quadratic penalty from reference values, with an important upper bound on the turbine thermal stress.

This problem is the closest thing to a standard problem of dynamic optimization in the Modelica community, having been used several times for benchmark purposes [5, 6, 43, 55, 64].

5.1.3 Double pendulum

One of the elementary examples of multibody mechanics in the Modelica Standard Library (MSL) is a damped double pendulum: `Modelica.Mechanics.Examples.Elementary.DoublePendulum`. While a double pendulum is a conceptually simple fourth-order system, the MSL model is an index-three DAE with more than a hundred variables and an algebraic loop and is thus computationally non-trivial. This model is used to formulate the optimal control problem of inverting the full pendulum with quadratic penalties on the states and torque.

5.1.4 Fourbar1

Another high-index example from MSL multibody mechanics is `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar1`, which consists of 4 bars connected by 6 revolute joints and a prismatic joint. The block diagram of the model was previously shown in Figure 1. This system has three large algebraic loops, one of which is nonlinear. It is noteworthy that this system can be modelled more efficiently by modelling the joints differently,

For Peer Review Only

as done in `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar2` and `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar_analytic`. However, we choose to work with Fourbar1 to get a problem that is significantly more challenging than the double pendulum example. Based on this model, we consider the optimization problem to control the translation along the prismatic joint `j2` in one end of the mechanism by applying a bounded torque to the revolute joint `j1` in the other end by minimizing a quadratic cost function.

5.1.5 HRSG

The next problem concerns startup of a Heat Recovery Steam Generator (HRSG). The model has been developed and studied as part of an industrial research project and a series of industrial Master's theses [1, 59, 62]. While the model is moderate in size, it contains high-fidelity modelling of thermodynamic properties of media, causing high NLP function evaluation times. The objective is a quadratic penalty on the deviation from the desired steady state, with bounds on input rates and temperature gradients in spatially discretized walls.

The torn BLT decomposition obtained with Scheme 4_5 is shown in Figure 2. The BLT-ordered incidence in Figure 2 is typical for Modelica models in that it is sparse, has a large amount of scalar, linear diagonal blocks, and a small amount of large algebraic loops with mostly linear incidences.

5.1.6 Distillation column

The final problem concerns optimal control of a binary distillation column, which separates methanol from n-propanol and has 40 trays. The model was developed by Diehl [24] and the Modelica implementation was based on the MATLAB implementation by Hedengren [36]. The considered scenario is a short reflux breakdown during steady state, with the objective to steer back to the desired steady state, using quadratic costs on the deviation of two tray temperatures and input signals from the high-purity steady state. This scenario and model has been previously used for benchmarking dynamic optimization algorithms [43, 46]. Just like the Car model of Section 5.1.1, this model is implemented in a flat rather than hierarchical manner.

5.2 *Benchmark setup*

To generate a large test suite, we start with the solution to each of the six *nominal* problems described in Section 5.1. We then randomly perturb the initial state $\boldsymbol{x}_0$ and solve the perturbed problem using the nominal solution as initial guess. The purpose of this approach is to emulate the setting of Model Predictive Control (MPC) [45], without actually doing MPC, which would have had the drawback of introducing correlation between the problems solved in each sample point.

The nominal value of each initial state variable $x_{0,i}$ is multiplicatively perturbed by independently, identically, normally distributed random variables with standard deviation σ to yield the new initial state variable $\bar{x}_{0,i}$, that is

$$\bar{x}_{0,i} = \nu_i \cdot x_{0,i}, \quad \nu_i \sim \mathcal{N}(1, \sigma^2), \quad i = 1 \ldots n_x. \tag{33}$$

The standard deviation σ is hand-picked for each problem to make the corresponding instances suitably difficult. The resulting perturbed problem, henceforth referred to as an *instance*, may be infeasible. To counteract this, we make sure that the initial state satisfies all the problem bounds and path inequality constraints. To satisfy the bounds on state variables, we project $\bar{x}_{0,i}$ inside its

For Peer Review Only

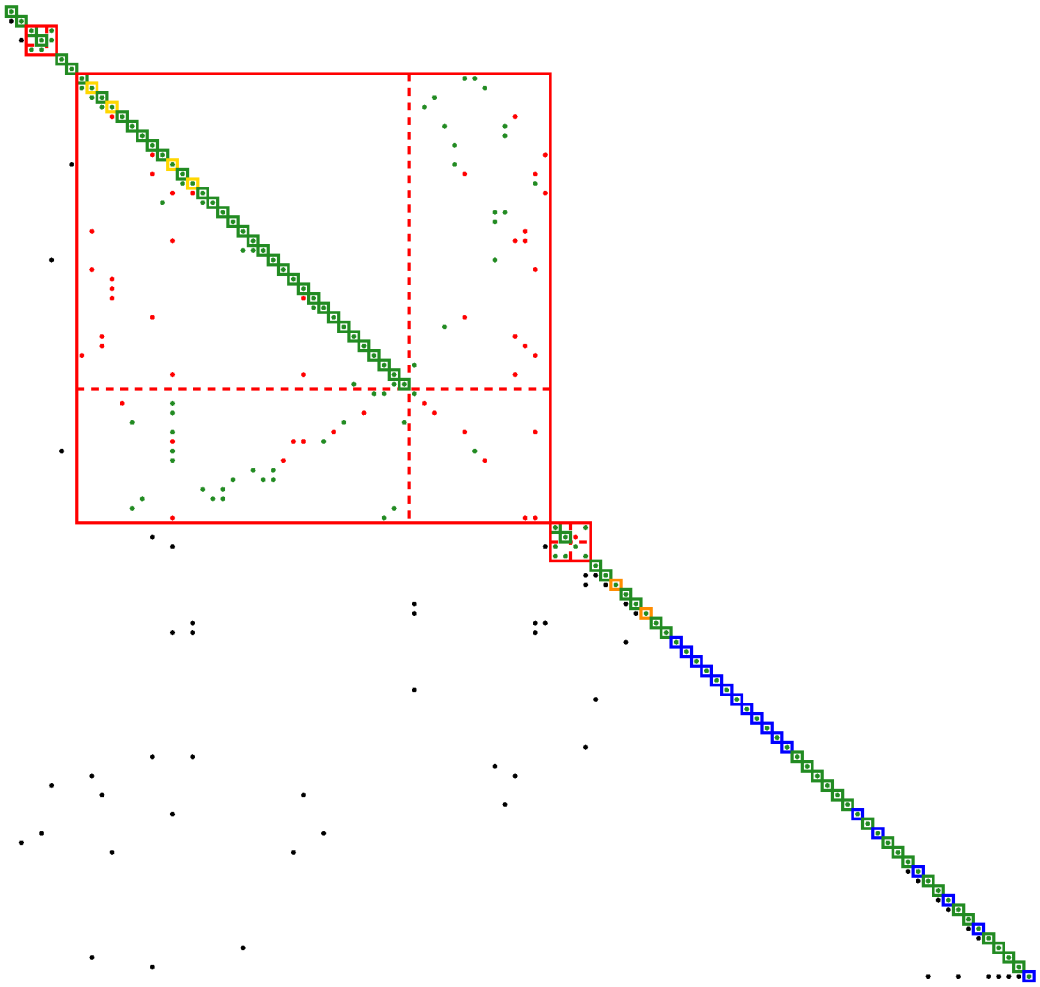


Figure 2. Torn, sparsity-preserving BLT decomposition of HRSG model. Linear incidences are marked by green dots, and nonlinear incidences are marked by red dots. Since we do not distinguish between linear and nonlinear incidences outside of the diagonal blocks, such incidences are marked by black dots. Torn blocks are marked by red edges. Variables, and their respective matched equations, that have been user-specified as actively bounded (and thus are not eliminated) are marked by orange edges. Differential variable derivatives (which are not eliminated), and their respective matched equations, are marked by blue edges. Variable-equation pairs along the diagonal that are not sparsity preserving—that is, do not satisfy (25)—are marked by yellow edges. The remaining variable-equation pairs along the diagonal are the ones used for elimination, which are marked by green edges.

bounds, and thus modify the first equation of (33) to

$$\bar{x}_{0,i} = \max\left(\min(\nu_i \cdot x_{0,i}, x_{0,i} + 0.9(x_{U,i} - x_{0,i}), x_{0,i} - 0.9(x_{0,i} - x_{L,i})\right), \tag{34}$$

where $x_{U,i}$ is the element of the bounds $\boldsymbol{U}$ that corresponds to x_i.[6] Projecting inside the feasible region of general path inequality constraints and algebraic variable bounds is more difficult. To satisfy these, we use JModelica.org's FMI-based DAE initialization algorithm to compute the value of $\bar{\boldsymbol{y}}_0$ that corresponds to $\bar{\boldsymbol{x}}_0$ and then check if they satisfy the constraints with a safety margin analogous to the one in (34); that is, the projected distance to the boundary should de-

[6]The factor 0.9 in (34) is used instead of 1.0 in order to project strictly inside the interior of the bounds, which is needed to satisfy the Linear Independence Constraint Qualification (LICQ) as the collocation algorithm in JModelica.org introduces x_0 as an optimization variable in order to treat general implicit initial equations.

crease by no more than 90%. If they do not, we discard the problem instance and generate a new one and repeat.

Table 2. Benchmark problems and the considered schemes for each problem, where $t_{\max}$ [s] is the allotted CPU time for each instance, n_y the number of non-eliminated algebraic variables, n the number of NLP variables divided by 1000, nnz J the number of nonzero elements in the NLP constraint Jacobian divided by 1000, nnz H the number of nonzero elements in the Hessian of the NLP Lagrangian divided by 1000.

Problem	n_x	n_u	n_e	n_c	σ	$t_{\max}$	Schemes	n_y	n	nnz J	nnz H
Car	13	3	60	3	0.1	30	0	23	9.7	37.4	9.4
							$1, 2, 3_{20}, 3_{30}, 3_{40}, 4_{20}, 4_{30}, 4_{40}$	4	6.3	30.4	10.0
							$3_5, 3_{10}, 4_5, 4_{10}$	5	6.4	30.0	9.2
CCPP	10	1	40	4	0.3	40	0	123	23.6	73	11.6
							$1, 3_{10}, 3_{20}, 3_{30}, 3_{40}$	3	4.3	19.7	4.2
							$2, 4_{10}, 4_{20}, 4_{30}, 4_{40}$	2	4.1	19.3	4.2
							3_5	6	4.7	21.3	4.2
							4_5	5	4.6	20.1	4.2
Dbl. pend.	4	1	100	3	0.3	50	0	124	40.4	123	24
							$1, 3_{10}, 3_{20}, 3_{30}, 3_{40}$	16	7.9	25	5
							$2, 4_{20}, 4_{30}, 3_{40}$	2	3.7	13	5
							3_5	17	8.2	27	5
							4_{10}	3	4.0	15	5
							4_5	5	4.6	17	5
Fourbar1	2	1	60	3	0.03	30	0	452	82.8	334	192
							$1, 3_{30}, 3_{40}$	246	45.5	215	160
							2	23	5.2	50	24
							3_{20}	247	45.7	216	161
							3_{10}	249	46.1	217	161
							3_5	255	47.2	220	162
							4_{40}	29	6.3	63	29
							4_{30}	30	6.5	63	32
							4_{20}	46	9.3	88	58
							4_{10}	85	16.4	127	107
							4_5	114	21.7	142	128
HRSG	18	3	25	5	0.3	60	0	84	15.9	56	6
							$1, 3_5, 3_{10}, 3_{20}, 3_{30}, 3_{40}$	56	12.4	47	6
							$2, 4_{30}, 4_{40}$	19	7.7	42	7
							4_{20}	20	7.9	41	6
							4_{10}	22	8.1	39	5
							4_5	23	8.2	39	5
Dist. col.	125	2	20	3	0.2	40	0	1000	78.7	291	97
							$1, 2$	2	17.9	315	140
							$3_{40}, 4_{40}$	8	18.2	162	64
							$3_{30}, 4_{30}$	10	18.4	151	56
							$3_{20}, 4_{20}$	14	18.6	135	44
							$3_{10}, 4_{10}$	26	19.3	123	34
							$3_5, 4_5$	55	21.1	120	31

We solve each instance with all schemes and use the values $\{5, 10, 20, 30, 40\}$ for μ_{tol}. The problem sizes and sparsity are detailed in Table 2, where we also specify the number of elements n_e and collocation points n_c used in each element. Each scheme is only allowed a certain CPU time $t_{\max}$ [s] amount of time for each problem instance, after which we regard the scheme as

For Peer Review Only

having failed. The time $t_{\max}$ has been chosen as approximately the average observed solution time of the slowest scheme over all instances for a given problem plus 5 estimated standard deviations. For some problems, some schemes yield identical results, as indicated in the table. For example, Car has no algebraic loops, and so SCHEME 1 and SCHEME 2 are the same for this problem.

The problems are solved with JModelica.org revision [8915], IPOPT 3.12.5, and the linear solver MA57 [38] with ordering by MeTiS [40]. The acceptable NLP tolerance in IPOPT is set equal to the NLP tolerance of 10^{-8}. To put a focus on convergence robustness rather than speed, the automatic scaling of MA57 is enabled and the pivot tolerance (see (24)) is increased from 10^{-8} to 10^{-4}. The IPOPT barrier parameter strategy is changed to adaptive, to avoid the issue of selecting the initial value of the barrier parameter and also because the authors favour this strategy. Other than the above exceptions, all the default values of the options of all the algorithms are used.

5.3 *Results*

In this section we present the results of the benchmark described in Sections 5.1 and 5.2. We generate 1000 instances whose initial states are in the interior of the constraints for each problem. We will use the same table headers throughout this section, with the following meanings. A scheme is considered to have *succeeded* on a problem instance if it converges to within tolerance of a solution within the maximum CPU time. A problem instance is considered *valid* if at least one scheme succeeds on it. Success [1] is the ratio of success of a scheme on the valid instances. For the instances on which all schemes succeeded, Time [s] is the average solution CPU time, σ_t [s] is the sample standard deviation of the solution time, and Iter [1] is the average number of iterations needed by a scheme.

5.3.1 Car

The results for 1000 instances of the Car problem are shown in Table 3. On 68.0% of the instances, all schemes succeed. 18.6% of the instances are invalid. The most significant difference between the schemes is that SCHEME 0 is approximately 35% slower than the others and slightly less robust. The dominant reason for failure is maximum CPU time, with some cases of convergence to a point of local infeasibility. A likely reason for the high percentage of invalid instances is that the nominal solution is close to the boundaries of feasibility, due to the high initial car velocity. The perturbed problems can thus end up being infeasible with significant probability, which is also why we chose the relatively small value of $\sigma = 0.1$.

Table 3. Scheme performances on Car.

SCHEME	Success	Time	σ_t	Iter
0	89.8%	8.9	4.5	104.0
1	97.2%	6.8	4.9	94.3
3_5	95.2%	6.4	5.0	103.9

5.3.2 CCPP

The results for 1000 instances of the CCPP problem are shown in Table 4. On 70.1% of the instances, all schemes succeed. 20.9% of the instances are invalid. We see that SCHEME 0 is an order of magnitude slower and also sometimes fails unlike the other schemes. There is little difference between the other schemes, which is to be expected, since tearing and sparsity preservation

For Peer Review Only

only makes minor changes to the problem as seen in Table 2. On almost all of the instances in which all schemes fail, all schemes except 0 report local infeasibility. SCHEME 0 only fails due to maximum CPU time.

Table 4. Scheme performances on CCPP.

SCHEME	Success	Time	σ_t	Iter
0	88.6%	13.6	5.3	101.9
1	100.0%	0.9	0.6	46.3
2	100.0%	0.9	0.5	46.4
3_5	100.0%	1.0	0.7	46.4
4_5	99.9%	1.0	0.6	46.5

5.3.3 *Double pendulum*

The results for 1000 instances of the Double pendulum problem are shown in Table 5. On 99.6% of the instances, all schemes succeed. All instances are valid. We see that SCHEME 1 reduces the solution time by an order of magnitude compared to SCHEME 0, which is further halved by SCHEME 3. The overzealous sparsity preservation that is needed to have an impact on the number of variables is not beneficial.

Table 5. Scheme performances on Double pendulum.

SCHEME	Success	Time	σ_t	Iter
0	99.6%	15.5	7.0	99.9
1	100.0%	2.1	1.0	72.4
2	100.0%	0.8	0.4	58.3
3_5	100.0%	2.2	1.0	71.1
4_{10}	100.0%	0.9	0.4	58.7
4_5	100.0%	1.0	0.5	58.9

5.3.4 *Fourbar1*

The results for 1000 instances of the Fourbar1 problem are shown in Table 6. On 56.1% of the instances, all schemes succeed. 34.0% of the instances are invalid. The dominant reasons for failure are restoration failure in IPOPT and maximum CPU time. We see that SCHEME 2 performs significantly better than all the other schemes, and that despite significant density in the problem after applying tearing, sparsity preservation actually does more harm than good. One might have suspected this already when inspecting Table 2, as Fourbar1 is the only problem where sparsity preservation significantly increases nnz J rather than decrease it. Also, the Lagrangian Hessian is relatively denser for this problem than the others, which we have neglected in the sparsity preservation.

5.3.5 *HRSG*

The results for 1000 instances of the HRSG problem are shown in Table 7. On 72.2% of the instances, all schemes succeed. 23.1% of the instances are invalid. The dominant reasons for failure are local infeasibility and maximum CPU time. We see that SCHEME 1 only offers a slight improvement over SCHEME 0, but the use of tearing and sparsity preservation improves both robustness and solution times.

For Peer Review Only

Table 6. Scheme performances on Four-bar1.

Scheme	Success	Time	σ_t	Iter
0	86.8%	8.9	2.4	15.6
1	87.1%	4.6	1.3	15.4
2	99.8%	1.4	0.4	15.2
3_{20}	87.1%	4.6	1.3	15.4
3_{10}	87.1%	4.6	1.3	15.4
3_5	86.8%	5.0	1.4	15.4
4_{40}	93.5%	1.6	0.5	15.3
4_{30}	95.2%	1.5	0.4	15.2
4_{20}	90.8%	1.7	0.5	15.4
4_{10}	85.8%	2.3	0.7	15.4
4_5	95.6%	3.2	0.9	15.4

Table 7. Scheme performances on HRSG.

Scheme	Success	Time	σ_t	Iter
0	96.0%	13.5	8.2	66.0
1	96.9%	11.1	7.2	68.8
2	99.3%	8.9	4.1	48.5
4_{20}	99.2%	7.7	4.4	49.8
4_{10}	99.1%	6.7	4.0	49.3
4_5	99.2%	6.8	4.2	49.7

5.3.6 Distillation column

When using IPOPT's adaptive barrier parameter strategy, Scheme 0 is unable to solve a single instance of the Distillation column problem. Although Scheme 0 works better with the monotone strategy for this problem, we do not change strategy only to accommodate Scheme 0. The results for the remaining schemes for 1000 instances of the Distillation column problem are thus shown in Table 8. On 74.2% of the instances, all remaining schemes succeed. 12.1% of the instances are invalid. On the valid instances, the sole reason for failure is maximum CPU time. We see that sparsity preservation approximately halves the solution time.

Table 8. Scheme performances on Distillation column. Scheme 0 failed all instances.

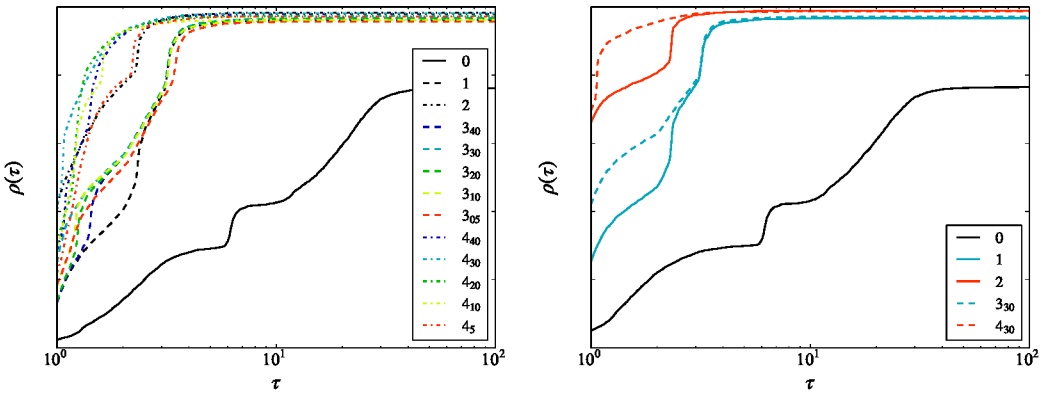
Scheme	Success	Time	σ_t	Iter
1	94.0%	14.3	4.2	13.0
3_{40}	96.1%	8.9	2.8	13.1
3_{30}	96.5%	6.3	2.3	13.2
3_{20}	95.2%	7.8	3.1	13.1
3_{10}	94.4%	7.5	2.5	13.0
3_5	92.9%	8.5	5.2	17.1

5.4 *Performance profiles*

In Section 5.3 we saw that Scheme 1 on average outperforms Scheme 0 for each considered problem, and that Schemes 2, $3_{\mu_{\text{tol}}}$, and $4_{\mu_{\text{tol}}}$ usually yields further improvements. To illustrate the aggregated results, Figure 3a shows the performance profile [26] for all schemes on the valid portion of the 6000 problem instances. The performance $\rho_s(\tau)$ of a scheme s is defined as the ratio of instances in which s solved the problem no slower than a factor τ of the fastest scheme of that instance. In particular, $\rho_s(1)$ is the ratio of instances in which s was the fastest scheme, and $\rho_s(\infty)$ is the ratio of instances in which s succeeded.

Without looking too closely at the overwhelming data of Figure 3a, we can see that the best value of μ_{tol} tends to be 30—with 20 being a close contender—for both Scheme $3_{\mu_{\text{TOL}}}$ and $4_{\mu_{\text{tol}}}$. For

For Peer Review Only



(a) Performance profile of all schemes.

(b) Performance profile of selected schemes.

Figure 3. Performance profiles for the valid portion of 6000 problem instances. Subfigure (a) shows all considered schemes with 5 different values of μ_{tol}. Subfigure (b) shows the same results but only for $\mu_{\text{tol}} = 30$.

the sake of clarity, we thus generate a new performance profile where we only consider $\mu_{\text{tol}} = 30$, which is shown in Figure 3b, where we clearly see the average superiority of SCHEME 4 both in terms of speed and robustness, although SCHEME 2 actually has slightly better robustness. In particular, we see that SCHEME 4_{30} is an order of magnitude faster than SCHEME 0 in approximately half of the considered instances.

5.5 *Computation times*

Regarding computation times, we have so far focused on the respective solution times of the schemes. For many applications, such as online MPC, these are the only times that matter. For other applications, the computation time of the full toolchain is important, in which case the proposed techniques add additional computational steps during preprocessing compared to SCHEME 0. The main parts of the offline computation times are model compilation, BLT analysis, collocation discretization, algorithmic differentiation graph construction (including first- and second-order derivatives). The main parts of the online computation times are NLP function evaluations and KKT matrix factorization. While SCHEME 4_{30} adds additional computation time compared to SCHEME 0 in the form of BLT analysis, it also results in a smaller DAE which may reduce computation in subsequent offline steps, allowing the additional computation time to be regained before we even reach the online computation.

Table 9 compares the offline and online computation times of SCHEME 0 and SCHEME 4_{30} for the six benchmark problems. The online times are the averages observed in Section 5.3. We see that for all except one problem, SCHEME 4_{30} has a lower total time. We also see that the offline times for SCHEME 4_{30} become significantly larger for problems with many DAE variables. Since the implementation of SCHEME 4_{30} is just a prototype with little regard for efficient offline computations (other than remaining in the realm of tractability), its implementation can probably be optimized to scale better. The bottleneck for Distillation column lies in the construction of the structural incidence matrix: Identifying the nonzero incidences and determining which of those are linear, which thus affects all schemes except 0.

Table 9. Offline and online computation times [s].

Problem	Scheme	Offline	Online	Total
Car	0	2.7	8.9	11.6
	4_{30}	2.7	6.8	9.5
CCPP	0	3.5	13.6	17.1
	4_{30}	3.6	0.9	4.5
Double pendulum	0	7.8	15.5	23.3
	4_{30}	7.1	0.8	7.9
Fourbar1	0	31.3	8.9	40.2
	4_{30}	43.4	1.5	44.9
HRSG	0	10.7	13.5	24.2
	4_{30}	12.5	8.9	21.4
Distillation column	0	18.9	∞	∞
	4_{30}	114.2	6.3	120.5

5.6 *Global collocation*

We have so far only evaluated the schemes when combined with local collocation. Global collocation (also known as pseudospectral) methods, where a large number of collocation points and a small number of elements are used instead of vice versa, have become increasingly popular the last decade. We will thus also consider these methods, but only briefly for the sake of brevity and also avoiding the inadvertent comparison between the performances of the two flavours of collocation. One important difference between local and global collocation to keep in mind is that the NLPs resulting from global collocation tend to be smaller, due to their potential spectral (exponential) convergence rate, but also more dense, due to their global temporal coupling.

We will limit our evaluation of global collocation to the Distillation column problem. Instead of using $n_e = 20$ and $n_c = 3$, we use $n_e = 1$ and $n_c = 15$. We also increase the maximum solution time $t_{\max}$ from 40 to 60. Just as we observed for local collocation, Scheme 0 is unable to solve the problem. The results for the remaining schemes are shown in Table 10. On 77.2% of the instances, all schemes succeed. 11.8% of the instances are invalid. Unlike what we saw for local collocation, much smaller values than 30 seem to be preferable for μ_{tol}, indicating that it is crucial to preserve what little sparsity there is in an NLP resulting from global collocation. However, if the problem dimensions allow it, it may be more efficient to combine Scheme 1 with dense rather than sparse numerical linear algebra, which is a possibility we do not consider further in this paper.

Table 10. Scheme performances on Distillation column with global collocation.

Scheme	Success	Time	σ_t	Iter
1	91.0%	32.4	6.2	10.8
3_{40}	98.1%	5.9	2.4	11.0
3_{30}	97.4%	4.2	1.2	10.9
3_{20}	97.1%	3.7	1.1	10.9
3_{10}	98.1%	1.8	1.2	11.1
3_5	97.5%	1.5	0.5	10.9

6. Conclusion

DAE-constrained optimization problems are usually solved by exposing the full DAE to a discretization method. In this paper we considered ways of preprocessing the DAE by symbolically

For Peer Review Only

eliminating most of the algebraic variables using techniques based on block-triangular orderings, tearing, and sparsity preservation, resulting in 4 different schemes. These schemes have been implemented in the open-source JModelica.org platform, which solves dynamic optimization problems where the system dynamics are described using the Modelica language. We evaluated these schemes on 6 different optimal control problems when combined with direct local collocation, and found that the scheme that utilizes all of the proposed techniques performed the best on average, often being an order of magnitude faster than the conventional scheme of exposing the full DAE to the discretization method. We found that a suitable value of μ_{tol} is 30 when employing local collocation, and that a significantly smaller value seems preferable for global collocation. We observed that the proposed techniques lend themselves well to typical Modelica models, due to the hierarchical modelling approach, but that they also show potential for flat DAEs.

Sparsity preservation was beneficial for the problems where it made a significant difference, excepting Fourbar1. A possible and tractable refinement of the proposed sparsity preservation procedure that may remedy this is to not only consider the sparsity of the NLP Jacobian, but also the Hessian of the NLP Lagrangian.

While block-triangular ordering and sparsity preservation guarantees preservation of numerical stability, tearing does not. Although numerical instability due to tearing does not appear to have been an issue for the considered problems, the authors are confident that there are industrially relevant problems where JModelica.org selects numerically troublesome tearing variables and residuals. This is a well-known drawback of tearing, which is difficult to address under typical circumstances. However, in the context of dynamic optimization a decent initial guess of the full solution is often required in order to solve the problem. Utilizing this initial guess, it should be tractable to design a numerical tearing algorithm which guarantees numerical stability (under the assumption that the initial guess is sufficiently close to the solution) using techniques such as those considered by Westerberg et al. [34, 63].

The used tearing algorithm first selects tearing variables and residuals to obtain causalized equations that are triangular and linear along the diagonal, and then selects additional tearing variables and residuals in order to not eliminate variables that are bounded, differential, or cause too much fill-in. Considering all of these criteria simultaneously, rather than sequentially, would enable fewer tearing variables and residuals to be selected.

Acknowledgements

This work was supported by the Swedish Research Council through the LCCC Linnaeus Center. Fredrik Magnusson is a member of the eLLIIT Excellence Center at Lund University. The authors sincerely thank Ali Baharev for his valuable comments.

References

[1] M. Åberg, *Optimisation-friendly modelling of thermodynamic properties of media*, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden, 2016.

[2] J. Åkesson, *Optimica—An extension of Modelica supporting dynamic optimization*, in *Proceedings of the 6th International Modelica Conference*, Bielefeld, Germany, 2008, pp. 57–66.

[3] J. Åkesson, K.E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, *Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems*, Comput. Chem. Eng. 34 (2010), pp. 1737–1749.

[4] J. Andersson, *A general-purpose software framework for dynamic optimization*, Ph.D. thesis, Arenberg Doctoral School, KU Leuven, 2013.

For Peer Review Only

[5] J. Andersson, J. Åkesson, F. Casella, and M. Diehl, *Integration of CasADi and JModelica.org*, in *Proceedings of the 8th International Modelica Conference*, Dresden, Germany, 2011, pp. 218–231.
[6] M. Axelsson, F. Magnusson, and T. Henningsson, *A framework for nonlinear model predictive control in JModelica.org*, in *Proceedings of the 11th International Modelica Conference*, Paris, France, 2015, pp. 301–310.
[7] B. Bachmann, L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson, *Parallel multiple-shooting and collocation optimization with OpenModelica*, in *Proceedings of the 9th International Modelica Conference*, Munich, Germany, 2012, pp. 659–668.
[8] A. Baharev, H. Schichl, and A. Neumaier, *Tearing systems of nonlinear equations – I. A survey*. Submitted for publication. Available at http://reliablecomputing.eu/baharev_tearing_survey.pdf.
[9] A. Baharev, H. Schichl, and A. Neumaier, *Tearing systems of nonlinear equations – II. A practical exact algorithm*. Submitted for publication. Available at http://reliablecomputing.eu/baharev_tearing_exact_algorithm.pdf.
[10] M. Baltes, R. Schneider, C. Sturm, and M. Reuss, *Optimal experimental design for parameter estimation in unstructured growth models*, Biotech. Progr. 10 (1994), pp. 480–488.
[11] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comput. Methods Appl. Mech. Engrg. 1 (1972), pp. 1–16.
[12] K. Berntorp and F. Magnusson, *Hierarchical predictive control for ground-vehicle maneuvering*, in *American Control Conference, 2015*, Chicago, IL, 2015, pp. 2771–2776.
[13] K. Berntorp, B. Olofsson, K. Lundahl, and L. Nielsen, *Models and methodology for optimal trajectory generation in safety-critical road–vehicle manoeuvres*, Vehicle System Dynamics 52 (2014), pp. 1304–1332.
[14] J.T. Betts, *Survey of numerical methods for trajectory optimization*, J. Guid. Contr. Dynam. 21 (1998), pp. 193–207.
[15] J.T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed., SIAM, Philadelphia, PA, 2010.
[16] L.T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*, MOS-SIAM, Philadelphia, PA, 2010.
[17] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.V. Peetz, and S. Wolf, *The functional mockup interface for tool independent exchange of simulation models*, in *Proceedings of the 8th International Modelica Conference*, Dresden, Germany, 2011, pp. 105–114.
[18] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Classics in Applied Mathematics, Vol. 14, SIAM, Philadelphia, PA, 1996.
[19] C. Büskens and D. Wassel, *The ESA NLP solver WORHP*, in *Modeling and Optimization in Space Engineering*, G. Fasano and J.D. Pintér, eds., Vol. 73, Springer, New York, NY, 2013, pp. 85–110.
[20] S.L. Campbell, *The numerical solution of higher index linear time varying singular systems of differential equations*, SIAM J. Sci. Stat. Comput. 6 (1985), pp. 334–348.
[21] F. Casella, F. Donida, and J. Åkesson, *Object-oriented modeling and optimal control: A case study in power plant start-up*, in *18th IFAC World Congress*, Milano, Italy, 2011, pp. 9549–9554.
[22] F.E. Cellier and E. Kofman, *Continuous System Simulation*, Springer, New York, NY, 2006.
[23] A.M. Cervantes, A. Wächter, R.H. Tütüncü, and L.T. Biegler, *A reduced space interior point strategy for optimization of differential algebraic systems*, Comput. Chem. Eng. 24 (2000), pp. 39–51.
[24] M. Diehl, *Real-time optimization for large scale nonlinear processes*, Ph.D. thesis, Heidelberg University, 2002.
[25] M. Diehl, H.G. Bock, J.P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, *Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations*, J. Proc. Cont. 12 (2002), pp. 577–585.
[26] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213.
[27] I.S. Duff, A. Erisman, and J.K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, United Kingdom, 1986.
[28] I.S. Duff and J.K. Reid, *The design of MA48: a code for the direct solution of sparse unsymmetric linear systems of equations*, ACM Trans. Math. Software 22 (1996), pp. 187–226.
[29] H. Elmqvist and M. Otter, *Methods for tearing systems of equations in object-oriented modeling*, in *European Simulation Multiconference*, Barcelona, Spain, 1994, pp. 326–332.
[30] R. Fletcher, *Block triangular orderings and factors for sparse matrices in LP*, in *Numerical Analysis 1997*, D. Griffiths, G. Watson, and D. Higham, eds., Longman, Harlow, United Kingdom, 1998, pp. 91–110.
[31] R. Franke, *Formulation of dynamic optimization problems using Modelica and their efficient solution*, in *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, 2002, pp. 315–322.
[32] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3*, 2nd ed., Wiley-IEEE Press, Piscataway, NJ, 2015.

For Peer Review Only

[33] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal. 10 (1973), pp. 345–363.
[34] P.K. Gupta, A.W. Westerberg, J.E. Hendry, and R.R. Hughes, *Assigning output variables to equations using linear programming*, AIChE J. 20 (1974), pp. 397–399.
[35] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2nd ed., Springer-Verlag, Berlin, Germany, 1996.
[36] J.D. Hedengren, *A nonlinear model library for dynamics and control, 2008*. Available online: http://www.hedengren.net/research/Publications/Cache_2008/NonlinearModelLibrary.pdf.
[37] J.E. Hopcroft and R.M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput. 2 (1973), pp. 225–231.
[38] HSL, *A collection of Fortran codes for large scale scientific computation*, software available at http://www.hsl.rl.ac.uk.
[39] A. Ilzhoefer, B. Houska, and M. Diehl, *Nonlinear MPC of kites under varying wind conditions for a new class of large-scale wind power generators*, Internat. J. Robust Nonlinear Control 17 (2007), pp. 1590–1599.
[40] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput. 20 (1998), pp. 359–392.
[41] G. Kron, *Diakoptics: The Piecewise Solution of Large-Scale Systems*, Vol. 2, MacDonald, London, United Kingdom, 1963.
[42] K. Krüger, R. Franke, and M. Rode, *Optimization of boiler start-up using a nonlinear boiler model and hard constraints*, Energy 29 (2004), pp. 2239–2251.
[43] E. Lazutkin, A. Geletu, S. Hopfgarten, and P. Li, *An analytical hessian and parallel-computing approach for efficient dynamic optimization based on control-variable correlation analysis*, Ind. Eng. Chem. Res. 54 (2015), pp. 12086–12095.
[44] B. Lennernäs, *A CasADi based toolchain for JModelica.org*, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden, 2013.
[45] J.M. Maciejowski, *Predictive Control with Constraints*, Prentice-Hall, Upper Saddle River, NJ, 2002.
[46] F. Magnusson and J. Åkesson, *Dynamic optimization in JModelica. org*, Processes 3 (2015), pp. 471–496.
[47] F. Magnusson, K. Berntorp, B. Olofsson, and J. Åkesson, *Symbolic transformations of dynamic optimization problems*, in *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, 2014, pp. 1027–1036.
[48] H.M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Manag. Sci. 3 (1957), pp. 255–269.
[49] S.E. Mattsson, H. Olsson, and H. Elmqvist, *Dynamic selection of states in Dymola*, in *Modelica Workshop 2000 Proceedings*, Lund, Sweden, 2000, pp. 61–67.
[50] S.E. Mattsson and G. Söderlind, *Index reduction in differential-algebraic equations using dummy derivatives*, SIAM J. Sci. Comput. 14 (1993), pp. 677–692.
[51] P. Meijer, *Tearing differential algebraic equations*, M.Sc. thesis, Centre for Mathematical Sciences, Lund University, Sweden, 2011.
[52] L.W. Nagel and D. Pederson, *SPICE (simulation program with integrated circuit emphasis)*, Tech. Rep. UCB/ERL M382, EECS Department, University of California, Berkeley, 1973.
[53] H.B. Pacejka, *Tire and Vehicle Dynamics*, 2nd ed., Butterworth-Heinemann, Oxford, United Kingdom, 2006.
[54] C.C. Pantelides, *The consistent initialization of differential-algebraic systems*, SIAM J. Sci. Stat. Comput. 9 (1988), pp. 213–231.
[55] P. Parini, *Object oriented modeling and dynamic optimization of energy systems with application to combined cycle power plant start-up*, M.Sc. thesis, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, 2015.
[56] P.C. Piela, T. Epperly, K. Westerberg, and A.W. Westerberg, *ASCEND: An object-oriented computer environment for modeling and analysis: The modeling language*, Comput. Chem. Eng. 15 (1991), pp. 53–72.
[57] A. Prata, J. Oldenburg, A. Kroll, and W. Marquardt, *Integrated scheduling and dynamic optimization of grade transitions for a continuous polymerization reactor*, Comput. Chem. Eng. 32 (2008), pp. 463–476.
[58] A.V. Rao, *A survey of numerical methods for optimal control*, in *Proceedings of the 2009 AAS/AIAA Astrodynamics Specialists Conference*, Pittsburgh, PA, 2009.
[59] H. Runvik, *Modelling and start-up optimization of a coal-fired power plant*, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden, 2014.
[60] S.M. Safdarnejad, J.D. Hedengren, N.R. Lewis, and E.L. Haseltine, *Initialization strategies for optimization of dynamic systems*, Comput. & Chem. Eng. 78 (2015), pp. 39–50.
[61] R.E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. 1 (1972), pp. 146–160.
[62] M. Thelander Andrén and C. Wedding, *Development of a solution for start-up optimization of a thermal power plant*, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden, 2015.
[63] A.W. Westerberg and F.C. Edie, *Computer-aided design, part 1 enhancing convergence properties by the choice of output variable assignments in the solution of sparse equation sets*, Chem. Eng. J. 2 (1971), pp. 9–16.

For Peer Review Only

[64] D.P. Word, J. Kang, J. Åkesson, and C.D. Laird, *Efficient parallel solution of large-scale nonlinear dynamic optimization problems*, Comput. Optim. and Appl. 59 (2014), pp. 667–688.
[65] A. Wächter and L.T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Math. Program. 106 (2006), pp. 25–57.
[66] V.M. Zavala, *Inference of building occupancy signals using moving horizon estimation and Fourier regularization*, J. Proc. Cont. 24 (2014), pp. 714–722.

For Peer Review Only