DAE Solvers for Large-Scale Hybrid Models

Erik Henningsson[1] Hans Olsson[1] Luigi Vanfretti[2]

[1]Dassault Systèmes AB, Lund, Sweden, {Erik.Henningsson, Hans.Olsson}@3ds.com
[2]Rensselaer Polytechnic Institute, Troy, NY, USA, vanfrl@rpi.edu

Abstract

We present a strategy for DAE mode simulations of large-scale Modelica models with state events. DAE solvers can be orders of magnitudes faster than traditional ODE solvers when simulating models with large algebraic loops. Such loops are common in, for example, power grid models. Central for our DAE mode approach is the accurate and efficient treatment of state events. Adapting, extending, and optimizing results known in the literature to the Modelica context resulted in a DAE mode implementation first released in Dymola 2019 and 3DEXPERIENCE 2019x. The implementation is verified by efficiency experiments featuring OpenIPSL power grid models. The run times for these models are competitive with domain-specific, state-of-the-art simulation tools.
Keywords: DAE mode, hybrid model, state event, large-scale, Modelica, power grid model

1 Introduction

As a high-level, equation based, and object-oriented language Modelica promotes easy construction, modification and reuse of models. It is therefore well suited for modeling large-scale, integrated physical systems, see e.g. (Baudette et al., 2018; Casella et al., 2016; Jorissen et al., 2015).

With the increased presence of such large-scale models, higher demands are put on Modelica tools to facilitate fast simulations. To meet those demands, special model structures are typically analyzed and exploited (Casella, 2015). Examples of strategies that have been successfully realized in Modelica tools, such as Dymola and the 3DEXPERIENCE platform, involve: multirate simulation (Thiele et al., 2014), mixed-mode simulation (Schiela and Olsson, 2000; Thiele et al., 2014), model decoupling and parallel execution (Elmqvist et al., 2014), and sparse solvers (Braun et al., 2017).

In this paper we will consider the strategy referred to as *DAE solver* or *DAE mode*. The name comes from the mathematical representations of Modelica models: hybrid *differential-algebraic equations (DAEs)*. When generating simulation code a Modelica tool performs a series of symbolic transformations involving common subexpression elimination, equation sorting, index reduction, and tearing (Cellier and Kofman, 2006). During this process the high-index DAE is transformed into an index-1 DAE, and then, by solving systems of equations, it is normally transformed into an ordinary differential equation (ODE). The latter can be integrated by an ODE solver like CVode (Hindmarsh et al., 2005). For most models the transformations make the numerics simpler and result in more robust and efficient simulations.

However, some numerical integrators, such as Dassl (Brenan et al., 1996), also allows integration of the index-1 DAE directly. For certain models, such DAE mode simulations can be orders of magnitude faster, among other things, due to more efficient treatment of algebraic loops. Significant speed-ups have, for example, been observed when simulating national- and continental-sized models of electrical power systems (Braun et al., 2017). Rosenbrock DAE integrators were used by (Olsson et al., 2017) to achieve fast and predictable run times for model-based embedded control.

The goal of this paper is to present a strategy for robust, accurate, and efficient simulation of *hybrid* DAEs using DAE integrators like Dassl. Because integration of the index-1 DAE is well-understood, the main focus will be on how to accurately and efficiently localize and treat state events. To achieve this, we will argue for an approach where the same generated code is used in DAE mode as in ODE mode. By using all the symbolic transformations and optimizations, the DAE fed to the integrators is kept to a minimal size. The trade-off here being some loss of sparsity (Braun et al., 2017; Magnusson, 2016). As an outlook we will also discuss further benefits and possibilities enabled by this approach.

Based on this strategy, DAE mode for hybrid DAEs was introduced in Dymola 2019 (released in June 2018) and 3DEXPERIENCE 2019x for a diverse selection of numerical integrators. The efficiency and accuracy of the implementation is verified by simulations of the Nordic power grid model *Nordic 44* from the OpenIPSL library (Vanfretti et al., 2017).

2 DAE mode for hybrid systems

2.1 Mathematical formulation

Mathematically, Modelica models are represented by hybrid DAEs. That is, differential-algebraic equations that may have discontinuities and/or may be controlled by discrete variables and conditions that change at events.[1] The

[1]Note that varying-structure models and multi-mode simulations are out of scope of this paper.

general form of the DAE is

$$F(t,\dot{x},x,y,d) = 0, \tag{1}$$

where t denotes the independent time and y the algebraic variables. Further, x are the differential variables and $\dot{x}$ are their time derivatives. The discrete variables, which may change value only at events, are denoted by d. For the initial value problem to be well-defined, appropriate initial conditions must also be supplied.

Throughout the simulation time- and state-dependent crossing functions

$$c = q(t,\dot{x},x,y,d) \tag{2}$$

are monitored for sign changes. The variable c represents the conditions of all if- and when-clauses. At zero-crossings an event is triggered. If the corresponding crossing function q_i depends on any of $\dot{x}$, x, or y it is called a state event, otherwise a time event. The former are more difficult to locate and their combination with DAE mode simulations is the main topic of this paper. When an event is triggered a reinitialization is performed using the DAE (1) and the crossing equations (2) together with additional discrete equations

$$d = \eta(t,\dot{x},x,y,\mathrm{pre}(d),c). \tag{3}$$

Here $\mathrm{pre}(d)$ are the previous values of the discrete variables. For details see (Olsson, 2017, Appendix C). This combination of equations defines a continuous-discrete mixed system of equations to be solved for the derivatives $\dot{x}$, the algebraic variables y, the discrete variables d, and the conditions c. Together, the three systems (1) – (3) define a hybrid differential-algebraic equation.

To construct simulation code a Modelica tool, such as Dymola, applies several symbolic transformations to the original hybrid DAE. These steps involve e.g. common subexpression elimination, sorting, index reduction, and tearing. For details see (Cellier and Kofman, 2006). During the process of reducing the index to one, the number of differential variables may decrease and the number of algebraic equations may increase. As the goal of these transformations is to transform the DAE (1) into an ODE the tool will select states $x(t) \in \mathbb{R}^{n_x}$ and solve for the state derivatives. Each derivative $\dot{x}_i$ that cannot be solved for symbolically is replaced by an algebraic variable $\hat{x}_i$ and the equation $\dot{x}_i = \hat{x}_i$.

For a typical Modelica model a significant part of the algebraic variables y in the original DAE (1) do not affect the dynamics of the model. These are the auxiliary variables and are not required during continuous simulation. We will therefore exclude them from the DAE provided to the numerical integrator. However, they must be computed when evaluating the crossing equations (2) and the discrete equations (3) and we must therefore consider them when locating and resolving events.

The symbolic transformations turns the original system into a sequence of assignment statements intertwined with smaller (nonlinear) systems of equations, the algebraic loops. These loops are then torn to minimize their size (Elmqvist and Otter, 1994). Denote by n_G the number of loops that affect the dynamics. Further, denote by z_i the iteration (tearing) variables of loop i, where $z_i(t) \in \mathbb{R}^{n_i}$, $n_i \geq 1$. These normally constitute a small subset of the algebraic variables. The algebraic loops are represented by the systems of equations

$$0 = G_i(z_i; t,x,z_1,\ldots,z_{i-1},d), \tag{4}$$

for $i = 1,\ldots,n_G$. Due to the equation sorting each algebraic loop is independent of later ones. The functions G_i do not depend on the state derivatives $\dot{x}$, since these were either solved for or substituted for an algebraic variable as described above.

The goal of the symbolic transformation is to causalize the DAE into an ODE,

$$\dot{x} = \hat{f}(t,x,d). \tag{5}$$

Here, the algebraic variables z and the algebraic loops are internal to the ODE. Thus, the evaluation of $\hat{f}$ requires the solution of the systems of equations defined in (4), which may be solved in sequence, separate from each other.

For the DAE mode approach presented in this paper we consider the DAE in the form it takes after all of the symbolic transformations have been performed, with one exception: the iterative solution of algebraic loops. Note that the loops that can be solved symbolically are still solved in that way, involving linear loops and the inversion of elementary functions like sin. Thus, we elevate, from the function $\hat{f}$, the loops that cannot be solved symbolically and arrive at the semi-explicit, index-1 DAE of interest for this paper

$$\dot{x} = f(t,x,z,d), \tag{6a}$$
$$0 = G(z; t,x,d), \tag{6b}$$

where $z = (z_1^{\mathrm{T}},\ldots,z_{n_G}^{\mathrm{T}})^{\mathrm{T}}$ and $G = (G_1^{\mathrm{T}},\ldots,G_{n_G}^{\mathrm{T}})^{\mathrm{T}}$. The important difference between $\hat{f}$ and f is that the evaluations of the latter do not require the solution of the loops (4), rather the algebraic variables z are inputs.

The remaining algebraic variables y can be computed from the variables in Equation (6). The computation of the subset of y that affects the dynamics is internalized in f and G. By construction, these computations are merely assignment statements.

Similarly, the computations of y can be internalized in the crossing functions, giving

$$c = Q(t,\dot{x},x,z,d).$$

Note that computing the auxiliary part of y may involve solving further (torn) algebraic loops, not considered in the dynamics.[2] Thus, computing all of y from

[2] Alternatively these algebraic loops for auxiliary variables may also be handled by the integrator, resulting in better predictors for the involved variables.

DOI 10.3384/ecp19157491

the variables in Equation (6) may be expensive. However, the crossing functions themselves q_i are typically cheap. Therefore, computing several crossing functions Q_i with the same input is not significantly slower than computing just one.

2.2 Continuous simulation in ODE mode

Since Modelica models are often stiff, implicit numerical time-stepping schemes are commonly used to integrate the ODE (5). This procedure requires, at each time step, the solution of one or more nonlinear systems of equations inside the integrator. For example, for a multistep method, such as the BDF methods implemented in Dassl, the system

$$\frac{1}{Ch_n}x_n - \hat{f}(t_n, x_n, d_n) = \text{old}(\hat{f}, x), \tag{7}$$

has to be solved for the next approximation x_n of the state. Here C denotes a method-dependent constant, h_n is the current step size, and $\text{old}(\hat{f}, x)$ is a linear combination of old $\hat{f}$-evaluation and x-approximations. Similar equations have to be solved to find the stages if using an implicit Runge–Kutta method, such as the Radau schemes (Hairer and Wanner, 1996).

The integrator equation (7) is typically solved by a quasi-Newton iterative method. In each iteration a linear system of equations is solved using the Jacobian

$$\hat{J} = \frac{1}{Ch_n}\mathrm{I} - \frac{\partial \hat{f}}{\partial x},$$

where I is the identity matrix. Even though the Jacobian is not updated each iteration, in fact not even with each time step, the evaluation of $\frac{\partial \hat{f}}{\partial x}$ is one of the major bottlenecks when simulating a large Modelica model.

The Jacobian $\hat{J}$ is normally approximated numerically using finite differences, which requires a large number of $\hat{f}$-evaluations. As previously mentioned, each $\hat{f}$-evaluation requires the solution of the algebraic loops (4). Thus, solving Equation (7) may involve treating nested nonlinear systems of equations.

To illustrate the problem, consider that the cost for constructing and factoring a Jacobian often grows superlinearly in the number of variables. This complexity depends on the sparsity structure of the Jacobian and what other optimizations are applied. In the worst case scenario the construction cost can grow quadratically and the factorization cost cubically. With this in mind, the cost for constructing (but not factoring) the integrator-Jacobian $\hat{J}$ can be approximated as

$$c_{\hat{J}} \approx \text{const.} \cdot n_x^{p_x - 1} \cdot \left(c_{\text{rem}}(n_x) + \sum_{j=1}^{n_G} n_i^{p_i} \right),$$

where $p_x \in [1,2]$ and $p_i \in [1,3]$ depend on how well sparsity and other optimizations can be utilized. The last factor is the cost of one $\hat{f}$-evaluation, where $c_{\text{rem}}(n_x)$ denotes the total cost of everything in the $\hat{f}$-evaluation, except the algebraic loops. This term typically grows with the number of states n_x. The factor $n_x^{p_x-1}$ is the number of $\hat{f}$-evaluations required. For certain models, the size n_i of some of the algebraic loops may be as large as the number of states n_x or even larger. Note that several optimizations are applied in Dymola to keep the exponents p_x and p_i small, with the aim to minimize the above cost. Especially when constructing the integrator Jacobian.

Further $\hat{f}$-evaluations are required to compute the residuals in the Newton iteration for the next step (7). Also at output points and events the model must be evaluated. However, for models like Nordic 44, considered in Section 4, the construction of integrator Jacobians dominate the simulation cost in ODE mode.

2.3 Continuous simulation in DAE mode

When integrating using the DAE mode proposed in this article, the algebraic loops are elevated and solved by the integrator. Rather than hiding the loops in Equation (5) they are handed to the integrator via the semi-explicit DAE formulation (6). When evaluating f the algebraic variables must be known prior to the evaluation. This means that, in DAE mode, the integrator has to handle also the iteration variables z.

One important benefit of the DAE mode approach presented in this paper is that the problem size is kept to a minimum by using the sorting and tearing information. The state vector consists of the vectors x and z. If the DAE solver was instead applied directly to the DAE (1) it would have had to solve for x and all of y.

Applying e.g. a multistep method to Equation (6) yields, in analog to Equation (7), the integrator equations

$$\begin{aligned} \frac{1}{Ch_n}x_n - f(t_n, x_n, z_n, d_n) &= \text{old}(f; x, z), \\ G(z_n; t_n, x_n, d_n) &= 0, \end{aligned} \tag{8}$$

to be solved for the next approximations x_n and z_n. The corresponding Jacobian needed for the quasi-Newton solution of these equations is

$$J = \begin{pmatrix} \frac{1}{Ch_n}\mathrm{I} - \frac{\partial f}{\partial x} & -\frac{\partial f}{\partial z_1} & -\frac{\partial f}{\partial z_2} & \cdots & -\frac{\partial f}{\partial z_{n_G}} \\ \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial z_1} & 0 & \cdots & 0 \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial z_1} & \frac{\partial G_2}{\partial z_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial G_{n_G}}{\partial x} & \frac{\partial G_{n_G}}{\partial z_1} & \frac{\partial G_{n_G}}{\partial z_2} & \cdots & \frac{\partial G_{n_G}}{\partial z_{n_G}} \end{pmatrix},$$

where we have considered the algebraic loops separate from each other to reveal the sparsity pattern that is given by construction.

The cost for a numerical approximation of the DAE mode Jacobian J differs in its structure from the cost of the ODE Jacobian $\hat{J}$. An approximation of the complexity

can be written as

$$c_J \approx \text{const.} \cdot \left(n_x + \sum_{j=1}^{n_G} n_i \right)^{p-1} \cdot \left(c_{\text{rem}}(n_x) + \sum_{j=1}^{n_G} n_i \right),$$

where $p \in [1,2]$ depends on how effectively sparsity can be used to minimize the number of f-evaluation. Since the state vector is longer in DAE mode the Jacobian is larger and more right-hand-side evaluations are typically needed. However, instead of solving the algebraic loops the DAE solver just inquires for their residuals G_i giving the smaller second factor in the cost. Comparing with the cost for approximating $\hat{J}$ one can conclude that computing one Jacobian in DAE mode is much faster when there are large algebraic loops with non-trivial sparsity structure ($p_i > 1$). Throughout this paper these are the type of models we consider.

2.4 Event handling in ODE mode

During integration the crossing functions (2) are monitored, applying special care to correctly handle multiple zero crossings in the same crossing function.

When zero crossings are detected in one or more crossing functions, the state x is interpolated and a root finding algorithm is applied to accurately find the time of the first zero crossing. Each of the crossing functions $\hat{Q}_i$ is scalar-valued and defines, together with Equation (5), a system of nonlinear equations

$$\begin{aligned} \dot{x} &= \hat{f}(t, P^x(t), d), \\ 0 &= \hat{Q}_i(t, \dot{x}, P^x(t), d), \end{aligned} \tag{9}$$

in the variables t and $\dot{x}$. The interpolation polynomial for the state x is denoted by $P^x(t)$. Note that the algebraic variables z have here been internalized in Q_i giving $\hat{Q}_i$, compare with f and $\hat{f}$ in Equations (5) and (6).

Noting that the algebraic variables are solved for with a high accuracy in ODE mode, the above root finding approach guarantees that the solution will be in a consistent state when a crossing is detected and an event iteration is started. See for example (Eich-Soellner and Führer, 2008, Chapter 6) for details on how to locate the first zero crossing using iterative methods.

2.5 Event handling in DAE mode

When simulating in DAE mode the integrator handles the algebraic variables z and approximates them to fit the supplied integrator tolerance. Similarly to the ODE case we may use the, now extended, state vector $(x;z)$ to monitor the crossing functions (2). Further, we may interpolate the whole extended state when solving for the first crossing

$$\dot{x} = f(t, P^x(t), P^z(t), d), \tag{10a}$$
$$0 = Q_i(t, \dot{x}, P^x(t), P^z(t), d). \tag{10b}$$

However, the integrator tolerance is several orders of magnitude larger than the tolerance for the solver of the algebraic loops in ODE mode. This means that the algebraic equations (4) will generally not be fulfilled at the time of crossing (10). The upcoming event iteration will then start in an inconsistent state; the algebraic equations not being accurately fulfilled. Severe problems may be experienced when Equations (1), (2), and (3) are to be simultaneously solved for a consistent restart state.[3]

Example 1. Consider the Modelica Standard Library model `EngineV6`, which is a multibody model of a V6 engine, see Figure 1. To compute the force generated by the combustion in an engine cylinder the piston velocity is monitored. An event is triggered and the integration is restarted when a piston velocity changes direction.

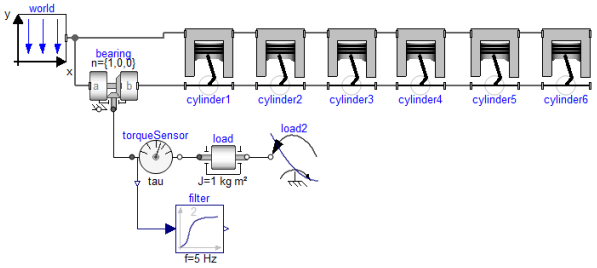

Figure 1. `EngineV6`, a multibody model of a V6 engine from the Modelica Standard Library.

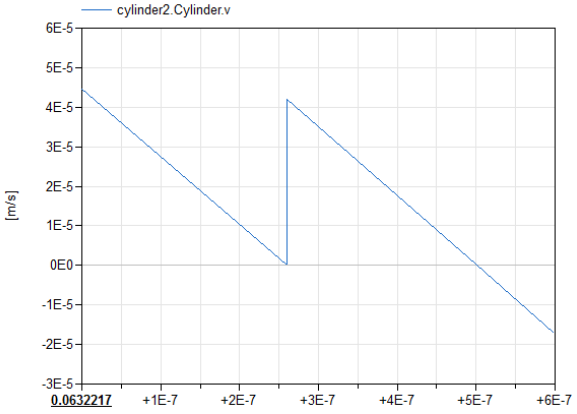

Figure 2. Inconsistent solution state causing empty events and problems to locate the correct time of the zero crossing.

Monitoring the crossing functions and locating the zero crossings according to Equation (10) gives the result of Figure 2, where the piston velocity of the second cylinder is used as an example. At time $t \approx 0.06322196$ Equation (10) signals for a zero crossing and an event is localized. Starting the event iteration, the crossing time t and interpolated state $P^x(t)$ are used as input and the algebraic variables y are solved for, cf. Section 2.1. Being an algebraic variable, the piston velocity `cylinder2.Cylinder.v`, is thus solved for with high

[3] Similarly, the state derivatives $\dot{x}$ may also be interpolated rather than computed from Equation (10a). However, the same problems as when interpolating z are to be expected.

DOI 10.3384/ecp19157491

accuracy. With the integrator error removed from the velocity it jumps up to its consistent value of approximately $4 \cdot 10^{-5}$ m/s. This renders the event empty since the corresponding crossing function $Q_i(t, \dot{x}, P^x(t), z, d)$ is again positive and no discrete variables were changed. The integration restarts and the process repeats a few times (cannot be seen in the figure since the subsequent velocity jumps are of very small scale). When sufficiently close to the correct crossing time $t \approx 0.06322220$ the event is at last correctly handled. △

To make sure that the solution is in a consistent state at zero crossings in DAE mode it is suggested by (Eich-Soellner and Führer, 2008, Section 6.3.8) to accurately solve the algebraic loops when crossing functions are evaluated. We will here adopt a slightly generalized and modified version of this idea. In our setting, and for each Q_i, we consider the following systems of equations

$$\dot{x} = f(t, P^x(t), z, d), \tag{11a}$$
$$0 = G(z; t, P^x(t), d), \tag{11b}$$
$$0 = Q_i(t, \dot{x}, P^x(t), z, d), \tag{11c}$$

to be solved when locating a zero-crossing. In contrast to the simple generalization (10) of the ODE case, where also the algebraic variables z were interpolated, we here only interpolate the original states x. The algebraic variables must be solved for using Equation (11b).

For each Q_i the system (11) has the unknowns t, $\dot{x}$, and z. These systems must be solved with high accuracy to guarantee that we get the correct crossing time and a consistent state for the event iteration. Additionally, solving the algebraic loops (11b) may be expensive, as we have previously discussed. However, the benefit of having the sorting and tearing information available here is that we do not have to solve for all the algebraic variables y simultaneously, rather the remaining can be computed from z. In Section 3.3 we will discuss further optimizations that can be applied to efficiently and accurately monitor and locate events in DAE mode.

3 Dymola DAE mode implementation

In Dymola and the 3DEXPERIENCE platform, DAE mode has been implemented for the solvers Dassl, Radau IIa, Sdirk34hw, and Esdirk of different orders. Thus offering a selection of both multistep and Runge–Kutta methods. With one of these solvers selected DAE mode is enabled by the command

```
Advanced.Define.DAEsolver = true.
```

In the current section we will present a few key features of this implementation. Most importantly the accurate and efficient handling of state events.

3.1 Reusing the simulation code

When initializing and when locating and resolving events the algebraic loops need to be accurately solved. For efficiency and robustness it is typically beneficial to use all the optimizations applied when generating ODE mode code, including e.g. sorting and tearing. This means that the code generated for ODE mode is needed also when integrating a hybrid system in DAE mode. To keep the simulation code simple and the code duplication to a minimum our implementation strategy is therefore to reuse the ODE code to the greatest extent possible.

As seen in Section 2.3 this strategy also allows us to minimize the size of the integrator equation (8) during continuous simulation in DAE mode. However, the drawback here is that tearing may cause fill-ins, i.e. the reduced nonlinear system $G_i = 0$ may be less sparse than the original system. In some cases this may even make simulations slower (Braun et al., 2017). On the other hand, it is concluded by (Magnusson, 2016) that tearing typically is beneficial for DAEs resulting from hierarchical Modelica models. One may additionally gain in efficiency by taking care to reduce fill-ins when tearing, especially for large models. To which extent tearing should be used and how it should be applied is considered out of scope of this paper.

As discussed in Section 2 the only difference between ODE and DAE mode continuous simulation is how the algebraic loops are handled. In ODE mode the algebraic loops (4) are solved for the algebraic variables z during the $\hat{f}$-evaluations. By small changes in the simulator code that handles the algebraic loops, these loops can easily be modified to instead take z as an input from the integrator. The input can then be used to compute the residuals $G(z)$, which are returned to the integrator for correction. All this without any need to change the code generation or the generated simulation code itself.

3.2 Utilizing the Jacobian structure

So far no changes are required to the generated simulation code. However, there is one piece of auxiliary information needed for efficient simulations.

The sparsity pattern of the integrator Jacobian is analyzed by Dymola when constructing the simulation code. Knowing all explicit dependencies of the functions $\hat{f}$ on the variables x (respectively f on $(x; z)$), Dymola can reduce the number of function evaluations required to construct a numeric Jacobian. Several columns can be grouped together and computed at the same time by utilizing column independencies.

Since the dependencies change in DAE mode the ODE mode analysis can not be reused. The DAE Jacobian J is larger and typically more sparse, cf. (Braun et al., 2017, Section 2.2). For example, partial explicit dependencies in each algebraic loop can be taken into account when constructing the sparsity pattern for the DAE mode Jacobian. In contrast, in ODE mode, where the algebraic loops are solved, all the iteration variables z_i for each loop depend on all of the loop inputs. To summarize, this enables Dymola to be more aggressive when constructing column groups in DAE mode.

Moreover, due to the increased size and sparsity of the

integrator Jacobian, the benefits of using sparse linear algebra for storing and factorization are even greater. In Dymola multithreaded SuperLU (Li, 2005) is used for this task and is fully compatible with DAE mode simulations.

3.3 Efficient and accurate event localization

In Section 2.5 we demonstrated that correctly monitoring and locating crossings can be expensive in DAE mode. Interpolating the algebraic variables z led to problems and instead the full equations (11) had to be considered. Additionally, to avoid having to discard time steps or output points it is important to evaluate the crossing functions often during continuous integration, normally after each time step. Considering that each solution of the equations may be expensive, the aggregated cost may become a major bottleneck for DAE mode simulations of hybrid models. However, there are several optimizations that can be performed to considerably shorten the time needed for event localization.

First assume, for the sake of simplicity, that there is only one crossing function Q and consider how zero crossings are typically localized in ODE mode. It is straightforward to rewrite Equation (9) as an equation

$$0 = \hat{Q}\big[t,\ \hat{f}(t,P^x(t),d),\ P^x(t),\ d\big],$$

in the single, scalar variable t. This equation can be efficiently solved with an iterative method, e.g. using regula falsi variants like the safeguard techniques (Eich-Soellner and Führer, 2008, Section 6.3.2).

The same techniques can be adopted in the DAE case to efficiently solve the system (11). This results in the nested systems of equations

$$0 = Q\Big[t,\ \ f\Big(t,P^x(t),G^{-1}\big(0;t,P^x(t),d\big),d\Big),\\ P^x(t),\ \ G^{-1}\big(0;t,P^x(t),d\big),\ \ d\Big].$$

In the outer equation the time of the crossing t is the only unknown. Given an approximation of t the polynomial P^x can be evaluated. Using this, the algebraic variables can be solved for from $0 = G(z)$. Then $\dot{x}$ can be computed and finally the crossing function residual. This nonlinear equation in t can be solved by applying the same iterative root finding techniques as in ODE mode.

Indeed, as the algebraic loops are solved also when $\hat{f}$ is evaluated the above described procedures for (9) and (11) are equivalent. This means that the same code can be used for event localization in ODE and DAE mode. The only difference is that, in the latter case the simulator code handling the algebraic loops must be told to solve them, and to do this with high accuracy. After the event is fully handled the code must again be told to only compute the residuals of the loops for continuous DAE mode simulation.

Handling several crossing functions at the same time is neither significantly more expensive, nor more difficult, cf. Section 2.1. Thus, the above discussed solution technique for crossing equations is easily generalized to several crossing functions.

Finally, and perhaps most importantly, we note that accurately solving the algebraic loops is only important when finding the correct crossing time and during the event iteration. During continuous simulations, and when no crossing function is close to zero, it is enough to consider the more direct formulation first discussed in Section 2.5. That is, we use the integrator approximations of both x and z and after each time step we evaluate

$$\begin{aligned} \dot{x} &= f(t,x,z,d),\\ c &= Q(t,\dot{x},x,z,d). \end{aligned} \tag{12}$$

If any of the variables c_i is close to zero we must switch to the accurate crossing function handling (11) so the correct crossing time can be located.

With these optimizations Dymola can accurately and efficiently locate and resolve state events. The algebraic loops must only be solved to a high accuracy when closing in on a crossing, when localizing the crossing, and when resolving the event. Typically, only on the order of ten solutions per state event is required, cf. Section 4.2. A remaining problem is how to efficiently handle the situation where a crossing function is close to zero throughout most of the simulation, but never crosses.

4 Application Example – Nordic 44

To verify the efficiency of the Dymola DAE mode implementation we here perform experiments with the Nordic 44 power grid model (Vanfretti et al., 2017).

4.1 Model description and test cases

Nordic 44 consists of 44 buses, 61 controlled generators, 67 lines, and 43 loads, which model the Nordic grid, see Figure 3. The model is part of the Open-Instance Power System Library (OpenIPSL), a Modelica library for power system dynamic analysis (Baudette et al., 2018).

The DAE representation (6) of Nordic 44, given by the symbolic transformations in Dymola 2019 FD01, consists of $n_x = 1013$ states and $n_G = 47$ torn algebraic loops. The first loop has $n_1 = 448$ iteration variables and the remaining have one each.

We will consider three different fault scenarios. Models for all of them have been added to the OpenIPSL library. They can also be found in the supplementary material to this article and at a dedicated GitHub repository[4]. The first two scenarios are reproductions of the experiments performed by (Vanfretti et al., 2016, Section 3). There, the second order Runge–Kutta scheme Rkfix2 was used with the fixed time step $h = 0.01$ s.

For the first scenario we introduce a line opening between Bus 5103 and Bus 5304 to occur at $t = 2$ s. The voltage for Bus 5304 is plotted in Figure 4, given as a result of the Dassl DAE mode simulation. To be able to

[4] GitHub: 2019_Modelica_Conf_DAESolvers4LargeHybridModels, https://github.com/ALSETLab/2019_Modelica_Conf_DAESolvers4LargeHybridModels

DOI 10.3384/ecp19157491

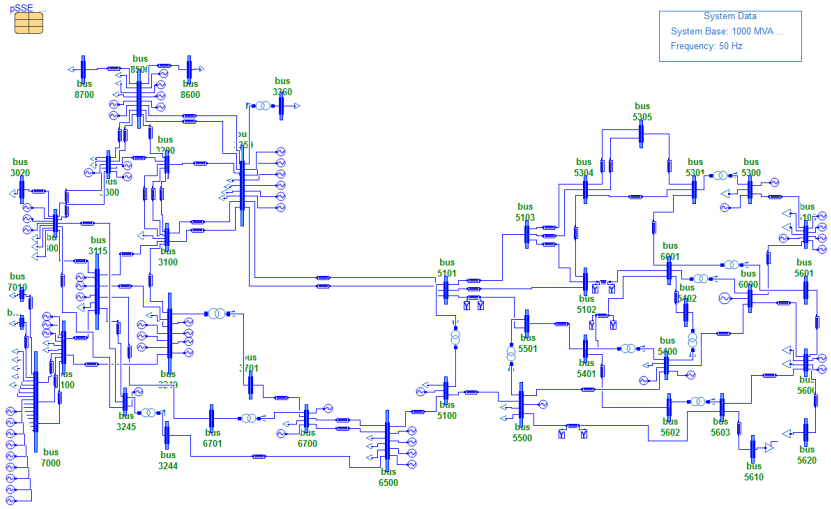

Figure 3. The Nordic 44 grid model.

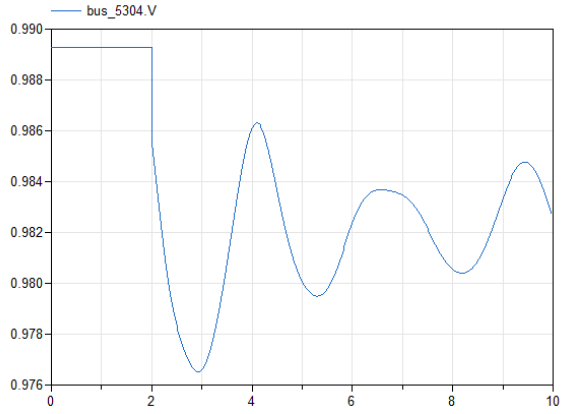

Figure 4. Voltage for Bus 5304 during a line fault between Bus 5103 and Bus 5304 occurring at $t = 2$ s.

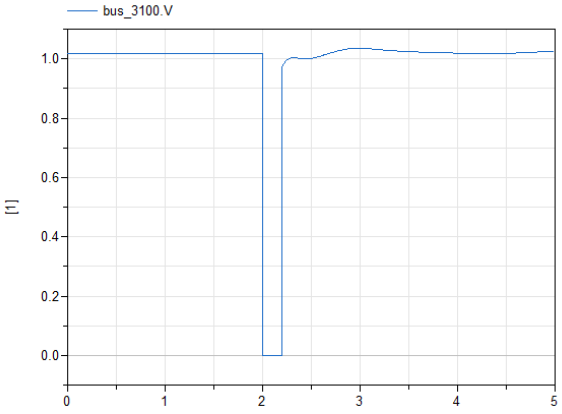

Figure 5. Voltage for Bus 3100 during a fault in this bus between $t = 2$ s and $t = 2.2$ s.

compare CPU-times with the Rkfix2 ODE mode simulations the tolerance 10^{-6} was chosen for Dassl. With this tolerance the error estimates of the two solvers are approximately the same.

In the second scenario a bus fault is instead simulated. At time $t = 2$ s Bus 3100 short-circuits and connects to the ground, with very small impedance, for 0.2 s. The simulated bus voltage is plotted in Figure 5. For this model, the Dassl tolerance 10^{-4} gives errors comparable to those of Rkfix2.

The final scenario also considers a bus fault, this time in Bus 5603. However, the model is also extended to include an additional generator connected to Bus 5610, see Figure 6. Compared to the other generators in the Nordic 44 model, a different excitation control system (IEEE Type AC2A Excitation System) is used. Depending on the generator field voltage, different control modes are used in the excitation system to change its outputs. To switch between the modes state events are required. This extended model has $n_x = 1313$ states and $n_G = 65$ algebraic loops of sizes $n_1 = 498$ and $n_2 = \cdots = n_{65} = 1$. Due to the extension the default Nordic 44 initial conditions do not define a steady state. To get close to steady state, a simulation is first performed until $t = 60$ s. Then, the bus fault occurs between $t = 61.05$ s and $t = 61.15$ s. The simulated generator field voltage (EFD) is plotted in Figure 7 together with the unmodified control output (EFD1). For comparable numerical errors the tolerance $5 \cdot 10^{-5}$ is used for Dassl.

4.2 Efficiency experiments

The CPU-times required to simulate the three different scenarios are listed in Table 1. All simulations in this section were run in Dymola 2019 FD01, with default settings if nothing else is specified, and using Visual Studio 2015 for model compilation. An ordinary Windows 7 (64-bit) laptop computer (Intel Core i7-6820HQ, 16 GB RAM) has been used for all experiments, including the reproduction of the Rkfix2 experiments ($h = 0.01$ s), reported by (Vanfretti et al., 2016). As mentioned above, the Dassl tolerances have been tuned to give comparable numerical errors between the two solvers.

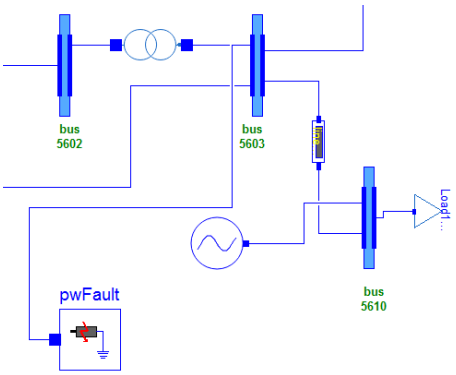

Figure 6. Nordic 44 extended with an extra generator featuring several control modes. The generator is connected to Bus 5610 and the fault occurs in Bus 5603.

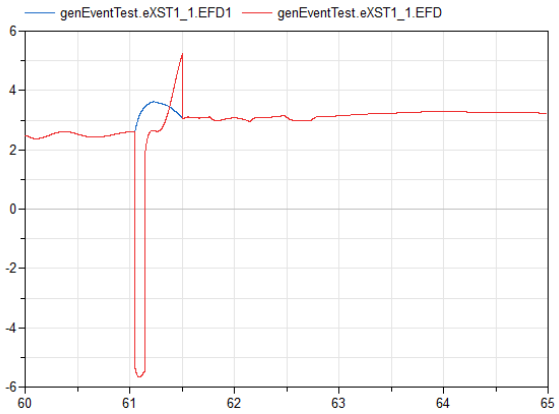

Figure 7. Generator field voltage (EFD) and the unmodified excitation control output (EFD1) for the extra generator at Bus 5610.

We observe that several orders of magnitude in simulation speed-up was gained by running Dassl in DAE mode; the construction of many expensive integrator Jacobians $\hat{J}$ makes the ODE mode times uncompetitive. We remind the reader that the Nordic 44 model has been specifically chosen as an example in this paper since it is difficult to handle efficiently in ODE mode. For most Modelica models ODE mode is more robust and as efficient. Finally, comparing with Rkfix2 we conclude that using an explicit method to altogether avoid integrator Jacobians does not pay off for these simulations.

All of the power grid faults are triggered at specific times, that is by time events. For the first scenario that is also the only event. This explains the very fast simulation time as the large algebraic loops only need to be solved during initialization and during the event iteration of the time event. In total each loop is solved only four times.

Table 1. CPU-times for the three Nordic 44 fault scenarios.

Fault	*Rkfix2*	*Dassl*	
	ODE mode	*ODE mode*	*DAE mode*
Line	587 s	2 015 s	4.21 s
Bus 3100	270 s	7 810 s	33.7 s
Bus 5603	344 s	49 800 s	121 s

However, the remaining two scenarios have several state events that must be located and resolved, namely 14 and 26, respectively. When the fault is triggered in Bus 3100, the excitation control system inside several of the generators reach their maximum limit. To prevent wind-up in the integral controllers their internal states are reset using state events. Since the desired set points cannot be reached the controllers continue to wind-up and several resets are required, cf. Figure 8. Even though reminiscent of Figure 2, the saw blade shape here represents a correct solution of the model. This can be easily verified by ODE mode simulations. Indeed, the fact that the accumulation of events is accurately handled when closing in on $t = 2.1$ s confirms the soundness of the event handling approach proposed by (Eich-Soellner and Führer, 2008) and used in this paper.

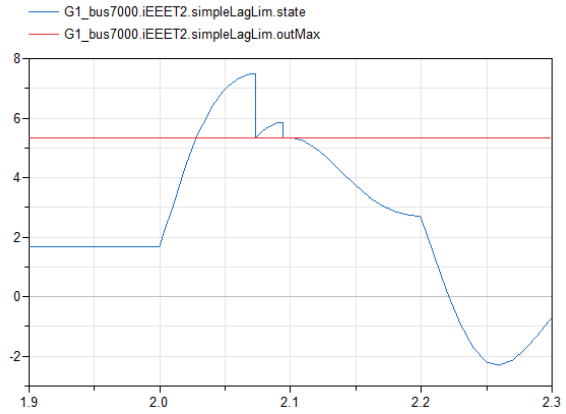

Figure 8. The fault in Bus 3100 triggers saturation in several generator excitation controllers. State events are issued to reset the internal controller state, as exemplified here with one of the generators connected to Bus 7000.

A similar analysis can be made of the third scenario with the extended Nordic 44 model. But there the state events instead represent switches between excitation control modes in the extra generator. Note that the higher number of state events and the larger algebraic loop are reflected by the longer simulation time.

As a final experiment we demonstrate the effect of one of the event handling optimizations presented in Section 3.3. Consider again the second scenario with the fault in Bus 3100. We rerun the Dassl DAE mode simulation, but during the simulation we monitor Equation (11) rather than Equation (12). This means that the algebraic loops (4) are solved throughout all of the simulation, not only when closing in on a zero crossing. This results in a run time of 197 s to be compared with 33.7 s for the efficient DAE mode implementation. In the former case the algebraic loops were solved 918 times, whereas in the

DOI 10.3384/ecp19157491

latter case only 143 times.

Finally, we compare with the simulations performed by (Vanfretti et al., 2016) of equivalent models using the domain-specific simulation tool PSS/E. On a computer slightly faster than the one used for this paper the first two scenarios run in 5 s, respectively 4 s. *We conclude that the Dymola DAE mode performance is competitive against the industry state of the art.* In fact, even faster for the first scenario. At the same time, looking at the second scenario, we note that there is room for further efficiency improvements in the event handling.

5 Additional DAE mode challenges – an outlook

Other than the accurate handling of events, DAE mode simulations pose a few additional challenges not experienced to the same extent during ODE mode simulations.

5.1 Robustness and discontinuities

The nonlinear system of equations (8) to be solved inside the integrator is larger in DAE mode. Solving for both the states x and the algebraic variables z simultaneously may cause robustness problems. Compare with ODE mode where the algebraic equations (4) are solved one at a time, separate from each other, as part of each $\hat{f}$-evaluation. The nonlinear equation solvers that treat the algebraic loops may be optimized for this purpose. For example, when solving algebraic loops with only one iteration variable even major problems, such as a singular Jacobian, often do not pose an insurmountable threat. When this singularity becomes part of a large system of equations it becomes a problem that is much more difficult to handle.

When interpolated values are of interest, as when monitoring events, and the DAE is of index 1, it is recommended by (Brenan et al., 1996, Section 5.4.2) to apply error control also to the algebraic variables z. However, this may cause failed simulations when algebraic variables are discontinuous in time, consider e.g. van der Pol's equation

$$\begin{aligned} \dot{x} &= -z, \\ 0 &= x - \left(\frac{z^3}{3} - z \right), \end{aligned} \tag{13}$$

(Hairer and Wanner, 1996, Section VI.1). Discontinuities may also arise when using the `noEvent` operator.

5.2 ODE-powered DAE mode simulation

With the strategy of using the same generated simulation code both in ODE and DAE mode an opportunity opens up to handle these DAE mode specific problems. The idea is simple and based on the fact that we can readily switch between the modes: we integrate in DAE mode until a problem is encountered. Then we switch to ODE mode and integrate past the problem. When it is deemed fine to continue in DAE mode, the switch back is made.

Note that switching to ODE mode comes with a considerable cost. In contrast to the event handling strategy previously discussed, we here want to perform continuous integration in ODE mode. This requires the construction of ODE Jacobians $\hat{J}$, which is expensive. An important goal for any implementation of this idea is probably to keep the number of ODE Jacobians to a minimum. If additionally the rest of the simulation runs smoothly in DAE mode it may still be considerably more efficient than plain ODE mode simulation.

We have made a prototype implementation of this idea using Dassl. The algorithm is simple: when the integrator gives up in DAE mode we do not stop the simulation, but rather, switch to ODE mode. While in ODE mode we allow for one Jacobian computation and simulate with this until Dassl asks for a second. Instead of computing it, we switch back to DAE mode and continue simulation until the integrator gives up again or the simulation terminates.

Example 2. As discussed above, when simulating van der Pol's equation (13) in DAE mode it normally fails when closing in on a discontinuity in z. The error estimate in this variable becomes large and cannot be made smaller by shorter time steps. However, using the prototype implementation introduced here we can successfully simulate past the discontinuities by temporarily switching to ODE mode, see Figure 9. For this simulation Dassl required 132 Jacobian-evaluations in DAE mode and only two Jacobian-evaluations in ODE mode. △

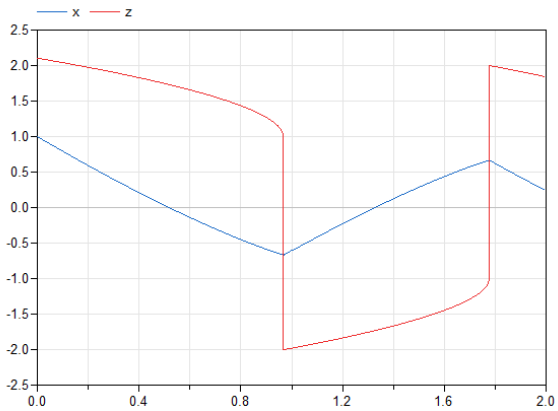

Figure 9. DAE mode solution of van der Pol's equation (13) using temporary switches to ODE mode to handle the discontinuities in z.

Of course, when simulating the van der Pol's equation there is no efficiency benefit in using DAE mode. However, for a production-level implementation that can handle large-scale models, the prototype implementation needs several improvements and tuning. The most important question is to decide when to switch, especially when to switch back to DAE mode. The simple prototype implementation presented above will often switch back too early. A more careful analysis of the state of the problem should probably be performed before switching back. Further questions involve how to best reinitialize the simulation and with which step size.

6 Conclusion

We have discussed and implemented a DAE mode strategy for hybrid DAEs based on the idea of one common code generation for ODE and DAE mode. By applying all of the symbolic transformations and optimizations we get an index-1 DAE of minimal size. The integrator only needs to handle the original states and the iteration (tearing) variables of the algebraic loops. Even though a smaller system is typically faster to simulate the tearing algorithm may cause fill-in making the reduced system more dense. How to best apply tearing in this context deserves deeper analysis but is out of scope of this paper.

Our DAE mode approach made it possible to accurately and efficiently handle state events, with minimal footprint on the generated code. To localize these events we have extended from results known in the literature to fit our Modelica context. Suggestions for optimizing the root finding were also discussed and implemented.

With these optimizations the algebraic loops only need to be solved with high accuracy during initialization, when closing in on a state event, when localizing it, and when resolving it. Most importantly, solving the loops is not required when constructing the integrator Jacobian or other evaluations of the dynamics. Typically, only on the order of ten solutions per state event are required. We argue that this is an acceptable cost for models with a moderate number of state events. For example, compare with the large number of loop solutions that are required to just construct the integrator Jacobian in ODE mode, cf. Section 2.2.

Therefore, DAE mode simulations can be vastly more efficient for models with large algebraic loops. As exemplified by simulations of the Nordic 44 model of the Nordic power grid, where orders of magnitude in simulation speed were gained. The measured simulation times are competitive with domain-specific, state-of-the-art simulation tools that have been optimized for more than three decades.

The presented DAE mode was made available in Dymola 2019 and 3DEXPERIENCE 2019x for a broad selection of numerical integrators. The implementation also features detailed DAE mode sparsity pattern analysis and is fully compatible with Dymola sparse solvers.

Acknowledgments

The authors thank Ricardo Rincon Ballesteros (Universidad Nacional de Colombia) for constructing the third simulation scenario of the application section specifically for this publication.

The authors also thank their colleagues for valuable feedback on the paper drafts.

The work of L. Vanfretti was supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, and in part by the CURENT Industry Partnership Program.

References

Maxime Baudette, Marcelo Castro, Tin Rabuzin, Jan Lavenius, Tetiana Bogodorova, and Luigi Vanfretti. OpenIPSL: Open-instance power system library — update 1.5 to "iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations". *SoftwareX*, 7:34–36, 2018.

Willi Braun, Francesco Casella, and Bernhard Bachmann. Solving large-scale modelica models: New approaches and experimental results using OpenModelica. In *Proceedings of the 12th International Modelica Conference*, pages 557–563. Linköping University Electronic Press, 2017.

Kathryn E. Brenan, Stephen L. Campbell, and Linda R. Petzold. *Numerical Solution of Initial-Value Problems in Differential–Algebraic Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996.

Francesco Casella. Simulation of large-scale models in Modelica: State of the art and future perspectives. In *Proceedings of the 11th International Modelica Conference*, pages 459–468. Linköping University Electronic Press, 2015.

Francesco Casella, Andrea Bartolini, Simone Pasquini, and Luca Bonuglia. Object-oriented modelling and simulation of large-scale electrical power systems using Modelica: A first feasibility study. In *IECON 2016 – 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6298–6304, 2016.

Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag, Berlin, Heidelberg, 2006.

Edda Eich-Soellner and Claus Führer. *Numerical methods in multibody dynamics*. Teubner, 2008.

Hilding Elmqvist and Martin Otter. Methods for tearing systems of equations in object-oriented modeling. *ESM'94 European Simulation Multiconference*, pages 326–332, 1994.

Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Parallel model execution on many cores. In *Proceedings of the 10th International Modelica Conference*, pages 363–370. Linköping University Electronic Press, 2014.

Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems*, volume 14. Springer-Verlag Berlin Heidelberg, second edition, 1996.

Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, pages 363–396, 2005.

Filip Jorissen, Michael Wetter, and Lieve Helsen. Simulation speed analysis and improvements of Modelica models for building energy simulation. In *Proceedings of the 11th International Modelica Conference*, pages 59–69. Linköping University Electronic Press, 2015.

Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, 2005.

Fredrik Magnusson. *Numerical and Symbolic Methods for Dynamic Optimization*. PhD thesis, Department of Automatic Control, Lund University, 2016.

Hans Olsson, editor. *Modelica – A Unified Object-Oriented Language For System Modeling: Language Specification*. Modelica Association, 2017. Version 3.4.

Hans Olsson, Sven Erik Mattsson, Martin Otter, Andreas Pfeiffer, Christoff Bürger, and Dan Henriksson. Model-based embedded control using Rosenbrock integration methods. In *Proceedings of the 12th International Modelica Conference*, pages 517–526. Linköping University Electronic Press, 2017.

Anton Schiela and Hans Olsson. Mixed-mode integration for real-time simulation. *Proceedings of Modelica 2000 Workshop*, pages 69–75, 2000.

Bernhard Thiele, Martin Otter, and Sven Erik Mattsson. Modular multi-rate and multi-method real-time simulation. In *Proceedings of the 10th International Modelica Conference*, pages 381–393. Linköping University Electronic Press, 2014.

Luigi Vanfretti, Tin Rabuzin, Maxime Baudette, and Mohammed Murad. iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations. *SoftwareX*, 5:84–88, 2016.

Luigi Vanfretti, Svein H. Olsen, V.S. Narasimham Arava, Giuseppe Laera, Ali Bidadfar, Tin Rabuzin, Sigurd H. Jakobsen, Jan Lavenius, Maxime Baudette, and Francisco J. Gómez-López. An open data repository and a data processing software toolset of an equivalent Nordic grid model matched to historical electricity market data. *Data in Brief*, 11:349–357, 2017.

DOI
10.3384/ecp19157491