Technische Universität Berlin
Institut für Mathematik

Regularization and Numerical Simulation of Dynamical Systems Modeled with Modelica

Randolf Altmeyer* ‡ and Andreas Steinbrecher† ‡

Preprint 2013/29

Preprint-Reihe des Instituts für Mathematik
Technische Universität Berlin

Report 2013/29 September 2013

Quasi-linear differential-algebraic equations (DAEs) are essential tools for modeling dynamical processes, e.g. for mechanical systems, electrical circuits or chemical reactions. These models are in general of higher index and contain so called hidden constraints which lead to instabilities and order reductions during numerical integration of the model equations.
In this article we consider dynamical systems modeled with quasi-linear DAEs of higher index using Modelica. We investigate a regularization of model equations in the Modelica framework in view of an efficient and robust numerical simulation.
We will present an approach for numerical integration of quasi-linear DAEs which is based on overdetermined DAEs obtained from dynamical models modeled with Modelica.

AMS(MOS) subject classification: 34A09, 65K05, 65L80

Keywords: dynamical systems, differential-algebraic equations, regularization, numerical integration, Modelica, QUALIDAES, MO2FOR

Authors address:

Randolf Altmeyer
Institut für Mathematik
Humboldt Universität zu Berlin
10099 Berlin, Germany
randolf.altmeyer@gmail.com

Andreas Steinbrecher
Institut für Mathematik, MA 4–5,
Technische Universität Berlin
Str. des 17. Juni 136
10623 Berlin, Germany
steinbrecher@math.tu-berlin.de

Regularization and Numerical Simulation of Dynamical Systems Modeled with Modelica

Randolf Altmeyer*‡ and Andreas Steinbrecher†‡

September 30, 2013

Abstract

Quasi-linear differential-algebraic equations (DAEs) are essential tools for modeling dynamical processes, e.g. for mechanical systems, electrical circuits or chemical reactions. These models are in general of higher index and contain so called hidden constraints which lead to instabilities and order reductions during numerical integration of the model equations.
In this article we consider dynamical systems modeled with quasi-linear DAEs of higher index using Modelica. We investigate a regularization of model equations in the Modelica framework in view of an efficient and robust numerical simulation.
We will present an approach for numerical integration of quasi-linear DAEs which is based on overdetermined DAEs obtained from dynamical models modeled with Modelica.

Keywords: dynamical systems, differential-algebraic equations, regularization, numerical integration, Modelica, QUALIDAES, MO2FOR

AMS(MOS) subject classification: 34A09, 65K05, 65L80

1 Introduction

The modeling of dynamical processes, like electrical circuits, multibody systems, thermal systems or flow problems often leads to model equations of differential equations with algebraic constraints, so called *differential-algebraic equations* (DAEs). For modeling such dynamical processes the modeling language Modelica [4] is a common and useful tool. Modelica is an object-oriented language for complex systems containing several subsystems, e.g., mechanical, electrical, or hydraulic subcomponents. There exists a large variety of modeling and simulation tools, e.g., OpenModelica[1], Dymola[2], MapleSim[3], SimulationX[4].

*Institut für Mathematik, Humboldt Universität zu Berlin, 10099 Berlin, Germany; randolf.altmeyer@gmail.com.

†Institut für Mathematik, TU Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany; steinbrecher@math.tu-berlin.de.

‡The author's work was supported by the ERC Advanced Grant "Modeling, Simulation and Control of Multi-Physics Systems" MODSIMCONMP

[1] https://www.openmodelica.org

[2] http://www.dymola.com

[3] http://www.maplesoft.com/products/maplesim/

[4] http://www.simulationx.com

To investigate the dynamical behaviour the model equations have to be integrated. The solution of such DAEs is restricted to satisfy algebraic constraints contained in the DAE. But in general not all constraints are stated in an explicit way in the DAE. In general there exist so called *hidden constraints* leading to a higher index. Therefore, the numerical treatment of DAEs is nontrivial in general and more complicated than for ordinary differential equations (ODEs), see [2, 10, 12, 13]. Among the effects arising from hidden constraints are instabilities, convergence problems or inconsistencies, for instance.
A way out of the dilemma of hidden constraints in model equations is to *remodel* or regularize the dynamical system, see for instance [6, 9, 12, 13, 19, 20].

The dynamical system is often modeled as initial value problem for *quasi-linear DAEs*

$$E(x,t)\dot{x} = k(x,t), \tag{1}$$

on the domain $\mathbb{I} = [t_0, t_f]$ with initial values $x(t_0) = x_0 \in \mathbb{R}^n$, where $E \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{n,n})$ is called the *leading matrix* of the quasi-linear DAE and $k \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^n)$ its *right-hand side*. Furthermore, $x : \mathbb{I} \to \mathbb{R}^n$ represents the *unknown variables*. The equations are assumed to be uniquely solvable for consistent initial values and nonredundant. In this article we restrict our considerations to quasi-linear DAEs (1) where we assume that the rank of the leading matrix E is constant for all $(x,t) \in \mathbb{R}^n \times \mathbb{I}$ and, furthermore, the rank of the partial derivatives of the (hidden) constraints with respect to x is constant for all $(x,t) \in \mathbb{R}^n \times \mathbb{I}$. Note that in general these assumptions are not necessary and can be relaxed, see [18].
As a special case of the quasi-linear DAE (1), we will investigate *semi-implicit DAEs* of the form

$$\begin{aligned} E_1(x,t)\dot{x} &= k_1(x,t), && \text{(2a)} \\ 0 &= k_2(x,t) && \text{(2b)} \end{aligned}$$

with $E_1 \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m_1,n})$, $k_1 \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m_1})$ and $k_2 \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m_2})$. We assume $m_1 + m_2 \geq n$.
In the following, for a differentiable time dependent function x we denote the total derivative with respect to t by $\dot{x}(t) = \mathrm{d}x(t)/\mathrm{d}t$. For a differentiable function f depending on x the (partial) derivative of $f(x)$ with respect to x is denoted by $f_{,x}(x) = \frac{\partial}{\partial x} f(x)$. The same notation is used for differentiable vector functions.

In this article we will discuss an approach for regularization and numerical integration of dynamical systems modeled with Modelica as illustrated in Figure 1. The first step consists of regularizing the model equations, via overdetermined formulations including all hidden constraints, see Section 2, while the second step performs the subsequent robust and efficient numerical integration using the Runge-Kutta method RADAU IIa of order 5, see Section 3. The combined approach of regularization and subsequent numerical integration is implemented in the software package `MO2FOR/QUALIDAES` which we present in Section 4. The applicability of `MO2FOR/QUALIDAES` for the numerical treatment of model equations provided in Modelica language is demonstrated at the example of a simple pendulum and of a car axis in Section 5.

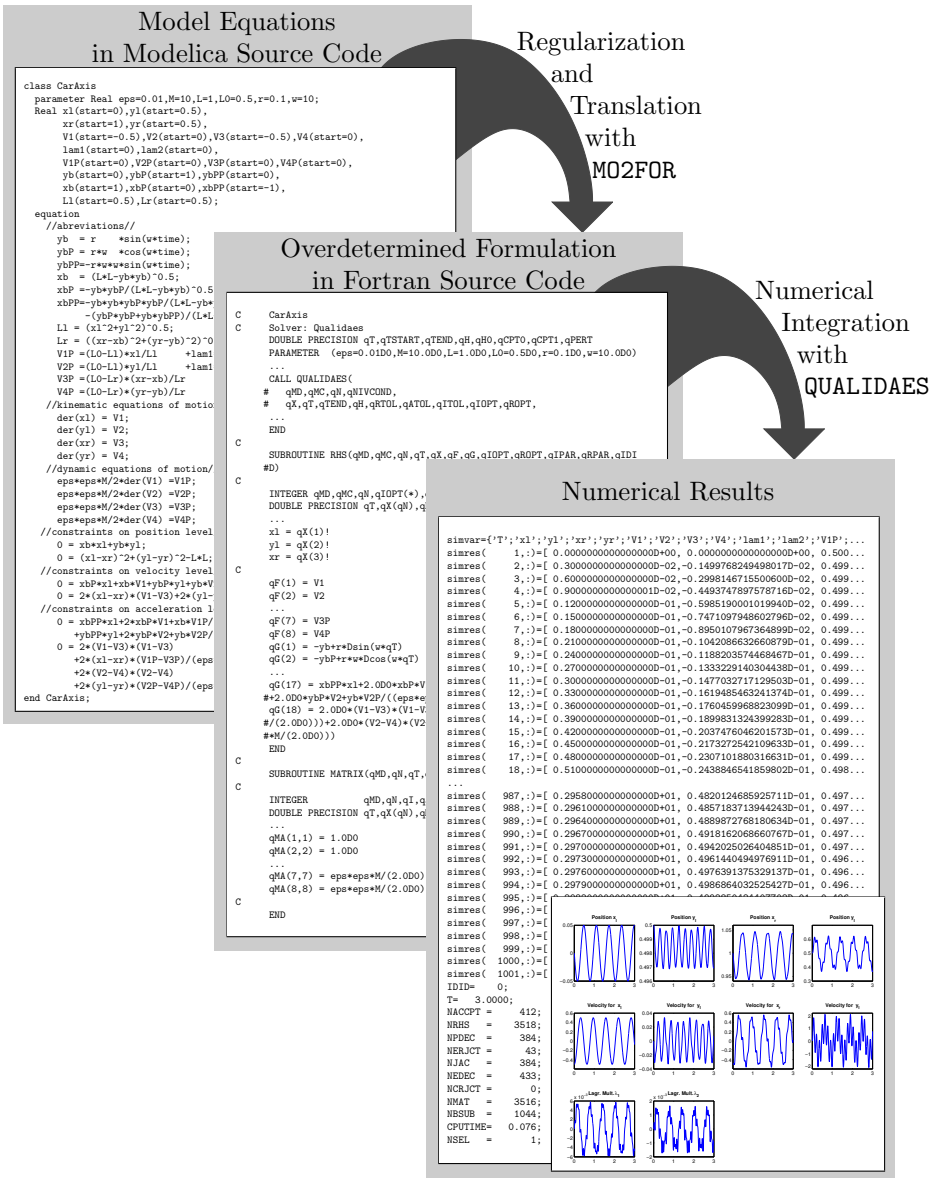

Figure 1: Scheme MO2FOR - QUALIDAES

2 Regularization using Overdetermined Formulations

In general, the solution of higher index DAEs is not only restricted by explicit constraints, as for instance in (2b), but also by so called hidden constraints which are not explicitly stated as equations, see [18]. These constraints impose additional consistency conditions on the initial values and on the solution, thereby causing problems for direct numerical integration of DAEs of higher index, see [1, 2, 6, 8, 10, 11, 12, 15, 16].

In the following we will investigate a regularization of quasi-linear DAEs (1) to overcome the difficulties in numerical simulation as mentioned above. Such a regularization should end up with a *regularized DAE* which

R1) has the same solution set as the original DAE (1) and where

R2) the regularized DAE contains no hidden constraints, i.e., all (hidden) constraints in the original DAE (1) have to be explicitly stated.

Two DAEs have the same solution set if every solution of one DAE solves also a second DAE and vice versa.
In [18, 19, 20] procedures are proposed to determine the hidden constraints of quasi-linear DAEs (1). Let us denote the hidden constraints by

$$0 = h(x,t) \tag{3a}$$

with

$$h(x,t) = \begin{bmatrix} h^0(x,t) \\ \vdots \\ h^{\nu_c}(x,t) \end{bmatrix} \tag{3b}$$

where h and h^i are vector functions $h(x,t) : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^{n_C}$ with $n_C = \text{rank}(h_{,x}(x,t))$ for all $(x,t) \in \mathbb{M}$ and $h^i(x,t) : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^{n_C^i}$, $i = 0, ..., \nu_c$, respectively, with $n_C = \sum_{i=0}^{\nu_c} n_C^i$. The set

$$\mathbb{M} = \{(x,t) \in \mathbb{R}^n \times \mathbb{I} \ : \ 0 = h(x,t)\},$$

denotes the *set of consistency* and restricts the set of solutions. Furthermore, ν_c denotes the *maximal constraint level* of the DAE, i.e., the highest level of existing (hidden) constraints, see [18, 19, 20]. Here, $0 = h^i(x,t)$ denotes the hidden constraints of level i, and, in particular, $0 = h^0(x,t)$ corresponds to the explicitly stated constraints in the quasi-linear DAE (1).
Adding the hidden constraints to the quasi-linear DAE (1) leads to the overdetermined DAE

$$E(x,t)\dot{x} = k(x,t), \tag{4a}$$
$$0 = h(x,t). \tag{4b}$$

The overdetermined formulation (4) is equivalent to the original DAE (1) in the sense that both have the same solution set. Furthermore, the overdetermined formulation (4) has the advantage that it satisfies the requirements R1) and R2) of a regularization. A further advantage of the overdetermined formulation (4) is the fact that it is not necessary to apply analytical manipulations for the determination of a square and for consistent initial values uniquely solvable system of DAEs and, furthermore, that the unknowns x are unchanged, i.e., a transformation of the state variables is not necessary and the number of unknowns is not increased. Therefore, this is the kind of formulation we are interested in.
However, on the other hand, in the Modelica framework it is difficult or impossible to model and integrate overdetermined systems like (4) since most modeling and integration tools are based on square and for consistent initial values

uniquely solvable systems. To overcome this problem it would be possible to transform the overdetermined formulation (4) into a square and for consistent initial values uniquely solvable system according to the regularization criteria R1) and R2).
As already mentioned the overdetermined DAE (4) and the original DAE (1) are equivalent. Therefore, there exist redundant equations in the overdetermined formulation. The number of degrees of freedom $n_D = n - n_C \leq n$ of the dynamical system is equal to the dimension of the set of consistency $\mathbb{M}$. Therefore, there exist $n - n_D$ equations in the original DAE (1), i.e., in the upper part of (4), which are made redundant by the constraints. These redundancies can be eliminated without changing the solution set by using a so called *dynamic selector* $S_D(x,t)$ which is defined as a matrix function of size $n_D \times n$ such that

$$\begin{bmatrix} S_D(x,t)E(x,t) \\ h_{,x}(x,t) \end{bmatrix} \text{ is nonsingular for all } (x,t) \in \mathbb{M}.$$

Therefore, we could use the so called *projected strangeness-free formulation*

$$\hat{E}(x,t)\dot{x} = \hat{k}(x,t), \tag{5a}$$
$$0 = h(x,t) \tag{5b}$$

with $\hat{E}(x,t) = S_D(x,t)E(x,t)$ and $\hat{k}(x,t) = S_D(x,t)k(x,t)$ to overcome the disadvantage of the overdetermined formulation (4) with respect to numerical integration. The obtained projected-strangeness-free form (5) is equivalent to the original DAE (1) in the sense that both have the same solution set. Furthermore, (5) is suitable for numerical treatment using stiff ODE solvers, in particular, also within the Modelica framework. Unfortunately, for complex dynamical systems the analytical determination of the dynamic selector S_D in advance is very difficult. For more details, we refer to [18, 19].

Example 2.1 The Simple Pendulum: Let us consider the simple pendulum

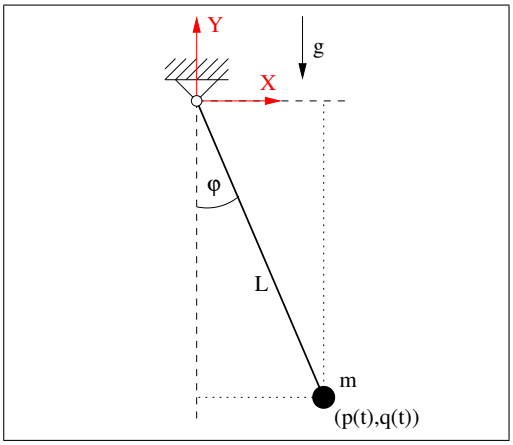

Figure 2: Topology of the mathematical pendulum

of length $L \neq 0$ and mass m under gravity with the gravitational acceleration g, see Figure 2. We choose absolute coordinates p and q denoting the X-Y-position

of the mass m in the two dimensional space $\mathbb{R}^2$. The equations of motion have the form

$$\begin{aligned} \dot{p} &= v, && \text{(6a)} \\ \dot{q} &= w, && \text{(6b)} \\ m\dot{v} &= -2p\lambda, && \text{(6c)} \\ m\dot{w} &= -mg - 2q\lambda, && \text{(6d)} \\ 0 &= p^2 + q^2 - L^2, && \text{(6e)} \end{aligned}$$

where v and w denote the velocities of the mass point in X- and Y-direction while λ corresponds to the Lagrange multiplier. The constraint (6e) has level 0. The hidden constraint of level 1 which is obtained by differentiating the constraint of level 0 (6e) with respect to t and replacing $\dot{x}$ and $\dot{y}$ using (6a),(6b) is given by

$$0 = 2pv + 2qw. \qquad \text{(6f)}$$

Furthermore, differentiating the constraint of level 1 (6f) with respect to t and replacing $\dot{x}$, $\dot{y}$, $\dot{v}$, and $\dot{w}$ using (6a)-(6d) yields the hidden constraint of level 2

$$0 = 2v^2 + 2w^2 + 2p(-2p\lambda)/m + 2q(-mg - 2q\lambda)/m. \qquad \text{(6g)}$$

Subsequent differentiations of the hidden constraint of level 2 (6g) adds no further constraints since $\dot{\lambda}$ cannot be replaced. As a consequence, the maximal constraint level is $\nu_c = 2$ and the regularized DAE via overdetermined formulation is given by equations (6a)-(6g). This regularized DAE consists of seven equations for the five unknowns $x = \begin{bmatrix} p & q & v & w & \lambda \end{bmatrix}^T$ and is therefore not solvable within the Modelica framework.
Towards determination of the projected strangeness-free formulation we have to determine a selector S_D to eliminate redundant equations, i.e., to eliminate redundant rows in the upper part of

$$\left[\frac{E(x,t)}{h_{,x}(x,t)}\right] = \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 2p & 2q & 0 & 0 & 0 \\ 2v & 2w & 2p & 2q & 0 \\ -8p\lambda/m & -2g - 8q\lambda/m & 4v & 4w & -4(p^2+q^2)\lambda)/m \end{array}\right].$$

The dynamic selector S_D can be chosen as

$$S_D(x,t) = \begin{bmatrix} q & -p & 0 & 0 & 0 \\ 0 & 0 & q & -p & 0 \end{bmatrix}$$

such that

$$\left[\frac{S_D(x,t)E(x,t)}{h_{,x}(x,t)}\right] = \left[\begin{array}{ccccc} q & -p & 0 & 0 & 0 \\ 0 & 0 & mq & -mp & 0 \\ \hline 2p & 2q & 0 & 0 & 0 \\ 2v & 2w & 2p & 2q & 0 \\ -8p\lambda/m & -2g - 8q\lambda/m & 4v & 4w & -4(p^2+q^2)\lambda)/m \end{array}\right]$$

is nonsingular for all consistent values $(x,t) \in \mathbb{M} = \{(x,t) \in \mathbb{R}^5 \times \mathbb{I} \; : \text{satisfying (6e)-(6g)}\}$, in particular, for $p^2 + q^2 = L^2 \neq 0$. We obtain the projected strangeness-free formulation

$$\begin{aligned} q\dot{p} - p\dot{q} &= qv - pw, && \text{(7a)} \\ mq\dot{v} - mp\dot{v} &= q(-2p\lambda) - p(-mg - 2q\lambda), && \text{(7b)} \\ 0 &= p^2 + q^2 - L^2, && \text{(7c)} \\ 0 &= 2pv + 2qw, && \text{(7d)} \\ 0 &= 2v^2 + 2w^2 + 2p(-2p\lambda)/m + 2q(-mg - 2q\lambda)/m. && \text{(7e)} \end{aligned}$$

◁

3 Numerical Approach

The projected-strangeness-free formulation (5) can be the starting point for numerical integration with numerical algorithms for stiff ODEs, e.g., implicit Runge-Kutta methods or BDF methods also within the Modelica framework.
Furthermore, we have seen in previous section that the overdetermined DAE (4) has the same solution set as the projected strangeness-free system (5), see also [18, 19]. In particular, the DAE (4) is solvable and does not contain contradictory equations. It is therefore a good idea to use adapted numerical integration schemes which are directly based on the overdetermined DAE system (4) instead of the projected strangeness-free formulation (5).
Let us motivate this by the implicit Euler method. For more details we refer to [19, 17]. Let τ_k denote the step size in the integration step $k=1, ..., N+1$ of the Euler scheme, t_k the discrete time point and x_k the approximation of the solution $x(t_k)$ at the point t_k. Discretization of the overdetermined system (4) leads to the overdetermined nonlinear system

$$0 = \begin{bmatrix} E(x_k, t_k)(x_k - x_{k-1}) - \tau_k k(x_k, t_k) \\ -\tau_k h(x_k, t_k) \end{bmatrix} \tag{8}$$

for the next iterate x_k.
Unfortunately, the nonlinear system (8) is no longer solvable because of discretization and rounding errors during numerical integration. Therefore, it is only possible to find an approximation $\tilde{x}_k$ which minimizes the residual $r \neq 0 \in \mathbb{R}^{n+n_C}$ with

$$r = \begin{bmatrix} r_D \\ r_C \end{bmatrix} = \begin{bmatrix} E(\tilde{x}_k, t_k)(\tilde{x}_k - x_{k-1}) - \tau_k k(\tilde{x}_k, t_k) \\ -\tau_k h(\tilde{x}_k, t_k) \end{bmatrix}$$

in a certain sense. Unfortunately, in general, such an approximation yields to $r_C \neq 0$ which in turn leads to unfulfilled constraints, i.e., $0 \neq h(x_k, t_k)$, not even within machine precision. This would lead to the typical difficulties in the numerical integration of higher index DAEs, i.e., instabilities, convergence problems, inconsistencies, or the solution drifts away from the original set of consistency.
In order to avoid these problems it is necessary to make sure that the constraints are always satisfied during numerical integration. This can be achieved if the

nonlinear system (8) is treated separately such that the next iterate x_k satisfies the lower part, i.e., the constraints, exactly or within a prescribed precision while x_k yields a minimal residual in the upper part, i.e., in the differential part. For more details see [19, 17].
The advantage of the direct discretization of the overdetermined formulation (4) is the fact that it is not necessary to determine the selector S_D analytically in advance and that the number of unknowns in the DAE is not increased. Another advantage with respect to the numerical integration is the possibility to add solution invariants, e.g., mass, impulse or energy conservation. Adding such conservation laws to the constraints $0 = h(x, t)$ often stabilizes numerical integration.
The described numerical approach is implemented in the software package `QUALIDAES` [17] which literally means "`QUA`si-`LI`near `DAE S`olver". This software package is suited for the direct numerical integration of regularized overdetermined model equations and is based on the 3-stage implicit Runge-Kutta method of type RADAU IIa of order 5, see [11, 12], as discretization of the (overdetermined) quasi-linear DAE (2). Although `QUALIDAES` is based on the presented regularization technique, i.e., `QUALIDAES` uses the projected-strangeness-free formulation (5), the user does not have to provide this projected-strangeness-free formulation. It is sufficient if the user provides the hidden constraints (3) together with the original DAE (1), i.e., the overdetermined regularization (4). In particular, it is not necessary to determine the dynamic selector S_D analytically. This is achieved automatically within the separated treatment of (8) by its numerical solution, described above. Moreover, in `QUALIDAES` it is not necessary to provide all hidden constraints. For a successful integration it is often enough to provide only the (hidden) constraints up to level $\nu_c - 1$. For more details on `QUALIDAES` see [17]. Note that `QUALIDAES` is not restricted to the Modelica framework. `QUALIDAES` can be applied to model equations provided in Fortran source code.

4 Numerical Integration using `MO2FOR/QUALIDAES`

In this section we describe the numerical integration of model equations for dynamical systems defined in Modelica using the software package `MO2FOR/QUALIDAES`. The model equations are already given in overdetermined form (4), i.e., in regularized form without further hidden constraints. If it is not possible or too difficult to provide all (hidden) constraints explicitly, then as many constraints as possible should be added.

The integration process consists of two steps. First the translator `MO2FOR` translates model equations provided in Modelica language into Fortran source code using the regularization approach via overdetermined formulations. The second step `QUALIDAES` performs the subsequent numerical integration of the regularized model equations in Fortran source code.

4.1 Translation of Modelica models with `MO2FOR`

`MO2FOR` which literally means "Modelica to Fortran" is a command line tool which accepts model definitions written in the Modelica language and outputs those definitions in Fortran. It is based on the latest Modelica specifications 3.3

and written in C++ without any external libraries.
The tool translates Modelica models in three steps. Each of these steps can be performed separately as well with appropriate command line options.

4.1.1 Parsing

The first step is to parse the model definitions into C++ data structures which can be manipulated in later steps. For each Modelica model an equivalent C++ object is created representing the model along with its variables, parameters, nested classes and model equations. Logical and algebraic expressions are converted into abstract syntax trees which can then easily be manipulated lateron. During parsing syntax and identifiers are checked for compliance with the Modelica specifications. The parsed model is an abstract representation of the original definition, i.e., references to other models, equations and expressions are not checked for semantic validity.

4.1.2 Instantiate

After a model has been parsed it has to be instantiated. This means:

1. The abstract model definition is replaced by a model tree linking all referenced models together. For instance, if the model definition of a pendulum is parsed with local variables of type "Rod" and "Mass" we have to look for model definitions of "Rod" and "Mass", instantiating their local variables etc. until we end up with basic types (e.g., Real, String, Boolean) which do not depend on other model definitions anymore. In case the model definition can not be found within the same Modelica file we check other Modelica files on the search path. The same procedure applies to extended and imported models.

2. The model tree is flattened into one single model, i.e., all properties of the referenced models are transferred to the instantiated model. In particular, all variables and equations are added. After flattening all variables and parameters have basic types.

Moreover, modifiers are applied, e.g., initial values are set, and other preparations for simulation are made. For instance, connect equations are replaced by regular equations analogous to Kirchhoff's current law [4].

4.1.3 Export

The instantiated model is modified and prepared for output. This means, in particular,

1. reordering of equations such that differential equations appear first,

2. rearranging terms within equations to comply with the form of a quasi-linear DAE, i.e., differential expressions on the left hand side and algebraic ones on the right hand side,

3. algebraic simplifications in all expressions to save processing time.

The modified model is exported by creating a Fortran file that contains all necessary functions, variable definitions and equations to simulate the model with QUALIDAES. We use fixed format Fortran source files. Therefore we have to add a string in front of each source code line depending on its type (comment, label, etc.). We do not care too much about the length of each line but try to wrap around lines after 72 characters in one line.

4.2 Integration using QUALIDAES

The previous translation of the Modelica model into a Fortran model with MO2FOR enables us to integrate the model equations using QUALIDAES. The translator MO2FOR creates, in addition to the Fortran model, an Makefile. Executing this Makefile the Fortran model equations will be compiled and linked to the solver QUALIDAES. Furthermore, the obtained executable file will be executed and the numerical results will be written into a *_res.m file which can be used in MATLAB for further postprocessing. The numerical integration will be performed with the default settings of the solver QUALIDAES, see [17]. The time domain $\mathbb{I}$ can be specified using the options of the translator MO2FOR during the translation. If no time domain is specified, the integration will be performed within the (default) time domain $\mathbb{I} = [0, 1]$,

5 Numerical Results

In Figure 1 the approach for the numerical treatment of models defined in Modelica using MO2FOR and QUALIDAES is illustrated. Let us discuss the procedure by the following two examples, the simple pendulum and the car axis.

Example 5.1 Consider the mathematical pendulum (see Example 2.1) of length $L = 1$ and with mass $m = 1$ under the influence of the gravitational acceleration $g = 13.7503716373295$. We start with the model equations (6) of the simple pendulum modeled with Modelica in the overdetermined formulation. The model equations in Modelica source code are given as follows.

```
class SimpPend "Simple planar pendulum"
  parameter Real m = 1;
  parameter Real g = 13.7503716373295;
  parameter Real L = 1;
  Real x(start = L);
  Real y(start = 0);
  Real vx(start = 0);
  Real vy(start = 0);
  Real lambda;

equation
  der(x) = vx;
  der(y) = vy;
  m * der(vx) = -2 * x * lambda;
  m * der(vy) = -m * g - 2 * y * lambda;
  0 = x ^ 2 + y ^ 2 - L ^ 2;
  0 = 2 * x * vx + 2 * y * vy;
  0 = 2 * vx * vx + 2 * vy * vy
       + 2 * x * (-2 * x * lambda) / m
       + 2 * y * (-m * g - 2 * y * lambda)/m;
end SimpPend;
```

With MO2FOR we produce Fortran source code as shown in the Appendix A. This Fortran source code is suitable for the numerical integration using QUALIDAES.

Compiling this Fortran source code, linking to `QUALIDAES` and executing the obtained executable file we obtain the numerical results shown in Figure 3.

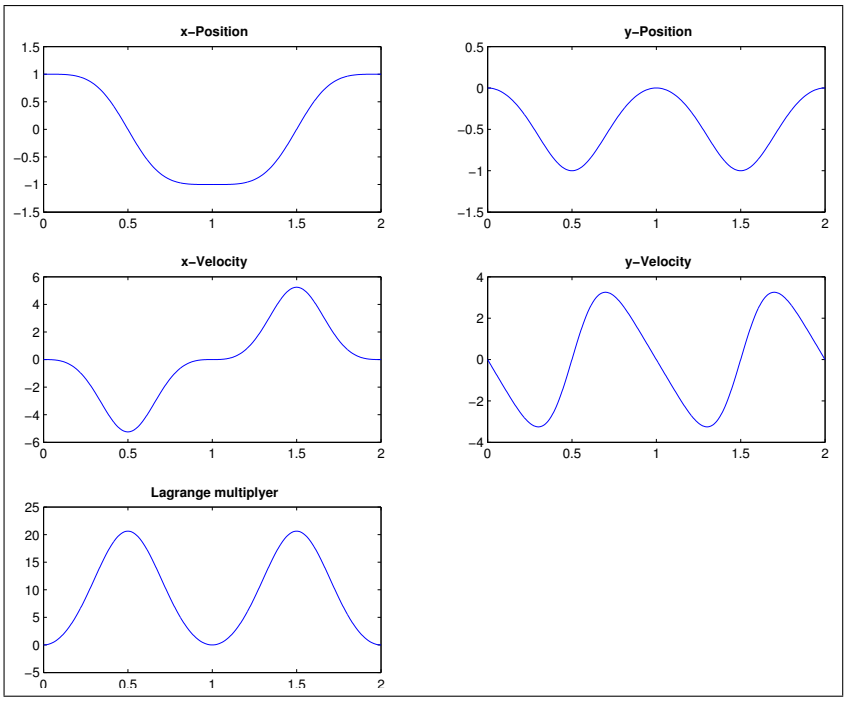

Figure 3: Numerical Results for the Simple Pendulum

Note that the gravitational acceleration chosen as $g = 13.7503716373295$ yields a periodic solution with period $T = 2$. Starting with initial values

$$x(0) = \left[\begin{array}{ccccc} p(0) & q(0) & v(0) & w(0) & \lambda(0) \end{array} \right]^T = \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \end{array} \right]^T$$

the exact solution after `TEND`$= 2s$ is known as

$$x(2) = \left[\begin{array}{ccccc} p(2) & q(2) & v(2) & w(2) & \lambda(2) \end{array} \right]^T = \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \end{array} \right]^T.$$

From the numerical integration we get at the end of the integration interval

$$\bar{x}(2) = \left[\begin{array}{c} \bar{p}(2) \\ \bar{q}(2) \\ \bar{v}(2) \\ \bar{w}(2) \\ \bar{\lambda}(2) \end{array} \right] = \left[\begin{array}{r} 0.999999999997e-00 \\ 2.506954991624e-06 \\ -1.964524883411e-11 \\ 7.831924620040e-06 \\ -1.723575179268e-05 \end{array} \right]$$

which yields the absolute error

$$err(2) = ||\bar{x}(2) - x(2)||_2 = 1.9097e-05$$

by a prescribed tolerance of `ATOL`=`RTOL`$= 1.0e-6$.
A comparison with other numerical integrators using suitable formulations of the model equations on the time domain $\mathbb{I} = [0\text{s}, 1000\text{s}]$ is shown in Figure 4.

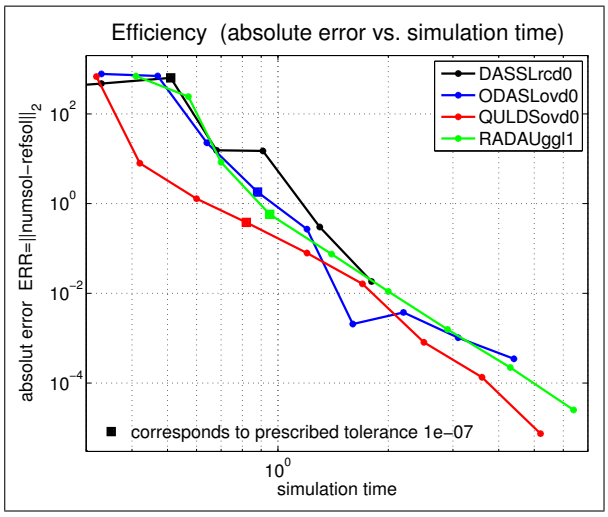

Figure 4: Efficiency for the Simple Pendulum - Obtained accuracy vs. Simulation time

QULDSovd0 :	`MO2FOR/QUALIDAES` with the overdetermined regularization (6a)-(6g)
DASSLrcd0 :	`DASSL` [2, 14] with the d-index 1 formulation (6a)-(6d),(6g)
ODASLovd0 :	`ODASSL` [3, 5] with the overdetermined regularization (6a)-(6g)
RADAUggl1 :	`RADAU5` [11, 12] with the Gear-Gupta-Leimkuhler (GGL) formulation [7]

Table 1: Efficiency: Used solvers and formulations

Here, the efficiency corresponds to the relation of obtained accuracy and the spent computation time.
We compare the efficiency of `MO2FOR/QUALIDAES` with the efficiency of different other integration methods as listed in Table 1.
Note that the model equations of the simple pendulum (6a)-(6e) have the structure of equations of motion for multi-body systems. The Gear-Gupta-Leimkuhler (GGL) formulation [7] is a specific regularization technique for such equations of motion and corresponds to the model equations including the hidden constraints of level 1 and the insertion of an extra Lagrange multiplier μ in the equations (6a),(6b). Therefore, the GGL formulation consists of the equations

$$\begin{aligned} p &= v - 2p\mu \\ q &= w - 2q\mu \end{aligned}$$

and (6c)-(6f). Furthermore, the integration code `ODASSL` based on the backward differentiation formula (BDF) of variable order up to 5 is only suited for this class of equations of motion exploiting its characteristic structure. The integration code `DASSL` is also based on the backward differentiation formula (BDF) of

variable order up to 5. It can be applied to nonlinear DAEs of differentiation index (d-index) at most 1 only. The integration code `RADAU5`, on the other hand, is based on the Runge-Kutta formula of type Radau IIa of fixed order 5 and can be applied to quasi-linear DAEs with constant leading matrix of d-index up to 2 and, if the DAE offers Hessenberg structure, up to d-index 3.
For the evaluation of the precision of the obtained numerical results, we use as reference solution the numerical solution obtained with `RADAU5` with the GGL formulation and a prescribed tolerance of 10^{-14}.
In Figure 4 the efficiency of the used solvers is illustrated. Here one can observe, that `ODASSL` and `RADAU5` offer a good performance while `DASSL` is only able to compute a numerical solution for prescribed tolerances up to 10^{-11}. Furthermore, one observe that the numerical integration with `QUALIDAES` with the overdetermined regularization (6a)-(6g) for the simple pendulum is very efficient, i.e., in order to obtain the same accuracy in the numerical solution the computational effort is less then using other integration methods. Note that here only the integration time within the numerical integrators is taken into account and not the preprocessing with `MO2FOR`. ◁

Example 5.2 Another example of interest is the car axis which is listed as benchmark example in the *Test Set for IVP Solvers*[5].

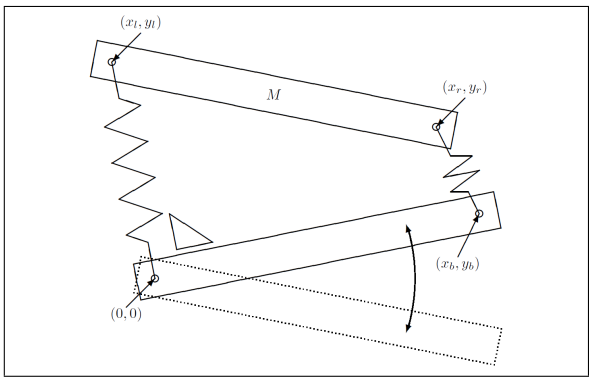

Figure 5: Topology of the Car Axis

The car axis problem is a rather simple multibody system in which the behaviour of a car axis on a bumpy road is modeled. The simplified problem is illustrated in Figure 5. The situation is modeled such that the left wheel at the origin $(0, 0)$ rolls on a flat surface and the right wheel at coordinates (x_b, y_b) rolls over a hill of height h every τ seconds. This means that the motion of y_b is known. The length of the axis denoted by l with $l^2 = x_b^2(t) + y_b^2(t)$ remains constant in time. Two springs carry the movement of the axis between the wheels over to the chassis of the car which is represented by the bar $(x_l, y_l) - (x_r, y_r)$ of mass m. The two springs are assumed to be massless and have Hooke constant $1/\varepsilon^2$ and length l_0 at rest. Therefore, there are two position constraints. First, the distance between (x_l, y_l) and (x_r, y_r) must remain constant l. Second, for simplicity of the model we assume that the left spring remains orthogonal to the axis.

[5] `http://www.dm.uniba.it/~testset`

Then the model equations are like equations of motion for multibody systems

$$\begin{aligned} \dot{p} &= v, \\ M\dot{v} &= f(p,v,t) - G^T(p,t)\lambda, \\ 0 &= g(p,t). \end{aligned}$$

The state $x = \begin{bmatrix} p^T & v^T & \lambda^T \end{bmatrix}^T$ with position $p^T = \begin{bmatrix} x_l & y_l & x_r & y_r \end{bmatrix}$, velocity $v = \dot{p}$ and Lagrange multipliers $\lambda^T = \begin{bmatrix} \lambda_1 & \lambda_2 \end{bmatrix}$ is restricted by the holonomic constraints

$$g(p,t) = \begin{bmatrix} x_l x_b + y_l y_b \\ (x_l - x_r)^2 + (y_l - y_r)^2 - l^2 \end{bmatrix}.$$

The mass matrix is given as $M = \varepsilon^2 \frac{m}{2} I_4$, where I_4 is the identity matrix of size 4×4. The function f of the applied forces and the constraint matrix G are given by

$$f(p,v,t) = \begin{bmatrix} (l_0 - l_l)\frac{x_l}{l_l} \\ (l_0 - l_l)\frac{y_l}{l_l} - \varepsilon^2 \frac{m}{2} \\ (l_0 - l_r)\frac{x_r - x_b}{l_r} \\ (l_0 - l_r)\frac{y_r - y_b}{l_r} - \varepsilon^2 \frac{m}{2} \end{bmatrix}, \; G^T(p,t) = \frac{\mathrm{d}}{\mathrm{d}t} g(p(t),t) = \begin{bmatrix} x_b & 2(x_l - x_r) \\ y_b & 2(y_l - y_r) \\ 0 & -2(x_l - x_r) \\ 0 & -2(y_l - y_r) \end{bmatrix}$$

where $l_l = \sqrt{x_l^2 + y_l^2}$ and $l_r = \sqrt{(x_r - x_b)^2 + (y_r - y_b)^2}$.
The excitation follows from the known functions

$$\begin{aligned} x_b(t) &= \sqrt{l^2 - y_b^2(t)} \\ y_b(t) &= r \sin(\omega t) \end{aligned}$$

with the following constants

$$\begin{array}{cccccccccccc} l & = & 1 & & \varepsilon & = & 10^{-2} & & h & = & 1/5 & \omega & = & 10 \\ l_0 & = & 1/2 & & m & = & 10 & & \tau & = & \pi/5 & r & = & 0.1 \end{array}$$

Consistent initial values are given by

$$p(0) = \begin{bmatrix} 0 \\ 1/2 \\ 1 \\ 1/2 \end{bmatrix}, \; v(0) = \begin{bmatrix} -1/2 \\ 0 \\ -1/2 \\ 0 \end{bmatrix}, \; \lambda_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

For more details see the description in the *Test Set for IVP Solvers*[6].
The model equations in Modelica source code are given as follows.

```
class CarAxis
  parameter Real eps=0.01,M=10,L=1,L0=0.5,r=0.1,w=10;
  Real xl(start=0),yl(start=0.5),
       xr(start=1),yr(start=0.5),
       V1(start=-0.5),V2(start=0),V3(start=-0.5),V4(start=0),
       lam1(start=0),lam2(start=0),
       V1P(start=0),V2P(start=0),V3P(start=0),V4P(start=0),
       yb(start=0),ybP(start=1),ybPP(start=0),
       xb(start=1),xbP(start=0),xbPP(start=-1),
       Ll(start=0.5),Lr(start=0.5);
  equation
    //abreviations//
      yb  = r    *sin(w*time);
```

[6] http://www.dm.uniba.it/~testset

```
    ybP = r*w  *cos(w*time);
    ybPP=-r*w*w*sin(w*time);
    xb  = (L*L-yb*yb)^0.5;
    xbP =-yb*ybP/(L*L-yb*yb)^0.5;
    xbPP=-yb*yb*ybP*ybP/(L*L-yb*yb)^1.5
         -(ybP*ybP+yb*ybPP)/(L*L-yb*yb)^0.5;
    Ll = (xl^2+yl^2)^0.5;
    Lr = ((xr-xb)^2+(yr-yb)^2)^0.5;
    V1P =(L0-Ll)*xl/Ll     +lam1*xb+2*lam2*(xl-xr);
    V2P =(L0-Ll)*yl/Ll     +lam1*yb+2*lam2*(yl-yr)-M*eps*eps/2;
    V3P =(L0-Lr)*(xr-xb)/Lr        -2*lam2*(xl-xr);
    V4P =(L0-Lr)*(yr-yb)/Lr        -2*lam2*(yl-yr)-M*eps*eps/2;
  //kinematic equations of motion//
    der(xl) = V1;
    der(yl) = V2;
    der(xr) = V3;
    der(yr) = V4;
  //dynamic equations of motion//
    eps*eps*M/2*der(V1) =V1P;
    eps*eps*M/2*der(V2) =V2P;
    eps*eps*M/2*der(V3) =V3P;
    eps*eps*M/2*der(V4) =V4P;
 //constraints on position level//
    0 = xb*xl+yb*yl;
    0 = (xl-xr)^2+(yl-yr)^2-L*L;
 //constraints on velocity level//
    0 = xbP*xl+xb*V1+ybP*yl+yb*V2;
    0 = 2*(xl-xr)*(V1-V3)+2*(yl-yr)*(V2-V4);
 //constraints on acceleration level//
    0 = xbPP*xl+2*xbP*V1+xb*V1P/(eps*eps*M/2)
      +ybPP*yl+2*ybP*V2+yb*V2P/(eps*eps*M/2);
    0 = 2*(V1-V3)*(V1-V3)
      +2*(xl-xr)*(V1P-V3P)/(eps*eps*M/2)
      +2*(V2-V4)*(V2-V4)
      +2*(yl-yr)*(V2P-V4P)/(eps*eps*M/2);
end CarAxis;
```

The obtained Fortran source code which is ready for simulation with `QUALIDAES` is shown in the Appendix B.
Compiling this Fortran source code, linking to `QUALIDAES` and executing the obtained executable file we obtain the numerical results shown in Figure 6.
From the *Test Set for IVP Solvers*[7] a reference solution at $t = 3s$ is known as

$$p(3) = \begin{bmatrix} 0.4934557842755629e-1 \\ 0.4969894602303324e+0 \\ 0.1041742524885400e+1 \\ 0.3739110272652214e+0 \end{bmatrix},\ v(3) = \begin{bmatrix} -0.7705836840321485e-1 \\ 0.7446866596327776e-2 \\ 0.1755681574942899e-1 \\ 0.7703410437794031e+0 \end{bmatrix},$$

$$\lambda(3) = \begin{bmatrix} -0.4736886750784630e-2 \\ -0.1104680411345730e-2 \end{bmatrix}$$

From the numerical integration we get at the end of the integration interval

$$\bar{p}(3) = \begin{bmatrix} 0.4934557505765604e-1 \\ 0.4969894262895335e+0 \\ 0.1041742738582200e+1 \\ 0.3739127436599655e+0 \end{bmatrix},\ \bar{v}(3) = \begin{bmatrix} -0.7705859110049050e-1 \\ 0.7444570127077410e-2 \\ 0.1754833790605591e-1 \\ 0.7702832036542340e+0 \end{bmatrix},$$

$$\bar{\lambda}(3) = \begin{bmatrix} -0.4736932227685571e-2 \\ -0.1104703372042073e-2 \end{bmatrix}$$

which yields an absolute error

$$err(3) = ||\bar{x}(3) - x(3)||_2 = 5.8529e-05$$

[7] `http://www.dm.uniba.it/~testset`

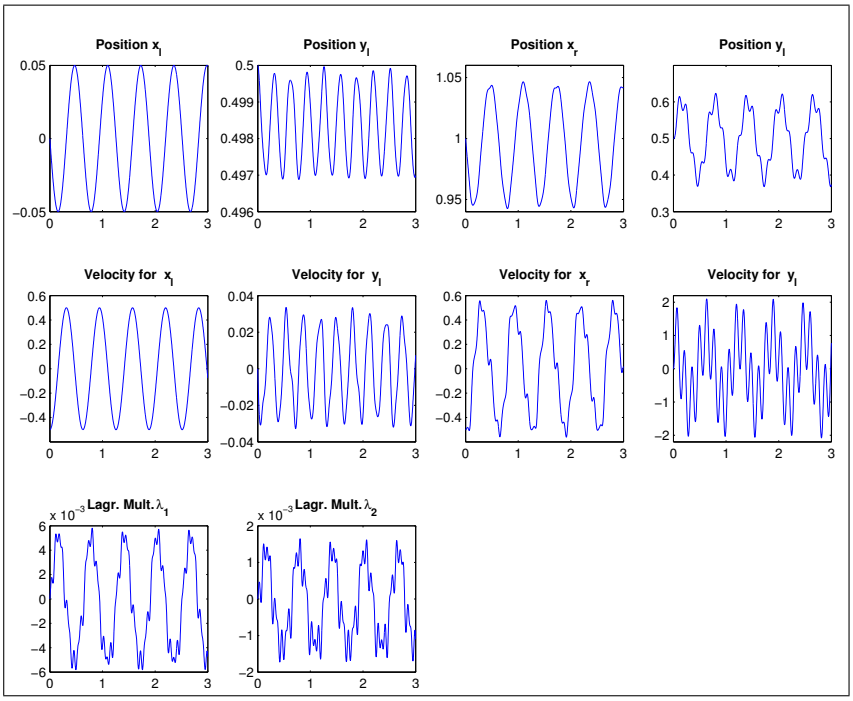

Figure 6: Numerical Results for the Car Axis

by a prescribed tolerance of ATOL=RTOL= $1.0e-6$.
A comparison with other numerical integrators using suitable formulations of the model equations on the time domain $\mathbb{I} = [0\text{s}, 500\text{s}]$ is shown in Figure 7.

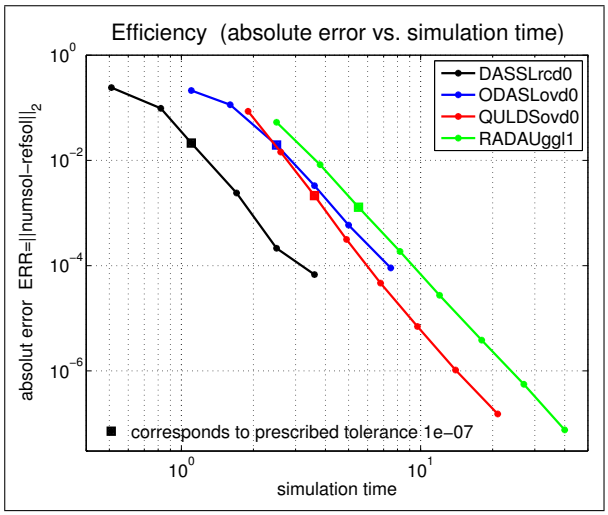

Figure 7: Efficiency for the Car Axis - Obtained accuracy vs. Simulation time

We compared the same integrators as in the previous example, see Table 1.
In Figure 7 we observe that the numerical integration with DASSL offers an

excellent efficiency up to a prescribed tolerance of 10^{-10}. A further improvement is not possible. The numerical integration with ODASSL behaves similar with a little more need of computational time. Higher accuracy is possible with use of RADAU5 and QUALIDAES where QUALIDAES is a little more efficient. ◁

6 Summary

In this article we discussed the efficient and robust numerical simulation of dynamical systems modeled with Modelica. We presented a regularization method for quasi-linear DAEs which yields an overdetermined DAE without hidden constraints. This DAE can usually not be solved numerically with existing codes. Therefore, with MO2FOR we translate the overdetermined model equation provided in Modelica into Fortran source code which can be integrated using the software package QUALIDAES. The applicability of QUALIDAES in combination with MO2FOR for the numerical treatment of model equations provided in Modelica language is demonstrated at the example of a simple pendulum and of a car axis.

A Fortran Source Code for the Model of the Simple Pendulum

```
C     SimpPend
C     Solver: Qualidaes
      IMPLICIT NONE
      INTEGER      qI,qIT,qJ,qK,qLIWORK,qLRWORK,qMD,qMC,qN,
     #             qNIVCOND,qSTARTN,qPRED,qNWTMAT,qNWTUPD,
     #             qDECOMPC,qDECOMPD,qSELCOMP
      PARAMETER    (qMD=4,qMC=3,qN=5,qNIVCOND=0)
      PARAMETER    (qLIWORK=1000,qLRWORK=10000)
      INTEGER      qIWORK(qLIWORK),qITOL,qIOPT(40),qIPAR(1),qIDID,qIJAC,
     #qIOUT,qIERR,qTOLMIN,qTOLMAX
      DOUBLE PRECISION qT,qTSTART,qTEND,qH,qH0,qCPT0,qCPT1,qPERT
      DOUBLE PRECISION qRWORK(qLRWORK),qX(qN),qROPT(40),qRPAR(5),qRTOL,q
     #ATOL,qNAN
      DOUBLE PRECISION  m,g,L
      EXTERNAL     QUALIDAES,IVCOND,RHS,MATRIX,SOLOUT,JAC
      INTRINSIC    DSIN,DCOS,DEXP,DABS,DTAN
      PARAMETER  (m=1.0D0,g=13.7503716373295D0,L=1.0D0)
C
C     .. SETTINGS ..
C     wrt the Problem
      qTSTART=0.0D0
      qTEND=2.0D0
      qH0=0.01D0
C
C     wrt Integration
      qIOPT( 1)=0
      qIOPT( 2)=0       ! LUN
      qIOPT( 3)=0       ! NIT
      qIOPT( 4)=0       ! STARTN
      qIOPT( 5)=0       ! SINDN
      qIOPT( 6)=1000000 ! NMAX
      qIOPT( 8)=0       ! PRED
      qIOPT( 9)=0       ! NWTMAT
      qIOPT(10)=0       ! NWTUPD
      qIOPT(11)=1       ! DECOMPC
      qIOPT(12)=0       ! DECOMPD
      qIOPT(13)=0       ! SELCOMP
      qIOPT(14)=0       ! AUTONOM
      qIOPT(15)=1       ! MASSTRKT
      qIOPT(17)=1       ! IVCNSST
      qROPT( 1)=0.0D0   ! UROUND
      qROPT( 2)=0.0D0   ! SAFE
      qROPT( 3)=0.0D0   ! THET
      qROPT( 4)=0.0D0   ! FNEWT
      qROPT( 5)=0.0D0   ! QUOT1
      qROPT( 6)=0.0D0   ! QUOT2
      qROPT( 7)=0.0D0   ! HMAX
      qROPT( 8)=0.0D0   ! FACL
      qROPT( 9)=0.0D0   ! FACR
C
C     .. output ..
      OPEN(UNIT=12,FILE='SimpPend_res.m')
C
C     .. rpar/lpar ..
      qIPAR(1)=1
      qRPAR(1)=qTSTART
      qRPAR(2)=(qTEND-qTSTART)/1000
C
C     .. Initialization ..
      qT=qTSTART
      qH=qH0
      qRTOL=1.0D-6
      qATOL=qRTOL
      qITOL=0
      qIJAC=0
      qIOUT=1
      qIDID=0
      qIERR=0
C
C     .. Initial values ..
      qX(1) = L! x
      qX(2) = 0.0D0! y
      qX(3) = 0.0D0! vx
      qX(4) = 0.0D0! vy
      qX(5) = 0.0D0! lambda
C
C     .. Solver Call ..
      qIDID=0
      CALL CPU_TIME(qCPT0)
      CALL QUALIDAES(
     #   qMD,qMC,qN,qNIVCOND,
     #   qX,qT,qTEND,qH,qRTOL,qATOL,qITOL,qIOPT,qROPT,
     #   IVCOND,RHS,MATRIX,JAC,qIJAC,SOLOUT,
     #   qIOUT,qLIWORK,qIWORK,qLRWORK,qRWORK,
     #   qRPAR,qIPAR,qIERR,qIDID)
C     .. OUTPUT STATISTICS ..
      CALL CPU_TIME(qCPT1)
      qCPT1=qCPT1-qCPT0
      WRITE(12,80021)qIDID,qT,
     #   qIWORK(1),qIWORK(2),qIWORK(6),
     #   qIWORK(10),qIWORK(4),qIWORK(7),
     #   qIWORK(11),qIWORK(3),qIWORK(8),
     #   qCPT1     ,          qIWORK(5)
80021 FORMAT('IDID=',I5,';',/,
     #   'T=',F9.4,';',/,
     #   'NACCPT =',I8 ,';',/,
     #   'NRHS   =',I8 ,';',/,
     #   'NPDEC  =',I8 ,';',/,
     #   'NERJCT =',I8 ,';',/,
     #   'NJAC   =',I8 ,';',/,
     #   'NEDEC  =',I8 ,';',/,
     #   'NCRJCT =',I8 ,';',/,
     #   'NMAT   =',I8 ,';',/,
     #   'NBSUB  =',I8 ,';',/,
     #   'CPUTIME=',F8.3,';',/,
     #   'NSEL   =',I8 ,';' ,/,'')
```

```
      CLOSE(12)
      END
C     --- --- --- --- --- --- ---
C         END OF MAIN
C     --- --- --- --- --- --- ---
C
      SUBROUTINE IVCOND(N,T,X,NCOND,COND,IPAR,RPAR,IERR)
      IMPLICIT NONE
      INTEGER          N,NCOND,IPAR(*),IERR
      DOUBLE PRECISION T,X(N),COND(NCOND),RPAR(*)
C     .. Initial Conditions for P ..
C      COND(1)=
C     .. Initial Conditions for V ..
C      COND(2)=
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF IVCOND
C     --- --- --- --- --- --- ---
C
      SUBROUTINE RHS(qMD,qMC,qN,qT,qX,qF,qG,qIOPT,qROPT,qIPAR,qRPAR,qIDI
     #D)
C
      IMPLICIT NONE
      INTEGER qMD,qMC,qN,qIOPT(*),qIPAR(*),qIDID
      DOUBLE PRECISION qT,qX(qN),qF(qMD),qG(qMC),qROPT(*),qRPAR(*)
      DOUBLE PRECISION  m,g,L
      DOUBLE PRECISION  x,y,vx,vy,lambda
      PARAMETER  (m=1.0D0,g=13.7503716373295D0,L=1.0D0)
C
      x = qX(1)!
      y = qX(2)!
      vx = qX(3)!
      vy = qX(4)!
      lambda = qX(5)!
C
      qF(1) = vx
      qF(2) = vy
      qF(3) = -2.0D0*x*lambda
      qF(4) = -m*g-2.0D0*y*lambda
      qG(1) = x**2.0D0+y**2.0D0-L**2.0D0
      qG(2) = 2.0D0*x*vx+2.0D0*y*vy
      qG(3) = 2.0D0*vx*vx+2.0D0*vy*vy+2.0D0*x*(-2.0D0*x*lambda)/(m)+2.0D
     #0*y*(-m*g-2.0D0*y*lambda)/(m)
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF RHS
C     --- --- --- --- --- --- ---
C
      SUBROUTINE MATRIX(qMD,qN,qT,qX,qMA,qIOPT,qROPT,qIPAR,qRPAR,qIDID)
C
      IMPLICIT NONE
      INTEGER          qMD,qN,qI,qJ,qIOPT(*),qIPAR(*),qIDID
      DOUBLE PRECISION qT,qX(qN),qMA(qMD,qN),qROPT(*),qRPAR(*)
      DOUBLE PRECISION  m,g,L
      DOUBLE PRECISION  x,y,vx,vy,lambda
      PARAMETER  (m=1.0D0,g=13.7503716373295D0,L=1.0D0)
C
      x = qX(1)!
      y = qX(2)!
      vx = qX(3)!
      vy = qX(4)!
      lambda = qX(5)!
C
      DO qI=1,qMD
        DO qJ=1,qN
          qMA(qI,qJ)=0.0D0
        END DO
      END DO
C
      qMA(1,1) = 1.0D0
      qMA(2,2) = 1.0D0
      qMA(3,3) = m
      qMA(4,4) = m
C
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF MAT
C     --- --- --- --- --- --- ---
C
      SUBROUTINE JAC()
      END
C     --- --- --- --- --- ---
C         END OF JAC
C     --- --- --- --- --- ---
C
      SUBROUTINE SOLOUT(NACCPT,TOLD,T,X,N,NN2,NN3,NN4,CONTX,H,C1M1,C2M1,
     #
     #                  IOPT,ROPT,IPAR,RPAR,IERR)
      IMPLICIT NONE
C     .. Arguments ..
      INTEGER          NACCPT,N,NN2,NN3,NN4,IOPT(*),IPAR(*),IERR,I
      DOUBLE PRECISION TOLD,T,H,X(N),CONTX(NN4),C1M1,C2M1,
     #                 ROPT(*),RPAR(*)
C     .. Locals ..
      DOUBLE PRECISION TOUT,XOUT(N)
      INTRINSIC        DSIN,DCOS,DSQRT
      EXTERNAL         QLDENSOUTV
100   CONTINUE
      TOUT=RPAR(1)
      IF(TOUT.LE.T)THEN
        IF(IPAR(1).EQ.1) WRITE(12,80003)
80003   FORMAT('simvar={''T'';',
     #       '''x'';',
     #       '''y'';',
     #       '''vx'';',
     #       '''vy'';',
     #       '''lambda'';'
     #'};')
C       .. Numerical solution ..
        CALL QLDENSOUTV(TOUT,XOUT,N,NN2,NN3,NN4,T,H,CONTX,C1M1,C2M1)
C       .. Output ..
        WRITE (12,80004)
     #    IPAR(1),TOUT,
     #    (XOUT(I),I=1,N)
80004   FORMAT('simres(',I6,',:)=[',6(D23.16,','),'];')
        RPAR(1)=IPAR(1)*RPAR(2)
        IPAR(1)=IPAR(1)+1
        GOTO 100
      END IF
      END
C     --- --- --- --- --- ---
C     END OF SOLOUT
C     --- --- --- --- --- ---
```

B Fortran Source Code for the Model of the Car Axis

```
C     CarAxis
C     Solver: Qualidaes
      IMPLICIT NONE
      INTEGER      qI,qIT,qJ,qK,qLIWORK,qLRWORK,qMD,qMC,qN,
     #             qNIVCOND,qSTARTN,qPRED,qNWTMAT,qNWTUPD,
     #             qDECOMPC,qDECOMPD,qSELCOMP
      PARAMETER    (qMD=8,qMC=18,qN=22,qNIVCOND=0)
      PARAMETER    (qLIWORK=1000,qLRWORK=10000)
      INTEGER      qIWORK(qLIWORK),qITOL,qIOPT(40),qIPAR(1),qIDID,qIJAC,
     #qIOUT,qIERR,qTOLMIN,qTOLMAX
      DOUBLE PRECISION qT,qTSTART,qTEND,qH,qH0,qCPT0,qCPT1,qPERT
      DOUBLE PRECISION qRWORK(qLRWORK),qX(qN),qROPT(40),qRPAR(8),qRTOL,q
     #ATOL,qNAN
      DOUBLE PRECISION  eps,M,L,L0,r,w
      EXTERNAL     QUALIDAES,IVCOND,RHS,MATRIX,SOLOUT,JAC
      INTRINSIC    DSIN,DCOS,DEXP,DABS,DTAN
      PARAMETER  (eps=0.01D0,M=10.0D0,L=1.0D0,L0=0.5D0,r=0.1D0,w=10.0D0)
C
C     .. SETTINGS ..
C     wrt the Problem
      qTSTART=0.0D0
      qTEND=3.0D0
      qH0=0.01D0
C
      qIOPT( 2)=0       ! LUN
      qIOPT( 3)=0       ! NIT
      qIOPT( 4)=0       ! STARTN
      qIOPT( 5)=0       ! SINDN
      qIOPT( 6)=1000000 ! NMAX
      qIOPT( 8)=0       ! PRED
      qIOPT( 9)=0       ! NWTMAT
      qIOPT(10)=0       ! NWTUPD
      qIOPT(11)=1       ! DECOMPC
      qIOPT(12)=0       ! DECOMPD
      qIOPT(13)=0       ! SELCOMP
      qIOPT(14)=0       ! AUTONOM
      qIOPT(15)=1       ! MASSTRKT
      qIOPT(17)=1       ! IVCNSST
      qROPT( 1)=0.0D0   ! UROUND
      qROPT( 2)=0.0D0   ! SAFE
      qROPT( 3)=0.0D0   ! THET
      qROPT( 4)=0.0D0   ! FNEWT
      qROPT( 5)=0.0D0   ! QUOT1
      qROPT( 6)=0.0D0   ! QUOT2
      qROPT( 7)=0.0D0   ! HMAX
      qROPT( 8)=0.0D0   ! FACL
      qROPT( 9)=0.0D0   ! FACR
C
```

```
C     .. output ..
      OPEN(UNIT=12,FILE='CarAxis_res.m')
C
C     .. rpar/lpar ..
      qIPAR(1)=1
      qRPAR(1)=qTSTART
      qRPAR(2)=(qTEND-qTSTART)/1000
C
C     .. Initialization ..
      qT=qTSTART
      qH=qH0
      qRTOL=1.0D-6
      qATOL=qRTOL
      qITOL=0
      qIJAC=0
      qIOUT=1
      qIDID=0
      qIERR=0
C
C     .. Initial values ..
      qX(1) = 0.0D0! xl
      qX(2) = 0.5D0! yl
      qX(3) = 1.0D0! xr
      qX(4) = 0.5D0! yr
      qX(5) = -0.5D0! V1
      qX(6) = 0.0D0! V2
      qX(7) = -0.5D0! V3
      qX(8) = 0.0D0! V4
      qX(9) = 0.0D0! lam1
      qX(10) = 0.0D0! lam2
      qX(11) = 0.0D0! V1P
      qX(12) = 0.0D0! V2P
      qX(13) = 0.0D0! V3P
      qX(14) = 0.0D0! V4P
      qX(15) = 0.0D0! yb
      qX(16) = 1.0D0! ybP
      qX(17) = 0.0D0! ybPP
      qX(18) = 1.0D0! xb
      qX(19) = 0.0D0! xbP
      qX(20) = -1.0D0! xbPP
      qX(21) = 0.5D0! Ll
      qX(22) = 0.5D0! Lr
C
C     .. Solver Call ..
      qIDID=0
      CALL CPU_TIME(qCPT0)
      CALL QUALIDAES(
     #   qMD,qMC,qN,qNIVCOND,
     #   qX,qT,qTEND,qH,qRTOL,qATOL,qITOL,qIOPT,qROPT,
     #   IVCOND,RHS,MATRIX,JAC,qIJAC,SOLOUT,
     #   qIOUT,qLIWORK,qIWORK,qLRWORK,qRWORK,
     #   qRPAR,qIPAR,qIERR,qIDID)
C     .. OUTPUT STATISTICS ..
      CALL CPU_TIME(qCPT1)
      qCPT1=qCPT1-qCPT0
      WRITE(12,80021)qIDID,qT,
     #   qIWORK(1),qIWORK(2),qIWORK(6),
     #   qIWORK(10),qIWORK(4),qIWORK(7),
     #   qIWORK(11),qIWORK(3),qIWORK(8),
     #   qCPT1     ,          qIWORK(5)
80021 FORMAT('IDID=',I5,';',/,
     #   'T=',F9.4,';',/,
     #   'NACCPT =',I8 ,';',/,
     #   'NRHS   =',I8 ,';',/,
     #   'NPDEC  =',I8 ,';',/,
     #   'NERJCT =',I8 ,';',/,
     #   'NJAC   =',I8 ,';',/,
     #   'NEDEC  =',I8 ,';',/,
     #   'NCRJCT =',I8 ,';',/,
     #   'NMAT   =',I8 ,';',/,
     #   'NBSUB  =',I8 ,';',/,
     #   'CPUTIME=',F8.3,';',/,
     #   'NSEL   =',I8 ,';' ,/,'')
      CLOSE(12)
      END
C     --- --- --- --- --- --- ---
C         END OF MAIN
C     --- --- --- --- --- --- ---
C
      SUBROUTINE IVCOND(N,T,X,NCOND,COND,IPAR,RPAR,IERR)
      IMPLICIT NONE
      INTEGER          N,NCOND,IPAR(*),IERR
      DOUBLE PRECISION T,X(N),COND(NCOND),RPAR(*)
C     .. Initial Conditions for P ..
C      COND(1)=
C     .. Initial Conditions for V ..
C      COND(2)=
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF IVCOND
C     --- --- --- --- --- --- ---
C
      SUBROUTINE RHS(qMD,qMC,qN,qT,qX,qF,qG,qIOPT,qROPT,qIPAR,qRPAR,qIDI
     #D)
C
      IMPLICIT NONE
      INTEGER qMD,qMC,qN,qIOPT(*),qIPAR(*),qIDID
      DOUBLE PRECISION qT,qX(qN),qF(qMD),qG(qMC),qROPT(*),qRPAR(*)
      DOUBLE PRECISION  eps,M,L,L0,r,w
      DOUBLE PRECISION  xl,yl,xr,yr,V1,V2,V3,V4,lam1,lam2,V1P,V2P,V3P
      DOUBLE PRECISION  V4P,yb,ybP,ybPP,xb,xbP,xbPP,Ll,Lr
      PARAMETER  (eps=0.01D0,M=10.0D0,L=1.0D0,L0=0.5D0,r=0.1D0,w=10.0D0)
C
      xl = qX(1)!
      yl = qX(2)!
      xr = qX(3)!
      yr = qX(4)!
      V1 = qX(5)!
      V2 = qX(6)!
      V3 = qX(7)!
      V4 = qX(8)!
      lam1 = qX(9)!
      lam2 = qX(10)!
      V1P = qX(11)!
      V2P = qX(12)!
      V3P = qX(13)!
      V4P = qX(14)!
      yb = qX(15)!
      ybP = qX(16)!
      ybPP = qX(17)!
      xb = qX(18)!
      xbP = qX(19)!
      xbPP = qX(20)!
      Ll = qX(21)!
      Lr = qX(22)!
C
      qF(1) = V1
      qF(2) = V2
      qF(3) = V3
      qF(4) = V4
      qF(5) = V1P
      qF(6) = V2P
      qF(7) = V3P
      qF(8) = V4P
      qG(1) = -yb+r*Dsin(w*qT)
      qG(2) = -ybP+r*w*Dcos(w*qT)
      qG(3) = -ybPP-r*w*w*Dsin(w*qT)
      qG(4) = -xb+(L*L-yb*yb)**0.5D0
      qG(5) = -xbP-yb*ybP/((L*L-yb*yb)**0.5D0)
      qG(6) = -xbPP-yb*yb*ybP*ybP/((L*L-yb*yb)**1.5D0)-(ybP*ybP+yb*ybPP)
     #/((L*L-yb*yb)**0.5D0)
      qG(7) = -Ll+(xl**2.0D0+yl**2.0D0)**0.5D0
      qG(8) = -Lr+((xr-xb)**2.0D0+(yr-yb)**2.0D0)**0.5D0
      qG(9) = -V1P+(L0-Ll)*xl/(Ll)+lam1*xb+2.0D0*lam2*(xl-xr)
      qG(10) = -V2P+(L0-Ll)*yl/(Ll)+lam1*yb+2.0D0*lam2*(yl-yr)-M*eps*eps
     #/(2.0D0)
      qG(11) = -V3P+(L0-Lr)*(xr-xb)/(Lr)-2.0D0*lam2*(xl-xr)
      qG(12) = -V4P+(L0-Lr)*(yr-yb)/(Lr)-2.0D0*lam2*(yl-yr)-M*eps*eps/(2
     #.0D0)
      qG(13) = xb*xl+yb*yl
      qG(14) = (xl-xr)**2.0D0+(yl-yr)**2.0D0-L*L
      qG(15) = xbP*xl+xb*V1+ybP*yl+yb*V2
      qG(16) = 2.0D0*(xl-xr)*(V1-V3)+2.0D0*(yl-yr)*(V2-V4)
      qG(17) = xbPP*xl+2.0D0*xbP*V1+xb*V1P/((eps*eps*M/(2.0D0)))+ybPP*yl
     #+2.0D0*ybP*V2+yb*V2P/((eps*eps*M/(2.0D0)))
      qG(18) = 2.0D0*(V1-V3)*(V1-V3)+2.0D0*(xl-xr)*(V1P-V3P)/((eps*eps*M
     #/(2.0D0)))+2.0D0*(V2-V4)*(V2-V4)+2.0D0*(yl-yr)*(V2P-V4P)/((eps*eps
     #*M/(2.0D0)))
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF RHS
C     --- --- --- --- --- --- ---
C
      SUBROUTINE MATRIX(qMD,qN,qT,qX,qMA,qIOPT,qROPT,qIPAR,qRPAR,qIDID)
C
      IMPLICIT NONE
      INTEGER          qMD,qN,qI,qJ,qIOPT(*),qIPAR(*),qIDID
      DOUBLE PRECISION qT,qX(qN),qMA(qMD,qN),qROPT(*),qRPAR(*)
      DOUBLE PRECISION  eps,M,L,L0,r,w
      DOUBLE PRECISION  xl,yl,xr,yr,V1,V2,V3,V4,lam1,lam2,V1P,V2P,V3P
      DOUBLE PRECISION  V4P,yb,ybP,ybPP,xb,xbP,xbPP,Ll,Lr
      PARAMETER  (eps=0.01D0,M=10.0D0,L=1.0D0,L0=0.5D0,r=0.1D0,w=10.0D0)
C
      xl = qX(1)!
      yl = qX(2)!
      xr = qX(3)!
      yr = qX(4)!
      V1 = qX(5)!
      V2 = qX(6)!
      V3 = qX(7)!
      V4 = qX(8)!
      lam1 = qX(9)!
      lam2 = qX(10)!
      V1P = qX(11)!
      V2P = qX(12)!
      V3P = qX(13)!
      V4P = qX(14)!
      yb = qX(15)!
      ybP = qX(16)!
      ybPP = qX(17)!
      xb = qX(18)!
      xbP = qX(19)!
      xbPP = qX(20)!
      Ll = qX(21)!
```

```
      Lr = qX(22)!
C
      DO qI=1,qMD
        DO qJ=1,qN
          qMA(qI,qJ)=0.0D0
        END DO
      END DO
C
      qMA(1,1) = 1.0D0
      qMA(2,2) = 1.0D0
      qMA(3,3) = 1.0D0
      qMA(4,4) = 1.0D0
      qMA(5,5) = eps*eps*M/(2.0D0)
      qMA(6,6) = eps*eps*M/(2.0D0)
      qMA(7,7) = eps*eps*M/(2.0D0)
      qMA(8,8) = eps*eps*M/(2.0D0)
C
      RETURN
      END
C     --- --- --- --- --- --- ---
C         END OF MAT
C     --- --- --- --- --- --- ---
C
      SUBROUTINE JAC()
      END
C     --- --- --- --- --- ---
C         END OF JAC
C     --- --- --- --- --- ---
C
      SUBROUTINE SOLOUT(NACCPT,TOLD,T,X,N,NN2,NN3,NN4,CONTX,H,C1M1,C2M1,
     #
     #                  IOPT,ROPT,IPAR,RPAR,IERR)
      IMPLICIT NONE
C     .. Arguments ..
      INTEGER          NACCPT,N,NN2,NN3,NN4,IOPT(*),IPAR(*),IERR,I
      DOUBLE PRECISION TOLD,T,H,X(N),CONTX(NN4),C1M1,C2M1,
     #                 ROPT(*),RPAR(*)
C     .. Locals ..
      DOUBLE PRECISION TOUT,XOUT(N)
      INTRINSIC        DSIN,DCOS,DSQRT
      EXTERNAL         QLDENSOUTV
100   CONTINUE
      TOUT=RPAR(1)
      IF(TOUT.LE.T)THEN
        IF(IPAR(1).EQ.1) WRITE(12,80003)
80003   FORMAT('simvar={''T'';',
     #       '''xl'';',
     #       '''yl'';',
     #       '''xr'';',
     #       '''yr'';',
     #       '''V1'';',
     #       '''V2'';',
     #       '''V3'';',
     #       '''V4'';',
     #       '''lam1'';',
     #       '''lam2'';',
     #       '''V1P'';',
     #       '''V2P'';',
     #       '''V3P'';',
     #       '''V4P'';',
     #       '''yb'';',
     #       '''ybP'';',
     #       '''ybPP'';',
     #       '''xb'';',
     #       '''xbP'';',
     #       '''xbPP'';',
     #       '''Ll'';',
     #       '''Lr'';'
     #'};')
C       .. Numerical solution ..
        CALL QLDENSOUTV(TOUT,XOUT,N,NN2,NN3,NN4,T,H,CONTX,C1M1,C2M1)
C       .. Output ..
        WRITE (12,80004)
     #    IPAR(1),TOUT,
     #    (XOUT(I),I=1,N)
80004   FORMAT('simres(',I6,',:)=[',23(D23.16,','),'];')
        RPAR(1)=IPAR(1)*RPAR(2)
        IPAR(1)=IPAR(1)+1
        GOTO 100
      END IF
      END
C     --- --- --- --- --- ---
C     END OF SOLOUT
C     --- --- --- --- --- ---
```

References

[1] M. Arnold. *Zur Theorie und zur numerischen Lösung von Anfangswertproblemen für differentiell-algebraische Systeme von höherem Index*. Fortschritt-Berichte VDI Reihe 20, Nr. 264. VDI–Verlag, Düsseldorf, 1998.

[2] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*, volume 14 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 1996.

[3] E. Eich-Soellner and C. Führer. *Numerical Methods in Multibody Dynamics*. B.G.Teubner, Stuttgart, 1998.

[4] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Computer Society Pr, Linköping, 2004.

[5] C. Führer. *Differential-algebraische Gleichungssysteme in mechanischen Mehrkörpersystemen - Theorie, numerische Ansätze und Anwendungen*. PhD thesis, Technische Universität München, 1988.

[6] C.W. Gear. Differential-algebraic equation index transformations. *SIAM Journal on Scientific and Statistic Computing*, 9:39–47, 1988.

[7] C.W. Gear, B. Leimkuhler, and G.K. Gupta. Automatic integration of Euler-Lagrange equations with constraints. *Journal of Computional and Applied Mathematics*, 12/13:77–90, 1985.

[8] C.W. Gear and L.R. Petzold. ODE methods for the solution of differential/algebraic systems. *SIAM Journal on Numerical Analysis*, 21:716–728, 1984.

[9] E. Griepentrog. Index reduction methods for differential-algebraic equations. In E. Griepentrog, M. Hanke, and R. März, editors, *Seminar Notes - Berliner Seminar on Differential-Algebraic Equations (Seminarbericht 92-1)*, Humboldt Universität zu Berlin, 1992.

[10] E. Griepentrog and R. März. *Differential-Algebraic Equations and Their Numerical Treatment*, volume 88 of *Teubner-Texte zur Mathematik*. BSB B.G.Teubner Verlagsgesellschaft, Leipzig, 1986.

[11] E. Hairer, C. Lubich, and M. Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Springer-Verlag, Berlin, Germany, 1989.

[12] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, Germany, 2nd edition, 1996.

[13] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations. Analysis and Numerical Solution*. EMS Publishing House, Zürich, Switzerland, 2006.

[14] L.R. Petzold. A description of DASSL: A differential/algebraic system solver. In R.S. Stepleman and et al., editors, *Scientific Computing*, pages 65–68. North Holland, Amsterdam, 1983.

[15] L.R. Petzold. Order results for implicit Runge-Kutta methods applied to differential/algebraic systems. *SIAM Journal on Numerical Analysis*, pages 837–852, 1986.

[16] B. Simeon, C. Führer, and P. Rentrop. Diffrential-algebraic equations in vehicle system dynamics. *Surveys on Mathematics for Industry*, 1:1–37, 1991.

[17] A. Steinbrecher. QUALIDAES: A software package for the numerical integration of quasi-linear differential-algebraic equations. Technical report, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany. in preparation.

[18] A. Steinbrecher. Analysis of quasi-linear differential-algebraic equations. Technical Report 11-2006, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, 2006.

[19] A. Steinbrecher. *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems*. PhD thesis, Technische Universität Berlin, 2006.

[20] A. Steinbrecher. Remodeling of dynamical systems to benefit numerical simulations. In *Proceedings of the MATHMOD 2012 - 7th Vienna International Conference on Mathematical Modelling (MATHMOD 2012, Wien, Austria, February 15-17, 2012)*, 2012.