Evaluation and Adaptation of Techniques for Higher Index DAE with Respect to Real-Time Simulation

Jörg Frochte
joerg.frochte@hs-bochum.de
Hochschule Bochum, FB Elektrotechnik und Informatik
Höseler Platz 2, 42579 Heiligenhaus

Abstract

In this paper we will evaluate approaches to the simulation of DAE of higher order under real-time conditions. Some of these approaches are new variants of well-known methods. For the purpose of evaluation we used, for example, explicit multistep methods with and without subsequent inexact projection, and BDF approaches under real-time termination conditions. The Hardware in the loop (HIL) Simulation requires plant models which can be simulated under hard real-time conditions. Thus the combination of DAE and real-time simulation is very interesting for model-based generation of plant models because the so-called hybrid differential algebraic equations (HDAE), see e.g. [11] Appendix C, are the mathematical foundation of modeling languages such as Modelica and Simscape. So beyond the numerical properties of a technique we will watch out wherever they are suitable for model-based code generation.

1 Introduction

The modeling process takes place on several levels of abstraction: In a certain way, the top level is the least abstract because on this level one can still deal with the real object, which is set by reality. The real object serves as a model for the "physical model". This process is based on physical simplification: Geometrical details and effects, e.g. temperature dependency, need to be neglected. The mathematical representation of this physical model is now the "mathematical model". This mathematical model is taken to construct a numerical model. The model finally gained is then used in simulation studies. Modelica tools or Simscape give the modeler an opportunity to design the plan model primarily on the physical level and not on the mathematical one. Latter one is carried out with the help of modeling tools such as Simulink or by directly coding the model to C/C++. If these model based code generation tools were used in a perfect world, there would be only little need for the modeler to bother about the mathematical level. On the numerical level there would be no bother at all. Unfortunately this is not the case up to now. Modelers still need a lot of knowledge about all levels of modeling, including the numerical

level. This is particularly true for real-time simulation. The purpose of this paper is to improve this type of simulation by evaluating and adapting established methods for real-time use case. It is well known that the numerical simulation of DAE with a high index requires a high level of effort and care. As part of an automatic code generation from physical models some approaches in the literature are hard to apply, if at all. The reason is that they require more knowledge than a DAE system without modeling-context can provide to a code generation software. Even with more data about the context the skills how to treat the system are hardly capable to be set in todays software expert systems. Beyond this in some techniques, like Baumgarte's method, parameters depend on the discretization and the used integration method. This has already been pointed out by Ascher et al. [1] and seems to be still state of the art for modern simulation tools, see [14]. In consequence, we do e.g. not consider Baumgarte's method in this paper because it seems impossible up to now to choose an optimal set of parameters automatically for such a general use case. Neither do we consider approaches limited to special application areas such as mechanical models because we are looking for general approaches to multi-physical tools.

1.1 Requirements for real-time simulation

Performance must be defined with respect to whether there are real-time requirements or not. Most numerical simulations are carried-out without real-time requirements. In such cases 'performance' means that the simulation is completed as quickly as possible. This goal is frequently followed by means of continually or temporarily large time steps. The latter are the outcome of adaptive techniques. On the one hand, a time step of Δt =1e-3 seconds is in most simulations considered to be abnormally small. Hence, it would only be used in an emergency for a limited number of time steps. On the other hand, Δt =1e-3 seconds is the typical time step size of a HIL simulation with hard real-time requirements. Hence, it is fairly reasonable to explore whether techniques, generally considered as inefficient in the literature, may perform well in real-time simulation. In the context of many approaches this question has not been answered yet because real-time simulation gains only little attention in numerical research. Even less attention is paid to high index DAE in this context. For example, explicit Runge-Kutta approaches combined with projection methods are often regarded to be quite efficient techniques in the field of differential algebraic equations with a high index (see e.g. Hairer et. al. [7]). But the used Runge-Kutta style solver DOPRI5 is unsuitable for real-time simulation, even if it is the basis of the MATLAB and Simulink solver ode45 and so nearly an industrial standard because as an adaptive solver it is designed to achieve accuracy and stability by chosen variable time steps sizes. We can therefore summarize that we are looking for approaches with fixed time step. The CPU costs per time step must be predictable, and the technique must meet high requirements with respect to stability and accuracy.

1.2 Test case: Lagrangian formulation of a pendulum

As test case we are going to use the Lagrangian formulation of a mathematical pendulum with a mass and a line length of one.

$$\begin{aligned} \frac{dx}{dt} &= u \quad ; \quad \frac{du}{dt} = \lambda x \quad ; \quad \frac{dy}{dt} = v \quad ; \quad \frac{dv}{dt} = \lambda y - g \\ 0 &= x^2 + y^2 - 1 \end{aligned}$$

We take this set of equations because it is a straightforward standard example of an index 3 problem, which is, for instance, discussed in [2], [7], [5]. So there are plenty of data available for. In order to compare the results with [5] and to obtain an exact period of 2, we choose g=13.7503716373294544. The initial conditions are $x = 1$, $y = 0$, $u = 0$ and $v = 0$. It is well known (see e.g. [7] p. 454ff.) that this index-3 formulation can be transferred to index 1- or index-0 formulation by repeatedly differentiating the algebraic constraint and replace it by one of its derivatives.

$$\begin{aligned} \text{Index} = 3 \qquad 0 &= x^2 + y^2 - 1 \\ \text{Index} = 2 \qquad 0 &= x\frac{dx}{dt} + y\frac{dy}{dt} \\ \text{Index} = 1 \qquad 0 &= \lambda x^2 + \lambda y^2 - gy + u^2 + v^2 \\ \text{Index} = 0 \qquad 0 &= x^2\frac{d\lambda}{dt} + 2\lambda xu + y^2\frac{d\lambda}{dt} + 2\lambda yv - gv + 2u\frac{du}{dt} + 2v\frac{dv}{dt} \end{aligned}$$

As we can see, it is in the Index-1 formulation easy to express lambda out of the other values:

$$\lambda := \frac{-gy + u^2 + v^2}{x^2 + y^2}$$

In other words, no matter if we choose to use an index-1 integrator such as IDA or a standard ODE integrator like CVODE - both are part of the Sundial software collection [8]. Generally it makes sense to stop at an index-1 formulation, as long as the formulation can be transformed into a form that is compatible to an ODE. But first we should explain why the way most Modelica tools deal with these kinds of equations seems to be unsuitable for us.

2 Considered and adapted techniques

2.1 Simulation using the Pantelides algorithm and dummy derivatives

Most Modelica tools use a mixture of the dummy derivatives approach [10] with the Pantelides Algorithm [12], which was designed to initialize DAE. The resulting algorithm is described by Cellier and Koffmann (see [2]). One reason for using it might be that it leads to a straightforward code generation that can be used with standard ODE integrators. The discussion by Cellier and Koffmann ([2], chapter 7.8) e.g. shows some problems that occur when this algorithm is applied to the

index-3-formulation of the pendulum. To cut a long story short: it fails completely, no matter if it is combined with techniques like inline- or mixed-mode integration. One may argue, as Cellier and Koffmann do, that a smart modeler should use the following index-1 formulation to describe the pendulum.

$$\begin{aligned} \frac{d\varphi}{dt} = \theta \quad &; \quad \frac{d\theta}{dt} = -g\sin(\varphi) \\ x = \cos(\varphi + 3\pi/2) \quad &; \quad y = \sin(\varphi + 3\pi/2) \end{aligned}$$

On the one hand, this represents a fairly sensible strategy and on the other hand, this means that we limit our options in the fields of library and model construction. Furthermore, we postulate that users are able to identify the point at which the differential index of a formulation rises especially in a more complex and connected model. Relying on the skilled modeler with deep knowledge in all modeling levels seems to be inappropriate. This is particularly true when the objective lies in the model-based generation of a software from a mathematical or physical modeling level. However, not all hope is lost because the Pantelides variant from [2] can handle this, at least for considerable number of problems, using a technique known from Dymola as "dymanic state selection". In the context of real-time simulation this workaround is associated with a number of drawbacks. In the very instance in which a state-switch becomes necessary, a multiple of the original CPU time is needed per time step. In general, this causes an overrun in the real-time simulation. Let us recapitulate that predictability of CPU-time is critical for real-time simulation. Otherwise we would always have to calculate our resources for the biggest possible CPU resource need during the simulation. Another point to be made is that this Pantelides variant is not fully theoretically understood. For example: For a while a lot of people thought that the structural index used by the Pantelides algorithm is +/-1 the same as the differential index of the numerical theory, but Reissig, Martinson and Barton manage to demonstrate (see [13]) that in some cases an equation of a differential index 1 may have an arbitrarily high structural index. Now it is proved that the Pantelides algorithm might transform a totally harmless set of equations into an unsuitable appearance. So in fact, the Pantelides algorithm and its variants - with or without dummy derivatives - are no silver bullets for higher index problems. Thus, it seems that the real-time simulation of DAE with higher index calls in for alternatives. We assume that they are already on the market but have not yet been modified for the use under real-time conditions. For this reason we will examine approaches and present slight modifications for real-time simulation in the following section.

2.2 Integration of the index-1- formulation with Multistep Method

So now we use the Index-1/Index-0 formulation with standard ODE-Solvers.

$$\lambda \quad := \quad \frac{-gy + u^2 + v^2}{x^2 + y^2}$$

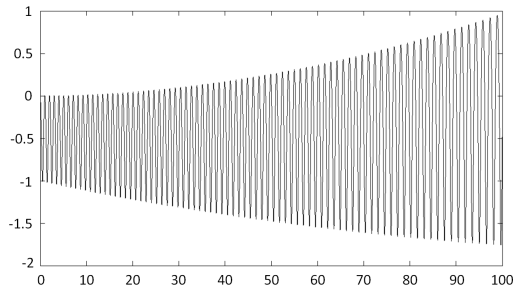

Figure 1: y-coordinate of the pendulum computed with the explicit Euler algorithm $\Delta t = 0.001$

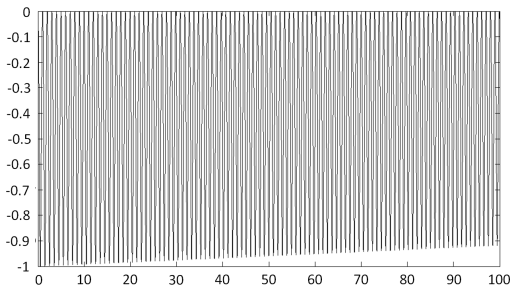

Figure 2: BDF(1), drift effect in y-coordinate with $\Delta t = 6.25e - 5$

$$\frac{dx}{dt} = u \quad ; \quad \frac{du}{dt} = \lambda x \quad ; \quad \frac{dy}{dt} = v \quad ; \quad \frac{dv}{dt} = \lambda y - g$$

A lot of people associate real-time and HIL with the explicit Euler algorithm. One reason for this might be that it is easy to generate code with an embedded explicit Euler from Simulink using the RTW. Unfortunately, the results for stiff systems are often devastating, see figure 1. As a consequence, ".. for physical models, MathWorks recommends implicit solvers, such as ode14x, ...", [9], or to switch for a better performance to local approaches. In this approach the local solver is by default the implicit Euler, in other words BDF(1). So maybe the implicit Euler is an approach for our ODE above? In the numerical community it is quite well known that it is not. The implicit Euler is afflicted with the drift effect (see figure 2), and unstable for this problem. But there are numerous alternatives with a fixed time step size.

2.2.1 Adams–Bashforth Methods

The failure experienced when using the Euler does not mean that all explicit methods fail. Multi-step methods like the Adams–Bashforth Methods, see e.g. [6] p. 357f. or [2] p. 122f. for details, share a number of features that we require: Firstly CPU costs are low, just one evaluation of the given right side f, and secondly one can achieve high accuracy if the solution is smooth enough. In general the third order is a good compromise between stability and accuracy. So we try

$$y_{n+1} = y_n + h\left(\frac{23}{12}f(t_n, y_n) - \frac{16}{12}f(t_{n-1}, y_{n-1}) + \frac{5}{12}f(t_{n-2}, y_{n-2})\right) .$$

The advantage of multi-step methods lies in the use of the history of the simulation. A minor drawback might be the start phase. At the first time step there obviously is no simulation history to rely on. For this reason we have two major strategies: We start with an order 1 approach for the first time step, next perform an order 2 approach and then continue the simulation with the order 3. Unfortunately, one

loses some accuracy in this starting phase because of the two steps with lower order. To minimize this effect one should split the first step into three smaller ones. For a real-time simulation this probably results in an overrun in the first time step, but on most HIL-Systems one overrun in the first time step is no problem (see e.g [4]). The other strategy is to assume that the initial situation has endured for a longer period of time, hence creating an artificial history. For our example this means $x(-2\Delta t) = x(-\Delta t) = x(0) = 1$; $y(-2\Delta t) = y(-\Delta t) = y(0) = 0$ and $u(-2\Delta t) = u(-\Delta t) = u(0) = 0$; $v(-2\Delta t) = v(-\Delta t) = v(0) = 0$. In other situations, however, its not physical. In other words, it cannot be generalized. The reason why we use it is to avoid initialization effects for the numerical test.

2.2.2 Backward Differentiation Formulas

Backward Differentiation Formulas, for a detailed description see [7] p. 246ff. or [2] p. 128ff., are in contrast to Adams–Bashforth Methods implicit linear multi-step methods. These methods are particularly suitable for solving stiff differential equations. Our ODEs generated out of DAE are very often fairly stiff, so they seem to be a natural approach. However, the use of implicit methods is objectionable because they seem to undermine the predictability of CPU resources. They would iterate until a given error boundary is reached. Knowing this, we use a fairly widespread approach to make the algorithm predictable. We limit the number of Newton-iterations. The Number of Newton iterations needed to fulfil a given limit of error depends, of course, both on the initial value and on the problem as a given, unchangeable factor. If we take the value from the last time step as the initial value for the next time step, it becomes clear that a smaller time step size leads to better initial values. In our test case with time step size Δt =1e-3 the average number of Newton iterations per time step to achieve a residual smaller than 1e-10 is 2. Because of its good mixture of stability and accuracy we choose the BDF(3) scheme.

$$\frac{11}{6}y_{n+1} - 3y_n + \frac{3}{2}y_{n-1} - \frac{1}{3}y_{n-2} - hf(t_{n+1}, y_{n+1}) = 0$$

For the initial values back in time we use the same approach as for the Adams–Bashforth Methods. Higher order BDF methods have good properties concerning DAE of low order index anyway. That is one reason why they are the basis of the DAE-Index-1-solver DASSL. Therefore it makes sense to test them for our transformed index-1 problem. The first challenge is: Can we avoid a drift effect using the typical HIL time step size? As you can see from the table in section 4 the approach looks very stable.

3 An inexact projection approach

Concerning stability, numerical literature identifies a residual risk in the restrictive application of high order BDF or Adams-Bashforth methods to high order DAE.

Maybe the typical small time step sizes in real-time simulation save our necks but we would like to explore whether stability can be improved by projection techniques (see e. g. [7], p. 470ff.). Unfortunately, the methods applied to this problem generally make use of information that an automatic code generator often does not have. The process of embedding these data is quite difficult. The successful application of these approaches, e.g. discussed in the paper [5], to a huge set of equations, which are not sorted by application domain, derived from a multi-physical tool such as Modelica tools, seems very unlikely at this moment. It seems easier to use a technique based on the concept of dealing with high order DAE by using overdetermined approaches (see e.g. [7] p. 477ff.). In this case one just assembles all the equations that arise during the generation of the index-1- formulation. This leads to a set of equations, containing more equations than degrees of freedom. In our test case this will look like this:

$$\begin{aligned} \frac{dx}{dt} = u \; ; \; \frac{dy}{dt} = v \quad ; \quad \frac{du}{dt} = \lambda x \; ; \; \frac{dv}{dt} = \lambda y - g \\ 0 = x^2 + y^2 - 1 \quad ; \quad 0 = xu + yv \end{aligned}$$

This analytical problem is in actual fact not overdetermined because we can find a solution, so that the residual is zero. The set of equations can be solved exactly. But we need to discretize these equations to simulate them. We do that using the BDF approach which leads to

$$\frac{dy}{dt} \approx \frac{11}{6} y_{n+1} - 3y_n + \frac{3}{2} y_{n-1} - \frac{1}{3} y_{n-2}$$

and similar expressions for the other derivatives. The result is a discrete system which cannot be solved exactly. However, this is not our goal anyway. We will perform only a very limited number of iterations of a kind of Gauss–Newton algorithm. The result is an inexact projection approach.

4 Results

We can easily compare the results at $t = 100$ because we know that the period of the solution is 2. There we define $RES := |x^2 - y^2 - 1|$ and $ERR := \Delta x + \Delta y + RES$. In the column $Newton - Steps$ the first summand is the number of steps per time step for the BDF part and the second for the inexact projection. Beyond this, we compute a reference solution based on the formulation presented in section 2.1. The reference solution is computed by ode15s. The parameter 'RelTol' is set to 1e-12 and AbsTol to 1e-14. The goal of this reference solution is to give us a feeling for the global error and its development. In Figure 3 we can see the difference between the Adams–Bashforth(3) followed by a single inexact projection step. In 4 the same is displayed for the BDF(3), with only one single newton step followed by a single inexact projection step. Both seems to be very stable. While the implicit approach performs better for $t = 100$, as one can see from the table above, the Adams–Bashforth(3) variant seems altogether preferable.

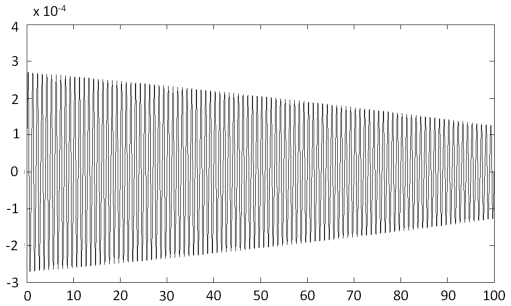

Figure 3: y-coordinate: Comparison Adams–Bash.(3) + projection approach, $\Delta t = 0.00025$ with ode15s

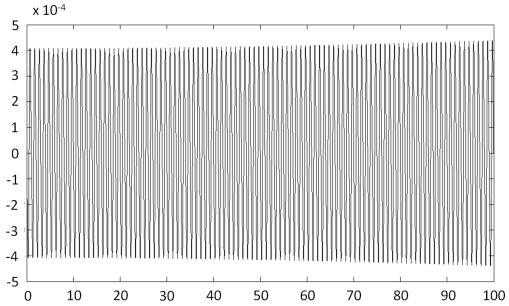

Figure 4: y-coordinate: Comparison Inexact BDF(3) + projection approach, $\Delta t = 0.00025$ with ode15s

Method	Newton -Steps	Pro- jection	Δt or Tol	Δx	Δy	RES	ERR
Results achieved with the methods discussed above							
BDF(3)	2+0	No	Δt = 1e-3	1.8e-5	5.8e-6	3.7e-5	6.2e-5
Precond-BDF(3)	1+0	No	Δt = 1e-3	2.2e-5	1.2e-5	4.5e-5	8.0e-5
Precond-BDF(3)	1+3	Yes	Δt = 1e-3	1.1e-9	4.4e-5	2.8e-10	4.4e-5
Precond-BDF(3)	1+3	Yes	Δt = 5e-4	2.6e-11	5.8e-6	1.7e-11	5.8e-6
BDF(3)	1+0	No	Δt = 1e-3	6.4e-5	7.6e-5	1.2e-4	2.7e-4
BDF(3)	1+0	No	Δt = 5e-4	9.4e-6	1.1e-5	1.8e-5	3.9e-5
BDF(3)	1+0	No	Δt = 2.5e-4	4.2e-6	5.7e-6	8.5e-6	1.8e-5
Adams–Bash.(3)	0+0	No	Δt = 1e-3	2.1e-5	1.1e-5	4.2e-5	7.5e-5
Adams–Bash.(3)	0+0	No	Δt = 5e-4	2.6e-6	9.6e-7	5.3e-6	8.9e-6
Adams–Bash.(3)	0+0	No	Δt = 2.5e-4	3.3e-7	8.0e-8	6.6e-7	1.0e-6
Adams–Bash.(3)	0+1	Yes	Δt = 1e-3	2.4e-9	7.1e-5	2.9e-10	7.1e-5
Adams–Bash.(3)	0+1	Yes	Δt = 5e-4	7.1e-11	1.2e-5	1.8e-11	1.2e-5
Adams–Bash.(3)	0+1	Yes	Δt = 2.5e-4	2.1e-12	2.3e-6	1.1e-12	2.3e-6
BDF(3)	1+1	Yes	Δt = 1e-3	1.0e-9	4.4e-5	1.7e-10	4.4e-5
BDF(3)	1+1	Yes	Δt = 5e-4	2.2e-11	5.8e-6	1.0e-11	5.8e-6
BDF(3)	1+1	Yes	Δt = 2.5e-4	5.9e-13	7.2e-7	6.7e-13	7.2e-7
Results from [5] with a different projection approach							
MKS-DAEOL	-	Yes	Tol =1e-5	2.4e-8	2.2e-4	7.2e-10	2.2e-4
MKS-DAEOL	-	Yes	Tol =1e-6	9.7e-9	1.4e-4	5.9e-11	1.4e-4

The BDF (3), however, seems to be more stable when it is used in a long term test with a maximum duration of one hour. Its tendency to run out of phase is more pronounced as in the case of the BDF(3), which generally steals a bit of system energy. Beyond this we see that the inexact approach with only one newton step and one projection step per time step performs very well for these typical HIL time step sizes. Moreover it shows very good reduction rates near the third order in time. We also tried to increase the performance or accuracy of the BDF approach by preconditioning it with the result of the Adams–Bashforth method, but this did not work well. In general both the explicit and the implicit approach are improved applying a single projection step using over-determinated system.

5 Proposal for a Parallel Real-time Algorithm

Our survey demonstrates that a high order BDF or Adams–Bashforth approach combined with a single projection step is the best technique for the problem class discussed in this paper. Wherever we use an explicit or an implicit approach we need at least to assemble a Jacobi matrix and factorize it into a QR- or a LU-form, respectively. If we performed an assembling and a decomposition in every iteration

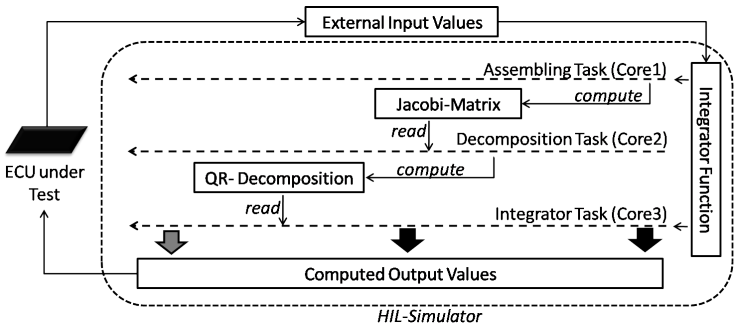

Figure 5: Tasks in a Parallel Real-time approach

step this would probably be hard to achieve in 1ms real-time. Thus, we intend to turn to a simplified Newton approach as described, for instance in [3] p. 52ff. However, the shift to a simplified Newton approach will not lead to a real-time compatible technique. Beyond this one has to use the power of current multicore architectures also that tend to be the standard for HILs as well. With four cores it is possible to keep all I/O and OS aspects on one core, e.g. Core4, and use the remaining three cores for the computation. The basic architecture is displayed in figure 5. As a result, the Jacobi matrix is only updated in the simplified Newton approach if the computation is finished. This procedure results in an algorithm whose turnaround is highly predictable time in real-time simulation.

6 Conclusion and Future Prospects

We gave an outlook on how we hope to produce an algorithm that can use the power of multi-core architectures to simulate industrial size DAE systems in real-time without making too much use of the unreliable approaches discussed in section 2.1 with their drawbacks and uncertainties. Of course, for problems with a low structural index of 0 or 1 it will always be more efficient to use the approaches of section 2.1. One of our future prospects is to verify that accuracy and stability can be retained using this parallel approach. Moreover, a deeper analysis of the process will be considered in future publications. We have shown that by using common multistep methods, explicit and implicit ones, it is even possible to simulate high order DAE with a very limited number of iterative steps. A single projection step can improve the results significantly. But beyond this, the presented approaches might

support a least square modeling paradigma. When ODE integrators are regarded as preconditioners for an overdetermined system, this obviously leads to a direct modeling of overdetermined systems, which may be quite useful in some applications.

References

[1] U. M. Ascher, H. Chin, and S. Reich. Stabilization of daes and invariant manifolds. *Numer. Math*, 67:131–149, 1993.

[2] F. c. E. Cellier and E. Kofman. *Continuous system simulation.* New York, NY: Springer, 2006.

[3] P. Deuflhard. *Newton methods for nonlinear problems. Affine invariance and adaptive algorithms.* Berlin: Springer, 2004.

[4] dSPACE GmbH. *dSPACE FAQ 242 Handling Overrun Situations*, 2011.

[5] E. Eich. Convergence results for a coordinate projection method applied to mechanical systems with algebraic constraints. *SIAM J. Numer. Anal.*, 30(5):1467–1482, 1993.

[6] E. Hairer, S. P. Nø rsett, and G. Wanner. *Solving ordinary differential equations. I: Nonstiff problems.* Berlin: Springer, 2010.

[7] E. Hairer and G. Wanner. *Solving ordinary differential equations. II: Stiff and differential-algebraic problems.* Berlin: Springer, 2010.

[8] A. Hindmarsh, P. Brown, K. Grant, R. S. S.L. Lee, D. Shumaker, and C. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.

[9] The MathWorks, Inc. *MATLAB 2011a Documentation, Simscape*, 2011.

[10] S. E. Mattsson and G. Soederlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput.*, 14(3):677–692, 1993.

[11] Modelica Association. *Modelica Language Specification - Version 3.2*, 2010.

[12] C. C. Pantelides. The consistent initialization of differential algebraic systems. *SIAM J. Sci. Stat. Comput.*, 9(2):213–231, 1988.

[13] G. Reissig, W. S. Martinson, and P. I. Barton. Differential-algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM J. Sci. Comput.*, 21(6):1987–1990, 2000.

[14] G. D. Wood and D. C. Kennedy. Simulating mechanical systems in Simulink with SimMechanics. *Technical Report 91124v00*, 2003.