Half-explicit Runge-Kutta methods for overdetermined semi-implicit Differential-Algebraic Equations

Simon Michael Bäse

13. April 2017

Abstract

The numerical solution of differential-algebraic equations can be expensive, if the underlying model has high dimensions. When using Runge-Kutta methods, large nonlinear systems of equations have to be solved during the iteration. The main focus for this thesis is to study half-explicit Runge-Kutta methods for a special class of differential-algebraic equations to make these computations more efficient. We exploit the structure and properties of the given differential-algebraic equations to construct an algorithm that allows to solve reduced nonlinear systems of equations. The algorithm is tested with multiple examples from the fields of mechanical, electrical and chemical sciences to validate the approach.

Contents

1 Introduction **1**

2 Differential-Algebraic Equations **2**
2.1 Preliminaries . . . 2
2.2 Structured Problems . . . 3

3 Half-Explicit Runge-Kutta Methods **10**
3.1 Definition . . . 10
3.2 Selection Procedure . . . 13
3.3 Numerical Study . . . 20

4 Implementation and Examples **25**
4.1 Implementation . . . 25
4.1.1 Input Values . . . 25
4.1.2 Runge-Kutta Methods . . . 27
4.1.3 Selectors . . . 28
4.1.4 Newton Methods . . . 29
4.1.5 Adaptive Step Size Control . . . 30
4.2 Academic Examples . . . 31
4.2.1 Differential Equations . . . 31
4.2.2 Algebraic Equations . . . 31
4.2.3 Differential-Algebraic Equations . . . 32
4.2.4 Selector Change Example . . . 34
4.3 Mathematical Pendulum . . . 36
4.4 Modified Pendulum in Changing Wind Conditions . . . 42
4.5 Spring-Mass Chain . . . 44
4.6 Linear Circuit with one CV Loop . . . 48
4.7 Chemical Akzo Nobel Problem . . . 51

5 Summary **54**

References **56**

Notation

This list may be used as a short introduction to the used notation. Please refer to the related definitions for the explanation of the used sub- und superscripts, as well as the distinction of variables and symbols by accents.

t	independent variable, mostly perceived as time
x, $\mathcal{X}$	state, mostly used in autonomous form representing $x(t)$
$\dot{x}$, $\dot{\mathcal{X}}$	first derivative of $x(t)$ with respect to t, i.e. $\dot{x}(t) = \frac{d}{dt}x(t)$
E	continuous mapping, leading matrix
f	continuous mapping, sufficiently differentiable, right-hand side
g	hidden constraints
$\mathcal{J}$	matrix of partial derivatives $\frac{\partial}{\partial x}g(x,t)$, called Jacobian
L	lower-triangular matrix
U	matrix which is partly upper-triangular
$\mathcal{R}$, $\mathcal{S}$	matrices, representing parts of U
P, Q	permutation matrices
I	identy matrix
$\mathbb{R}$	set of real numbers
$\mathbb{D}$	open set
$\mathbb{I}$	domain of independent variable t
$\mathcal{C}$, $\mathcal{C}^1$	set of continuous and continuously differentiable functions
φ^a	selector for algebraic components
φ^{a_r}	selector for required algebraic components
φ^d	selector for differential components
Φ^a	algebraic columns selector
Φ^d	differential columns selector
ν_c	maximal constraint level
ν_s	strangeness-index
$a_{i,j}, b_j, c_i$	coefficients of Butcher-tableau of Runge-Kutta method
s	stages of Runge-Kutta method
h	step size to discretize time domain
p	convergence order
$\mathcal{O}$	Landau symbol
$\Vert\,.\,\Vert$	norm and corresponding matrix norm
$\operatorname{rank}(\,.\,)$	rank of matrix
$(\,.\,)^T$	transpose of matrix or vector
$\oplus$	operation, refer to Notation 3.2
$A(i\!:\!j, k\!:\!l)$	matrix colon notation, refer to Notation 3.10

1 Introduction

The numerical simulation of dynamical systems with differential-algebraic equations increasingly gains importance and is widely discussed across many disciplines. For instance, these models emerge when describing problems in mechanical, electrical or chemical engineering [BH93,BCP96,KM06,SPW12]. An important topic to consider is how complex systems can be solved efficiently - not only regarding execution speed but also memory usage. The translation of classical Runge-Kutta methods to differential-algebraic equations leads to the problem of solving high-dimensional nonlinear systems of equations [KM06]. If the structure of the given differential-algebraic equation can be utilized, the use of half-explicit Runge-Kutta methods can reduce the dimensions of these nonlinear systems [BH93]. In the course of this work we will show how to use the properties of overdetermined systems for half-explicit methods. The aim is to construct an algorithm that carries over most of the classical settings of Runge-Kutta methods and is applicable for a large set of differential-algebraic equations.

This thesis is organized as follows. Chapter 2 contains the definitions of basic terms for differential-algebraic equations that are used throughout the thesis. We present an approach to regularize quasi-linear differential-algebraic equations to derive the desired formulation of overdetermined semi-implicit differential-algebraic equations. Further, we gain understanding of the hidden constraints within the given system. In Chapter 3 we describe the generalization of classical Runge-Kutta methods to suit differential-algebraic equations. We then modify this approach to obtain half-explicit Runge-Kutta methods for overdetermined semi-implicit differential-algebraic equations by exploiting the properties and structure. A problem that arises with half-explicit methods is to determine which components of the current state should be treated algebraically and which differentially. We introduce a selection procedure that determines these automatically. Thereafter, a short numerical study to validate the approach is given. In Chapter 4 we present the implementation and describe the setup and note tricks to enhance the performance of the program. Then, we discuss the behavior of multiple examples with different settings. The results are documented with figures and tables. Chapter 5 provides a summary of the thesis and gives an outlook for the next steps.

2 Differential-Algebraic Equations

2.1 Preliminaries

A general *initial value problem for differential-algebraic equations* can be written in the form

$$F(x(t), \dot{x}(t), t) = 0, \tag{2.1a}$$

$$x(t_0) = x_0, \tag{2.1b}$$

where $F : \mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_{\dot{x}} \to \mathbb{R}^u$ is a continuous function, $\mathbb{D}_x, \mathbb{D}_{\dot{x}} \subset \mathbb{R}^n$ are open, $\mathbb{I} = [t_0, t_f] \subseteq \mathbb{R}$ is a compact interval and $x_0 \in \mathbb{D}_x$. Here, (2.1a) is called *state equation*, (2.1b) is called *initial condition* and x_0 is called *initial value*. The continuous differentiable function $x : \mathbb{I} \to \mathbb{D}_x$ is called *state* and $t \in \mathbb{I}$ is called *independent variable*.

Definition 2.1 (Solution)**.** *A function $x \in \mathcal{C}^1(\mathbb{I}, \mathbb{R}^n)$ is called* solution *of (2.1a), if it satisfies (2.1a) pointwise. It is called* solution of the initial value problem *(2.1), if it additionally satisfies (2.1b).*

In the following, we are going to assume that the given problem has a solution. To continue the discussion of different solvability concepts, existence and uniqueness of solutions we refer to [KM06]. For the numerical treatment it will be important to initiate the integration process with an initial condition that corresponds to a solution. That is why we introduce the following term.

Definition 2.2 (Consistency)**.** *An initial condition (2.1b) is called* consistent *with (2.1a), if the corresponding initial value problem (2.1) has at least one solution.*

There are many different indices to classify differential-algebraic equations regarding the level of difficulty to find their solution either analytically or numerically [KM06, Meh12, Ste06]. These difficulties can arise from the underlying constraints emerging from the algebraic equations. Here, we only give a vague definition of the strangeness-index to understand how such a value can be derived. For a thorough introduction and examination of the strangeness concept refer to [KM06].

Definition 2.3 (Strangeness-index)**.** *Given a differential-algebraic equation (2.1a), the minimum number of times ν_s all or part of F has to be transformed and differentiated to obtain a reformulated system, where one part states all constraints and another part describes the dynamical behavior is called* strangeness-index*. If $\nu_s = 0$ the differential-algebraic equation is called* strangeness-free*.*

2.2 Structured Problems

Consider differential-algebraic equations of the form

$$E(x,t)\,\dot{x}(t) \;=\; f(x,t), \tag{2.2}$$

where $E \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n,n})$ is called *leading matrix*, $f \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^n)$ is called *right-hand side* and $\mathbb{D}_x \subset \mathbb{R}^n$. Systems of the form (2.2) are called *quasi-linear differential-algebraic equations*. They can be defined more generally, where the leading matrix is not square. For a thorough discussion on the properties and numerical treatment of quasi-linear differential-algebraic equations we refer to [Ste06]. Generally, these systems can lead to problems of higher strangeness-index, which results in complications during the numerical integration [BCP96, KM06]. Therefore, we will present a regularization procedure to reduce the strangeness-index of the system [Ste06, Ste15].

Procedure 2.4. We will follow the formulation in [Ste15]. Consider a quasi-linear differential-algebraic equation (2.2) which has a unique solution. Assume that $\operatorname{rank}(E(x,t))$ is constant for all $(x,t) \in \mathbb{D}_x \times \mathbb{I}$ and f is sufficiently smooth.

Initialization:
Set $i = 0$, $E^0(x,t) = E(x,t)$, $f^0(x,t) = f(x,t)$ and $\mathbb{M}_{-1} = \mathbb{D}_x \times \mathbb{I}$.

Iteration:
Start each iteration step with the initial system $E^i(x,t)\,\dot{x} = f^i(x,t)$.

1. If $E^i(x,t)$ is nonsingular for all $(x,t) \in \mathbb{M}_{i-1}$ the procedure terminates with $\nu = i$.

2. Suppose there exists a matrix function $Z^i \in \mathcal{C}(\mathbb{M}_{i-1}, \mathbb{R}^{n,n})$ which is nonsingular for all $(x,t) \in \mathbb{M}_{i-1}$ such that

$$Z^i(x,t)E^i(x,t) = \begin{bmatrix} \tilde{E}_1^i(x,t) \\ 0 \end{bmatrix} \quad \text{and} \quad Z^i(x,t)f^i(x,t) = \begin{bmatrix} \tilde{f}_1^i(x,t) \\ \tilde{f}_2^i(x,t) \end{bmatrix}$$

with $\operatorname{rank}(\tilde{E}_1^i(x,t)) = \operatorname{rank}(E^i(x,t))$ for all $(x,t) \in \mathbb{M}_i$, where $\mathbb{M}_i = \{(x,t) \in \mathbb{M}_{i-1} : 0 = \tilde{f}_2^i(x,t)\}$.

3. Partition the initial system by multiplying both sides with Z^i from the left to obtain

$$\begin{bmatrix} \tilde{E}_1^i(x,t) \\ 0 \end{bmatrix} \dot{x} = \begin{bmatrix} \tilde{f}_1^i(x,t) \\ \tilde{f}_2^i(x,t) \end{bmatrix},$$

where $0 = \tilde{f}_2^i(x,t)$ represent the constraints of level i.

4. Replace the constraints by their derivative and reorder the system of Step 3 to
$$\begin{bmatrix} \tilde{E}_1^i(x,t) \\ \frac{\partial}{\partial x}\tilde{f}_2^i(x,t) \end{bmatrix} \dot{x} = \begin{bmatrix} \tilde{f}_1^i(x,t) \\ -\frac{\partial}{\partial t}\tilde{f}_2^i(x,t) \end{bmatrix}.$$

5. Finally, continue the iteration with
$$E^{i+1}(x,t) = \begin{bmatrix} \tilde{E}_1^i(x,t) \\ \frac{\partial}{\partial x}\tilde{f}_2^i(x,t) \end{bmatrix} \text{ and } f^{i+1}(x,t) = \begin{bmatrix} \tilde{f}_1^i(x,t) \\ -\frac{\partial}{\partial t}\tilde{f}_2^i(x,t) \end{bmatrix},$$
and set $i = i + 1$. $\triangleleft$

Remark 2.5. The existence of the matrix function Z^i in Step 2 can be guaranteed if the leading matrix $E^i(x,t)$ has constant rank for all $(x,t) \in \mathbb{M}_{i-1}$ and $\mathbb{M}_{i-1}$ is smooth (see [Ste06] Remark 3.5.12). $\triangleleft$

Definition 2.6 (Hidden constraints)**.** *Assume that Procedure* 2.4 *applied to the quasi-linear differential-algebraic equation (2.2) terminates in iteration step $i = \nu$. Then $\nu_c = \nu - 1$ is called* maximal constraint level *and the constraints*
$$0 = \begin{bmatrix} \tilde{f}_2^0(x,t) \\ \vdots \\ \tilde{f}_2^{\nu_c}(x,t) \end{bmatrix} =: g(x,t),$$
where $g \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^m)$, are called hidden constraints *of the quasi-linear differential-algebraic equation (2.2).*

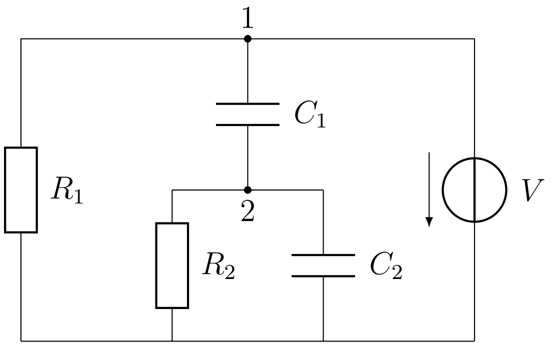


Figure 1: Linear Circuit with one CV Loop

Example 2.7 (Linear Circuit with one CV Loop)**.** The electrical circuit in Figure 1 can be modeled with the equations

$$\begin{aligned} 0 &= \dot{q}_1 + e_1 + i_V, \\ 0 &= \dot{q}_2 - \dot{q}_1 + e_2, \\ 0 &= e_1 - \sin(100t), \\ 0 &= q_1 - e_1 + e_2, \\ 0 &= q_2 - e_2, \end{aligned} \tag{2.3}$$

where the unknowns q_1 and q_2 are the charges of the capacitors, e_1 and e_2 are the potentials in the nodes $\{1, 2\}$ and i_V is the current through the voltage source. We assume that $C_1 = C_2 = R_1 = R_2 = 1$ and that the voltage of the circuit is given by $v(t) = \sin(100t)$. For more details about the model we refer to [Bäc07]. We will now derive the hidden constraints of the system with Procedure 2.4.

Initialization:

Set

$$i = 0, \quad E^0(x,t) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad f^0(x,t) = \begin{bmatrix} e_1 + i_V \\ e_2 \\ e_1 - \sin(100t) \\ q_1 - e_1 + e_2 \\ q_2 - e_2 \end{bmatrix}.$$

Iteration:

$E^0(x,t)$ and $f^0(x,t)$ are already in the desired form and we can choose Z^0 as the identity matrix. Here

$$0 = \begin{bmatrix} e_1 - \sin(100t) \\ q_1 - e_1 + e_2 \\ q_2 - e_2 \end{bmatrix} = \tilde{f}_2^0$$

are the constraints of level 0. Moving forward to Step 4 we get

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{i}_V \end{bmatrix} = \begin{bmatrix} e_1 + i_V \\ e_2 \\ 100\cos(100t) \\ 0 \\ 0 \end{bmatrix}.$$

Set

$$i = 1, \quad E^1(x,t) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 \end{bmatrix} \quad \text{and} \quad f^1(x,t) = \begin{bmatrix} e_1 + i_V \\ e_2 \\ 100\cos(100t) \\ 0 \\ 0 \end{bmatrix}.$$

Since $E^1(x,t)$ is singular the iteration does not terminate. We can choose

$$Z^1(x,t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 1 \end{bmatrix}$$

and partition the system in

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{i}_V \end{bmatrix} = \begin{bmatrix} e_1 + i_V \\ e_2 \\ 100\cos(100t) \\ 0 \\ 2e_1 + 2i_V + e_2 + 100\cos(100t) \end{bmatrix}.$$

We obtain

$$0 = 2e_1 + 2i_V + e_2 + 100\cos(100t) = \tilde{f}_2^1$$

as the constraint of level 1. Differentiating the constraint in Step 4 yields

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{i}_V \end{bmatrix} = \begin{bmatrix} e_1 + i_V \\ e_2 \\ 100\cos(100t) \\ 0 \\ 10000\sin(100t) \end{bmatrix}.$$

Here, we see that the leading matrix is non-singular and the iteration will terminate for $i = 2$. Hence, this system has the maximal constraint level $\nu_c = 1$. ◁

Remark 2.8. Note that the strangeness-index of (2.2) is closely related to the maximal constraint level and we find $\nu_s = \max\{0, \nu_c\}$ for most uniquely solvable quasi-linear differential-algebraic equations, where $m = n$. This becomes untrue if the leading matrix $E(x,t)$ is not square, where Procedure 2.4 has to be generalized. In that case the strangeness-index is an upper bound for the maximal constraint level [Ste06]. ◁

Definition 2.9 (Semi-implicit differential-algebraic equations)**.** *Consider differential-algebraic systems of the form*

$$E(x,t)\,\dot{x}(t) \;=\; f(x,t), \tag{2.4a}$$
$$0 \;=\; g(x,t), \tag{2.4b}$$

where $E \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n,n})$*,* $f \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^n)$ *and* $g \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^m)$ *with* $\mathbb{D}_x \subset \mathbb{R}^n$*. Assume that (2.4) has a unique solution* $x \in \mathcal{C}^1(\mathbb{I}, \mathbb{D}_x)$ *such that* $\{(x,t) \in \mathbb{D}_x \times \mathbb{I} : 0 = g(x,t)\}$ *for all* $t \in \mathbb{I}$*. Then (2.4) is called* (overdetermined) semi-implicit differential-algebraic equation*.*

Definition 2.10 (Regularization)**.** *A differential-algebraic equation*

$$\hat{E}(\hat{x},t)\,\dot{\hat{x}}(t) \;=\; \hat{f}(\hat{x},t), \tag{2.5}$$

is called regularization *of (2.2) if*

1. *there exists a unique bijective mapping* $\Psi : \mathbb{D}_x \to \mathbb{D}_{\hat{x}}$ *such that* $\Psi(x)$ *is a solution of (2.5) and* $\Psi(\hat{x})^{-1}$ *is a solution of (2.2),*
2. *the number of hidden constraints in (2.5) is smaller than the number of hidden constraints in (2.2).*

Proposition 2.11. *Assume that Procedure* 2.4 *is applied to the quasi-linear differential-algebraic equation (2.2). Then the differential-algebraic equation (2.4) is a regularization of (2.2), where (2.4a) equals (2.2) and (2.4b) are the hidden constraints of (2.2).*

Proof. The assertion follows directly from the construction in Procedure 2.4. The solution x of (2.2) is invariant under nonsingular transformation and differentiation of constraints (see [Ste06] Lemma 3.5.8 and Lemma 3.5.10). Also, with each iteration step we find at least one unknown (hidden) constraint. □

Remark 2.12. If a semi-implicit differential-algebraic system is derived as in Procedure 2.4 every consistent initial value x is in the set $\mathbb{M} = \{\mathbb{M}_0 \times \mathbb{M}_1 \times \ldots \times \mathbb{M}_\nu\}$, where the sets $\mathbb{M}_i, i \in \{0, \ldots, \nu\}$ are defined as in Procedure 2.4 Step 2. ◁

Example 2.13 (Mathematical Pendulum)**.** Examine a rigid pendulum as in Figure 2 of length l and mass m in an environment with gravitational acceleration g. We will be interested in the position (x,y) of the mass point during a given time interval $[t_0, t_f]$. The movement of the mass point can be

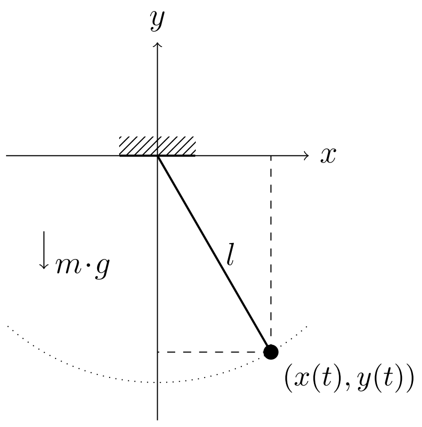


Figure 2: Mathematical Pendulum

modeled with the Euler-Lagrange equations (see e.g. [Cal96]). The Lagrange function is given by

$$\mathcal{L} = T - U - \lambda C,$$

where the kinetic energy $2T = m(\dot{x}^2 + \dot{y}^2)$, the potential energy $U = mgy$ and the constraint is defined by length $C = x^2 + y^2 - l^2 = 0$. With $q = [x, y, \lambda]^T$, where λ is the associated Lagrange multiplier, the Euler-Lagrange equations are given by

$$0 = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{q}}\mathcal{L}(q, \dot{q})\right) - \frac{\partial}{\partial q}\mathcal{L}(q, \dot{q}). \tag{2.6}$$

By introducing order reductions $v = \dot{x}$ and $w = \dot{y}$ we obtain the directional velocities and the system of equations (2.6) can be written in the form of a quasi-linear differential-algebraic equation

$$\begin{aligned} \dot{x} &= v, \\ \dot{y} &= w, \\ m\dot{v} &= -2x\lambda, \\ m\dot{w} &= -2y\lambda - mg, \\ 0 &= x^2 + y^2 - l^2. \end{aligned} \tag{2.7}$$

This system has strangeness-index $\nu_s = 2$ (see e.g. [KM06]). To derive the hidden constraints we can use Procedure 2.4. Differentiation of the constraint yields the additional constraints

$$\begin{aligned} 0 &= 2xv + 2yw, \\ 0 &= 2v^2 + 2w^2 - \frac{4}{m}(x^2 + y^2)\lambda - 2gy. \end{aligned} \tag{2.8}$$

Then, we can write system (2.7) and (2.8) in the overdetermined semi-implicit form as introduced in Definition 2.9

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{w} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} v \\ w \\ \text{-}2x\lambda \\ \text{-}2y\lambda - mg \\ x^2 + y^2 - l^2 \end{bmatrix}, \tag{2.9a}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - l^2 \\ 2xv + 2yw \\ 2v^2 + 2w^2 - \frac{4}{m}(x^2 + y^2)\lambda - 2gy \end{bmatrix}. \tag{2.9b}$$

This system has strangeness-index $\nu_s = 1$, because there are still dependencies in the equations of (2.9a) and (2.9b). $\triangleleft$

3 Half-Explicit Runge-Kutta Methods

Half-explicit Runge-Kutta methods for differential-algebraic equations were proposed in [HLR89]. The advantages of these methods are that we can apply the classical Runge-Kutta approach as known for non-stiff ordinary differential equations. These are fairly easy to implement and give a lot of flexibility regarding convergence and stability investigations. Further, we are going to use one-step Runge-Kutta methods that do not need a starting procedure and, therefore, can rely on a consistent initial value. Additionally, we only need to solve reduced nonlinear systems of equations. We will extend the idea to the numerical treatment of overdetermined semi-implicit differential-algebraic equations. For this, we split the current state x into different parts, which will be treated with an explicit s-stage Runge-Kutta method and a special Newton method, respectively. First, we will give a general definition of the methods. Then, we discuss how the different parts of the current state can be determined automatically. Lastly, we will have a short numerical study to propose convergence of the methods.

3.1 Definition

An introduction to the numerical integration of ordinary systems of differential equations via Runge-Kutta methods can be found in [Pla10] or [SPW12]. The general idea of a Runge-Kutta step within the Runge-Kutta method is to use the derivatives of the system to approximate where the state will evolve to. Unfortunately, for most differential-algebraic equations these derivatives can not be calculated explicitly. That is why we have to generalize the idea of Runge-Kutta methods.

Definition 3.1 (Runge-Kutta methods for differential-algebraic equations)**.** *The discretization of a general differential-algebraic equation (2.1) on* $[t_0, t_f] = \mathbb{I}$*,* $x(t_0) = x_0$ *with an* s-stage Runge-Kutta method *is defined by the* Butcher-tableau

$$\begin{array}{c|ccc}
c_1 & a_{11} & \cdots & a_{1s} \\
\vdots & \vdots & & \vdots \\
c_s & a_{s1} & \cdots & a_{ss} \\
\hline
 & b_1 & \cdots & b_s
\end{array}$$

where $A = [a_{i,j}]_{i,j=1,\dots,s} \in \mathbb{R}^{s,s}$*,* $[b_j]_{j=1,\dots,s} \in \mathbb{R}^s$ *and* $[c_i]_{i=1,\dots,s} \in \mathbb{R}^s$*. The time domain* $\mathbb{I}$ *is discretized with step size* h *such that* $t_0 < t_0 + h = t_1 < t_0 + 2h =$

$t_2 < \ldots < t_f$. Then the approximation to the solution x_1 at $t_1 = t_0 + h$ is calculated by

$$\left.\begin{aligned} \mathcal{X}_i &= x_0 + h \cdot \textstyle\sum_{j=1}^{s} a_{i,j}\, \dot{\mathcal{X}}_j, \qquad &(3.1a)\\ 0 &= F(\mathcal{X}_i, \dot{\mathcal{X}}_i, t_0 + c_i h), &(3.1b) \end{aligned}\right\} \quad \textit{for } i = 1, ..., s$$

$$x_1 = x_0 + h \cdot \textstyle\sum_{i=1}^{s} b_i\, \dot{\mathcal{X}}_i. \qquad (3.1c)$$

The respective approximation for the following time step can be calculated iteratively by using x_1 as the initial value and repeating the steps above.

We will consider Runge-Kutta methods where $a_{i,j} = 0$ for $i \geq j$. These methods are called *explicit Runge-Kutta methods* and allow to explicitly solve (3.1a). We will elaborate on this advantage in the following formulation of Algorithm 3.3. In the general case, one has to solve a nonlinear system of size $sn \times sn$ in (3.1b) to obtain $\dot{\mathcal{X}}_i$. To reduce the numerical costs, we will exploit the structure of overdetermined semi-implicit differential-algebraic equations to solely solve nonlinear systems of size $m \times m$. In contrast to (3.1), we will partition the current state x into the parts x^d and x^a. The vector x^d holds the *differential components* of x that are treated with the Runge-Kutta method. The vector x^a holds the *algebraic components* of x that are derived from the solution of a nonlinear system of equations. At this point, we do not know how to calculate the proposed partition. We will describe how to select components for the different parts depending on the properties of the system in Section 3.2. That is why we will introduce the following operation.

Notation 3.2. Use $\oplus$ as an operation to symbolize the partition of the current state into differential and algebraic components, i.e.

$$\oplus : \mathbb{R}^{n-m} \times \mathbb{R}^{m} \to \mathbb{R}^{n}.$$

For example, the notation $x_0 = x_0^d \oplus x_0^a$ means that the vector x_0 is equal to the combination of the separate vectors x_0^d and x_0^a with respect to permutations and dimensions. ◁

Algorithm 3.3 (Half-explicit Runge-Kutta method)**.** To compute a numerical solution of an overdetermined semi-implicit differential algebraic equation (2.4) with consistent initial value $x_0 = x_0^d \oplus x_0^a$, use the steps

$$\left.\begin{aligned} \mathcal{X}_i^d &= x_0^d + h\cdot \textstyle\sum_{j=1}^{i-1} a_{i,j}\, \dot{\mathcal{X}}_j^d, && (3.2a)\\ 0 &= g(\mathcal{X}_i^d \oplus \mathcal{X}_i^a, t_0 + c_i h), && (3.2b)\\ E(\mathcal{X}_i, t_0 + c_i h)\, \dot{\mathcal{X}}_i &= f(\mathcal{X}_i, t_0 + c_i h), && (3.2c) \end{aligned}\right\} \text{ for } i = 1, ..., s$$

$$x_1^d = x_0^d + h\cdot \textstyle\sum_{j=1}^{s} b_j\, \dot{\mathcal{X}}_j^d, \qquad (3.2d)$$

$$0 = g(x_1^d \oplus x_1^a, t_0 + h). \qquad (3.2e)$$

The values a_{ij}, b_j and c_i are the coefficients of the chosen explicit s-stage Runge-Kutta method. The time domain $\mathbb{I}$ is discretized with step size h such that $t_0 < t_0+h = t_1 < t_0+2h = t_2 < \ldots < t_f$. Then, x_1 is the approximation to the solution of (2.4) at $t_1 = t_0 + h$. $\triangleleft$

Remark 3.4. The general outline of Algorithm 3.3 is to calculate the intermediate differential components $\mathcal{X}_i^d$ with an explicit Runge-Kutta step in (3.2a). Next, we solve the nonlinear system of equations in (3.2b) for the intermediate algebraic components $\mathcal{X}_i^a$. Then, we solve (3.2c) to determine the derivative $\dot{\mathcal{X}}_i$ of the intermediate state $\mathcal{X}_i = \mathcal{X}_i^d \oplus \mathcal{X}_i^a$ and repeat (3.2a), (3.2b) and (3.2c) for s times according to the stages of the given Runge-Kutta method. Lastly, we calculate the differential components x_1^d with the final Runge-Kutta step in (3.2d) and obtain the algebraic components x_1^a by solving (3.2e). The approximation of the solution at $t_1 = t_0 + h$ is then given with $x_1 = x_1^d \oplus x_1^a$.

Starting with a consistent initial value $x_0 = x_0^d \oplus x_0^a$, we can describe the iteration in more detail as follows:

$i = 1:$ $\mathcal{X}_1^d = x_0^d$ in (3.2a). With (3.2b), $\mathcal{X}_1^a = x_0^a$, since $\mathcal{X}_1^d \oplus \mathcal{X}_1^a = x_0$ is a consistent value of system (2.4). Then, we solve (3.2c) for $\dot{\mathcal{X}}_1 = \dot{\mathcal{X}}_1^d \oplus \dot{\mathcal{X}}_1^a$ with $\mathcal{X}_1 = \mathcal{X}_1^d \oplus \mathcal{X}_1^a$.

$i = 2:$ We solve (3.2a) explicitly for $\mathcal{X}_2^d$ with $\dot{\mathcal{X}}_1^d$. If we insert $\mathcal{X}_2^d$ into (3.2b), we obtain a nonlinear system of equations for $\mathcal{X}_2^a$. Then, we solve (3.2c) for $\dot{\mathcal{X}}_2^d = \dot{\mathcal{X}}_2^d \oplus \dot{\mathcal{X}}_2^a$ with $\mathcal{X}_2 = \mathcal{X}_2^d \oplus \mathcal{X}_2^a$.

$\vdots$

$i = s:$ We solve (3.2a) explicitly for $\mathcal{X}_s^d$ with $\dot{\mathcal{X}}_1^d, ..., \dot{\mathcal{X}}_{s-1}^d$. If we insert $\mathcal{X}_s^d$ into (3.2b), we obtain a nonlinear system of equations for $\mathcal{X}_s^a$. Then, we solve (3.2c) for $\dot{\mathcal{X}}_s^d = \dot{\mathcal{X}}_s^d \oplus \dot{\mathcal{X}}_s^a$ with $\mathcal{X}_s = \mathcal{X}_s^d \oplus \mathcal{X}_s^a$.

Finally, we calculate x_1^d by solving (3.2d) explicitly with $\dot{\mathcal{X}}_1^d, ..., \dot{\mathcal{X}}_s^d$ and solve (3.2e) with x_1^d for x_1^a. Then $x_1 = x_1^d \oplus x_1^a$.

Here, we see that we can derive $\mathcal{X}_i^d$ explicitly in (3.2a) for each step. We only have to solve the reduced $m \times m$ nonlinear system of equations (3.2b) for $\mathcal{X}_i^a$. The same holds for x_1^d and x_1^a in (3.2d) and (3.2e), respectively. That is why this method is classified as a half-explicit Runge-Kutta method. $\triangleleft$

Remark 3.5. As a short note on the given names, one can now see that the differential components are always carried on with the derivatives of the current state from (2.4a) within the Runge-Kutta step and that the algebraic components are resolved within an algebraic method solving the nonlinear system of equations derived from (2.4b). $\triangleleft$

3.2 Selection Procedure

So far, we made the assumption that the algebraic and differential components are known for each step. In this section we will examine how to find these components with *selectors*. First, we will give a general criterion to support the investigation. Following, we will make an approach to determine the selectors by decomposing the Jacobian of the constraints $0 = g(x,t)$. Then, we will study examples to show the difficulties when working without prior knowledge of the selectors. These will lead to the introduction of a special pivoting technique within the decomposition of the Jacobian which depends on the system of equations $E(x,t)\,\dot{x} = f(x,t)$. Also, we will illustrate how the selection of differential and algebraic components can change within the given time domain. Finally, we will take a closer look on how we have to restrict the set of overdetermined semi-implicit differential-algebraic equations to make an automatic choice of selectors possible.

Criterion 3.6. *Given an overdetermined semi-implicit differential-algebraic system (2.4) and* $\mathcal{J}_g(x,t) := \partial g(x,t)/\partial x$. *Find* component selectors $\varphi^d : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^{n-m}$ *and* $\varphi^a : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^m$ *and* column selectors $\Phi^d : \mathbb{R}^{n,n} \times \mathbb{I} \to \mathbb{R}^{n,n-m}$ *and* $\Phi^a : \mathbb{R}^{m,n} \times \mathbb{I} \to \mathbb{R}^{m,m}$ *such that*

1. $\Phi^d(E(x,t),t)$ *has rank* $n-m$,
2. $E(x,t)\,(\varphi^d(\dot{x},t) \oplus \varphi^a(\dot{x},t)) = f(x,t)$ *has a unique solution,*
3. $\mathcal{J}_g(x,t)$ *has rank* m, $\Phi^a(\mathcal{J}_g(x,t),t)$ *is regular and*
4. $\varphi^d(x,t) \oplus \varphi^a(x,t) = x$,

for all $(x,t) \in \mathbb{D}_x \times \mathbb{I}$.

The Jacobian of the hidden constraints (2.4b) is an (underdetermined) nonlinear system of equations of size $n \times m$, where $m \leq n$. In our investigation, we will consider the case where $m < n$, because otherwise Algorithm 3.3 reduces to solely solving (3.2e). With the following approach we will describe how the selection of the components derives from the decomposition of the Jacobian of the constraints.

Approach 3.7. To meet Criterion 3.6 we would need to select components such that the corresponding selected remainder matrix of $\mathcal{J}_g$ is regular. A first approach is to take a closer look at the Newton iteration which solves the nonlinear system of equations in (3.2b) and (3.2e). For this, we fix t, initialize with $\widehat{\mathcal{X}}_0 = \mathcal{X}_i$ and omit index i to iterate

$$\left.\begin{aligned} \mathcal{J}_g(\widehat{\mathcal{X}}_k, t)\,\Delta\widehat{\mathcal{X}}_k &= -g(\widehat{\mathcal{X}}_k, t), \\ \widehat{\mathcal{X}}_{k+1} &= \widehat{\mathcal{X}}_k + \Delta\widehat{\mathcal{X}}_k, \end{aligned}\right\} \text{ for } k = 0, 1, \dots . \qquad \begin{aligned}(3.3\text{a})\\(3.3\text{b})\end{aligned}$$

In the case $m < n$, (3.3a) is an underdetermined system of equations. We will use a LU decomposition with full pivoting (see e.g. [GL13]) of $\mathcal{J}_g(\widehat{\mathcal{X}}_k, t)$ such that

$$\underbrace{L^{-1}P\,\mathcal{J}_g(\widehat{\mathcal{X}}_k, t)\,Q}_{U}\,Q^T\Delta\widehat{\mathcal{X}}_k = -L^{-1}P\,g(\widehat{\mathcal{X}}_k, t), \tag{3.4}$$

where $L \in \mathbb{R}^{m,m}$ is a lower-triangular matrix, $P \in \mathbb{R}^{m,m}$ and $Q \in \mathbb{R}^{n,n}$ are permutation matrices and $U \in \mathbb{R}^{m,n}$. Then, by splitting $U = [\,\mathcal{R}\ \mathcal{S}\,]$, where $\mathcal{R} \in \mathbb{R}^{m,m}$ is a upper-triangular and regular matrix and $\mathcal{S} \in \mathbb{R}^{m,n-m}$, and $Q = [\,Q_1\,Q_2\,]$, where $Q_1 \in \mathbb{R}^{n,m}$ and $Q_2 \in \mathbb{R}^{n,n-m}$, we get

$$[\,\mathcal{R}\ \mathcal{S}\,]\begin{bmatrix} Q_1^T\Delta\widehat{\mathcal{X}}_n \\ Q_2^T\Delta\widehat{\mathcal{X}}_n \end{bmatrix} = -L^{-1}P\,g(\widehat{\mathcal{X}}_n, t). \tag{3.5}$$

Here, we propose that it is sufficient to solely iterate the components of $\widehat{\mathcal{X}}_n$ corresponding to Q_1^T in (3.5) to make (3.3) converge. In other words, we obtain the *algebraic selector* and *differential selector*, i.e.

$$\begin{aligned} \varphi^a(\mathcal{X}, t) &= Q_1^T\mathcal{X} = \mathcal{X}^a, \\ \varphi^d(\mathcal{X}, t) &= Q_2^T\mathcal{X} = \mathcal{X}^d, \end{aligned}$$

and the respective column selectors, i.e.

$$\begin{aligned} \Phi^a(\mathcal{J}_g(\mathcal{X}, t), t) &= \mathcal{J}_g(\mathcal{X}, t)\,Q_1 = \mathcal{J}_g^a(\mathcal{X}, t), \\ \Phi^d(E(\mathcal{X}, t), t) &= E(\mathcal{X}, t)\,Q_2 = E^d(\mathcal{X}, t). \end{aligned}$$

Unfortunately, this selection procedure neglects some of the assumptions we made in Criterion 3.6. It is possible to find examples (see the continuation of Example 2.13 below), where $\mathcal{R}$ in (3.5) is a regular matrix, but $\Phi^d(E(x,t),t)$ has a lower rank than $n-m$. $\triangleleft$

Definition 3.8. *A nonlinear mapping $E(x,t) \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n,n})$ is called* regularly reducible *if*

$$\tilde{P}\, E(x,t)\, \tilde{Q} = \begin{bmatrix} \bar{E}(x,t) & 0 \\ 0 & 0 \end{bmatrix},$$

where $\bar{E}(x,t) \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{l,l})$ is point-wise nonsingular for all $(x,t) \in \mathbb{D}^n \times \mathbb{I}$ and $\tilde{P}, \tilde{Q} \in \{0,1\}^{n,n}$ are permutation matrices. Additionally, let

$$\tilde{Q} =: [\, \tilde{Q}_2\, \tilde{Q}_1 \,], \tag{3.6}$$

such that $\tilde{Q}_2 \in \{0,1\}^{n,l}$ and $\tilde{Q}_1 \in \{0,1\}^{n,n-l}$.

Definition 3.9. *Let $E(x,t)$ be regularly reducible, set $r = n-l$ and define $\varphi^{a_r} : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^r$ by*

$$\varphi^{a_r}(\mathcal{X}, t) = \tilde{Q}_1^T \mathcal{X} = \mathcal{X}^{a_r}.$$

The components selected by φ^{a_r} are called required algebraic components *of $\mathcal{X}$. The other algebraic components selected by φ^a are called* variable algebraic components *of $\mathcal{X}$. Further, the differential components selected by φ^d can be considered as* variable differential components *of $\mathcal{X}$.*

Example 2.13 (Continued)**.** Consider system (2.9) with mass $m = 1$, length $l = 1$, gravitational acceleration $g = 11.16$ and the consistent starting value

$$x_0 = [\, 0.5\,,\, \text{-}(\tfrac{3}{4})^{\frac{1}{2}}\,,\, \text{-}0.5\,,\, \text{-}(\tfrac{1}{9})^{\frac{1}{2}}\,,\, 5\,]^T.$$

Then,

$$\mathcal{J}_g(x_0, t_0) \approx \begin{bmatrix} 1 & \text{-}3^{\frac{1}{2}} & 0 & 0 & 0 \\ \text{-}1 & \text{-}(\frac{1}{3})^{\frac{1}{2}} & 1 & \text{-}3^{\frac{1}{2}} & 0 \\ \text{-}20 & 12.3 & \text{-}2 & \text{-}2(\frac{1}{3})^{\frac{1}{2}} & \text{-}4 \end{bmatrix}.$$

The LU decomposition with complete pivoting of $\mathcal{J}_g(x_0, t_0)$ yields

$$U \approx \left[\begin{array}{ccc|cc} \text{-}20 & -1.2 & 12.3 & -2 & -4 \\ 0 & -1.7 & -1.2 & 1.1 & 0.2 \\ 0 & 0 & -1.1 & -0.1 & -0.2 \end{array}\right]$$
$$\underbrace{\hphantom{-20\quad -1.2\quad 12.3}}_{\mathcal{R}\text{ regular}} \quad \underbrace{\hphantom{-2\quad -4}}_{\mathcal{S}}$$

and

$$Q = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \underbrace{0 \quad 0 \quad 0}_{Q_1} & & & \underbrace{0 \quad 1}_{Q_2} & \end{array}\right].$$

Thus, we obtain the symbolized selections

$$\varphi^a([\begin{array}{ccccc} x & y & v & w & \lambda \end{array}]^T, t) = Q_1^T \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} x \\ w \\ y \end{bmatrix}$$

and

$$\varphi^d([\begin{array}{ccccc} x & y & v & w & \lambda \end{array}]^T, t) = Q_2^T \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} v \\ \lambda \end{bmatrix}.$$

Here, λ is not selected as an algebraic component though it is a required algebraic component. Hence, we have to find another way to determine the selectors. ◁

Notation 3.10. In the following, we will use colon notation for matrices. Let $A = [a_{p,q}]_{\substack{p=1,\dots,m \\ q=1,\dots,n}} \in \mathbb{R}^{m,n}$, then

$$A(i:j, k:l) := \begin{bmatrix} a_{i,k} & a_{i,k+1} & \dots & a_{i,l} \\ a_{i+1,k} & a_{i+1,k+1} & \dots & a_{i+1,l} \\ \vdots & \ddots & \ddots & \vdots \\ a_{j,k} & \dots & a_{j,l-1} & a_{j,l} \end{bmatrix},$$

where $1 \leq i \leq j \leq m$ and $1 \leq k \leq l \leq n$. ◁

To always include the required algebraic components, we introduce a special pivoting technique within the LU decomposition. The following procedure will simply select the first r pivots within the set of columns corresponding to the required algebraic components $\mathcal{J}_g\tilde{Q}_1$ and other than that will proceed similarly to a classical LU decomposition with full pivoting.

Procedure 3.11. Let $\mathcal{J} \in \mathbb{R}^{m,n}$ with $m \leq n$, r the number of required algebraic components and $\hat{\mathcal{J}} := \mathcal{J}\tilde{Q}_1$, where $\tilde{Q}_1$ is defined as in (3.6). Initialize the matrices $L \in \mathbb{R}^{m,m}$ and $U \in \mathbb{R}^{m,n}$ as $L = I$ and $U = 0$. For $i = 1, \ldots, r$:

1. Find the largest entry $|\,\hat{\mathcal{J}}^{(i)}_{j,\hat{k}}\,|$ in the columns $\hat{\mathcal{J}}(i:m, i:r)$ corresponding to the required algebraic components.
2. Swap rows j and i of $\mathcal{J}$ and L^0, and columns $k = \hat{k}_{\tilde{Q}_1}$ and i of $\mathcal{J}$ and U, where $\hat{k}_{\tilde{Q}_1}$ denotes the index of the pivotal column of $\hat{\mathcal{J}}$ embedded within $\mathcal{J}$.
3. Compute column i of L with entries $L_{j,i} = \mathcal{J}_{j,i}\,/\,\mathcal{J}_{j,j}$ for $j = i{+}1, \ldots, m$.
4. Compute row i of U with entries $U_{i,j} = \mathcal{J}_{i,j}$ for $j = i, \ldots, n$.
5. Update $\mathcal{J}$ with entries $\mathcal{J}_{j,k} = \mathcal{J}_{j,k} - L_{j,i}\,U_{i,k}$ for $j = i+1, \ldots, m$ and $k = i+1, \ldots, n$.

For $i = r+1, \ldots, m$:

6. Find the largest entry $|\,\mathcal{J}^{(i)}_{j,k}\,|$ in the rest matrix $\mathcal{J}(i:m, i:n)$.
7. Repeat Step 2, 3, 4 and 5.

Lastly, if $n > m$ perform the final column swap:

8. Find the largest entry $|\,\mathcal{J}^{(m+1)}_{j,k}\,|$ in $\mathcal{J}(m:m, m:n)$.
9. Swap columns k and m of $\mathcal{J}$ and U.

Then, the iteration yields a decomposition of $\mathcal{J}$ such that

$$L\,U = P\,\mathcal{J}\,Q,$$

where L is a lower-triangular matrix, $U(1:m, 1:m)$ is a upper-triangular matrix and $P \in \{0,1\}^{m,m}$ and $Q \in \{0,1\}^{n,n}$ are the permutation matrices obtained by the row and column swaps, respectively. $\triangleleft$

Remark 3.12. By using the LU decomposition introduced in Procedure 3.11 for (3.4), we achieve that all required algebraic components and their corresponding columns in $\mathcal{J}_g$ are selected with Q_1 in (3.5). $\triangleleft$

Example 3.13. Consider the differential-algebraic system

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + \frac{1}{2}x_2 + x_3 \end{bmatrix},$$

with the starting value $x_0 = [\,x_1^0, x_2^0, x_3^0\,]^T = [\,1, 2, 2\,]^T$. This system has a solution and

$$\mathcal{J}_g(x_0, t_0) = \begin{bmatrix} 2 & 1 & 0 \\ 1 & \frac{1}{2} & 1 \end{bmatrix}.$$

Obviously, $\dot{x}_3$ is a free component in the solution of $E(x,t)\,\dot{x} = f(x,t)$. Thus, x_3 is required to be treated algebraically. With the decomposition presented in Procedure 3.11 we obtain

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & \frac{1}{2} \\ 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathcal{J}_g(x_0, t_0) \left[\begin{array}{cc|c} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array}\right].$$

Hence, x_1 is the variable algebraic component. The remaining component x_2 will be treated as a variable differential component. ◁

Example 2.13 (Continued). Applying the decomposition presented in Procedure 3.11 to $\mathcal{J}_g$ of the mathematical pendulum at the given starting value yields

$$Q = \left[\underbrace{\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array}}_{Q_1} \middle| \underbrace{\begin{array}{cc} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{array}}_{Q_2}\right],$$

and thus the symbolized selection

$$\varphi^a([\ x \quad y \quad v \quad w \quad \lambda\ ]^T, t) = Q_1^T \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} \lambda \\ w \\ y \end{bmatrix}$$

and

$$\varphi^d([\begin{array}{ccccc} x & y & v & w & \lambda \end{array}]^T, t) = Q_2^T \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} x \\ v \end{bmatrix}.$$

Here, λ is a required algebraic component, w, y are variable algebraic components and x, v are variable differential components. $\triangleleft$

As mentioned before the selection of algebraic and differential components can change during the given time interval. In general, it is required to change the selection once the corresponding Jacobian is close to singular. In this situation the Newton-method to solve the nonlinear system of equations is not or only slowly converging. If we take the perspective of the LU decomposition one can easily see that the selection changes once the pivots change.

Example 2.13 (Continued)**.** We will start with a different initial value which represents the mass of the pendulum passing by the equilibirum position with a given velocity. Set

$$x_0 = [\,0\,,\,\text{-}1\,,\,\text{-}3.5^{\frac{1}{2}}\,,\,0\,,\,7.33\,]^T,$$

which should be roughly on the trajectory resulting from the prior initial value. Applying the decomposition presented in Procedure 3.11 to $\mathcal{J}_g$ at the new starting value yields

$$Q = \left[\begin{array}{ccc|cc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ \underbrace{1 \quad 0 \quad 0}_{Q_1} & & & \underbrace{0 \quad 0}_{Q_2} & \end{array}\right],$$

and thus the symbolized selection

$$\varphi^a([\begin{array}{ccccc} x & y & v & w & \lambda \end{array}]^T, t) = Q_1^T \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} \lambda \\ x \\ y \end{bmatrix}$$

and

$$\varphi^d([\begin{array}{ccccc} x & y & v & w & \lambda \end{array}]^{\mathsf{T}}, t) = Q_2^{\mathsf{T}} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ v \\ w \\ \lambda \end{bmatrix} = \begin{bmatrix} w \\ v \end{bmatrix}.$$

Here, we see that the selection has changed because of the different state. Still, λ is a required algebraic component, but now x, y are variable algebraic components and v, w are variable differential components. This selection might be surprising because usually at least one positional and one dynamical component should be selected. But, since the selection procedure relies on the pivots in the Jacobian and forces certain components to be selected, such a selection is possible. $\triangleleft$

We can summarize that the discussed algorithmic selection procedure can be used for overdetermined semi-implicit differential algebraic equations as in Definition 2.9, where the leading matrix $E(x, t)$ is regularly reducible and the system satisfies Criterion 3.6.

3.3 Numerical Study

In the numerical study we are going to investigate the progression of the algebraic and differential components within Algorithm 3.3. First, we will take a closer look at the properties of the Newton method used. The algebraic components of the solution depend only on solving $0 = g(x^d \oplus x^a, t)$ in step (3.2b) and (3.2e), where x^d are the progressed differential components of step (3.2a) and (3.2d), respectively. Since, generally, for Algorithm 3.3 the Jacobian $\mathcal{J}_g$ is not square we have to make slight modifications to the Newton method.

Definition 3.14 (Divided Newton method)**.** *For a given function $g : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^m$, $m \leq n$ and an initial guess $x_0 = x_0^d \oplus x_0^a$ the iteration*

$$\left.\begin{aligned} \begin{bmatrix} 0 & I \\ \mathcal{R}(x_k) & \mathcal{S}(x_k) \end{bmatrix} \begin{bmatrix} \Delta\, x_k^a \\ \Delta\, x_k^d \end{bmatrix} &= \begin{bmatrix} 0 \\ -g(x_k, t) \end{bmatrix}, \\ \begin{bmatrix} x_{k+1}^a \\ x_{k+1}^d \end{bmatrix} &= \begin{bmatrix} x_k^a \\ x_k^d \end{bmatrix} + \begin{bmatrix} \Delta\, x_k^a \\ \Delta\, x_k^d \end{bmatrix}, \\ x_{k+1} &= x_{k+1}^d \oplus x_{k+1}^a, \end{aligned}\right\} \quad k = 0, 1, \dots \qquad \begin{aligned} &(3.7a) \\ &(3.7b) \\ &(3.7c) \end{aligned}$$

where $\mathcal{R}(x_k) \in \mathbb{R}^{m,m}$, $\mathcal{S}(x_k) \in \mathbb{R}^{m,n-m}$, for all $k \in \mathbb{N}_0$, as introduced in Approach 3.7 *and identity matrix $I \in \{0,1\}^{n-m,n-m}$, is called* divided Newton method.

Remark 3.15. When rearranging (3.7a) to

$$\begin{bmatrix} \Delta\, x_k^a \\ \Delta\, x_k^d \end{bmatrix} = \begin{bmatrix} -\mathcal{R}(x_k)^{-1}\mathcal{S}(x_k) & \mathcal{R}(x_k)^{-1} \\ I & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -g(x_k,t) \end{bmatrix},$$

we get

$$\Delta\, x_k^a \;=\; -\mathcal{R}(x_k)^{-1} g(x_k,t) \quad \text{and} \quad \Delta\, x_k^d \;=\; 0.$$

Thus, with this technique the algebraic components x_k^a are iterated as in a classical Newton method and the differential components x_k^d remain untouched. This coalesces with the approach to modify only the algebraic components to meet the constraint $0 = g(x_d \oplus x_a, t)$. ◁

We will use a modified version of the affine covariant Newton-Mysovskikh theorem as discussed in [Deu04] to prove convergence of the divided Newton method.

Theorem 3.16 (Newton-Mysovskikh). *Let $g : \mathcal{M} \to \mathbb{R}^m$ be a continuously differentiable mapping with $\mathcal{M} \subset \mathbb{R}^n$ open, convex and $m \leq n$. Derive $\mathcal{R}(x)$ and $\mathcal{S}(x)$ from $\mathcal{J}_g(x)$ as described in Approach* 3.7*, where $\mathcal{R}(x) \in \mathbb{R}^{m,m}$, $\mathcal{S}(x) \in \mathbb{R}^{m,n-m}$ and suppose that $\mathcal{R}(x)$ is invertible for each $x \in \mathcal{M}$. Assume the following affine covariant Lipschitz condition holds:*

$$\|\, \mathcal{R}(y)^{-1}(\mathcal{R}(y) - \mathcal{R}(x))(y - x) \,\| \leq L \,\|\, y - x \,\|^2$$

for $x, y \in \mathcal{M}$. For the initial guess $x_0 = x_0^a \oplus x_0^d$ assume that

$$\varepsilon_0 := L\, \|\Delta x_0^a\| < 2,$$

$$\rho = \frac{\|\Delta x_0\|}{1 - \frac{1}{2}\varepsilon_0}, \quad \overline{S}(x_0,\rho) \subset \mathcal{M},$$

where $\overline{S}(x_0,\rho) \subset \mathcal{M}$ denotes the closure of the open ball $S(x_0,\rho)$ of radius ρ around x_0. Then the sequence $\{x_k\}$ of divided Newton iterates remains in $S(x_0,\rho)$, is a Cauchy sequence and converges to a solution $x_* = x_*^a \oplus x_*^d \in \overline{S}(x_0,\rho)$, where $g(x_*) = 0$.

Moreover, the following error estimates hold

$$\|\, x_{k+1}^a - x_k^a \,\| \;\leq\; \tfrac{1}{2}\, L \,\|\, x_k^a - x_{k-1}^a \,\|^2,$$

$$\|\, x_k^a - x_*^a \,\| \;\leq\; \frac{\|\, x_k^a - x_{k+1}^a \,\|}{1 - \frac{1}{2} L \,\|\, x_k^a - x_{k+1}^a \,\|}\,.$$

Proof. We will follow the proof in [Deu04], but make the necessary modifications to suit it to the divided Newton method. First, the iteration is used for k and $k-1$, where $s\Delta_{k-1} := [\,\Delta\, x^a_{k-1}\,\Delta\, x^d_{k-1}\,]^T$:

$$\left\|\begin{bmatrix}\Delta\, x^a_k \\ \Delta\, x^d_k\end{bmatrix}\right\| = \left\|\begin{bmatrix}-\mathcal{R}(x_k)^{-1}\mathcal{S}(x_k) & \mathcal{R}(x_k)^{-1} \\ I & 0\end{bmatrix}\begin{bmatrix}0 \\ g(x_k)\end{bmatrix}\right\|$$

$$= \left\|\begin{bmatrix}* & \mathcal{R}(x_k)^{-1} \\ I & 0\end{bmatrix}\left(\begin{bmatrix}0 \\ g(x_k)\end{bmatrix} - \begin{bmatrix}0 \\ g(x_{k-1})\end{bmatrix} - \begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}) & \mathcal{S}(x_{k-1})\end{bmatrix}\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix}\right)\right\|$$

$$= \left\|\begin{bmatrix}* & \mathcal{R}(x_k)^{-1} \\ I & 0\end{bmatrix}\int_0^1\left(\begin{bmatrix}0 \\ \mathcal{J}_g(x_{k-1}+s\Delta_{k-1})\end{bmatrix} - \begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}) & \mathcal{S}(x_{k-1})\end{bmatrix}\right)\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix} ds\right\|.$$

Further, since $\Delta x^d_k = 0$ for all $k \in \mathbb{N}_0$:

$$= \left\|\begin{bmatrix}* & \mathcal{R}(x_k)^{-1} \\ I & 0\end{bmatrix}\int_0^1\left(\begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}+s\Delta_{k-1}) & \mathcal{S}(x_{k-1}+s\Delta_{k-1})\end{bmatrix} - \begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}) & \mathcal{S}(x_{k-1})\end{bmatrix}\right)\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix} ds\right\|$$

$$= \int_0^1 \left\|\begin{bmatrix}* & \mathcal{R}(x_k)^{-1} \\ I & 0\end{bmatrix}\left(\begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}+s\Delta_{k-1}) & \mathcal{S}(x_{k-1}+s\Delta_{k-1})\end{bmatrix} - \begin{bmatrix}0 & I \\ \mathcal{R}(x_{k-1}) & \mathcal{S}(x_{k-1})\end{bmatrix}\right)\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix}\right\| ds$$

$$\leq \int_0^1 L\,\left\|\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix}\right\|^2 s\,ds \;=\; \frac{1}{2}\,L\,\left\|\begin{bmatrix}\Delta\, x^a_{k-1} \\ \Delta\, x^d_{k-1}\end{bmatrix}\right\|^2,$$

The rest of the proof follows directly from the proof in [Deu04]. □

The next observation we make is that the intermediate system within Algorithm 3.3 that results from the regularization and selection yields a strangeness-free system.

Lemma 3.17. *Consider the selector $\varphi^d : \mathbb{R}^n \times \mathbb{I} \to \mathbb{R}^{n-m}$ and the selected semi-implicit differential algebraic equation*

$$\begin{aligned} 0 &= \varphi^d(E(x,t)\dot{x} - f(x,t), t), \\ 0 &= g(x,t), \end{aligned} \tag{3.8}$$

where $E \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n,n})$, $f \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^n)$, $x \in \mathbb{D}_x$, $g \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^m)$ and $\mathbb{D}_x \subset \mathbb{R}^n$. Assume that

$$\begin{aligned} m &= \operatorname{rank}(\mathcal{J}_g(x,t)) = \mathit{const}, \\ n - m \leq &\operatorname{rank}(E(x,t)) = \mathit{const}, \end{aligned}$$

for all $(x,t) \in \mathbb{D}_x \times \mathbb{I}$ such that $g(x,t) = 0$. Then, with respect to Criterion 3.6*, system (3.8) is strangeness-free.*

Proof. With the properties set in Criterion 3.6, system (3.8) can also be formulated as

$$\begin{aligned} \varphi^d(E(x,t)\,\dot{x}, t) &= \varphi^d(f(x,t), t), \\ 0 &= g(x,t), \end{aligned}$$

or shorter, where the following is true even if the selector changes during the given time interval,

$$\begin{aligned} E^{\bar{d}}(x,t)\,\dot{x} &= f^d(x,t), \\ 0 &= g(x,t), \end{aligned}$$

where $E^{\bar{d}} \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n-m,n})$ is defined such that $E^{\bar{d}}(x,t)\,\dot{x} = \varphi^d(E(x,t)\,\dot{x}, t)$, $\varphi^d(f(x,t),t) =: f^d \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^{n-m})$, $g \in \mathcal{C}(\mathbb{D}_x \times \mathbb{I}, \mathbb{R}^m)$ and

$$\begin{aligned} m &= \operatorname{rank}(\mathcal{J}_g(x,t)) = \mathit{const}, \\ n - m &= \operatorname{rank}(E^{\bar{d}}(x,t)) = \mathit{const}. \end{aligned}$$

The construction of the selectors in Procedure 3.11 guarantees that

$$\operatorname{rank}\left(\begin{bmatrix} E^{\bar{d}}(x,t) \\ \mathcal{J}_g(x,t) \end{bmatrix}\right) = \operatorname{rank}(E^{\bar{d}}(x,t)) + \operatorname{rank}(\mathcal{J}_g(x,t)) = n.$$

With the given prerequisites we can refer to the proof of Lemma 3.5.1 in [Ste06]. □

For the strangeness-free setting we can make use of the numerical analysis of half-explicit Runge-Kutta methods for general differential-algebraic equations in [LM13]. We will introduce the following terms to classify Runge-Kutta methods.

Definition 3.18. *Let x_1 be the approximation of one integration step of Algorithm* 3.3 *with consistent start value $x_0 = x(t)$. Then*

$$\delta x_h(t) = x_1 - x(t+h)$$

is called local discretization error *of the one-step method at time t, where $x(t+h)$ is the exact solution of the given differential-algebraic equation. Further, if the local discretization error satisfies*

$$\delta x_h(t) = \mathcal{O}(h^{p+1}),$$

then the method is called convergent of order p, *if*

$$x_n - x(t+nh) = \mathcal{O}(h^p),$$

where x_n is the approximation after n steps and $x(t+nh)$ is the exact solution at time $t+nh$.

In [LM13] one can find consistency properties of different methods and proof of convergence for half-explicit Runge-Kutta methods of convergence order $p = 2$. The concept of the proof can be generalized for higher order methods. A minor difference in the employed method can be found in how the derivatives of the stage variables within the Runge-Kutta method are determined. In (3.2c) we solve

$$E(\mathcal{X}_i\,, t_0 + c_i h)\,\dot{\mathcal{X}_i} \;= f(\mathcal{X}_i, t_0 + c_i h)$$

for $\dot{\mathcal{X}_i}$ at the time point $t + c_i h$, rather than approximating the derivative, as e.g.

$$\dot{\mathcal{X}_i} = \frac{\mathcal{X}_i\, -\, x_0}{c_i h}.$$

We expect the approximation at the exact time point to be more precise than the approximation by a secant. Further notes about the advantages and disadvantages of this approach can be found in Section 5.

Remark 3.19. One could expect that only half-explicit Runge-Kutta schemes are applicable as presented in [HLR89] or [BH93]. In these we get additional order conditions, because the algebraic treatment introduces errors when proceeding with the differential components. In Algorithm 3.3 the solution to (3.2a) and (3.2d) only uses the derivatives of the differential components in the Runge-Kutta step. Technically, the differential components are completely isolated from the algebraic components, because of the requirements in Criterion 3.6 and the construction of Procedure 3.11. That is why we do not have to expect any additional order conditions and presumably can use all classical Runge-Kutta methods. ◁

4 Implementation and Examples

In this section we are going to give a general introduction to the implementation of Algorithm 3.3 and investigate some explanatory systems of overdetermined semi-implicit differential-algebraic equations.

4.1 Implementation

A first implementation was done in Matlab. The code is distributed at

`https://github.com/simonbaese/HERKosiDAE`

and is available under the MIT license. In the following, we will give a brief overview of the functionality and discuss some adjustments and tricks that were used to boost the performance of the program. For more details and information about the employed subroutines we refer to appendix **??** or the manual distributed with the program.

4.1.1 Input Values

For convenience, the gray boxes will mark the inputs of Algorithm 3.3. The easiest way to set up a new problem is to use the mask available in the folder `'../Examples/Mask for new Problem/'`, which contains the files

`func_E.m`	$\widehat{=}$	defines $E(x,t)$,
`func_f.m`	$\widehat{=}$	defines $f(x,t)$,
`func_g.m`	$\widehat{=}$	defines $g(x,t)$,
`func_J.m`	$\widehat{=}$	optionally defines Jacobian of $g(x,t)$,
`run.m`	$\widehat{=}$	defines variables, settings and executes algorithm.

The definition of the model functions `E`, `f`, `g` and `J` is straight-forward. We use the additional input parameter `var` to pass constants from the executing file to the functions. For the execution of the main routine `HERKosiDAE` in `run.m` we will need the following input parameters.

`delta`	$\widehat{=}$	Differentiation limit for numerical differentiation
`tol`	$\widehat{=}$	Tolerance for Newton iteration
`ptol`	$\widehat{=}$	Tolerance for pivots in Procedure 2.4
`Ab,c,s,p`	$\widehat{=}$	Definition and parameters of Runge-Kutta method
`ssc`	$\widehat{=}$	Boolean option 1 Adaptive step size control 0 Constant step size
`eps0`	$\widehat{=}$	Desired accuracy for adaptive step size control
`beta`	$\widehat{=}$	Safety factor for adaptive step size control
`Jopt`	$\widehat{=}$	Boolean option 1 Jacobian of $g(x,t)$ analytically defined in `func_J.m` 0 Use numerical differentiation of $g(x,t)$
`Estat`	$\widehat{=}$	Boolean option 1 Leading matrix $E(x,t)$ is invariant 0 Leading matrix $E(x,t)$ is not invariant
`Nopt`	$\widehat{=}$	Boolean option 1 Use simplified Newton method 0 Use classical Newton method
`func`	$\widehat{=}$	String of employed functions
`var`	$\widehat{=}$	Container for constants
`x0`	$\widehat{=}$	Initial value
`t0,tf`	$\widehat{=}$	Start and finish time
`h0`	$\widehat{=}$	Initial step size

4.1.2 Runge-Kutta Methods

In the program one can employ the definition and parameters of different Runge-Kutta methods with the subroutine `getRKmethod`. For the investigation of the examples we are going to use the following Runge-Kutta methods with stages s and convergence order p.

(a) Forward Euler method, $s = 1$, $p = 1$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

(b) Heun's method, $s = 2$, $p = 2$

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(c) Kutta's third-order method, $s = 3$, $p = 3$

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

(d) Classical fourth-order method, $s = 4$, $p = 4$

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

(e) Brasey-Hairer 5-Stage HEM4 method [BH93], $s = 5$, $p = 4$

$$
\begin{array}{c|ccccc}
0 & & & & & \\
\frac{3}{10} & \frac{3}{10} & & & & \\
\frac{4-\sqrt{6}}{10} & \frac{1+\sqrt{6}}{30} & \frac{11-4\sqrt{6}}{30} & & & \\
\frac{4+\sqrt{6}}{10} & \frac{-79-31\sqrt{6}}{150} & \frac{-1-4\sqrt{6}}{30} & \frac{24+11\sqrt{6}}{25} & & \\
1 & \frac{14+5\sqrt{6}}{6} & \frac{-8+7\sqrt{6}}{6} & \frac{-9-7\sqrt{6}}{4} & \frac{9-\sqrt{6}}{4} & \\
\hline
 & 0 & 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9}
\end{array}
$$

4.1.3 Selectors

The algebraic and differential selectors are determined in the subroutine `selectors`. The input vector `FV` contains the information of the required algebraic components. It is used when decomposing the Jacobian of $g(x,t)$ in the subroutine `lusp` (Procedure 3.11). The outputs `sa`, `SA` and `SD`, where `sa` is the algebraic selector φ^a as defined before and

$$
\begin{aligned}
\texttt{SA} &= (\varphi^a)^T \varphi^a, \\
\texttt{SD} &= (\varphi^d)^T \varphi^d,
\end{aligned}
$$

are sparse matrices. Since `SA` and `SD` have a density of n^{-1} (quotient of non-zero and zero elements) one can expect a performance gain for larger dimensions n when using sparse matrices [Alt14]. We use these inflated selectors to remember the position of the selected components. Consider the easy operations with given $\mathcal{X}_i$

$$\mathcal{X}_i^a = \mathtt{SA} \cdot \mathcal{X}_i,$$
$$\mathcal{X}_i^d = \mathtt{SD} \cdot \mathcal{X}_i.$$

Now, $\mathcal{X}_i^a$ and $\mathcal{X}_i^d$ contain additional zero elements. Conveniently, the desired components are at the correct position and one can easily obtain

$$\mathcal{X}_i = \mathcal{X}_i^a + \mathcal{X}_i^d,$$

where + is a classical sum, which is a cheap operation. Additionally, the subroutine returns the vector `q` which holds the information for the position of the selected algebraic components. This format is used in the subroutine `numjacobian` to differentiate only the components of $g(x,t)$ that will be selected later, if applicable.

4.1.4 Newton Methods

The user can decide to use a simplified Newton method `simpledivnewton` or a classical Newton method `divnewton` by setting the option `Nopt` to 1 or 0, respectively. Both methods work with a selected Jacobian of $g(x,t)$ as described in Definition 3.14. The implementation of the subroutine `divnewton` is pretty straight-forward. See e.g. [Pla10] for more details. This method computes the numerically expensive inverse of the Jacobian in every iteration step. To reduce the computational effort we introduce the subroutine `simpledivnewton` that proceeds as follows:

1. Initialize with $\mathcal{X}_i$, $\mathcal{X}_{i+1}^d$, `SA`, `sa`, `q`, t and `tol`,
2. Set vector of selected algebraic components $\mathcal{X}_i^a = \mathtt{SA} \cdot \mathcal{X}_i$,
3. Retrieve Jacobian $\mathcal{J}_g$ of $g(\mathcal{X}_{i+1}^d + \mathcal{X}_i^a, t)$ with subroutine `numjacobian` or evaluation of `func_J` (use `q` to calculate only the necessary columns),
4. Select algebraic columns of Jacobian by $\mathcal{J}_g^a = \mathcal{J}_g \cdot \mathtt{sa}$,
5. Calculate QR decomposition of $\mathcal{J}_g^a$, where $QR = \mathcal{J}_g^a P$,
6. Set $C = PR^{-1}$ (this is the only time an inverse is calculated),
7. Set $k = 0$, $\mathcal{X}_k^a = \mathcal{X}_i^a$ and iterate

   (a) Evaluate $g_k = g(\mathcal{X}_{i+1}^d + \mathcal{X}_k^a, t)$,

   (b) Set $\mathcal{X}_{k+1}^a = \mathcal{X}_k^a - \texttt{sa} \cdot CQ^T g_k$ and $k = k + 1$,

   until $|\sum g_k| < \texttt{tol}$,

8. Return $\mathcal{X}_{i+1}^a = \mathcal{X}_k^a$.

The resulting vector $\mathcal{X}_{i+1} = \mathcal{X}_{i+1}^d + \mathcal{X}_{i+1}^a$ then solves $0 = g(\mathcal{X}_{i+1}, t)$. In this iteration we used the QR decomposition which decomposes a matrix A such that $AP = QR$, where Q is an orthogonal matrix ($QQ^T = Q^TQ = I$), R is an upper triangular matrix and P is a permutation matrix. This makes the inversion in Step 6 and 7(b) numerically cheap. For more details refer to [Pla10].

4.1.5 Adaptive Step Size Control

To use adaptive step size control, the option `ssc` has to be set to 1. The user can then define the desired accuracy `eps0` and a safety factor `beta` $\in (0, 1)$ for the step size adjustment. The adaptive step size control follows the very basic theme

1. Initialize the current step with $\mathcal{X}_i$, t, h, p, `eps0` and `beta`,
2. Approximate $\mathcal{X}_{i+1}^h$ and $\mathcal{X}_{i+1}^{\frac{h}{2}}$ (obviously, the latter requires two steps),
3. Set $\varepsilon = \|\mathcal{X}_{i+1}^h - \mathcal{X}_{i+1}^{\frac{h}{2}}\| \cdot (2^p - 1)^{-1}$,
4. Compare ε and `eps0`,
   (a) If $\varepsilon \leq \texttt{eps0}$, accept current step,
   set $t = t + h$ and then $h = h \cdot \texttt{beta}\,(\frac{\texttt{eps0}}{\varepsilon})^{\frac{1}{p+1}}$ for the next step,
   (b) If $\varepsilon > \texttt{eps0}$, set $h = h \cdot \texttt{beta}\,(\frac{\texttt{eps0}}{\varepsilon})^{\frac{1}{p}}$ and reiterate.

Technically, this procedure estimates the error with a Richardson extrapolation. We then adjust the step size regarding the estimated error. The safety factor `beta` is used to avoid many iterations if the problem suddenly gets more stiff. Additionally, there is a slight adjustment in the implementation to hit the desired finish time t_f. For more details about adaptive step size control we refer to [SPW12].

4.2 Academic Examples

At first we are going to examine simple academic examples to demonstrate different possibilities for the inputs of Algorithm 3.3 and get convergence results for the used Runge-Kutta methods with and without selector changes.

4.2.1 Differential Equations

To use the implementation of Algorithm 3.3 for systems of differential equations we can use the form

$$\begin{aligned} \begin{bmatrix} E(x,t) \end{bmatrix} \begin{bmatrix} \dot{x} \end{bmatrix} &= \begin{bmatrix} f(x,t) \end{bmatrix}, \\ \begin{bmatrix} \quad \end{bmatrix} &= \begin{bmatrix} \quad \end{bmatrix}, \end{aligned}$$

where $E(x,t)$ has full rank for all $(x,t) \in \mathbb{D}_x \times \mathbb{I}$. In this case all components of x will be selected as differential components. Algorithm 3.3 will only use steps (3.2a), (3.2c) and (3.2d) and therefore act similarly to a classical Runge-Kutta method.

4.2.2 Algebraic Equations

Since Algorithm 3.3 is designed for overdetermined semi-implicit differential algebraic equations where $m \leq n$ (see Definition 2.9), one has to use systems of the form

$$\begin{aligned} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} \dot{x} \end{bmatrix} &= \begin{bmatrix} f(x,t) \end{bmatrix}, \\ \begin{bmatrix} 0 \end{bmatrix} &= \begin{bmatrix} g(x,t) \end{bmatrix}, \end{aligned}$$

where $g(x,t) = f(x,t)$ for all $(x,t) \in \mathbb{D}_x \times \mathbb{I}$, respectively, to calculate approximations for a nonlinear system of equations $g(x,t)$. In this setting all components of x are selected as required algebraic components. Subsequently, Algorithm 3.3 will only use step (3.2e), when $s = 0$, and proceed like a classical Newton method.

4.2.3 Differential-Algebraic Equations

To obtain first convergence results, let us consider the strangeness-free differential-algebraic system

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x \\ x-y \end{bmatrix}, \qquad 0 = x - y\,,$$

with starting values $t_0 = 0$, $x(t_0) = 1$ and $y(t_0) = 1$. This system has the solution $x(t) = e^t$ and $y(t) = e^t$. To get the convergence order of the used Runge-Kutta method without losses from the used Newton method and Procedure 3.11, we will use high precision for the increment in the differentiation limit and low tolerances. For a simple test, use the following options and inputs for multiple runs over step size h_0.

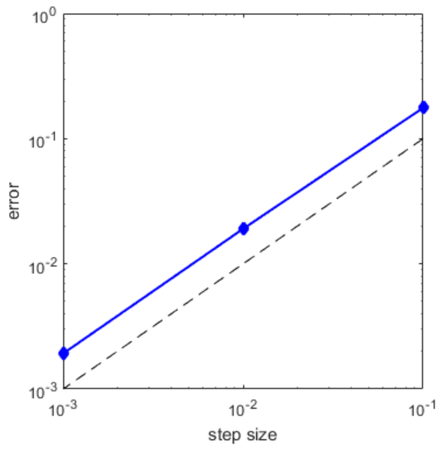
`ssc` $= 0$	`Estat` $= 1$	`Nopt` $= 0$	`Jopt` $= 0$

$[\,t_0, t_f\,] = [\,0, 1\,]$	`tol` $=$ 1e$-$15
$[\,x(t_0), y(t_0)\,]^T = [\,1, 1\,]^T$	`ptol` $=$ 1e$-$15
$h_0^{-1} = \{10, 100, 1000\}$	`delta` $=$ 1e$-$15

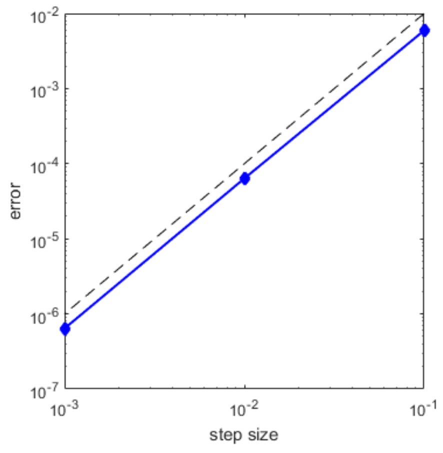
We are going to examine the convergence order of the Runge-Kutta methods (a) - (d) by comparing the numerical results to the solution of the system at $t_f = 1$, where

$$\texttt{error} = \|\,[\,x(1), y(1)\,]^T - [\,e, e\,]^T\,\|.$$

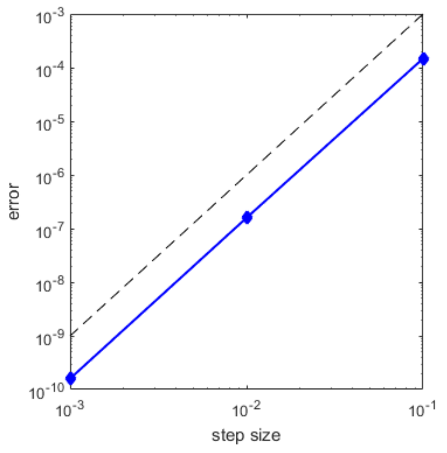
In Figure 3 we use a log-log plot to verify the convergence order. The dashed line gives a reference to the expected slope of the plotted data points. We can see that the error decreases for decreasing step size h_0 and the expected orders of methods (a) - (d) are met.



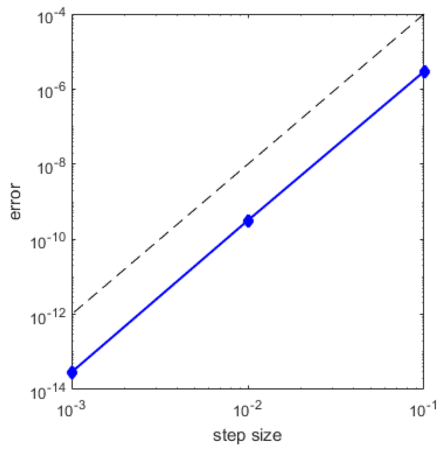
(a) Forward Euler, $p = 0.9812$



(b) Heun, $p = 1.9838$



(c) Kutta 3rd-order, $p = 2.9828$



(d) Classical 4th-order, $p = 4.0043$

Figure 3: Log-log plots, Academic Example

4.2.4 Selector Change Example

In this example we are going to investigate how the selection of the variable differential and algebraic components can change in a given time interval and if a selector change influences the results. Let us use the trigonometric system

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} y \\ -\sin(t)\,z \\ x^2 + y^2 - 1 \end{bmatrix}, \tag{4.1a}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 1 \\ xy - \sin(t)\,yz \end{bmatrix}, \tag{4.1b}$$

where $[\,t_0, t_f\,] = (0, 0.5\pi)$ and set the starting value $x_0 = [\,\sin(t_0), \cos(t_0), 1\,]$. This system has the solution

$$s = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \\ 1 \end{bmatrix}, \tag{4.2}$$

and the Jacobian of (4.1b)

$$\mathcal{J}_g = \begin{bmatrix} 2x & 2y & 0 \\ y & x - \sin(t)\,z & -\sin(t)\,y \end{bmatrix}. \tag{4.3}$$

Again, one can easily see that z is a free component in (4.1a) and thus has to be a required algebraic component. If we insert (4.2) in (4.3) we get

$$\mathcal{J}_g(s,t) = \begin{bmatrix} 2\sin(t) & 2\cos(t) & 0 \\ \cos(t) & \sin(t) - \cos(t) & -\sin(t)\cos(t) \end{bmatrix}. \tag{4.4}$$

When applying Procedure 3.11 to (4.4) one has to compare the entries $2\sin(t)$ and $2\cos(t)$ in Step 6 to find the new pivot to determine the variable algebraic component. Thus we can expect that the selection will change at $t = 0.25\pi$. Let us use the following options and inputs for multiple runs over step size h_0.

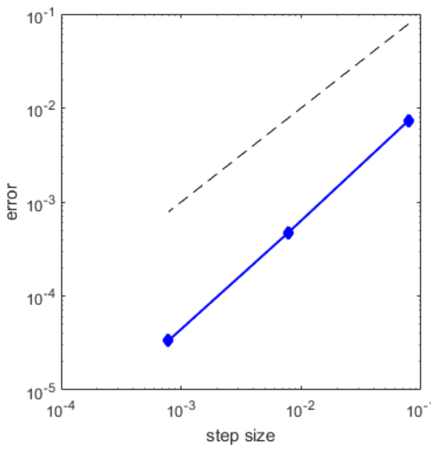
`ssc` $= 0$ `Estat` $= 1$ `Nopt` $= 0$ `Jopt` $= 0$

$[\,t_0, t_f\,] = [\,0.125\pi, 0.375\pi\,]$ `tol` $=$ 1e−15
$[\,x(t_0), y(t_0), z(t_0)\,]^T = [\,\sin(t_0), \cos(t_0), 1\,]^T$ `ptol` $=$ 1e−15
$h_0 = \{0.025\pi,\ 0.0025\pi,\ 0.00025\pi\}$ `delta` $=$ 1e−15

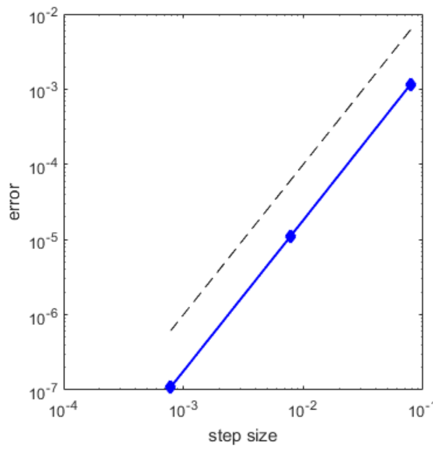
As expected the selectors change when $t > 0.25\pi$ for every tested Runge-Kutta method (a) - (d) and also when using the subroutine `simpledivnewton` instead of `divnewton`. We still get the expected convergence orders depending on the used Runge-Kutta method, where

$$\texttt{error} = \|[\,x(t_f), y(t_f), z(t_f)\,]^T - [\,\sin(0.375\pi), \cos(0.375\pi), 1\,]^T\|,$$

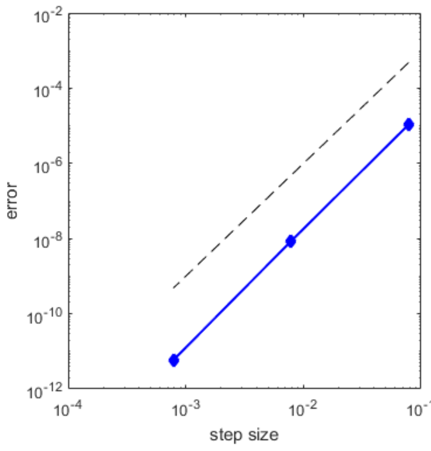
as documented in Figure 4.



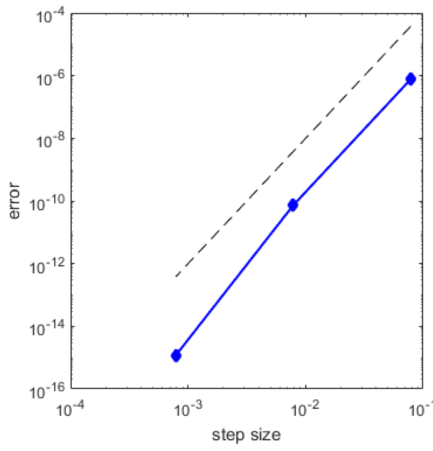
(a) Forward Euler, $p = 1.1704$



(b) Heun, $p = 2.0123$



(c) Kutta 3rd-order, $p = 3.1339$



(d) Classical 4th-order, $p = 4.4095$

Figure 4: Log-log plots, Selector Change Example

4.3 Mathematical Pendulum

As introduced in Example 2.13, we will consider the overdetermined semi-implicit differential-algebraic system

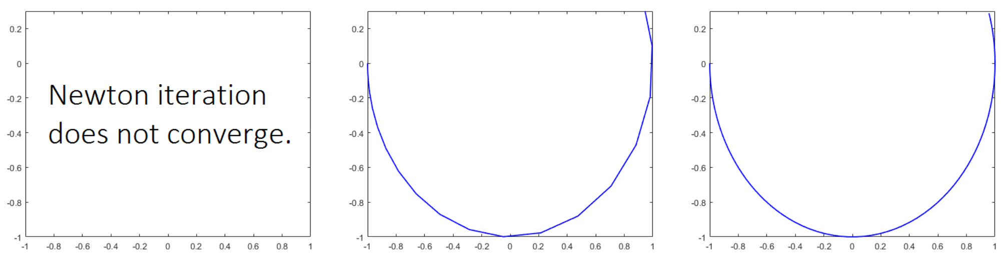
$$
\begin{aligned}
\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{w} \\ \dot{\lambda} \end{bmatrix} &= \begin{bmatrix} v \\ w \\ \text{-}2x\lambda \\ \text{-}2y\lambda - mg \\ x^2 + y^2 - l^2 \end{bmatrix}, \\
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} &= \begin{bmatrix} x^2 + y^2 - l^2 \\ 2xv + 2yw \\ 2v^2 + 2w^2 - \frac{4}{m}(x^2 + y^2)\lambda - 2gy \end{bmatrix}.
\end{aligned}
\tag{4.5}
$$

For a first test use the following options and inputs to investigate the result behavior for Runge-Kutta methods (a) - (d) and step sizes h_0. Modify the gravitational acceleration such that each swing period is exactly two seconds long [Fü88]. Note that $v(t_0), w(t_0)$ and $\lambda(t_0)$ depend on $x(t_0)$ and $v(t_0)$. We will discuss other inputs to investigate precision later.

`ssc` $= 0$	`Estat` $= 1$	`Nopt` $= 0$	`Jopt` $= 0$

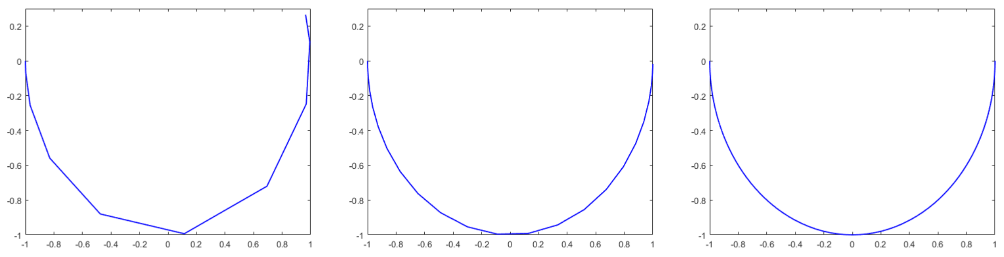
$x(t_0) = -1$	$[\,t_0, t_f\,] = [\,0, 1\,]$
$v(t_0) = 0$	$h_0^{-1} = \{10, 25, 100\}$
$m = 1$	`tol` $=$ 1e$-$10
$l = 1$	`ptol` $=$ 1e$-$15
$g = 13.7503716373294544$	`delta` $=$ 1e$-$13

Let us analyze the following plots showing the position (x, y) of the mass point in the given time interval, which should represent approximately a half period of one pendulum swing. In Figure 5 we see that method (a) either does not converge or not calculate satisfying results for low step counts. The mass point accelerates relatively, such that the system gains energy, and will overshoot the destination. For method (b) and (c) this effect is still present as seen in Figure 6 and Figure 7, but decreases perceptibly when h_0 decreases. The best results were obtained with method (d). As referenced in



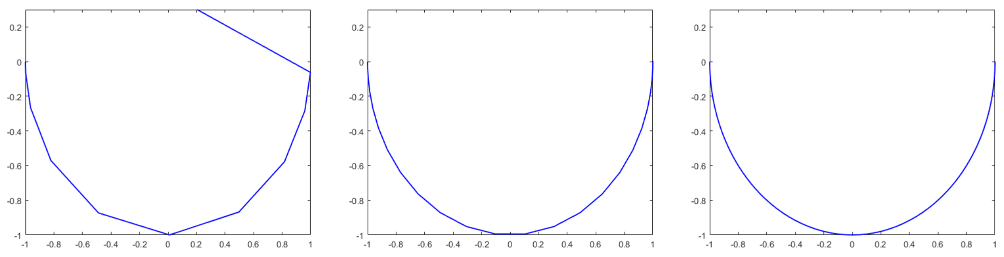
$h_0 = 1/10$ $h_0 = 1/25$ $h_0 = 1/100$

Figure 5: Position (x, y), Mathematical Pendulum, Forward Euler



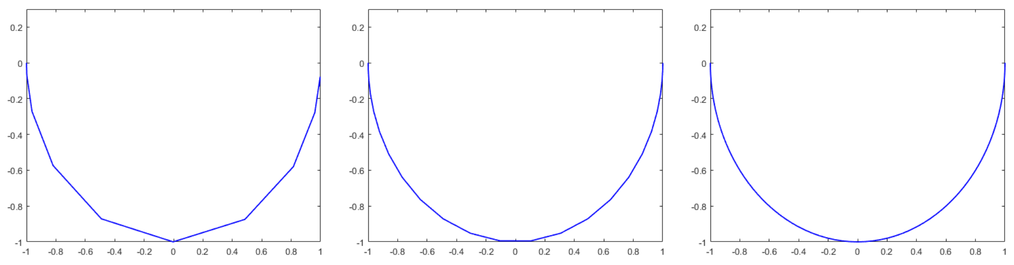
$h_0 = 1/10$ $h_0 = 1/25$ $h_0 = 1/100$

Figure 6: Position (x, y), Mathematical Pendulum, Heun



$h_0 = 1/10$ $h_0 = 1/25$ $h_0 = 1/100$

Figure 7: Position (x, y), Mathematical Pendulum, Kutta 3rd-order



$h_0 = 1/10$ $h_0 = 1/25$ $h_0 = 1/100$

Figure 8: Position (x, y), Mathematical Pendulum, Classical 4th-order

Figure 8 even with $h_0 = 1/10$ we could get respectable approximations for the final step. Due to discretization approximations at intermediate points for $h_0 = 1/10$ are less accurate, but when decreasing h_0 better results are obtained. Next we will crank up precision to investigate the convergence behavior of Algorithm 3.3. As an example, we will use method (c), options as before and the following inputs.

$x(t_0) = -1$	$[t_0, t_f] = [0, 2]$
$v(t_0) = 0$	$h_0^{-1} = 100, 300, 600, 1000$
$m = 1$	$\texttt{tol} = 1\text{e}{-13}$
$l = 1$	$\texttt{ptol} = 1\text{e}{-15}$
$g = 13.7503716373294544$	$\texttt{delta} = 1\text{e}{-14}$

For the given time interval the pendulum should swing for one period, i.e. return to the start point. We are going to determine the error as

$$\texttt{error} = \| [x(t_f), y(t_f), v(t_f), w(t_f), \lambda(t_f)]^T - [-1, 0, 0, 0, 0]^T \|.$$

In Figure 9 we can see that with decreasing step size h_0, the error also decreases. Furthermore, we are still able to reach the expected convergence order $p \approx 3$ as for method (c).

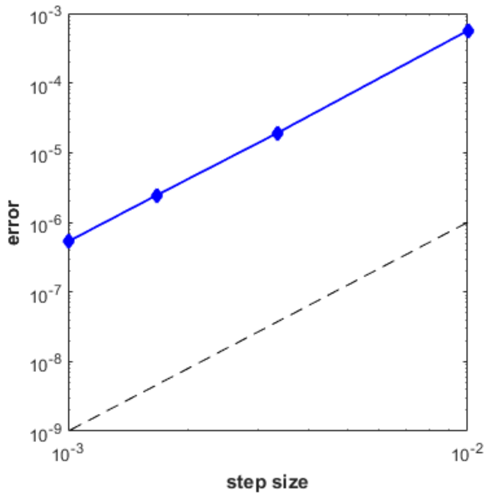


Figure 9: Log-log plot, Mathematical Pendulum, $p = 3.0266$

Let us adjust the inputs again to observe the long time behavior of Algorithm 3.3. We will use method (e), as in the numerical tests where we were able to reach the best results, and the following options and inputs.

`ssc` $= 1$	`Estat` $= 1$	`Nopt` $= 1$	`Jopt` $= 0$

$x_0 = -1$	$[\,t_0, t_f\,] = [\,0, \{20, 200, 2000\}\,]$
$v_0 = 0$	$h_0^{-1} = 100$
$m = 1$	`tol` $= 1\text{e}{-13}$
$l = 1$	`ptol` $= 1\text{e}{-15}$
$g = 13.7503716373294544$	`delta` $= 1\text{e}{-14}$

We will test with different safety factors `beta` and steadily increase the desired accuracy `eps0` for the adaptive step size control. We will then document the required steps and the error in the final step as before for 10, 100 and 1000 periods. Additionally, we will run tests for a formulation that results from a different regularization, where we replace the constraints with their derivatives. The resulting system has strangeness-index $\nu_s = 0$, but omits the constraints respecting position and velocity. Consider

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{w} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} v \\ w \\ -2x\lambda \\ -2y\lambda - mg \\ x^2 + y^2 - l^2 \end{bmatrix}, \tag{4.6}$$
$$0 = 2v^2 + 2w^2 - \tfrac{4}{m}(x^2 + y^2)\lambda - 2gy\,.$$

Note that we have to include the given constraint in (4.6), because λ is a required algebraic component and the Jacobian of $g(x, t)$ needs to be regular as stated in Criterion 3.6. The results are listed in Table 1. The tests did not encounter any unpleasant surprises, because the mathematical pendulum is a non-stiff problem. Generally, we can summarize that using system (4.5) over (4.6) gains roughly ten times higher precision per ten periods. Also, using `beta` $= 0.7$ over `beta` $= 0.9$ can increase precision two or sometimes three times. By using system (4.6) over (4.5) and also for larger `beta` we

System	t_f	`beta`	`eps0`	1e−5	1e−6	1e−7	1e−8	1e−9	1e−10	1e−11	1e−12
(4.5)	20	0.7	`error`	3.33e−2	3.07e−3	2.51e−4	2.24e−5	2.20e−6	1.62e−7	1.85e−8	1.15e−8
			steps	577	918	1451	2319	3611	5745	9093	14346
(4.6)	20	0.7	`error`	4.83e−1	6.61e−2	6.20e−3	6.12e−4	7.15e−5	8.48e−6	1.21e−6	1.20e−7
			steps	528	821	1295	2046	3231	5106	8075	12772
(4.5)	20	0.9	`error`	1.90e−2	4.56e−3	7.48e−4	8.00e−5	7.37e−6	6.32e−7	5.29e−8	1.18e−8
			steps	493	743	1147	1813	2852	4490	7097	11178
(4.6)	20	0.9	`error`	2.34e−1	3.04e−2	3.45e−3	5.24e−4	1.20e−4	2.10e−5	3.40e−6	5.52e−7
			steps	444	676	1041	1618	2530	3982	6287	9947
(4.5)	200	0.7	`error`	9.88e+0	3.50e−1	3.04e−2	3.09e−3	3.53e−4	3.63e−5	4.30e−6	6.04e−7
			steps	5738	9069	14485	23179	36087	57456	90824	143440
(4.6)	200	0.7	`error`	∅	∅	1.71e+1	5.01e−1	5.16e−2	6.37e−3	9.33e−4	1.50e−5
			steps			13091	20457	32294	51042	80734	127702
(4.5)	200	0.9	`error`	4.94e+0	5.98e−1	8.97e−2	1.06e−2	1.13e−3	1.24e−4	1.39e−5	1.64e−6
			steps	4902	7403	11457	18087	28499	44868	70962	111758
(4.6)	200	0.9	`error`	∅	1.97e+1	5.90e+0	3.80e−1	8.75e−2	1.58e−2	2.61e−3	4.05e−4
			steps		7279	10447	16177	25282	39799	62855	99452
(4.5)	2000	0.7	`error`	1.70e+1	2.03e+1	7.66e+0	3.28e−1	3.68e−2	3.84e−3	4.58e−4	5.49e−5
			steps	56506	90705	144836	231758	360838	574544	908571	1434361
(4.6)	2000	0.7	`error`	∅	∅	∅	∅	9.07e+0	2.10e+1	1.12e+0	1.50e−1
			steps					327359	511227	807500	1277112
(4.5)	2000	0.9	`error`	6.71e+0	7.25e−1	1.48e+1	1.41e+0	1.17e−1	1.31e−2	1.48e−3	1.70e−4
			steps	47963	74876	114505	180791	284983	448649	709593	1117553
(4.6)	2000	0.9	`error`	∅	∅	∅	∅	1.04e+1	5.18e+0	5.60e+0	4.19e−1
			steps					259170	399526	628916	994592

Table 1: Test results mathematical pendulum

obtain lower step counts, as expected. The tests denoted by "$\oslash$" failed to converge. Figure 10 shows the numerical solution of the different components of the state for the first ten periods of system (4.5) with `eps0` $= 1\text{e}{-7}$ and `beta` $= 0.7$.

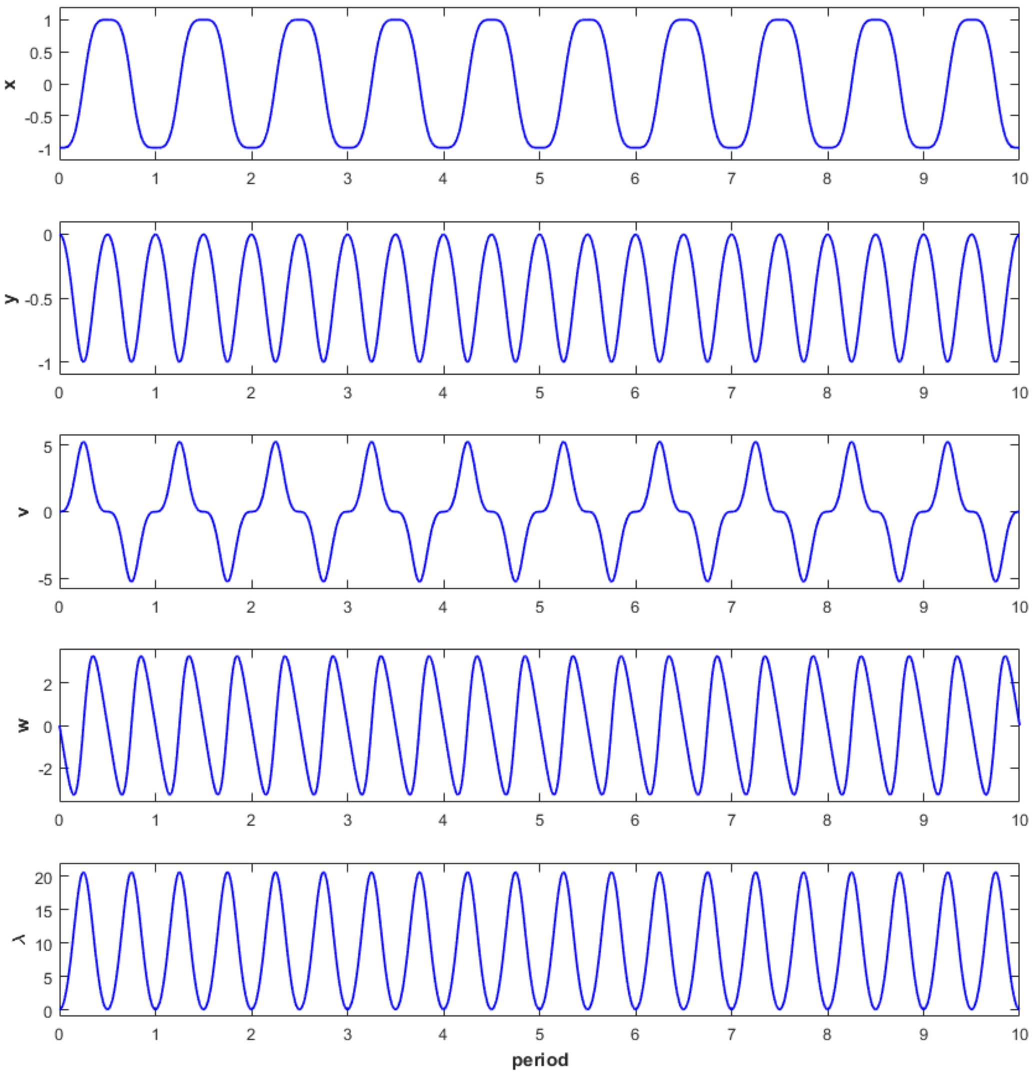


Figure 10: Numerical solution of (4.5) for ten periods

4.4 Modified Pendulum in Changing Wind Conditions

We will examine a modified pendulum where the length $l : \mathbb{I} \to \mathbb{R}_{>0}$ is variable over time and the whole system is under the influence of horizontal wind. The model is derived similarly to Example 2.13 but we will consider an external force within the Lagrange equation as

$$F_{ex} = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{q}}\mathcal{L}(q,\dot{q})\right) - \frac{\partial}{\partial q}\mathcal{L}(q,\dot{q}).$$

With $F_{ex} = [\, m\alpha \;\; 0 \;\; 0 \,]^T$, where $\alpha : \mathbb{I} \to \mathbb{R}$, working in a horizontal direction and order reductions $\dot{x} = v$ and $\dot{y} = w$, we can derive the system

$$\begin{aligned} \dot{x} &= v \\ \dot{y} &= w \\ m\dot{v} &= -2x\lambda + m\alpha \\ m\dot{w} &= -2y\lambda - mg \\ 0 &= x^2 + y^2 - l^2. \end{aligned} \tag{4.7}$$

Then with the regularization discussed in Procedure 2.4 we can write system (4.7) in the overdetermined semi-implicit form as

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{w} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} v \\ w \\ -2x\lambda + m\alpha \\ -2y\lambda - mg \\ x^2 + y^2 - l^2 \end{bmatrix},$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - l^2 \\ 2xv + 2yw \\ 2v^2 + 2w^2 + 2x\alpha - \frac{4}{m} l^2\lambda - 2gy \end{bmatrix}.$$

We mainly consider this system to test if Algorithm 3.3 works well in the case that both input parameters in $f(x,t)$ and $g(x,t)$ are necessary to evaluate the functions. Especially in (3.2b), (3.2c) and (3.2e) the algorithm depends on c given by the Runge-Kutta method. We certainly do not have an analytical solution for the problem, but it offers a chance to play around a little bit. We will use method (c) and the following options and inputs.

`ssc` $= 0$ `Estat` $= 1$ `Nopt` $= 0$ `Jopt` $= 0$

$x(t_0) = 0$	$[\,t_0, t_f\,] = [\,0, 50\,]$
$v(t_0) = 0$	$h_0 = 1/50$
$m = 2$	`tol` $= 1\text{e}{-10}$
$l = 2$	`ptol` $= 1\text{e}{-15}$
$g = 9.78$	`delta` $= 1\text{e}{-12}$

Note that again $y(t_0)$, $w(t_0)$ and $\lambda(t_0)$ depend on the given starting values. Additionally, we will use the following functions to simulate changing horizontal wind and pulling up (shortening) the pendulum.

$\alpha(t) = 1 - 0.04\,t$	$l(t) = 0.5 + 0.03\,(50 - t)$

As a result we get the position (x, y) of the mass point of the crane pendulum for $t = [\,0, 50\,]$ as drawn in Figure 11.

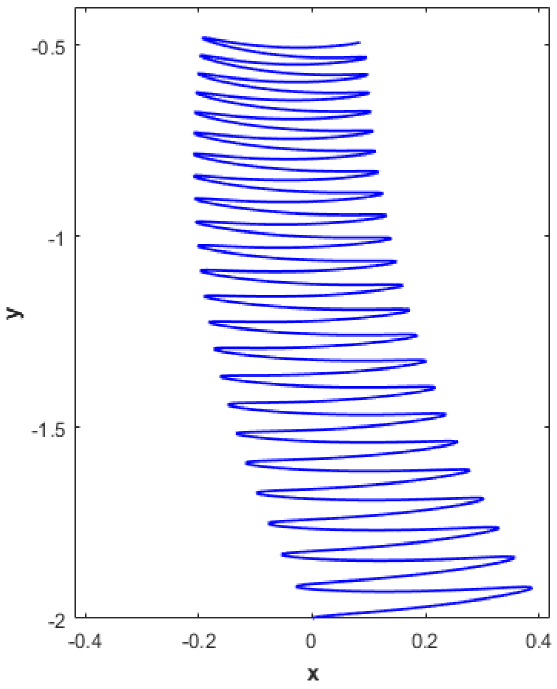


Figure 11: Position (x, y), Modified Pendulum with $\alpha(t)$ and $l(t)$

4.5 Spring-Mass Chain

In this example we are going to investigate a spring-mass chain with three masses and two springs as in Figure 12. The inner mass is constrained to

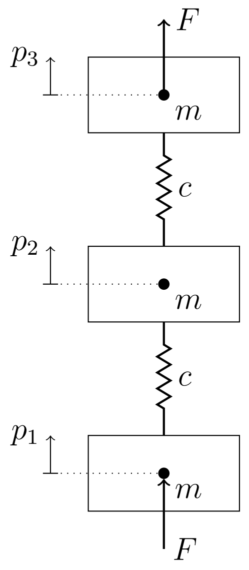


Figure 12: Spring-mass chain

follow a prescribed path, while the forces on the outer masses act as a control. We can model the scenario with the equations (see e.g. [Ste16])

$$\begin{aligned}
\dot{p_1} &= v_1, \\
\dot{p_2} &= v_2, \\
\dot{p_3} &= v_3, \\
m\dot{v_1} &= F - c(p_1 - p_2), \\
m\dot{v_2} &= c(p_1 - p_2) - c(p_2 - p_3), \\
m\dot{v_3} &= F + c(p_2 - p_3), \\
0 &= \sin(t) - p_2,
\end{aligned} \tag{4.8}$$

where the unknowns p_i and v_i are the positions and velocities of the mass points $i \in \{1, 2, 3\}$, and F describes the acting forces on the outer masses. The constants m and c describe the mass of the mass points and the stiffness of the springs. Again, with Procedure 2.4 we can find the hidden constraints of system (4.8) and formulate the regularized overdetermined semi-implicit system

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \\ \dot{p}_3 \\ \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_3 \\ \dot{F} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ F - c(p_1 - p_2) \\ c(p_1 - p_2) - c(p_2 - p_3) \\ F + c(p_2 - p_3) \\ p_2 - \sin(t) \end{bmatrix},$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} p_2 - \sin(t) \\ v_2 - \cos(t) \\ \frac{c}{m}(p_1 - p_2) - \frac{c}{m}(p_2 - p_3) + \sin(t) \\ \frac{c}{m}(v_1 - v_2) - \frac{c}{m}(v_2 - v_3) + \cos(t) \\ \frac{c}{m^2}(-3c(p_1 - p_2) + 3c(p_2 - p_3) + 2F) - \sin(t) \end{bmatrix}.$$

We will use method (d) and following options and inputs for the investigation.

`ssc` $= 1$	`Estat` $= 1$	`Nopt` $= 0$	`Jopt` $= 0$

$[\,t_0, t_f\,] = [\,0, 400\,]$	`eps0` $=$ 1e$-$7
$h_0 = 1/1000$	`beta` $= 0.8$
$m = 1$	`tol` $=$ `eps0`
$c = 1/6$	`ptol` $=$ 1e$-$15
	`delta` $=$ 1e$-$14

With the starting value

$$x(t_0) = [\,p_1(0), p_2(0), p_3(0), q_1(0), q_2(0), q_3(0), F(0)\,]^T = [\,0, 0, 0, \text{-}2, 1, \text{-}2, 0\,]^T$$

we are given the analytical solution

$$\begin{aligned} p_1^*(t) &= -2\sin(t), & p_2^*(t) &= \sin(t), & p_3^*(t) &= -2\sin(t), \\ v_1^*(t) &= -2\cos(t), & v_2^*(t) &= \cos(t), & v_3^*(t) &= -2\cos(t), \\ F^*(t) &= 1.5\sin(t), \end{aligned}$$

and determine the error of the numerical approximation $x(t)$ with

$$\texttt{error} = \| \, x(t) - [\ p_1^*(t), p_2^*(t), p_3^*(t), v_1^*(t), v_2^*(t), v_3^*(t), F^*(t)\ ]^T \, \|.$$

For the first setting the numerical solution and the error for the given time interval are plotted in Figure 13 and Figure 14, respectively. We can ob-

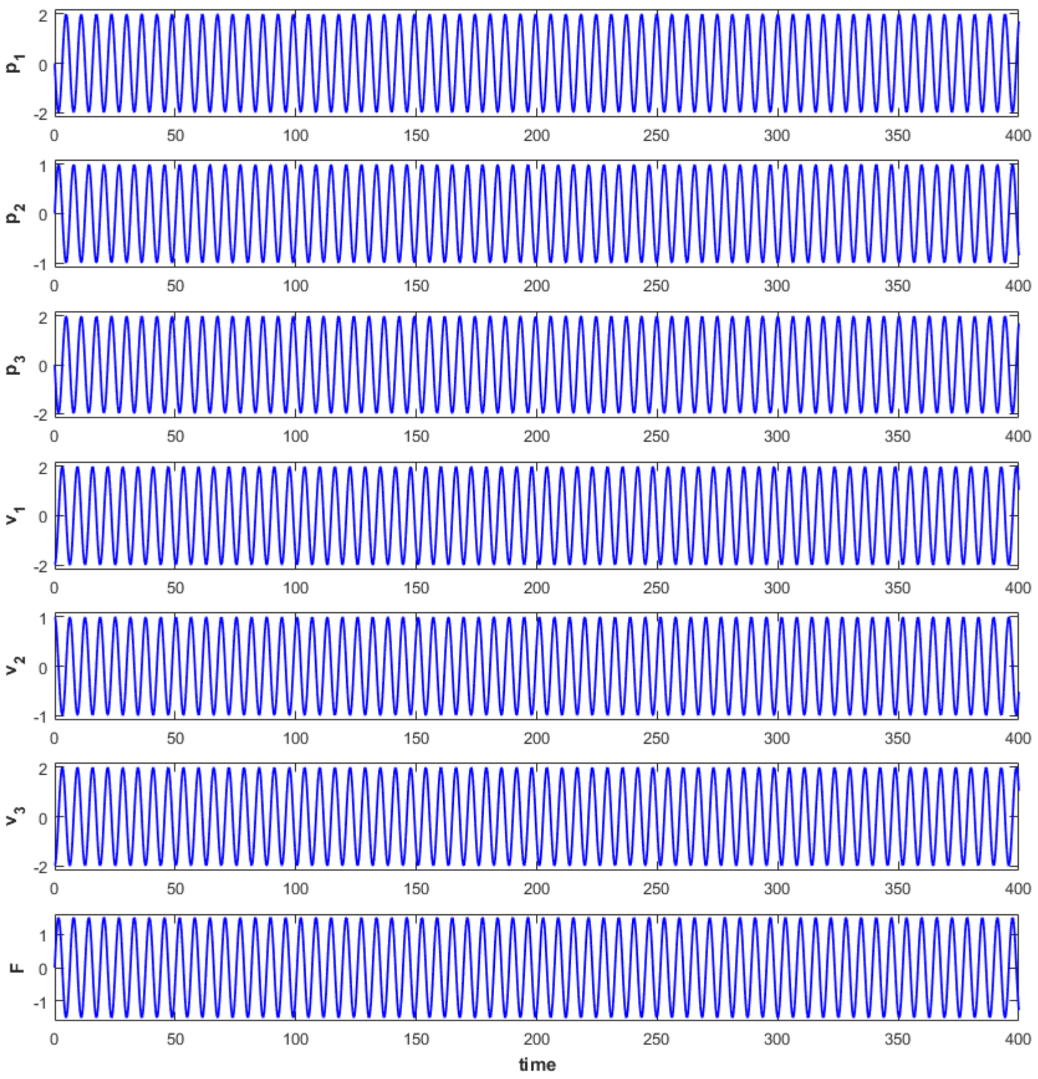


Figure 13: Numerical solution for Spring-Mass Chain

serve that the numerical approximation does not deviate from the analytical solution and that the error does not accumulate. For the next test we are going to use the same settings but iterate the desired accuracy from

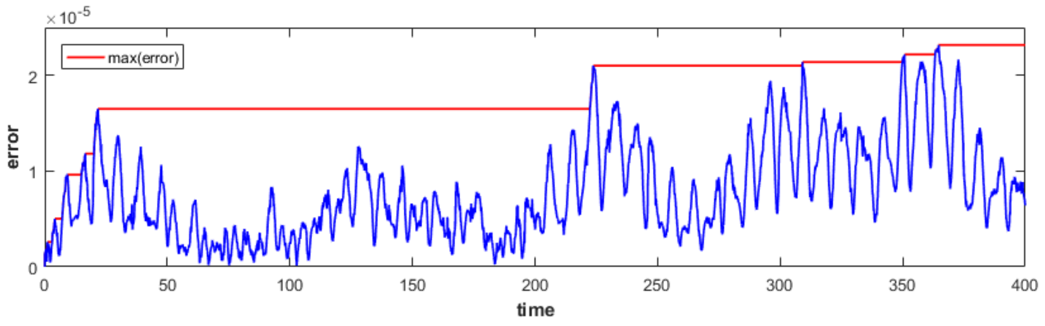


Figure 14: Error of numerical solution for Spring-Mass Chain

`eps0` = 1e−3 to `eps0` = 1e−13 and alter the option `Nopt` = $\{0, 1\}$ to determine the behavior of Algorithm 3.3 with the different subroutines `divnewton` and `simpledivnewton`. Figure 15 shows that the performance advantage of `simpledivnewton` might be questionable for high accuracies. The fact that

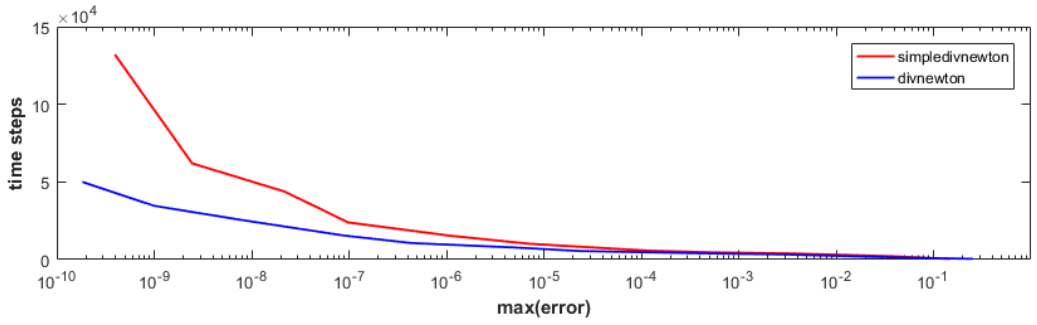


Figure 15: Step count for desired accuracy in numerical solution

more steps are required is easily explained because the adaptive step size control counteracts the simplification in the Newton iteration. Once a larger step is taken the quality of the calculated Jacobian decreases and the step might not be accepted. An approach to resolve this issue would be to find a better initial guess for the Newton iteration. This could be done by extrapolating previously known approximations within the Runge-Kutta step for the algebraic components [Ste06], but is not yet implemented.

4.6 Linear Circuit with one CV Loop

Let us return to Example 2.7 describing a model of a linear circuit. We already determined the hidden constraints and can write system (2.3) in the overdetermined semi-implicit form

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot q_1 \\ \dot q_2 \\ \dot e_1 \\ \dot e_2 \\ \dot i_V \end{bmatrix} = \begin{bmatrix} e_1 + i_V \\ e_2 \\ e_1 - \sin(100t) \\ q_1 - e_1 + e_2 \\ q_2 - e_2 \end{bmatrix},$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} e_1 - \sin(100t) \\ q_1 - e_1 + e_2 \\ q_2 - e_2 \\ 2e_1 + e_2 + 2i_V + 100\cos(100t) \end{bmatrix}.$$

We will use method (d) and the following options and inputs for our investigation.

`ssc` $= 1$	`Estat` $= 1$	`Nopt` $= 1$	`Jopt` $= 0$

$[\,t_0, t_f\,] = [\,0, 1\,]$	`tol` $=$ 1e$-$10
$h_0 = 1/1000$	`ptol` $=$ 1e$-$15
`eps0` $=$ 1e$-$10	`delta` $=$ 1e$-$13
`beta` $= 0.9$	

With the starting value

$$x(t_0) = [\,q_1(0), q_2(0), e_1(0), e_2(0), i_V(0)\,]^T = [\,0, 0, 0, 0, \text{-}50\,]^T$$

we are given the analytical solution (see [Bäc07])

$$\begin{aligned}
e_1^*(t) &= \sin(100t),\\
e_2^*(t) &= \tfrac{100}{40001}\cos(100t) + \tfrac{20000}{40001}\sin(100t) - \tfrac{100}{40001}\, e^{-\frac{1}{2}t},\\
q_1^*(t) &= e_1(t) - e_2(t),\\
q_2^*(t) &= e_2(t),\\
i_V^*(t) &= -\tfrac{2000100}{40001}\cos(100t) - \tfrac{50001}{40001}\sin(100t) + \tfrac{50}{40001}\, e^{-\frac{1}{2}t}.
\end{aligned}$$

This example is interesting, because multiple components are required to be treated algebraically (compare Procedure 3.11). Only one of the four algebraic components is a variable algebraic component. We are going to determine the error during the given time interval as

$$\begin{aligned}
\texttt{error} = \|\, [\, q_1(t), q_2(t), e_1(t), e_2(t), i_V(t)\,]^T - \\
[\, q_1^*(t), q_2^*(t), e_1^*(t), e_2^*(t), i_V^*(t)\,]^T \,\|
\end{aligned}$$

Figure 16 suggests that the error propagates when lengthening the time interval. But, several numerical tests with different settings showed that the error is bounded.

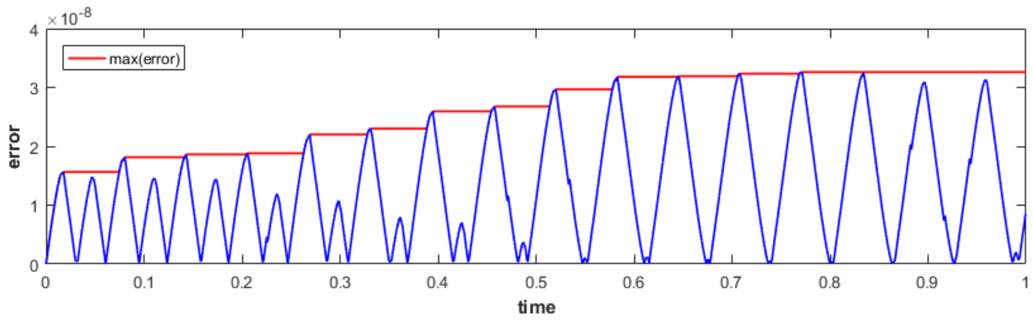


Figure 16: Error of numerical solution for Linear Circuit

Further, the required algebraic component e_1 is completely independent of the differential treatment of the system. Thus, this component is solely determined with the used Newton-method in step (3.2b) and (3.2e) of Algorithm 3.3. This is also reflected when investigating the error for each component of the approximation. Figure 17 shows that the error of the used Runge-Kutta method does not carry over to the component e_1.

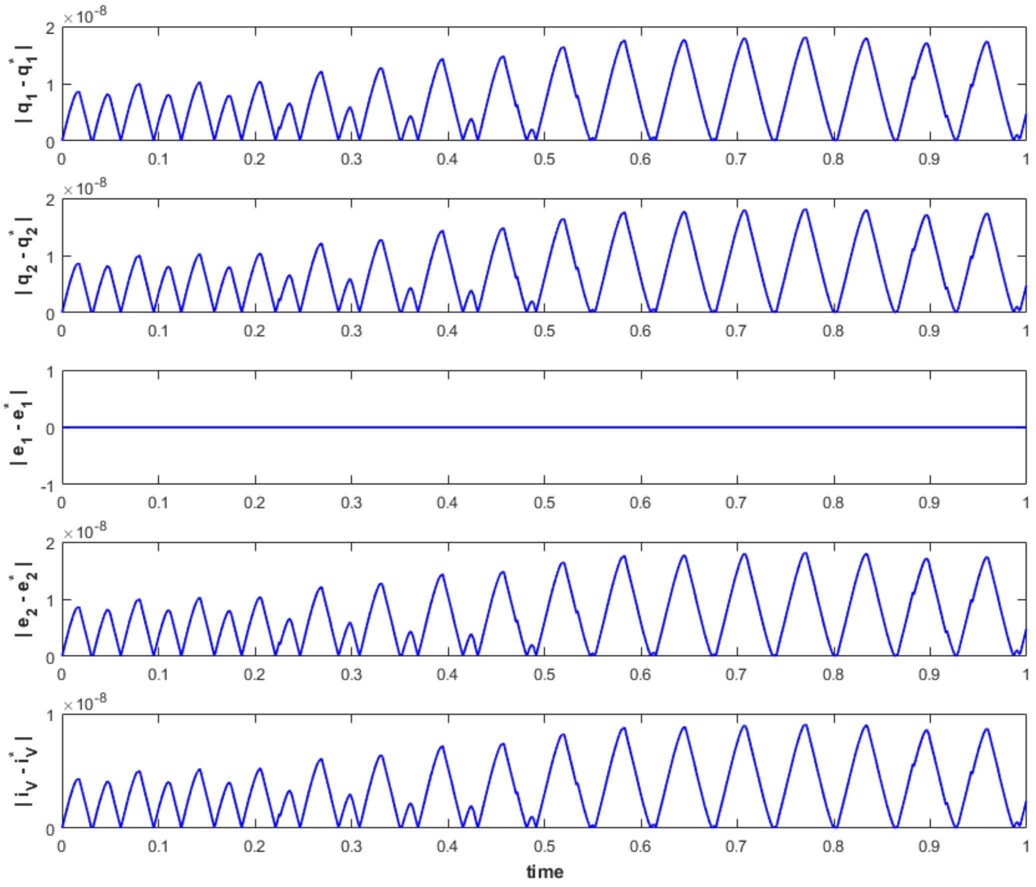


Figure 17: Error in components of numerical solution for Linear Circuit

4.7 Chemical Akzo Nobel Problem

The following example is taken from the test set [MMI06]. It describes the mixing process of two chemicals with the a steady inflow of carbon dioxide. The solution vector holds the information regarding the concentration of the given chemicals and the chemicals resulting from the process. For the chemical details and information about the model refer to [Sto98]. The problem can be formulated as a strangeness-free overdetermined semi-implicit differential-algebraic system given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \\ \dot{y}_5 \\ \dot{y}_6 \end{bmatrix} = \begin{bmatrix} -2r_1 + r_2 - r_3 - r_4 \\ -0.5r_1 - r_4 - 0.5r_5 + F_{in} \\ r_1 - r_2 + r_3 \\ -r_2 + r_3 - 2r_4 \\ r_2 - r_3 + r_5 \\ K_s\, y_1 y_4 - y_6 \end{bmatrix},$$

$$0 = K_s\, y_1 y_4 - y_6,$$

where the auxiliary variables are defined as

$$r_1 = k_1 y_1^4 \sqrt{y_2}, \qquad r_2 = k_2 y_3 y_4, \qquad r_3 = k_2 y_1 y_5 K^{-1},$$
$$r_4 = k_3 y_1 y_4^2, \qquad r_5 = k_4 y_6^2 \sqrt{y_2}, \qquad F_{in} = klA(p(\mathrm{CO}_2)/H - y_2),$$

with constants $k_1 = 18.7$, $k_2 = 0.58$, $k_3 = 0.09$, $k_4 = 0.42$, $K = 34.4$, $klA = 3.3$, $K_s = 115.83$, $p(\mathrm{CO}_2) = 0.9$ and $H = 737$.

Let us use method (e) and the following options and inputs to examine how the adaptive step size control performs for this example.

`ssc` $= 1$	`Estat` $= 1$	`Nopt` $= 1$	`Jopt` $= 0$

$[\,t_0, t_f\,] = [\,0, 180\,]$	`tol` $=$ 1e−7
$h_0 = 1/100$	`ptol` $=$ 1e−15
`eps0` $=$ 1e−7	`delta` $=$ 1e−14
`beta` $= 0.78$	

With the starting value

$$y(t_0) = [\,0.444, 0.00123, 0, 0.007, 0, 0.444 \cdot 0.007 K_s\,]^T$$

we are given the reference solution

$$y^*(t_f) = \begin{bmatrix} 0.1150794920661702 \\ 0.0012038314715677 \\ 0.1611562887407974 \\ 0.0003656156421249 \\ 0.0170801088526440 \\ 0.0048735313103074 \end{bmatrix}.$$

The components of the solution over the given time interval are plotted in Figure 18. Here we see that the solution behaves quite smoothly, except that component y_2 falls drastically in the beginning.

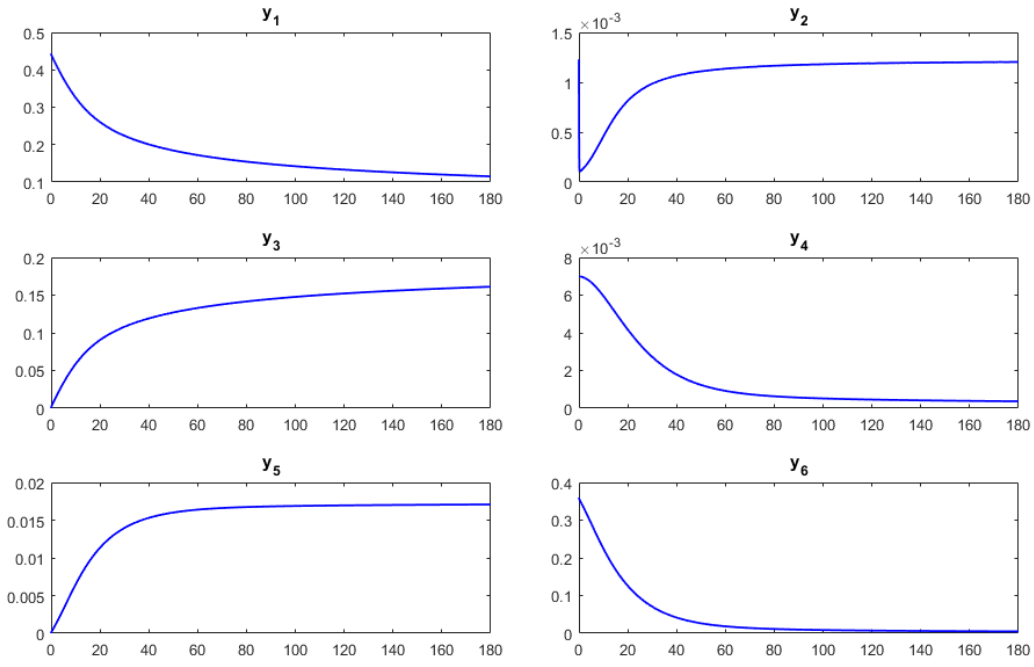


Figure 18: Components of Chemical Akzo Nodel Problem

This behavior is also reflected in Figure 19, where the step size is small in the beginning, because the problem is stiff, but later changes to rather large steps. In this setting we get step sizes as high as $h \approx 3.5$ with the adaptive

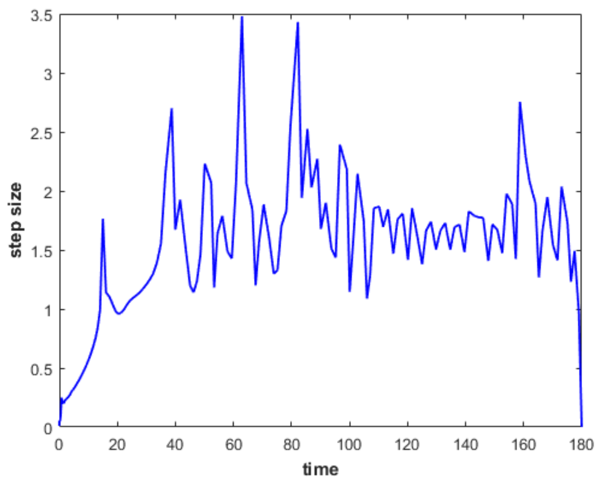


Figure 19: Step size with adaptive step size control

step size control discussed above. In the following we will measure the error with the numerical solution $y(t_f)$ by

$$\texttt{error} = \| \, y(t_f) - y^*(t_f) \, \|.$$

Steadily raising `eps0` $= 1\text{e}{-6}$ to `eps0` $= 1\text{e}{-16}$ shows that the desired accuracy can be met with the given settings. This certainly does not hold for very high accuracies, because we get close to the employed machine's precision. Figure 20 documents the required amount of steps to reach set accuracies.

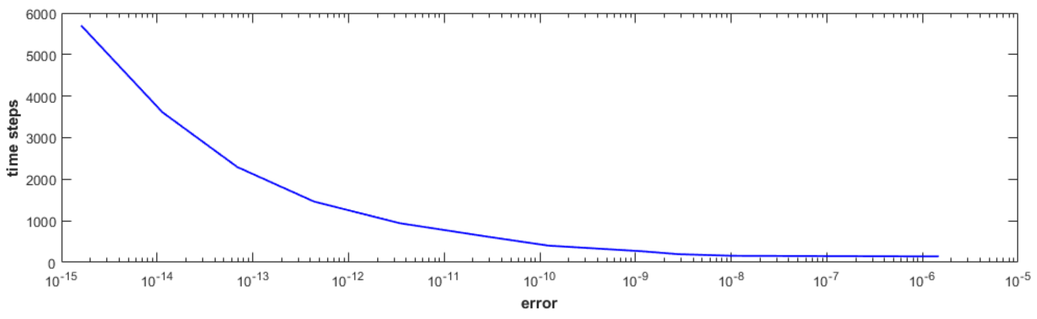


Figure 20: Error versus required steps

5 Summary

In this thesis we presented an approach to use half-explicit Runge-Kutta methods to solve overdetermined semi-implicit differential-algebraic equations. Starting with general quasi-linear differential-algebraic equations we followed the path of regularization to obtain an overdetermined formulation containing the hidden constraints of the system. Considering these we drew the theoretical outline of the numerical treatment of such systems by half-explicit Runge-Kutta methods. A key prerequisite of Algorithm 3.3 is Criterion 3.6. We then introduced a procedure which makes a automatic choice of the selectors, selecting whether a component should be treated algebraically or differentially, possible. Lastly, we presented the implementation of Algorithm 3.3 and tested its functionality with several examples.

The observant reader will already have noticed that we admitted some major drawbacks with the current approach. First, Criterion 3.6 minimizes the set of possible differential-algebraic equations drastically. Serving more information about the system may provide a generalized approach. For example, by adding the derivative $\dot{x}_0$ of the initial state $x_0 = x_0^a \oplus x_0^d$, we could iterate, following the notation of Algorithm 3.3,

$$\left.\begin{aligned}
\mathcal{X}_i^d &= x_0^d + h \cdot \textstyle\sum_{j=1}^{i-1} a_{i,j}\, \dot{\mathcal{X}}_j^d, \\
0 &= g(\mathcal{X}_i^d \oplus \mathcal{X}_i^a, t_0 + c_i h), \\
\dot{\mathcal{X}}_i &= \begin{cases} \dot{x}_0 & \text{for } i = 0, \\ \frac{\mathcal{X}_i - x_0}{c_i h} & \text{for } i > 0, \end{cases}
\end{aligned}\right\} \quad \text{for } i = 1, ..., s$$

$$\begin{aligned}
x_1^d &= x_0^d + h \cdot \textstyle\sum_{j=1}^{s} b_j\, \dot{\mathcal{X}}_j^d, \\
0 &= g(x_1^d \oplus x_1^a, t_0 + h).
\end{aligned}$$

That way we do not need to assume that $E(x,t)$ is regularly reducible, but admit a less precise approximation of $\dot{\mathcal{X}}_i$ within the iteration. This might lead to additional order conditions for the used Runge-Kutta method as discussed in [BH93]. Also in that case the selection procedure needs to be reviewed. But only then is a useful study of the stability properties of the algorithm possible (see e.g. [KM07]).

Additionally, one might argue that the regularization of the given system is too computationally expensive. In some cases it might be nearly impossible on paper or only attainable with costly use of computer algebra systems such as Mathematica. Then, when summarizing the computational effort, it only makes sense to apply Algorithm 3.3 if the regularized system is used for multiple calculations.

Generally we investigated only the precision of Algorithm 3.3 without concern of the computation speed. The current implementation shall be viewed as a test to validate that such an approach is viable. The praised computational advantages by solving reduced nonlinear systems can not be savored because execution times in Matlab are not competitive compared to implementations in e.g. Fortran or C++. Additionally, the current implementation does not exploit the given structure well enough to reach fast convergence within the Newton iteration. Thus, most of the execution time is spent with solving nonlinear systems. In Section 4.5 we could even observe that the adaptive step size control and simplified Newton method may counteract each other. Also, one should consider to review the selection procedure, because the selection is not necessary in each time step. In Criterion 3.6 we already stated that the Jacobian of the hidden constraints has to be regular. This property can be used to develop certain mechanisms to trigger the selection procedure once the matrix is close to singular. These ideas forecast a lot of room for tweaks to speed up the program.

References

[Alt14] Yair Altman. *Accelerating MATLAB Performance: 1001 Tips to Speed Up MATLAB Programs.* Chapman and Hall/CRC, Boca Raton, FL, United States of America, 2014.

[Bäc07] Simone Bächle. *Numerical Solution of Differential-Algebraic Systems Arising in Circuit Simulation.* PhD thesis, Technische Universität Berlin, 2007.

[BCP96] Kathryn Brenan, Stephen Campbell, and Linda Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* SIAM Publications, Philadelphia, PA, United States of America, 1996.

[BH93] Valérie Brasey and Ernst Hairer. Half-Explicit Runge-Kutta Methods for Differential-Algebraic Systems of Index 2. *SIAM Journal on Numerical Analysis, Vol. 30(2), 538-552*, 1993.

[Cal96] Malvin Calkin. *Lagrangian and Hamiltonian Mechanics.* World Scientific Publishing, Singapore, 1996.

[Deu04] Peter Deuflhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms.* Springer-Verlag, Berlin, Germany, 2004.

[Fü88] Claus Führer. *Differential-algebraische Gleichungssysteme in mechanischen Mehrkörpersystemen : Theorie, numerische Ansätze und Anwendungen.* PhD thesis, Technische Universität München, 1988.

[GL13] Gene Golub and Charles Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, United States of America, 2013.

[HLR89] Ernst Hairer, Christian Lubich, and Michel Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods.* Springer-Verlag, Berlin, Germany, 1989.

[KM06] Peter Kunkel and Volker Mehrmann. *Differential-Algebraic Equations. Analysis and Numerical Solution.* European Mathematical Society Publishing House, Zürich, Schwitzerland, 2006.

[KM07] Peter Kunkel and Volker Mehrmann. Stability properties of differential-algebraic equations and spin-stabilized discretizations. *Electronic Transactions on Numerical Analysis, Vol. 26, 385-420*, 2007.

[LM13] Vu Linh and Volker Mehrmann. Efficient integration of matrix-valued non-stiff DAEs by half-explicit methods. *Journal of Computational and Applied Mathematics, Vol. 262, 346-360*, 2013.

[Meh12] Volker Mehrmann. Index concepts for differential-algebraic equations. ePrints, Institut für Mathematik, Technische Universität Berlin, Germany, 2012.

[MMI06] Francesca Mazzia, Cecilia Magherini, and Felice Iavernaro. Test Set for Initial Value Problem Solvers. Report 43, Department of Mathematics, University of Bari, Italy, 2006.

[Pla10] Robert Plato. *Numerische Mathematik kompakt.* Vieweg+Teubner Verlag, Wiesbaden, Germany, 2010.

[SPW12] Karl Strehmel, Helmut Podhaisky, and Rüdinger Weiner. *Numerik gewöhnlicher Differentialgleichungen. Nichtsteife, steife und differential-algebraische Gleichungen.* Vieweg+Teubner Verlag, Wiesbaden, Germany, 2012.

[Ste06] Andreas Steinbrecher. *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems.* PhD thesis, Technische Universität Berlin, 2006.

[Ste15] Andreas Steinbrecher. Regularization of Quasi-Linear Differential-Algebraic Equations. *IFAC-PapersOnLine, Vol. 48(1), 300-305*, 2015.

[Ste16] Andreas Steinbrecher. Signature Method Based Regularization and Numerical Integration of DAEs. Technical Report 2016/01, Institut für Mathematik, Technische Universität Berlin, 2016.

[Sto98] Walter Stortelder. *Parameter Estimation in Nonlinear Dynamical Systems.* PhD thesis, Centrum Wiskunde en Informatica Amsterdam, 1998.