

**Reducing the Index of Differential-Algebraic
Equations by Exploiting Underlying
Structures**

Ross McKenzie

A thesis
submitted to the School of Mathematics
of Cardiff University
in partial fulfilment of the requirements
for the degree of
PhD of Mathematics

2016

SUMMARY. Differential-algebraic equations arise from the equation based modelling of physical systems, such as those found for example in engineering or physics. This thesis is concerned with square, sufficiently smooth, potentially non-linear differential-algebraic equations. Differential-algebraic equations can be classified by their index. This is a measure of how far a differential-algebraic equation is from an equivalent ordinary differential equation. To solve a differential-algebraic equation one usually transforms the problem to an ordinary differential equation, or something close to one, via an index reduction algorithm. This thesis examines how the index reduction (using dummy derivatives) of differential-algebraic equations can be improved via structural analysis, specifically the Signature Matrix method. Improved and alternative algorithms for finding dummy derivatives are presented and then a new algorithm for finding globally valid universal dummy derivatives is presented. It is also shown that the structural index of a differential-algebraic equation is invariant under order reduction.

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed (candidate)

Date

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD of Mathematics.

Signed (candidate)

Date

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated, and the thesis has not been edited by a third party beyond what is permitted by Cardiff University's Policy on the Use of Third Party Editors by Research Degree Students. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed (candidate)

Date

Statement 3

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed (candidate)

Date

Statement 4: Previously Approved Bar On Access

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loans after expiry of a bar on access previously approved by the Academic Standards and Quality Committee.

Signed (candidate)

Date

Acknowledgements

It would not have been possible to write this thesis and carry out the work within it without the help and support of so many people around me. To mention just a few here feels like an injustice to anyone I may miss out, but I shall endeavour to do my best.

This thesis would not have been possible without the expertise and guidance of my supervisor John Pryce, for his constant support I am and have been extremely grateful.

When visiting Ned Nedialkov over several summers in Canada he always did his utmost to make me feel welcome and to make my stay enjoyable and productive, for that I thank him.

I would also like to thank Peter Harman for his guidance and industrial perspective.

Without Guangning Tan to act as a constant sounding board and friend I have no doubt my time as a PhD student would have been a much less rewarding and fruitful one. I particularly would like to thank him for entertaining me on my visits to McMaster.

I thank the Leverhulme Trust and Cardiff University Mathematics Department for their financial support during the completion of this thesis.

My time in the applied mathematics group at Cardiff was made much more enjoyable due to the many opportunities afforded to me by the SIAM Student Chapter there and for that I would like to thank everyone involved in the chapter.

I would like to thank my family, particularly my parents Robert and Lindsey, whose support I cannot thank them for enough.

Last but by no means least I would like to thank all my friends I've made both in Cardiff and elsewhere without whom I'm quite convinced I wouldn't be where I am today, you know who you are and thank you, really, thank you.

Contents

Chapter 1. Introduction	1
1.1. Motivation	2
1.2. Common Issues, A Summary of Notation and Terms	4
1.3. Different Index concepts	6
Chapter 2. Structural Analysis	11
2.1. Introduction to Structural Analysis - The Signature Matrix Method	11
2.2. A Brief Overview	11
2.3. Linear Assignment Problems	13
2.4. Signature Method By Example	21
2.5. Signature Method - Basics	25
2.6. Standard Solution Scheme	28
2.7. Exploiting DAE BTFs	37
2.8. Classifying Non-Canonical Offsets	39
Chapter 3. Dummy Derivatives	45
3.1. Introduction to Dummy Derivatives	45
3.2. Original Dummy Derivative Algorithm	46
3.3. Reordered Dummy Derivative Algorithm	52
3.4. Using Structural Analysis to Simplify Dummy Derivatives	54
3.5. Alternative Algorithms	66
3.6. Dummy Pivoting	79
Chapter 4. Exploiting Non-Canonical Offsets—Universal Dummy Derivatives	87
4.1. The Basic Algorithm	87

4.2. The Reduced Universal Dummy Derivative Form	95
4.3. Numerical Results for Universal Dummy Derivatives	103
Chapter 5. Order Reduction Leaving the Structural Index Unchanged	109
5.1. Introduction to Order Reduction - Why It's Non-Trivial for DAEs	109
5.2. Order Reduction and the Structural Index	111
5.3. Invariant DOF Under Order Reduction	114
5.4. Order Reduction and Canonical Offsets	117
5.5. Invariant Structural Index Under Order Reduction	130
Chapter 6. Conclusions and Future Work	133
6.1. Conclusions	133
6.2. Future Work	134
Appendix A. Code for the Simple Pendulum Using the Signature Matrix Method	137
Appendix B. Code for the Simple Pendulum Using Universal Dummy Derivatives	145
Bibliography	155

CHAPTER 1

Introduction

Differential-algebraic equations (henceforth DAEs) arise from the equation based modelling of physical systems, such as those found in engineering or physics, with problems specifically arising from chemical distillation [61], electronic circuits [60] and robotics [7]. Models are now frequently built interactively using different components from large libraries in environments such as GPROMS [43], MAPLESIM [26], SIMULINK [27] and an assortment of tools that use the Modelica modelling language (e.g. OpenModelica [16], Dymola [12] and SimulationX [23]). This way of modelling systems can lead to large scale DAEs [17]. A common notion in DAE theory is the *differentiation index* [6], see §1.2. It is well known that solving a high index (larger than one) DAE directly is numerically difficult [19], hence modelling tools usually perform some structural analysis to determine the index of the problem. Up until recently this analysis was usually based on Pantelides' algorithm [41], although we will be using the Signature Matrix method [45] (see Chapter 2) for our analysis, as it applies to DAEs of arbitrary order (see Chapter 5) and provides us with extra structural information we wish to exploit (see Chapter 3). After finding a DAE's index the problem is usually to convert (or reduce) the DAE to an equivalent system of index 1 or 0 (an ODE), as these problems are easily solvable by commonly used solvers, e.g. DASSL [42], SUNDIALS [22] and recently the MATLAB solver ode15i [57]. Thus our aim is to reduce a large class of high index DAEs to index 1 or 0 algorithmically. We now provide a brief overview of this thesis. Chapter 1 provides an introduction to differential-algebraic equations and different ways of classifying them. Chapter 2 provides an overview of the Structural Analysis via a Signature Matrix, with detail and derivations not previously seen in the literature. The chapter concludes with new material on classifying different potential offset vectors, which has proved fruitful for developing new index reduction algorithms. Chapter 3 gives a brief

overview of the Dummy Derivative index reduction procedure and then goes on to give new algorithms to speed up the procedure. The chapter provides new material on unifying the method with the Signature Matrix method and goes on to give several algorithms for finding necessary dummy derivatives and dummy derivatives efficiently using a block form. Chapter 4 gives a new algorithm for finding dummy derivatives that are globally valid, thereby removing some of the complications inherent in the original algorithm. Finally Chapter 5 provides a proof that the structural index is invariant under a specific form of order reduction. Such a result was not known in the literature previously and is of course very important in practice. If the structural index could increase under order reduction then one would have to be particularly careful when applying the dummy derivative algorithm, since it can yield an index 1 DAE (or ODE) with order larger than 1 when applied to a high index DAE of order 1. Chapter 6 provides a summary of conclusions for each chapter and future works the author believes would prove fruitful.

1.1. Motivation

Before giving any motivation we need to define what we mean by DAEs. We write $\left(x_j^{(\ell)}\right)_{(j,\ell)\in J}$ with $J = \{(j, \ell) \mid j \in \{1, \dots, n\} \text{ and } \ell \in \{0, \dots, l_j\}\}$ to compactly denote n variables and their derivatives with respect to t up to some order l_j , where l_j depends on variable x_j . We define a general DAE:

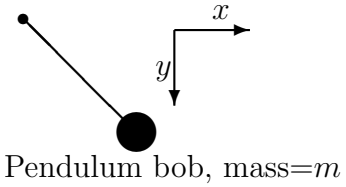
DEFINITION 1.1.1 (A Differential Algebraic Equation). A differential algebraic equation, DAE, is a set of m equations and n variables, dependent on a variable t usually considered to be time. We label equations by i and variables by j so that we have derivatives of each variable up to some order l_j . More compactly

$$(1) \quad f_i(t; \left(x_j^{(\ell)}\right)) = 0,$$

where $\ell \in \{0, \dots, l_j\}$ and $i \in \{1, \dots, n_1\}$, $j \in \{1, \dots, n_2\}$.

We avoid the case where the DAE is over or under determined and only look at the case where $n = n_1 = n_2$ for the majority of this thesis. Perhaps the most straightforward

and informative physical example of a DAE is found by considering the simple pendulum in Cartesian coordinates. We measure from left to right in the x direction, downwards in the y direction and employ Newton's law to find equations for the force in these directions. We keep the length of the pendulum fixed, so that it will be moving in a circle. The variables x and y represent displacement in the horizontal and vertical directions respectively and λ is a multiple of tension in the rod, all of which depend on time t . We let G be the constant for acceleration due to gravity, L be the length of the rod and obtain the following equations:

$$(2) \quad \begin{cases} a(t) = m\ddot{x}(t) + \lambda(t)x(t) & = 0, \\ b(t) = m\ddot{y}(t) + \lambda(t)y(t) - mG & = 0, \\ c(t) = x^2(t) + y^2(t) - L^2 & = 0. \end{cases}$$


For the duration of the thesis we will assume $m = 1$ for simplicity. We now briefly check these equations make sense via dimensional analysis and discover how λ relates to tension. Firstly $[\ddot{x}] = [LT^{-2}]$ which from a means that $[\lambda x] = [\lambda L]$ and thus for consistency $[\lambda] = [T^{-2}]$. Now, $[\text{tension}] = [MLT^{-2}]$ and thus for consistency we must have that $[\lambda] = [\frac{\text{tension}}{ML}]$. We also note that b and c are dimensionally correct, since b concerns only the dimension $[LT^{-2}]$ and c has terms only in $[L^2]$. This formulation (2) contains both differential and algebraic equations and it will be shown later that it requires differentiation in order to solve for x , y and λ . Hence (2) is indeed an interesting DAE (all ODEs are DAEs, albeit not interesting ones for the purposes of this thesis). Clearly this is an artificial example, as one could instead write the simple pendulum in polar coordinates as an ODE in θ :

$$(3) \quad \ddot{\theta} + \frac{G}{L} \sin \theta = 0.$$

Rewriting any arbitrary DAE as an ODE, or something 'close' to an ODE (index 1 DAE, see Definition 1.2.1 for more details), is an open non-trivial problem. There have been techniques developed that attempt to solve this problem, the Dummy Derivative method [29] being the most commonly employed. This thesis aims to develop a method of doing such a conversion by exploiting the structural analysis of the Signature Matrix method of [45] to inform the

Dummy Derivative algorithm [30] to efficiently find globally valid conversions for a wide class of DAEs.

1.2. Common Issues, A Summary of Notation and Terms

We begin by considering the simple DAE (where $u(t)$ is some arbitrary function):

$$\begin{cases} x_1(t) - u(t) = 0, \\ x_1(t) - \dot{x}_2(t) = 0. \end{cases}$$

Solving this DAE requires *integrating* $x_2(t) = \int u(t) dt + C$ where C is a constant. So this is actually an ODE (or index 0 DAE, see Definition 1.2.1). Consider however the DAE where instead of differentiating $x_2(t)$ we differentiate $x_1(t)$ in the second equation:

$$(4) \quad \begin{cases} x_1(t) - u(t) = 0, \\ \dot{x}_1(t) - x_2(t) = 0. \end{cases}$$

Solving requires *differentiating* the first equation, yielding $x_2(t) = \dot{u}(t)$, so this is not an ODE (it's an index 2 DAE).

Let us consider again the simple pendulum example (2) and attempt to solve it naively. First let's try to reduce the problem to an ODE where we can solve for the highest order derivatives of each variable in each equation. First λ appears undifferentiated throughout, so we will need to differentiate equations a and b . This will mean we now have equations for $x^{(3)}(t)$ and $y^{(3)}(t)$, so we will have to differentiate $c(t)$ three times. This gives us the following, which can be solved to an ODE in $x^{(3)}(t)$, $y^{(3)}(t)$ and $\dot{\lambda}(t)$:

$$(5) \quad \begin{cases} \dot{a}(t) = x^{(3)}(t) + \dot{\lambda}(t)x(t) + \lambda(t)\dot{x}(t) & = 0, \\ \dot{b}(t) = y^{(3)}(t) + \dot{\lambda}(t)y(t) + \lambda(t)\dot{y}(t) & = 0, \\ c^{(3)}(t) = 6\dot{x}(t)\ddot{x}(t) + 2x(t)x^{(3)}(t) + 6\dot{y}(t)\ddot{y}(t) + 2y(t)y^{(3)}(t) & = 0. \end{cases}$$

However, it turns out this problem has 7 degrees of freedom, whereas our original DAE formulation has only 2—one will obtain the same solution when starting at a consistent point of (2), but there are many solutions to (5) other than just those that satisfy (2). This phenomenon is called *integration drift*. This drift occurs because in solving our ODE (5) we do not satisfy the so called *hidden constraints* of the problem:

$$(6) \quad \begin{cases} a(t) = \ddot{x}(t) + \lambda(t)x(t) & = 0, \\ b(t) = \ddot{y}(t) + \lambda(t)y(t) - G & = 0, \\ c(t) = x^2(t) + y^2(t) - L^2 & = 0, \\ \dot{c}(t) = 2x(t)\dot{x}(t) + 2y(t)\dot{y}(t) & = 0, \\ \ddot{c}(t) = 2\dot{x}(t)^2 + 2x(t)\ddot{x}(t) + 2\dot{y}(t)^2 + 2y(t)\ddot{y}(t) & = 0. \end{cases}$$

e.g for the pendulum the length of our rod will vary, since the circle constraint equation c is not explicitly satisfied. There are two intuitive ways one may attempt to satisfy these constraints that will be explored in this thesis. The first is to think of these equations as specifying a *consistent manifold* and project to it—this is done in the Signature matrix method, see Chapter 2 and [45]. The second is to add the equations to the system directly, then locally choose some of the variables to be considered as algebraic (so as to retain a square system) — this is done in the Dummy Derivative method, see Chapter 3 and [29]. The above example motivates us to define a concept of distance from an equivalent ODE, given in [3] for first order DAEs:

DEFINITION 1.2.1 (Differentiation Index). The minimum number of times that all (or part) of a DAE $F(t, \mathbf{x}, \dot{\mathbf{x}}) = 0$ has to be differentiated with respect to t to determine $\dot{\mathbf{x}}$ as a function of \mathbf{x} and t is the differentiation index.

Note: this index may be locally valid depending on the underlying structure of the DAE. For the purposes of this thesis we consider it to be the largest globally valid number of differentiations. The differentiation index is only defined for first order DAEs, $F(t, \mathbf{x}, \dot{\mathbf{x}}) = 0$.

If we reformulate to order 1 we get:

$$(7) \quad \dot{x}(t) - u(t) = 0,$$

$$(8) \quad \dot{y}(t) - v(t) = 0,$$

$$(9) \quad \dot{u}(t) - \lambda(t)x(t) = 0,$$

$$(10) \quad \dot{v}(t) - \lambda(t)y(t) - G = 0,$$

$$(11) \quad (x(t))^2 + (y(t))^2 - L^2 = 0..$$

Differentiating (11) and using (7) and (8) to simplify gives:

$$(12) \quad u(t)x(t) + v(t)y(t) = 0.$$

Differentiating (12) and using equations (7)—(11) to simplify yields:

$$(13) \quad \lambda(t)L^2 - Gy(t) + (u(t))^2 + (v(t))^2 = 0.$$

Differentiating (13) and using equations (7)—(12) to simplify leads to:

$$(14) \quad \dot{\lambda}(t)L^2 - 3Gv(t) = 0.$$

We can now use equations (7)—(14) to solve for $\dot{x}(t)$, $\dot{y}(t)$, $\dot{\lambda}(t)$, $\dot{u}(t)$ and $\dot{v}(t)$ so the DAE has differentiation index 3.

1.3. Different Index concepts

There are several different concepts of index for DAEs other than the differentiation index, such as the *perturbation index*, *tractability index*, *geometric index*, *strangeness index*, *kroneker index*, *nilpotency index* and *structural index* to name a few, for more information see [34]. What follows is a brief overview of some of these concepts to give the reader an overview of what is meant when one talks about the index of a DAE. Throughout this thesis we shall use the differentiation and structural index.

1.3.1. Derivative Array. A common notion in DAE index classification is the Campbell-Gear derivative array equations [25]. This is a the set of equations necessary such that one can find the differentiation index (i.e. solve for first order derivatives of all variables.) For example, consider the first order DAE:

$$F(t, \mathbf{x}, \dot{\mathbf{x}}) = 0.$$

Following the process illustrated for the simple pendulum in the previous section one will have to differentiate some set of these equations to solve for each derivative uniquely. Given a more complicated DAE than the simple pendulum it may not be obvious which equations need to be differentiated, so we differentiate each equation ν_d times say:

$$\begin{aligned} \frac{d}{dt} F(t, \mathbf{x}, \dot{\mathbf{x}}) &= 0, \\ \frac{d^2}{dt^2} F(t, \mathbf{x}, \dot{\mathbf{x}}) &= 0, \\ &\vdots \\ \frac{d^{\nu_d}}{dt^{\nu_d}} F(t, \mathbf{x}, \dot{\mathbf{x}}) &= 0. \end{aligned}$$

This enlarged set of equations is called the derivative array (of level ν_d). If we are able to solve for all first order derivatives of all variables at level ν_d (and not able to at a previous level) then ν_d is the differentiation index of the DAE $F(t, \mathbf{x}, \dot{\mathbf{x}}) = 0$. As we can see from considering such an array for the simple pendulum the number of equations one has to consider in this approach can become very large, so we have a preference to working with a reduced derivative array if possible.

1.3.2. Strangeness Index. The strangeness index was developed by Kunkel and Mehrmann in [24]. It is found as the result of a larger hypothesis, for simplicity we illustrate the concept on the linear constant coefficient DAE of differentiation index 2 and delay the definition until after:

$$\begin{cases} \dot{x}(t) - y(t) - u(t) = 0, \\ x(t) - v(t) = 0. \end{cases}$$

In matrix form this is given by:

$$\begin{pmatrix} I_s & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} = \begin{pmatrix} 0 & I \\ I_s & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{pmatrix} u(t) \\ v(t) \end{pmatrix}.$$

Here $u(t)$ and $v(t)$ are arbitrary forcing functions and I_s is an identity matrix of size s , corresponding to the number of differential variables. Kunkel and Mehrmann consider the I_s entry in both position (1, 1) of the left hand matrix and (2, 1) of the right hand matrix as ‘strange’, one has some number of differential and algebraic variables and thus must differentiate as was demonstrated above in order to find $\dot{y}(t)$. If one differentiates the second equation and subtracts the first from it then they get:

$$y(t) + u(t) + \dot{v}(t) = 0$$

and a new matrix system:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} = \begin{pmatrix} 0 & I \\ I_s & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{pmatrix} u(t) + \dot{v}(t) \\ v(t) \end{pmatrix}$$

where this strangeness has been eliminated. Hence this problem has strangeness index 1.

Given an arbitrary first order constant coefficient linear DAE

$$E\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{f}(t),$$

one looks for permutation matrices P and Q (via a singular value decomposition (SVD)) such that the DAE reduces to a canonical form:

$$\begin{pmatrix} I_s & 0 & 0 & 0 \\ 0 & I_a & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \dot{\mathbf{x}}(t) = \begin{pmatrix} 0 & * & 0 & * \\ 0 & * & 0 & * \\ 0 & 0 & I_a & 0 \\ I_s & 0 & 0 & 0 \end{pmatrix} \mathbf{x}(t) + \mathbf{f}(t).$$

Here $*$ is some potentially non-zero entry. If I_s exists then the system exhibits some strangeness, a differentiation is performed, new permutation matrices are found and the new differentiated system's I_s are checked iteratively. If I_s does not exist then the system is strangeness free and we stop our iteration. The number of iterations of this procedure to arrive at a strangeness free DAE is called the strangeness index. For a full discussion of the strangeness index concept and its use in numerical simulation we refer to [54].

1.3.3. Perturbation Index. A DAEs index in some sense quantifies how difficult a DAE is to solve numerically, the perturbation index considers this problem explicitly. When solving a DAE numerically the resulting solution will be slightly removed from the actual solution, this index gives us a way of measuring the sensitivity of the DAE to such perturbations, [4] and [13].

DEFINITION 1.3.1 (Perturbation Index). Consider a DAE and a slightly perturbed DAE:

$$F(t, \mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0},$$

$$F(t, \hat{\mathbf{x}}, \dot{\hat{\mathbf{x}}}) = \boldsymbol{\delta}.$$

If \mathbf{x} is a solution to the unperturbed DAE, then the DAE has perturbation index k along \mathbf{x} if k is the smallest number such that for a solution to the perturbed DAE $\hat{\mathbf{x}}$ the following bound holds with constant C :

$$\|\mathbf{x} - \hat{\mathbf{x}}\| \leq C(\|\mathbf{x}(t_0) - \hat{\mathbf{x}}(t_0)\|_\infty + \|\boldsymbol{\delta}\| + \|\dot{\boldsymbol{\delta}}\| + \dots + \|\boldsymbol{\delta}^{(k-1)}\|).$$

For example, recall DAE (4) and perturb the system to:

$$\begin{cases} \hat{x}_1(t) - u(t) = \delta(t), \\ \dot{\hat{x}}_1(t) - \hat{x}_2(t) = 0, \end{cases}$$

so that $\hat{x}_1(t) = u(t) + \delta(t)$ and $\hat{x}_2 = \dot{u}(t) + \dot{\delta}(t)$. Consider our norm to be the standard Euclidean norm for simplicity, although the following argument works for any norm:

$$\|\mathbf{x}(t) - \hat{\mathbf{x}}(t)\| = \left\| \begin{pmatrix} \delta(t) \\ \dot{\delta}(t) \end{pmatrix} \right\| \leq \left\| \begin{pmatrix} \delta(t) \\ 0 \end{pmatrix} \right\| + \left\| \begin{pmatrix} 0 \\ \dot{\delta}(t) \end{pmatrix} \right\| \leq C \left(\left\| \begin{pmatrix} \delta(t_0) \\ \dot{\delta}(t_0) \end{pmatrix} \right\|_{\infty} + \|\delta(t)\| + \|\dot{\delta}(t)\| \right).$$

Hence we have a perturbation index of 2. From [18] we have that the perturbation index is always equal to or one greater than the differentiation index. The perturbation index is extended for $k = 0$ in [34] so that, when defined, the perturbation index equals the differentiation index, but a further discussion is outside the scope of this thesis.

1.3.4. Geometric Index. The leading principle behind the geometric index [50] and [52] (which given a smooth DAE is equal to the differentiation index) is to think of DAEs as differential equations on manifold, e.g. an ODE is differential equation on the manifold \mathbb{R}^n . The idea is to construct a sequence of sub-manifolds via local charts (each stage in the sequence being equivalent to a level of differentiation in the derivative array). The number of differentiations one goes through to write the DAE as a differential equation on a manifold is called the geometric index.

1.3.5. Structural Index. Rather than looking at the numerical features of a given DAE the structural index aims to use structural properties to generate a necessary set of differentiation steps to produce a reduced derivative array. In practice this is usually done either by the graph theoretical Pantelides algorithm [41] or the matrix based signature matrix method [45]. We defer an in depth discussion of how this index is calculated to Chapter 2. In all that follows, unless otherwise stated, when we refer to the index of a DAE we will mean its structural index, denoted ν_s .

CHAPTER 2

Structural Analysis

2.1. Introduction to Structural Analysis - The Signature Matrix Method

The *Signature matrix method* originally given in [45] by Pryce is a technique that studies a DAEs underlying structure to identify all necessary hidden constraints. Originally the approach was discovered by means of looking for a Taylor series solution to arbitrary order for fully non-linear DAE's [44], although we show in Chapter 3 that we can use the method to inform other solution methods. The general idea is that, given a high index DAE we know there exist some number of hidden constraints that are derivatives of equations given by the original problem formulation in variables at higher order than in the original problem. One seeks to match some subset of all derivatives of each equation at an arbitrary order to a minimum number of derivatives at an arbitrary order that are necessary to solve for the highest order derivatives of each variable. In §2.3 and §2.4 we show how one can derive the method from first principles, which has not been done in detail in the literature. In §2.5 we give an overview of the method and demonstrate how it works on simple examples. In §2.8 we describe the different types of offset vector the method can use to provide a valid solution scheme—such distinctions have not been previously studied in the literature and are new material for the purposes of this thesis, without which the method of §4 would not be possible. In §2.7 we give natural sparsity patterns and block forms one can consider when applying the signature method.

2.2. A Brief Overview

We give a brief overview of the signature matrix method to give later exposition context. Take a system of n equations $f_i = 0$ in n unknowns x_j , where the equations may have derivatives of the n unknowns. The unknowns x_j are all functions of t , usually considered to

be time, so that we do not have any partial derivatives in our initial formulation. We begin by finding the problem's signature matrix Σ , with entries:

$$(15) \quad \sigma_{i,j} = \begin{cases} \text{Order of highest derivative of } x_j \text{ in } f_i & \text{if } x_j \text{ occurs in } f_i, \\ -\infty & \text{otherwise.} \end{cases}$$

The approach then finds a *highest value transversal* (HVT) for our problem. A transversal T is any choice of n positions (i, j) in an $n \times n$ matrix, say H , such that only one entry in each row and column of H is selected. We say the value of a transversal T is $\text{Val}(T) = \sum_{(i,j) \in T} \sigma_{i,j}$. A HVT, say T_1 , is such that $\text{Val}(T_1) \geq \text{Val}(T)$ for all transversals T . We denote the value of a HVT as $\text{Val}(\Sigma)$ and say a DAE is structurally non singular if $\text{Val}(\Sigma) \geq 0$.

We then find non-negative integer valued offset vectors \mathbf{c} and \mathbf{d} satisfying:

$$(16) \quad \sigma_{i,j} \leq d_j - c_i \quad \text{with equality on a HVT,}$$

and we call such offset vectors valid. We will usually consider another constraint on the offsets, $\min_i c_i = 0$, and we call such offsets normalised.

The method uses a System Jacobian \mathbf{J} with entries:

$$(17) \quad \mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j^{(d_j - c_i)}} = \begin{cases} \frac{\partial f_i}{\partial x_j^{(\sigma_{i,j})}} & \text{if } d_j - c_i = \sigma_{i,j}, \\ 0 & \text{elsewhere.} \end{cases}$$

A stage by stage solution process, with stages indexed by k , is constructed using equations

$$(18) \quad f_i^{(k+c_i)} \quad \forall i \text{ such that } k + c_i \geq 0$$

to solve for variables

$$(19) \quad x_j^{(k+d_j)} \quad \forall j \text{ such that } k + d_j \geq 0$$

at each stage.

Throughout our analysis we will write Σ and \mathbf{J} in tableau form as appropriate, e.g. for a

3×3 DAE:

$$\Sigma = \begin{array}{cccc} & x_1 & x_2 & x_3 & c_i \\ f_1 & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_1 \\ f_2 & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_2 \\ f_3 & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_3 \\ d_j & d_1 & d_2 & d_3 \end{array}$$

$$\mathbf{J} = \begin{array}{cccc} & x_1^{(d_1)} & x_2^{(d_2)} & x_3^{(d_3)} & c_i \\ f_1^{(c_1)} & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_1 \\ f_2^{(c_2)} & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_2 \\ f_3^{(c_3)} & \left(\begin{array}{ccc} * & * & * \end{array} \right) & c_3 \\ d_j & d_1 & d_2 & d_3 \end{array}$$

entries * that equal $-\infty$ and (structural) 0's will be left blank in Σ and \mathbf{J} respectively. We will also make use of a *sparsity pattern* (a subset of positions (i, j)) corresponding to non blank entries in these tableau's, S for Σ and S_0 for \mathbf{J} . From this we have the following equation for calculating the structural index, which is always an upper bound on the differentiation index ν_s :

$$(20) \quad \nu_s = \max_i c_i + \begin{cases} 1 & \text{if some } d_j \text{ is zero,} \\ 0 & \text{otherwise.} \end{cases}$$

2.3. Linear Assignment Problems

We seek to find an algorithm for finding the offsets from an arbitrary structurally non-singular signature matrix Σ , as specified by (16). We consider linear programming problems (specifically, primal problems) and their duals. It is worth noting the dual to a dual problem is the primal problem. We will view maximisation problems as primal and minimisation problems as dual. They are of the form given in Table 1. We now give an explanation, from

TABLE 1. Primal and dual problems.

Maximise f_1	Minimise f_2
$\mathbf{e}^T \mathbf{x} = f_1$	$\mathbf{b}^T \mathbf{y} = f_2$
$A\mathbf{x} \leq \mathbf{b}$	$A^T \mathbf{y} \geq \mathbf{e}$
$\mathbf{x} \geq \mathbf{0}$	$\mathbf{y} \geq \mathbf{0}$

[2], as to why these problems are considered dual to each other. Let \mathbf{x} and \mathbf{y} be feasible solutions (satisfy all constraints) of the primal and dual problems respectively, then we have

that $\mathbf{y}^T A \geq \mathbf{e}^T$, so $\mathbf{y}^T A \mathbf{x} \geq \mathbf{e}^T \mathbf{x}$ because $\mathbf{x} \geq \mathbf{0}$. Similarly, $\mathbf{y}^T A \mathbf{x} \leq \mathbf{y}^T \mathbf{b}$, combining these inequalities gives $\mathbf{e}^T \mathbf{x} \leq \mathbf{y}^T \mathbf{b}$. We now note $\mathbf{b}^T \mathbf{y} = \mathbf{y}^T \mathbf{b}$, since \mathbf{b} and \mathbf{y} are column vectors. Thus it follows that $f_1 \leq f_2$. This gives us the following theorem:

THEOREM 2.3.1. *(The Weak Duality Theorem) If $\mathbf{x} \in \mathbb{R}^n$ is a solution to the primal and $\mathbf{y} \in \mathbb{R}^m$ is a solution to the dual problem, then*

$$\mathbf{e}^T \mathbf{x} \leq \mathbf{y}^T A \mathbf{x} \leq \mathbf{b}^T \mathbf{y}$$

as proved above. In fact this can be strengthened to give the following theorem from [5] and [2]:

THEOREM 2.3.2. *(The Strong Duality Theorem) If either the primal or dual problem has a finite optimal value then*

- i) *The optimal values coincide.*
- ii) *Optimal solutions to both problems exist.*

We now formulate the structural analysis of the pendulum (2) as a minimisation problem as an illustration of the general method that applies to an arbitrary DAE. Firstly we need the DAE's signature tableau as in §2.2:

$$\Sigma = \begin{array}{cccc} & x & y & \lambda & c_i \\ a & \left(\begin{array}{ccc} 2 & -\infty & 0 \end{array} \right) & 0 & & \\ b & \left(\begin{array}{ccc} -\infty & 2 & 0 \end{array} \right) & 0 & & \\ c & \left(\begin{array}{ccc} 0 & 0 & -\infty \end{array} \right) & 2 & & \\ d_j & 2 & 2 & 0 & \end{array} .$$

The objective function is $\text{Val}(\Sigma) = \sum_j d_j - \sum_i c_i$. The constraints come from $d_j - c_i \geq \sigma_{i,j} \quad \forall (i,j) \in S$, see (34). So in the pendulum we have the following constraints:

$$\begin{cases} d_1 - c_1 \geq 2, \\ d_3 - c_1 \geq 0, \\ d_2 - c_2 \geq 2, \\ d_3 - c_2 \geq 0, \\ d_1 - c_3 \geq 0, \\ d_2 - c_3 \geq 0, \end{cases}$$

which give the following matrix system:

$$\begin{pmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \geq \begin{pmatrix} 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

This gives us a problem of the form $A^T y \geq e$ which we now find the dual of, using Table 1. The primal problem is thus to maximise $2x_{1,1} + 0x_{1,3} + 2x_{2,2} + 0x_{2,3} + 0x_{3,1} + 0x_{3,2}$ —we've chosen this way of numbering the entries of \mathbf{x} for comparison with the $\sigma_{i,j}$ later. We have

the following constraints:

$$\left\{ \begin{array}{l} -x_{1,1} - x_{1,3} \leq -1, \\ -x_{2,2} - x_{2,3} \leq -1, \\ -x_{3,1} - x_{3,2} \leq -1, \\ x_{1,1} + x_{3,1} \leq 1, \\ x_{2,2} + x_{3,2} \leq 1, \\ x_{1,3} + x_{2,3} \leq 1, \end{array} \right.$$

which in matrix notation yields:

$$\begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,3} \\ x_{2,2} \\ x_{2,3} \\ x_{3,1} \\ x_{3,2} \end{pmatrix} \leq \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The vector \mathbf{b} (the RHS of the above inequality) corresponds to the coefficient vector in the objective function of the dual problem. This leads us to formulate the system as a linear assignment problem as given in Table 2 We now prove that these problems are in fact the

TABLE 2. Pendulum primal and dual problems.

Maximise f_1	Minimise f_2
$\sum_{(i,j) \in S} \sigma_{i,j} x_{i,j} = f_1$	$\sum_j d_j - \sum_i c_i = f_2 \quad \forall (i,j)$
$\sum_{i (i,j) \in S} x_{i,j} = 1 \quad \forall j$	$d_j - c_i \geq \sigma_{i,j} \quad \forall (i,j) \in S$
$\sum_{j (i,j) \in S} x_{i,j} = 1 \quad \forall i$	$d_j \geq 0 \quad \forall j$
$x_{i,j} \geq 0 \quad \forall (i,j)$	$c_i \geq 0 \quad \forall i$

dual of one another. We consider only the primal and using techniques outlined in [2] recover the dual. We begin by noticing that it's possible to write the equality-to-1 constraints in Table 2 on $x_{i,j}$ in several forms, as given in the Table 3. We choose the formulation given

TABLE 3. Cases for the primal formulation.

Case 0	Case 1	Case 2	Case 3
$\sum_{i (i,j) \in S} x_{i,j} = 1 \quad \forall j$	$\sum_{i (i,j) \in S} x_{i,j} \leq 1 \quad \forall j$	$\sum_{i (i,j) \in S} x_{i,j} \leq 1 \quad \forall j$	$\sum_{i (i,j) \in S} x_{i,j} \geq 1 \quad \forall j$
$\sum_{j (i,j) \in S} x_{i,j} = 1 \quad \forall i$	$\sum_{j (i,j) \in S} x_{i,j} = 1 \quad \forall i$	$\sum_{j (i,j) \in S} x_{i,j} \geq 1 \quad \forall i$	$\sum_{j (i,j) \in S} x_{i,j} = 1 \quad \forall i$
	Case 4	Case 5	Case 6
	$\sum_{i (i,j) \in S} x_{i,j} \geq 1 \quad \forall j$	$\sum_{i (i,j) \in S} x_{i,j} = 1 \quad \forall j$	$\sum_{i (i,j) \in S} x_{i,j} = 1 \quad \forall j$
	$\sum_{j (i,j) \in S} x_{i,j} \leq 1 \quad \forall i$	$\sum_{j (i,j) \in S} x_{i,j} \leq 1 \quad \forall i$	$\sum_{j (i,j) \in S} x_{i,j} \geq 1 \quad \forall i$

by case 3:

$$(21) \quad \text{Maximise} \quad \sum_{(i,j) \in S} \sigma_{i,j} x_{i,j},$$

subject to:

$$(22) \quad - \sum_{i|(i,j) \in S} x_{i,j} \leq -1,$$

$$(23) \quad \sum_{j|(i,j) \in S} x_{i,j} = 1,$$

$$(24) \quad x_{i,j} \geq 0 \quad (i,j) \in S.$$

If any $\sigma_{i,j} = -\infty$, and the corresponding $x_{i,j} \neq 0$, then the objective function is always negative infinite and thus no feasible solution will exist. Hence, $\sigma_{i,j}$ are only considered when $(i,j) \in S$. We also note that $\mathbf{b} = (-1, \dots, 1)^T$ with a number of components equal to the number of $\sigma_{i,j} \in S$ with the -1 's coming from the RHS of (22) and the 1 's coming from the RHS of (23). Also, we note that $\mathbf{e} = (\sigma_{i,j} \mid (i,j) \in S)^T$ with entries ordered in rows and then columns, so that we read off values in Σ from left to right row by row. We now notice that this choice of constraints will give us a matrix, A , that has a -1 in it for every $\sigma_{i,j} \in S$ corresponding to (22) and a 1 in it for every $\sigma_{i,j} \in S$ corresponding to (23), so in each column there are exactly two entries, one -1 and one 1 . Hence, A^T will have exactly one 1 and one -1 in each row. Hence $A^T \mathbf{y} \geq \mathbf{e}$ will give equations of the form (something from the second half of \mathbf{y}) $-$ (something from the first half of \mathbf{y}) $\geq (\sigma_{i,j} \mid (i,j) \in S)$, which

are exactly the equations generated by (34) as we predicted, thus our \mathbf{y} is $(c_i\text{'s}, d_j\text{'s})^T$. From [2] we have ‘If there are any equality constraints in the primal problem the corresponding dual variables are not sign restricted’. Hence the different ways to formulate the primal problem, shown in Table 3 will give several possible choices of non zero variables in the dual. It is therefore possible to get unconstrained c_i values. However, if we had all $c_i < 0$ then we would try to find invalid canonical offsets, which is why we reformulated to give us unconstrained d_j ’s using Case 3. Thus, Case 3 gives us $d_j - c_i \geq \sigma_{i,j}$ on a transversal (since we restricted ourselves to S) and $c_i \geq 0$ which give us one hidden constraint, namely $d_j \geq 0$, since $\sigma_{i,j} \geq 0 \quad \forall (i, j) \in S$, showing the problems in Table 2 are dual to one another.

It is always possible to permute Σ such that a HVT is on the main diagonal by permuting rows and columns and provide an algorithm for finding the offsets from an $n \times n$ signature matrix, with a HVT on the main diagonal (Note: we use Python-style in all algorithms in this thesis, i.e. indentation is used to show the scope of a statement):

Algorithm 1 Algorithm to Find Offset Vectors

```

1: for  $j = 1 : n$ 
2:    $d_j = \max_{i=1:n} \sigma_{i,j}$ 
3: for  $i = 1 : n$ 
4:    $c_i = 0$ 
5: loop until  $c_i = c_i^{old}$  for all  $i$ 
6:   for  $i = 1 : n$ 
7:      $c_i^{old} = c_i$ 
8:   for  $i = 1 : n$ 
9:      $c_i = d_i - \sigma_{i,i}$ 
10:  for  $j = 1 : n$ 
11:     $d_j = \max_{i=1:n} (\sigma_{i,j} + c_i)$ 
12: return  $\mathbf{c}, \mathbf{d}$ 

```

We now prove that this algorithm terminates if and only if a HVT is on the main diagonal. Firstly suppose it terminates. Then $c_i = d_i - \sigma_{i,i}$ and therefore $\sigma_{i,i} = d_i - c_i$, which from (34) means there is a HVT on the main diagonal.

Conversely, if there is a HVT on the main diagonal then the algorithm will terminate, since each d_j will either increase or remain constant with each iteration of the loop. If they remain constant over all iterations then the algorithm will terminate, since each c_j will be fixed. If however during the loop we raise a c_i in the second **for** loop then the equations will satisfy the condition of having equality on a HVT, but may not be greater than $\sigma_{i,j}$ everywhere. The third **for** condition in the loop will increase the d_j so that (34) is satisfied, although possibly without equality on the HVT. This process will then repeat. Formally, if we consider each loop iteration as a function, at each stage we input a vector \mathbf{c} and get a new one \mathbf{c}' . So that we have a map:

$$(25) \quad \phi: \mathbf{c} \mapsto \mathbf{c}',$$

which is a weakly monotone function. That is, if we have two n -vectors \mathbf{c} and \mathbf{c}^* with $c_i \leq c_i^* \quad \forall i \in \{1, \dots, n\}$ then $\phi(\mathbf{c}) \leq \phi(\mathbf{c}^*)$.

PROPOSITION 2.3.3. *Index the \mathbf{c} found at each call of ϕ in the algorithm by \mathbf{c}^n , the algorithm yields an element-wise weakly monotone sequence $\mathbf{c}^0 \leq \mathbf{c}^1 \leq \dots \leq \mathbf{c}^n$.*

PROOF. We start by setting $\mathbf{c}^0 = \mathbf{0}$ with lines 4-6 of the algorithm. We have:

$$\phi(\mathbf{c}^0) = \mathbf{c}^1, \phi(\mathbf{c}^1) = \mathbf{c}^2, \dots,$$

and since $\mathbf{c}^0 = \mathbf{0}$ we have:

$$\mathbf{c}^1 = \phi(\mathbf{c}^0) = \phi(\mathbf{0}) \geq \mathbf{0} = \mathbf{c}^0,$$

due to the algorithm never letting an element of \mathbf{c} be negative. If we repeat this we get:

$$\mathbf{c}^2 = \phi(\mathbf{c}^1) \geq \phi(\mathbf{c}^0) = \mathbf{c}^1,$$

and thus the argument follows by induction. □

Since our algorithm is weakly monotone and starts at the zero vector it must terminate at the canonical offsets if there exist valid offsets. Take a valid offset \mathbf{c}^* , so $\phi(\mathbf{c}^*) = \mathbf{c}^*$ and $\mathbf{c}^* \geq \mathbf{0}$, since $\mathbf{c}^1 = \phi(\mathbf{c}^0) \leq \phi(\mathbf{c}^*) = \mathbf{c}^*$ we have $\mathbf{c}^1 \leq \mathbf{c}^*$ and hence by induction and choice of \mathbf{c}^n found by the algorithm has the property $\mathbf{c}^n \leq \mathbf{c}^*$ and we have a bound on our offsets. There are a finite number of \mathbf{c}^n , so the algorithm must terminate at some point, say $\mathbf{c}^{**} \leq \mathbf{c}^*$. Since \mathbf{c}^* is an arbitrary valid offset we have termination to the element-wise smallest offsets, clearly these offsets are normalised, consider a \mathbf{c}^* with all entries larger than m , then $\mathbf{c}^* - \mathbf{m}$ is a normalised offset - but we can always choose a element-wise smaller valid offset to be \mathbf{c}^* and hence \mathbf{c}^{**} must be canonical. For a full description of valid, normalised and canonical offsets see Definitions 2.5.3, 2.5.4 and 2.5.5 although for now it is sufficient to have an understanding from (16).

THEOREM 2.3.4 (Existence of canonical offsets). *There exists an element-wise minimal pair of offset vectors, called the canonical offsets, provided there exists valid offsets.*

We proved this above, but we provide a second proof to help the reader understand the algorithm.

PROOF. If there exist valid offset vectors there will exist normalised offset vectors, found by subtracting unit vectors from \mathbf{c} and \mathbf{d} repeatedly until we have normalised offsets. Hence we can use existence of normalised offsets. Given two normalised sets of offset vectors \mathbf{c}' , \mathbf{d}' and \mathbf{c}'' , \mathbf{d}'' we take the element-wise minimum and produce element-wise smaller offset vectors \mathbf{c} and \mathbf{d} . These offsets satisfy the following inequalities, with equality on a HVT:

$$d'_j - c'_i \geq \sigma_{i,j},$$

$$d''_j - c''_i \geq \sigma_{i,j}.$$

It is easiest for explanation to consider taking the first set of offsets and amending them with any lower values in the second set. Clearly if a $c''_i \leq c'_i$ then choosing it will not change the inequality, although it might break equality on a HVT. If it broke equality on the HVT at position, say $\sigma_{i,j}$ then it must be true that $d''_j \leq d'_j$, since there is still equality on a HVT

in the second set of equations. If reducing a non HVT d'_j to a d''_j makes an inequality false in some position $\sigma_{i,j}$ then it must be true that the $c''_i \leq c'_i$, since that same inequality holds in the second set of equations. Hence taking the element-wise minimum of two normalised offset vectors produces a new normalised offset vector and there must exist canonical offsets if there exist valid offsets. \square

Note, since finding a HVT is dual to finding valid offsets, from Theorem 2.3.1 we have that our algorithm terminates provided there is a HVT on the main diagonal.

2.4. Signature Method By Example

Recall the simple pendulum DAE (2). We look for a solution by means of a power series, specifically a Taylor series. That is, x is to be of the form $\sum_{n=0}^{\infty} x_n t^n$, and y, λ similarly, where without loss we take $t = 0$ as the point to be expanded about. If we expand each variable as a power series and then substitute into equation (2) we obtain:

$$(26) \quad \begin{cases} a(t) = 1.2x_2 + 2.3x_3t + \dots + (x_0 + x_1t + \dots)(\lambda_0 + \lambda_1t + \dots) & = 0, \\ b(t) = 1.2y_2 + 2.3y_3t + \dots + (y_0 + y_1t + \dots)(\lambda_0 + \lambda_1t + \dots) - G & = 0, \\ c(t) = (x_0 + x_1t + \dots)(x_0 + x_1t + \dots) + (y_0 + y_1t + \dots)(y_0 + y_1t + \dots) - L^2 & = 0. \end{cases}$$

Each of a, b, c can also be represented as a power series where coefficients a_n, b_n, c_n must vanish for all n . Thus if we simplify (26) we can equate coefficients and get constraint equations on the variables. Expanding brackets yields:

$$(27) \quad \begin{cases} a(t) = 1.2x_2 + 2.3x_3t + \dots + x_0 + \lambda_0 + (x_0\lambda_1 + x_1\lambda_0)t + (x_0\lambda_2 + x_1\lambda_1 + x_2\lambda_0)t^2 + \dots & = 0, \\ b(t) = 1.2y_2 + 2.3y_3t + \dots + y_0 + \lambda_0 + (y_0\lambda_1 + y_1\lambda_0)t + (y_0\lambda_2 + y_1\lambda_1 + y_2\lambda_0)t^2 + \dots - G & = 0, \\ c(t) = x_0^2 + 2x_0x_1t + (2x_0x_2 + x_1^2)t^2 + \dots + y_0^2 + 2y_0y_1t + (2y_0y_2 + y_1^2)t^2 + \dots - L^2 & = 0, \end{cases}$$

TABLE 4. Power series coefficient equations for the simple pendulum.

Stage -2	Stage -1	Stage 0	Further Stages
		$1.2x_2 + x_0\lambda_0 = a_0$...
		$1.2y_2 + y_0\lambda_0 - G = b_0$...
$x_0^2 + y_0^2 - L^2 = c_0$	$2x_0x_1 + 2y_0y_1 = c_1$	$2x_0x_2 + x_1^2 + 2y_0y_2 + y_1^2 = c_2$...

collecting terms

$$(28) \quad \begin{cases} a(t) = (1.2x_2 + x_0\lambda_0) + (2.3x_3 + x_0\lambda_1 + x_1\lambda_0)t + (3.4x_4 + x_0\lambda_2 + x_1\lambda_1 + x_2\lambda_0)t^2 + \dots & = 0, \\ b(t) = (1.2y_2 + y_0\lambda_0) + (2.3y_3 + y_0\lambda_1 + y_1\lambda_0)t + (3.4y_4 + y_0\lambda_2 + y_1\lambda_1 + y_2\lambda_0)t^2 + \dots & = 0, \\ c(t) = (x_0^2 + y_0^2 - L^2) + (2x_0x_1 + 2y_0y_1)t + (2x_0x_2 + x_1^2 + 2y_0y_2 + y_1^2)t^2 + \dots & = 0, \end{cases}$$

and equating coefficients yields Table 4. We have arranged the coefficients in Table 4 by noting that only c_0 involves x_0 , y_0 and no higher order terms (h.o.t.s) so we can solve it for x_0 or y_0 by giving one as a trial value. Now we have a similar situation with c_1 and can use it to solve for x_1 , y_1 giving one as a trial value and using already computed x_0 and y_0 from c_0 . Consider the system given by a_0 , b_0 and c_2 , we now have equations for x_2 , y_2 and λ_0 and no other h.o.t.s, so can solve this system, similarly we can use a_1 , b_1 and c_3 to get x_3 , y_3 and λ_1 and so on. We have indexed the stages starting from -2 because we have to solve two systems before getting a 3×3 system. We write the equations at stage 0 in matrix notation:

$$\mathbf{0} = \begin{pmatrix} 1.2 & 0 & x_0 \\ 0 & 1.2 & y_0 \\ 2x_0 & 2y_0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ y_2 \\ \lambda_0 \end{pmatrix} + \begin{pmatrix} 0 \\ -G \\ x_1^2 + y_1^2 \end{pmatrix},$$

and stage 1

$$\mathbf{0} = \begin{pmatrix} 2.3 & 0 & x_0 \\ 0 & 2.3 & y_0 \\ 2x_0 & 2y_0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ y_3 \\ \lambda_1 \end{pmatrix} + \begin{pmatrix} l.o.ts \\ l.o.ts \\ l.o.ts \end{pmatrix},$$

where *l.o.ts* means terms involving lower order coefficients. The positions in the first two diagonal entries appear to have the form $(n+1).(n+2)$, in general we have the following

formula for the coefficients:

$$(29) \quad \left\{ \begin{array}{l} a_n = (n+1)(n+2)x_{n+2} + \sum_0^n x_i \lambda_{n-i}, \\ b_n = (n+1)(n+2)y_{n+2} + \sum_0^n y_i \lambda_{n-i}, \\ c_n = \sum_0^n x_i x_{n-i} + \sum_0^n y_i y_{n-i}, \end{array} \right.$$

so that at each stage we solve

$$\left\{ \begin{array}{l} a_n = (n+2)(n+1)x_{n+2} + x_0 \lambda_n + l.o.ts, \\ b_n = (n+2)(n+1)y_{n+2} + y_0 \lambda_n + l.o.ts, \\ c_{n+2} = 2x_{n+2}x_0 + 2y_{n+2}y_0 + l.o.ts. \end{array} \right.$$

We would like to have the same matrix multiplying our unknowns being solved for at each 3×3 stage so we make a change of unknowns so that $A_k = a_k k!$ (and similarly for other variables), which gives us the following equations at each stage:

$$(30) \quad \left. \begin{array}{l} \frac{A_n}{n!} = \frac{X_{n+2}}{n!} + \frac{Y_0 \Lambda_n}{n!} + l.o.ts \\ \frac{B_n}{n!} = \frac{Y_{n+2}}{n!} + \frac{Y_0 \Lambda_n}{n!} + l.o.ts \\ \frac{C_{n+2}}{(n+2)!} = \frac{2X_{n+2}X_0}{(n+2)!} + \frac{2Y_{n+2}Y_0}{(n+2)!} + l.o.ts \end{array} \right\}.$$

which gives us the following system in matrix form at stage n

$$\begin{pmatrix} 1 & 0 & X_0 \\ 0 & 1 & Y_0 \\ 2X_0 & 2Y_0 & 0 \end{pmatrix} \begin{pmatrix} X_{n+2} \\ Y_{n+2} \\ \Lambda_n \end{pmatrix} + \begin{pmatrix} l.o.ts \\ l.o.ts \\ l.o.ts \end{pmatrix} = 0.$$

Since the a_k 's are power series coefficients the A_k 's are derivatives (at $t = 0$), our solution method is to find:

$$(31) \quad x(t_0 + h) = \sum_{n=0}^{N_x} \frac{x^{(n)}(t_0)}{n!} h^n,$$

where h is a given step size and N_x is the final order of Taylor series used for variable x . For illustration purposes we present a solution to simple pendulum done via this method in Matlab with: $(x(0), y(0), \dot{x}(0), \dot{y}(0)) = (6, 8, -.8, .6)$, $L = 10$, $G = 9.81$ with a step size of .001 and a Taylor series of order 18 in Figure 1 and see it looks as expected. For an error heuristic we see how well energy is preserved in Figure 2. The Matlab script is presented in Appendix A.

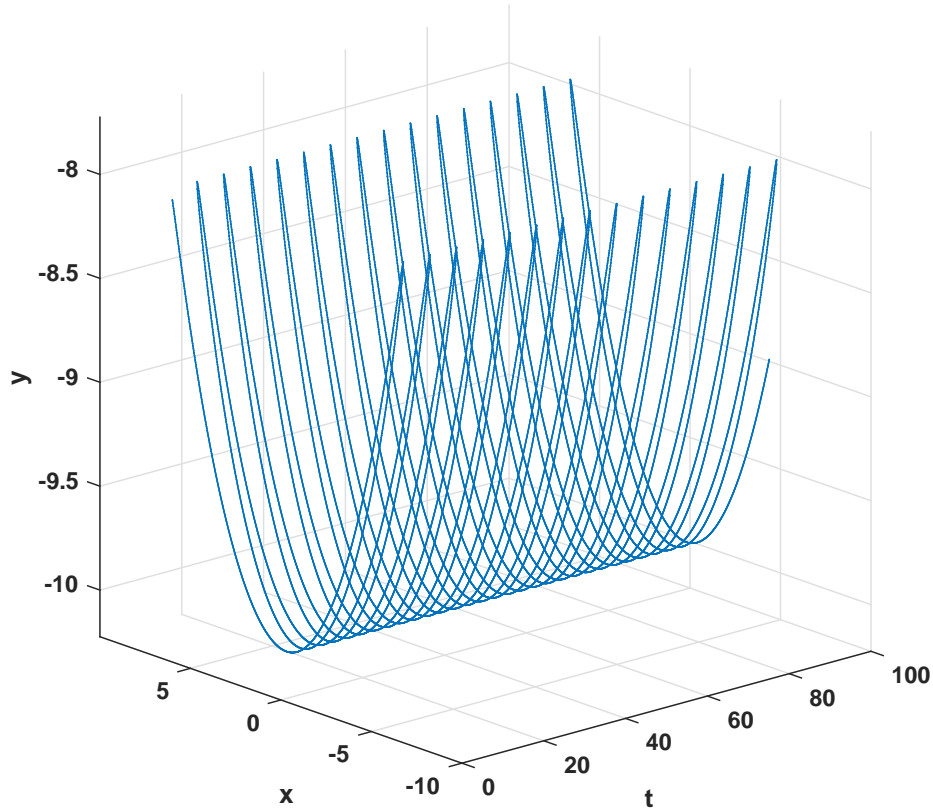


FIGURE 1. A solution to simple pendulum DAE via a Taylor series approach.

In general we will consider the idea of a consistent manifold. That is, a manifold specified by all the equations involving initial variables. For the pendulum our consistent manifold could live in (x, \dot{x}, y, \dot{y}) space and be given by $(c, \dot{c}) = 0$ but could equally live in $(x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, \lambda)$ and be $(a, b, c, \dot{c}, \ddot{c}) = 0$ and so on. We find all required Taylor coefficients

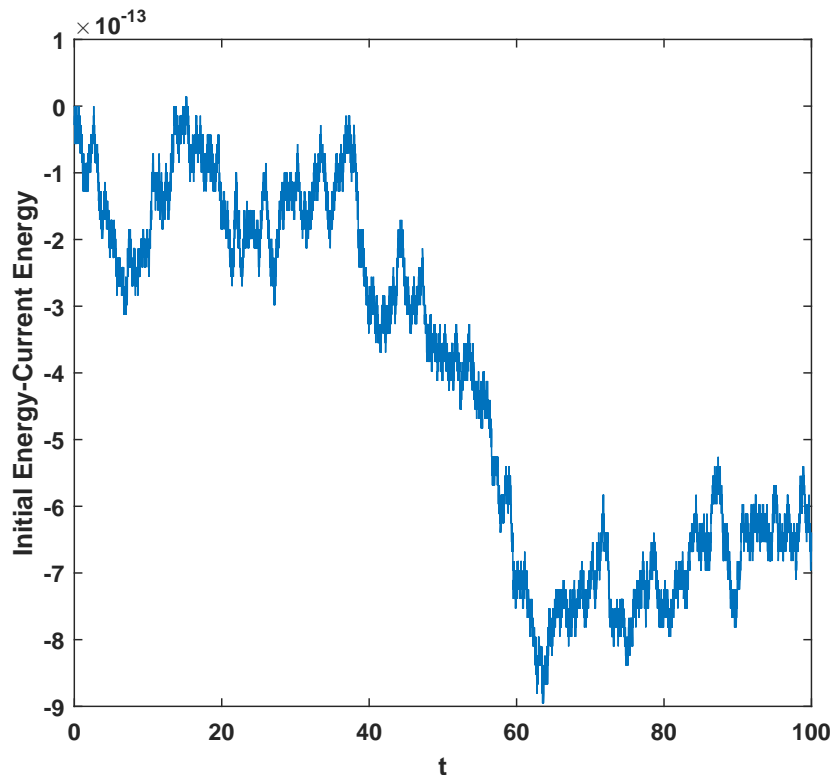


FIGURE 2. Change in energy for simple pendulum DAE via a Taylor series approach.

at t_0 for the initial variables, which will give us a point off the consistent manifold (specified by $c_0 = 0$ and $c_1 = 0$ for the pendulum). We choose to project our solution onto this manifold by dropping a perpendicular, for example using a Gauss-Newton method, and then starting again at time $t_0 + h$. For the pendulum we find Taylor coefficients for x and y up to some order $N + 1$ and then differentiate them to get N coefficients for \dot{x} and \dot{y} . We then project this solution onto the consistent manifold and repeat the process for our new time eventually reaching the desired end time and reading the values of x , y and λ . It is worth noting that the consistent manifold will always lie in the space of all initial conditions, but can be considered as ‘living in’ some larger space, given by any number of coordinates.

2.5. Signature Method - Basics

Given that we now have an idea as to how a Taylor solution scheme should look for an arbitrary DAE we proceed to give the signature matrix method. Firstly we need to define what is meant by a *signature matrix* for a DAE of form (1):

DEFINITION 2.5.1 (Signature Matrix). The signature matrix, Σ , for a DAE with a set of equations f_i and variables x_j where $i, j \in \{1, \dots, n\}$ has entries :

$$(32) \quad \sigma_{i,j} = \begin{cases} \text{Order of highest derivative of } x_j \text{ in } f_i & \text{if } x_j \text{ occurs in } f_i, \\ -\infty & \text{otherwise.} \end{cases}$$

The signature matrix for the simple pendulum (2) is therefore:

$$(33) \quad \Sigma = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} x & y & \lambda \\ \begin{pmatrix} 2 & -\infty & 0 \\ -\infty & 2 & 0 \\ 0 & 0 & -\infty \end{pmatrix} \end{array},$$

where we write corresponding equation and variable names around the matrix.

DEFINITION 2.5.2 (Transversals). A transversal for a square matrix is any set of matrix positions (i, j) so that an entry in each row and column is chosen only once. A transversal for an $m \times n$ matrix, with $m \geq n$ is a set of n positions (i, j) so that $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$ where no i or j is repeated, similarly for the case when $m \leq n$ we look for m matrix positions. The value of a transversal is the sum of all entries in the matrix, say $a_{i,j}$, such that (i, j) belongs to the transversal.

It is useful for our purposes to consider the, non-unique, highest value transversal, HVT. Consider the signature matrix for the pendulum

$$\Sigma = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} x & y & \lambda \\ \begin{pmatrix} 2^\bullet & -\infty & 0^\circ \\ -\infty & 2^\circ & 0^\bullet \\ 0^\circ & 0^\bullet & -\infty \end{pmatrix} \end{array}.$$

This has two HVTs, marked by \bullet , $\{(1, 1), (2, 3), (3, 2)\}$, and \circ , $\{(1, 3), (2, 2), (3, 1)\}$, in the signature matrix above.

DEFINITION 2.5.3 (Valid Offsets). We call positive valued n -vectors \mathbf{c} and \mathbf{d} satisfying

$$(34) \quad \sigma_{i,j} \leq d_j - c_i \quad \text{with equality on any HVT}$$

valid offset vectors.

DEFINITION 2.5.4 (Normalised Offsets). We call valid offsets \mathbf{c} and \mathbf{d} normalised if $\min_i c_i = 0$.

DEFINITION 2.5.5 (Canonical Offsets). We call the element-wise minimum normalised offset vectors the canonical offsets.

Theorem 2.3.4 shows us that given valid offsets the canonical offsets exist. Historically canonical offsets have always been used for the structural analysis of a DAE (since they represent doing less differentiations), we will show in Chapter 4 that it is sometimes preferable to drop this restriction. The offsets represent the way in which solving for some variables must be ‘pushed back’ a stage in the Taylor series the solution of a DAE, as was demonstrated in Table 4.

DEFINITION 2.5.6 (Degrees of Freedom). The number of degrees of freedom, DOF, of a DAE is the number of initial conditions required to specify a unique solution.

For example, the pendulum has 2 DOF; the first might be imposed by specifying either x or y and the second by specifying either \dot{x} or \dot{y} , the position and velocity of the pendulum respectively. The degrees of freedom are given by

$$(35) \quad \sum_j d_j - \sum_i c_i.$$

We delay the proof of this because we need notation not yet introduced.

DEFINITION 2.5.7. (The function $\text{Val}(\Sigma)$) Consider a HVT, T , then we say $\text{Val}(\Sigma) = \sum_{(i,j) \in T} \sigma_{i,j}$.

LEMMA 2.5.8. $\text{DOF} = \text{Val}(\Sigma)$ iff $d_j - c_i = \sigma_{i,j}$ for all (i, j) in a HVT.

PROOF. Consider an arbitrary HVT T_1 , then

$$\sum_{(i,j) \in T_1} \sigma_{i,j} = \text{DOF} = \sum_j d_j - \sum_i c_i,$$

using (35) and hence,

$$0 = \sum_{(i,j) \in T_1} d_j - c_i - \sigma_{i,j}.$$

From (34) each individual term in the summation is non-negative and hence 0. Meaning $\sigma_{i,j} = d_j - c_i$ for all $\sigma_{i,j}$ on the HVT T_1 , since T_1 is arbitrary we have equality on any HVT. Conversely if we have equality on any HVT in (34) then summing across a HVT T_2 we have

$$\sum_{(i,j) \in T_2} \sigma_{i,j} = \sum_j d_j - \sum_i c_i = \text{DOF}$$

using (35). □

2.6. Standard Solution Scheme

We now seek to develop a way of solving our DAE given a signature matrix. From (18, 19) it is clear that at stage 0 we have exactly derivatives up to order c_i of equations f_i , for example consider $c_i=l$, then at stage l we use f_i , at stage $l+1$ use f'_i etc... until at stage 0 we are using $f_i^{(l)}$. Similarly we solve derivatives up to order d_j of variables x_j at stage 0, which is what we meant earlier by ‘pushing back’ the solution of some variables.

DEFINITION 2.6.1 (System Jacobian). The $n \times n$ Jacobian matrix used for finding Taylor coefficients in SA is called the System Jacobian, denoted by \mathbf{J} .

$$\mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j^{(d_j - c_i)}}.$$

Hence, recalling the signature matrix (33), for the simple pendulum we have the following System Jacobian:

$$\mathbf{J} = \begin{array}{cccc} & \ddot{x} & \ddot{y} & \lambda & c_i \\ a & \left(\begin{array}{ccc} a_{\ddot{x}} & 0 & a_{\lambda} \end{array} \right) & & 0 \\ b & \left(\begin{array}{ccc} 0 & b_{\ddot{y}} & b_{\lambda} \end{array} \right) & & 0 \\ \ddot{c} & \left(\begin{array}{ccc} \ddot{c}_x & \ddot{c}_y & 0 \end{array} \right) & & 2 \\ d_j & 2 & 2 & 0 \end{array}$$

Since the c_i 's belong to equations they are written around the matrix in their corresponding row, similarly the d_j are written in their corresponding column, we also write equations being used and variables being solved for in their respective rows and columns—we call such a representation the *matrix tableau*, above we have a *Jacobian tableau*, similarly such a representation for Σ (we omit derivatives of equations and variables in this case) is termed the *signature tableau*. To proceed in our analysis we need the following lemma, from [20] (and [10]):

LEMMA 2.6.2 (Griewank's Lemma). *Suppose $f(t, x_1, x_2, \dots, x_n)$ is a smooth function that contains derivatives of x_j not exceeding order m_j . Then the $(m_j + 1)^{st}$ derivative of x_j occurs linearly in $\dot{f} = \frac{df}{dt}$, and*

$$\frac{\partial f'}{\partial x_j^{(m_j+1)}} = \frac{\partial f}{\partial x_j^{(m_j)}}.$$

The first assertion of this lemma follows by way of the chain rule when proving the second assertion, which we now prove.

PROOF. Take a DAE:

$$u = f(t; \left(x_j^{(\ell)} \right)_{(j,\ell) \in J}) = 0,$$

then by the chain rule

$$(36) \quad \dot{u} = f_{x_1}(t; \left(x_j^{(\ell)} \right)) \dot{x}_1 + \dots + f_{x_1^{(l_1)}}(t; \left(x_j^{(\ell)} \right)) x_1^{(l_1+1)} + \dots$$

where $f_{x_j} = \frac{\partial f}{\partial x_j}$ and $+\dots$ is contributions from x_2 onwards. Writing (36) as a sum over J yields:

$$\dot{u} = \sum_{(j,l) \in J} f_{x_j^{(\ell)}}(t; (x_j^{(\ell)})) x_j^{(\ell+1)}.$$

Therefore, letting l_{j^*} be the highest order derivative (HOD) of variable x_{j^*}

$$\frac{\partial \dot{u}}{\partial x_{j^*}^{(l_{j^*}+1)}} = \frac{\partial}{\partial x_{j^*}^{(l_{j^*}+1)}} \left(\sum_{(j,l) \in J} f_{x_j^{(\ell)}}(t; (x_j^{(\ell)})) x_j^{(\ell+1)} \right) = f_{x_{j^*}^{(l_{j^*})}},$$

because $x_{j^*}^{(l_{j^*}+1)}$ does not appear in any other terms of the series. \square

Following this lemma we can give a more informative formula for the System Jacobian

J:

$$(37) \quad \mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j^{(d_j - c_i)}} = \begin{cases} \frac{\partial f_i}{\partial x_j^{(\sigma_{i,j})}} & \text{if } d_j - c_i = \sigma_{i,j}, \\ 0 & \text{elsewhere.} \end{cases}$$

This final equality comes from the condition that $d_j - c_i = \sigma_{i,j}$ on a HVT and that if $d_j - c_i > \sigma_{i,j}$ then we will be differentiating by a variable that doesn't occur in the function. This ensures we only ever differentiate equations by variables that occur in them and thus have a well defined System Jacobian. If we have a HVT then (34) guarantees that there will be a structural non-zero, at least one in every row and column, in the Jacobian and hence ensures it's structurally non-singular and the problem is thus solvable provided we don't get numerical non-singularity along the solution process. We wish to use this System Jacobian to develop a stage-wise solution process. We illustrate this with the simple pendulum, for ease we will rename the variables and equations as follows: $a = f_1$, $b = f_2$, $c = f_3$, $x = x_1$, $y = x_2$ and $\lambda = x_3$ giving us the functions and variables at each stage listed in Table 5. At any stage k we use equations

$$(38) \quad f_i^{(k+c_i)} \quad \forall i \text{ such that } k + c_i \geq 0$$

TABLE 5. Stage for the simple pendulum

k	Equations being used	Variables being found
-2	f_3	x_1, x_2
-1	\dot{f}_3	\dot{x}_1, \dot{x}_2
0	f_1, f_2, \ddot{f}_3	$\ddot{x}_1, \ddot{x}_2, x_3$
1	$\dot{f}_1, \dot{f}_2, f_3^{(3)}$	$x_1^{(3)}, x_2^{(3)}, \dot{x}_3$
...

to solve for variables

$$(39) \quad x_j^{(k+d_j)} \quad \forall j \text{ such that } k + d_j \geq 0.$$

Thus at any stage $k < 0$ we are using derivatives found either at stage k or at a previous stage, due to the incremental nature of the formula. The stage where we first get an $n \times n$ matrix for the n equations and variables is stage 0. We use k as our stage number, and begin at stage $k_{min} = -\max_j d_j$. Our stages are therefore numbered: $k_{min}, k_{min} + 1, \dots, 0, 1, \dots, k_{max}$, where k_{max} is some arbitrary Taylor series order being used.

Recall the formula for the System Jacobian (37). We show this is the natural Jacobian when we use these equations and variables at each stage of our method. To do this we will consider the System Jacobian using equations (38, 39) which tell us that at step 0 we are using equations $f_i^{(c_i)}$ to solve for variables $x_j^{(d_j)}$, if we form the matrix with entries

$$(40) \quad \frac{\partial f_i^{(c_i)}(t; (x_j^{(\ell)}))}{\partial x_j^{(d_j)}}$$

and use the fact $d_j \geq c_i$ and d_j must be at least as large as the maximum derivative of x_j we can apply Lemma 2.6.2 to obtain:

$$\frac{\partial f_i^{(c_i)}(t; (x_j^{(\ell)}))}{\partial x_j^{(d_j)}} = \frac{\partial f_i(t; (x_j^{(\ell)}))}{\partial x_j^{(d_j - c_i)}},$$

whenever $d_j - c_i$ is the highest order derivative of variable x_j in f_i and 0 otherwise. Hence, from (37) we have that (40) is just the (i, j) th entry of the System Jacobian evaluated at a point. From (40) we have a matrix of the coefficients of $x_j^{(d_j)}$ in $f_i^{(c_i)}$ by Lemma 2.6.2. For

example, if $c_i = 1$ by the chain rule we have:

$$(41) \quad \frac{df_i}{dt} = \sum_{(j,a) \in J} f_{i,x_j^{(a)}} x_j^{(a+1)}.$$

Here $f_{i,x_j^{(a)}}$ is the partial derivative of f_i with respect to variable $x_j^{(a)}$. Our solution method from equations (18, 19) means that at the next stage we solve for the derivative of the variable previously solved for using the derivatives of the equations previously used. This is due to us considering the equations at each stage as our new ‘starting equations’ so instead of considering \ddot{f} as the second differential of f we consider it as the first differential of \dot{f} , which is why considering $c_i = 1$ is sufficient.

We will now use the notation \mathbf{J}_k to denote the System Jacobian at stage k (where $k < 0$), that is the System Jacobian found by considering only equations and variables being used at stage k and changing d_j to $d_j + k$ and c_i to $c_i + k$ in the formula above. To proceed we need the following definition:

DEFINITION 2.6.3 (Submatrix). Given two matrices A and B , the matrix B is a submatrix of A if it can be obtained by deleting rows and columns of A or possibly reordering rows and columns of A .

Since the above definition does not take order in to account given a matrix:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ b_{1,1} & b_{1,2} & b_{1,3} \\ c_{1,1} & c_{1,2} & c_{1,3} \end{pmatrix}$$

the following two matrices are both submatrices of A :

$$B_1 = \begin{pmatrix} a_{1,1} & a_{1,2} \\ b_{1,1} & b_{1,2} \end{pmatrix}, \quad B_2 = \begin{pmatrix} b_{1,1} \\ a_{1,1} \end{pmatrix}.$$

DEFINITION 2.6.4 (Leading Submatrix). Given two matrices A and B , the matrix B is a leading submatrix of A if it comprises the first p rows and first q columns of A for some choice of p and q .

From Lemma 2.6.2 it can be seen that this leads to us having a nested form for the Jacobian at each stage, where \mathbf{J}_k is a leading submatrix of $\mathbf{J}_K \forall K \geq k$, assuming rows and columns are ordered in descending order of offsets. From (38, 39) it can be seen that after ordering rows and columns \mathbf{J}_k will start in the upper left corner of \mathbf{J}_K . Formally we have the following lemma

LEMMA 2.6.5. *If we order rows and columns of \mathbf{J} in descending order of offset values then \mathbf{J}_k is a leading submatrix of \mathbf{J} for all $k < 0$.*

PROOF. By definition the System Jacobian at negative stages is:

$$\mathbf{J}_{k_{i,j}} = \frac{\partial f_i^{(c_i+k)}}{\partial x_j^{(d_j+k)}} \quad \text{if } c_i + k \geq 0 \text{ and } d_j + k \geq 0$$

if $d_j - c_i > \sigma_{i,j}$ then f_i has no terms of order $x_j^{(d_j-c_i)}$, so by differentiating $c_i + k$ times using Lemma 2.6.2 we see $f_i^{(c_i+k)}$ must have no terms of order $x_j^{d_j+k}$ and $\mathbf{J}_{k_{i,j}} = 0$. Alternatively consider the case $d_j - c_i = \sigma_{i,j}$, by definition $\sigma_{i,j}$ is the H.O.D. of x_j in f_i , so by Lemma 2.6.2 we have:

$$\mathbf{J}_{k_{i,j}} = \frac{\partial f_i^{(c_i+k)}}{\partial x_j^{(d_j+k)}} = \frac{\partial f_i^{(c_i)}}{\partial x_j^{(d_j-c_i)}},$$

which is by definition \mathbf{J} . □

From above it is clear that the System Jacobian, when ordered with respect to descending offsets, at each negative step is a submatrix of \mathbf{J} , and $\mathbf{J}_k = \mathbf{J}_0 \quad \forall k \geq 0$, thus we write \mathbf{J} for any \mathbf{J}_k , $k \geq 0$.

For example with the pendulum there are two System Jacobians throughout the solution process, the smaller at negative stages and the larger at non-negative stages. For convenience we order the signature tableau with descending offsets and show the submatrix relevant for

negative stages:

$$\Sigma = \begin{array}{cccc} & x & y & \lambda & c_i \\ c & \left(\begin{array}{cc|c} 0 & 0 & -\infty \end{array} \right) & & & 2 \\ a & \left(\begin{array}{cc|c} 2 & -\infty & 0 \end{array} \right) & & & 0 \\ b & \left(\begin{array}{cc|c} -\infty & 2 & 0 \end{array} \right) & & & 0 \\ d_j & 2 & 2 & 0 & \end{array}$$

We have two Jacobians, separated by a line below, the smaller occurs at stages $k = -2, -1$ and the larger for all $k \geq 0$:

$$\mathbf{J} = \begin{pmatrix} \left(\begin{array}{cc|c} c_x & c_y & 0 \end{array} \right) \\ a_{\ddot{x}} & 0 & a_\lambda \\ 0 & b_{ij} & b_\lambda \end{pmatrix}.$$

DEFINITION 2.6.6 (Structural Full Row Rank). We say a numerical matrix of size $m \times n$, where $m \leq n$, has structural full row rank if and only if it has a transversal that contains no 0 elements.

DEFINITION 2.6.7 (Structural Full Row Rank of Σ). We say a signature matrix Σ of size $n \times n$, has structural full row rank iff S has a transversal.

LEMMA 2.6.8. *If \mathbf{J} has a transversal then the System Jacobian at a negative stage is of structural full row rank.*

PROOF. To the right of any nested matrix all values will be $-\infty$ in Σ and 0 in \mathbf{J} , due to $d_j - c_i < 0$ in those places when ordered with descending offsets. Hence, if there's a valid transversal for \mathbf{J} it must have m values in an $m \times n$ submatrix found from a negative step. Thus, each negative step System Jacobian of size $m \times n$ has a transversal of size m and is thus of structural full row rank. \square

We make a distinction between matrices which are structurally singular and genuinely, that is numerically, singular.

DEFINITION 2.6.9 (Genuinely Non-Singular). A matrix $A(t)$ is genuinely non-singular if there exists a non-zero expression for its determinant at some $t = \hat{t}$.

EXAMPLE 2.6.10. The matrix:

$$\begin{pmatrix} x & y \\ y & x \end{pmatrix}$$

is genuinely non-singular, as it's non-singular for all values except when $x = \pm y$

DEFINITION 2.6.11 (Structurally Non-Singular). A matrix $A(t)$ is structurally non-singular if there exists a transversal that contains no zero-entries (or $-\infty$ entries for Σ).

EXAMPLE 2.6.12. The matrix:

$$\begin{pmatrix} x & y \\ 0 & 0 \end{pmatrix}$$

is structurally singular as every transversal uses a zero entry.

It is important to note a matrix may be structurally non-singular but be genuinely singular, e.g.

$$\begin{pmatrix} x & y \\ x & y \end{pmatrix}.$$

LEMMA 2.6.13. *If \mathbf{J} is genuinely non-singular then the System Jacobian at all negative stages is genuinely of full row rank.*

PROOF. If \mathbf{J} is of genuine full row rank then any subset of rows of \mathbf{J} is of genuine full row rank. If we order our Jacobian with descending offsets then consider a subset of rows so that all the rows of a negative stage System Jacobian are included, and then disregard the columns that are all zero (on the right hand side of the matrix), since $d_j - c_i > \sigma_{i,j}$ there. This has no effect on the rows being linearly independent, then we have the negative stage Jacobian which must be of genuine full row rank. \square

We now consider each stage k as using m_k equations to solve for n_k variables. Here m_k is the number of i for which $k + c_i \geq 0$ and n_k is the number of j for which $k + d_j \geq 0$, so that the following then holds.

LEMMA 2.6.14.

1. The m_k and n_k both increase with k , from 0 for k sufficiently negative, to n for $k \geq 0$.
2. $n_k \geq m_k \quad \forall k$.
3. $\sum_k (n_k - m_k) = DOF = \sum_j d_j - \sum_i c_i$.

We seek to prove each statement in turn.

1. As k increases towards 0 there are clearly going to be more, or at the least the same number of, equations and variables that satisfy the inequalities than there were at a previous stage, since it is only the stage number that changes the $2n$ equations for m_k and n_k at each stage, not the offsets. This holds until we reach step 0, where there will be n equations and n variables, since $c_i \geq 0$ and $d_j \geq 0$. We can also increase these values from 0 by setting $k = -\max_{i,j} (d_j, c_i) - 1$ as our initial stage. \square

2. $\sigma_{i,j} \geq 0 \quad \forall (i,j) \in S$. Now assume Σ is structurally non-singular and pick a HVT from S , say T . Let $(i,j) \in T$, from (34) we have $d_j - c_i \geq 0$ and hence $d_j \geq c_i$, meaning $k + d_j \geq k + c_i$ and hence from (38, 39) there are at least as many variables as equations at any stage k , since there is a one to one map, T , between the i 's and the j 's. Any $(i,j) \notin S$ yields a 0 entry in $\mathbf{J}_{i,j}$ and thus plays no role in the solution scheme. \square

3. Consider a matrix, E , with a 1 in entry $E_{i,k}$ if equation i is used in stage k and a 0 otherwise, and another, V , which is the same for variables. Here the number of stages starts at $-\max_j d_j$ (since all prior stages will have columns with only 0's in) and goes up to stage -1 (since stage 0 and on will have column's with all 1's in). For example, for the simple pendulum we have:

$$E = \begin{array}{c} k = \\ c \\ a \\ b \end{array} \begin{array}{cc} -2 & -1 \\ \left(\begin{array}{cc} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{array} \right), & \mathbf{V} = \begin{array}{c} k = \\ x \\ y \\ \lambda \end{array} \begin{array}{cc} -2 & -1 \\ \left(\begin{array}{cc} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{array} \right).$$

The DOF of a DAE is defined as the dimension of the set of valid initial conditions. An initial condition is needed every time there is one or more variable than equation. Therefore the DOF is given by the sum of all entries in V take the sum of all entries in E , because this will give us the total number of variable found minus the total number of equations used, as all non-negative k stages will have an equal number of equations and variables as outlined above. If we number the columns of E and V by k then a sum down the k column of E is m_k and down the k column of V is n_k . Hence summing down columns and then across rows gives:

$$\sum_k (n_k - m_k) = \text{DOF}.$$

We now consider a sum across rows first. It should be clear that row j of V will give us d_j , since we have d_j occurrences of variable x_j before stage 0, hence the name offset. Similarly row i of E will give us c_i . Hence a sum across rows and then down columns gives us:

$$\sum_j d_j - \sum_i c_i = \text{DOF}.$$

□

2.7. Exploiting DAE BTFS

We discuss block triangular forms (BTFS) natural to the Signature method, for more detail see [47] and [49]. We previously discussed two useful sparsity patterns for the signature method. We recap and expand on them here, giving an example to illustrate the ideas. We have the natural sparsity pattern for a DAE, the set where the entries of Σ are finite:

$$(42) \quad S = \{(i, j) \mid \sigma_{i,j} > -\infty\} \quad (\text{the sparsity pattern of } \Sigma).$$

and we have a more informative BTF coming from the sparsity pattern of the system Jacobian **J**:

$$(43) \quad S_0 = S_0(\mathbf{c}, \mathbf{d}) = \{(i, j) \mid d_j - c_i = \sigma_{i,j}\} \quad (\text{the sparsity pattern of } \mathbf{J}).$$

$$S_0(\mathbf{c}, \mathbf{d}) \subseteq S \quad \text{for any } \mathbf{c}, \mathbf{d}.$$

Unless otherwise stated we will only consider an *irreducible* BTF, that is one such that each block contains no possible sub-blocks based on the sparsity pattern associated with the BTF. In applications a Block BTF based on S_0 is usually significantly finer than one based on S and as such we call an irreducible BTF (unique up to possible re-ordering of blocks [47]) based on S the coarse BTF and on S_0 the fine BTF. We now define the concept of a local offset:

DEFINITION 2.7.1 (local offsets). The offsets found by treating each fine block in the fine BTF as a stand alone DAE are termed the local offsets and are denoted $\hat{\mathbf{c}}$ and $\hat{\mathbf{d}}$.

Local offsets are useful when trying to break a large DAE down in to smaller problems, one can then solve the whole DAE using the blocks in a stage-wise manner due to the following theorem from [48] and [40]:

THEOREM 2.7.2. *The difference between local and global offsets is a constant over a fine block.*

We call the difference between local and global offsets the lead time of a block l and denote it K_l .

EXAMPLE 2.7.3. Consider a modified double pendula DAE:

$$(44) \quad \left\{ \begin{array}{ll} f_1 = \ddot{x}_1 + x_1 x_3 & = 0, \\ f_2 = \ddot{x}_2 + x_2 x_3 - G & = 0, \\ f_3 = x_1^2 + x_2^2 - L^2 & = 0, \\ f_4 = \ddot{x}_4 + x_4 x_6 & = 0, \\ f_5 = (x_5^{(3)})^2 + x_5 x_6 - G & = 0, \\ f_6 = x_4^2 + x_5^2 - (L + c x_3)^2 + \ddot{x}_3 & = 0. \end{array} \right.$$

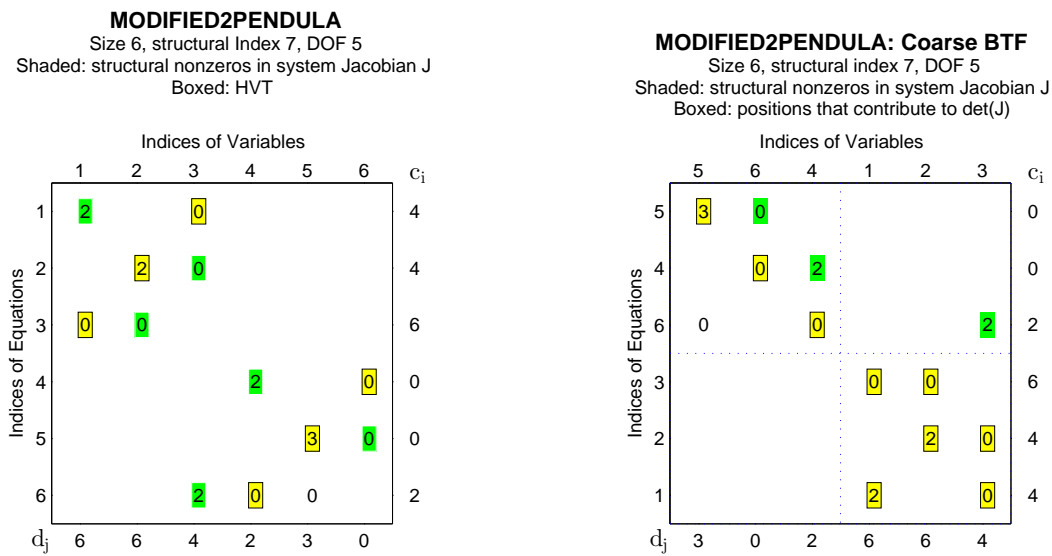
We present the different BTFs indicated by dotted lines in Figure 3. We see the fine BTF does indeed produce more blocks than the coarse BTF. The local offsets associated with the fine BTF allow us to consider a different staged solution scheme, i.e. we can solve each block using stage counters determined by the local offsets think of variables found by previous blocks as driving functions, see [48].

2.8. Classifying Non-Canonical Offsets

In the literature only canonical offsets are considered (due to uniqueness and providing a shorter solution scheme), this section expands the classes of potential offset vectors to be used in the structural analysis. We seek to illuminate the space of all possible offset vectors to better understand other solution schemes that could be employed. We start by noting that an offset vector classified by Definitions 2.5.3, 2.5.4 and 2.5.5 will potentially have some overlap, for example the canonical offset vectors are both valid and normalised. Let \mathcal{V} be the set of valid offset vectors, \mathcal{N} the set of normalised offset vectors and \mathcal{C} the singleton set containing the canonical offset vectors, i.e. $\mathcal{C} = \{(\text{canonical } \mathbf{c}, \text{canonical } \mathbf{d})\}$, so that $\mathcal{C} \subseteq \mathcal{N} \subset \mathcal{V}$. When we refer to a set of offset vectors we will mean those that belong that set only, i.e when we say valid offsets \mathbf{c} and \mathbf{d} we mean those in $\mathcal{V} \setminus \mathcal{N}$, unless context clearly requires a different meaning. Given a structurally non-singular $n \times n$ Σ there exist n equality constraints found on a HVT, T , of the form:

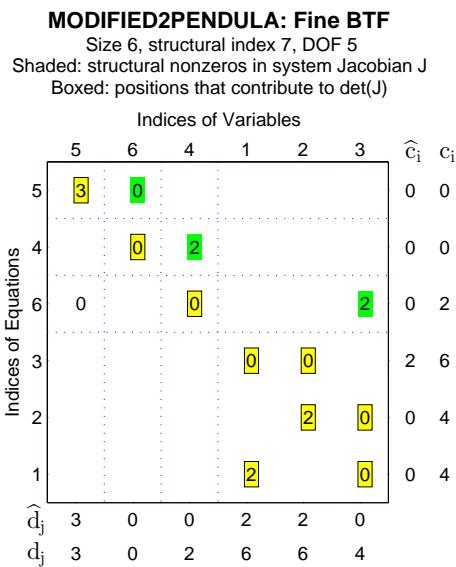
$$d_j - c_i = \sigma_{i,j},$$

and some number of inequalities equal to the entries in S that are not on a HVT. It is possible to use the n equalities to substitute for \mathbf{d} (or \mathbf{c}) in the inequalities and be left with only inequalities in \mathbf{c} (or \mathbf{d}), generating a feasible region of offset vectors \mathbf{c} (or \mathbf{d}). We can then use these feasible regions to recover an equivalent diagram for \mathbf{d} (or \mathbf{c}), below we show these regions for \mathbf{c} .



(a) original structure

(b) coarse structure



(c) fine structure

FIGURE 3. Structure of Equation (44) and its block-triangularizations.

EXAMPLE 2.8.1. We consider a DAE with structure given by:

$$\begin{cases} A(x) = 0. \\ B(x, \dot{y}) = 0. \end{cases}$$

This system has the following signature tableau:

$$\Sigma = \begin{array}{ccc} & x & y & c_i \\ A & \begin{pmatrix} 0 & -\infty \end{pmatrix} & 0 \\ B & \begin{pmatrix} 0 & 1 \end{pmatrix} & 0 \\ d_j & 0 & 1 \end{array}$$

If we explicitly write out the equations for the offsets given by (34) we get the following system of equations:

$$\begin{cases} d_1 - c_1 = 0, \\ d_2 - c_2 = 1, \\ d_1 - c_2 \geq 0. \end{cases}$$

If we solve the first two to get the c 's in terms of the d 's then substitute our values into the third the constraints reduce to $c_2 \leq c_1$, $c_1 \geq 0$ and $c_2 \geq 0$. For normalised offsets we have $c_1 = 0$ and $c_2 \in \mathbb{N}_0$ or $c_1 \in \mathbb{N}_0$ and $c_2 = 0$. This is represented graphically in Figure 4, where the blue dots represent a normalised choice of offsets and the green dots represent a valid choice, with red shading to indicate infeasible regions. The canonical offset vector is found by means of a dual linear programming method with objective function $\min \sum_{i=1}^2 c_i$.

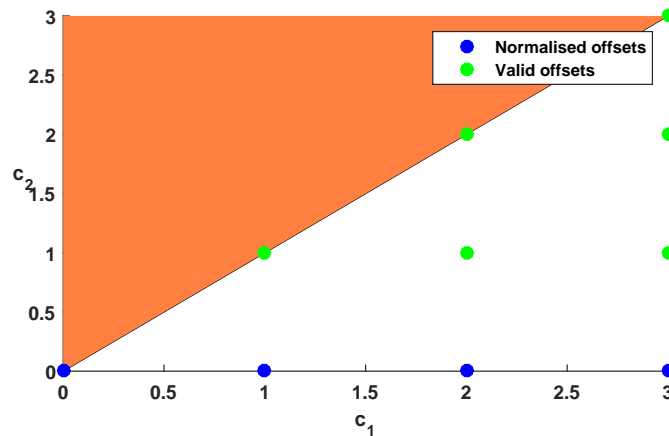


FIGURE 4. Feasible c_i for example 2.8.1, with infeasible region shaded.

EXAMPLE 2.8.2. We now present an example where there is a finite choice of normalised offset values, specifically two. Consider a DAE with structure given by:

$$\begin{cases} A(\dot{x}, y) = 0, \\ B(x, y) = 0. \end{cases}$$

with signature tableau:

$$\Sigma = \begin{array}{ccc} & x & y & c_i \\ A & \begin{pmatrix} 1 & 0 \end{pmatrix} & 0 \\ B & \begin{pmatrix} 0 & 0 \end{pmatrix} & 0 \\ d_j & 1 & 0 \end{array}$$

Writing out the equations given by (34) gives us:

$$\begin{cases} d_1 - c_1 = 1, \\ d_2 - c_2 = 0, \\ d_1 - c_2 \geq 0, \\ d_2 - c_1 \geq 0. \end{cases}$$

When simplified to inequalities involving only the c 's yields:

$$\begin{cases} c_2 \leq c_1 + 1, \\ c_2 \geq c_1. \end{cases}$$

Here we have two constraints, as shown by Figure 5.

EXAMPLE 2.8.3. We now stretch our understanding to systems of 3 equations in 3 unknowns. Consider a DAE with structure given by:

$$\begin{cases} A(y) = 0, \\ B(x, z) = 0, \\ C(\dot{x}, y) = 0, \end{cases}$$

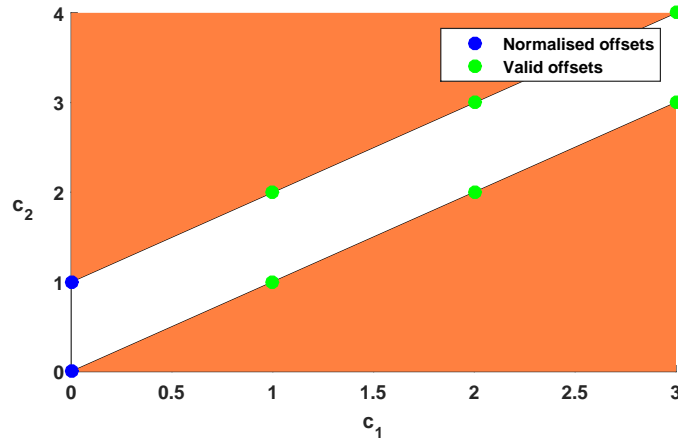


FIGURE 5. Feasible c_i for example 2.8.2.

with signature tableau:

$$\Sigma = \begin{array}{cccc} & x & y & z & c_i \\ A & \left(\begin{array}{ccc} -\infty & 0 & -\infty \end{array} \right) & 0 \\ B & \left(\begin{array}{ccc} 0 & -\infty & 0 \end{array} \right) & 0 \\ C & \left(\begin{array}{ccc} 1 & 0 & -\infty \end{array} \right) & 0 \\ d_j & 1 & 0 & 0 & \end{array}$$

Once again we look at the equations generated by (34), the equality constraints are:

$$\begin{cases} d_1 - c_3 = 1, \\ d_2 - c_1 = 0, \\ d_3 - c_2 = 0, \end{cases} \rightarrow \begin{cases} d_1 = 1 + c_3, \\ d_2 = c_1, \\ d_3 = c_2, \end{cases}$$

and inequality constraints are:

$$\begin{cases} d_1 - c_2 \geq 0, \\ d_2 - c_3 \geq 0, \end{cases}$$

which reduce to:

$$\begin{cases} c_3 \geq c_2 - 1, \\ c_3 \leq c_2, \end{cases}$$

which gives rise to Figure 6. We have two planes of possible offset vectors, with an infinite

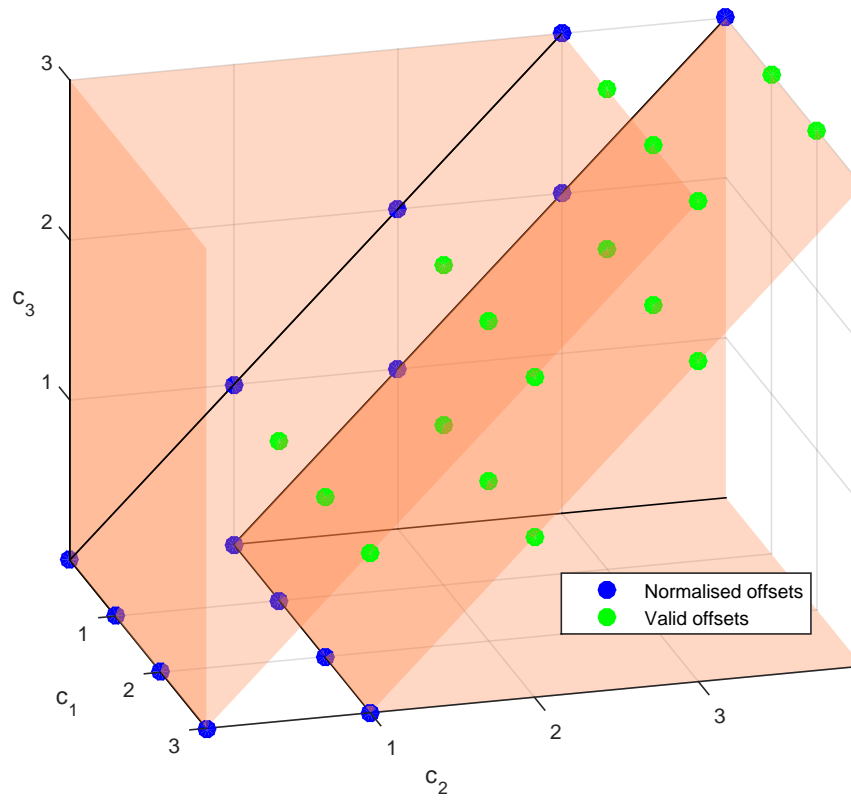


FIGURE 6. Feasible c_i for example 2.8.3.

number of both valid and normalised offsets.

CHAPTER 3

Dummy Derivatives

3.1. Introduction to Dummy Derivatives

Usually to solve a DAE one differentiates parts of the DAE in order to reveal the hidden constraints of the problem, as done in Chapter 2. The Dummy Derivative (DD) approach adds the hidden constraint equations to the system directly, as opposed to solving them stage-wise as done above in the signature matrix method. This makes the DAE over-determined. To get a square DAE again the method adds extra dependent variables to the system for each constraint equation added, specifically the method finds a subset of appearing derivatives of variables to be considered algebraic. In [30] it is proved that this approach yields a DAE with the same solution as the original DAE. The advantage of this approach is by solving the resulting index 1 DAE we satisfy all constraint equations of the original DAE automatically. Unfortunately this approach is inherently local, as will be shown in §3.2, which means one may have to reformulate the problem (sometimes periodically) throughout numerical simulation, as explained in §3.6. In §3.4 and §3.5 we use the signature matrix method to inform our choices of dummy derivatives by finding so-called *necessary* dummy derivatives, removing choice in the algorithm and increasing computational efficiency, this yields different classes of DDs and new algorithms not before seen in the literature. In particular this Chapter highlights choice in the block form not previously known, gives new insights in to why certain derivatives must or can be DDs and gives several new ways of approaching dummy derivative reformulation, linking our approach to a new method from [54] in §3.5.2.

3.2. Original Dummy Derivative Algorithm

We will be re-ordering the original method of [30] somewhat, thus to avoid confusion with readers already familiar with DDs we present the original algorithm in §3.2.1, for reference and provide a brief example for illustration.

3.2.1. The method as presented originally. Although originally the algorithm was given using a block lower triangular (BLT) form we shall, at least initially, consider the algorithm over one block. This simplifies notation considerably and allows us to note interactions between blocks later to find a reduced block triangular form (BTF) with which to find DDs. It also highlights a subtlety in the block form choice not mentioned in the original paper [29].

We define notation needed in the DD method, but first present a necessary definition for our analysis of the algorithm, see [45] for more details:

DEFINITION 3.2.1 (Differential Algebraic Operator). A differential algebraic operator (DAO) is a vector of size m acting on n dependent variables, say x_1, \dots, x_n , which are themselves regarded as functions of an independent variable t . More formally we have an \mathbb{R}^m valued function \mathbf{f} of an independent variable t , dependent variables x_1, \dots, x_n , and finitely many derivatives of the dependent variables. If we index derivatives so that x_{j0} corresponds to $x_j(t)$, x_{j1} to $x'_j(t)$ and so on then we have the following map:

$$(45) \quad \mathbf{f} : \mathbb{R} \times \mathbb{R}^{\mathcal{J}} \rightarrow \mathbb{R}^m,$$

where \mathcal{J} is the index set:

$$(46) \quad \mathcal{J} = \{(j, l) : j = 1, \dots, n; l = 0, 1, \dots\}.$$

Consider a DAE to be given by $\mathcal{F}x = 0$, where \mathcal{F} is a (column n -vector) differential-algebraic operator (DAO) and x are n unknowns dependent on some variable t , we then have the following notation:

- (1) $\boldsymbol{\nu} = \boldsymbol{\nu}(\mathcal{F})$, a column n -vector of non-negative integers, containing the minimum number of differentiations of each equation to get an ODE, usually found by Pantelides' Method [41] or as the \mathbf{c} in SA. We use \mathbf{c} in throughout this thesis as it provides more information than Pantelides' method that we wish to exploit [45].
- (2) $D^\nu = \text{diag}(\frac{d^{\nu_1}}{dt^{\nu_1}}, \dots, \frac{d^{\nu_n}}{dt^{\nu_n}})$, regarded as a DAO.
- (3) The differentiated problem denoted by $\mathcal{G}x = D^{\nu(\mathcal{F})}\mathcal{F}x = 0$.
- (4) A symbolic vector, z , of *highest order derivatives* (HODs) of $x_i(t)$ in $\mathcal{G}x$, with first entry equal to the HOD of x_1 etc...
- (5) A system of equations $g(z) = 0$ —The equations in $\mathcal{G}x = 0$ —.

There are three main stages in finding DDs:

- (1) Get $\boldsymbol{\nu}$.
- (2) Obtain a differentiated problem $\mathcal{G}x = 0$.
- (3) Perform the index reduction algorithm. Loop through stages that select derivatives to be considered as algebraic variables in the solution process.

For ease of analysis later we will assume, without loss of generality, that the equations (and variables) have been sorted into descending order with respect to number of differentiations needed, i.e. have been sorted with descending d_j (and c_i). We consider each stage in turn by the superscript $[\kappa]$. The index reduction part of the algorithm, given in as much brevity as reasonable—for full details see [30]—is given in Algorithm 2.

Algorithm 2 The Dummy Derivative Algorithm

Initialise: $z^{[1]} \leftarrow z$, $g^{[1]}(z^{[1]}) \leftarrow g(z)$, $G^{[1]} = \frac{\partial g^{[1]}}{\partial z^{[1]}} \leftarrow \frac{\partial g}{\partial z}$, $\kappa = 1$.

- 1: Let m be the number of differentiated equations in $g^{[0]}$.
 - 2: **while** $g^{[\kappa]}$ has m differentiated equations, with $m > 0$
 - 3: Let $h^{[\kappa]} = 0$ be the first m rows of $g^{[\kappa]} = 0$, so that $H^{[\kappa]} = \frac{\partial h^{[\kappa]}}{\partial z^{[\kappa]}}$.
 - 4: Choose m columns of $H^{[\kappa]}$ producing a square non-singular $M^{[\kappa]}$.
 - 5: Form $\hat{z}^{[\kappa]}$, the HODs of variables occurring in $M^{[\kappa]}$.
 - 6: Make variables occurring in $\hat{z}^{[\kappa]}$ into dummy derivatives.
 - 7: Add equations $h^{[\kappa]} = 0$ to the DAE.
 - 8: Omit one differentiation: $g^{[\kappa+1]} \leftarrow D^{-1}h^{[\kappa]}$, $z^{[\kappa+1]} \leftarrow D^{-1}\hat{z}^{[\kappa]}$, $G^{[\kappa+1]} \leftarrow M^{[\kappa]}$.
 - 9: Let m be the number of differentiated equations in $g^{[\kappa+1]}$.
 - 10: Set $\kappa = \kappa + 1$.
-

EXAMPLE 3.2.2. We use an example found in [30] and apply the algorithm above to find DDs. Consider a linear constant coefficient DAE, with known smooth forcing functions $u_1(t), \dots, u_4(t)$:

$$(47) \quad \mathcal{F}x = \mathcal{F} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{pmatrix} = \begin{cases} a(t) = x_1(t) + x_2(t) & +u_1(t) = 0, \\ b(t) = x_1(t) + x_2(t) + x_3(t) & +u_2(t) = 0, \\ c(t) = x_1(t) & +x_3'(t) + x_4(t) + u_3(t) = 0, \\ d(t) = 2\ddot{x}_1(t) + \ddot{x}_2(t) + \ddot{x}_3(t) + \dot{x}_4(t) & +u_4(t) = 0. \end{cases}$$

We need a number of differentiations for each equation to get $\mathcal{G}x = 0$, to do this we'll compute the signature matrix to find \mathbf{c} :

$$(48) \quad \Sigma = \begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 & c_i \\ a & \left(\begin{array}{cccc} 0^\bullet & 0 & -\infty & -\infty \end{array} \right) & 2 & & & \\ b & \left(\begin{array}{cccc} 0 & 0^\bullet & 0 & -\infty \end{array} \right) & 2 & & & \\ c & \left(\begin{array}{cccc} 0 & -\infty & 1^\bullet & 0 \end{array} \right) & 1 & & & \\ d & \left(\begin{array}{cccc} 2 & 2 & 2 & 1^\bullet \end{array} \right) & 0 & & & \\ d_j & 2 & 2 & 2 & 1 & \end{array} .$$

Hence, $\boldsymbol{\nu} = \mathbf{c} = (2, 2, 1, 0)$ and our differential operator, D^ν , is:

$$D^\nu = \begin{pmatrix} \frac{d^2}{dt^2} & 0 & 0 & 0 \\ 0 & \frac{d^2}{dt^2} & 0 & 0 \\ 0 & 0 & \frac{d}{dt} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and the differentiated problem $D^\nu \mathcal{F}x = \mathcal{G}x = 0$ is:

$$(49) \quad \mathcal{G}x = D^\nu \mathcal{F}x = \begin{cases} \ddot{a}(t) = \ddot{x}_1(t) + \ddot{x}_2(t) & + \ddot{u}_1(t) = 0, \\ \ddot{b}(t) = \ddot{x}_1(t) + \ddot{x}_2(t) + \ddot{x}_3(t) & + \ddot{u}_2(t) = 0, \\ \dot{c}(t) = \dot{x}_1(t) & + \ddot{x}_3(t) + \dot{x}_4(t) + \dot{u}_3(t) = 0, \\ d(t) = 2\ddot{x}_1(t) + \ddot{x}_2(t) + \ddot{x}_3(t) + \dot{x}_4(t) & + u_4(t) = 0. \end{cases}$$

The vector of HODs, for $\mathcal{G}x = 0$, is given by $\mathbf{z}^{[1]} = (\ddot{x}_1, \ddot{x}_2, \ddot{x}_3, \dot{x}_4)$ and the current equations are $\mathbf{g}^{[1]}(\mathbf{z}^{[1]}) = (\ddot{a}, \ddot{b}, \dot{c}, d)^T$ and

$$\frac{\partial \mathbf{g}^{[1]}}{\partial \mathbf{z}^{[1]}} = G^{[1]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \\ \begin{matrix} \ddot{a} \\ \ddot{b} \\ \dot{c} \\ d \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 \end{pmatrix} \end{matrix}.$$

Stage 1

Let $\mathbf{h}^{[1]} = (\ddot{a}, \ddot{b}, \dot{c})^T$, i.e. the differentiated equations, then

$$\frac{\partial \mathbf{h}^{[1]}}{\partial \mathbf{z}^{[1]}} = H^{[1]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \\ \begin{matrix} \ddot{a} \\ \ddot{b} \\ \dot{c} \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix}.$$

We now have two possibilities for selecting columns to get a non-singular matrix, either we choose 1,3,4 or we choose 2,3,4. For the purposes of this example we can make either selection arbitrarily (since the resulting matrices $M^{[1]}$ are the same). In practice this selection would usually be made using an estimate of the condition number of $M^{[1]}$ at each (or at least some) time steps. Choosing columns 1,3,4 yields:

$$M^{[1]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_3 & \dot{x}_4 \\ \begin{matrix} \ddot{a} \\ \ddot{b} \\ c' \end{matrix} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \end{matrix}.$$

We have differentiated variables $\widehat{\mathbf{z}}^{[1]} = (\ddot{x}_1, \ddot{x}_3, \dot{x}_4)$ which are to be made DDs. We now omit one differentiation (re-initialise for the next stage of the reduction procedure) by setting

$\mathbf{g}^{[2]} = D^{-1}\mathbf{h}^{[1]} = (\dot{a}, \dot{b}, c)^T$, $\mathbf{z}^{[2]} = D^{-1}\widehat{\mathbf{z}}^{[1]} = (x_1, x_3, x_4)$ and $G^{[2]} = M^{[1]}$.

Stage 2

We now have a vector of differentiated equations $\mathbf{h}^{[2]} = (\dot{a}, \dot{b})^T$ and thus:

$$H^{[2]} = \begin{matrix} & x_1 & x_3 & x_4 \\ \dot{a} & \left(\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \end{array} \right) \\ \dot{b} & \end{matrix}$$

We have to remove column three to get a square non-singular matrix:

$$M^{[2]} = \begin{matrix} & x_1 & x_3 \\ \dot{a} & \left(\begin{array}{cc} 1 & 0 \\ 1 & 1 \end{array} \right) \\ \dot{b} & \end{matrix}$$

We have differentiated variables $\widehat{\mathbf{z}}^{[2]} = (x_1, x_3)$ which are to be made DDs. We now omit one differentiation by setting $\mathbf{g}^{[3]} = D^{-1}\mathbf{h}^{[2]} = (a, b)^T$, $\mathbf{z}^{[3]} = D^{-1}\widehat{\mathbf{z}}^{[2]} = (x_1, x_3)$ and $G^{[3]} = M^{[2]}$.

Stage 3

We have no undifferentiated equations, so the algorithm ends. Equation (50) presents the

resulting index 1 DAE, with DDs marked in prime notation, i.e. $\dot{x} = x'$ and $\ddot{x} = x''$:

$$(50) \quad \begin{cases} a(t) = x_1(t) & +x_2(t) & & + u_1(t) = 0, \\ b(t) = x_1(t) & +x_2(t) + x_3(t) & & + u_2(t) = 0, \\ \dot{a}(t) = x'_1(t) & +\dot{x}_2(t) & & + \dot{u}_1(t) = 0, \\ \dot{b}(t) = x'_1(t) & +\dot{x}_2(t) + x'_3(t) & & + \dot{u}_2(t) = 0, \\ c(t) = x_1(t) & & + x'_3(t) + x_4(t) + u_3(t) = 0, \\ \ddot{a}(t) = x''_1(t) & +\ddot{x}_2(t) & & + \ddot{u}_1(t) = 0, \\ \ddot{b}(t) = x''_1(t) & +\ddot{x}_2(t) + x''_3(t) & & + \ddot{u}_2(t) = 0, \\ \dot{c}(t) = x'_1(t) & & + x''_3(t) + x'_4(t) + \dot{u}_3(t) = 0, \\ d(t) = 2x''_1(t) & +\ddot{x}_2(t) + x''_3(t) + x'_4(t) + u_4(t) = 0. \end{cases}$$

3.3. Reordered Dummy Derivative Algorithm

To make it simpler to draw comparisons between SA and DDs we reorder the index reduction part of the original algorithm as presented in [29] to become Algorithm 3, doing this allows the DD Jacobians G to be directly comparable with ‘equivalent’ stage SA Jacobians \mathbf{J} . We start the algorithm from stage 0 rather than 1. We remove the appearance of the matrix $M^{[\kappa]}$, since $G^{[\kappa+1]} = M^{[\kappa]}$ this matrix is only useful for bookkeeping. We also now compute H at the end of a stage as opposed to G so that we can have matrix $G^{[\kappa]}$ computed at stage κ rather than stage $\kappa - 1$. Computing our matrices somewhat out of order means it’s easier to treat stage 0 as a special case in the algorithm, since we need to initialise stage κ with matrix $H^{[\kappa-1]}$ but $H^{[0]}$ needs to be computed from $G^{[0]}$ and thus cannot be given as an input initially.

EXAMPLE 3.3.1. We use the same example as before (Example 3.2.2) to illustrate this altered dummy derivative algorithm. Consider again equation (47) and its corresponding signature matrix (48) As before the HODs are $\mathbf{z}^{[0]} = (\dot{x}_1, \ddot{x}_2, \ddot{x}_3, \dot{x}_4)$, the current equations are $\mathbf{g}^{[0]}(\mathbf{z}^{[0]}) = (\ddot{a}, \ddot{b}, \dot{c}, d)^T$. Our algorithm is thus:

Algorithm 3 The Reordered Dummy Derivative Algorithm

Initialise: $z = z^{[0]}$, $g(z) = g^{[0]}(z^{[0]})$, $\kappa = 0$

- 1: **if** $\kappa = 0$
 - 2: $G^{[0]} = \frac{\partial g^{[0]}}{\partial z^{[0]}}$
 - 3: Let m be the number of differentiated equations in $g^{[0]}(z^{[0]})$
 - 4: Let $H^{[0]}$ be the first m rows of $G^{[0]}$
 - 5: $\kappa = \kappa + 1$
 - 6: **else**
 - 7: **while** $H^{[\kappa-1]} \neq []$
 - 8: Let $G^{[\kappa]}$ be m columns of $H^{[\kappa-1]}$ such that we have a non-singular matrix
 - 9: Make the corresponding variables used in $G^{[\kappa]}$ into DDs
 - 10: Omit one differentiation to get $z^{[\kappa]}$, $g^{[\kappa]}(z^{[\kappa]})$
 - 11: (where we only consider variables and equations in $G^{[\kappa]}$)
 - 12: Let m be the number of differentiated equations in $g^{[\kappa]}(z^{[\kappa]})$
 - 13: Let $H^{[\kappa]}$ be the first m rows of $G^{[\kappa]}$,...
 - 14: (the rows using differentiated equations in $g^{[\kappa]}(z^{[\kappa]})$)
 - 15: $\kappa = \kappa + 1$
 - 16: Consider the new system using all equations $g^{[\kappa]}(z^{[\kappa]})$, where $\kappa \geq 0, \dots$
 - 17: and dummy derivatives for $z^{[\kappa]}$, where $\kappa > 0$, as well as all original variables.
-

Stage 0

Initialise and then remove undifferentiated equations, since $m = 3$ we get:

$$G^{[0]} = \frac{\partial \mathbf{g}^{[0]}}{\partial \mathbf{z}^{[0]}} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \\ \ddot{a} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 \end{pmatrix} \end{matrix} \quad \text{and} \quad H^{[0]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \\ \ddot{a} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix}.$$

Stage 1

We now have two possibilities for selecting columns to get a non-singular matrix. Choosing to omit column 2 gives $G^{[1]}$ below. Therefore \ddot{x}_1 , \ddot{x}_3 and \dot{x}_4 are made DDs, denoted by x''_1 , x''_3 and x'_4 . Reducing the order of differentiation by 1 gives $\mathbf{z}^{[1]} = (\dot{x}_1, \dot{x}_3, x_4)$, current

equations are $\mathbf{g}^{[1]}(\mathbf{z}^{[1]}) = (\dot{a}, \dot{b}, c)^T$, so $m = 2$ and we get $H^{[1]}$ as below:

$$G^{[1]} = \begin{array}{c} \ddot{x}_1 \quad \ddot{x}_3 \quad \dot{x}_4 \\ \ddot{a} \begin{pmatrix} 1 & 0 & 0 \\ \dot{b} \begin{pmatrix} 1 & 1 & 0 \\ \dot{c} \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \end{array} \quad \text{and} \quad H^{[1]} = \begin{array}{c} \dot{x}_1 \quad \dot{x}_3 \quad x_4 \\ \dot{a} \begin{pmatrix} 1 & 0 & 0 \\ \dot{b} \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Stage 2

We now have only one possibility, so must have:

$$G^{[2]} = \begin{array}{c} \dot{x}_1 \quad \dot{x}_3 \\ \dot{a} \begin{pmatrix} 1 & 0 \\ \dot{b} \begin{pmatrix} 1 & 1 \end{pmatrix} \end{array}.$$

Thus \dot{x}_1 and \dot{x}_3 are made DDs, denoted by x'_1 and x'_3 . Reducing the order of differentiation by 1 gives $\mathbf{z}^{[2]} = (x_1, x_3)$, the current equations are $\mathbf{g}^{[2]}(\mathbf{z}^{[2]}) = (a, b)^T$, so $m = 0$ and $H^{[2]} = []$ and the algorithm ends, yielding an index 1 system equivalent to the one found in [29], Equation (50).

Note: The zeros in the third column of $H^{[1]}$ mean we never choose x_4 as a DD, this is (as one would hopefully expect) always the case for undifferentiated variables, as explained in the following section. Also, the variables and equations used in $G^{[\kappa]}$ are just $D\mathbf{z}^{[\kappa]}$ and $D\mathbf{g}^{[\kappa]}(\mathbf{z}^{[\kappa]})$.

3.4. Using Structural Analysis to Simplify Dummy Derivatives

If we consider $H^{[\kappa]}$ to be a matrix of size $n_\kappa \times m_\kappa$, then from the algorithm in §3.3 we have $\binom{m_\kappa}{n_\kappa}$ potential index 1 systems at stage $\kappa + 1$. Thus the potential number of index 1 systems obtainable by the method can be very large for practical examples. We would like to use the structural analysis of Chapter §2 to inform our choice of DDs and thus limit the potential number of systems considered at each stage. Note, it is of course possible and done in practice to compute all possible Jacobians at a stage then store only those that are

numerically non-singular, but this again can be costly or lead to memory issues for large problems [46].

3.4.1. Similarities in the methods. There are several similarities between SA and DDs, a preliminary summary is given in [32] and a more detailed expansion is given in [33], this section expands on both these papers. Firstly, as was said above, the ν used in DDs is the same as \mathbf{c} in SA, as from [45] we see that Pantelides algorithm [41] and SA can be used interchangeably. Therefore we have that $D^\nu = \text{diag}(\frac{d^{c_1}}{dt^{c_1}}, \dots, \frac{d^{c_n}}{dt^{c_n}})$ and have the following equality:

$$\mathcal{G}x = D^\nu \mathcal{F}x = D^{\mathbf{c}} \mathcal{F}x.$$

We are differentiating each equation f_i , c_i times, so the maximum derivative for each variable x_j in $\mathcal{G}x = 0$ will equal $\max_i(\sigma_{i,j} + c_i)$; from (34) this is d_j . Hence, the 0th stage system in DDs (Algorithm 3) is the 0th stage system in SA. Thus the differentiated problem can be written:

$$f_i^{(c_i)}(t, x_j^{(d_j)}; \text{lower order derivatives}) = 0, \quad \text{for } i, j = 1, \dots, n.$$

Therefore we must have that:

$$\mathbf{z}^{[0]} = (x_1^{(d_1)}, \dots, x_n^{(d_n)}).$$

The formula for the DD Jacobian matrix $G^{[0]}$ can now be written in this SA based notation to show it actually equals SA's stage 0 Jacobian \mathbf{J} .

$$(51) \quad G^{[0]} = \frac{\partial \mathbf{g}^{[0]}}{\partial \mathbf{z}^{[0]}} = \left(\frac{\partial f_i^{(c_i)}}{\partial x_j^{(d_j)}} \right) = \mathbf{J}.$$

Going to the next stage in DDs by reducing the order of differentiation by 1 is equivalent to reducing the offset vector \mathbf{c} by 1 after removing its zero entries (and consequently reducing \mathbf{d} by 1 also). Therefore at stage 1 in DDs we will be considering the equations used in stage -1 of SA, since SA increases the order of differentiation by one at each stage. This leads us to the following observation.

LEMMA 3.4.1. *The equations used at stage k in SA are equal to those used at stage $\kappa = -k$ in DDs (when writing down $G^{[\kappa]}$), for each stage k between k_{min} and 0.*

PROOF. We have already shown that at stage 0 both methods use the same equations. In DDs we now remove all equations such that $c_i = 0$. We then omit one differentiation and repeat. Hence we remove, at stage 1, equations such that $c_i - 1 = 0$, and by induction, at stage κ equations such that $c_i - \kappa = 0$, where κ is the stage number. From (38) these are exactly the equations considered at stage $-\kappa = k$ in SA. \square

Due to the above lemma we will now use the term ‘equivalent stage’ to mean DD stage κ when talking about SA stage $-k = \kappa$ and vice versa. We take notation from [45] in order to write down the k^{th} stage System Jacobian found in SA. Consider each variable x_j as a function of an independent variable t and let x_{jl} represent $x_j^{(l)}(t)$. Then for an $n \times n$ DAE we have the index set:

$$\mathcal{J} = \{(j, l) \mid j = 1, \dots, n; l = 0, 1, \dots\},$$

similarly for the equations we use the set:

$$\mathcal{I} = \{(i, l) \mid i = 1, \dots, n; l = 0, 1, \dots\}.$$

This gives us a notation for the variables used at each stage in the SA:

$$I_k = \{(i, l) \in \mathcal{I} \mid l = k + c_i\},$$

$$J_k = \{(j, l) \in \mathcal{J} \mid l = k + d_j\},$$

where the offsets are taken to be canonical (see Definition 2.5.5) unless otherwise stated. At each SA stage we have:

$$m_k = |\mathcal{I}_k| = |\{j \mid d_j + k \geq 0\}|, \quad n_k = |\mathcal{J}_k| = |\{i \mid c_i + k \geq 0\}|.$$

TABLE 1. The results of SA on the linear DAE (47)—also showing DDs by a prime.

k	Equations being used	Variables being found
-2	a, b	x_1, x_2, x_3
-1	\dot{a}, \dot{b}, c	$x'_1, \dot{x}_2, x'_3, x_4$
0	$\ddot{a}, \ddot{b}, \dot{c}, d$	$x''_1, \ddot{x}_2, x''_3, x'_4$

We write $\mathbf{f}_{\mathbf{I}_k}$ to mean the set of equations used at stage k in SA and $\mathbf{f}_{\mathbf{I}_{\leq k}}$ to mean the set of equations used between SA stage k_{\min} and k —i.e. $(\mathbf{f}_{\mathbf{I}_{k_{\min}}}, \mathbf{f}_{\mathbf{I}_{k_{\min}+1}}, \dots, \mathbf{f}_{\mathbf{I}_k})$. We use a similar notation for the variables.

Again, from [45] we have that the system Jacobian used at stage k in SA is given by:

$$(52) \quad \mathbf{J}_k = \frac{\partial \mathbf{f}_{\mathbf{I}_k}}{\partial \mathbf{x}_{\mathbf{J}_k}}.$$

Recall the note at the end of §3.3, which gives us the following lemma:

LEMMA 3.4.2. *In DDs if at stage κ , $d_j = \kappa + 1$ then column j cannot be in $G^{[\kappa+1]}$.*

PROOF. If $1 \leq i \leq m_k$ and $j > m_k$ then $k + c_i > 0$ and $k + d_j \leq 0$, hence $d_j - c_i < 0$ and thus cannot be equal to $\sigma_{i,j}$, so that $(\mathbf{J}_k)_{ij} = 0$ due to the definition of the System Jacobian in Equation (37). Since we have $G^{[0]} = \mathbf{J}_0 = \mathbf{J}$ and the DD algorithm is reducing the order of differentiation by one at each stage, if the column referring to x_j appears in \mathbf{J}_k and $H^{[-k-1]}$ then its entries must be the same. Thus columns with $d_j = \kappa + 1$ cannot be selected to form $G^{[-k]}$, as they will be columns of structural zeros. \square

Thus columns representing variables that are undifferentiated cannot be chosen as DDs, as one would expect, otherwise we would not introduce new dummy variables to the system.

EXAMPLE 3.4.3. Consider example 3.2.2 and recall the index 1 system given in equation (50). Compare this with the SA results in Table 1, the variables that became DDs are marked by prime notation. In Example 3.2.2 we make a subset of the variables found at stage k in SA into DDs at stage $-k + 1 = \kappa + 1$ in the DD scheme, using the same equations in both cases, see Table 1.

The total number of DDs introduced will be $\sum_i c_i$, since this is the total number of new equations introduced and we identify one DD with each new equation. At each stage the variables that produced DDs are a subset of the variables that produced DDs at the previous stage (necessarily excluding those with $d_j = \kappa + 1$), of size m_{k-1} , with each variable being differentiated one time less than in the previous stage. So, the DDs will be a subset of the variables solved for at the equivalent stage +1 of SA, thus we have the following:

THEOREM 3.4.4. *The matrix $G^{[\kappa]}$ is a submatrix of (it may be equal to) \mathbf{J}_k , where $\kappa = -k$.*

Before going into a deeper comparison we consider the 0 DOF case as this simplification yields some interesting observations.

3.4.2. DDs and SA in the 0 DOF case. We begin by noting the following lemma for the offsets of a 0 DOF DAE:

LEMMA 3.4.5. *If a square non-singular DAE has 0 DOF and, without loss, has been reordered to put a HVT on the main diagonal of Σ then $\mathbf{d} = \mathbf{c}^T$.*

PROOF. If we have 0 DOF then $\sum_j d_j - \sum_i c_i = 0$ from 2.6.14 and by (34) $d_i - c_i \geq 0, \quad \forall i$. Hence, the non-negative numbers $d_i - c_i$ sum to 0 and thus must equal 0. \square

We also include a second shorter proof to help illustrate the point to the reader:

PROOF. From (34) we have $d_j = c_i + \sigma_{i,j} = c_i$ on a HVT as all $\sigma_{i,j} = 0$ on the HVT. \square

This gives us the following theorem:

THEOREM 3.4.6. *If we have 0 DOF then $G^{[-k]} = \mathbf{J}_k$ for each stage k between k_{min} and 0.*

PROOF. We have no choice in our selection of $G^{[\kappa]}$ since $H^{[\kappa-1]}$ (of size $m \times n$ say) must contain only m columns of structural non-zeros, since $n - m$ other columns correspond to undifferentiated variables due to Lemma 3.4.5. Noting that $\mathbf{J}_0 = G^{[0]}$ completes the proof. \square

Hence in the 0 DOF case Theorem 3.4.4 can be made more precise.

EXAMPLE 3.4.7. In [7] the authors introduce a DAE for modelling a robot arm. It is reformulated to be structurally well posed in [44]:

$$(53) \quad \left\{ \begin{array}{l} 0 = D = \ddot{x}_1 - [2(\dot{x}_1 + \dot{x}_3)^2 c(x_3) + \dot{x}_1^2 d(x_3) + (2x_3 - x_2) \cdot (a(x_3) + 2b(x_3)) + a(x_3)w], \\ 0 = E = \ddot{x}_2 - [-(2(\dot{x}_1 + \dot{x}_3)^2 c(x_3) + \dot{x}_1^2 d(x_3)) + (2x_3 - x_2) \cdot (1 - 3a(x_3) - 2b(x_3)) - a(x_3)w + u_2], \\ 0 = F = \ddot{x}_3 - [-(2(\dot{x}_1 + \dot{x}_3)^2 c(x_3) + \dot{x}_1^2 d(x_3)) + (2x_3 - x_2) \cdot (a(x_3) - 9b(x_3)) - 2\dot{x}_1^2 c(x_3) \\ \quad - d(x_3)(\dot{x}_1 + \dot{x}_3)^2 - (a(x_3) + b(x_3))w], \\ 0 = G = \cos x_1 + \cos(x_1 + x_3) - p_1(t), \\ 0 = H = \sin x_1 + \sin(x_1 + x_3) - p_2(t), \\ 0 = K = w - (u_1 - u_2). \end{array} \right.$$

Here

$$p_1(t) = \cos(e^t - 1) + \cos(t - 1),$$

$$p_2(t) = \sin(1 - e^t) + \sin(1 - t),$$

$$a(s) = \frac{2}{2 - \cos^2 s}, \quad b(s) = \frac{\cos s}{2 - \cos^2 s},$$

$$c(s) = \frac{\sin s}{2 - \cos^2 s}, \quad d(s) = \frac{\cos s \sin s}{2 - \cos^2 s}.$$

We arrange the equations and variables such that we clearly illustrate the block triangular structure of the problem. Structural Jacobian, signature matrix and offsets (with a HVT marked by \bullet and $-\infty$ entries left blank) for this DAE are, where F_w means $\partial F/\partial w$ and so

on:

$$\Sigma = \begin{array}{c} \begin{array}{cccccc} & x_1 & x_3 & w & x_2 & u_2 & u_1 & c_i \\ G & \left(\begin{array}{cccccc} 0^\bullet & 0 & & & & & & \\ 0 & 0^\bullet & & & & & & \\ 2 & 1 & 0^\bullet & 0 & & & & \\ 1 & 2 & 0 & 0^\bullet & & & & \\ 1 & 1 & 0 & 2 & 0^\bullet & & & \\ & & 0 & & 0 & 0^\bullet & & \end{array} \right) & \begin{array}{l} 4 \\ 4 \\ 2 \\ 2 \\ 0 \\ 0 \end{array} \\ H \\ D \\ F \\ E \\ K \\ d_j \end{array} & , & \begin{array}{c} \begin{array}{cccccc} & x_1^{(4)} & x_3^{(4)} & \ddot{w} & \ddot{x}_2 & u_2 & u_1 \\ G^{(4)} & \left(\begin{array}{cccccc} G_{x_1} & G_{x_3} & 0 & 0 & 0 & 0 \\ H_{x_1} & H_{x_3} & 0 & 0 & 0 & 0 \\ \ddot{D} & D_{\ddot{x}_1} & 0 & D_w & D_{x_2} & 0 & 0 \\ \ddot{F} & 0 & F_{\ddot{x}_3} & F_w & F_{x_2} & 0 & 0 \\ E & 0 & 0 & 0 & E_{\ddot{x}_2} & E_{u_2} & 0 \\ K & 0 & 0 & 0 & 0 & K_{u_2} & K_{u_1} \end{array} \right) & \begin{array}{l} 4 \\ 4 \\ 2 \\ 2 \\ 0 \\ 0 \end{array} \\ H^{(4)} \\ \ddot{D} \\ \ddot{F} \\ E \\ K \end{array} \end{array} \end{array}$$

For later observations we note Σ has four coarse (also fine) blocks, two of size 2×2 and two of size 1×1 , although for the time being we will treat it as having only one block. Working through the DD algorithm yields $\boldsymbol{\nu} = (4, 4, 2, 2, 0, 0)$, and thus the differentiated system \mathcal{G} is:

$$\begin{aligned} G^{(4)} &= 0, & H^{(4)} &= 0, \\ D^{(2)} &= 0, & F^{(2)} &= 0, \\ E &= 0, & K &= 0. \end{aligned}$$

Stage 0

The vector of HODs is $\mathbf{z}^{[0]} = (x_1^{(4)}, x_3^{(4)}, \ddot{w}, \ddot{x}_2, u_2, u_1)^T$ and $\mathbf{g}^{[0]} = (G^{(4)}, H^{(4)}, D^{(2)}, F^{(2)}, E, K)^T$.

Thus we have a DD Jacobian of the form:

$$\frac{\partial \mathbf{g}^{[0]}}{\partial \mathbf{z}^{[0]}} = G^{[0]} = \begin{array}{c} \begin{array}{ccccccc} & x_1^{(4)} & x_3^{(4)} & \ddot{w} & \ddot{x}_2 & u_2 & u_1 & c_i \\ G^{(4)} & \left(G_{x_1}^{(4)} & G_{x_3}^{(4)} & 0 & 0 & 0 & 0 \right) & 4 \\ H^{(4)} & \left(H_{x_1}^{(4)} & H_{x_3}^{(4)} & 0 & 0 & 0 & 0 \right) & 4 \\ \ddot{D} & \left(\ddot{D}_{x_1}^{(4)} & 0 & \ddot{D}_{\ddot{w}} & \ddot{D}_{\ddot{x}_2} & 0 & 0 \right) & 2 \\ \ddot{F} & \left(0 & \ddot{F}_{x_3}^{(4)} & \ddot{F}_{\ddot{w}} & \ddot{F}_{\ddot{x}_2} & 0 & 0 \right) & 2 \\ E & \left(0 & 0 & 0 & E_{\ddot{x}_2} & E_{u_2} & 0 \right) & 0 \\ K & \left(0 & 0 & 0 & 0 & K_{u_2} & K_{u_1} \right) & 0 \end{array} \\ d_j & 4 & 4 & 2 & 2 & 0 & 0 \end{array}.$$

By Griewank's Lemma 2.6.2 this is equivalent to \mathbf{J} . Removing equations with $c_i = 0$ yields:

$$H^{[0]} = \begin{array}{c} \begin{array}{ccccccc} & x_1^{(4)} & x_3^{(4)} & \ddot{w} & \ddot{x}_2 & u_2 & u_1 & c_i \\ G^{(4)} & \left(G_{x_1}^{(4)} & G_{x_3}^{(4)} & 0 & 0 & 0 & 0 \right) & 4 \\ H^{(4)} & \left(H_{x_1}^{(4)} & H_{x_3}^{(4)} & 0 & 0 & 0 & 0 \right) & 4 \\ \ddot{D} & \left(\ddot{D}_{x_1}^{(4)} & 0 & \ddot{D}_{\ddot{w}} & \ddot{D}_{\ddot{x}_2} & 0 & 0 \right) & 2 \\ \ddot{F} & \left(0 & \ddot{F}_{x_3}^{(4)} & \ddot{F}_{\ddot{w}} & \ddot{F}_{\ddot{x}_2} & 0 & 0 \right) & 2 \end{array} \\ d_j & 4 & 4 & 2 & 2 & 0 & 0 \end{array}.$$

Stage 1

We are now forced to remove the last two columns of $H^{[0]}$ to get a non-singular matrix, choosing $x_1^{(4)}$, $x_3^{(4)}$, \ddot{w} , \ddot{x}_2 as DDs and reducing the order of differentiation:

$$G^{[1]} = \begin{array}{c} \begin{array}{cccc} & x_1^{(4)} & x_3^{(4)} & \ddot{w} & \ddot{x}_2 & c_i \\ G^{(4)} & \left(G_{x_1}^{(4)} & G_{x_3}^{(4)} & 0 & 0 \right) & 4 \\ H^{(4)} & \left(H_{x_1}^{(4)} & H_{x_3}^{(4)} & 0 & 0 \right) & 4 \\ \ddot{D} & \left(\ddot{D}_{x_1}^{(4)} & 0 & \ddot{D}_{\ddot{w}} & \ddot{D}_{\ddot{x}_2} \right) & 2 \\ \ddot{F} & \left(0 & \ddot{F}_{x_3}^{(4)} & \ddot{F}_{\ddot{w}} & \ddot{F}_{\ddot{x}_2} \right) & 2 \end{array} \\ d_j & 4 & 4 & 2 & 2 \end{array}, \quad H^{[1]} = \begin{array}{c} \begin{array}{cccc} & x_1^{(3)} & x_3^{(3)} & \dot{w} & \dot{x}_2 & c_i \\ G^{(3)} & \left(G_{x_1}^{(3)} & G_{x_3}^{(3)} & 0 & 0 \right) & 3 \\ H^{(3)} & \left(H_{x_1}^{(3)} & H_{x_3}^{(3)} & 0 & 0 \right) & 3 \\ \dot{D} & \left(\dot{D}_{x_1}^{(3)} & 0 & \dot{D}_{\dot{w}} & \dot{D}_{\dot{x}_2} \right) & 1 \\ \dot{F} & \left(0 & \dot{F}_{x_3}^{(3)} & \dot{F}_{\dot{w}} & \dot{F}_{\dot{x}_2} \right) & 1 \end{array} \\ d_j & 3 & 3 & 1 & 1 \end{array}.$$

Stage 2

Since $H^{[1]}$ is square $G^{[2]} = H^{[1]}$. As $c_i - \kappa = 0$ no rows (and therefore no columns) are removed at this stage, so that:

$$H^{[2]} = \begin{array}{c} \ddot{x}_1 \quad \ddot{x}_3 \quad w \quad x_2 \quad c_i \\ \ddot{G} \begin{pmatrix} \ddot{G}_{\ddot{x}_1} & \ddot{G}_{\ddot{x}_3} & 0 & 0 \\ \ddot{H} \begin{pmatrix} \ddot{H}_{\ddot{x}_1} & \ddot{H}_{\ddot{x}_3} & 0 & 0 \\ D \begin{pmatrix} D_{\ddot{x}_1} & 0 & D_w & D_{x_2} \\ F \begin{pmatrix} 0 & F_{\ddot{x}_3} & F_w & F_{x_2} \end{pmatrix} \\ d_j \quad 2 \quad 2 \quad 0 \quad 0 \end{pmatrix} \end{pmatrix} \end{pmatrix} \end{pmatrix} \cdot \end{array}$$

Stage 3

Again, $H^{[2]}$ is already square so $G^{[3]} = H^{[2]}$.

$$H^{[3]} = \begin{array}{c} \dot{x}_1 \quad \dot{x}_3 \quad w \quad x_2 \quad c_i \\ \dot{G} \begin{pmatrix} \dot{G}_{\dot{x}_1} & \dot{G}_{\dot{x}_3} & 0 & 0 \\ \dot{H} \begin{pmatrix} \dot{H}_{\dot{x}_1} & \dot{H}_{\dot{x}_3} & 0 & 0 \end{pmatrix} \\ d_j \quad 1 \quad 1 \quad 0 \quad 0 \end{pmatrix} \end{pmatrix} \cdot \end{array}$$

Stage 4

Then:

$$G^{[4]} = \begin{array}{c} \dot{x}_1 \quad \dot{x}_3 \quad c_i \\ \dot{G} \begin{pmatrix} \dot{G}_{\dot{x}_1} & \dot{G}_{\dot{x}_3} \\ \dot{H} \begin{pmatrix} \dot{H}_{\dot{x}_1} & \dot{H}_{\dot{x}_3} \end{pmatrix} \end{pmatrix} \\ d_j \quad 1 \quad 1 \end{pmatrix} \cdot \end{array}$$

Finally we get $H^{[4]} = []$ and the algorithm terminates. By Griewank's Lemma 2.6.2 we have $G^{[-k]} = \mathbf{J}_k$ for all k between -4 and 0 inclusively. We list a comparison between the SA and DD algorithms in Table 2.

The DDs are equivalent to the differentiated variables solved for at each prior stage in SA as expected and we have no choice in selecting them, due to the 0 DOF in this example.

TABLE 2. DDs and SA stages for the robot arm.

DD stage	SA stage	Equations being used	Variables being found	DDs selected
4	-4	G, H	x_1, x_3	x'_1, x'_3
3	-3	\dot{G}, \dot{H}	\dot{x}_1, \dot{x}_3	x''_1, x''_3
2	-2	\ddot{G}, \ddot{H}, D, F	$\ddot{x}_1, \ddot{x}_3, w, x_2$	$x^{(3)}_1, x^{(3)}_3, w', x'_2$
1	-1	$G^{(3)}, H^{(3)}, \dot{D}, \dot{F}$	$x^{(3)}_1, x^{(3)}_3, \dot{w}, \dot{x}_2$	$x^{(4)}_1, x^{(4)}_3, w'', x''_2$
0	0	$G^{(4)}, H^{(4)}, \ddot{D}, \ddot{F}, E, K$	$x^{(4)}_1, x^{(4)}_3, \ddot{w}, \ddot{x}_2, u_2, u_1$	N/A

3.4.3. Structurally Necessary Dummy Derivatives. Because the equations used in each equivalent stage of DDs and SA are the same and variables in each DD stage are a subset of those in the SA stage we have the following theorem:

THEOREM 3.4.8. *If there are an equal number of variables n_k and equations m_k used at stage k in the SA then we will have no choice when finding $G^{[-k+1]}$ in the DD scheme, i.e. $H^{[k]}$ is square.*

That is, if there is a stage in the SA that introduces no degrees of freedom then we can use that stage to find some dummy derivatives without carrying out the algorithm:

COROLLARY 3.4.9 (Structurally Necessary Dummy Derivatives). *If $m_k = n_k$ in the SA scheme at stage k then all subsequent derivatives of variables in $z^{[k]}$ used by the DD scheme at stage $\kappa = -k$ must be DDs in the final index 1 system, i.e. $D_{\mathbf{z}^{[k]}}, \dots, D^{(-k)}_{\mathbf{z}^{[k]}}$ must be DDs. We call such DDs structurally necessary.*

More precisely we have the following:

THEOREM 3.4.10. *If there exists a k such that $m_k = n_k$ and $d_j - k \geq 0$ then $x_j^{(d_j - k + 1)}, \dots, x_j^{(d_j)}$ must be DDs irrespective of the numerical values in the Jacobian (i.e. irrespective of the choice of index 1 system) and are hence termed structurally necessary dummy derivatives.*

This gives us the following improved DD algorithm, where we can identify structurally necessary DDs (unless there are 0 DOF a these will only be a subset of all DDs needed to give an index 1 formulation) without computing numerically the Jacobians:

Algorithm 4 $m_K = n_K$ Algorithm

- 1: **for** $K = k_{\min} : -1$
 - 2: Find SA solution scheme
 - 3: Note stages where $m_K = n_K$
 - 4: Make subsequent derivatives of such variables DDs
 - 5: **for** $K = 0 : -k_{\min}$
 - 6: Work through DDs algorithm, but:
 - 7: keep columns for already known DDs from step 4 when finding $G^{[K]}$
-

For 0 DOF systems this identifies all DDs as one might expect. Clearly Algorithm 4 finds all structurally necessary DDs. Applying the first half of 4 (finding structurally necessary DDs) to the robot arm gives us the following staged solution scheme, produced by a prototype extension (not yet released) to the authors' code DAESA [31]:

Do a pass through the Structural Analysis scheme:

K = -4: Make the following derivatives into dummy derivatives

$x1', x1'', x1''', x1''''$, $x2', x2'', x2''', x2''''$;

K = -3: No dummy derivatives can be discovered structurally at this stage;

K = -2: Make the following derivatives into dummy derivatives

$x3', x3'', x4', x4''$;

K = -1: No dummy derivatives can be discovered structurally at this stage.

One should compare this with Table 2 to convince themselves of the result.

EXAMPLE 3.4.11. Recall the modified double pendula DAE (44) and the associated signature matrix and offsets in Figure 3. Table 3 gives the SA solution stages for this problem. In Table 3 we see all higher order derivatives of variables found at stage 4 in DDs will be made DDs (i.e. those used in SA stages $-3, -2$ and -1 or DD stages 1, 2 and 3). Consider now the different BTFs as shown previously in Figure 3. Applying Corollary 3.4.9 corresponds to solving the first coarse block as a stand alone system and then using it to solve the second coarse block. Compare this with the $\mathbf{z}^{[\kappa]}$ and $\mathbf{g}^{[\kappa]}(\mathbf{z}^{[\kappa]})$ found in DDs

TABLE 3. SA stages for equation (44).

SA stage	m_k	n_k
-6	1	2
-5	1	2
-4	3	3
-3	3	4
-2	4	5
-1	4	5
0	6	6

in Table 4 (we have re-ordered equations and variables so they correspond with the coarse block ordering). Note we get the same result applying our DAESA function to the problem:

Dummy Derivative solution scheme for 'modified2pendula' problem

Do a pass through the Structural Analysis scheme:

k = -6: No dummy derivatives can be discovered structurally at this stage;

k = -5: No dummy derivatives can be discovered structurally at this stage;

k = -4: Make the following derivatives into dummy derivatives

$$x_1''''', x_1''''', x_1^{(5)}, x_1^{(6)}, x_2''''', x_2''''', x_2^{(5)}, x_2^{(6)}, x_3', x_3'', x_3''', x_3''''';$$

k = -3: No dummy derivatives can be discovered structurally at this stage;

k = -2: No dummy derivatives can be discovered structurally at this stage;

k = -1: No dummy derivatives can be discovered structurally at this stage;

k = 0: No dummy derivatives can be discovered structurally at this stage.

TABLE 4. Dummy derivative stages for equation (44).

Dummy derivative stage κ	$\mathbf{z}^{[\kappa]}$	$\mathbf{g}^{[\kappa]}(\mathbf{z}^{[\kappa]})$
0	$(x_1^{(6)}, x_2^{(6)}, x_3^{(4)}, x_5^{(3)}, \ddot{x}_4, x_6)$	$(f_3^{(6)}, f_1^{(4)}, f_2^{(4)}, \ddot{f}_6, f_4, f_5)^T$
1	$(x_1^{(5)}, x_2^{(5)}, x_3^{(3)}, \ddot{x}_5)$	$(f_3^{(5)}, f_1^{(3)}, f_2^{(3)}, \dot{f}_6)^T$
2	$(x_1^{(4)}, x_2^{(4)}, \ddot{x}_3, \dot{x}_5)$	$(f_3^{(4)}, \ddot{f}_1, \ddot{f}_2, f_6)^T$
3	$(x_1^{(3)}, x_2^{(3)}, \dot{x}_3, x_5)$	$(f_3^{(3)}, \dot{f}_1, \dot{f}_2)^T$
4	$(\ddot{x}_1, \ddot{x}_2, x_3)$	$(\ddot{f}_3, f_1, f_2)^T$
5	(\dot{x}_1)	$(\dot{f}_3)^T$
6	(x_1)	$(f_3)^T$

Due to our ordering in the DD algorithm we introduce DDs for $D\mathbf{z}^{[\kappa]}$ at stage κ , e.g. at stage 4 we are left with the 3×3 system given by the first coarse block in our BTF as expected. We note this algorithm for structurally necessary DDs looks similar to solving for DDs based on the coarse BTF, see §3.5.1 for why this is not quite the case.

EXAMPLE 3.4.12. This improved DD algorithm does indeed achieve our goal of reducing the total number of potentially needed index 1 systems: Consider again the DAE (44), working through the structural analysis we see that at stage $k = -4$ we have $m_k = n_k$. At stage -4 we are solving for $\ddot{x}_1, \ddot{x}_2, x_3$, so we know to keep columns corresponding to these variables when working through DDs. For example, with \bullet indicating a structural non-zero and a blank indicating a structural zero we have $G^{[0]}$ and $H^{[0]}$:

$$\begin{array}{c}
 \begin{array}{cccccc}
 & x_1^{(6)} & x_2^{(6)} & x_3^{(4)} & \ddot{x}_4 & x_5^{(3)} & x_6 & c_i \\
 f_1^{(4)} & \bullet & & \bullet & & & & 4 \\
 f_2^{(4)} & & \bullet & \bullet & & & & 4 \\
 f_3^{(6)} & \bullet & \bullet & & & & & 6 \\
 f_4 & & & & \bullet & & \bullet & 0 \\
 f_5 & & & & & \bullet & \bullet & 0 \\
 f_6 & & & \bullet & \bullet & & & 2 \\
 d_j & 6 & 6 & 4 & 2 & 3 & 0 &
 \end{array}
 \end{array}
 \quad
 \begin{array}{c}
 \begin{array}{cccccc}
 & x_1^{(6)} & x_2^{(6)} & x_3^{(4)} & \ddot{x}_4 & x_5^{(3)} & x_6 & c_i \\
 f_1^{(4)} & \bullet & & \bullet & & & & 4 \\
 f_2^{(4)} & & \bullet & \bullet & & & & 4 \\
 f_3^{(6)} & \bullet & \bullet & & & & & 6 \\
 \ddot{f}_6 & & & \bullet & \bullet & & & 2 \\
 d_j & 6 & 6 & 4 & 2 & 3 & 0 &
 \end{array}
 \end{array}
 .$$

We must keep the first three columns and hence only have to check 3 matrices for non-singularity (in practice for best condition number), as opposed to the 15 we would otherwise have to check—although clearly in this case inspection tells us we choose the first 4 columns.

3.5. Alternative Algorithms

3.5.1. Using BTFs. Using a block decomposition may yield a way of reducing the size of potential $G^{[\kappa]}$ matrices at each stage, which should offer computational speed up when checking the condition number of each Jacobian when doing dummy pivoting. Before giving an algorithm for finding DDs on blocks we ask if Algorithm 4 was already doing something similar to a BTF for us. We consider a fine block decomposition (which is itself a BTF

within the coarse BTF, see §2.7) and ask if we could further reduce potential index 1 choices when considering m_k being equal to n_k during our SA stages.

THEOREM 3.5.1. *If given an $n \times n$ DAE and there exists a $k \in \{k_{\min}, \dots, -1\}$ such that $n_k = m_k = \mu$ for some $0 < \mu < n$, then the DAE must decompose into 2 coarse blocks of size μ and $(n - \mu)$ (it may decompose further).*

PROOF. Similar to the proof of Lemma 3.4.2 one partitions the matrix into the following and then notes the top right block is empty (i.e. all its entries are $-\infty$) by (34).

$$\Sigma = \left[\begin{array}{ccc|ccc} & & & & & & 1 \\ & & & & & & \vdots \\ & & & & & & \\ & & & & & & \mu \\ \hline & & & & & & \mu + 1 \\ & & & & & & \vdots \\ & & & & & & n \\ 1 & \dots & \mu & \mu + 1 & \dots & n & \end{array} \right].$$

□

This means, we cannot have a coarse block irreducible DAE with $n_k = m_k$, unless $k \geq 0$ or $k < k_{\min}$ and therefore $n_k = m_k$ never occurs when solving via fine blocks unless the DAE has m fine blocks and we are solving fine block m , i.e. are at global stage 0. Hence we turn our attention to the interactions between blocks rather than trying to optimise further within a block. We define local offsets associated with an arbitrary block form:

DEFINITION 3.5.2 (Block Local Offsets). We denote the canonical local offsets associated with an arbitrary block form, were its blocks treated as stand alone systems, as \check{c}_i and \check{d}_j .

Note: the local offsets $\hat{\mathbf{c}}$ and $\hat{\mathbf{d}}$ are the block local offsets for the fine BTF. Given a block form of Σ with L blocks we have Algorithm 5. In the following discussion we will call DDs

Algorithm 5 Block based Dummy Derivatives

```

1: for  $l = 1 : L$ 
Initialise:  $z_l^{[0]}, g_l^{[0]}(z_l^{[0]})$ ,  $\kappa_l = 0$ 
2:   if  $j = 0$ 
3:      $G_l^{[0]} = \frac{\partial g_l^{[0]}}{\partial z_l^{[0]}}$ 
4:     Let  $m$  be the number of differentiated equations in  $g_l^{[0]}(z_l^{[0]})$ 
5:     Let  $H_l^{[0]}$  be the first  $m$  rows of  $G_l^{[0]}$ 
6:      $\kappa = \kappa + 1$ 
7:   else
8:     while  $H_l^{[\kappa_l-1]} \neq [ ]$ 
9:       Let  $G_l^{[\kappa_l]}$  be  $m$  columns of  $H_l^{[\kappa_l-1]}$  such that we have a non-singular matrix
10:      Make the corresponding variables used in  $G_l^{[\kappa_l]}$  DDs
11:      Omit one differentiation to get  $z_l^{[\kappa_l]}, g_l^{[\kappa_l]}(z_l^{[\kappa_l]}), \dots$ 
12:      (where we only consider variables and equations in  $G_l^{[\kappa_l]}$ )
13:      Let  $m$  be the number of differentiated equations in  $g_l^{[\kappa_l]}(z_l^{[\kappa_l]})$ 
14:      Let  $H_l^{[\kappa_l]}$  be the first  $m$  rows of  $G_l^{[\kappa_l]}$ , ...
15:      (the rows using differentiated equations in  $g_l^{[\kappa_l]}(z_l^{[\kappa_l]})$ )
16:       $\kappa_l = \kappa_l + 1$ 
17:      Consider the new system using all equations  $g_l^{[\kappa_l]}(z_l^{[\kappa_l]})$ , where  $\kappa_l \geq 0, \dots$ 
18:      and dummy derivatives for  $z_l^{[\kappa_l]}$ , where  $\kappa_l > 0$  and original variables, ...
19:      as well as all equations  $g_l^{(\hat{\mathbf{c}})}, \dots, g_l^{(\mathbf{c})}, \dots$ 
20:      and dummy derivatives for all variables  $x_j^{(\check{d}_j)}, \dots, x_j^{(d_j)}$ .

```

block necessary dummy derivatives if they are found at the end of Algorithm 5, i.e. if they can be found by using a block form's local offsets and the global canonical offsets. There is no explicit interaction between blocks in Algorithm 5 (the inter block dependencies are taken into account on lines 19 and 20) and therefore the majority of the algorithm could be performed in parallel. In our discussions that follow we will restrict ourselves to thinking about coarse and fine BTFs, as described in §2.7, as these tend to be the most natural when using the signature matrix method.

To prove that Algorithm 5 gives a suitable choice of DDs, we would like an analogue of a theorem in [48] that asserts the difference between local and global offsets is constant on

a fine block form to hold for an arbitrary block form. This would mean that we introduce only as many DDs as differentiated equations when taking the interactions between blocks into account. Consider however the following example:

EXAMPLE 3.5.3. Consider a DAE with the following signature matrix:

$$\begin{array}{c} \begin{array}{cc} & c_i \quad \check{c}_i \\ \left(\begin{array}{c|cc} 0 & 1 & \\ \hline & 0 & 0 \\ & 0 & 1 \end{array} \right) & 0 & 0 \\ & 1 & 0 \\ & 0 & 0 \end{array} \\ d_j \quad 0 \quad 1 \quad 1 \\ \check{d}_j \quad 0 \quad 0 \quad 1 \end{array} .$$

Here we are block triangularising over coarse blocks. There is not a constant difference between coarse local and global offsets, so it's possible we introduce more DDs than we do equations when using Algorithm 5.

The potential issue in Example 3.5.3 is actually not a problem at all due to the following theorem (if it were, then Algorithm 5 would not be a valid method for producing DD schemes):

THEOREM 3.5.4. *Given a BTF of Σ the difference between the sum of any block's local offsets and global offsets is equal with respect to \mathbf{c} and \mathbf{d} .*

PROOF. Take an $n \times n$ signature matrix Σ and put its HVT on the main diagonal. Let S be any subset of $\{1, \dots, n\}$ and $\check{\mathbf{c}}$ and $\check{\mathbf{d}}$ be any valid offsets. Then, since $\check{d}_i - \check{c}_i = \sigma_{i,i}$ we have:

$$\sum_{i \in S} \check{d}_i - \sum_{i \in S} \check{c}_i = \sum_{i \in S} \sigma_{i,i} = \sum_{i \in S} d_i - \sum_{i \in S} c_i = \text{Val}(S),$$

which is independent of $\check{\mathbf{c}}$ and $\check{\mathbf{d}}$. So, for any other valid offsets, say \mathbf{c} and \mathbf{d} we have:

$$\sum_{i \in S} \check{d}_i - \sum_{i \in S} d_i = \sum_{i \in S} \check{c}_i - \sum_{i \in S} c_i.$$

□

This is not what one might first expect: in our coarse block algorithm one might expect to differentiate an entire block a number of times to solve a later block, this theorem shows that actually you may only need to differentiate some parts of the block to retain a square index 1 system using lines 19 and 20 in Algorithm 5, a subtlety in the choice of block form not explored in [30].

THEOREM 3.5.5. *Algorithm 5 gives a suitable choice of DDs that could otherwise have been found by considering the entire system and original DD algorithm.*

PROOF. If one is able to carry out the above algorithm then we have a block form whose diagonal sub-matrices are structurally non-singular. Because each block's coarse local offsets are a constant away from the global, by Lemma 2.6.2 we must have a non-singular Jacobian at each global stage of DDs if we have a non-singular Jacobian at each coarse local stage. Again, by Lemma 2.6.2 we see that a valid choice for the global stage DD algorithm is an amalgam of variables found at positive and negative local stages, using each block's lead times (if we consider adding variables and equations at the end of the algorithm as doing local stages $0, \dots, \max_j (d_j - \check{d}_j)$). \square

Due to Theorem 3.5.1 and Example 3.4.7 one may think when restricting to coarse blocks Algorithm 5 is just a restating of Algorithm 4—consider a DAE with signature matrix:

$$\begin{array}{cccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & c_i \\
 f_1 & \left(\begin{array}{ccc|ccc}
 5 & 0 & & & & & & 0 \\
 & & 0 & 4 & & & & 0 \\
 0 & & 0 & & & & 0 & 4 \\
 \hline
 & & & & 0 & 0 & & 6 \\
 & & & & & 2 & 0 & 4 \\
 & & & & & & 0 & 4 \\
 \hline
 & & & & 2 & & 0 & 4
 \end{array} \right) & , & \begin{array}{|c|c|c|c|c|c|}
 \hline
 k & -6 & -5 & -4 & -3 & -2 & -1 \\
 \hline
 m_k & 1 & 1 & 4 & 4 & 4 & 4 \\
 \hline
 n_k & 2 & 3 & 5 & 5 & 5 & 5 \\
 \hline
 \end{array} . \\
 d_j & 5 & 0 & 4 & 6 & 6 & 4 &
 \end{array}$$

Looking for stages where $m_k = n_k$ we find no structurally necessary DDs. However, using Algorithm 5 we find block necessary DDs for $x_4^{(3)}, \dots, x_4^{(6)}, x_5^{(3)}, \dots, x_5^{(6)}$ and $\dot{x}_6, \dots, x_6^{(4)}$.

We see that Algorithm 5 restricted to the coarse BTF yields a valid set of DDs that could be found using the global offsets and no BTF. All we've done is take a BTF, so one might assume in general it generates all possible sets of valid DDs that could be obtained globally, as certainly seems to be asserted in the original paper [29]. Consider however:

EXAMPLE 3.5.6. Consider a DAE with signature tableau as follows:

$$\Sigma = \begin{array}{c} \begin{array}{cccccc} & x_1 & x_2 & x_3 & x_4 & x_5 & c_i \\ f_1 & \left(\begin{array}{cc|ccc} 1 & 0 & & & 0 \\ 2 & 1 & & & \end{array} \right) & 1 \\ f_2 & & & & & & 0 \\ f_3 & & & 1 & 0 & & 2 \\ f_4 & & & 2 & 1 & 0 & 1 \\ f_5 & & & & 2 & 1 & 0 \end{array} \\ d_j & 2 & 1 & 3 & 2 & 1 \end{array} .$$

Note that the coarse BTF given above is the same as the fine BTF and $\mathbf{c} = \check{\mathbf{c}} = \hat{\mathbf{c}}$ and $\mathbf{d} = \check{\mathbf{d}} = \hat{\mathbf{d}}$. Algorithm 5 will give a DD for either \ddot{x}_1 or \dot{x}_2 from the first block, that is Algorithm 5 cannot find a set of DDs where neither of \ddot{x}_1 and \dot{x}_2 are selected. However, if we carry out the original DD algorithm we see that there is an index one system that uses only derivatives of variables x_3 , x_4 and x_5 as DDs. Thus there are index 1 systems we may 'miss' using a BTF that would otherwise be available if considering the DAE as a whole.

We now show that the choice of block form does directly affect the number of block necessary DDs. We assert the fine BTF is a good choice. Before giving an example of Algorithm 5 using the fine BTF we wonder if there exists a more informative version of Algorithm 4 based on the fine BTF. Consider again equation (44), its signature matrices in Figure 3 and its stages in Table 4. For stages -2 and -1 $m_k \neq n_k$ because we have to introduce initial values for \ddot{x}_5 and \dot{x}_5 . If we did not have to do this we would again have a square system, this time with 4 equations—the original coarse block containing equations 1, 2, 3 and variables 1, 2, 3 and a fine block containing equation 6 and variable 4. We note that variable 5 does not actually appear in the DD solution scheme until stage 0 because

it does not have an associated equation. Note: because they will not appear in the DD scheme we can ignore any variables that must be specified as IVs by the SA when checking for square systems. This gives rise to the following definition

DEFINITION 3.5.7. Fine block local numbers of equations and variables are written as:

$$\widehat{m}_{k,l} = \left| \{j \text{ in block } l \mid \widehat{d}_j + k \geq 0\} \right|, \quad \widehat{n}_{k,l} = \left| \{i \text{ in block } l \mid \widehat{c}_i + k \geq 0\} \right|,$$

where k is taken to be a local SA stage associated with a fine block.

Doing this we end up with an improved fine block based analogue of Algorithm 4. For each fine block l do the following (where $\widehat{k}_{\min,l}$ is the $-\max_j d_j$ such that j is in block l):

Algorithm 6 Fine BTF based structural DD algorithm

- 1: **for** $K = \widehat{k}_{\min,l} : -1$
 - 2: Find SA solution scheme
 - 3: Note stages where $\widehat{m}_{k,l} = \widehat{n}_{k,l}$
 - 4: Make subsequent derivatives of such variables DDs
 - 5: **for** $K = 0 : -\widehat{k}_{\min,l}$
 - 6: Work through DDs algorithm, but:
 - 7: keep columns relating to already known DDs from step 4 when finding $G^{[K]}$
-

Knowing this it motivates us to continue using the fine BTF to find DDs, since we have a better set of structurally necessary DDs (we will term such DDs *fine block structurally necessary DDs*) than was previously found using the entire signature matrix. Before proving this is indeed a valid method we illustrate it on a double pendulum:

EXAMPLE 3.5.8. Consider again Equation (44). We carry out Algorithm 5 on this DAE to illustrate the advantages of the fine BTF. The local offsets tell us that we must have DDs shown in Table 5—found by comparing the offsets via lines 19 and 20 of the Algorithm. The only block we have to select DDs in is block 4 (equivalent to the simple pendulum). For ease of checking the Jacobians that follow we now consider this as a stand alone DAE (as would

TABLE 5. DDs from the fine blocks for example (44).

<i>Block Number</i>	<i>Fine block structural DDs</i>
1	No dummy derivatives for this block
2	No dummy derivatives for this block
3	\dot{x}_4, \ddot{x}_4
4	$x_1^{(3)}, x_1^{(4)}, x_1^{(5)}, x_1^{(6)}, x_2^{(3)}, x_2^{(4)}, x_2^{(5)}, x_2^{(6)}, \dot{x}_3, \ddot{x}_3, x_3^{(3)}, x_3^{(4)}$

be done should Algorithm 5 be done in parallel over fine blocks) and give the differentiated problem $\mathcal{G}x = 0$:

$$\begin{cases} f_1 = \ddot{x}_1 + x_1 x_3 = 0, \\ f_2 = \ddot{x}_2 + x_2 x_3 - G = 0, \\ f_3 = x_1^2 + x_2^2 - L^2 = 0, \end{cases} \quad \begin{cases} \dot{f}_3 = f_4 = 2\dot{x}_1 x_1 + 2\dot{x}_2 x_2 = 0, \\ \ddot{f}_3 = f_5 = 2\ddot{x}_1 x_1 + 2\dot{x}_1^2 + 2\ddot{x}_2 x_2 + 2\dot{x}_2^2 = 0. \end{cases}$$

Performing the DD algorithm gives two possible index 1 systems, where the choice of DDs is given in Table 6. We are able to find 14 block necessary DDs without ever having to

TABLE 6. Possible DD choices from (44).

<i>Stage 1</i>	<i>Stage 2</i>
x_1''	x_1'
x_2''	x_2'

compute a numerical Jacobian and reduced our problem size by half at the outset.

A brief complexity analysis follows. Assume the DAE decomposes into L fine blocks labelled by a subscript l . At each stage of DDs the original algorithm has a complexity of order:

$$\sum_{k=-1}^{k_{\min}} (n_k)^3,$$

because the selection of DDs is usually done via a QR decomposition in practice, which is an $\mathcal{O}(n^3)$ operation. The proposed algorithm has a complexity of order:

$$\sum_{l=1}^L \left(\sum_{k=-1}^{\widehat{k}_{\min,l}} (\widehat{n}_{k,l})^3 \right),$$

where \widehat{c}_i in the second summation is taken to be in block l . Assuming the system decomposes into relatively small fine blocks, i.e. $\widehat{n} \ll n$ and some $\widehat{c}_i < c_i$ —from test models in DAESA this is usually the case—this should offer good numerical speed up.

Importantly the large reduction in potential DDs and problem size makes dummy pivoting less problematic, since we will have to consider smaller systems of equations with a reduced number of potential variables to choose from, see §3.6 for more detail.

Consider again Example 3.5.6, we are reminded that there are index 1 systems we may ‘miss’ using a fine BTF that would otherwise be available if considering the DAE as a whole. Whilst this seems like a rather large oversight of our method, this potential ‘oversight’ will never make an otherwise solvable (by DDs) DAE unsolvable:

THEOREM 3.5.9. *If the DAE is solvable then Algorithm 5 restricted to the fine BTF will always be able to select a valid set of DDs.*

PROOF. If the system is solvable then there must exist DD matrices globally, i.e. there is a choice $G^{[0]}, \dots, G^{[\max_i c_i]}$ for which each matrix is non-singular. We also know that there exist global SA system Jacobian $\mathbf{J}_0, \dots, \mathbf{J}_{-\max_i c_i}$ that have full row rank. If the SA scheme succeeds globally then it succeeds over fine-blocks. That is, over each fine block l there are $\mathbf{J}_{0,l}, \dots, \mathbf{J}_{-\max_i \widehat{c}_{i,l}}$ that have full row rank. Since we have that any $\mathbf{J}_0 = G^{[0]}$ and subsequent DD matrices are sub matrices of previous SA matrices it must be possible to select a non-singular DD matrix on each fine block if each SA Jacobian has full row rank in that block. □

Thus, although it is possible to find non-singular choices of DDs globally that cannot be found over a fine block these choices are somehow a redundant selection—although admittedly it might be possible the global selections have better condition numbers there will exist a fine block selection if the DAE is solvable.

3.5.2. Relation to another approach in the literature. An approach in the literature that seeks to improve upon the Dummy Derivative method is found in [54] and [55] and is termed the *structural approach for regularization*. One begins by adding to the system all equations $\dot{f}_i, \dots, f_i^{(c_i)}$ for each equation i , this gets us a reduced *derivative array*, see [8] or 1.3. We now have a system with $M = \sum_i c_i + n$ equations in only n unknowns, so we have to introduce $\sum_i c_i$ new unknowns we wish to solve for, i.e. the approach is similar to doing DDs in one go, rather than in stages. The method does this by using a HVT, T with entries given by (i, j_i) . It adds new variables as follows:

$$(54) \quad \mathbf{w}_{\mathbf{j}_i} = \begin{pmatrix} w_{j_i}^{(\sigma_{i,j_i}+1)} \\ \vdots \\ w_{j_i}^{(\sigma_{i,j_i}+c_i)} \end{pmatrix} = \begin{pmatrix} x_{j_i}^{(\sigma_{i,j_i}+1)} \\ \vdots \\ x_{j_i}^{(\sigma_{i,j_i}+c_i)} \end{pmatrix} \quad \text{for } c_i > 0 \quad \text{and} \quad \mathbf{w}_{\mathbf{j}_i} = [] \quad \text{for } c_i = 0.$$

Of course this choice of HVT is, in general, non-unique. A weighting is suggested to choose a HVT that is valid in a maximal neighbourhood of a consistent point, so that one may have to change their selection dynamically as one does in dummy pivoting. Each HVT is assigned a local weighting coefficient:

$$\omega_T = \prod_{(i,j) \in T} |J_{i,j}|.$$

Intuitively this method looks very similar to DDs, one seeks to add the same equations to the system and introduce an equivalent number of ‘new’ algebraic variables to solve for that were previously derivatives of our original variables. In fact one can see any index 1 system found by the above approach must also be obtainable using Dummy Derivatives.

LEMMA 3.5.10. *Adding variables to the DAE as in equation (54) produces a valid Dummy Derivative system.*

PROOF. Each entry of Σ that is on a HVT must have a structurally non-singular System Jacobian entry. Consider a DAE that is one irreducible coarse block for simplicity without loss of generality. In DDs if $c_i = 0$ we remove that equation, in this approach if $c_i = 0$ we add no variables to ‘match’ to that equation, essentially removing it from consideration. If $c_i > 0$ we consider the approach above as stage-wise and note since we are using a HVT $d_{j_i} = \sigma_{i,j_i} + c_i$, we add variables for $x_{j_i}^{(\max_{j_i} d_{j_i})}$, then set $d_{j_i} = d_{j_i} - 1$ and iterate until $d_{j_i} = 0$ —which is exactly what we do in DDs. Since we are working on a HVT each equivalent DD Jacobian must be (at least structurally) non-singular, so we have an alternative way to find a DD system. \square

This looks similar to the approach based on a BTF above—we have a method that identifies a potential DD system, so we ask if it can identify all potential DD systems. However, the following example tells us it cannot.

EXAMPLE 3.5.11. Consider a DAE:

$$\begin{cases} f_1 = x_1 + x_2 + tx_3 = 0, \\ f_2 = 3\dot{x}_1 + \dot{x}_2 + x_3 = 0, \\ f_3 = 2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 = 0, \end{cases}$$

which has signature tableau (with HVTs marked by $*$, \circ , \bullet and \triangle):

$$\Sigma = \begin{array}{cccc} & x_1 & x_2 & x_3 & c_i \\ f_1 & \left(\begin{array}{ccc} 0^* & 0^\circ & 0^{\bullet\triangle} \end{array} \right) & 2 & & \\ f_2 & \left(\begin{array}{ccc} 1^{\circ\bullet} & 1^{*\triangle} & 0 \end{array} \right) & 1 & & \\ f_3 & \left(\begin{array}{ccc} 2^\triangle & 2^\bullet & 2^{*\circ} \end{array} \right) & 0 & & \\ d_j & 2 & 2 & 2 & \end{array} .$$

We therefore have initial DD Jacobian:

$$G^{[0]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 \\ \begin{matrix} \ddot{f}_1 \\ \dot{f}_2 \\ f_3 \end{matrix} & \begin{pmatrix} 1 & 1 & t \\ 3 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} \end{matrix}.$$

Which is non-singular provided $2 - t \neq 0$. Proceeding with the algorithm produces:

$$H^{[0]} = \begin{matrix} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 \\ \begin{matrix} \ddot{f}_1 \\ \dot{f}_2 \end{matrix} & \begin{pmatrix} 1 & 1 & t \\ 3 & 1 & 0 \end{pmatrix} \end{matrix}.$$

We therefore have 3 potential choices for $G^{[1]}$, listed in Table 7.

TABLE 7. Possible DD choices for Example 3.5.11.

Choose x''_1 and x''_2	Choose x''_1 and x''_3	Choose x''_2 and x''_3
$G^{[1]} = \begin{matrix} & \dot{x}_1 & \dot{x}_2 \\ \dot{f}_1 \\ f_2 \end{matrix} \begin{pmatrix} 1 & 1 \\ 3 & 1 \end{pmatrix}$	$G^{[1]} = \begin{matrix} & \dot{x}_1 & \dot{x}_3 \\ \dot{f}_1 \\ f_2 \end{matrix} \begin{pmatrix} 1 & t \\ 3 & 0 \end{pmatrix}$	$G^{[1]} = \begin{matrix} & \dot{x}_2 & \dot{x}_3 \\ \dot{f}_1 \\ f_2 \end{matrix} \begin{pmatrix} 1 & t \\ 1 & 0 \end{pmatrix}$

In Table 8 we present all possible index 1 DD systems and indicate which can be found using a HVT above and note there are 2 systems (from 2 different branches) that the method cannot find.

TABLE 8. All DD choices for Example 3.5.11.

Choose x''_1, x''_2, x'_1 HVT: *	Choose x''_1, x''_2, x'_2 HVT: \circ	Choose x''_1, x''_3, x'_1 HVT: N/A
Choose x''_1, x''_3, x'_3 HVT: \bullet	Choose x''_2, x''_3, x'_3 HVT: \triangle	Choose x''_2, x''_3, x'_2 HVT: N/A

EXAMPLE 3.5.12. To further example 3.5.11 consider a DAE with signature matrix, offsets and HVTs (given by $*$, \circ , \bullet and Δ) given below:

$$\Sigma = \begin{array}{cccc} & x_1 & x_2 & x_3 & c_i \\ f_1 & \left(\begin{array}{ccc} 0^* & 0^\circ & 0^{\bullet\Delta} \end{array} \right) & 2 \\ f_2 & \left(\begin{array}{ccc} 1^\Delta & 1^\bullet & 1^{*\circ} \end{array} \right) & 1 \\ f_3 & \left(\begin{array}{ccc} 2^{\circ\bullet} & 2^{*\Delta} & -\infty \end{array} \right) & 0 \\ d_j & 2 & 2 & 2 \end{array} .$$

This has System Jacobian:

$$\mathbf{J} = \begin{array}{ccc} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 \\ \ddot{f}_1 & \left(\begin{array}{ccc} \bullet & \bullet & \bullet \end{array} \right) \\ \dot{f}_2 & \left(\begin{array}{ccc} \bullet & \bullet & \bullet \end{array} \right) \\ f_3 & \left(\begin{array}{ccc} \bullet & \bullet & 0 \end{array} \right) \end{array} = G^{[0]},$$

where \bullet indicates a structural non zero. The method above can only select index 1 systems using either x_1'' and x_3'' or x_2'' and x_3'' , however DDs can select the following $G^{[1]}$:

$$G^{[1]} = \begin{array}{cc} & \dot{x}_1 & \dot{x}_2 \\ \dot{f}_1 & \left(\begin{array}{cc} \bullet & \bullet \end{array} \right) \\ f_2 & \left(\begin{array}{cc} \bullet & \bullet \end{array} \right) \end{array} .$$

Hence we have an index 1 system that cannot be found using the structural approach for regularization, that can be found using DDs.

The following remains an open question: if every choice of index 1 system the above method produces has a singular Jacobian then will every index 1 system it misses also have a singular Jacobian? Clearly for this example this is the case, if columns 1 and 3 and 2 and 3 are linearly dependent then so are columns 1 and 2, we conjecture this is the case in general, which if true would give us a more practical way of finding DDs.

3.6. Dummy Pivoting

We briefly discuss the issue of dummy pivoting as first examined in [30], we give an insight in to how best to visualise the problem and present an example that shows a ‘worst’ case number of potential index 1 systems. When proceeding along a numerical solution it is possible that some chosen matrices $G^{[\kappa]}$ will become singular, [28]. The above examples don’t have time dependent G^κ although there’s no reason a G^κ can’t be time varying—which may lead to singularity at some time t . Consider for example the simple pendulum (2), when proceeding through the DD algorithm 3 we get:

$$H^{[0]} = \ddot{C} \begin{pmatrix} \ddot{x} & \ddot{y} & \lambda \\ 2x & 2y & 0 \end{pmatrix}.$$

We have a choice between selecting either column one or column two. As our solution trajectory approaches the x axis we want to select column one, as it approaches the y axis we want to select column two, assuming our pendulum is moving across the x axis we will have to change our choice of index 1 system dynamically. It is therefore necessary to change our choice of $G^{[\kappa]}$ as our numerical solution evolves with time. Such a change is called dummy pivoting or dynamic selection of states [28]. The main problem with making such a change is that changing a $G^{[\kappa]}$ will (frequently, but not necessarily) produce a need to change all subsequent $G^{[\kappa+i]}$. This problem is worsened the higher the DAE index (there will be multiple stages and thus more potential $G^{[\kappa]}$ to change) or when the larger the problem size (this will result in estimating condition numbers of potentially large matrices frequently throughout the solution process). One method of avoiding the former problem is to start by attempting to reduce the order of the problem and block triangularising as in §3.5, a technique that in practice frequently reduces the order. The solver Dymola seems to do this (it seems to also do behind the scenes algebraic manipulations to reduce the block sizes as well), however it is sometimes not possible to reduce the DAE this way and one can crash the solver due to there being too many matrices to dynamically switch between. If one slightly modifies the example of [51] (see Example 3.6.2) so that every entry in Σ is a structural non-zero then the

DAE with a structural index of 13 (of size 25×25) is enough to crash Dymola (using $> 1\text{GB}$ of system memory before running out of memory [21]), as the number of potential index 1 systems grows with the index (one has to consider more stages). Clearly such a large index is unlikely to occur in a practical example, but less extreme cases are indeed feasible. Another alternative is to select which states must or cannot be chosen as DDs, this is implemented in some tools, such as is done in Dymola with each variable given a `stateSelect` parameter that can be `prefer`, `default` or `avoid`. Selecting dynamic states before solving however needs to be informed by problem specific knowledge, which it is not always possible a user will have. It is also dangerous as the user may avoid choosing structurally necessary DDs and thus not find any index 1 system at all if they are not careful. There are few effective ways to measure when it is necessary to perform dummy pivoting, one such method is to monitor the condition number of $G^{[\kappa]}$, as in [15] or to use an adaptive step size ODE solver and monitor the step-size as in [14] the former potentially being computationally expensive whilst the latter risks dummy pivoting when it is not needed. It is also possible that if the step size is too large one misses points where a pivot was needed and thus has solution regions that are not valid, similar to what happens when checking switching conditions for hybrid systems [59]. The following example offers some insight in to the number of different potential Jacobians $G^{[\kappa]}$.

EXAMPLE 3.6.1.

$$(55) \quad \left\{ \begin{array}{l} f_1 = \ddot{x}_1^2(t) + \dot{x}_5^2(t) + u_1(t) = 0, \\ f_2 = \ddot{x}_4^2(t) + \ddot{x}_2^2(t) + u_2(t) = 0, \\ f_3 = \dot{x}_1^2(t) + x_3^2(t) + u_3(t) = 0, \\ f_4 = x_4^2(t) + x_3^2(t) + u_4(t) = 0, \\ f_5 = \dot{x}_5^2(t) + \dot{x}_2^2(t) + u_5(t) = 0. \end{array} \right.$$

Here $u_1(t), \dots, u_5(t)$ are arbitrary driving functions. The non-linearity gives stages in the DD algorithm that may need pivoting. This DAE (55) has signature tableau and offsets:

$$\Sigma = \begin{array}{cccccc} & x_1 & x_2 & x_3 & x_4 & x_5 & c_i \\ f_1 & \left(\begin{array}{cccccc} 2^\bullet & & & & & 1^\circ \end{array} \right) & 1 \\ f_2 & & \left(\begin{array}{cccccc} & 2^\bullet & & 2^\circ & & \end{array} \right) & 0 \\ f_3 & & & \left(\begin{array}{cccccc} & & 1^\circ & & 0^\bullet & \end{array} \right) & 2 \\ f_4 & & & & \left(\begin{array}{cccccc} & & & 0^\circ & 0^\bullet & \end{array} \right) & 2 \\ f_5 & & & & & \left(\begin{array}{cccccc} & & & & 1^\circ & 1^\bullet \end{array} \right) & 1 \\ d_j & 3 & 2 & 2 & 2 & 2 & \end{array} .$$

Equation (55) is irreducible and has two HVTS, marked by \bullet and \circ . Every entry of Σ corresponds to a structurally non-singular entry in each \mathbf{J}_k and their related DD stage Jacobians. For ease of checking entries in the DD stage Jacobians that follow we present all equations and their derivatives specified by \mathbf{c} :

$$\left\{ \begin{array}{l} f_1 = \ddot{x}_1^2(t) + \dot{x}_5^2(t) + u_1(t) = 0, \\ f_2 = \ddot{x}_4^2(t) + \dot{x}_2^2(t) + u_2(t) = 0, \\ f_3 = \dot{x}_1^2(t) + x_3^2(t) + u_3(t) = 0, \\ f_4 = x_4^2(t) + x_3^2(t) + u_4(t) = 0, \\ f_5 = \dot{x}_5^2(t) + \dot{x}_2^2(t) + u_5(t) = 0, \end{array} \right. \quad \left\{ \begin{array}{l} \dot{f}_1 = f_6 = 2x_1^{(3)}(t)\ddot{x}_1(t) + 2\dot{x}_5(t)\dot{x}_5(t) + \dot{u}_1(t) = 0, \\ \dot{f}_3 = f_7 = 2\dot{x}_1(t)\dot{x}_1(t) + 2\dot{x}_3(t)x_3(t) + \dot{u}_3(t) = 0, \\ \dot{f}_4 = f_8 = 2\dot{x}_4(t)x_4(t) + 2\dot{x}_3(t)x_3(t) + \dot{u}_4(t) = 0, \\ \dot{f}_5 = f_9 = 2\ddot{x}_5(t)\dot{x}_5(t) + 2\ddot{x}_2(t)\dot{x}_2(t) + \dot{u}_5(t) = 0, \end{array} \right.$$

$$\left\{ \begin{array}{l} \ddot{f}_3 = f_{10} = 2x_1^{(3)}(t)\dot{x}_1(t) + 2\ddot{x}_1^2(t) + 2\ddot{x}_3(t)x_3(t) + 2\dot{x}_3^2(t) + \ddot{u}_3(t) = 0, \\ \ddot{f}_4 = f_{11} = 2\ddot{x}_4(t)x_4(t) + 2\dot{x}_4^2(t) + 2\ddot{x}_3(t)x_3(t) + 2\dot{x}_3^2(t) + \ddot{u}_4(t) = 0. \end{array} \right.$$

Stage 0

In the DD algorithm, we have initial Jacobians:

$$G^{[0]} = \begin{array}{c} \begin{array}{cccccc} x_1^{(3)} & \ddot{x}_2 & \ddot{x}_3 & \ddot{x}_4 & \ddot{x}_5 & c_i \end{array} \\ \begin{array}{l} \dot{f}_1 \begin{pmatrix} 2\ddot{x}_1 & 0 & 0 & 0 & 2\dot{x}_5 \end{pmatrix} \\ \dot{f}_2 \begin{pmatrix} 0 & 2\ddot{x}_2 & 0 & 2\ddot{x}_4 & 0 \end{pmatrix} \\ \ddot{f}_3 \begin{pmatrix} 2\dot{x}_1 & 0 & 2x_3 & 0 & 0 \end{pmatrix} \\ \ddot{f}_4 \begin{pmatrix} 0 & 0 & 2x_3 & 2x_4 & 0 \end{pmatrix} \\ \dot{f}_5 \begin{pmatrix} 0 & 2\dot{x}_2 & 0 & 0 & 2\dot{x}_5 \end{pmatrix} \end{array} \\ \begin{array}{cccccc} d_j & 3 & 2 & 2 & 2 & 2 \end{array} \end{array}, \quad H^{[0]} = \begin{array}{c} \begin{array}{ccccc} x_1^{(3)} & \ddot{x}_2 & \ddot{x}_3 & \ddot{x}_4 & \ddot{x}_5 \end{array} \\ \begin{array}{l} \dot{f}_1 \begin{pmatrix} 2\ddot{x}_1 & 0 & 0 & 0 & 2\dot{x}_5 \end{pmatrix} \\ \ddot{f}_3 \begin{pmatrix} 2\dot{x}_1 & 0 & 2x_3 & 0 & 0 \end{pmatrix} \\ \ddot{f}_4 \begin{pmatrix} 0 & 0 & 2x_3 & 2x_4 & 0 \end{pmatrix} \\ \dot{f}_5 \begin{pmatrix} 0 & 2\dot{x}_2 & 0 & 0 & 2\dot{x}_5 \end{pmatrix} \end{array} \end{array}.$$

Stage 1

We need to find $G^{[1]}$. We do this by choosing any 4 columns, since all 5 possibilities are (at least structurally) non-singular. Numerically one potentially has to pivot between different systems, for example say you chose columns 1, 2, 3, 4, and $\dot{x}_2 \rightarrow 0$ at some time t then you would pivot to the system with columns 1, 3, 4, 5. Choose columns 1, 2, 3, 4, giving DDs for $x_1^{(3)}$, \ddot{x}_2 , \ddot{x}_3 and \ddot{x}_4 . The new Jacobians are:

$$G^{[1]} = \begin{array}{c} \begin{array}{cccc} \ddot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 \end{array} \\ \begin{array}{l} f_1 \begin{pmatrix} 2\ddot{x}_1 & 0 & 0 & 0 \end{pmatrix} \\ \dot{f}_3 \begin{pmatrix} 2\dot{x}_1 & 0 & 2x_3 & 0 \end{pmatrix} \\ \dot{f}_4 \begin{pmatrix} 0 & 0 & 2x_3 & 2x_4 \end{pmatrix} \\ f_5 \begin{pmatrix} 0 & 2\dot{x}_2 & 0 & 0 \end{pmatrix} \end{array} \end{array} \quad \text{and} \quad H^{[1]} = \begin{array}{c} \begin{array}{cccc} \ddot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 \end{array} \\ \begin{array}{l} \dot{f}_3 \begin{pmatrix} 2\dot{x}_1 & 0 & 2x_3 & 0 \end{pmatrix} \\ \dot{f}_4 \begin{pmatrix} 0 & 0 & 2x_3 & 2x_4 \end{pmatrix} \end{array} \end{array}.$$

Stage 2

Choose two columns from $H^{[1]}$ to form a square non-singular matrix. There are three choices. As above it may be necessary to pivot chosen DDs numerically. Say we choose columns 1

and 3, then we introduce DDs for \ddot{x}_1 and \dot{x}_3 and get the following Jacobian:

$$G^{[2]} = \begin{matrix} & \dot{x}_1 & x_3 \\ f_3 & \begin{pmatrix} 2\dot{x}_1 & 2x_3 \\ f_4 & 0 & 2x_3 \end{pmatrix} \end{matrix}.$$

Removing undifferentiated equations yields $H^{[2]} = []$ and the algorithm terminates. Note: we have selected the following dummy derivatives: x_1'' , $x_1^{(3)}$, x_2'' , x_3' , x_3'' and x_4'' . If we do not check for non-singular matrices at each stage we would have $\binom{5}{4}\binom{4}{2} = 30$ possible index 1 systems at the end of the algorithm. If however we check for structural singularity at each stage (as is done above) we get 9 possible index 1 systems. Listed in Table 9 are selected DDs.

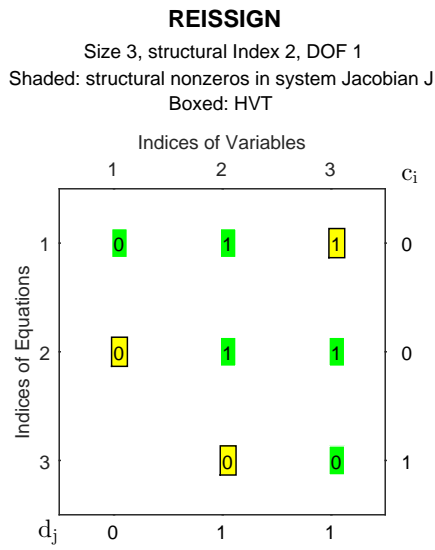
TABLE 9. Possible DD index 1 systems from (55).

<i>Stage 1</i>	<i>Stage 2</i>
$x_1^{(3)}, x_2'', x_3'', x_4''$	x_1'', x_3' or x_1'', x_4' or x_3', x_4'
$x_1^{(3)}, x_2'', x_3'', x_5''$	x_1'', x_3'
$x_1^{(3)}, x_2'', x_4'', x_5''$	x_1'', x_4'
$x_1^{(3)}, x_3'', x_4'', x_5''$	x_1'', x_3' or x_1'', x_4' or x_3', x_4'
$x_2'', x_3'', x_4'', x_5''$	x_3', x_4'

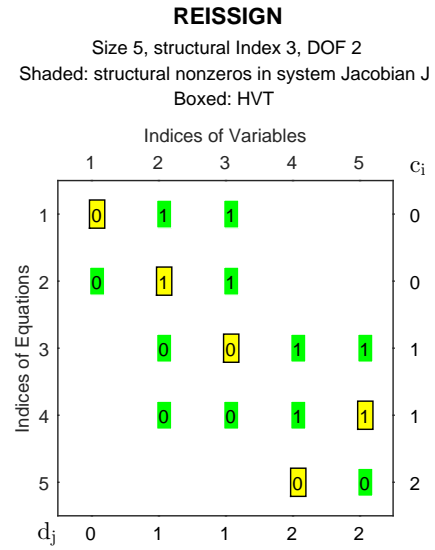
If one thinks of the number of possible DD index 1 systems as a tree diagram, with stage 1 producing the root and each subsequent stage producing branching nodes it is possible to make pivoting easier. In applications where the DAE is usually of large size but (relatively) low index the tree will likely be wide but shallow. Pivoting can then be done across nodes on each level, i.e. you can pivot between all nodes at one level (restricting yourself to nodes coming from one parent node), starting at the node that gives you a singular matrix $G^{[\kappa]}$ and considering its leaves, then if no nodes at the current level give a non singular matrix go up a level and repeat. Storing such a diagram may take a lot of memory, but it's highly likely many nodes will in fact have the same $G^{[\kappa]}$. Consider Table 9. We see there are actually

only three distinct nodes at stage 2, so we only have to store 3 Jacobians, not 9 and check 3 Jacobians for non-singularity in the worst case.

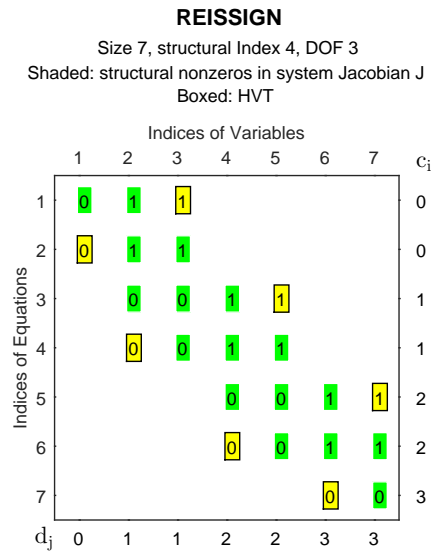
EXAMPLE 3.6.2. We conclude this section by looking at a ‘worst case scenario’. The example from [51] forms a DAE with arbitrarily high structural index (if n is the problem structural index it has size $2n - 1$). Such a DAE will also therefore have arbitrarily many DD stages (A DAE of structural index n will have $n - 1$ DD stages, as each stage reduces the index by 1). We modify the example to have a signature matrix that is dense (all finite entries in Σ are on a HVT), which causes a great deal of choice at each DD stage. Using such a trivial example it’s possible for current DD solvers to break down with relatively small problem size due to running out of system memory. Since the equations are arbitrary for actual DD stages (one assumes they’re picked so that any possible DD selections can be non-singular at some time step) we omit the equations and instead present the first three signature matrices for the process, produced by DAESA in Figure 1. Consider for example the 5×5 DAE of index 3, at stage 1 one has to choose 3 from 4 variables, then at stage 2 one has to choose 1 from 2, giving a total of 8 potential index 1 systems. For the index 4 DAE of size 7×7 one has 6 choices at stage 1, 4 choices at stage 2 and 2 choices at stage 3, giving 48 possible index 1 systems. In general we will have $(2n - 2) \cdot (2n - 4) \cdot \dots \cdot 2$ possible index 1 systems for an index n DAE of this form.



(a) Size 3



(b) Size 5



(c) Size 7

FIGURE 1. Modified Reissig example.

Exploiting Non-Canonical Offsets—Universal Dummy Derivatives

This chapter provides a new algorithm that eliminates the inherent problem of dummy pivoting (see §3.6) by transforming the DAE to a new problem where the same selection of DDs can work throughout the numerical integration. If it's possible to find globally valid DDs for a system that previously only had local ones, we no longer run into problems where we cannot store a sufficient number of index 1 DD systems due to running out of memory (see Example 3.6.2 at the end of §3.6). We can also then exploit structural information (e.g. block forms) throughout our solution, rather than in potentially small regions. The algorithms given in this chapter are only suggested for use if a static selection of DDs cannot be found, either by noting there are 0 DOF, or by finding a set of non-singular $G^{[\kappa]}$ that do not depend on t and are non-singular or by using problem knowledge to find a set of $G^{[\kappa]}$ that depend on t but are still non-singular throughout integration.

4.1. The Basic Algorithm

We would like to consider again the simple pendulum (2), where we rename a, b, c as f_1, f_2, f_3 for clarity later. It is a small physical example that exhibits pivoting so we hope to gain insights into why we need to pivot and how we could avoid it in general by attempting to eliminate pivoting in this problem.

EXAMPLE 4.1.1. We proceed to carry out the DD algorithm 3. Recall from §3.4 that the DD algorithm proceeds in stages, using matrices $G^{[\kappa]}$ and $H^{[\kappa]}$ for $\kappa = 0, 1, \dots$ where $G^{[0]}$ is the $n \times n$ system Jacobian $\mathbf{J} = \partial f_i^{(c_i)} / \partial x_j^{(d_j)}$. Deleting appropriate rows of $G^{[\kappa]}$ gives $H^{[\kappa]}$. Deleting appropriate columns of $H^{[\kappa]}$ to form a nonsingular matrix gives $G^{[\kappa+1]}$. We have an initial Jacobian (with highest order equations and variables) and a secondary non-square

Jacobian:

$$G^{[0]} = \begin{array}{cccc} & \ddot{x} & \ddot{y} & \lambda & c_i \\ f_1 & \left(\begin{array}{ccc} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{array} \right) & & & 0 \\ f_2 & & & & 0 \\ \ddot{f}_3 & & & & 2 \\ d_j & 2 & 2 & 0 & \end{array} \quad \text{and} \quad H^{[0]} = \begin{array}{ccc} \ddot{x} & \ddot{y} & \lambda \\ \ddot{f}_3 & \left(\begin{array}{cc} 2x & 2y \\ & 0 \end{array} \right) & \end{array}.$$

We have to select 1 column from $H^{[0]}$ to get a square non-singular matrix $G^{[1]}$. We cannot choose column 3 because it is structurally 0, as was shown is always the case with columns corresponding to non-differentiated variables in §3.4. We thus either choose column 1 or column 2. We will have to change our choice depending on if x or y are close to 0, in practice this means changing our dummy derivatives whenever the pendulum comes close to crossing an axis. Assuming we have a pendulum that doesn't loop back on itself and starts in say the upper left quadrant this means in one period of oscillation dummy derivatives will have been changed at least five times (start with \ddot{x}, \dot{x}) or potentially as many as 6 times (start with \ddot{y}, \dot{y}). We want to eliminate the need to choose between any variables and instead pick both, since any choice may become invalid and it will be expensive to pivot between potential systems as this happens for a general DAE, see example 3.6.2 at the end of §3.6 for a worst case scenario.

Since we do not have to dummy pivot for DAEs where a static choice of $G^{[\kappa+1]}$ exists or where only one choice for $G^{[\kappa+1]}$ exists, we wish to try to reformulate our problem to be close to the latter case. If we want to eliminate the choice of candidate dummy derivatives (DDs that can be selected as dummy derivatives in at least one DD reduced index 1 formulation) at this stage one way we can do this is by adding equations to the DAE so that we can choose a square submatrix of $H^{[0]}$ that contains the columns corresponding to all candidate dummy derivatives, i.e. \ddot{x} and \ddot{y} . This will still give us multiple choices (e.g. one could pick some subset of old variables and new variables) so we try to add equations that can give us a choice that's valid for all time. Note: introducing new equations and variables and then

choosing the new variables' derivatives to be DDs is counter intuitive to the method whereby we seek to 'match' each differentiated equation with a differentiated variable (if we're just adding equations and matching variables we're not progressing our anti-pivot fix). Since in this example we have one more candidate DD than we do equation we need to add one equation to 'square-up' the system. Since we're at stage $0 \rightarrow 1$ in the DD algorithm our equation will have to have (at least) one differentiation specified by the offsets, i.e. will have a c offset ≥ 1 . To keep the equation simple (and because we don't intend to use the new variable in the DD scheme) we will look for an equation of the form (where f is some as yet unknown function):

$$(56) \quad Z_1 := f(x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, t) - z_1(t) = 0.$$

Noting that we only care about coefficients of \ddot{x} and \ddot{y} in \dot{Z}_1 (these are the only ones that appear in $H^{[0]}$) means that we can actually look for an equation of the form:

$$(57) \quad Z_1 := \bar{f}(\dot{x}, \dot{y}, t) - z_1(t) = 0,$$

where \bar{f} is some as yet unknown function. For simplicity take it to be linear, so we add:

$$(58) \quad Z_1 := \alpha \dot{x} + \beta \dot{y} - z_1(t) = 0$$

to the original DAE, where α and β are some parameters that are yet to be determined and z_1 is some new variable to solve for. We would like the $H^{[0]}$ for our new DAE to be of the form:

$$H^{[0]} = \begin{matrix} & \ddot{x} & \ddot{y} & \lambda & z_1' \\ \ddot{f}_3 & \begin{pmatrix} 2x & 2y & 0 & 0 \\ \alpha & \beta & 0 & -1 \end{pmatrix} \\ \dot{Z}_1 & & & & \end{matrix},$$

where now the parameters α and β can be chosen at run time so the row using new equations (newly introduced equations of the form $Z_i = 0$) in $G^{[1]}$ is roughly orthogonal to the row using old equations (equations part of the original DAE including hidden constraints) in

$G^{[1]}$. We chose (α, β) to be orthogonal to (x, y) so that the resulting DDs matrix $G^{[1]}$ has good condition number. It is now possible to pick x'' and y'' as DDs for all time provided we update α and β along the solution to keep the resulting $G^{[1]}$ matrix non-singular. Checking the condition number of the corresponding G matrix and using a Gram-Schmidt or QR type procedure for ‘new’ rows if the matrix becomes ill conditioned would be a reasonable way of updating α and β dynamically in practice. We have now eliminated the pivot choice at this stage (it will be shown later in the example that we’ve actually just pushed the choice back to a later stage). Consider our new DAEs signature matrix and canonical offsets:

$$\Sigma = \begin{array}{c} \begin{array}{ccccc|c} & x & y & \lambda & z_1 & c_i \\ f_1 & 2^\bullet & -\infty & 0^\circ & -\infty & 0 \\ f_2 & -\infty & 2^\circ & 0^\bullet & -\infty & 0 \\ f_3 & 0^\circ & 0^\bullet & -\infty & -\infty & 2 \\ \hline Z_1 & 1 & 1 & -\infty & 0 & 0 \end{array} \\ d_j \quad \begin{array}{cccc} 2 & 2 & 0 & 0 \end{array} \end{array},$$

where the block structure is shown, see §2.7. Unfortunately we see that entries in positions $(4, 1)$ and $(4, 2)$ are both structurally zero (meaning $d_j - c_i \neq \sigma_{i,j}$), so won’t appear in $H^{[0]}$. Recalling Equation (34) one can see that changing $d_4 = c_4$ (because entry $(4, 4)$ must be on a HVT) to 1 will not affect the other offsets—the block structure means the change in d_4 does not change any of c_1, c_2 and c_3 and a change to 1 for c_4 satisfies Equation (34) so none of d_1, d_2 and d_3 are affected. Due to our choice of Z_1 we see that $d_4 = c_4 = 1$ makes the entries in positions $(4, 1)$ and $(4, 2)$ structurally non zero and therefore the matrix $H^{[0]}$ will be as we want it. We therefore have the modified problem:

$$(59) \quad \begin{cases} f_1(t) = m\ddot{x}(t) + \lambda(t)x(t) & = 0, \\ f_2(t) = m\ddot{y}(t) + \lambda(t)y(t) - mG & = 0, \\ f_3(t) = x^2(t) + y^2(t) - L^2 & = 0, \\ Z_1(t) = \alpha\dot{x}(t) + \beta\dot{y}(t) - z_1(t) & = 0. \end{cases}$$

We proceed to carry out the DDs algorithm for this new problem to see if we have eliminated the need to pivot. Our initial Jacobians are:

$$G^{[0]} = \begin{array}{ccccc} & \ddot{x} & \ddot{y} & \lambda & z_1 & c_i \\ f_1 & \left(\begin{array}{cccc} 1 & 0 & x & 0 \end{array} \right) & 0 \\ f_2 & \left(\begin{array}{cccc} 0 & 1 & y & 0 \end{array} \right) & 0 \\ f_3 & \left(\begin{array}{cccc} 2x & 2y & 0 & 0 \end{array} \right) & 2 \\ \dot{Z}_1 & \left(\begin{array}{cccc} \alpha & \beta & 0 & -1 \end{array} \right) & 1 \\ d_j & 2 & 2 & 0 & 1 \end{array} \quad \text{and} \quad H^{[0]} = \begin{array}{cccc} & \ddot{x} & \ddot{y} & \lambda & z_1 \\ \ddot{f}_3 & \left(\begin{array}{cccc} 2x & 2y & 0 & 0 \end{array} \right) \\ \dot{Z}_1 & \left(\begin{array}{cccc} \alpha & \beta & 0 & -1 \end{array} \right) \end{array}.$$

As expected we can now choose x'' and y'' as DDs.

We proceed to finding the subsequent stage Jacobians for this new DAE:

$$G^{[1]} = \begin{array}{ccc} & \dot{x} & \dot{y} & c_i \\ \dot{f}_3 & \left(\begin{array}{cc} 2x & 2y \end{array} \right) & 1 \\ Z_1 & \left(\begin{array}{cc} \alpha & \beta \end{array} \right) & 0 \\ d_j & 1 & 1 \end{array} \quad \text{and} \quad H^{[1]} = \begin{array}{cc} & x' & y' \\ f'_3 & \left(\begin{array}{cc} 2x & 2y \end{array} \right) \end{array}.$$

We see that we have actually just delayed our choice of DDs by a stage. If we repeat the procedure as before we hope to eliminate all DD choices from the DAE (since there are no further stages to push the choice back to). We seek to add an equation to the system so that we can always choose the variables that are DD candidates at this stage. Using the same

rationale as before, we need the equation to appear at stage 2, so need the corresponding c_i to be at least two. We also only care about coefficients of \dot{x} and \dot{y} in \dot{Z}_2 , which gives us the following equation:

$$(60) \quad Z_2 := \gamma x + \delta y - z_2(t) = 0.$$

Here again γ and δ are parameters to be chosen at run time (ideally using an orthogonality condition) and z_2 is some new variable to be solved for. So that our new DAE's signature matrix with canonical offsets is:

$$\Sigma = \begin{array}{c} \begin{array}{cccccc} & x & y & \lambda & z_1 & z_2 & c_i \\ f_1 & \left(\begin{array}{ccc|cc} 2^\bullet & -\infty & 0^\circ & -\infty & -\infty \\ -\infty & 2^\circ & 0^\bullet & -\infty & -\infty \\ 0^\circ & 0^\bullet & -\infty & -\infty & -\infty \end{array} \right) & 0 \\ f_2 & & & & & & 0 \\ f_3 & & & & & & 2 \\ \hline Z_1 & \left(\begin{array}{ccc|cc} 1 & 1 & -\infty & 0 & -\infty \\ 0 & 0 & -\infty & -\infty & 0 \end{array} \right) & 0 \\ Z_2 & & & & & & 0 \\ \hline d_j & 2 & 2 & 0 & 0 & 0 \end{array} \end{array} .$$

Now to have all necessary entries of $H^{[0]}$ and $H^{[1]}$ structurally non zero we need to set $c_4 = d_4 = 1$ and $c_5 = d_5 = 2$. Note that adding these equations to the system does not change the original solution of the DAE, because the equations we add form a new block dependent on the original DAE's (block 1 above) solution. We now consider the DD stages of our new system to ensure we have indeed eliminated the need to pivot to get:

$$H^{[0]} = \begin{array}{c} \begin{array}{cccccc} & x'' & y'' & \lambda & z_1' & z_2'' & c_i \\ f_3'' & \left(\begin{array}{ccc|cc} 2x & 2y & 0 & 0 & 0 \\ \alpha & \beta & 0 & -1 & 0 \\ \gamma & \delta & 0 & 0 & -1 \end{array} \right) & 2 \\ Z_1' & & & & & & 1 \\ Z_2'' & & & & & & 2 \\ \hline d_j & 2 & 2 & 0 & 1 & 2 \end{array} \end{array} .$$

We see that a valid choice is x'' , y'' and z_2'' —what we’re doing is adding DDs for all variables in an old block that doesn’t contain an equation introduced at this stage in the original system and ‘borrowing’ DDs from the new blocks that do use an equation introduced at this stage in the original system. We’re forcing ourselves into a scheme that is unobtainable by performing a block decomposition of the new DAE and doing DDs on each block. We note this as an important consequence of this method—we are directly showing that it’s possible to find useful DD schemes by not doing a block decomposition, a fact missed in the original paper [30]. We also note that this provides a useful application for non-canonical offsets (see section 2.8)—previously it was assumed canonical offsets were always the best choice for making a DAE amenable to numerical solution. When proceeding to find $H^{[1]}$ we see that the equation introduced for stage 1 is removed and we select x' and y' as DDs and the DD algorithm then finishes. Hence we have found a way to have a static selection of dummy derivatives, albeit with a dynamically varying set of parameters.

Before giving a general algorithm for adding such equations to the system we need the following definition:

DEFINITION 4.1.2. Let the m_κ and n_κ be the number of rows and columns in the DD matrix $H^{[\kappa-1]}$ respectively.

We now use the insight gained by Example 4.1.1 to produce an algorithm (Algorithm 7) for adding such equations and finding globally valid (universal) DDs for an arbitrary DAE.

Algorithm 7 The Universal Dummy Derivative Algorithm

- 1: When finding the matrix $G^{[\kappa]}$:
 - 2: **if** $m_\kappa = n_\kappa$
 - 3: Proceed according to Algorithm 3
 - 4: **else**
 - 5: Create a vector say, candidatelist, of variables occurring in $H^{[\kappa-1]}, \dots$
 with structurally non zero columns, to order $d_j - \kappa$
 - 6: $S = \text{size}(\text{candidatelist}) - m_\kappa$
 - 7: **for** $j = 1 : S$ (using a new vector, parameterlist, in each equation)
 - 8: Add an equation to the DAE of the form:
 - 9:
$$\sum_{i=1}^{\text{size}(\text{candidatelist})} \text{candidatelist}(i) \times \text{parameterlist}(i) - v_j = 0$$
 - 10: Set each new equation's offsets equal to κ
 - 11: Add DDs to the system for the differentials of entries in candidatelist
 - 12: **if** $\kappa > 1$
 - 13: **for** $j = 1 : S$
 - 14: Add DDs to the system for $v_j^{(2)}, \dots, v_j^{(\max c_i)}$
 - 15: Form the now non-square $G^{[\kappa]}$ comprising of all rows for variables in candidatelist
 - 16: Proceed with DD algorithm, i.e form $H^{[\kappa]}$ and proceed to the next stage
-

Because Algorithm 7 is quite involved we go through each line here, giving the intuitive reason it makes sense referring to Example 4.1.1 where needed. The initial if statement on lines 2-4 tell us that we do indeed have a choice of dummy derivatives at stage κ . It is of course possible that the choice is in fact not present due to structurally zero columns, always static selections or problem specific knowledge, but this condition is general enough to be easily implementable. In Example 4.1.1 we needed to know all potential variables that could be made DDs at stage κ so that we could choose them all, e.g. stage 1 we have both x' and y' in candidatelist for the simple pendulum. We then need to know how many equations to add to 'square up' the system, which is what S tells us in line 6. We then work out the

structure of each new equation to add to the DAE (lines 8 and 9) and change to valid offsets rather than canonical so they appear at the relevant DD stage (line 10). Lines 11-14 are DD bookkeeping, adding DDs for everything that was previously a potential DD as well as DDs should the block be used in previous stages as an $S \times S$ block. Note: the DAE changes structure on each pass of this algorithm, which is why we need to reform $G^{[\kappa]}$ and $H^{[\kappa]}$ at the end of the algorithm. We simplify our additional equations by the following observation in the following section, to yield a reduced Universal Dummy Derivative (UDD) formulation.

4.2. The Reduced Universal Dummy Derivative Form

In this section we aim to further our understanding of UDDs to find a reduced formulation that eliminates the need to pivot whilst also removing unnecessary equations introduced by Algorithm 7. We saw in example 4.1.1 that the equation Z_2 is somehow redundant at stage 1, since we end up choosing z_2'' as a DD. However, since we need this equation for stage 2 it is necessary for the method to eliminate pivoting and so we are in a sense ‘stuck with it’ introducing an extra DD at stage 1. We revisit Example 4.1.1 to gain further insights into this problem:

EXAMPLE 4.2.1. Recall example 4.1.1 and note that the second equation we introduced is almost the antiderivative of the first (structurally the equation can be considered to be the antiderivative). We can therefore condense both Z_1 and Z_2 into only one additional equation and finish with a DAE that has signature matrix and valid offsets as shown in Equation (61) and now only selects x'' , x' , y'' and y' as DDs (i.e. we no longer need z_2''). The reason this is possible to do is perhaps not clear at first glance (since whilst such a reduction is clearly structurally valid it may fail numerically). Due to Griewank’s Lemma [45] our new DAE’s DD Jacobians $G^{[1]}$ and $G^{[2]}$ are equal, so if a set of parameters works for stage 1 it will also

work for stage 2.

$$(61) \quad \Sigma = \begin{array}{ccccc} & x & y & \lambda & z_2 & c_i \\ f_1 & \left(\begin{array}{ccc|c} 2^\bullet & -\infty & 0^\circ & -\infty \end{array} \right) & 0 \\ f_2 & \left(\begin{array}{ccc|c} -\infty & 2^\circ & 0^\bullet & -\infty \end{array} \right) & 0 \\ f_3 & \left(\begin{array}{ccc|c} 0^\circ & 0^\bullet & -\infty & -\infty \end{array} \right) & 2 \\ Z_2 & \left(\begin{array}{ccc|c} 0 & 0 & -\infty & 0 \end{array} \right) & 2 \\ d_j & 2 & 2 & 0 & 2 \end{array} .$$

We have the following algorithm to find a static selection of DDs and remove unnecessary added equations, which is an improvement on Algorithm 7:

Algorithm 8 The Improved Universal Dummy Derivative Algorithm

- 1: When finding the matrix $G^{[\kappa]}$:
 - 2: **if** $m_\kappa = n_\kappa$
 - 3: Proceed according to Algorithm 3
 - 4: **else**
 - 5: Create a vector say, candidatelist, of variables occurring in $H^{[\kappa-1]}$, ...
with structurally non zero columns, to order $d_j - \kappa$
 - 6: $S = \text{size}(\text{candidatelist}) - m_\kappa$
 - 7: **for** $j = 1 : S$ (using a new vector, parameterlist, in each equation)
 - 8: Add an equation to the DAE of the form:
 - 9:
$$\sum_{i=1}^{\text{size}(\text{candidatelist})} \text{candidatelist}(i) \times \text{parameterlist}(i) - v_j = 0$$
 - 10: Set each new equation's offsets equal to κ
 - 11: Add DDs to the system for the differentials of entries in candidatelist
 - 12: **if** $\kappa > 1$
 - 13: **for** $j = 1 : S$
 - 14: Add DDs to the system for $v_j^{(2)}, \dots, v_j^{(\max c_i)}$
 - 15: Form the now non-square $G^{[\kappa]}$ comprising of all rows for variables in candidatelist
 - 16: Proceed with DD algorithm, i.e form $H^{[\kappa]}$ and proceed to the next stage
 - 17: Tidy up final system:
 - 18: Check if any equation introduced is the antiderivative (barring new variables and parameters) of one introduced at a later κ stage.
 - 19: Remove all such equations and any corresponding new variables and DDs from the system.
-

The final check at the end is not needed to keep a static selection of dummy derivatives (without it we have Algorithm 7), but can reduce the size of the resulting index 1 problem. Note also line 14 where we introduce extra dummy derivatives for the new variables—we need the new equations at some stage κ but we do not need them at previous κ stages and

will instead solve their block based DD system at those stages, which by construction is always square. Let us consider the following example:

EXAMPLE 4.2.2. Consider a problem with signature and initial H matrix as below, for the purposes of this example we do not need full equations for this DAE as the algorithm works on a purely structural level we will just assume equations are chosen so that all possible candidate DDs are in fact candidate DDs:

$$\Sigma = \begin{array}{r} \begin{array}{cccccc} & x_1 & x_2 & x_3 & x_4 & x_5 & c_i \\ f_1 & \left(\begin{array}{cccc} 2 & & & 1 \end{array} \right) & 1 \\ f_2 & & \left(\begin{array}{cc} 2 & 2 \end{array} \right) & 0 \\ f_3 & 1 & & \left(\begin{array}{cc} 0 & 0 \end{array} \right) & 2 \\ f_4 & & & \left(\begin{array}{cc} 0 & 0 \end{array} \right) & 2 \\ f_5 & & \left(\begin{array}{cc} 1 & 1 \end{array} \right) & & 1 \\ d_j & 3 & 2 & 2 & 2 & 2 \end{array} \end{array} \quad \text{and} \quad H^{[0]} = \begin{array}{r} \begin{array}{ccccc} & x_1^{(3)} & \ddot{x}_2 & \ddot{x}_3 & \ddot{x}_4 & \ddot{x}_5 \\ f_1 & \left(\begin{array}{ccccc} \bullet & 0 & 0 & 0 & \bullet \end{array} \right) \\ f_3 & \left(\begin{array}{ccccc} \bullet & 0 & \bullet & 0 & 0 \end{array} \right) \\ f_4 & \left(\begin{array}{ccccc} 0 & 0 & \bullet & \bullet & 0 \end{array} \right) \\ f_5 & \left(\begin{array}{ccccc} 0 & \bullet & 0 & 0 & \bullet \end{array} \right) \end{array} \end{array}.$$

In $H^{[0]}$ \bullet is being used to denote a structurally non-zero entry. We have

$$(62) \quad \text{candidatelist} = (x_1'', x_2', x_3', x_4', x_5')$$

and $S = 1$. This makes sense: there are only two possible choices of $G^{[1]}$. All variables that appear in a $G^{[1]}$ are also in candidatelist, either we have

$$G^{[1]} = \begin{array}{r} \begin{array}{cccc} & \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \ddot{x}_4 \\ f_1 & \left(\begin{array}{cccc} \bullet & 0 & 0 & 0 \end{array} \right) \\ f_3 & \left(\begin{array}{cccc} \bullet & 0 & \bullet & 0 \end{array} \right) \\ f_4 & \left(\begin{array}{cccc} 0 & 0 & \bullet & \bullet \end{array} \right) \\ f_5 & \left(\begin{array}{cccc} 0 & \bullet & 0 & 0 \end{array} \right) \end{array} \end{array} \quad \text{or} \quad \begin{array}{r} \begin{array}{cccc} & \ddot{x}_1 & \ddot{x}_3 & \ddot{x}_4 & \ddot{x}_5 \\ f_1 & \left(\begin{array}{cccc} \bullet & 0 & 0 & \bullet \end{array} \right) \\ f_3 & \left(\begin{array}{cccc} \bullet & \bullet & 0 & 0 \end{array} \right) \\ f_4 & \left(\begin{array}{cccc} 0 & \bullet & \bullet & 0 \end{array} \right) \\ f_5 & \left(\begin{array}{cccc} 0 & 0 & 0 & \bullet \end{array} \right) \end{array} \end{array}.$$

We add an equation of the form:

$$(63) \quad Z_1 := \alpha \ddot{x}_1 + \beta \ddot{x}_2 + \gamma \ddot{x}_3 + \delta \ddot{x}_4 + \epsilon \ddot{x}_5 - v_1 = 0$$

to the DAE and choose x_1'' , x_2'' , x_3'' , x_4'' and x_5'' to be Universal DDs, setting $d_6 = c_6 = 1$.

We form the following non square Jacobian:

$$G^{[1]} = \begin{matrix} & x_1'' & x_2'' & x_3'' & x_4'' & x_5'' \\ f_1 & \bullet & 0 & 0 & 0 & \bullet \\ f_3' & \bullet & 0 & \bullet & 0 & 0 \\ f_4' & 0 & 0 & \bullet & \bullet & 0 \\ f_5 & 0 & \bullet & 0 & 0 & \bullet \end{matrix} \quad \text{and get} \quad H^{[1]} = \begin{matrix} & x_1'' & x_2'' & x_3'' & x_4'' & x_5'' \\ f_3' & \bullet & 0 & \bullet & 0 & 0 \\ f_4' & 0 & 0 & \bullet & \bullet & 0 \end{matrix}$$

again, this system is not square, we have

$$(64) \quad \text{candidatelist} = (x_1', x_3, x_4)$$

and $S = 1$ again. We add an equation of the form:

$$(65) \quad Z_2 := \zeta x_1' + \eta x_3 + \theta x_4 - v v_1 = 0$$

to the DAE and choose x_1'' , x_3' , x_4' and $v v_1''$ to be Universal DDs, setting $d_7 = c_7 = 2$.

Let us just confirm our choice of DDs is valid (given the potential need to update our parameters to keep any structurally non singular G matrix numerically non singular). Our enlarged system has signature matrix and initial H matrix:

$$\Sigma = \begin{matrix} & x_1 & x_2 & x_3 & x_4 & x_5 & v_1 & v v_1 & c_i \\ f_1 & 2 & & & & 1 & & & 1 \\ f_2 & & 2 & & 2 & & & & 0 \\ f_3 & 1 & & 0 & & & & & 2 \\ f_4 & & & 0 & 0 & & & & 2 \\ f_5 & & 1 & & & 1 & & & 1 \\ \hline Z_1 & 2 & 1 & 1 & 1 & 1 & 0 & & 1 \\ Z_2 & 1 & & 0 & 0 & & & 0 & 2 \\ \hline d_j & 3 & 2 & 2 & 2 & 2 & 1 & 2 & \end{matrix} \quad \text{and} \quad H^{[0]} = \begin{matrix} & x_1^{(3)} & \ddot{x}_2 & \ddot{x}_3 & \ddot{x}_4 & \ddot{x}_5 & \dot{v}_1 & \dot{v} \dot{v}_1 \\ \dot{f}_1 & \bullet & 0 & 0 & 0 & \bullet & 0 & 0 \\ \ddot{f}_3 & \bullet & 0 & \bullet & 0 & 0 & 0 & 0 \\ \ddot{f}_4 & 0 & 0 & \bullet & \bullet & 0 & 0 & 0 \\ \dot{f}_5 & 0 & \bullet & 0 & 0 & \bullet & 0 & 0 \\ \hline \dot{Z}_1 & \alpha & \beta & \gamma & \delta & \epsilon & -1 & 0 \\ \ddot{Z}_2 & \zeta & 0 & \eta & \theta & 0 & 0 & -1 \end{matrix}$$

Here as expected a valid $G^{[1]}$ is:

$$G^{[1]} = \begin{array}{c} f_1 \\ \dot{f}_3 \\ \dot{f}_4 \\ f_5 \\ Z_1 \\ \dot{Z}_2 \end{array} \left(\begin{array}{cccccc|c} \ddot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 & \dot{x}_5 & v\dot{v}_1 & \\ \bullet & 0 & 0 & 0 & \bullet & & 0 \\ \bullet & 0 & \bullet & 0 & 0 & & 0 \\ 0 & 0 & \bullet & \bullet & 0 & & 0 \\ 0 & \bullet & 0 & 0 & \bullet & & 0 \\ \alpha & \beta & \gamma & \delta & \epsilon & & 0 \\ \zeta & 0 & \eta & \theta & 0 & & -1 \end{array} \right) \quad \text{yielding} \quad H^{[1]} = \begin{array}{c} \dot{f}_3 \\ \dot{f}_4 \\ \dot{Z}_2 \end{array} \left(\begin{array}{cccccc|c} \ddot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 & \dot{x}_5 & v\dot{v}_1 & \\ \bullet & 0 & \bullet & 0 & 0 & & 0 \\ 0 & 0 & \bullet & \bullet & 0 & & 0 \\ \zeta & 0 & \eta & \theta & 0 & & -1 \end{array} \right).$$

Again, as expected a valid choice for $G^{[2]}$ is:

$$G^{[2]} = \begin{array}{c} f_3 \\ f_4 \\ Z_2 \end{array} \left(\begin{array}{ccc} \dot{x}_1 & x_3 & x_4 \\ \bullet & \bullet & 0 \\ 0 & \bullet & \bullet \\ \zeta & \eta & \theta \end{array} \right)$$

where the algorithm terminates.

One may think a way to tidy up the above system would be to increase $c_6 = d_6 = 2$ and remove the final equation and variable. Structurally this will still work, but numerically we may fall into trouble as $G^{[2]}$ may be singular since Z_2 is not the structural antiderivative of Z_1 . e.g. consider the following $G^{[1]}$ and $G^{[2]}$ matrices:

$$G^{[1]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 2 \end{pmatrix} \quad \text{and} \quad G^{[2]} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

so that $G^{[1]}$ is non singular (with determinant 1) whereas $G^{[2]}$ is singular.

We now go about proving the above algorithm provides a valid DD scheme (one that would be found by doing the DD algorithm to the enlarged DAE in one pass) with same solution as the original problem.

THEOREM 4.2.3. *If a variable's derivative does not appear in any potential structural DD scheme the corresponding entries in the H matrix are all structurally 0.*

PROOF. For a variables' derivative to be chosen as a DD it must be that there exists a transversal using an entry in its correspond H matrix column. For there to exist such a transversal there must be a structural non-zero in that column. If there is not then any subsequent G matrix found by selecting that column will be structurally singular and thus not produce a valid DD scheme. \square

We further clarify all potential DD candidates (i.e. DDs that exist in some potential DD scheme) by the following Theorem:

THEOREM 4.2.4. *Given any rectangular matrix with more columns than rows, and of structural full row rank every non empty column contains an element of some transversal.*

PROOF. Assume for the sake of contradiction there exists a non empty column that does not contain any elements that belong to a transversal. Without loss of generality rearrange so that a non 0 entry is in the bottom right corner, say position (m_κ, n_κ) . Since we have a valid DD scheme at the stage (i.e. the system is of full row rank) we know there exists some transversal, T say. This transversal contains some element (m_κ, J) , for some $J \in \{1, \dots, n_\kappa - 1\}$. We can therefore form a new transversal of the form $\{T(m_\kappa, J)\} \cup (m_\kappa, n_\kappa)$. \square

Theorems 4.2.3 and 4.2.4 are why we consider all non empty columns in our candidatelist, of course if one knew some extra information about the DAE they are trying to find a UDD formulation for then one could further reduce the candidatelist accordingly. We now seek to justify our removal of equations in the last part of the algorithm above.

THEOREM 4.2.5. *If a scheme produced by the above algorithm yields equations that can be removed the solution is the same as the scheme without removing those equations.*

PROOF. Structurally it is clear that such removal still leaves us with the same potential choice of DDs, since if such a removal is possible we will have (at least) structurally identical equations at some stage κ , (at least one) of which is being treated as its own block system until its new variable is needed to make a square G matrix. Numerically if such a reduction is possible then the size of the candidatelist vector must remain unchanged. Since we only remove equations in our DD stages we have two possibilities, either the difference between the size of candidatelist and m_κ has stayed the same, or it has increased. If it has stayed the same then the proposed removal of equations gives us the same structural matrix, and by Griewank's Lemma the same numerical matrix. If it has reduced we will be adding new equations that are not the antiderivatives of equations previously introduced. In which case we can still make our corresponding G matrix non-singular by varying only the new parameters introduced at this stage, since we know the rows are linearly independent. \square

Note: The above proof yields a necessary condition for introducing equations that can be removed that could shorten our algorithm. If at some stage $c_i - \kappa > 1$ for each i considered at that stage then at the next stage we must introduce equations which are the 'antiderivatives' of the ones introduced at this stage. We could shorten Algorithm 8 by introducing such a condition. Finally we have the following:

THEOREM 4.2.6. *The above algorithm provides an always static selection of DDs that is always valid (provided one chooses suitable parameters throughout integration) and has same solution as the original DAE.*

PROOF. Theorem 4.2.3 and Theorem 4.2.5 give us that we have a valid DD scheme, hence all that is left to prove is that such equation additions do not change the solution set. As illustrated in above examples the inclusion of additional equations in the described manner is equivalent to adding more dependent blocks to the system (of size $S \times S$ at each stage), which will not change the original block's solution. \square

Note: ‘same solution’ in the above Theorem may be confusing on a first read since the reformulated DAE is of a larger size. We mean that the value of any $x_i(t)$ in the original DAE is also the value of that same $x_i(t)$ in the reformulated DAE.

4.3. Numerical Results for Universal Dummy Derivatives

We solve the simple pendulum index 1 universal DD reformulation given below:

$$(66) \quad \left\{ \begin{array}{ll} f_1(t) = x''(t) + \lambda(t)x(t) & = 0, \\ f_2(t) = y''(t) + \lambda(t)y(t) - g & = 0, \\ f_3(t) = x^2(t) + y^2(t) - L^2 & = 0, \\ Z_2(t) = \gamma x(t) + \delta y(t) - z_2(t) & = 0, \\ \dot{f}_3(t) = 2x(t)x'(t) + 2y(t)y'(t) & = 0, \\ \dot{Z}_2(t) = \gamma x'(t) + \delta y'(t) - \dot{z}_2(t) & = 0, \\ \ddot{f}_3(t) = 2x(t)x''(t) + 2x'^2(t) + 2y(t)y''(t) + 2y'^2(t) & = 0, \\ \ddot{Z}_2(t) = \gamma x''(t) + \delta y''(t) - \ddot{z}_2(t) & = 0, \end{array} \right.$$

in MATLAB using ode45 (using variable step size with initial conditions $x = 6$, $y = -8$, $x' = y' = 0$ and parameters $L = 10$ and $G = 9.81$) by reformulating the problem as an ODE in z_2 and z_2' and switching parameters whenever the angle between (x, y) and (γ, δ) becomes small as a proof of concept. The Matlab cod for this can be found in Appendix B. We give a plot of the result (using a tolerance of 10^{-8} in ode45) in Figure 4.3.

We compare the solution with one produced by DAETS (see[**39**], [**36**], [**37**], [**38**] and [**9**]), an accurate order 30 Taylor series solution using a tolerance of 10^{-12} , see Figure 4.3. We briefly present some information on switching the system (changing γ and δ). If we choose our switching condition so that we switch when the angle between (γ, δ) and (x, y) (say θ) is less than $\pi/4$, i.e. trying to keep the G matrices well conditioned then the change in energy from $t = 0$ to $t = 100$ grows to around 10^{-5} for a reasonable tolerance of 10^{-8} . If we instead switch after every time to step to a new (γ, δ) orthogonal to (x, y) this change is

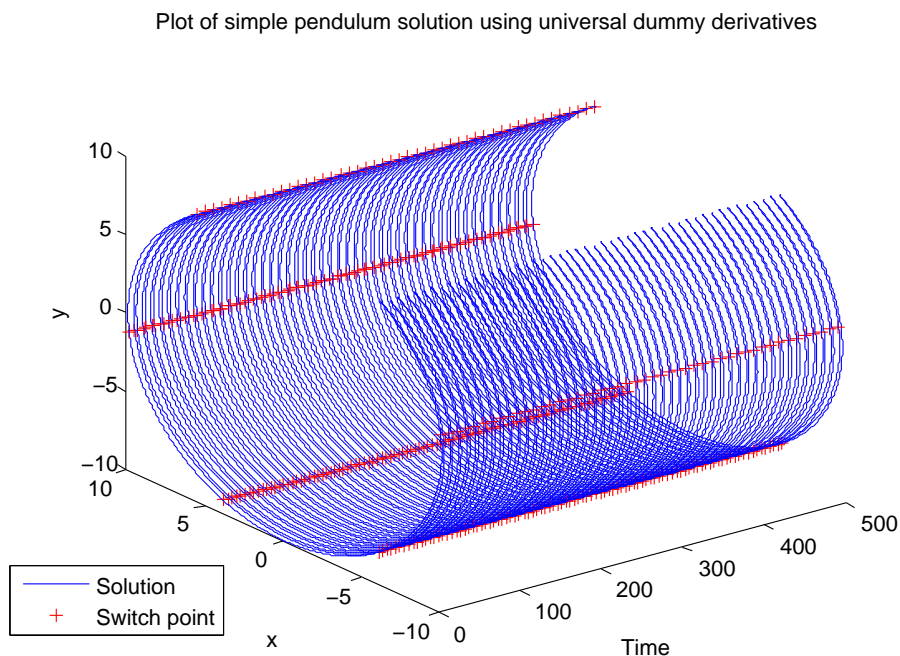


FIGURE 1. A plot of the numerical solution for Equation (66).

greatly reduced to around 10^{-11} . We show some indicative numerical results, showing the maximum difference from $t = 0$ to $t = 100$ for the simple pendulum with initial conditions as above using the theoretical energy from:

$$(67) \quad \frac{\dot{x}^2 + \dot{y}^2}{2} - Gy$$

and the actual energy found in our numerical solution in Table1: We conclude this chapter

TABLE 1. Energy error for simple pendulum.

Tolerance	Switch Every Step	Switch when $\theta \leq \frac{\pi}{4}$	Switch when $\theta \leq \frac{\pi}{6}$	Switch when $\theta \leq \frac{\pi}{8}$
1e-5	7.36e-10	8.8e-2	1.8e-1	3.33e-1
1e-6	7.45e-11	6.5e-3	2.34e-2	1.95e-2
1e-7	1.28e-11	5.56e-4	1.2e-3	1.5e-3
1e-8	1e-11	5.18e-5	1.13e-4	1.31e-4
1e-9	6.49e-12	4.78e-6	8.86e-6	8.37e-6
1e-10	1.49e-11	4.75e-7	7.83e-7	9.03e-7

with three figures showing the change in energy with tolerance set to 10^{-8} with switching conditions to switch at every step (Figure 4.3) and switching when $\theta \leq \frac{\pi}{4}$ (Figures 4.3

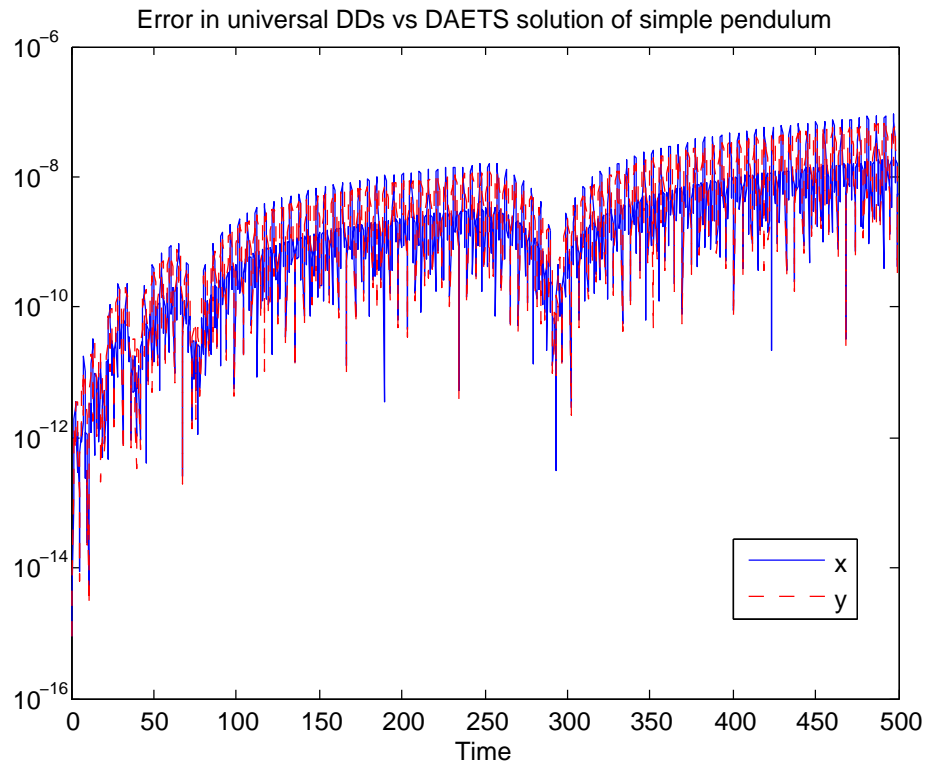


FIGURE 2. Difference between the solution of the Universal Dummy Derivative index 1 formulation of the simple pendulum solved in MATLAB by reformulating to an ODE and using ode45 and the solution to the original index 3 formulation solved via an order 30 Taylor Series method using DAETS.

and 4.3). We see that the choice of a suitable tolerance and switching condition for the method needs further development, the author conjectures using an index 1 solver rather than converting to an ODE and using an ODE solver is a good starting point for such future works.

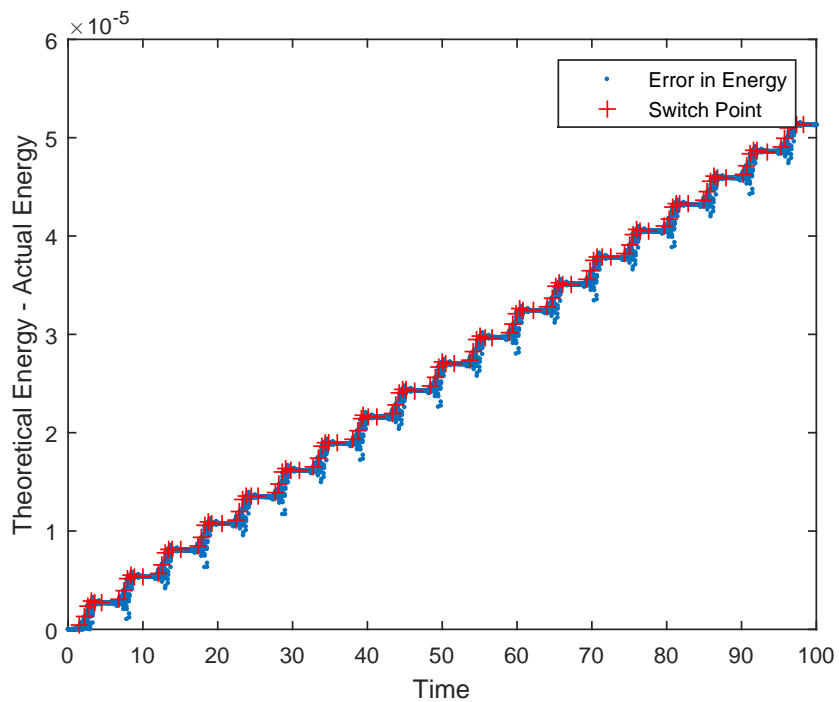


FIGURE 3. Change in energy when switching if $\theta \leq \frac{\pi}{4}$.

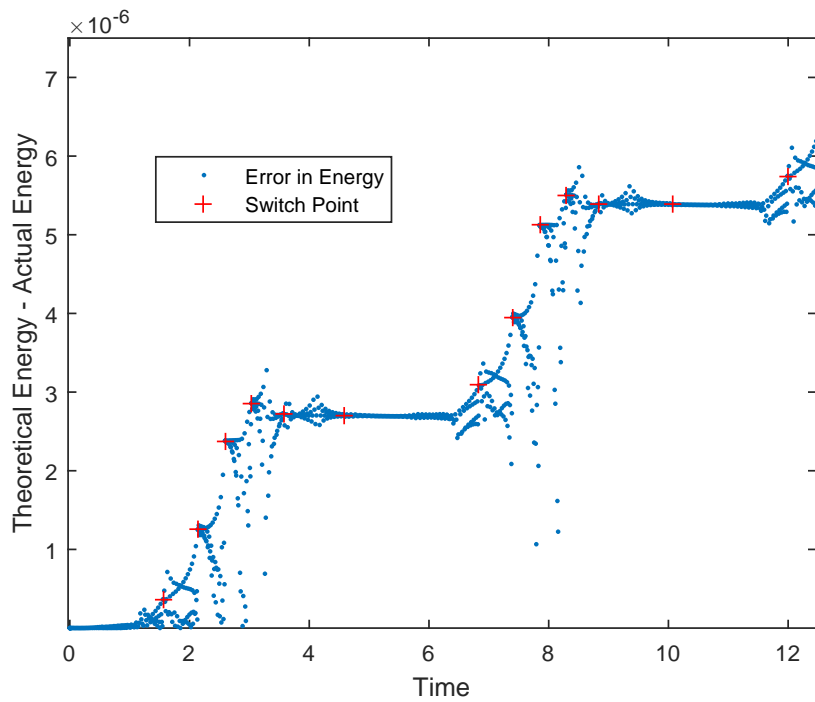


FIGURE 4. Zoom for the change in energy when switching if $\theta \leq \frac{\pi}{4}$.

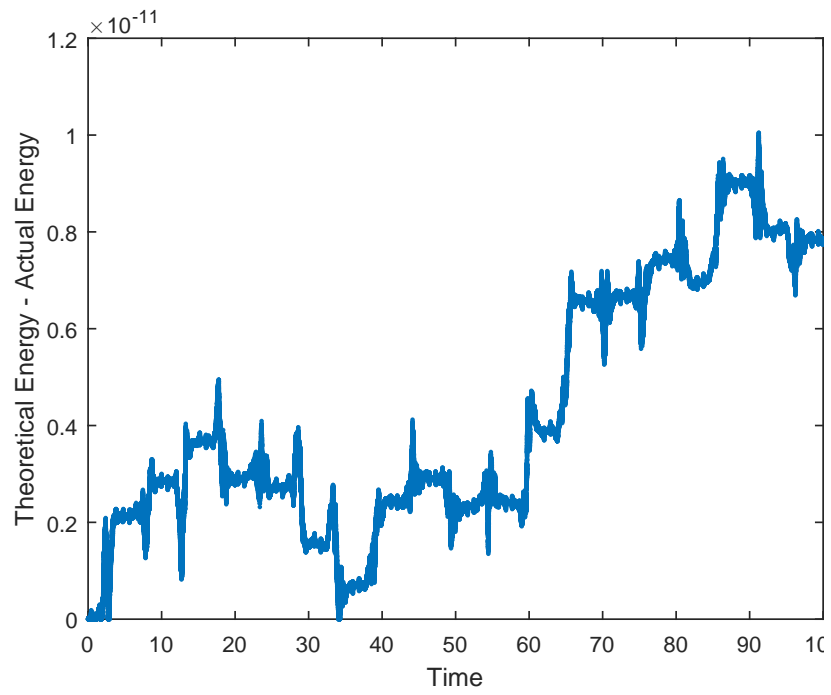


FIGURE 5. Change in energy when switching every step.

Order Reduction Leaving the Structural Index Unchanged

Some notes on the history of the material in this chapter are in order. The author first noted that though it is clear order reduction for DAEs can be done in the same way as for ODEs and in an obvious sense does not change the solution set, it is less clear that it does not essentially change the structural analysis. The author, Ned Nedialkov and John Pryce worked on this problem together during a research visit of Nedialkov to Cardiff University. The three of us formulated how order reduction changes the signature matrix and then Guangning Tan formulated and gave initial proofs of some of the theorems in this chapter. This chapter restates that material with expanded proofs. This chapter therefore represents the joint work of all those involved. In particular the author would like to acknowledge Tan for his contributions.

When one applies the SA of Pryce and the index reduction methods of Chapters 3 and 4 the DAE's order will likely increase. Most standard DAE solvers (e.g DASSL and SUNDIALS) can only solve order 1 DAEs, so some order reduction procedure will thus be necessary. If this order reduction is not done carefully one can increase the index of the DAE, negating the previously done index reduction. This chapter provides a method of carrying out order reduction that does not change the structural index. Whilst the method is somehow the intuitive approach to the problem such a result was not previously known in the literature.

5.1. Introduction to Order Reduction - Why It's Non-Trivial for DAEs

Usually texts dealing with numerical solution fo DAEs restrict themselves to order 1 DAEs, [3], however in practice when modelling e.g multi-body mechanical systems higher order derivatives can occur. The natural solution to this is to treat a high order (larger than 1) DAE as one would treat a high order ODE (see for example [58] and [56]) and

introduce new equations and variables that reduce the order of the problem. However, such a technique is shown in [62], [1], [11], [35] and [53] to potentially change the (differentiation and strangeness) index of a DAE. We take the following DAE with signature tableau from [62] where it is shown the differentiation index can increase if one naively carries out an order reduction process and see how the structural index behaves:

$$(68) \quad \begin{cases} f_1(t) = \ddot{x}_1(t) + \dot{x}_1(t) + x_2(t) & = 0, \\ f_2(t) = x_1(t) & = 0, \end{cases} \quad \Sigma = \begin{array}{cccc} & x_1 & x_2 & c_i \\ f_1 & \begin{pmatrix} 2 & 0 \\ 0 & \end{pmatrix} & & 0 \\ f_2 & & & 2 \\ d_j & 2 & 0 & \end{array}.$$

From (20) we have $\nu_s = 3$. This makes sense, one differentiates f_2 twice in order to solve for \ddot{x}_1 and then to solve for \dot{x}_2 one must differentiate both f_1 and f_2 one more time, using $x_3^{(3)}$ to solve for \dot{x}_2 . We carry out a naive order reduction procedure where we add the following equations and variables:

$$(69) \quad \begin{cases} f_3(t) = x_3(t) - \dot{x}_1(t) & = 0, \\ f_4(t) = x_4(t) - \dot{x}_2(t) & = 0, \end{cases}$$

so that our new order 1 DAE and signature tableau are:

$$(70) \quad \begin{cases} f_1(t) = \dot{x}_3(t) + x_3(t) + x_2(t) & = 0, \\ f_2(t) = x_1(t) & = 0, \\ f_3(t) = x_3(t) - \dot{x}_1(t) & = 0, \\ f_4(t) = x_4(t) - \dot{x}_2(t) & = 0, \end{cases} \quad \Sigma = \begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 & c_i \\ f_1 & & 0 & 1 & & 1 \\ f_2 & 0 & & & & 3 \\ f_3 & 1 & & 0 & & 2 \\ f_4 & & 1 & & 0 & 0 \\ d_j & 3 & 1 & 2 & 0 & \end{array}.$$

Therefore our order reduced equation (70) has index 4 and our method of order reduction has indeed increased the structural index. Of course this approach to order reduction is very alien if one is working in the above framework, so we take a moment to explain the motivation. If one has everything written vectorised, i.e. Equation (68) were written as it is in [62]:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \ddot{\mathbf{x}} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \dot{\mathbf{x}} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \mathbf{x} = \mathbf{f}(t),$$

then perhaps the most natural approach is indeed to do as is done above and reduce the higher order vector $\ddot{\mathbf{x}}$. Intuitively the index increasing makes some sense, we have added a derivative for x_2 that otherwise did not appear in the system, therefore to solve for the highest order of derivative of x_2 we will have to do more differentiations, which will increase the index. If we instead only replace high order derivatives that occur in the original DAE we would have the following order one formulation with signature tableau:

$$(71) \quad \begin{cases} f_1(t) = \dot{x}_3(t) + x_3(t) + x_2(t) & = 0, \\ f_2(t) = x_1(t) & = 0, \\ f_3(t) = x_3(t) - \dot{x}_1(t) & = 0, \end{cases} \quad \Sigma = \begin{array}{cccc} & x_1 & x_2 & x_3 & c_i \\ f_1 & & 0 & 1 & 0 \\ f_2 & 0 & & & 2 \\ f_3 & 1 & & 0 & 1 \\ d_j & 2 & 0 & 1 & \end{array}.$$

Therefore our new order reduced DAE (71) has structural index 3, which is what we would expect. We will mean this approach to order reduction for the remainder of this chapter and show it leaves the structural index unchanged.

5.2. Order Reduction and the Structural Index

We wish to develop an order reduction method that does not change the index (and DOF) of our original DAE. To do this we consider a stage wise process where we eliminate the HOD of a given variable if it has HOD larger than 1. We do this iteratively until all derivatives are at most of order 1. This approach allows us to consider just adding one

equation and variable to our DAE and then observe how Σ reacts, clearly if the index does not change over any iteration it will not change over all iterations. We start by defining some notation. Given an $n \times n$ DAE of form (1) we denote the HOD of variable x_j occurring in the system as h_j , i.e.

$$(72) \quad h_j = \max_i \sigma_{i,j},$$

we then form the set of all variables that occur at a higher order:

$$(73) \quad J = \{j \mid h_j \geq 2\}.$$

We choose some x_k with $k \in J$ and introduce a new variable x_{n+1} and a new equation:

$$(74) \quad f_{n+1} = x_{n+1} - \dot{x}_k = 0,$$

and replace all $x_k^{(p)}$ by $x_{n+1}^{(p-1)}$ for $p \geq 1$ in our original equations. We term our new $(n+1) \times (n+1)$ system as an order reduced system of our original DAE and similarly call x_k an order reduced variable. To reduce to an order 1 DAE one will have to repeat the above process for each $k \in J$ a number of times equal to each $h_k - 1$ for $k \in J$.

We now consider how such a process changes our signature matrix. Assume we have an $n \times n$ DAE, for the sake of convenience later we assume we have an HVT on the main diagonal, so that $\sigma_{n,n}$ is on the HVT and want to order reduce variable x_n . We therefore have the following signature matrix:

$$\Sigma = \begin{matrix} & x_1 & \dots & x_n \\ \begin{matrix} f_1 \\ \vdots \\ f_n \end{matrix} & \begin{pmatrix} \sigma_{1,1} & \dots & \sigma_{1,n} \\ \vdots & \ddots & \vdots \\ \sigma_{n,1} & \ddots & \sigma_{n,n} \end{pmatrix} \end{matrix}.$$

We now construct the signature matrix resulting from applying one stage of order reduction, denoting our new sigma entries by a bar, i.e. our new signature matrix is denoted $\bar{\Sigma}$ with entries $\bar{\sigma}_{i,j}$. We only replace instances of the variable that we are order reducing.

Hence

$$(75) \quad \sigma_{i,j} = \bar{\sigma}_{i,j} \quad \text{for } i = 1, \dots, n \quad \text{and} \quad j = 1, \dots, n-1.$$

In the enlarged system's signature matrix $\bar{\Sigma}$, the last two columns are trivial only in the last row, so we have:

$$(76) \quad \bar{\Sigma} = \begin{matrix} & x_1 & \dots & x_{n-1} & x_n & x_{n+1} \\ \bar{f}_1 & \left(\begin{array}{cccccc} \sigma_{1,1} & \dots & \sigma_{1,n-1} & \bar{\sigma}_{1,n} & \bar{\sigma}_{1,n+1} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \sigma_{n-1,1} & \dots & \sigma_{n-1,n-1} & \bar{\sigma}_{n-1,n} & \bar{\sigma}_{n-1,n+1} \\ \bar{f}_n & \left(\begin{array}{cccccc} \sigma_{n,1} & \dots & \sigma_{n,n-1} & \bar{\sigma}_{n,n} & \bar{\sigma}_{n,n+1} \\ \bar{f}_{n+1} & \left(\begin{array}{cccccc} -\infty & \dots & -\infty & 1 & 0 \end{array} \right) \end{array} \right) \end{matrix} \end{matrix}.$$

We now make the following observations about $\bar{\sigma}_{i,n+1}$ based on $\sigma_{i,n}$ for $i = 1, \dots, n$:

- If $\sigma_{i,n} = 0$ then x_n appears in f_i and thus must appear in \bar{f}_i with no replacements. Therefore $\bar{\sigma}_{i,n} = 0$ and $\bar{\sigma}_{i,n+1} = -\infty$.
- If $\sigma_{i,n} > 0$ then x_n may appear in \bar{f}_i . Therefore $\bar{\sigma}_{i,n} = 0$ or $\bar{\sigma}_{i,n} = -\infty$ and $\bar{\sigma}_{i,n+1} = \sigma_{i,n} - 1$.
- If $\sigma_{i,n} = -\infty$ then x_n does not appear in either f_i or \bar{f}_i . Therefore $\bar{\sigma}_{i,n} = \bar{\sigma}_{i,n+1} = -\infty$.

The above observations along with equation (76) allow us to break our problem in to the following two cases, which we will refer to throughout this chapter:

Case 1: $\sigma_{n,n} = 0$. Then $\bar{\sigma}_{n,n} = 0$ and $\bar{\sigma}_{n,n+1} = -\infty$.

Case 2: $\sigma_{n,n} > 0$. Then $\bar{\sigma}_{n,n} = 0$ or $\bar{\sigma}_{n,n} = -\infty$ and $\bar{\sigma}_{n,n+1} = \sigma_{n,n} - 1$.

We proceed in stages to show the index doesn't change, first in §5.3 we show the DOF of our DAE are unchanged under order reduction. In §5.4 we examine the canonical offsets in Case 1 and Case 2 respectively. Finally in 5.5 we show the index remains unchanged under order reduction, if the order reduction is done as above.

5.3. Invariant DOF Under Order Reduction

We begin by recalling Lemma 2.5.8 and seeing that a DAE's DOF is equal to $\text{Val}(\Sigma)$. We make a short observation about HVTs in $\bar{\Sigma}$: from equation (76) it is clear that since other entries in the last row of $\bar{\Sigma}$ are not finite an HVT of $\bar{\Sigma}$ must have an entry either using $\bar{\sigma}_{n+1,n}$ or $\bar{\sigma}_{n+1,n+1}$, which should (it will be proven shortly) make either $\bar{\sigma}_{n,n+1}$ or $\bar{\sigma}_{n,n}$ be in an HVT respectively. This fact is a result of the following theorem:

THEOREM 5.3.1. *For an order reduced DAE $\text{Val}(\Sigma) = \text{Val}(\bar{\Sigma})$.*

PROOF. We begin by showing that in both Case 1 and Case 2 $\text{Val}(\Sigma) \geq \text{Val}(\bar{\Sigma})$.

Case 1: choose a transversal in $\bar{\Sigma}$ given by:

$$\bar{T}_1 = \{(1, 1), (2, 2), \dots, (n, n), (n + 1, n + 1)\}.$$

The first $n - 1$ terms on the diagonal of $\bar{\Sigma}$ are in the HVT of Σ so are finite. Since we are in Case 1 $\bar{\sigma}_{n,n} = \bar{\sigma}_{n+1,n+1} = 0$, so this is a finite transversal. We now try to find a lower bound on $\text{Val}(\bar{\Sigma})$ in Case 1:

$$\begin{aligned} \text{Val}(\bar{\Sigma}) &\geq \sum_{(i,j) \in \bar{T}_1} \bar{\sigma}_{i,j} \\ &= \sum_{i=1}^{n-1} \bar{\sigma}_{i,i} + \bar{\sigma}_{n,n} + \bar{\sigma}_{n+1,n+1} \\ (77) \quad &= \sum_{i=1}^{n-1} \sigma_{i,i} + \sigma_{n,n} + 0 \\ &= \text{Val}(\Sigma) . \end{aligned}$$

Case 2: choose a new transversal \overline{T}_2 :

$$\overline{T}_2 = \{(1, 1), (2, 2), \dots, (n-1, n-1), (n, n+1), (n+1, n)\}.$$

As before this transversal is finite, the first $n-1$ terms are part of an HVT in Σ while $\overline{\sigma}_{n,n+1} = \sigma_{n,n} - 1$ and $\overline{\sigma}_{n+1,n} = 1$. We now try to find a lower bound on $\text{Val}(\overline{\Sigma})$ in Case 2:

$$\begin{aligned} \text{Val}(\overline{\Sigma}) &\geq \sum_{(i,j) \in \overline{T}_2} \overline{\sigma}_{i,j} \\ (78) \quad &= \sum_{i=1}^{n-1} \overline{\sigma}_{i,i} + \overline{\sigma}_{n,n+1} + \overline{\sigma}_{n+1,n} \\ &= \sum_{i=1}^{n-1} \sigma_{i,i} + \sigma_{n,n} - 1 + 1 \\ &= \text{Val}(\Sigma) . \end{aligned}$$

Hence we have that $\text{Val}(\overline{\Sigma}) \geq \text{Val}(\Sigma)$ in both cases.

We go on to bound $\text{Val}(\overline{\Sigma})$ above by $\text{Val}(\Sigma)$. Since $\overline{\Sigma}$ has a finite transversal in both cases it must have an HVT, say \overline{T}_3 :

$$\overline{T}_3 = \{(i_1, 1), (i_2, 2), \dots, (i_n, n), (i_{n+1}, n+1)\}.$$

We now make some observations on where each i_j must lie. For $j = 1, \dots, n-1$ we must have i_j between 1 and n , since $\overline{\sigma}_{n+1,j} = -\infty$. Therefore either $i_n = n+1$ (Case A) or $i_{n+1} = n+1$ (Case B). We consider Case A, using equation (76) we must have:

- $\overline{\sigma}_{i_n, n} = \overline{\sigma}_{n+1, n} = 1$.
- i_{n+1} is between 1 and n , since we have an HVT entry in the final row.
- $\overline{\sigma}_{i_n, n} + \overline{\sigma}_{i_{n+1}, n+1} = 1 + \overline{\sigma}_{i_{n+1}, n+1}$, by construction of $\overline{\Sigma}$.

We also note that since $\bar{\sigma}_{i_{n+1},n+1}$ is finite (it's in an HVT) then $\sigma_{i_{n+1},n}$ is also finite, by definition of $\bar{\Sigma}$. We also have that i_1, \dots, i_{n+1} is a permutation of $1, \dots, n+1$ and $i_n = n+1$ we have that $i_1, \dots, i_{n-1}, i_{n+1}$ is a permutation of $1, \dots, n$ and therefore the transversal:

$$T_4 = \{(i_1, 1), (i_2, 2), \dots, (i_{n-1}, n-1), (i_{n+1}, n)\}$$

is a transversal of Σ . Now we bound $\text{Val}(\bar{\Sigma})$ from above:

$$\begin{aligned} \text{Val}(\bar{\Sigma}) &= \sum_{(i,j) \in \bar{T}_3} \bar{\sigma}_{i,j} \\ &= \sum_{j=1}^{n-1} \bar{\sigma}_{i_j,j} + \bar{\sigma}_{i_n,n} + \bar{\sigma}_{i_{n+1},n+1} \\ &= \sum_{j=1}^{n-1} \bar{\sigma}_{i_j,j} + 1 + \bar{\sigma}_{i_{n+1},n+1} \\ &= \sum_{j=1}^{n-1} \sigma_{i_j,j} + \sigma_{i_{n+1},n} \\ &\leq \text{Val}(\Sigma) . \end{aligned}$$

We now treat Case B, that is if $i_{n+1} = n+1$. In this case we have $\bar{\sigma}_{i_{n+1},n+1} = \bar{\sigma}_{n+1,n+1} = 0$ and i_n is between 1 and n , since we have an HVT entry in the final row. We can also take, without loss of generality, $\bar{\sigma}_{i_n,n} = 0$ and $\bar{\sigma}_{i_n,n+1} = -\infty$, as otherwise we take a new transversal:

$$\bar{T}_5 = (\bar{T}_3 \setminus \{(i_n, n), (n+1, n+1)\}) \cup \{(i_n, n+1), (n+1, n)\}$$

and observe the following:

$$\begin{aligned}
\sum_{(i,j) \in \overline{T_5}} \bar{\sigma}_{i,j} &= \sum_{(i,j) \in \overline{T_3}} \bar{\sigma}_{i,j} - (\bar{\sigma}_{i_n,n} + \bar{\sigma}_{n+1,n+1}) + (\bar{\sigma}_{i_n,n+1} + \bar{\sigma}_{n+1,n}) \\
&= \sum_{(i,j) \in \overline{T_3}} \bar{\sigma}_{i,j} - (0 + 0) + (\bar{\sigma}_{i_n,n+1} + 1) \\
&> \sum_{(i,j) \in \overline{T_3}} \bar{\sigma}_{i,j}.
\end{aligned}$$

This is a contradiction, since $\overline{T_3}$ is an HVT. Therefore we have:

$$\begin{aligned}
\text{Val}(\overline{\Sigma}) &= \sum_{(i,j) \in \overline{T_3}} \bar{\sigma}_{i,j} \\
&= \sum_{j=1}^{n-1} \bar{\sigma}_{i_j,j} + \bar{\sigma}_{i_n,n} + \bar{\sigma}_{i_{n+1},n+1} \\
&= \sum_{j=1}^{n-1} \bar{\sigma}_{i_j,j} + 0 + 0 \\
&\leq \text{Val}(\Sigma).
\end{aligned}$$

We have bounded $\text{Val}(\overline{\Sigma})$ above and below by $\text{Val}(\Sigma)$, so they must be equal, i.e. the DOF doesn't change between matrices, completing the proof. \square

As stated prior to Theorem 5.3.1 we can now say something about an HVT in Case 1 and Case 2:

COROLLARY 5.3.2. $\overline{T_1}$ and $\overline{T_2}$ are HVTs in Cases 1 and 2 respectively.

PROOF. Due to equations (77) and (78). \square

5.4. Order Reduction and Canonical Offsets

We now go on to study how our order reduction procedure changes the offsets. In §5.4.1 we revisit Algorithm 1 and make an observation necessary to study the offsets in Case 1 (§5.4.2) and Case 2 (§5.4.3).

5.4.1. The c-d Algorithm Revisited. Recall Algorithm 1 for finding the canonical offsets of a DAE given its signature matrix, as in that algorithm we assume henceforth that an HVT is put on the main diagonal of Σ . For ease of notation going forward we reorder Algorithm 1 slightly in Algorithm 9.

Algorithm 9 Reduced Algorithm to Find Offset Vectors

- 1: Initialize: $\mathbf{c} \leftarrow \mathbf{0}$
 - 2: **while** Not converged
 - 3: Set $\mathbf{d} \leftarrow \mathbf{d}(\mathbf{c})$
 - 4: Set $\mathbf{c} \leftarrow \mathbf{c}(\mathbf{d})$
 - 5: **Return:** Canonical offsets \mathbf{c} and \mathbf{d} .
-

Where:

$$\mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) \quad \text{means } d_j = \max_i (c_i - \sigma_{i,j}) \quad \text{for all } j$$

and:

$$\mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) \quad \text{means } c_i = d_i - \sigma_{i,i} \quad \text{for all } i.$$

We then have the following lemma that lets us initialise Algorithm 9 at a $\mathbf{c} \neq \mathbf{0}$:

LEMMA 5.4.1. *Denote the canonical offsets as \mathbf{c}^* and \mathbf{d}^* for Σ . Initialising Algorithm 9 by a vector \mathbf{c}' which is element wise smaller (or equal to) \mathbf{c}^* will still produce the canonical offsets \mathbf{c}^* and \mathbf{d}^* .*

PROOF. As was done in Equation (25) we define a function ϕ that performs one iteration of Algorithm 9:

$$\phi(\mathbf{c}) = \mathbf{c}(\mathbf{d}(\mathbf{c})).$$

By proposition 2.3.3 ϕ yields a monotone increasing set of new vectors \mathbf{c} at each stage until we reach a canonical \mathbf{c} , more specifically (for fixed j) we have:

$$(79) \quad \max_i (c'_i + \sigma_{i,j}) - \sigma_{i,i} \leq \max_i (c_i + \sigma_{i,j}) - \sigma_{i,i}$$

thus we will never ‘overshoot’ our canonical \mathbf{c} . Since Algorithm 9 converges monotonically to canonical offsets this means we will still converge to canonical offsets if we initialise at a non zero \mathbf{c} vector. \square

5.4.2. Canonical Offsets in Case 1. We begin our investigation of how the offsets for an order reduced DAE behave by considering Case 1, everything in this section only applies to Case 1. We begin with the following observation for d_n and c_n :

LEMMA 5.4.2. *Let \mathbf{c} and \mathbf{d} be canonical offsets of Σ , then:*

$$(80) \quad d_n = c_n = \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n}).$$

PROOF. Let T be an HVT of Σ . We begin by showing:

$$(81) \quad \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n}) + \sigma_{n,j} \leq d_j \quad \text{for } j = 1, \dots, n-1,$$

that is:

$$d_j - \sigma_{n,j} \geq c_i + \sigma_{i,n} \quad \text{for } i, j = 1, \dots, n-1.$$

Assume for the sake of contradiction we have some r and k such that:

$$(82) \quad d_k - \sigma_{n,k} < c_r + \sigma_{r,n}.$$

We split our analysis in to two cases. **Case A:** $r = k$ and **Case B:** $r \neq k$.

Case A: Take a transversal:

$$T_1 = (T \setminus \{(k, k), (n, n)\}) \cup \{(k, n), (n, k)\}.$$

Then:

$$\begin{aligned}
\text{Val}(T_1) &= \text{Val}(T) - \sigma_{k,k} - \sigma_{n,n} + \sigma_{k,n} + \sigma_{n,k} \\
&= \text{Val}(T) - d_k + c_k + \sigma_{k,n} + \sigma_{n,k} \\
&> \text{Val}(T) \\
&= \text{Val}(\Sigma) .
\end{aligned}$$

This contradicts the fact that T is an HVT in Σ .

Case B: We form a new $n \times n$ matrix Γ with entries:

$$(83) \quad \gamma_{i,j} = \begin{cases} d_j - c_i & \text{if } i = k \text{ and } j = r, \\ \sigma_{i,j} & \text{otherwise,} \end{cases}$$

That is, we form a new matrix that is equal to Σ everywhere except where (82) holds, where we force a structural non-zero. We observe that in Γ Equation (34) holds and therefore T must be an HVT of Γ . We now take a new transversal of Γ :

$$T_2 = (T \setminus \{(k, k), (r, r), (n, n)\}) \cup \{(n, k), (r, n), (k, r)\}.$$

We now try to find a lower bound on the value of this transversal:

$$\begin{aligned}
\text{Val}(T_2) &= \sum_{(i,j) \in T_2} \gamma_{i,j} \\
&= \sum_{i=1}^n \gamma_{i,i} - \gamma_{k,k} - \gamma_{r,r} - \gamma_{n,n} + \gamma_{n,k} + \gamma_{r,n} + \gamma_{k,r} \\
&= \sum_{(i,j) \in T} \gamma_{i,j} - \sigma_{k,k} - \sigma_{r,r} - \sigma_{n,n} + \sigma_{n,k} + \sigma_{r,n} + (d_r - c_k) \\
&= \sum_{(i,j) \in T} \gamma_{i,j} - (d_k - c_k) - (d_r - c_r) - 0 + \sigma_{n,k} + \sigma_{r,n} + d_r - c_k \\
&= \sum_{(i,j) \in T} \gamma_{i,j} - d_k + c_r + \sigma_{n,k} + \sigma_{r,n} \\
&> \sum_{(i,j) \in T} \gamma_{i,j} \quad \text{by Equation 82}
\end{aligned}$$

which contradicts T being an HVT in Γ and hence Equation (81) holds. We now choose an initial vector \mathbf{c}' to initialise Algorithm 9 that is element wise smaller than \mathbf{c} so that by lemma 5.4.1 Algorithm 9 converges to the canonical offsets:

$$\mathbf{c}' = (c_1, \dots, c_{n-1}, 0).$$

Note that in the first $n - 1$ positions this vector is canonical and therefore Algorithm 9 will return:

$$(84) \quad d_j = \max_i (c_i + \sigma_{i,j}) \quad \text{for } j = 1, \dots, n - 1$$

and

$$(85) \quad c_i = d_i - \sigma_{i,i} \quad \text{for } i = 1, \dots, n - 1.$$

Hence a full run of Algorithm 9 will produce:

$$\begin{cases} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}') = (d_1, \dots, d_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})), \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (c_1, \dots, c_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})), \end{cases}$$

by Equation (81) the next iteration is:

$$\begin{cases} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) = (d_1, \dots, d_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})), \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (c_1, \dots, c_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})). \end{cases}$$

Hence:

$$\begin{aligned} \mathbf{d} &= (d_1, \dots, d_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})), \\ \mathbf{c} &= (c_1, \dots, c_n - 1, \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n})), \end{aligned}$$

is a fixed point of Algorithm 9, which completes the proof. \square

We now seek to use the above lemma to write the canonical offsets of $\bar{\Sigma}$ in terms of those for Σ in the following lemma:

LEMMA 5.4.3. *Let \mathbf{c} and \mathbf{d} be canonical offsets for Σ , then:*

$$\bar{\mathbf{c}} = (c_1, \dots, c_n, d_n - 1),$$

$$\bar{\mathbf{d}} = (d_1, \dots, d_n, d_n - 1),$$

are the canonical offsets for $\bar{\Sigma}$.

PROOF. Let $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{d}}$ be valid offsets for $\bar{\Sigma}$, i.e. we have:

$$\tilde{d}_j - \tilde{c}_i \geq \bar{\sigma}_{i,j} \quad \text{for all } i, j$$

with equality on an HVT of $\bar{\Sigma}$, say \bar{T} . We begin by finding a lower bound on offsets of $\bar{\Sigma}$ so that we can initialise Algorithm 9 at a suitable \mathbf{c} vector. We show:

$$(86) \quad \tilde{c}_i \geq c_i \quad \text{and} \quad \tilde{d}_j \geq d_j \quad \text{for all } i, j = 1, \dots, n-1.$$

We prove this bound by contradiction, assume there exists a k between 1 and $n-1$ such that $\tilde{c}_k < c_k$ and hence $\tilde{d}_k < d_k$. We form two new vectors:

$$\mathbf{C} = (\tilde{c}_1, \dots, \tilde{c}_{n-1}, \max(\tilde{d}_{n+1} + 1, \tilde{d}_n)) \quad \text{and} \quad \mathbf{D} = (\tilde{d}_1, \dots, \tilde{d}_{n-1}, \max(\tilde{d}_{n+1} + 1, \tilde{d}_n)).$$

We verify that \mathbf{C} and \mathbf{D} are valid offsets for Σ , that is Equation (34) holds with equality on an HVT of Σ . We treat various ranges of i, j values separately:

(a): $i = 1, \dots, n-1$ and $j = 1, \dots, n-1$. We have the following inequality:

$$D_j - C_i = \tilde{d}_j - \tilde{c}_i \geq \bar{\sigma}_{i,j} = \sigma_{i,j}.$$

(b): $i = n$ and $j = 1, \dots, n-1$. We have the following inequality:

$$\begin{aligned} D_j - C_n &= \tilde{d}_j - \max(\tilde{d}_{n+1} + 1, \tilde{d}_n) \\ &\geq \tilde{d}_j - \tilde{d}_n \\ &= \tilde{d}_j - \tilde{c}_n \quad (\text{since } \tilde{d}_n - \tilde{c}_n = \bar{\sigma}_{n,n} = 0) \\ &\geq \bar{\sigma}_{n,j} \\ &= \sigma_{n,j}. \end{aligned}$$

(c): $i = 1, \dots, n$ and $j = n$. We have the following inequality:

$$\begin{aligned}
D_n - c_i &= \max(\tilde{d}_{n+1} + 1, \tilde{d}_n) - \tilde{c}_i \\
&= \max(\tilde{d}_{n+1} + 1 - \tilde{c}_i, \tilde{d}_n - \tilde{c}_i) \\
&\geq \max(\bar{\sigma}_{i,n+1} + 1, \bar{\sigma}_{i,n}) \\
&\geq \sigma_{i,n}.
\end{aligned}$$

Finally we need equality on a HVT constraints (and to address position (n, n)).

(d): $i = j = 1, \dots, n - 1$.

$$D_i - C_i = \bar{\sigma}_{i,i} = \sigma_{i,i}.$$

(e): $i = j = n$.

$$D_n - C_n = 0 = \sigma_{n,n}.$$

Therefore **C** and **D** are valid offsets for Σ , which is a contradiction since **c** and **d** are the canonical offsets for Σ . Thus Equation (86) holds and we can initialize Algorithm 9 with:

$$\bar{\mathbf{c}} = (c_1, \dots, c_{n-1}, 0, 0)$$

and arrive at the canonical offsets of $\bar{\Sigma}$ by lemma 5.4.1. Due to Equations (84) and (85) Algorithm 9 will return:

$$d_j = \max_i(c_i + \sigma_{i,j}) \quad \text{for } j = 1, \dots, n - 1.$$

and

$$c_i = d_i - \sigma_{i,i} = \bar{d}_i - \bar{\sigma}_{i,i}.$$

We proceed to run Algorithm 9:

$$\begin{cases} \bar{\mathbf{d}} \leftarrow \mathbf{d}(\bar{\mathbf{c}}) = (d_1, \dots, d_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n}), \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})), \\ \bar{\mathbf{c}} \leftarrow \mathbf{c}(\bar{\mathbf{d}}) = (c_1, \dots, c_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n}), \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})) \end{cases}$$

The next iteration is:

$$\begin{cases} \bar{\mathbf{d}} \leftarrow \mathbf{d}(\bar{\mathbf{c}}) = (d_1, \dots, d_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1}) + 1, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})), \\ \bar{\mathbf{c}} \leftarrow \mathbf{c}(\bar{\mathbf{d}}) = (c_1, \dots, c_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1}) + 1, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})) \end{cases}$$

The next iteration is:

$$\begin{cases} \bar{\mathbf{d}} \leftarrow \mathbf{d}(\bar{\mathbf{c}}) = (d_1, \dots, d_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1}) + 1, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})), \\ \bar{\mathbf{c}} \leftarrow \mathbf{c}(\bar{\mathbf{d}}) = (c_1, \dots, c_{n-1}, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1}) + 1, \max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1})) \end{cases}$$

We are therefore at a fixed point. To complete the proof we note:

$$\max_{1 \leq i \leq n-1} (c_i + \bar{\sigma}_{i,n+1}) = \max_{1 \leq i \leq n-1} (c_i + \sigma_{i,n} - 1) = d_n - 1$$

and therefore have:

$$\bar{\mathbf{c}} = (c_1, \dots, c_n, d_n - 1), \quad \text{and} \quad \bar{\mathbf{d}} = (d_1, \dots, d_n, d_n - 1).$$

which completes the proof. □

To complete the section we have the following lemma that writes the canonical offsets for Σ in terms of those for $\bar{\Sigma}$:

LEMMA 5.4.4. *Given canonical offsets of $\bar{\Sigma}$ (say $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$) then:*

$$\mathbf{c} = (\bar{c}_1, \dots, \bar{c}_n) \quad \text{and} \quad \mathbf{d} = (\bar{d}_1, \dots, \bar{d}_n)$$

are the canonical offsets for Σ .

PROOF. Let $\tilde{\mathbf{c}}, \tilde{\mathbf{d}}$ and \mathbf{c}, \mathbf{d} be valid and canonical offsets for Σ respectively, i.e:

$$\tilde{d}_j - \tilde{c}_i \geq \sigma_{i,j}, \quad d_j - c_i \geq \sigma_{i,j} \quad \text{for all } i, j$$

with equality on some HVT, say T . We wish to find a suitable \mathbf{c} to initialise Algorithm 9. We show:

$$(87) \quad \tilde{c}_i \geq \bar{c}_i \quad \text{and} \quad \tilde{d}_j \geq \bar{d}_j \quad \text{for } i, j = 1, \dots, n-1.$$

Assume for the sake of contradiction there exists a k between 1 and $n-1$ such that $\tilde{c}_k < \bar{c}_k$ and hence $\tilde{d}_k < \bar{d}_k$. By Equation (86) we have that $\bar{c}_k > \tilde{c}_k \geq c_k$ which is a contradiction by lemma 5.4.3. Hence we choose the following vector to initialise Algorithm 9:

$$\mathbf{c} = (\bar{c}_1, \dots, \bar{c}_{n-1}, 0)$$

which produces:

$$\begin{cases} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_n), \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (\bar{c}_1, \dots, \bar{c}_{n-1}, \bar{c}_n), \end{cases}$$

The next iteration is:

$$\begin{cases} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_n), \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (\bar{c}_1, \dots, \bar{c}_{n-1}, \bar{c}_n), \end{cases}$$

since:

$$\begin{aligned} \max_{1 \leq i \leq n} (\bar{c}_i + \sigma_{i,n}) &= \max \left\{ \max_{1 \leq i \leq n} (\bar{c}_i + \bar{\sigma}_{i,n}), \max_{1 \leq i \leq n} (\bar{c}_i + \bar{\sigma}_{i,n+1}) + 1 \right\} \\ &= \max \{ \bar{d}_n, \bar{d}_{n+1} + 1 \} \\ &= \bar{d}_n. \end{aligned}$$

Therefore:

$$\mathbf{c} = (\bar{c}_1, \dots, \bar{c}_n) \quad \mathbf{d} = (\bar{d}_1, \dots, \bar{d}_n)$$

which completes the proof. \square

5.4.3. Canonical Offsets in Case 2. We now restrict ourselves to Case 2. The structure of this section is similar to that of §5.4.2, first we seek to represent the canonical offsets of $\bar{\Sigma}$ in terms of those for Σ and then we do the converse. We begin with the following lemma:

LEMMA 5.4.5. *If \mathbf{c} and \mathbf{d} are the canonical offsets for Σ then:*

$$\bar{\mathbf{c}} = (c_1, \dots, c_{n-1}, c_n, \max\{1, \max_{i \in K} c_i\} - 1)$$

and

$$\bar{\mathbf{d}} = (d_1, \dots, d_{n-1}, \max\{1, \max_{i \in K} c_i\}, d_n - 1)$$

are the canonical offsets for $\bar{\Sigma}$, where:

$$K = \{i \mid \bar{\sigma}_{i,n} = 0\}.$$

PROOF. Let $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{d}}$ be valid offsets for $\bar{\Sigma}$. We seek a lower bound on canonical offsets for $\bar{\Sigma}$, so that we can initialise Algorithm 9 at a non zero \mathbf{c} vector. We show:

$$(88) \quad \tilde{c}_i \geq c_i \quad \tilde{d}_j \geq d_j \quad \text{for } i, j = 1, \dots, n-1.$$

Assume for the sake of contradiction there exists a k between 1 and $n-1$ such that $\tilde{c}_k < c_k$ and hence $\tilde{d}_k < d_k$. We let:

$$\mathbf{C} = (\tilde{c}_1, \dots, \tilde{c}_{n-1}, \tilde{c}_n) \quad \text{and} \quad \mathbf{D} = (\tilde{d}_1, \dots, \tilde{d}_{n-1}, \tilde{d}_{n+1} + 1)$$

We verify such offsets are valid for Σ , treating various ranged of i, j separately:

(a): $i, j = 1, \dots, n-1$. We have the following inequality:

$$D_j - C_i = \tilde{d}_j - \tilde{c}_i \geq \bar{\sigma}_{i,j} = \sigma_{i,j}.$$

(b): $i = n$ and $j = 1, \dots, n - 1$. We have the following inequality:

$$D_j - C_n = \tilde{d}_j - \tilde{c}_n \geq \bar{\sigma}_{n,j} = \sigma_{n,j}.$$

(c): $i = 1, \dots, n - 1$ and $j = n$. We have the following inequalities:

$$D_n - c_i = \tilde{d}_{n+1} + 1 - \tilde{c}_i \geq \begin{cases} \bar{\sigma}_{i,n+1} = \sigma_{i,n} & \text{if } \sigma_{i,n} > 0 \\ \tilde{c}_{n+1} + 1 - \tilde{c}_i = \tilde{d}_n - \tilde{c}_i \geq \bar{\sigma}_{i,n} = \sigma_{i,n} & \text{if } \sigma_{i,n} = 0 \text{ or } -\infty. \end{cases}$$

Finally we need equality on a HVT constraints (and to address position (n, n)).

(d): $i = j = 1, \dots, n - 1$.

$$D_i - C_i = \tilde{d}_i - \tilde{c}_i = \bar{\sigma})i, i = \sigma_{i,i}.$$

(e): $i = j = n$.

$$D_n - C_n = \tilde{d}_{n+1} + 1 - \tilde{c}_n = \bar{\sigma}_{n,n+1} + 1 = \sigma_{n,n}.$$

Therefore \mathbf{C} and \mathbf{D} are valid offsets for Σ , which is a contradiction, since \mathbf{c} and \mathbf{d} are the canonical offsets for Σ . Hence we initialise Algorithm 9 with:

$$\bar{\mathbf{c}} = (c_1, \dots, c_n, 0, 0)$$

we have:

$$\left\{ \begin{array}{l} \bar{\mathbf{d}} \leftarrow \mathbf{d}(\bar{\mathbf{c}}) = (d_1, \dots, d_{n-1}, \max_{1 \leq i \leq n+1} (c_i + \bar{\sigma}_{i,n}), \max_{1 \leq i \leq n+1} (c_i + \bar{\sigma}_{i,n+1})) \\ \qquad \qquad \qquad = (d_1, \dots, d_{n-1}, \max\{1, \max_{i \in K} c_i\}, d_n - 1), \\ \bar{\mathbf{c}} \leftarrow \mathbf{c}(\bar{\mathbf{d}}) = (c_1, \dots, c_{n-1}, d_{n-1}, \max\{1, \max_{i \in K} c_i\} - \bar{\sigma}_{n+1,n}) \\ \qquad \qquad \qquad = (c_1, \dots, c_{n-1}, c_n, \max\{1, \max_{i \in K} c_i\} - 1) \end{array} \right.$$

The next iteration is:

$$\left\{ \begin{array}{l} \bar{\mathbf{d}} \leftarrow \mathbf{d}(\bar{\mathbf{c}}) = (d_1, \dots, d_{n-1}, \max\{1, \max_{i \in K} c_i\}, d_n - 1), \\ \bar{\mathbf{c}} \leftarrow \mathbf{c}(\bar{\mathbf{d}}) = (c_1, \dots, c_{n-1}, c_n, \max\{1, \max_{i \in K} c_i\} - 1). \end{array} \right.$$

Here we used:

$$\max_{1 \leq i \leq n+1} (c_i + \bar{\sigma}_{i,n+1}) = \max_{1 \leq i \leq n} (c_i + \sigma_{i,n}) - 1 = d_n - 1,$$

$$d_n - 1 - \bar{\sigma}_{n,n+1} = d_n - \sigma_{n,n} = c_n.$$

Therefore, since they are a fixed point of Algorithm 9:

$$\bar{\mathbf{c}} = (c_1, \dots, c_n, \max\{1, \max_{i \in K} c_i\} - 1),$$

$$\bar{\mathbf{d}} = (d_1, \dots, d_{n-1}, \max\{1, \max_{i \in K} c_i\}, d_n - 1)$$

are the canonical offsets for $\bar{\Sigma}$. We now represent the canonical offsets of Σ in terms of those for $\bar{\Sigma}$ in the following lemma:

LEMMA 5.4.6. *Let $\bar{\mathbf{c}}$ and $\bar{\mathbf{d}}$ be the canonical offsets for $\bar{\Sigma}$, then:*

$$(89) \quad \mathbf{c} = (\bar{c}_1, \dots, \bar{c}_{n-1}, \bar{c}_n) \quad \text{and} \quad \mathbf{d} = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_{n+1} + 1)$$

are the canonical offsets of Σ .

Let $\tilde{\mathbf{c}}$, $\tilde{\mathbf{d}}$ and \mathbf{c} , \mathbf{d} be valid and canonical offsets of Σ respectively. We find a bound on the canonical offsets for Σ so that we can initialise Algorithm 9 from a non zero \mathbf{c} vector.

We show:

$$\tilde{c}_i \geq \bar{c}_i \quad \text{and} \quad \tilde{d}_j \geq \bar{d}_j \quad \text{for } i, j = 1, \dots, n-1.$$

Assume for the sake of contradiction there exists a k between 1 and $n-1$ such that $\tilde{c}_k < \bar{c}_k$ and therefore $\tilde{d}_k < \bar{d}_k$, by Equation (88) $\bar{c}_k > \tilde{c}_k \geq c_k$ which is a contradiction. Therefore we can initialise Algorithm 9 with:

$$\mathbf{c} = (\bar{c}_1, \dots, \bar{c}_{n-1}, 0).$$

We have:

$$\left\{ \begin{array}{l} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) = (\bar{d}_1, \dots, \bar{d}_{n-1}, \max_{1 \leq i \leq n} (\bar{c}_i + \sigma_{i,n})) \\ \phantom{\mathbf{d} \leftarrow \mathbf{d}(\mathbf{c})} = (\bar{d}_1, \dots, \bar{d}_{n-1}, \max_{1 \leq i \leq n} (\bar{c}_i + \bar{\sigma}_{i,n+1}) + 1) \\ \phantom{\mathbf{d} \leftarrow \mathbf{d}(\mathbf{c})} = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_{n+1} + 1) \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (\bar{c}_1, \dots, \bar{c}_{n-1}, \bar{c}_n) \end{array} \right.$$

The next iteration is:

$$\left\{ \begin{array}{l} \mathbf{d} \leftarrow \mathbf{d}(\mathbf{c}) = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_{n+1} + 1) \\ \mathbf{c} \leftarrow \mathbf{c}(\mathbf{d}) = (\bar{c}_1, \dots, \bar{c}_{n-1}, \bar{c}_n) \end{array} \right.$$

Therefore, since they are a fixed point of Algorithm 9, the canonical offsets for Σ are:

$$\mathbf{c} = (\bar{c}_1, \dots, \bar{c}_n) \quad \text{and} \quad \mathbf{d} = (\bar{d}_1, \dots, \bar{d}_{n-1}, \bar{d}_{n+1} + 1)$$

as required. □

5.5. Invariant Structural Index Under Order Reduction

We now prove the structural index is invariant under order reduction:

THEOREM 5.5.1. *The structural index of a DAE is unchanged by order reduction.*

PROOF. Let \mathbf{c} and \mathbf{d} be the canonical offsets for Σ and $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{d}}$ be the canonical offsets for $\bar{\Sigma}$, we have the following inequalities in Case 1 and Case 1 by §5.4.2 and §5.4.3:

$$\text{Case 1: } \bar{c}_{n+1} = d_n - 1 = c_n - 1 < c_n \leq \max_{1 \leq i \leq n} c_i,$$

$$\text{Case 2: } \bar{c}_{n+1} = \max\{1, \max_{i \in K}\} - 1 \leq \max_{1 \leq i \leq n} c_i.$$

In both cases we have:

$$\bar{c}_{n+1} \leq \max_{1 \leq i \leq n} c_i.$$

Therefore we have:

$$\max_{1 \leq i \leq n+1} \bar{c}_i = \max_{1 \leq i \leq n} c_i$$

and the structural index therefore remains unchanged. □

CHAPTER 6

Conclusions and Future Work

This thesis set out to explore if the Signature Matrix method could be better exploited in the index reduction of DAEs. In this final chapter we give some conclusions on the work contained in this thesis as well as offer some potential avenues for future research that may prove valuable to the wider DAE community.

6.1. Conclusions

The following are the main research contributions of this thesis:

- **Illustrating the Signature Matrix Method by example.** It is the authors opinion that uptake of new methods for analysis of DAEs is a slow process, since the community all has their own index concepts and methods that work for their problems. It is our hope that the exposition of Chapter 2 goes some way to illuminating the method for those not familiar with it and hence increases uptake of this powerful method.
- **Use of non-canonical offset vectors.** Classically only canonical offset vectors were used when applying the Signature Matrix method to a DAE. Due to the exploration of non-canonical offsets in Chapter 2 and the use of them in Chapter 4 it is our hope that further analysis is done using non-canonical offset vectors.
- **Development of alternative Dummy Derivative algorithms.** The algorithms developed in Chapter 3 shed some light on the potential dummy derivatives for a given DAE as well as offer possibilities for numerical speed up of the algorithm.
- **Provided a concise overview of dummy pivoting.** It is our hope that the material in §3.6 provides the reader with a good understanding of the issues and challenges associated with the dynamic selection of dummy derivatives.

- **Developed a new dummy derivative style index reduction method that avoids dummy pivoting.** In Chapter 4 we developed a new algorithm for reducing the index of a DAE that removes the need to change the structure of the resulting index 1 system locally. Since dummy pivoting is a large shortfall of the dummy derivative algorithm it is our hope that our Universal Dummy Derivative algorithm will allow modellers to solve problems that previously needed a large amount of dummy pivoting much more easily.
- **Proved the structural index is invariant under order reduction.** This result is of course necessary if one wishes to perform order reduction on a DAE. Before this thesis the result was assumed to be true, but not proved, we hope this exposition of the problem can illuminate the issue for other index concepts.

6.2. Future Work

- **Relation between Dummy Derivatives and the structural approach for regularization.** In §3.5.2 an assertion is made about the relationship between the two algorithms, it would prove useful if this were found to be correct or incorrect as the proof may shed some light on how the HVT of a Signature Matrix can influence the choice of dummy derivatives.
- **Further exploration of the Universal Dummy Derivatives approach.** If one could derive conditions on when is best to switch parameters it would increase the speed and reliability of the approach—at present one has to monitor condition numbers of matrices (which for large problems can be expensive), or do some problem specific trick. We believe that if the method could be extended such that by introducing symbolic functions one can eliminate the need for pivoting then it would potentially become the standard approach for reducing the index of an arbitrary DAE.
- **Implementation of index reduction algorithms in DAESA.** The MATLAB code DAESA finds structural information about a DAE and provides a Signature

Matrix method based solution scheme. We believe implementing several index reduction procedures (e.g. those in Chapter 3 or the Universal Dummy Derivative algorithm) would increase uptake in the software and allow modellers to easily solve practical problems.

APPENDIX A

Code for the Simple Pendulum Using the Signature Matrix Method

```
1 % solves the simple pendulum DAE
2 %   x'' + x*la      = 0
3 %   y'' + y*la - G = 0
4 %   x^2 + y^2 - L^2 = 0
5 % where
6 %   x,y are the horizontal and vertical coordinates of the pendulum,
7 %       with y pointing down.
8 %   la (=lambda) is a Lagrange multiplier, equivalent to tension in the
9 %       pendulum.
10 %   G = gravity, L = length of pendulum, both are positive constants.
11 % It uses a Taylor series method based on Structural Analysis, taken to
12 % torder "k-stages", which implies the series for x,y have order torder+2
13 % and la has order torder.
14 %
15 % On entry
16 %   t holds initial time t_0
17 %   z is a column vector holding guesses for x and y at t_0
18 %   zp is a column vector holding guesses for x' and y' at t_0
19 %   hstep is a fixed time-step.
20 %   nsteps is the number of steps taken, so (final t) = (t_0) +
21 %   hstep*nsteps.
22 % On exit
23 %   tlist is a 1 by (nsteps+1) array holding the time points t,t+hstep,...,
```

```

24 %     t+nsteps*hstep.
25 %     zlist is a 2 by (nsteps+1) array whose i-th column holds the
26 %     computed [x;y] at the i-th time point.
27 %     zplist is a 2 by (nsteps+1) array whose i-th column holds the
28 %     computed [x';y'] at the t_0+(i-1)*hstep timestep.
29 %
30 %
31 % Algorithm note:
32 % The code is only valid for pendulum DAE, for other problems the code list
33 % in onestep would need to be changed, along with J and the steps
34 % projecting to the consistent manifold, as well as various FOR loop
35 % indices in line with the c-i and d-j found via Structural Analysis.
36 %
37 % Input Note:
38 % z, zp need not contain consistent initial conditions. Consistent points
39 % are computed from the inputs before finding later values of x and y.
40 %
41 % Constants G,L,L2 are visible to main function and its subfunctions.
42 % So also is Binom, which holds Pascal's triangle, used by ADtimes.
43 %
44 % Inputs to the function are done in the following order
45 % simpend(initial t, initial x and y as column vector,...
46 %     initial x' and y' as column vector , step length, number of steps,...
47 %     gravity constant, length constant, order of Taylor series for la -
48 %     i.e. the minimal order Taylor series used)
49
50 function [tlist,zlist,zplist] = RMsimpend(t, z, zp, hstep, nsteps, G, L, ...
    torder)
51
52 %Get list of t's
53 tlist = zeros(1,nsteps+1);

```

```
54 %Creates an n+1 vector that holds entries:
55 %t, t+hstep, ..., t+nsteps*hstep
56 for j=1:nsteps+1
57 tlist(j)=t+(j-1)*hstep;
58 end
59
60 %Set up initial zlists
61 zlist=z;
62 zplist=zp;
63 %For speed define L^2
64 L2=L^2;
65
66 %Get binomial coefficients
67 %Preallocate binomial table
68 binom=zeros(torder+3,torder+3);
69 %Get vector in form [1 0...0]
70 binomrow=eye(1,torder+3);
71 %Change first row of binomial table for binomrow
72 binom(1,:) = binomrow;
73
74 for k=2:torder+3
75 %Get new binomial row
76 binomrow = binomrow +[0 binomrow(1:end-1)];
77 %Change row of zeros for binomial row
78 binom(k,:) = binomrow;
79 end
80 %Perform a first step to make initial conditions consistent
81 %Project to consistent manifold set-up
82 h = zlist(1,1)^2+zlist(2,1)^2-L2;
83 ratio = 1/sqrt(1+h/L2);
84 xconsistent = zlist(1,1)*ratio;
```

```
85 yconsistent = zlist(2,1)*ratio;
86 M=(xconsistent*zplist(1,1)+yconsistent*zplist(2,1))/L2;
87 xpconsistent=zplist(1,1)-M*xconsistent;
88 ypconsistent=zplist(2,1)-M*yconsistent;
89
90
91 %Project to consistent manifold
92 zlist= [xconsistent; yconsistent];
93 zplist = [xpconsistent; ypconsistent];
94
95
96 for steps= 1:nsteps
97 onestep(steps);
98 end
99
100 function onestep(steps)
101 %Preallocate arrays
102 x=zeros(1,torder+2);
103 y=zeros(1,torder+2);
104 la=zeros(1,torder);
105 v1=zeros(1,torder);
106 v2=zeros(1,torder);
107 v4=zeros(1,torder);
108 v5=zeros(1,torder);
109 v6=zeros(1,torder);
110 v7=zeros(1,torder+2);
111 v8=zeros(1,torder+2);
112 v9=zeros(1,torder+2);
113 f1=zeros(1,torder);
114 f2=zeros(1,torder);
115 f3=zeros(1,torder+2);
```

```
116
117 %Use current initial conditions
118 x(1) = zlist(1,steps);
119 y(1) = zlist(2,steps);
120 x(2) = zplist(1,steps);
121 y(2) = zplist(2,steps);
122
123 %Initialise J
124 J = [1 0 x(1)
125      0 1 y(1)
126      2*x(1) 2*y(1) 0];
127
128 %Stage k=-2
129 %C
130 v7(1) = ADtimes(x,x,0);
131 v8(1) = ADtimes(y,y,0);
132 v9(1) = v7(1)+v8(1);
133 v10(1) = L2;
134 f3(1) = v9(1)-v10(1);
135
136 %Stage k=-1
137 %C'
138 v7(2) = ADtimes(x,x,1);
139 v8(2) = ADtimes(y,y,1);
140 f3(2) = v7(2) + v8(2);
141
142 for i=1:torder+1
143 %Stage k=i
144 %Initial guess
145 x(i+2) = 0;
146 y(i+2) = 0;
```

```
147 la(i) =0;
148 %A^(i-1)
149 v1(i) = ADtimes(x,la,i-1);
150 v2(i) = x(i+2);
151 f1(i) = v1(i)+v2(i);
152 %B^(i-1)
153 v4(i) = ADtimes(y,la,i-1);
154 v5(i) = y(i+2);
155 v6(i) = v4(i)+v5(i);
156 if i == 1
157 f2(1) = v6(1)-G;
158 else
159 f2(i) = v6(i);
160 end
161 %C^(i+1)
162 v7(i+2) = ADtimes(x,x,i+1);
163 v8(i+2) = ADtimes(y,y,i+1);
164 f3(i+2) = v7(i+2) + v8(i+2);
165
166 %Get next set of variables
167 VAR = -J\[f1(i);f2(i);f3(i+2)];
168 x(i+2)=VAR(1);
169 y(i+2)=VAR(2);
170 la(i)=VAR(3);
171 end
172
173 %Get final variable values (using horners method for speed).
174
175 %preacolate taylor series'
176 xnew = 0;
177 ynew = 0;
```

```
178 lanew = 0;
179 xpnew = 0;
180 ypnew = 0;
181
182 %horner's method for x
183 for xi=torder+3:-1:2
184 xnew=x(xi)+xnew;
185 xnew=hstep*(1/(xi-1))*xnew;
186 end
187 xnew=xnew+x(1);
188
189 %horner's method for y
190 for yi=torder+3:-1:2
191 ynew=y(yi)+ynew;
192 ynew=hstep*(1/(yi-1))*ynew;
193 end
194 ynew=ynew+y(1);
195
196 %horner's method for la
197 for lai=torder+1:-1:2
198 lanew=la(lai)+lanew;
199 lanew=hstep*(1/(lai-1))*lanew;
200 end
201 lanew=lanew+la(1);
202
203 %horner's method for x'
204 for xpi=torder+3:-1:3
205 xpnew=x(xpi)+xpnew;
206 xpnew=hstep*(1/(xpi-2))*xpnew;
207 end
208 xpnew=xpnew+x(2);
```

```
209
210 %horner's method for y'
211 for ypi=torder+3:-1:3
212 ypnew=y(ypi)+ypnew;
213 ypnew=hstep*(1/(ypj-2))*ypnew;
214 end
215 ypnew=ypnew+y(2);
216
217 %Project to consistent manifold set-up
218 h = xnew^2+ynew^2-L2;
219 ratio = 1/sqrt(1+h/L2);
220 xconsistent = xnew*ratio;
221 yconsistent = ynew*ratio;
222 M=(xconsistent*xpnew+yconsistent*ypnew)/L2;
223 xpconsistent=xpnew-M*xconsistent;
224 ypconsistent=ypnew-M*yconsistent;
225
226
227 %Project to consistent manifold
228 zlist(1,steps+1) = xconsistent;
229 zlist(2,steps+1) = yconsistent;
230 zplist(1,steps+1) = xpconsistent;
231 zplist(2,steps+1) = ypconsistent;
232 end
233
234 %Define ADtimes (Leibnitz Rule)
235 function w = ADtimes(u,v,p)
236 %u,v vectors, p the highest order TC
237 w = sum(binom(p+1,1:p+1).*u(1:p+1).*v(p+1:-1:1));
238 end
239 end
```


APPENDIX B

Code for the Simple Pendulum Using Universal Dummy Derivatives

```
1 function [t_all,xy_all,itbreak,xpyp_all,z_all,condition_all] = ...
    T_pendzode2(xy0,tend,tol)
2 % T_PENDZODE tests PENDZODE, which implements a solution of the pendulum by
3 % the "DDs plus extra variable" method. PENDZODE calls a helper function
4 % PENDZSOL, which does most of the work. Both are written as part of this
5 % file, as nested functions, which avoids global variables that would be
6 % needed otherwise.
7 %
8 % Input:
9 %   xy0 = [x0,y0], the initial position in xy coordinates. Should be
10 %       approximately consistent; note pendulum length is L=10.
11 %       Initial velocity is zero.
12 %   tend Integration is from t=0 to t=tend.
13 % Output:
14 %   t_all,xy_all
15 %       the solution converted back to x,y coordinates.
16 %   itbreak List of indices within t_all of t points where a coordinate
17 %       switch of the extra variable z occurred. It is defined by  $z = a x$ 
18 %       +  $b y$ , and the parameters a, b were changed at these points.
19 %
20 % One can use itbreak by code such as this:
21 %   [t,xy,itbreak] = T_pendzode2(...)
22 %   plot3(t,xy(:,1),-xy(:,2), t(itbreak),xy(itbreak,1),-xy(itbreak,2),'r+')
```

```
23 % An 'OutputFcn' is used. Its role is to decide when a switch of a,b is
24 % needed, and terminate the current section of integration. It can also be
25 % used to print to command window, etc.
26
27 % Notes:
28 % 1.Solving the nonlinear stage is done by Newton iteration (it could be by
29 % solving a quadratic equation). As expected, the method runs into
30 % trouble when bx-ay approaches 0, making one of the relevant Jacobians
31 % singular. Either stepsize goes to 0 and ODE solver gives up; or
32 % solution jumps to "the other" quadratic branch, giving nonsense.
33 % 2.Passing a sensible initial guess to the Newton method is a nuisance of
34 % this approach.
35
36 % Set vars & consts, global to this function & used by nested functions:
37 G=9.81; Lsq=100;
38 xy0 = xy0(:)'; %Force to be row vector
39 xyguess = xy0;
40 % Initial velocity is 0, for now
41 xpyp0 = [0 0];
42
43 % MAIN LOOP, of sections of integration each with a constant ab = [a,b].
44 % Choose it orthogonal to initial (x,y), and continue with that till
45 % orthogonality is lost too much.
46 solver = @ode45;
47 % solver = @ode113
48 options = odeset('OutputFcn',@outfun,...
49 'AbsTol',tol, 'RelTol',tol);
50
51 %Initialise arrays to accumulate over all sections:
52 t0 = 0;
53 t_all = [];
```

```
54 xy_all = [];
55 xpyp_all=[];
56 z_all=[];
57 condition_all=[];
58 %This is a weird thing: you can't use tend - I suspect ode45 doesn't want
59 %to solve from t0:tend in one step... what happens is the step size
60 %decreases to 0 and a Jacobian (the one in the projection) goes
61 %singular.
62 while t0<tend-1e-8
63 %ab = [-xy0(2); xy0(1)];
64 %ab=[1;1];
65 %Do QR factorisation to get alpha and beta
66 [Q,~]=qr(xy0');
67 ab=Q(:,2);
68 %ab=ab-((ab(1)*xy0(1)+ab(2)*xy0(2))/(xy0(1)*xy0(1)+xy0(2)*xy0(2)))*transpose(xy0);
69 %ab = ab/norm(ab,2);
70 % fprintf('(Re)setting ab = [%8g,%8g]\n', ab);
71 % INTEGRATION
72 % z = ax+by and dz/dt = adx/dt+bdy/dt so compute their initial values
73 z0 = xy0*ab; %row*column
74 zp0 = xpyp0*ab;
75
76 [tt,ww] = solver(@pendzode,[t0 tend],[z0 zp0],options);
77
78 % Postprocessing to convert z solution to (x,y)
79 npt = numel(tt);
80 xy = zeros(npt,2);
81 xpyp=zeros(npt,2);
82 lcondition=zeros(npt,1);
83 xyguess = xy0; %as we now return to start of integration
84 for jj=1:npt
```

```
85 [~,xpyp(jj,:),xy(jj,:)] = pendzsol(ww(jj,:),xyguess);
86 xyguess = xy(jj,:);
87 end
88 zzz=ww(:,1);
89 % xy data to set z initial values for next section
90 [~,xpyp0,xy0] = pendzsol(ww(end,:),xyguess);
91
92 % Add this section to returned cumulative solution
93 for i=1:npt
94 A=[xy(i,1),xy(i,2);a,b];
95 lcondition(i,1)=cond(A);
96 end;
97 condition_all=[condition_all;lcondition];
98 t0 = tt(end);
99 t_all = [t_all; tt];
100 xy_all = [xy_all; xy];
101 xpyp_all=[xpyp_all;xpyp];
102 z_all=[z_all;zzz];
103
104 end
105
106 % Find the section-breaks: they are where t value occurs twice.
107 itbreak = find(~diff(t_all));
108
109 % NESTED FUNCTIONS
110 function wp = pendzode(~,w)
111 % Description of ODE to put into Matlab ODE IVP solvers.
112 % w holds [z,zp] where zp = dz/dt.
113
114 % xyguess is the main architectural problem at present.
115 % Make it accessible by using nested functions, and
```

```
116 % - Initialise before calling ODE solver.
117 % - Update inside this function to latest values of x,y.
118
119 [xJ0,~,xJm2] = pendzsol(w,xyguess);
120 wp = [w(2); xJ0(4)]; % forms [dz/dt,dzp/dt]
121 % Update the global variables used as trial values by PENDZSOL:
122 xyguess = xJm2;
123 end
124
125 function [xJ0,xJm1,xJm2] = pendzsol(z-zp,xy0)
126 % PENDZSOL is a support function for PENDZODE, which uses a "dummy
127 % derivative" approach to represent the simple pendulum DAE as an ODE in a
128 % form that can be input to MATLAB's ODE IVP solvers.
129 % Input:
130 %   z-zp   Vector holding [z,zp], the given values of z and dz/dt (at
131 %          some value of t).
132 %   xy0    Vector holding initial guess of x,y to use in a Newton
133 %          iteration.
134 % Defined in the main function
135 %   pp     Vector of length 2 holding parameters a,b.
136 % Output:
137 %   XJ0    Vector holding [xdd,ydd,lam,zpp].
138 %   XJm1   Vector holding [xd,yd].
139 %   XJm2   Vector holding [x,y].
140
141 % PENDZSOL computes (at an arbitrary time t) values of relevant state
142 % variables & derivatives, in terms of a new "genuine" state variable
143 %    $z = a x + b y$ 
144 % and its derivative  $zp = dz/dt$ , where a, b are constants passed in the
145 % parameter vector pp. One of the computed variables is the 2nd derivative
146 %  $zpp = d(zp)/dt$ , thus setting up a 2nd-order ODE for  $z=z(t)$ . All the other
```

```

147 % quantities have become "dummy", given as functions of z and zp.
148 % In this way solving the pendulum is equivalent (locally, as long as a
149 % relevant Jacobian is nonsingular) to solving the ODE.
150
151 % The solution process uses trial values of x,y to get started; these are
152 % passed in x0,y0. Actually the solution is *locally* determined uniquely
153 % by z, and x0,y0 just fixes which of two global solutions (roots of a
154 % quadratic) is wanted.
155
156 % For derivatives, suffix p means genuine, suffix d means dummy.
157
158 a=ab(1); b=ab(2);
159 z=z-zp(1); zp=z-zp(2);
160
161 % Stage k=-2. Solve
162 %   0 = C = x^2+y^2 - L^2
163 %   0 = D = z - (a x + b y)
164 % for x & y as a function of z.
165
166 %Newton iteration:
167 errold = Inf;
168 x = xy0(1); y = xy0(2);
169 while true
170 G2 = [2*x 2*y; -a -b];
171 CD = [(x^2+y^2 - Lsq); (z - (a*x + b*y))];
172 d_xJm2 = -G2\CD; %Newton correction
173 err = norm(d_xJm2);
174 if err==0 || err>=errold, break, end%go to machine precision
175 x = x + d_xJm2(1);
176 y = y + d_xJm2(2);
177 errold = err;

```

```

178 end
179 xJm2 = [x y];
180 % %      DEBUG!!
181 %      fprintf('PENDZSOL: |change|=%g from guess [%g %g] to found [%g %g]\n'...
182 %          , norm(xy0-xJm2), xy0, xJm2);
183
184 % Stage k=-1. Solve
185 % 0 = C' = 2(x xd + y yd)
186 % 0 = D' = zp - (a xd + b yd)
187 % for xd & yd as a function of z & zp, using known x,y.
188 G1 = [2*x 2*y; -a -b];
189 CpDp = [0; zp];
190 xJm1 = (-G1\CpDp)'; %linear so Newton gets exact result.
191 xd = xJm1(1);
192 yd = xJm1(2);
193
194 % Stage k=0. Solve
195 % 0 = A = xdd + x lam
196 % 0 = B = ydd + y lam - G
197 % 0 = C'' = 2(x xdd + xd^2 + y ydd + yd^2)
198 % 0 = D'' = zpp - (a xdd + b ydd)
199 % for xdd, ydd, lam & zpp as a function of z & zp.
200 G0 = ...
201 [ 1  0  x 0;
202  0  1  y 0;
203 2*x 2*y 0 0;
204 -a  -b  0 1];
205 ABCppDpp = ...
206 [0;
207 -G;
208 2*(xd^2 + yd^2);

```

```

209 0];
210 xJ0 = (-G0\ABCppDpp)'; %linear so Newton gets exact result.
211 end
212
213 function status = outfun(tt,ww,flag)
214 % Also uses xyguess in the call to PENDZSOL. As OUTFUN is called by the
215 % solver at each step, these values will be continually updated by
216 % calls to PENDZODE.
217 % To break every step change status to 1.
218 status = 0;
219 switch flag
220 case 'init'%, fprintf('Initialising outfun\n');
221 case []
222 for j=1:numel(tt) % this is in case odeset's 'refine' is >1.
223 %t=tt(j);
224 w = ww(:,j); %z=w(1); zp=w(2);
225 [~,~,xjm2] = pendzsol(w,xyguess);
226 %
227 %           fprintf('t,z,zp=%9.5f % 8g % 8g, xdd ydd lam zpp=% ...
228 %           8g % 8g % 8g % 8g, xd yd=% 8g % 8g, x y=% 8g % 8g\n'...
229 %           ,t,z,zp,xj0,xjm1,xjm2);
230 %           y=xjm2(2); xd=xjm1(1); yd=xjm1(2); energy = -g*y + ...
231 %           0.5*(xd^2+yd^2);
232 %           fprintf('t, energy=%g %g\n',t,energy);
233 end
234 % quit if angle between [x y] and [a b] is <= about 30 degrees:
235 if abs(xjm2*ab)/(norm(xjm2)*norm(ab)) >= cos(pi/4)
236 %if cond([xjm2;transpose(ab)])>100
237 %           fprintf('outfun set status=1 to terminate integration\n');
238 status = 1;
239 end
240 case 'done'%, fprintf('Finalising outfun\n');

```


238 `end`

239 `end`

240 `end`

Bibliography

- [1] ARÉVALO, C., AND LÖTSTEDT, P. Improving the accuracy of BDF methods for index 3 differential-algebraic equations. *BIT Numerical Mathematics* 5, 3 (1995), 297–308.
- [2] BEALE, E. *Mathematical Programming in Practice*. Topics in operational research. Pitman, 1968.
- [3] BRENNAN, K., CAMPBELL, S., AND PETZOLD, L. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, second ed. SIAM, Philadelphia, 1996.
- [4] BUJAKIEWICZ, P., AND VAN DEN BOSCH, P. Determination of perturbation index of a DAE with maximum weighted matching algorithm. In *Proceedings., IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, 1994* (1994), pp. 129–136.
- [5] BURKE, J. Lecture notes for course Math 407 at the University of Washington, 2012. <http://www.math.washington.edu/~burke/crs/407/notes/section4.pdf>.
- [6] CAMPBELL, S. L., AND GEAR, C. W. The index of general nonlinear DAEs. *Numerische Mathematik* 72 (1995), 173–196.
- [7] CAMPBELL, S. L., AND GRIEPENTROG, E. Solvability of general differential algebraic equations. *SIAM J. Sci. Comput.* 16, 2 (1995), 257–270.
- [8] CAMPBELL, S. L., AND HOLLENBECK, R. Automatic differentiation and implicit differential equations. In *Computational Differentiation: Techniques, Applications and Tools* (1996), pp. 215–227.
- [9] CORLESS, R., AND ILIE, S. Polynomial cost for solving IVP for high-index DAE. *BIT Numerical Mathematics* 48, 1 (2008), 29–49.
- [10] CORLISS, G. F., AND CHANG, Y. F. Solving ordinary differential equations using Taylor series. *ACM Trans. Math. Software* 8, 2 (1982), 114–144.
- [11] DE BOOR, C., AND KREISS, H. O. On the condition of the linear systems associated with discretized BVPs of ODEs. *SIAM J. Numer. Anal.* 23, 5 (October 1986), 936–939.
- [12] DYNASYM AB. Dymola, dynamic modeling laboratory, user’s manual, 2004. <http://www.inf.ethz.ch/personal/cellier/Lect/MMPS/Refs/Dymola5Manual.pdf>.
- [13] E. HAIRER, C. LUBICH, M. R. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Springer-Verlag, 1989.

- [14] FEEHERY, W. F., BANGA, J. R., AND BARTON, P. I. A novel approach to dynamic optimization of ODE and DAE systems as high-index problems, 1995.
- [15] FEEHERY, W. F., AND BARTON, P. I. A differentiation-based approach to dynamic simulation and optimization with high-index differential-algebraic equations. In *Computational Differentiation: Techniques, Applications, and Tools*, M. Berz, C. Bischof, G. Corliss, and A. Griewank, Eds. SIAM, Philadelphia, PA, 1996, pp. 239–252.
- [16] FRITZSON, P. OpenModelica users guide, 2015. <https://openmodelica.org/svn/OpenModelica/trunk/doc/OpenModelicaUsersGuide.pdf>.
- [17] FRITZSON, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*, second ed. Wiley-IEEE Press, 2015.
- [18] GEAR, C. W. Differential algebraic equations, indices, and integral algebraic equations. *SIAM J. Numer. Anal.* 27, 6 (Nov. 1990), 1527–1534.
- [19] GEAR, C. W. An introduction to numerical methods for ODEs and DAEs. In *Real-time integration methods for mechanical system simulation*. Springer, 1990, pp. 115–126.
- [20] GRIEWANK, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in applied mathematics. SIAM, Philadelphia, PA, 2000.
- [21] HARMAN, P. Personal Communication, 2015.
- [22] HINDMARSH, A. C., BROWN, P. N., GRANT, K. E., LEE, S. L., SERBAN, R., SHUMAKER, D. E., AND WOODWARD, C. S. SUNDIALS, Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Trans. Math. Softw.* 31, 3 (2005), 363–396.
- [23] ITI GMBH. SimulationX user manual, 2009. <http://www.tu.kielce.pl/~rokach/instr/mud/UserManual.pdf>.
- [24] KUNKEL, P., AND MEHRMANN, V. Canonical forms for linear differential-algebraic equations with variable coefficients. *Journal of Computational and Applied Mathematics* 56, 3 (1994), 225 – 251.
- [25] KUNKEL, P., AND MEHRMANN, V. *Differential-Algebraic Equations Analysis and Numerical Solution*. European Mathematical Society, 2006.
- [26] MAPLESOFT. MapleSim user’s guide, 2014. http://www.maplesoft.com/documentation_center/maplesim6/MapleSimUserGuide.pdf.
- [27] MATHWORKS. Simulink user’s guide, 2016. http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf.
- [28] MATTSSON, S., OLSSON, H., AND ELMQVIST, H. Dynamic selection of states in DYMOLA. In *Modelica 2000 Workshop Proceedings* (2000), pp. 61–67.

- [29] MATTSSON, S. E., AND SÖDERLIND, G. A new technique for solving high-index differential-algebraic equations using dummy derivatives. In *IEEE Symposium on Computer-Aided Control System Design* (Philadelphia, PA, USA, 1992), pp. 218–224.
- [30] MATTSSON, S. E., AND SÖDERLIND, G. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput.* *14*, 3 (1993), 677–692.
- [31] MCKENZIE, R., NEDIALKOV, N. S., PRYCE, J., AND TAN, G. DAESA user guide. Tech. rep., Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, 2013.
- [32] MCKENZIE, R., AND PRYCE, J. D. Structural analysis and dummy derivatives: Some relations. In *Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science* (2015), Springer International Publishing, pp. 293–301.
- [33] MCKENZIE, R., AND PRYCE, J. D. Structural analysis based dummy derivative selection for differential algebraic equations. *BIT Numerical Mathematics* (2017). To Appear, accepted for publication.
- [34] MEHRMANN, V. Index concepts for differential-algebraic equations. Tech. rep., Institut für Mathematik, TU Berlin, 2012.
- [35] MEHRMANN, V., AND SHI, C. Transformation of high order linear differential-algebraic systems to first order. *Numerical Algorithms* *42*, 3 (2006), 281–307.
- [36] NEDIALKOV, N. S., AND PRYCE, J. D. Solving differential-algebraic equations by Taylor series (I): Computing Taylor coefficients. *BIT* *45* (2005), 561–591.
- [37] NEDIALKOV, N. S., AND PRYCE, J. D. Solving differential-algebraic equations by Taylor series (II): Computing the System Jacobian. *BIT* *47*, 1 (March 2007), 121–135.
- [38] NEDIALKOV, N. S., AND PRYCE, J. D. Solving differential-algebraic equations by Taylor series (III): The DAETS code. *JNAIAM* *3*, 1–2 (2008), 61–80. ISSN 17908140.
- [39] NEDIALKOV, N. S., AND PRYCE, J. D. DAETS user guide. Tech. rep., Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, 2008–2009.
- [40] NEDIALKOV, N. S., PRYCE, J. D., AND TAN, G. Algorithm 948: DAESA: A Matlab tool for structural analysis of differential-algebraic equations: Software. *ACM Trans. Math. Softw.* *41*, 2 (Feb. 2015), 12:1–12:14.
- [41] PANTELIDES, C. C. The consistent initialization of differential-algebraic systems. *SIAM. J. Sci. Stat. Comput.* *9* (1988), 213–231.
- [42] PETZOLD, L. Description of DASSL: a differential/algebraic system solver. In *Proceedings of IMACS World Congress* (1982).

- [43] PROCESS SYSTEMS ENTERPRISE LTD. gPROMS introductory user guide, 2004. http://eng1.jcu.edu.au/Current%20Students/general/downloads/gPROMS/introductory_guide_231.pdf.
- [44] PRYCE, J. D. Solving high-index DAEs by Taylor Series. *Numerical Algorithms* 19 (1998), 195–211.
- [45] PRYCE, J. D. A simple structural analysis method for DAEs. *BIT* 41, 2 (2001), 364–394.
- [46] PRYCE, J. D., AND MCKENZIE, R. A new look at dummy derivatives for differential-algebraic equations. In *Mathematical and Computational Approaches in Advancing Modern Science and Engineering* (2016), Springer International Publishing, pp. 713–723.
- [47] PRYCE, J. D., NEDIALKOV, N. S., AND TAN, G. Graph theory, irreducibility, and structural analysis of differential-algebraic equation systems. Cardiff University, McMaster University. In preparation., 2014.
- [48] PRYCE, J. D., NEDIALKOV, N. S., AND TAN, G. DAESA: A Matlab tool for structural analysis of differential-algebraic equations: Theory. *ACM Trans. Math. Softw.* 41, 2 (Feb. 2015), 9:1–9:20.
- [49] PRYCE, JOHN D. AND NEDIALKOV, N. S., TAN, G., AND MCKENZIE, R. Exploiting block triangular form for solving DAEs: Reducing the number of initial values. In *Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science* (2015), Springer International Publishing, pp. 367–375.
- [50] REICH, S. On a geometrical interpretation of differential-algebraic equations. *Circuits, Systems and Signal Processing* 9, 4 (1990), 367–382.
- [51] REISSIG, G., MARTINSON, W. S., AND BARTON, P. I. Differential–algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM J. Sci. Comput.* 21, 6 (1999), 1987–1990.
- [52] RHEINBOLDT, W. C. Differential-algebraic systems as differential equations on manifolds. *Mathematics of Computation* 43, 168 (1984), 473–482.
- [53] SAND, J. On implicit Euler for high-order high-index DAEs. *Appl. Numer. Math.* 42, 1-3 (Aug. 2002), 411–424.
- [54] SCHOLZ, L., AND STEINBRECHER, A. A combined structural-algebraic approach for the regularization of coupled systems of DAEs. Tech. rep., Institut für Mathematik, TU Berlin, 2013.
- [55] SCHOLZ, L., AND STEINBRECHER, A. Regularization of DAEs based on the signature method. *BIT Numerical Mathematics* 56, 1 (2016), 319–340.
- [56] SHAMPINE, L. F. *Numerical Solution of Ordinary Differential Equations*. Chapman & Hall, New York, 1994.
- [57] SHAMPINE, L. F. Solving $0 = F(t, y(t), y'(t))$ in Matlab. *Journal of Numerical Mathematics* 10, 4 (2002), 291–310.
- [58] SHAMPINE, L. F., AND GORDON, M. K. *Computer Solution of Ordinary Differential Equations. The Initial Value Problem*. W. H. Freeman and Company, San Francisco, 1975.

- [59] SHORTEN, R., WIRTH, F., MASON, O., WULFF, K., AND KING, C. Stability criteria for switched and hybrid systems. *SIAM Review* 49, 4 (2007), 545–592.
- [60] TISCHENDORF, C. Regularization of electrical circuits. *IFAC-PapersOnLine* 48, 1 (2015), 312 – 313. 8th Vienna International Conference on Mathematical Modelling, MATHMOD 2015.
- [61] WASHINGTON, I., AND SWARTZ, C. On the numerical robustness of differential-algebraic distillation models, October 23–26 2011.
- [62] WUNDERLICH, L. *Analysis and Numerical Solution of Structured and Switched Differential-Algebraic Systems*. PhD thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, 2008.