A TOOL FOR AUTOMATIC INDEX ANALYSIS OF DIFFERENTIAL-ALGEBRAIC EQUATIONS

A Tool for Automatic Index Analysis of Differential-Algebraic Equations

By
Ning Liu, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

Master of Science

McMaster University

MASTER OF SCIENCE (2006)
(Computing and Software)

McMaster University
Hamilton, Ontario
Canada

TITLE:	A Tool for Automatic Index Analysis of Differential-Algebraic Equations
AUTHOR:	Ning Liu B.Sc. (McMaster University)
SUPERVISOR:	Dr. Ned Nedialkov and Dr. Sanzheng Qiao
NUMBER OF PAGES:	vii, 82

Abstract

Systems of differential-algebraic equations (DAEs) arise in applications such as circuit simulation, models of chemical processes, optimal control, and multi-body dynamics. Informally, the index of a DAE is the number of differentiations needed to convert it to an ordinary differential equation. The index generally indicates the difficulty of solving a DAE problem. The higher the index of a DAE, the more difficult it is to solve it numerically.

Structural index analysis plays a crucial role in solving DAE problems. In this thesis, we present two methods for index analysis, namely, Pryce's structural analysis (SA) and Linninger's symbolic-numeric (SN) analysis. We provide a Matlab tool implementing these two approaches: an Automatic Structural Index Analyzer (ASIA). We discuss the underlying algorithms, which include generating a signature matrix and computing SA index, computing a system Jacobian, and generating a symbolic-numeric matrix and computing SN index. We also present implementation issues and illustrate how ASIA is used.

Numerical experiments show that ASIA can produce reliable structural information. We also show examples on which structural analysis fails, and how ASIA detects such situations.

Acknowledgements

First, I would like to express my sincere thanks and deep appreciation to my supervisors, Dr. Nedialko Nedialkov and Dr. Sanzheng Qiao, for their constant support, encouragement, and guidance through my entire graduate studies. This thesis would not be what it is now without their enormous suggestions and corrections. I have learned much from them in both academic research and non-academic fields.

I thank Dr. Douglas Down and Dr. Christopher Anand, for reviewing my thesis and their valuable suggestions and comments. I appreciate all the faculty members and fellow students in the department of computing and software, for inspiring me and sharing many good ideas with me.

I am grateful to my dearest parents. Without their selfless love and support in many years, I would never accomplish any goal in my life. I also thank my cousin Kang Liu and his wife, for their great help during my studies.

I would not forget Jim Cai, Yi Luo, Kelvin, Omar, Shu Wang, and all other friends in Canada, for the friendship and all cheerful moments we had.

Last but never least, my special thanks to Shanshan Chen, who brings a new and hopeful dimension to my life mission.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We consider an initial value problem (IVP) of a system of differential-algebraic equations (DAEs) of the form

$$f_i(t, x_j \text{ and their derivatives}) = 0, \quad 1 \leq i, j \leq n, \tag{1.1}$$

where $x_j = x_j(t)$ are dependent variables, and t is an independent variable. We assume that f_i are sufficiently differentiable.

Informally, the index of a DAE system is defined as the number of differentiations needed to convert it into an ordinary differential equation (ODE) system. In this thesis, we study and implement two index analysis algorithms [Pry98, KL04]: one allows derivatives of order higher than one to appear in the system, and the other allows first order derivatives only. We also provide a tool for automatic index analysis of DAEs.

1.1 Background

Solving the initial value problem of a DAE system has been an active research area for over two decades. Many physical and engineering problems occur as a system of nonlinear differential equations with algebraic constraints.

Various numerical methods have been developed for solving DAEs, including backward differentiation (BDF) and implicit Runge-Kutta (IRK) methods [KEBP96]. These methods are used widely, and have been proven to be efficient and reliable in many situations. However, most of these approaches target lower index problems and sometimes require special structure of the problem. It was thought that DAEs of index higher than one are less significant in practice. Nevertheless, recent research in different areas shows that many problems are naturally and easily modeled as high-index DAEs. Generally, high-index DAEs are much more difficult to solve than index-1 or 2 problems.

Many studies have been conducted for index analysis of DAE systems. Pantelides [Pan88] derives a criterion for determining how to differentiate a subset of equations in a nonlinear DAE system to provide further constraints for initial values. He proposed an algorithm based on graph theory to locate those subsets of equations to be differentiated.

In [MS93], Mattsson and Söderlind present an index reduction algorithm for DAEs. They differentiate parts of the DAE analytically and replace the derivatives intro-

duced by differentiation by a new algebraic variable called a dummy derivative. The resulting augmented system is at most index 1.

Campbell and Gear's derivative array equations [CG95a] is another attempt of studying DAE indices. Comparing with Pryce's structural analysis [Pry98, Pry01], which we will introduce shortly, this approach requires symbolic preprocessing of the system. Therefore, it can be difficult to implement.

In [Pry98, Pry01], Pryce presents a direct, easily applied method for structural analysis (SA) of a DAE. It generalizes the method of Pantelides [Pan88], but it is more straightforward and can be applied to DAEs of any order. Based on this, Nedialkov and Pryce [NP05b] introduce a new approach for solving DAEs in the form (1.1). The DAEs can be of high index, fully implicit and contain derivatives of orders higher than one. Nedialkov [NP05b] implements the method into a prototype DAE solver, and Zhang redesigns and implements the solver HIDAETS in [Zha05].

Generally speaking, the index obtained by structural analysis is an upper bound for the true index of a DAE system [KEBP96]. However, recent research has shown that the actual differential index could be greater than the structural rank in some problems [GRB00]. In [KL04], S. Chowdhry, H. Krendl, and A. Linninger propose a new approach for index analysis. Their method includes symbolic as well as numerical information of the system in order to overcome the shortcomings of over/underestimating the correct DAE index.

1.2 Motivation and Contribution

Index analysis plays a crucial role in solving DAE systems. Due to the lack of high-index DAE solvers, early work on index analysis deals mainly with index reduction [Gea88, Gea90]. In general, it consists of directly searching for derivatives of algebraic variables by multiple differentiations of algebraic constraints. However, the symbolic processing of nonlinear equations in various applications can be complex. The structural analysis is developed to overcome this defect. In an SA, a DAE system is represented by a matrix of variables and their derivatives. Each entry σ_{ij} in the matrix indicates the appearance of the ith variable in the jth equation.

Pryce's SA [Pry01] is considered to be one of the easy-to-use and efficient index analysis methods. An automatic tool which applies this analysis is needed, so that it can help users from different application areas to reliably determine the index of a given system. However, this SA fails in particular problems due to various reasons, that are unclassified and not fully understood. A tool, which can help to identify the patterns in those problems where the SA fails, is important to future index analysis studies.

Linninger's symbolic-numeric (SN) analysis [KL04] is another structural approach. It shows some advantages in determining differential indices for DAE systems with derivatives appears linearly. We also want to have an integrated software that provides results of the SN analysis as a reference and complement to Pryce's SA analysis, which

can also assist improving the SA approach in various ways.

In this thesis, we present the algorithms for our Automatic Structural Index Analyzer (ASIA), including generating structural information, computing a system Jacobian, and computing the SN index. We implement these algorithms in Matlab using the technique of operator overloading. This avoids complex source code transformation. We provide a user-friendly interface of ASIA, so that the software can be easily used by people without advanced knowledge of index analysis. We conduct extensive numerical tests for both DAE and ODE problems using ASIA. The experiments show that our software can perform accurate index computation and return useful information for comparing two types of analyses and helping to develop heuristics for better index analysis algorithms.

1.3 Thesis Structure

The thesis is organized as follows.

In Chapter 2, we introduce the main steps of Pryce's SA method and Linninger's SN analysis. We illustrate them using simple examples.

Chapter 3 consists of two parts. We first present a detailed algorithm for implementing the SA, computing the system Jacobian, and computing the SN index through operator overloading. All the algorithms are demonstrated by pseudo code. In the second part, we discuss some issues from a software design perspective. We

present a general description of ASIA, its informal specifications, the interface design, and instructions of how to install and use this tool.

In Chapter 4, we perform numerical experiments on 8 problems. Among them, we show examples where both analyses succeed, as well as some cases where transformation of equation formulation is required to obtain correct results.

In Chapter 5, we give a conclusion of this thesis and discuss future work.

Chapter 2

Theory of Index Analysis

In this chapter, we introduce two approaches for index analysis on which our tool is based.

2.1 Pryce's Structural Analysis

In general, Pryce's SA is composed of the following steps.

1. Generate a structural matrix.

2. Solve a linear assignment problem for the matrix obtained in the previous step.

3. Compute the structural index.

We first introduce some definitions and notation and then present the details of each step with simple examples.

This SA is also called a *signature method*, as it involves an $n \times n$ matrix $\Sigma = (\sigma_{ij})$, the *signature matrix*, defined for (1.1) as:

$$\sigma_{ij} = \begin{cases} \text{highest order of derivative of the } j\text{th variable that occurs in the} \\ i\text{th equation; or} \\ -\infty \text{ if the } j\text{th variable does not occur in the } i\text{th equation.} \end{cases} \tag{2.1}$$

A *Linear Assignment Problem* (LAP) is defined as a task of maximizing the effectiveness of assigning a set of jobs to a group of workers, where each worker i has an effectiveness measure for job j. More details about LAP can be found in [Ber91]. In the content of SA, we use an LAP solver to find the *highest value transversal* (HVT) of a signature matrix Σ (2.1). A *transversal* of an $n \times n$ Σ is an n-element subset of Σ, with one element in each row and column. The *highest value transversal* is a *transversal* T, where the sum of all elements in T, denoted by $\|T\|$, is maximum.

A *consistent point* of a DAE system is a set of values of x_j and their derivatives at a given t that determine a unique solution.

The *degree of freedom* of (1.1) is the number of independent initial conditions required.

Given a DAE system of the form (1.1), we perform the following steps of SA.

1. Form the $n \times n$ signature matrix Σ defined in (2.1).

Example 2.1. Through the rest of this chapter, we use the single pendulum

problem [AP98]:

$$\begin{aligned} 0 = f &= x'' + x\lambda, \\ 0 = g &= y'' + y\lambda - G, \\ 0 = h &= x^2 + y^2 - L^2, \end{aligned} \tag{2.2}$$

where $G > 0$, $L > 0$ are constants, and the dependent variables are $x(t)$, $y(t)$, and $\lambda(t)$.

Its signature matrix Σ, labeled by equations and variables, is

$$\begin{array}{c} \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{ccc} x & y & \lambda \end{array} \\ \begin{pmatrix} 2 & -\infty & 0 \\ -\infty & 2 & 0 \\ 0 & 0 & -\infty \end{pmatrix} \end{array}.$$

2. Solve the LAP for Σ and find the HVT.

Example 2.2. For the single pendulum problem, two possible HVTs can be easily found at positions $(f, \lambda), (g, y), (h, x)$ and $(f, x), (g, \lambda), (h, y)$, where both $\|T\| = 2$.

For large systems, efficient algorithms for solving LAP can be found in [Ber91] and [Duf81].

3. Solve the *dual* of HVT T in linear programming (LP) sense to find two n-dimensional integer vectors $\mathbf{c}$ and $\mathbf{d}$, which maximize $z = \sum_j d_j - \sum_i c_i$, with all $c_i \geq 0$, satisfying

$$\begin{cases} d_j - c_i \geq \sigma_{ij} & \text{for all } i, j = 1, \ldots, n, \\ d_j - c_i = \sigma_{ij} & \text{for all } (i, j) \in T. \end{cases} \tag{2.3}$$

The vectors $\mathbf{c}$ and $\mathbf{d}$ are called *equation offsets* and *variable offsets* respectively.

Example 2.3. The offsets for the single pendulum problem are $\mathbf{c} = (0, 0, 2)$ and $\mathbf{d} = (2, 2, 0)$:

$$\begin{array}{c|ccc|c} & x & y & \lambda & c_i \\ f & 2 & -\infty & 0* & 0 \\ g & -\infty & 2* & 0 & 0 \\ h & 0* & 0 & -\infty & 2 \\ \hline d_j & 2 & 2 & 0 & \end{array} \quad \text{and} \quad \begin{array}{c|ccc|c} & x & y & \lambda & c_i \\ f & 2* & -\infty & 0 & 0 \\ g & -\infty & 2 & 0* & 0 \\ h & 0 & 0* & -\infty & 2 \\ \hline d_j & 2 & 2 & 0 & \end{array}$$

Here the HVTs are marked with $*$.

4. Form the system Jacobian $\mathbf{J}$, where

$$\mathbf{J} = \frac{\partial\left(f_1^{(c_1)}, \ldots, f_n^{(c_n)}\right)}{\partial\left(x_1^{(d_1)}, \ldots, x_n^{(d_n)}\right)}. \tag{2.4}$$

By Griewank's lemma in [NP05a], the authors proved that (2.4) is equivalent

to

$$\mathbf{J}_{ij} = \begin{cases} \dfrac{\partial f_i}{\partial x_j^{(\sigma_{ij})}} & \text{if } d_j - c_i = \sigma_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

Example 2.4. In our single pendulum problem, we compute $\mathbf{J}$ as

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{bmatrix}$$

5. Form the enlarged system obtained by taking derivatives of f_i up to c_ith order.

Example 2.5. For the single pendulum, denoting $X = (x, x', x''; y, y', y''; \lambda)$, we have

$$F(t, X) = \begin{pmatrix} x'' + x\lambda \\ y'' + y\lambda - G \\ x^2 + y^2 - L^2 \\ 2xx' + 2yy' \\ 2xx'' + 2yy'' + 2x'^2 + 2y'^2 \end{pmatrix}$$

6. The structure analysis succeeds if (t^*, X^*) is a solution point of $F(t, X)$, and $\mathbf{J}$ is nonsingular there. We call (t^*, X^*) a consistent point of F as defined previously. In our example, it is easy to check that $\det \mathbf{J} = -2(x^2 + y^2) = -2L^2 \neq 0$, for all x and y. Therefore, the SA succeeds on the single pendulum problem.

7. Upon the success of SA, we can conclude the following properties of a DAE in a neighborhood of (t^*, X^*).

   - The DAE has DOF $\Sigma d_j - \Sigma c_i$, which is also the value of the HVT.
   - The DAE has a differentiation index [Pry01] v_d less than or equal to the structural index ν_T, where

$$\nu_T = \max_i c_i + \begin{cases} 1 & \text{if some } d_j = 0, \\ 0 & \text{if all } d_j > 0. \end{cases} \tag{2.5}$$

   Example 2.6. According to the offset vectors **c** and **d**, the single pendulum system has DOF 2, and $v_d = \nu_T = 3$.

In [Pry01], the author concludes that for all problems where Pantelides' [Pan88] algorithm applies, the SA returns the same index ν_T as in Pantelides' algorithm.

2.2 The Symbolic Numeric Index Analysis

The SN analysis is another type of structural approach to compute the indices of DAE problems. It is only applicable to first order systems. The overall process contains two major steps: generating a symbolic numeric incidence matrix and computing its rank. Similarly, we first give some definitions and notation. Then we illustrate each step by simple examples.

A *symbolic numeric incidence* (SNI) matrix P of a DAE system is an $n \times 2n$ matrix, whose columns represent dependent variables x_j and their first-order derivatives x'_j. The values of its entries are defined as

$$P_{ij} = \begin{cases} c & \text{if } x_j \text{ or } x'_j \text{ appears linearly with coefficient } c \text{ in } f_i, \\ * & \text{if } x_j \text{ or } x'_j \text{ appears nonlinearly in } f_i, \text{ or} \\ 0 & \text{if } x_j \text{ or } x'_j \text{ does not appear in } f_i. \end{cases} \tag{2.6}$$

Example 2.7. A simple example of SNI matrix is

$$0 = f = x' + x\lambda + 2y, \qquad \begin{array}{c} \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} x' & y' & \lambda' & x & y & \lambda \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & * & 2 & * \end{pmatrix} \end{array}$$

A set of *σv-operations* $\Omega = \{+, -, \times, /\}$ is defined for symbols of SNI matrix entries. The operations are binary compositions of addition, subtraction, multiplication and division over three types of SNI matrix entries: $\{0, c, *\}$. The rule of computation is similar to regular arithmetic operations except that any operation involving '$*$' produces '$*$', indicating that the nonlinearity is preserved under σv-operations. The complete binary operations definition is in Table 2.1.

The *σv-differentiation* is a method that performs regular differentiation of algebraic equations using its SNI matrix. For an algebraic equation e_i and a variable x_j, the σv-differentiation of e_i is defined as follows.

1. If x_j appears linearly with coefficient c in e_i (which corresponds to $(e_i, x_j) = c$ in

$\circ \in \Omega$	$+$	$-$	$\times$	$/$
$0 \circ \{c, *\}$	$\{c, *\}$	$\{-c, *\}$	0	0
$0 \circ 0$	0	0	0	N/A
$c_1 \circ \{0, c_2\}$	$c_1 + \{0, c_2\}$	$c_1 - \{0, c_2\}$	$c_1 \times \{0, c_2\}$	N/A or c_1/c_2
$* \circ \{0, c\}$	$*$	$*$	$*$	N/A or $*$
$\{c, *\} \circ *$	$*$	$*$	$*$	$*$
$0 \circ *$	$*$	$*$	0	0

Table 2.1: Rules of σv-operations

the SNI matrix), after differentiation, x_j is eliminated from e_i, while x_j' appears linearly with coefficient c (which corresponds to $(e_i, x_j) = 0$ and $(e_i, x_j') = c$ in the SNI matrix).

2. If x_j appears nonlinearly in e_i (which corresponds to $(e_i, x_j) = *$ in the SNI matrix), after differentiation, both x_j and x_j' appear nonlinearly in e_i (which corresponds to $(e_i, x_j) = *$ and $(e_i, x_j') = *$ in the SNI matrix).

3. If x_i does not appear in e_i (which corresponds to $(e_i, x_j) = 0$ in the SNI matrix), after differentiation, both (e_i, x_j) and (e_i, x_j') remain 0 in the SNI matrix.

By definition, the σv-differentiation is rigorous in linear cases, which means that it is capable of reflecting additions/subtractions of coefficients in linear terms. However, since each nonlinear term is simply represented by a symbol '$*$', the *σv-differentiation* may ignore the effect of multiplication/division between nonlinear terms, as we will see in the case studies later. Based on the *σv-differentiation*, we can give the algorithm of the symbolic numeric index analysis [KL04].

Given a DAE in the form of (1.1), we perform the following steps.

1. Initialize a variable **index** to 0. Form the symbolic numerical incidence matrix P according to (2.6).

   Example 2.8. Similarly, we use the following example to illustrate each step of the symbolic numeric index analysis. For a DAE system with independent variables $X = (x_1, \ldots, x_4)$,

$$\begin{aligned} 0 = f_1 &= x_1' - x_4, \\ 0 = f_2 &= x_2' - x_2 + x_4, \\ 0 = f_3 &= x_3' - x_1^2 + x_2^2, \\ 0 = f_4 &= x_1 + x_2 + 4x_3, \end{aligned} \tag{2.7}$$

   its SNI matrix P is

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1' & x_2' & x_3' & x_4' & x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 4 & 0 \end{pmatrix} \end{array} \tag{2.8}$$

2. Compute the numerical rank of P. One approach to determine matrix rank is to apply a symbolic-numeric type LU decomposition, which involves both regular coefficient and special variable elimination in SNI matrices. However, this rank determination may be inaccurate due to limitations of floating-point arithmetic.

Generally speaking, singular value decomposition can handle rank-deficiency in the presence of round-off error. It is done by comparing small singular values with a given tolerance ϵ. The strategy of managing numerical issues in the implementation will be discussed in the next chapter. More details about rank determination can be found in [GL96].

Example 2.9. For (2.8), since f_4 does not contain any derivative of x_i, the rank of the SNI matrix is 3, which indicates rank deficiency.

3. If the SNI is full-rank in step 2, the index of the DAE system is 0. If rank-deficiency is detected, a loop of σv-differentiation is applied. Each of the algebraic constraints in the system is differentiated once.

Example 2.10. In (2.8), the only algebraic equation is f_4. Therefore, it is differentiated as

$$\begin{array}{cc} & \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{array}{c} f_4 \\ \Rightarrow \ f_4' \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 4 & 0 \\ 1 & 1 & 4 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{2.9}$$

4. After the differentiation, known derivatives will be substituted into the new equations obtained from the previous step according to (2.1).

Example 2.11. To eliminate x_1', x_2', and x_3', we subtract f_1, f_2, and f_3 in (2.8)

from f_4 in (2.9):

$$\begin{array}{cc} & \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{array}{c} \\ f_4' \\ \Rightarrow \quad \bar{f}_4' \end{array} & \begin{pmatrix} 1 & 1 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & * & * & 0 & 0 \end{pmatrix} \end{array} \tag{2.10}$$

Note that the x_4 terms in f_1 and f_2 are cancelled out during the elimination.

5. Increase variable **index** by 1. The process is completed, if the SNI matrix becomes full-rank after substitution, otherwise we go back to step 3 and differentiate algebraic equations once more. The loop continues until a full-rank SNI matrix is detected.

   The termination of the whole process is another issue that may arise in practice. In general, if some variable x_i or its first derivative appears nonlinearly in the system, by the continuous differentiations of the algebraic constraints and substitutions of variables, we will have the nonlinear terms appear in every algebraic constraint in the system. This will transform the SNI matrix into a full-rank one. However, it is possible that the linear coefficient of a particular variable x_i is repeatedly being canceled after one or several rounds of processing. In this case, we may have an infinite loop. This indicates that the original DAE system is not well-setup and, consequently, nonsolvable, since one can never obtain the expression of x_i' by differentiation.

Example 2.12. Replacing the last row of (2.8) by (2.10), we obtain a new matrix:

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ \bar{f}_4' \end{array} \begin{array}{c} \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 \\ * & * & 0 & 0 \\ * & * & 0 & 0 \end{pmatrix} \end{array}$$

whose rank is still 3. We differentiate $\bar{f}_4'$ again in order to obtain the derivative of x_4.

$$\begin{array}{c} \\ \bar{f}_4' \\ \Rightarrow \; f_4'' \end{array} \begin{array}{c} \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ * & * & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \end{pmatrix} \end{array} \tag{2.11}$$

Performing substitution of x_1' and x_2' in (2.11) will give us

$$\begin{array}{c} \\ f_4'' \\ \Rightarrow \; \bar{f}_4'' \end{array} \begin{array}{c} \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \\ \begin{pmatrix} * & * & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & * \end{pmatrix} \end{array} \tag{2.12}$$

Finally, a third differentiation of (2.12) and substitution yield the desired form

that contains a x_4' term.

$$
\begin{array}{cc}
 & \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\
\begin{array}{c} \\ \bar{f}_4'' \\ \Rightarrow \; f_4''' \\ \Rightarrow \; \bar{f}_4''' \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & * & * & 0 & * \\ * & * & 0 & * & * & * & 0 & * \\ 0 & 0 & 0 & * & * & * & 0 & * \end{pmatrix}
\end{array}
$$

Upon completion, the final SNI matrix is

$$
\begin{array}{cc}
 & \begin{array}{cccc} x_1' & x_2' & x_3' & x_4' \end{array} \quad \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\
\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ \bar{f}_4''' \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & * & * & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 & * \end{pmatrix}
\end{array}
$$

It is not hard to see that the matrix is structurally full rank. Therefore the symbolic numeric index analysis succeeds and returns index 3. Meanwhile, if

we apply SA on this problem, it returns a Σ matrix and offsets:

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ d_j \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 1* & -\infty & -\infty & 0 \\ -\infty & 1 & -\infty & 0* \\ 0 & 0 & 1* & -\infty \\ 0 & 0* & 0 & -\infty \end{pmatrix} \\ \begin{array}{cccc} 1 & 1 & 1 & 0 \end{array} \end{array} \begin{array}{c} c_i \\ 0 \\ 0 \\ 0 \\ 1 \\ \end{array}$$

where the structural index is $\nu_T = \max_i c_i + 1 = 2$, which is less than the true index 3. One of the ways the SA can fail is when it is unable to catch the cancellation of x_4 in the process, as we have shown in the SNI analysis.

Chapter 3

Numerical Software

In this chapter, we present first the algorithms used in ASIA. Then we describe our program from a software engineering perspective, to discuss issues arising in the design and implementation, and explain how to use ASIA.

3.1 The Algorithms

Our automatic index-analysis software includes three major functional components: Structural index analysis, Computing the system Jacobian, and Symbolic-numeric index analysis. These components are implemented using operator overloading to avoid complicated source code transformation. Different algorithms for arithmetic operations and elementary functions have been applied in each component. We illustrate the process using pseudo code and simple examples.

3.1.1 Implementation of Structural Index Analysis

The Structural index analysis component implements Pryce's SA introduced in the previous chapter. Given a DAE system of the form (1.1), written in the required format, upon completion of the program, the software returns the Taylor index ν_T, equation and variable offsets $\mathbf{c}$ and $\mathbf{d}$, and HVT T.

The code for implementing the SA is separated into two parts:

- generating the signature matrix, and
- computing the index.

To generate the signature matrix, we associate a derivative array[1] with each variable and subexpression in an equation. The elements of these arrays indicate the highest order of derivatives of independent variables present in the DAE system.

Example 3.1. If a DAE system has independent variables $X = \{x_1, \ldots, x_4\}$, a possible subexpression and its corresponding derivative array could be:

$$x_1' + \cos x_3 + x_4'' + x_1 \times x_3 + 5, \qquad \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

Informally, operator overloading is applied in generating the signature matrix in the following way.

[1]This is not S.L. Campbell's derivative array approach in [CG95b].

1. Initially, each independent variable is assigned a derivative array.

2. When one or more variables are combined together by an arithmetic operation or elementary function, a new derivative array is created for this expression accordingly, whose entries are determined by rules that we will introduce shortly.

3. The previous step repeats until a derivative array is formed for each equation in the DAE system. Then, this derivative array is the corresponding row in the signature matrix.

Initialization of derivative arrays. From (2.1), we initialize the derivative array V of an independent variable x_i as:

$$V_j = \begin{cases} 1 & \text{if } i = j; \text{ or} \\ -\infty & \text{otherwise.} \end{cases} \tag{3.1}$$

Binary operations of derivative arrays. If an expression contains two variables or subexpressions, which are combined by one of the four binary arithmetic operators $\{+, -, \times, /\}$, the highest order derivative of each independent variable that appears in the resulting expression is obviously the higher one between the two components. Accordingly, the derivative array of the resulting expression is the componentwise maximum of the derivative arrays of two constituent variables or subexpressions. If one of the operands is a constant, the resulting derivative array is simply a copy of the counterpart of the other non-constant operand.

We use addition of derivative arrays as an example, where other operations follow the same procedure. Let S_1 and S_2 be two expressions with associated derivative arrays $S_1.array$ and $S_2.array$. We also assume both arrays are of the same size throughout the examples in this chapter. Then we have the following algorithm to compute the result of $S = S_1 + S_2$.

PLUS(S_1, S_2)

```
1  if  neither S1 nor S2 is constant
2     then S.array ← max(S1.array, S2.array)
3     else if  S1 is a constant
4             then S.array ← S2.array
5     else if  S2 is a constant
6             then S.array ← S1.array
7                  return S
```

Here max computes the componentwise maximum of $S_1.array$ and $S_2.array$.

Differentiation. As introduced before (1.1), the SA can handle DAEs with arbitrarily high-order differentiations. When a variable or expression is differentiated p times, by definition, each component of a derivative array is increased by p. Note that in (3.1), $-\infty$ is used to represent absence of independent variables in a corresponding subexpression. Therefore, when such expressions are differentiated, adding a constant

subexpressions	derivative array
x_1	$(0, -\infty, -\infty, -\infty)$
x_3	$(-\infty, -\infty, 0, -\infty)$
x_4	$(-\infty, -\infty, -\infty, 0)$
$x_1 \times x_3$	$(0, -\infty, 0, -\infty)$
$x_1 \times x_3 + 5$	$(0, -\infty, 0, -\infty)$
x_1'	$(1, -\infty, -\infty, -\infty)$
x_4''	$(-\infty, -\infty, -\infty, 2)$
$\cos x_3$	$(-\infty, -\infty, 0, -\infty)$
$x_4'' + x_1 \times x_3 + 5$	$(0, -\infty, 0, 2)$
$\cos x_3 + x_4'' + x_1 \times x_3 + 5$	$(0, -\infty, 0, 2)$
$x_1' + \cos x_3 + x_4'' + x_1 \times x_3 + 5$	$(1, -\infty, 0, 2)$

Table 3.1: Steps of generating a derivative array

n to the $-\infty$ will not destroy the correct representation.

Nonlinear smooth functions. We also implement common smooth functions (sin, cos, exp, power, log, etc.) for derivative arrays. Since applying these nonlinear functions does not change the highest order of a derivative, the resulting derivative array remains unchanged.

Example 3.2. Here we choose the same example used previously to illustrate the process of parsing an equation and generating the corresponding derivative array. With the same expression in Example 3.1, the steps of operator overloading execution are shown in Table 3.1.

The second part of the program is to compute the Taylor index ν_T (2.5) based on the signature matrix. The main step is to solve a linear assignment problem and find the HVT defined in the previous chapter.

We provide a LSOVLE function to compute the HVT and offsets of the signature matrix. In this function, we invoke a third party LAP solver from [JV03] through Matlab's external interface. This solver is based on the shortest augmenting path algorithm. A detailed description can be found in [JV87]. Some technical issues are also presented here. The LAP solver is written in C++ and actually computes the smallest value transversal (SVT) of an input matrix. In order to avoid dangerous computation involving infinities in C++ and to compute the HVT instead of the SVT, we pass a modified signature matrix, which is obtained by setting negative infinity to a large positive number and then reversing the sign of the whole matrix, to the LAP solver.

Upon completion of LSOLVE, we have the following output: two index vectors *rows* and *cols*, where $(rows_i, cols_i)$ are the entries of Σ that contribute to the HVT; and the two offsets **c** and **d** defined in (2.3).

Lastly, we have all the information to determine the structural index ν_T defined in (2.5). Overall, the function for determining the index is as follows.

COMPUTE-SAINDEX(Σ)

1 $[rows, cols, c, d]$ = LSOLVE(Σ)

2 $HVT = \sum \sigma_{rows_i, cols_i}$

3 **if** any component of d is zero

4 **then** $index = \max(c) + 1$

5 **else** $index = \max(c)$

6 **return** $HVT, rows, cols, c, d, index$

3.1.2 Computing the System Jacobian

In [NP05a], Nedialkov and Pryce present a source code translation algorithm for computing the Jacobian. Although likely very efficient, it is complicated in implementation. Another approach is applied in ADOL-C [GW04]. Here we implement a scheme based on operator overloading. Before we introduce the rules of computing gradient arrays, let us introduce two definitions.

A *code list* is a sequence of variables and expressions that consists of fundamental arithmetic operations and other functions.

The *code offset* of an expression v is the unique value such that

$$\alpha(v) = \min_j (d_j - \sigma_j(v)) \geq 0, \tag{3.2}$$

where $\sigma_j(v)$ is the highest order of derivative of x_j on which v depends. The computation of $\alpha(v)$ can be based on the following rules.

1. If $v = x_j$, then

$$\alpha(v) = d_j.$$

2. If $v = f_i$, then

$$\alpha(v) = c_i.$$

3. If v is a function of variables X, then

$$\alpha(v) = \min(\alpha(X)).$$

4. If $v = d^p u/dt^p$, then

$$\alpha(v) = \alpha(u) - p.$$

The complete proof of this is in [NP05a].

With the two definitions, we are ready to show how to apply the operator overloading of gradient arrays on the expressions in the code list. Generally, the strategy for computing the Jacobian is described below.

1. For each variable or expression in the code list, we associate three types of information.
   - A gradient array to store the gradient vector of this variable or expression.
   - A code offset computed following the rules introduced above.
   - The value of this expression at the given point.
2. Initially, for each variable x_i, every component of its gradient array is set to 0, except the ith one, which is set to 1. The initial code offsets for x_i are the

code list	gradient array	code offset
x	(1, 0, 0)	2
y	(0, 1, 0)	2
λ	(0, 0, 1)	0

Table 3.2: Initialization for the Single Pendulum

corresponding variable offsets d_i obtained from the SA analysis. We also store the initial value of each x_i, which is given as a consistent point.

3. The gradient arrays are propagated through the code list under special rules specified for each fundamental arithmetic operation and elementary function.

4. The code offset of the expression is computed based on those of its subexpressions.

5. The value of the expression is evaluated and stored.

6. The above three steps repeat until the gradient array for the right-hand-side expression of each equation in the system is obtained. These gradient arrays consist of the Jacobian matrix evaluated at the given consistent point.

Initialization of gradient array and code offset. We set the components of an associated gradient array of independent variable x_i to be the ith unit vector e_i. The code offset of x_i is d_i obtained from the SA by (3.2).

Example 3.3. For the single pendulum, we have the initialization in Table 3.2.

Differentiation. In [NP05a], the authors present Griewank's Lemma, which states that if v, a function of x_j and its derivatives, does not depend on any derivatives of x_j higher than the qth, then

$$\frac{\partial v^{(p)}}{\partial x_j^{(q+p)}} = \frac{\partial v}{\partial x_j^{(q)}}, \qquad \text{for all } p \geq 0. \tag{3.3}$$

Denote

$$\nabla_k = \left(\frac{\partial}{\partial x_1^{(k+d_1)}}, \ldots, \frac{\partial}{\partial x_n^{(k+d_n)}} \right). \tag{3.4}$$

By setting $q = k + d_j$ in (3.3), and applying the notation of ∇_k, we have

$$\nabla_{k+p} \left(\frac{d^p v}{dt^p} \right) = \nabla_k(v), \tag{3.5}$$

and if $k = -\alpha(v)$, $v = d^p u/dt^p$ then (3.5) becomes

$$\nabla_{-\alpha(v)}(v) = \nabla_{-\alpha(u)}(u). \tag{3.6}$$

A detailed proof is in [NP05a]. By (3.2) and (3.6), the differentiation operator preserves the gradient array of the operand, and the code offset decreases by the order of differentiation.

DIF(u, p)

1 **if** $v = d^p u/dt^p$

2 **then** $v.gradient \leftarrow u.gradient$

3 $v.offset \leftarrow u.offset - p$

4 **return** v

Arithmetic operations on gradient arrays. We show how to compute the result of binary arithmetic operations of two gradient arrays. First, each ∇_k obeys regular rules of gradient operations. For example, if $v = xy$, then

$$\nabla_k(v) = \nabla_k(x)y + \nabla_k(y)x.$$

Generally speaking, if v is an algebraic function $f(x, y, \ldots)$, we have

$$\nabla_k(v) = \frac{\partial f}{\partial x}\nabla_k(x) + \frac{\partial f}{\partial y}\nabla_k(y) + \cdots.$$

Second, from (3.2), we know that for any v in the code list, it does not depend on any derivatives of x_j higher than the $(d_j - \alpha(v))$th. Therefore, applying (3.4) in (3.3) when $k > -\alpha(v)$, we obtain

$$\nabla_k(v) = 0, \quad \text{if} \quad k > -\alpha(v).$$

Following the two properties above, we can conclude the rules of computing gradient arrays through arithmetic operations as follows. For $v = v_1 \circ v_2$, where $\circ \in \{+, -, *, /\}$, the gradient of v is computed by the normal rules of gradient operations with one prerequisite that if the code offset of v_1 is greater than that of v_2, then we replace the gradient of v_1 by 0 in the formula, and vice versa. We show the pseudo code of multiplication as an example.

TIMES(v_1, v_2)

1 **if** $v_1.offset = v_2.offset$

code list	gradient array	code offset
$t_1 = x_0$	(1, 0, 0)	2
$t_2 = \lambda_0$	(0, 0, 1)	0
$t_3 = x_0\lambda_0$	(0, 0, x_0)	0
$t_4 = x_0''$	(1, 0, 0)	0
$t_5 = t_4 + t_3$	(1, 0, x_0)	0
$f = t_5$	(1, 0, x_0)	0

Table 3.3: Evaluation of code list for $f = x'' + x\lambda$.

```
2    then v.gradient ← v1 × v2.gradient + v2 × v1.gradient
3         v.offset ← v1.offset
4    else if v1.offset > v2.offset
5              then v.gradient ← v1 × v2.gradient
6                   v.offset ← v2.offset
7              else if v2.offset > v1.offset
8                        then v.gradient ← v2 × v1.gradient
9                             v.offset ← v1.offset
10                            return v
```

Example 3.4. For the single pendulum, the initialization step is shown in (3.2). Now we illustrate the evaluation of gradient code list in Tables 3.3, 3.4, and 3.5.

code list	gradient array	code offset
$t_1 = y_0$	$(0, 1, 0)$	2
$t_2 = \lambda_0$	$(0, 0, 1)$	0
$t_3 = y_0\lambda_0$	$(0, 0, y_0)$	0
$t_4 = y_0''$	$(0, 1, 0)$	0
$t_5 = t_4 + t_3$	$(0, 1, y_0)$	0
$t_6 = t_5 + G$	$(0, 1, y_0)$	0
$g = t_6$	$(0, 1, y_0)$	0

Table 3.4: Evaluation of code list for $g = y'' + y\lambda + G$.

code list	gradient array	code offset
$t_1 = x_0$	$(1, 0, 0)$	2
$t_1 = y_0$	$(0, 1, 0)$	2
$t_3 = x_0^2$	$(2x_0, 0, 0)$	2
$t_4 = y_0^2$	$(0, 2y_0, 0)$	2
$t_5 = t_4 + t_3$	$(2x_0, 2y_0, 0)$	2
$t_6 = t_5 - L^2$	$(2x_0, 2y_0, 0)$	2
$h = t_6$	$(2x_0, 2y_0, 0)$	2

Table 3.5: Evaluation of code list for $h = x^2 + y^2 - L^2$.

3.1.3 Implementation of Symbolic Numeric Analysis

Similar to the SA, the symbolic numeric analysis is also divided into two major steps: generating the SNI matrix and computing the symbolic numeric index. We will first introduce the rules of operator overloading and then the algorithm for computing a symbolic numeric index.

In (2.6), each row in the SNI matrix is defined to represent coefficients of linear terms as well as the first order derivatives of each variable x_i. Accordingly, we define a coefficient array associated with each variable and subexpression in the equation. Along with the computation, the coefficient arrays are gradually updated, and finally form the rows of the SNI matrix.

Example 3.5. Using the same notation of SNI matrix from (2.6), here is a simple example of an equation, where the independent variables are $X = \{x_1, x_2, x_3\}$, and its associated coefficient array:

$$0 = 2x_1' + \cos x_3 + x_1 \times x_3 + 5, \qquad \begin{array}{cccccc} x_1' & x_2' & x_3' & x_1 & x_2 & x_3 \\ \multicolumn{6}{c}{\left(\begin{array}{cccccc} 2 & 0 & 0 & * & 0 & * \end{array}\right)} \end{array}$$

Initialization of coefficient array. Similar to the SA, our program for SNI analysis also requires an initialization of the coefficient arrays for each independent variable x_i.

Example 3.6. Given the same DAE system as (2.7), the initialization step is shown

code list	coefficient array
x_1	(0, 0, 0, 0, 1, 0, 0, 0)
x_2	(0, 0, 0, 0, 0, 1, 0, 0)
x_3	(0, 0, 0, 0, 0, 0, 1, 0)
x_4	(0, 0, 0, 0, 0, 0, 0, 1)

Table 3.6: Initialization of generating SNI matrix

in Table 3.6.

Arithmetic operations on coefficient arrays. Since the numbers stored in the array correspond to coefficients, when we add or subtract two variables or other expressions in the code list, we only need to perform the same operation on the corresponding coefficients of each variable in the two operands to obtain a new coefficient array for the resulting expression. If a constant is added to or subtracted from an expression, the coefficient array is unchanged.

PLUS(S_1, S_2)

1 **if** neither S_1 nor S_2 is constant
2 **then** $S.array \leftarrow S_1.array + S_2.array$
3 **else if** S_1 is a constant
4 **then** $S.array \leftarrow S_2.array$
5 **else if** S_2 is a constant
6 **then** $S.array \leftarrow S_1.array$
7 **return** S

The multiplication and division for coefficient arrays are slightly more complicated than addition and subtraction. When two subexpression are combined with a nonlinear operation, all variables that appear in both operands will become nonlinear. We use multiplication as an example to show the algorithm. For division, an extra check for divide-by-zero is imposed.

TIMES(S_1, S_2)

```
1  if  neither S1 nor S2 is constant
2     then for i ← 1 to n
3              do if either S1.array(i) or S2.array(i) is nonzero
4                    then S.array(i) ← *
5                    else  S.array(i) ← 0
6     else  if  S1 is a constant
7              then S.array ← S1 × S2.array
8     else  if  S2 is a constant
9              then S.array ← S2 × S1.array
10                   return S
```

Differentiation of coefficient array. Since the SNI analysis only handles first order DAE systems, the differentiation operation can only be applied to an expression

code list	coefficient array
x_1	(0, 0, 0, 0, 1, 0, 0, 0)
x_4	(0, 0, 0, 0, 0, 0, 0, 1)
x_1'	(1, 0, 0, 0, 0, 0, 0, 0)
$x_1' - x_4$	(1, 0, 0, 0, 0, 0, 0, −1)

Table 3.7: Evaluation of $f_1 = x_1' - x_4$.

that does not contain any derivatives. Therefore, the result is obtained by shifting the coefficients of x_i to the coefficients of x_i'.

Nonlinear smooth functions. The appearance of nonlinear functions, such as trigonometric functions, the exponential function, etc., in an equation simply indicates nonlinearity of variables. Accordingly, all nonzero components in the coefficient array will be set to $*$ to reflect this fact. Here we use a cos function as an example.

Cos(S_1)

1 **if** $S_1.array(i)$ is nonzero

2 **then** $S.array(i) \leftarrow *$

3 **else** $S.array(i) \leftarrow 0$

4 **return** S

Example 3.7. We illustrate the whole process of generating the SNI matrix for the DAE system (2.7) in Tables 3.7, 3.8, 3.9, and 3.10.

Computing the SNI matrix involves symbolic-numeric Gaussian elimination on

code list	coefficient array
x_2	(0, 0, 0, 0, 0, 1, 0, 0)
x_4	(0, 0, 0, 0, 0, 0, 0, 1)
x_2'	(0, 1, 0, 0, 0, 0, 0, 0)
$x_2' - x_2$	(0, 1, 0, 0, 0, −1, 0, 0)
$x_2' - x_2 + x_4$	(0, 1, 0, 0, 0, −1, 0, 1)

Table 3.8: Evaluation of $f_2 = x_2' - x_2 + x_4$.

code list	coefficient array
x_1	(0, 0, 0, 0, 1, 0, 0, 0)
x_2	(0, 0, 0, 0, 0, 1, 0, 0)
x_3	(0, 0, 0, 0, 0, 0, 1, 0)
x_3'	(0, 0, 1, 0, 0, 0, 0, 0)
x_2^2	(0, 0, 0, 0, 0, ∗, 0, 0)
x_1^2	(0, 0, 0, 0, ∗, 0, 0, 0)
$x_3' - x_1^2$	(0, 0, 1, 0, ∗, 0, 0, 0)
$x_3' - x_1^2 - x_2^2$	(0, 0, 1, 0, ∗, ∗, 0, 0)

Table 3.9: Evaluation of $f_3 = x_3' - x_1^2 - x_2^2$.

code list	coefficient array
x_1	(0, 0, 0, 0, 1, 0, 0, 0)
x_2	(0, 0, 0, 0, 0, 1, 0, 0)
x_3	(0, 0, 0, 0, 0, 0, 1, 0)
$4x_3$	(0, 0, 0, 0, 0, 0, 4, 0)
$x_1 + x_2$	(0, 0, 0, 0, 1, 1, 0, 0)
$x_1 + x_2 + 4x_3$	(0, 0, 0, 0, 1, 1, 4, 0)

Table 3.10: Evaluation of $f_4 = x_1 + x_2 + 4x_3$.

the SNI matrix. Function COMPUTE-SNINDEX initializes the index to be 0, and then invokes FINDRANK which executes LU factorization steps to determine the rank of the SNI matrix. If the matrix is found to be structurally full rank, the program stops and returns index 0; otherwise, FINDALGEQNS is called to find algebraic equations in the system and differentiates them once by calling DIFFEQNS. Accordingly, the index is also increased by 1. After differentiation, LU factorization is executed again to compute the rank. This procedure is repeated until a full-rank matrix is determined.

COMPUTE-SNINDEX($SNImatrix, dim$)

```
1  index ← 0
2  tempRank ← FINDRANK(SNImatrix, dim, tol)
3  if tempRank = dim
4     then exit
5     else  algeqns ← FINDALGEQNS(SNImatrix, dim)
6           while true
7              do SNImatrix ← DIFFEQNS(SNImatrix, dim, algeqns)
8                 index ← index + 1
9                 tempRank ← FINDRANK(SNImatrix, dim, tol)
10                if tempRank = Dim
11                   then break
12                   else  algeqns ← FINDALGEQNS(SNImatrix, dim)
```

13 **return** *index*

The algorithm implemented in FINDRANK is symbolic-numeric type of LU factorization with partial pivoting. It performs equivalence transformations and permutations with those of the regular linear algebra, but using the symbolic numeric operations defined in (2.1). As mentioned in the previous chapter, round off error may affect the correctness of the result. To avoid comparing the pivot with zero, we introduce a user defined tolerance, so any number smaller than the tolerance will be regarded as zero.

3.2 Software

In this section, we follow the procedure of software development and describe the specification and design issues raised during the development.

3.2.1 General description

As mentioned previously, the system contains three components, which deploy the three analyses and computations introduced in Chapter 2. In these three components, the computation of SA and SNI are independent. The user needs to provide a system of DAEs written in the specified form. The system returns the signature matrix Σ, offsets $\mathbf{c}$ and $\mathbf{d}$, the HVT T, and the stuctural index ν_T; or the SNI matrix, and

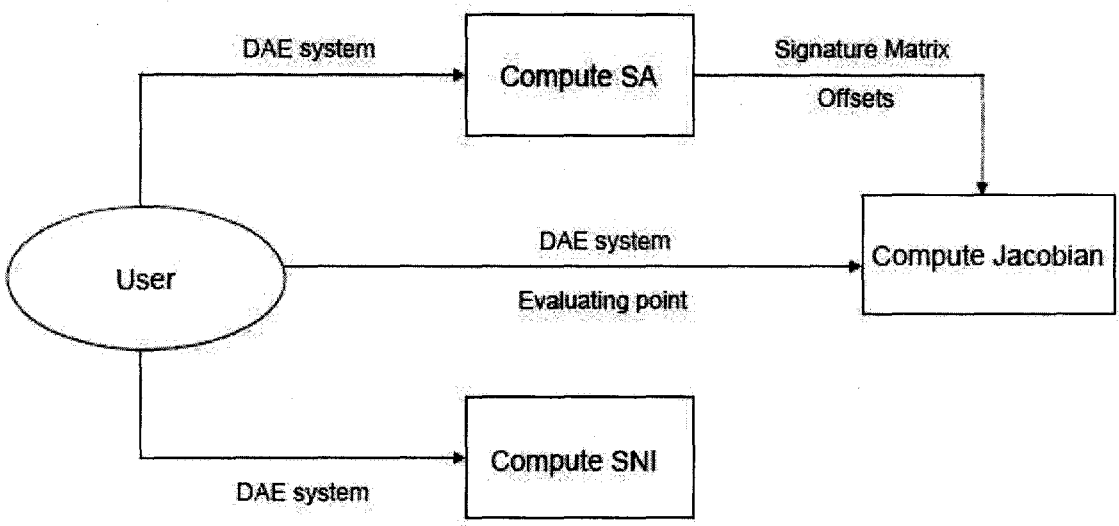

Figure 3.1: Hierarchy diagram of the system.

SNI index in the case of SNI analysis. The computation of system Jacobian partially relies on the results from the SA component. The user is responsible for preparing a system of DAEs and a point at which the Jacobian is evaluated. In addition, the variable offset $\mathbf{d}$ obtained from SA computation is also required. Upon completion, the system returns the Jacobian matrix, a condition number indicating the reliability of the computation. A sketch of the system is presented in Figure 3.1.

3.2.2 Specification

Here we give an informal functional specification of our software package.

1. The software handles a DAE system of the form (1.1). The DAE system must have an equal number of equations and variables. It should consist of funda-

mental arithmetic operations $\{+, -, \times, /\}$, differential operators, smooth continuous functions such as trigonometric functions, the exponential function, etc. No nonsmooth functions, such as min, abs, etc., are allowed in the DAE system.

2. The goal of this software is to facilitate automatic index analysis of DAE systems. We want to provide the user with a tool that can apply different types of analysis methods. This tool can give the structural information obtained through the analysis. Because of the potential failure of both structural analyses in some problems, we also want the software to produce useful information and suggestions on possible adjustments that can be done to correct the inaccurate index computation.

3. The output of the software contains several components.

   - The corresponding signature matrix of the input DAE.
   - A signature matrix object containing the signature matrix. This is used for an auxiliary function to print the numerical results in a figure.
   - Structural information: the HVT, the index vectors of Σ entries contributing to the HVT, offsets $\mathbf{c}$ and $\mathbf{d}$, and structural index.
   - The symbolic-numeric matrix of the input system.
   - The SNI index.
   - The system Jacobian evaluated at a given point.

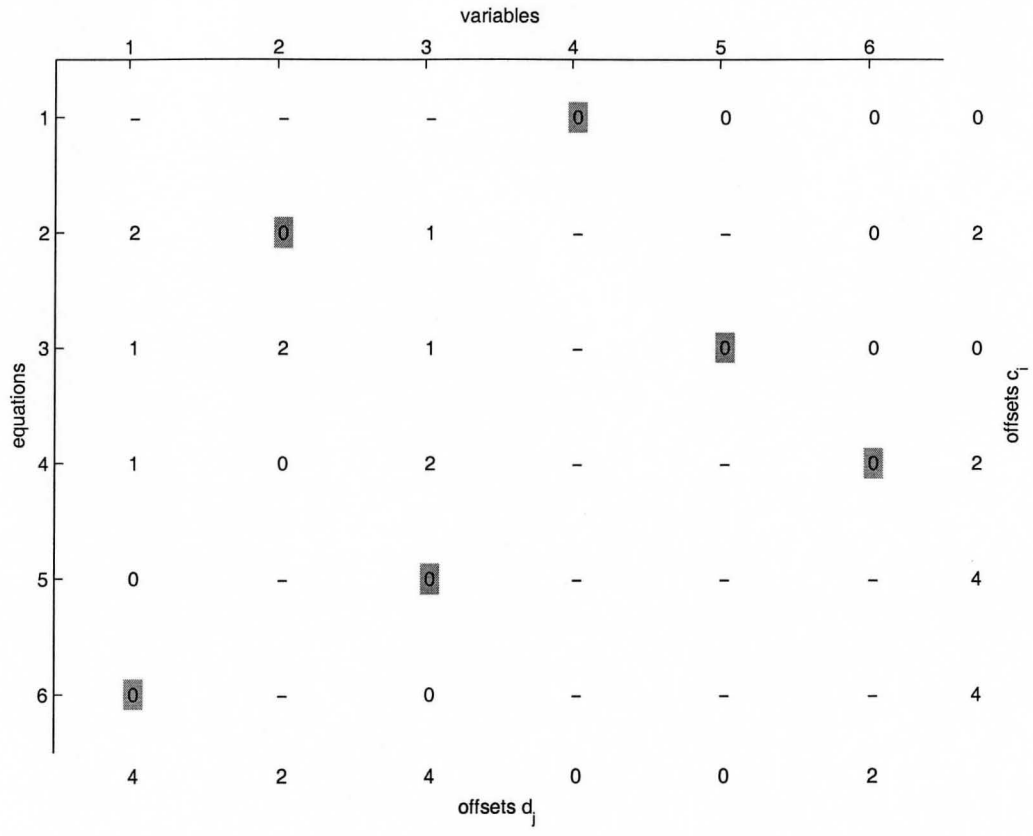

Figure 3.2: Signature matrix of robotic arm

- The condition number of the Jacobian.

4. An auxiliary function is provided to plot a figure to show the structural information. A sample print is in Figure 3.2.

3.2.3 Interface

The goal of the software is to provide a tool for scientists and engineers to analyze the index of DAE systems that arise in applications. Therefore, users of this tool may or

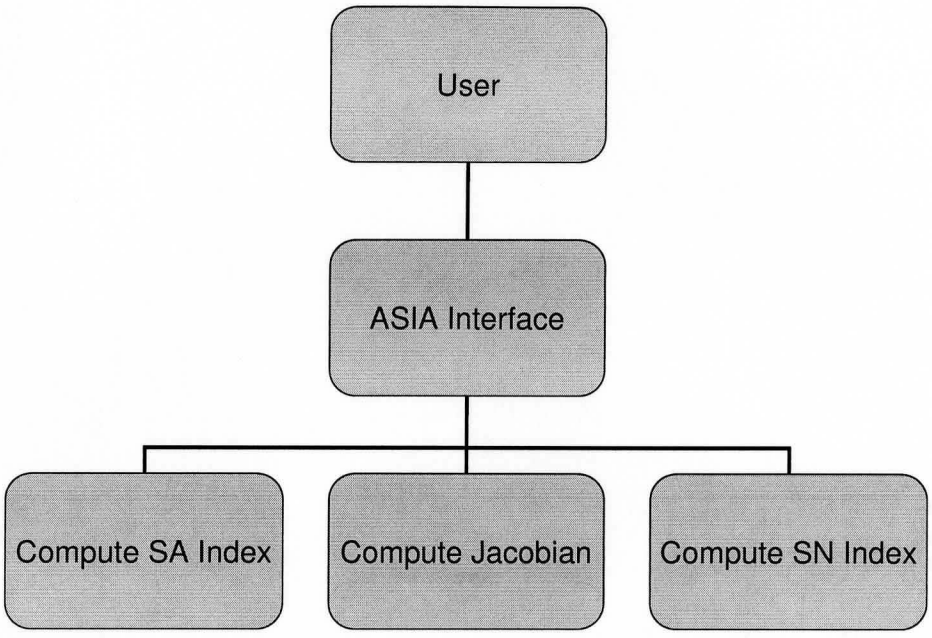

Figure 3.3: Interface design of ASIA

may not have advanced knowledge on the theory of index analysis, but they require a reliable result on the structural index of a DAE problem to conduct an appropriate strategy in solving the problem numerically on the next step. On the other hand, due to the lack of generic algorithms that always return correct structural indices for general problems, one also expect the software to return a reference that can indicate the correctness of the index computation.

We design the interface of ASIA to be compact and straightforward, so that the users need not to set the complicated parameters of index analysis to be able to obtain

the result they require. A driver is implemented, so that the user only supplies the system of equations and the dimension of the problem, and the driver invokes the corresponding functions to perform the analysis. This structure is shown in Figure 3.3. After finishing the computation, the driver returns an object. It contains the computed index as well as the Jacobian matrix and its condition number, which serve as the reference for the correctness of the computation. The interface is given as

```
function myresult = DAEAnl(myFunc, dim, option, y)
```

The arguments are listed below.

`myFunc` The name of the file defining the target DAE system. Its format is shown in the next subsection.

`dim` The dimension of the DAE system.

`option` A string determines the mode of computation.

- FULL: The software performs both Pryce's structural analysis and the Symbolic-numeric Analysis, and computes the system Jacobian. Only applicable to first order systems.
- SIMPLE: The software performs Pryce's structural analysis only, and computes the system Jacobian. Applicable to systems of arbitrary orders.

y An optional argument that specifies the point at which the system Jacobian is evaluated. If this argument is not provided, we randomly choose a set of numbers, which are uniformly distributed on the unit interval with mean 0, using Matlab random number generating function.

3.2.4 Installation and Usage

This section describes how to install and use the ASIA package. ASIA uses the LAP solver from [JV03][2]. This third party library is included in our package. In order to compile the Matlab/C++ API function, one needs to have the corresponding C/C++ compiler installed. We have compiled and tested on Solaris 9 with GNU gcc/g++ v3.4 compiler. After compilation, ASIA can be used in any platform with Matlab version 6.5 or later installed.

Content of the package

The file structure of the ASIA package is shown in Figure 3.4. In this diagram, entities in bold font represent folders, while others represent single functions.

Installing ASIA

The following steps are needed to install ASIA.

- Download ASIA package.

[2]The algorithm can be found in [JV87]

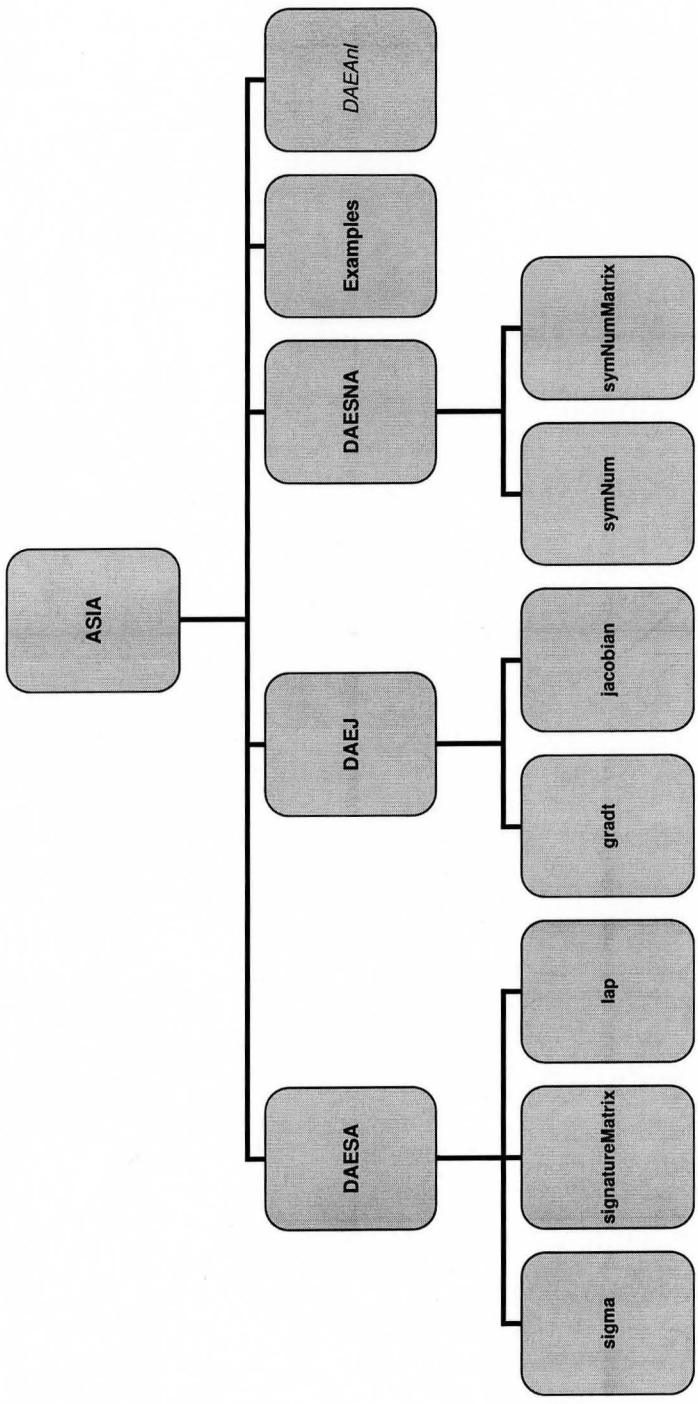

Figure 3.4: Organization of ASIA.

- Add related directories to Matlab path.
- Set up the mex compiler.
- Compile the mex file in /lap folder.

Using ASIA

The user provides a system of equations and the dimension of the problem. Below we go through each step of analyzing the single pendulum problem to illustrate the use of ASIA.

1. **Set system equations.** The user must give a DAE system written in required form. Here is the definition of the single pendulum problem in pendulum.m.

```
function f=pendulum(n, y, t)

f(1)=dif(y(1),2)+y(1)*y(3);
f(2)=dif(y(2),2)+y(2)*y(3)-1.0;
f(3)=y(1)*y(1)+y(2)*y(2)-1.0*1.0;
```

2. **Invoke the DAEAnl.** After setting up the equations, we are ready to use the analyzer. To invoke the software, one can type the following line in the Matlab command window.

```
singlePendulumResult = DAEAnl(@singlePendulum, 3, 'SIMPLE', [1, 0,
0])
```

The object returned by the above command contains the following attributes.

- `sigMatrix`: the signature matrix of the DAE system
- `hvt`: the highest value transversal of the signature matrix
- `indexVector`: two index vectors, containing indices of the entries contributing to HVT
- `SAindex`: the structural index of the DAE system
- `offset`: the offset vectors $\mathbf{c}$ and $\mathbf{d}$ where $d(i) - c(j) \geq s(i,j)$
- `point`: the consistent point at which the Jacobian is evaluated
- `jacob`: the system Jacobian
- `condJ`: the condition number of the Jacobian
- `symnMatrix`: the SNI matrix
- `SNindex`: the SN index of the DAE system

To fetch an attribute, one can use the corresponding functions provided. Some examples are shown below.

```
>> singlePendulum = DAEAnl(@pendulum, 3, 'SIMPLE');
```

```
>> s = getSigMatrix(singlePendulum)

s =

     2  -Inf     0

  -Inf     2     0

     0     0  -Inf

>> i = getSAindex(singlePendulum)

i =

     3

>> [c d] = getOffset(singlePendulum)

c =

     0     0     2

d =

     2     2     0

>> condJ = getcondJ(singlePendulum)

condJ =

    2.6006
```

3. **Interpretation of the result.** According to the theory of Pryce's analysis [NP05b], the nonsingularity of the Jacobian serves as an indicator of the correctness of HVT computation. Here, if the condition number is moderate compared to the norm of the input vector at which the Jacobian is evaluated, then we consider the HVT and structural index computed by ASIA to be reliable.

Chapter 4

Numerical Experiments

We conduct numerical experiments using ASIA on various DAE problems, including the single pendulum [AP98], the double pendula [Pry98], the two-link robotic arm problem [Pry01] and its modified version, the car axis problem [MI03], an example of SA failure from [KL04], the chemical Akzo Nobel problem [MI03], the transistor amplifier problem [MI03], and the ring modulator problem in ODE, DAE and modified DAE formats [MI03].

4.1 Structure of the Experiments

For each problem, we present

1. Background information

2. Mathematical description

3. Numerical results

In the numerical results, we present the following information.

1. **Structural information of the problem.** Four types of structural information that we present are listed below. We compare our solution with the solution in the literature to verify the correctness of ASIA.

   - Structural Index
   - HVT
   - Variable Offsets
   - Equation Offsets

2. **The system Jacobian matrix with its condition number.** We present this information as an auxiliary reference to show the reliability of index computation.(See Chapter 2 for details.)

3. **Comparison between the two index analyses.** If applicable, we compare the two analyses to obtain some heuristic approaches for computing correct DAE indices.

4.2 Single Pendulum

4.2.1 Background Information

These equations model the behavior of a pendulum movement. It is a nonstiff DAE system with 3 variables.

4.2.2 Mathematical Description

The equations are

$$0 = x'' + x\lambda,$$

$$0 = y'' + y\lambda - G,$$

$$0 = x^2 + y^2 - L^2,$$

where L is the length of the pendulum, G is a gravity; x, y are Cartesian coordinates of the pendulum bob with y pointing downwards; λ characterizes the tension in the string. We choose $L = 1$ for simplicity. (See also Equation 2.2 in Chapter 2.)

4.2.3 Numerical Results

In [AP98], the single pendulum is given as a nonstiff DAE problem of index 3. The structural information returned by our program is shown in Table 4.1.

The signature matrix of the single pendulum is given in Table 4.2. Here the entries

Structural information	Results
Structural Index	3
HVT	2
Variable Offsets	(0, 0, 2)
Equation Offsets	(2, 2, 0)

Table 4.1: Structural index analysis of single pendulum

$$\begin{array}{c} \\ f \\ g \\ h \\ d_j \end{array} \begin{array}{c} \begin{array}{ccc} x & y & \lambda \end{array} \\ \begin{pmatrix} 2 & -\infty & 0* \\ -\infty & 2* & 0 \\ 0* & 0 & -\infty \end{pmatrix} \\ \begin{array}{ccc} 2 & 2 & 0 \end{array} \end{array} \begin{array}{c} c_i \\ 0 \\ 0 \\ 2 \\ \end{array}$$

Table 4.2: Signature matrix for single pendulum

marked by $*$ represent those in the HVT.

In [AP98], the authors point out that this problem can have arbitrary initial values. We evaluate the system Jacobian at a consistent point and compute its condition number, as shown in Table 4.3.

For the single pendulum, we can see that ASIA returns the correct index and structural information. The system Jacobian evaluated at one initial point is structurally nonsingular, therefore the computed result from ASIA is reliable.

var.	value
x	$-4.5766268835131380e-001$
y	$8.8912589867298431e-001$
λ	3.6673776960190421
condition number	$2.6180339887498999e+000$

Table 4.3: Consistent point and condition number for single pendulum

4.3 Double Pendula

4.3.1 Background Information

Double pendula consist of one pendulum attached to another. The problem is a simple example of dynamic system which can exhibit chaotic behavior. It contains 4 differential equations and 2 algebraic constraints.

4.3.2 Mathematical Description

The problem is given in the form

$$\begin{aligned} 0 &= x'' + x\lambda & 0 &= u'' + u\kappa \\ 0 &= y'' + y\lambda - g & 0 &= v'' + v\kappa - g \\ 0 &= x^2 + y^2 - L^2 & 0 &= u^2 + v^2 - (L + c\lambda)^2 \end{aligned}$$

Here we take $L = 1$, $g = 1$, and $c = 0.1$; x, y, λ, u, v, and κ are the dependent variables.

4.3.3 Numerical Results

This double pendula problem is given as a nonstiff DAE problem of index 5 in [Pry01]. The structural information returned by our program is shown in Table 4.4. The signature matrix of the double pendula is given in Table 4.5. A consistent point is $(1, 0, 0, 1, 0, 0)$. We evaluate the system Jacobian at this point and obtain the result

Structural information	Results
Structural Index	5
HVT	4
Variable Offsets	(2, 2, 4, 0, 0, 2)
Equation Offsets	(4, 4, 2, 2, 2, 0)

Table 4.4: Structural index analysis of double pendula

	x	y	λ	u	v	κ	c_i
f_1	2	$-\infty$	$0*$	$-\infty$	$-\infty$	$-\infty$	2
f_2	$-\infty$	$2*$	0	$-\infty$	$-\infty$	$-\infty$	2
f_3	$0*$	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	4
f_4	$-\infty$	$-\infty$	$-\infty$	2	$-\infty$	$0*$	0
f_5	$-\infty$	$-\infty$	0	$-\infty$	$2*$	$-\infty$	0
f_6	$-\infty$	$-\infty$	0	$0*$	0	$-\infty$	2
d_j	4	4	0	2	2	2	

Table 4.5: Signature matrix for double pendula

in Table 4.6. The condition number of the Jacobian is $2.808452574757240e + 000$. From this, we can conclude that the Jacobian matrix is structurally nonsingular at the consistent point. Therefore, the structural index computed for the double pendula is reliable.

	x	y	λ	u	v	κ
f_1	1	0	1	0	0	0
f_2	0	1	0	0	0	0
f_3	2	0	0	0	0	0
f_4	0	0	0	1	0	1
f_5	0	0	0	0	1	0
f_6	0	0	-0.2	2	0	0

Table 4.6: Jacobian matrix for double pendula

4.4 Two-link Robotic Arm

4.4.1 Background Information

This example is a slight simplification of equations for the prescribed-path control of a two-link robotic arm [Pry98]. It is a DAE of 3 differential equations and 3 algebraic constraints.

4.4.2 Mathematical Description

The problem is of the form

$$
\begin{aligned}
0 &= x_1'' - [2c(x_3)(x_1' + x_3')^2 + d(x_3)x_1'^2 \\
&\quad + (2x_3 - x_2)(a(x_3) + 2b(x_3)) + a(x_3)u_1 - a(x_3)u_2], \\
0 &= x_2'' - [-2c(x_3)(x_1' + x_3')^2 - d(x_3)x_1'^2 \\
&\quad + (2x_3 - x_2)(1 - 3a(x_3) - 2b(x_3)) - a(x_3)u_1 + (a(x_3) + 1)u_2], \\
0 &= x_3'' - [-2c(x_3)(x_1' + x_3')^2 - d(x_3)x_1'^2 \\
&\quad + (2x_3 - x_2)(a(x_3) - 9b(x_3)) - 2x_1'^2 c(x_3) - d(x_3)(x_1' + x_3')^2 \\
&\quad - (a(x_3) + b(x_3))u_1 + (a(x_3) + b(x_3))u_2], \\
0 &= \cos x_1 + \cos(x_1 + x_3) - p_1(t), \\
0 &= \sin x_1 + \sin(x_1 + x_3) - p_2(t),
\end{aligned}
$$

Structural information	Results
Structural Index	3
HVT	2
Variable Offsets	(0, 0, 0, 2, 2)
Equation Offsets	(2, 2, 2, 0, 0)

Table 4.7: Structural index analysis of robotic arm

where

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

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

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

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

By construction, the solution has $x_1 = 1 - e^t$, $x_3 = e^t - t$.

4.4.3 Numerical Results

In [CG95b], the authors show that this DAE has index 5 via their derivative-array equation method. The structural information obtained through our experiment is shown in Table 4.7.

The software returns index 3, which is lower than the correct index 5. In order to confirm the failure of Pryce's analysis in this example, we compute the system Jacobian and its condition number at the consistent point, as shown in Table 4.8.

It is clear to see that we have a large condition number, which indicates that the

var.	value
x_1	-2.6692966676192422
x_2	2.6578533275805367
x_3	2.3692966676192442
μ_1	$2.1507094761478751e+001$
μ_2	$2.2158319076934220e+001$
condition number	$7.9259796473774460e+016$

Table 4.8: Consistent point and condition number for robotic arm

system Jacobian is likely singular, therefore the index computation above may be incorrect.

Alternatively, we identify common subexpressions in the original equations, as done in [Pry01], and have the following modified version.

$$0 = x_1'' - [\nu + X(a(x_3) + 2b(x_3)) + a(x_3)\omega],$$

$$0 = x_2'' - [-\nu + X(1 - 3a(x_3) - 2b(x_3)) - a(x_3)\omega + \mu_2],$$

$$0 = x_3'' - [-\nu + X(a(x_3) - 9b(x_3)) - 2{x_1'}^2 c(x_3) - d(x_3){Y'}^2 - (a(x_3) + b(x_3))\omega],$$

$$0 = \cos x_1 + \cos(Y) - p_1(t),$$

$$0 = \sin x_1 + \sin(Y) - p_2(t),$$

$$0 = \omega - (\mu_1 - \mu_2),$$

where

$$X = 2x_3 - x_2, \quad Y = x_1 + x_3,$$

$$\nu = 2{Y'}^2 c(x_3) + {x_1'}^2 d(x_3).$$

Structural information	Results
Structural Index	5
HVT	0
Variable Offsets	(0, 2, 0, 2, 4, 4)
Equation Offsets	(4, 2, 4, 0, 0, 2)

Table 4.9: Structural index analysis of modified robotic arm

The introduction of a new variable ω is essential. Applying the new system into our software, we obtain the correct structural information in Table 4.9. We use the same consistent point shown in Table 4.8, and calculate the system Jacobian again. The condition number obtained is $2.660343827951100e + 001$. We can see clearly that the condition number for the modified system is small, indicating that the index computation is correct.

4.5 Car Axis

4.5.1 Background Information

The problem is a stiff DAE system consisting of 8 differential and 2 algebraic equations. It is originally defined in [Sch94].

4.5.2 Mathematical Description

The problem is of the form

$$\begin{aligned} p' &= q, \\ Kq' &= f(t,p,\lambda), \qquad p,q \in \mathbb{R}^4,\ \lambda \in \mathbb{R}^2,\ 0 \le t \le 3, \\ 0 &= \phi(t,p), \end{aligned}$$

with initial conditions $p(0) = p_0$, $q(0) = q_0$, $p'(0) = q_0$, $q'(0) = q_0'$, $\lambda(0) = \lambda_0$, and $\lambda'(0) = \lambda_0'$.

The matrix K reads $\epsilon^2 \frac{M}{2} I_4$, where I_4 is the 4×4 identity matrix. The function $f : \mathbb{R}^9 \to \mathbb{R}^4$ is given by

$$f(t,p,\lambda) = \begin{pmatrix} (l_0 - l_l)\dfrac{x_l}{l_l} & +\lambda_1 x_b + 2\lambda_2(x_l - x_r) \\ (l_0 - l_l)\dfrac{y_l}{l_l} & +\lambda_1 y_b + 2\lambda_2(y_l - y_r) - \epsilon^2\dfrac{M}{2} \\ (l_0 - l_r)\dfrac{x_r - x_b}{l_r} & -2\lambda_2(x_l - x_r) \\ (l_0 - l_r)\dfrac{y_r - y_b}{l_r} & -2\lambda_2(y_l - y_r) - \epsilon^2\dfrac{M}{2}. \end{pmatrix}.$$

Here, $(x_l, y_l, x_r, y_r)^T = p$, and l_l and l_r are given by

$$\sqrt{x_l^2 + y_l^2} \quad \text{and} \quad \sqrt{(x_r - x_b)^2 + (y_r - y_b)^2}.$$

Furthermore, the functions $x_b(t)$ and $y_b(t)$ are defined by

$$x_b(t) = \sqrt{l^2 - y_b^2(t)},$$

$$y_b(t) = r \sin(\omega t).$$

l	$=$	2	ϵ	$=$	10^{-2}	τ	$=$ $\pi/5$
l_0	$=$	$1/2$	M	$=$	10	ω	$=$ 10

Table 4.10: Constants in car axis

The function $\phi : \mathbb{R}^5 \to \mathbb{R}^2$ reads

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

The constants are listed in Table 4.10.

Consistent initial values are

$$p_0 = \begin{pmatrix} 0 \\ \\ 1/2 \\ \\ 1 \\ \\ 1/2 \end{pmatrix}, \quad q_0 = \begin{pmatrix} -1/2 \\ \\ 0 \\ \\ -1/2 \\ \\ 0 \end{pmatrix},$$

$$q_0' = \frac{2}{M\epsilon^2} f(0, p_0, \lambda_0), \quad \lambda_0 = \lambda_0' = (0,0)^T.$$

The indices of the variables p, q, and λ are 1, 2, and 3, respectively.

4.5.3 Numerical Results

In the reference solution for this problem, the DAE system is given as index 3. The structural information obtained through our software is given in Table 4.11. A consistent point and the corresponding condition number of the Jacobian evaluated at this point are given in Table 4.12.

Structural information	Results
Structural Index	3
HVT	4
Variable Offsets	(1, 1, 1, 1, 0, 0, 0, 0, 2, 2)
Equation Offsets	(2, 2, 2, 2, 1, 1, 1, 1, 0, 0)

Table 4.11: Structural index analysis of car axis

var.	value
x_1	$0.4934557842752397e-001$
x_2	0.4969894602300073
x_3	$0.1041742524885424e+001$
x_4	0.3739110272653672
x_5	$-0.7705836840358462e-001$
x_6	$0.7446866592147278e-002$
x_7	$0.1755681575356589e-001$
x_8	0.7703410437798304
x_9	$-0.4736886590853484e-002$
x_{10}	$-0.1104680331259640e-002$
condition number	$8.7782354298555460e+003$

Table 4.12: Consistent point and condition number for car axis

From this experiment, we can see that the condition number of the Jacobian is not large. This indicates that the Jacobian is structurally nonsingular, which further confirms the correctness of index computation for this problem.

On the other hand, since this problem is in first order, the SNI analysis is another applicable approach. Our software returns the SN index as 3, which is consistent with the result from the SA analysis. This is an example when the two analyses report the same index. In the case of systems which are nonlinear with respect to their highest-order derivatives, the coincidence is expected due to the nature of the SNI analysis, in which only linear coefficients are recorded. In the next example, we discuss the situation when the two analyses contradict with each other, and why the SNI analysis may provide correct results in some particular cases.

4.6 Example of Failures in Structural Analysis

4.6.1 Background Information

This example is due to Chowdhry, Krendl, and Linninger [KL04], in which they use it to show a situation where Pantelides' algorithm and their SN analysis contradict in index computation. It contains 3 differential equations and 1 algebraic constraint. (See also Chapter 2 for details.)

Structural information	Results
Structural Index	2
HVT	2
Variable Offsets	(0, 0, 0, 1)
Equation Offsets	(1, 1, 1, 0)

Table 4.13: Structural index analysis of an example in [KL04]

4.6.2 Mathematical Description

The equations are in the form of

$$0 = x_1' - x_4,$$

$$0 = x_2' - x_2 + x_4,$$

$$0 = x_3' - x_1^2 - x_2^2,$$

$$0 = x_1 + x_2 + 4x_3.$$

4.6.3 Numerical Results

The correct index of this problem is 3. However, the structural index returned by our software is 2, as shown in Table 4.13. The signature matrix of this problem is given in Table 4.14. Since no consistent point is given in the original literature, we select a sufficient number of random points, which are uniformly distributed in the unit interval with mean 0, to compute the system Jacobian. The results show that the Jacobian always has a very large condition number. This is an indication of likely incorrectness of our structural analysis.

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ d_j \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & -\infty & -\infty & 0* \\ -\infty & 1* & -\infty & 0 \\ 0 & 0 & 1* & -\infty \\ 0* & 0 & 0 & -\infty \end{pmatrix} \\ \begin{array}{cccc} 1 & 1 & 1 & 0 \end{array} \end{array} \begin{array}{c} c_i \\ 0 \\ 0 \\ 0 \\ 1 \end{array}$$

Table 4.14: Signature matrix for the example in [KL04]

On the other hand, the result of SNI analysis returned from our software shows that the system has index 3. The detailed steps of computation are explained in Chapter 2. From this example, we can see that when a DAE system is first order and linear with respect to the highest-order derivatives, the SNI analysis returns an accurate differential index. Both Pantelides' approach and Pryce's SA may or may not capture the cancellation between linear coefficients, therefore they may fail to produce the correct structural index of the problem. For first order linear systems, the SN index may serve as a good reference to the result from the SA analysis, since it is rigorous for linear problems.

4.7 Chemical Akzo Nobel

4.7.1 Background Information

This IVP is a stiff system of 6 non-linear Differential Algebraic Equations of index 1. The problem originates from Akzo Nobel Central research in Amsterdam, The

Netherlands [Sto98]. It describes a chemical process in which 2 species are mixed, while carbon dioxide is continuously added [MI03].

4.7.2 Mathematical Description

The problem is of the form

$$M\frac{dy}{dt} = f(y), \quad y(0) = y_0, \quad y'(0) = y_0',$$

with

$$y \in \mathbb{R}, \quad 0 \leq t \leq 180.$$

The matrix M is given in the form of

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and the function f is defined as

$$f = \begin{pmatrix} -2.0 \cdot r_1 & +r_2 & -r_3 & -r_4 & & \\ -0.5 \cdot r_1 & & & -r_4 & -0.5 \cdot r_5 & +F_{in} \\ r_1 & -r_2 & +r_3 & & & \\ & -r_2 & +r_3 & -2.0 \cdot r_4 & & \\ & r_2 & -r_3 & & +r_5 & \\ K_s \cdot y_1 \cdot y_4 - y_6 & & & & & \end{pmatrix},$$

where the r_i and F_{in} are auxiliary variables given by

$$\begin{aligned} r_1 &= k_1 \cdot y_1^4 \cdot y_2^{\frac{1}{2}}, \\ r_2 &= k_2 \cdot y_3 \cdot y_4, \\ r_3 &= \frac{k_2}{K} \cdot y_1 \cdot y_5, \\ r_4 &= k_3 \cdot y_1 \cdot y_4^2, \\ r_5 &= k_4 \cdot y_6^2 \cdot y_2^{\frac{1}{2}}, \\ F_{in} &= klA \cdot \left(\frac{p(CO_2)}{H} - y_2 \right). \end{aligned}$$

The values of the constant parameters are

$$\begin{aligned} k_1 &= 18.7, & k_4 &= 0.42, & K_s &= 115.83, \\ k_2 &= 0.58, & K &= 34.4, & p(CO2) &= 0.9, \\ k_3 &= 0.09, & klA &= 3.3, & H &= 737. \end{aligned}$$

Structural information	Results
Structural Index	1
HVT	5
Variable Offsets	(0, 0, 0, 0, 0, 0)
Equation Offsets	(1, 1, 1, 1, 1, 0)

Table 4.15: Structural index analysis of chemical Akzo Nobel

var.	value
y_1	$4.4400000000000001e-01$
y_2	$1.2300000000000000e-03$
y_3	$0.0000000000000000e+00$
y_4	$7.0000000000000001e-03$
y_5	$0.0000000000000000e+00$
y_6	$3.5999964000000001e-01$
condition number	$1.0000005115012430e+000$

Table 4.16: Consistent point and condition number for chemical Akzo Nobel

4.7.3 Numerical Results

In the reference solution, the system is given as index 1. The structural information obtained from our software are shown in Tables 4.15.

A consistent point given in the reference, as well as the computed condition number of the Jacobian at this point, are shown in Table 4.16. From the computation, we can see our software returns the correct structural information. The Jacobian evaluated at the consistent point is well-conditioned. We also ran the SNI analysis on this problem, and obtained the same differential index 1. This is another proof of the correctness of our computation.

4.8 Transistor Amplifier

4.8.1 Background Information

This problem is a stiff DAE consisting of 8 equations. The formulation we use here is taken from [EHR89].

4.8.2 Mathematical Description

The problem is of the form

$$M\frac{dy}{dt} = f(y), \quad y(0) = y_0, \quad y'(0) = y'_0,$$

with

$$y \in \mathbb{R}^8, \quad 0 \leq t \leq 0.2.$$

The matrix M is

$$M = \begin{pmatrix} -C_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_1 & -C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -C_3 & C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_3 & -C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -C_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -C_5 & C_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_5 & -C_5 \end{pmatrix},$$

and the function f is defined as

$$f = \begin{pmatrix} -\frac{U_e(t)}{R_0} + \frac{y_1}{R_0} \\ -\frac{U_b}{R_2} + y_2(\frac{1}{R_1} + \frac{1}{R_2}) - (\alpha - 1)g(y_2 - y_3) \\ -g(y_2 - y_3) + \frac{y_3}{R_3} \\ -\frac{U_b}{R_4} + \frac{y_4}{R_4} + \alpha g(y_2 - y_3) \\ -\frac{U_b}{R_6} + y_5(\frac{1}{R_5} + \frac{1}{R_6}) - (\alpha - 1)g(y_5 - y_6) \\ -g(y_5 - y_6) + \frac{y_6}{R_7} \\ -\frac{U_b}{R_8} + \frac{y_7}{R_8} + \alpha g(y_5 - y_6) \\ \frac{y_8}{R_9} \end{pmatrix},$$

where g and U_e are auxiliary functions given by

$$g(x) = \beta(e^{\frac{x}{U_T}} - 1) \quad \text{and} \quad U_e(t) = 0.1\sin(200\pi t).$$

The values of the constants are

$$U_b = 6,$$
$$U_F = 0.026,$$
$$\alpha = 0.99,$$
$$\beta = 10^{-6},$$

and

$$R_0 = 1000,$$
$$R_k = 9000, \quad \text{for} \quad k = 1, \ldots, 9,$$
$$C_k = k \cdot 10^{-6}, \quad \text{for} \quad k = 1, \ldots, 9.$$

Structural information	Results
Structural Index	1
HVT	5
Variable Offsets	(0, 1, 0, 0, 1, 0, 0, 1)
Equation Offsets	(1, 1, 1, 1, 1, 1, 1, 1)

Table 4.17: Structural index analysis of transistor amplifier

4.8.3 Numerical Results

The original system is defined by 8 differential equations only. Therefore, the structural index obtained from our SA computation is 0. However, one can easily determine that the coefficient matrix M is rank deficient, where the second, fifth, and eighth equations are linearly dependent with the first, fourth, and seventh equations. By explicitly eliminating the linearly dependent rows in M, we obtain a modified system with 5 differential equations and 3 algebraic constraints.

In the reference solution, the system is determined as index 1. The structural information of the modified system returned by our software is shown in Table 4.17.

A consistent point and the condition number of the Jacobian are given in Table 4.18. From the computation, we can see the Jacobian is likely nonsingular at the consistent point, which indicates our index computation is reliable.

var.	value
y_1	$0.0000000000000000e+00$
y_2	$3.0000000000000000e+00$
y_3	$3.0000000000000000e+00$
y_4	$6.0000000000000000e+00$
y_5	$3.0000000000000000e+00$
y_6	$3.0000000000000000e+00$
y_7	$6.0000000000000000e+00$
y_8	$0.0000000000000000e+00$
condition number	$2.6189826195363230e+002$

Table 4.18: Consistent point and condition number for transistor amplifier

4.9 Ring Modulator

4.9.1 Background Information

Originally, this problem is a stiff ODE system of 15 non-linear differential equations [MI03]. If we let the parameter C_s be 0, it becomes a DAE system with 11 differential equations and 4 algebraic constraints. The original problem is taken from [WKS92].

4.9.2 Mathematical Description

$$\frac{dy}{dt} = f(y), \quad y(0) = y_0,$$

with

$$y \in \mathbb{R}^{15}, \quad 0 \leq t \leq 10^{-3}.$$

The definition of $f(y)$ is given in [MI03].

Structural information	Results
Structural Index	0
HVT	15
Variable Offsets	(0, 0,...,0)
Equation Offsets	(1, 1,...,1)

Table 4.19: Structural index analysis of ring modulator (ODE)

Structural information	Results
Structural Index	1
HVT	11
Variable Offsets	(0, 0,...,0)
Equation Offsets	(1, 1, 0, 0, 0, 0, 1, 1,...,1)

Table 4.20: Structural index analysis of ring modulator (DAE)

4.9.3 Numerical Result

Here we give the structural results for both ODE and DAE formats of the problem.

For the ODE case, the result returned by our software is shown in Table 4.19. The system Jacobian is the identity matrix with condition number 1.

For the DAE case, the correct index is 2. However, the SA analysis obtains index 1 as shown in Table 4.20. Given a consistent point at 0, we compute the Jacobian and its condition number. Due to the large condition number $8.150552506528627e+016$ obtained, we can conclude that the structural index 1 we computed previously is

Structural information	Results
Structural Index	2
HVT	10
Variable Offsets	(0, 0, 0, 0, 1, 0, 0,...,0)
Equation Offsets	(1, 1, 0, 0, 0, 0, 1,...,1)

Table 4.21: Structural index analysis of modified ring modulator (DAE)

likely unreliable. If we look at the formulas of the four algebraic equations

$$\begin{aligned} f_1: &\quad 0 = y_{10} - q(U_{D1}) + q(U_{D4}), \\ f_2: &\quad 0 = -y_{11} + q(U_{D2}) - q(U_{D3}), \\ f_3: &\quad 0 = y_{12} + q(U_{D1}) - q(U_{D3}), \\ f_4: &\quad 0 = -y_{13} - q(U_{D2}) + q(U_{D4}), \end{aligned}$$

we can see that these four equations are linearly dependent in terms of the auxiliary function $q(U_{D_i})$. We modify the original problem so that the third equation is substituted by

$$\begin{aligned} \bar{f}_3 &= f_1 - f_2 + f_3 - f_4 \\ &= y_{10} + y_{11} + y_{12} + y_{13}. \end{aligned}$$

For the modified system, we apply the SA analysis again and obtain the correct structural information as in Table 4.21. With the same consistent point, we have a significant smaller condition number $1.288620749590146e + 006$.

4.10 Summary of Numerical Experiments

Conclusions from our experiments are given below.

1. ASIA is accurate for computing the structural information of DAE systems when the SA analysis succeeds. We have tested on problems with index 1 to index 5. Our software successfully returns the correct structural indices, HVTs, offsets, as well as system Jacobian evaluated at consistent points. For first order problems, ASIA also provides the SN indices and SNI matrices as references.

2. The condition number of the Jacobian is a good indication of correctness of index computation. In our examples, structural nonsingularity of the Jacobian matrix at a consistent point always corresponds to a correct index computation.

3. For some problems, the SA analysis can fail and produce a structurally singular Jacobian. We have illustrated that common subexpression elimination and linear transformation can transform the system into a form such that the analysis succeeds.

4. For first order linear systems, we have shown that the SN analysis can provide accurate differential index in some particular cases, while the SA analysis fails.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

We have presented a software for automatic structural index analysis. For a given DAE system programmed in the required format, we compute the signature matrix through operator overloading. After that, we solve a linear assignment problem and obtain the HVT. Then, we solve the dual of the HVT to find the two offsets. Based on this information, we return the structural index accordingly.

The second part of our index analysis is to compute the system Jacobian at a given consistent point. Similarly, we evaluate each equation by operator overloading. We compute the condition number of the Jacobian to determine its structural singularity, which indicates the correctness of the index computation in the previous step.

Third, for first order systems, we apply the symbolic-numeric analysis. We form the SNI matrix, and perform the symbolic-numeric differentiation to obtain the SN index of the problem.

In addition, we discussed some design and implementation issues of the software, and gave detailed instructions on how to use the package.

We have reported detailed numerical results on 8 DAE and ODE test problems. We showed that ASIA can provide accurate index computation on applications from different disciplines. We compared the results from the two analyses and gave explanations on the situation when they contradict with either other. This can be used as important reference for future refinement of the algorithms. We also showed some initial attempts of transforming a given DAE system into another form, which can help to eliminate the failures of structural analysis.

5.2 Future Work

From the numerical experiments, we can see that in some cases both structural analyses cannot compute the correct index for the original problem. Certain transformations may help to provide a solvable form of the problem, but these transformations have not been systematically studied. Future work in index analysis may include developing heuristics for transforming equations. Based on that, an automatic preprocessing module may be added to the tool. Furthermore, a more general algorithm

for structural index analysis may be developed.

The examples that we have tested are taken from the literature with small to moderate sizes. In future, large scale industrial applications can be studied using ASIA. Due to the nature of large matrix computation, some optimization may need to be applied to the software to meet specific requirements.

Bibliography

[AP98] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, July 1998.

[Ber91] D. P. Bertsekas. *Linear Network Optimization.* MIT Press, 1991.

[CG95a] S. L. Campbell and C. W. Gear. The index of general nonlinear DAEs. *Numerische Mathematik*, 72:173–196, 1995.

[CG95b] S.L. Campbell and E. Griepentr. Solvability of general differential algebraic equations. *SIAM J. Sci. Comput.*, 16:257–270, 1995.

[Duf81] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software*, 7(3):315–330, 1981.

[EHR89] C. Lubich E. Hairer and M. Roche. The numerical solution of differential-algebraic systems by runge-kutta methods. *Lecture Notes in Mathematics*, 1490, 1989.

[Gea88] C. W. Gear. Differential algebraic equations index transforms. *SIAM J. Sci. Stat. Comput.*, 9:39–47, 1988.

[Gea90] C. W. Gear. Differential algebraic equations, indices, and integral algebraic equations. *SIAM J. Numer. Anal.*, 27:1527–1534, 1990.

[GL96] G. H. Golub and C. F. Van Loan. *Matrix Computation.* John Hopkins University Press, 1996.

[GRB00] W. Martinson G. Reiszig and P. Barton. Differential-algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM J. Sci. Comput.*, 2000.

[GW04] A. Griewank and A. Walther. On the efficient generation of Taylor expansions for DAE solutions by automatic differentiation. Technical report, Technische Universität Dresden, Department of Mathematics, Institute of Scientific Computing, 2004.

[JV87] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[JV03] R. Jonker and A. Volgenant. LAP web page, May 2003. LAP is available at `http://www.magiclogic.com/assignment.html`.

[KEBP96] S. L. Campbell K. E. Brenan and L. S. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* SIAM, 1996.

[KL04] S. Chowdhry H. Krendl and A. A. Linninger. Symbolic numeric index analysis algorithm for differential algebraic equations. *Ind. Eng. Chem. Res.*, 43:3886–3894, 2004.

[MI03] F. Mazzia and F. Iavernaro. Test set for initial value problem solvers. Technical report, Department of Mathematics, University of Bari, Italy, August 2003.

[MS93] S. E. Mattsson and G. Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM. J. Sci. Comput.*, 14:677–692, 1993.

[NP05a] N. Nedialkov and J. D. Pryce. Solving differential-algebraic equations by Taylor series(II): computing the System Jacobian. *BIT*, 2005. submitted.

[NP05b] N. S. Nedialkov and J. D. Pryce. Solving differential-algebraic equations by Taylor series(I): computing Taylor coefficients. *BIT*, pages 561–591, 2005.

[Pan88] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM. J. Sci. Stat. Comput*, 9:213–231, 1988.

[Pry98] J. D. Pryce. Solving high-index DAEs by Taylor series. *Numerical Algorithms*, 14:195–211, 1998.

[Pry01] J. D. Pryce. A simple structural analysis method for DAEs. *BIT*, 41(2):364–394, 2001.

[Sch94] S. Schneider. *Intégration de systèmes d'équations différentielles raides et différentielles-algébriques par des méthodes de collocations et méthodes générales linéaires.* PhD thesis, Université de Genève, 1994.

[Sto98] W.J.H. Stortelder. *Parameter Estimation in Nonlinear Dynamical Systems.* PhD thesis, University of Amsterdam, March, 1998.

[WKS92] P. Rentrop W. Kampowski and W. Schmidt. Classification and numerical simulation of electric circuits. *Surveys of Mathematics of Industries*, 2(1):23–65, 1992.

[Zha05] W.H. Zhang. Design and implementation of a solver for high-index differential-algebraic equations. Master's thesis, McMaster University, 2005.